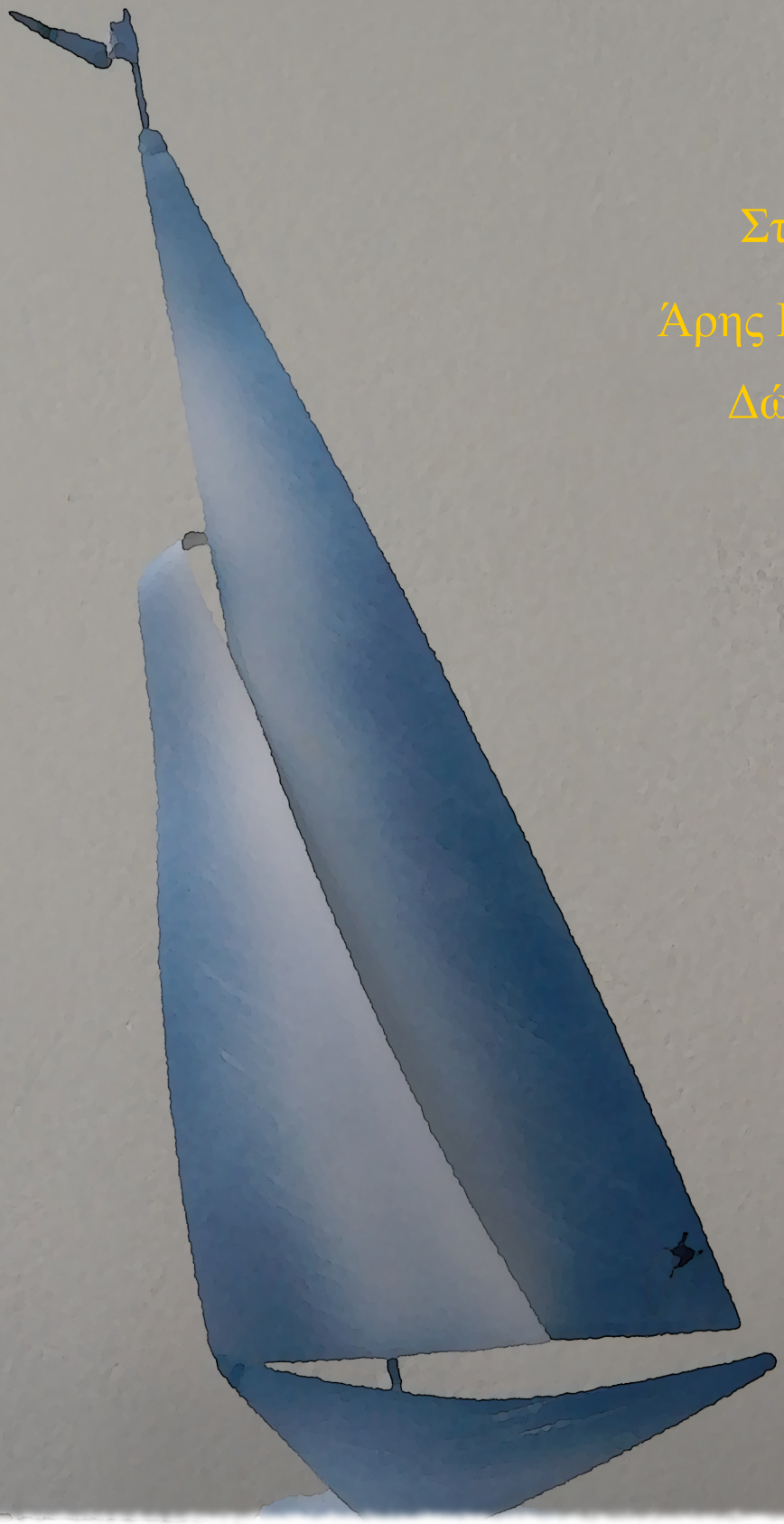
# Θεμελίωση Επιστήμης Υπολογιστών



Στάθης Ζάχος

Άρης Παγουρτζής

Δώρα Σούλιου

Ελληνικά Ακαδημαϊκά Ηλεκτρονικά Συγγράμματα και Βοηθήματα
www.kallipos.gr

HEALLINK
Σύνδεσμος Ελληνικών Ακαδημαϊκών Βιβλιοθηκών

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

ΕΣΠΑ 2007-2013
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

# Θεμελίωση Επιστήμης Υπολογιστών

**Συγγραφείς**
Ευστάθιος Ζάχος
Αριστείδης Παγουρτζής
Δώρα Σούλιου

**Κριτικός αναγνώστης**
Βασίλης Ζησιμόπουλος



ΣΥΝΔΕΣΜΟΣ ΕΛΛΗΝΙΚΩΝ ΑΚΑΔΗΜΑΪΚΩΝ ΒΙΒΛΙΟΘΗΚΩΝ

Εθνικό Μετσόβιο Πολυτεχνείο Ηρώων Πολυτεχνείου 9, 15780 Ζωγράφου

www.kallipos.gr

ISBN: 978-960-603-409-1

# Περιεχόμενα

# Πρόλογος

Σκοπός του συγγράμματος αυτού είναι η εισαγωγή στις θεμελιώδεις έννοιες της Επιστήμης των Υπολογιστών, με έμφαση σε αυτές που σχετίζονται με τη Θεωρητική Πληροφορική και τη Θεωρία Υπολογισμού.

Περιλαμβάνονται: Εισαγωγή στις Έννοιες του Υπολογισμού και του Αλγορίθμου, Θεωρία Αυτομάτων, Τυπικών Γλωσσών και Γραμματικών, Σχεδίαση και Ανάλυση Αλγορίθμων, Αλγόριθμοι Γράφων, Λογική για την Επιστήμη των Υπολογιστών, Μοντέλα Υπολογισμού, Υπολογισιμότητα και Υπολογιστική Πολυπλοκότητα, Βάσεις Δεδομένων, Λειτουργικά Συστήματα, Τεχνητή Νοημοσύνη, Κρυπτογραφία.

Το παρόν βιβλίο περιλαμβάνει σημαντικό μέρος της ύλης που διδάσκεται στους φοιτητές των μαθημάτων "Εισαγωγή στην Επιστήμη Υπολογιστών" της Σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών ΕΜΠ και "Θεμελιώδη Θέματα Επιστήμης Υπολογιστών" της Σχολής Εφαρμοσμένων Μαθηματικών και Φυσικών Επιστημών ΕΜΠ.

Ευχαριστίες ανήκουν σε όλα τα μέλη του Εργαστηρίου Λογικής και Υπολογισμών του ΕΜΠ, που επί σειρά ετών συμμετείχαν στις διδακτικές και ερευνητικές δραστηριότητες του εργαστηρίου, συμβάλλοντας έτσι έμμεσα ή άμεσα στην δημιουργία του παρόντος συγγράμματος. Τα ονόματά τους αναφέρονται στην ιστοσελίδα του εργαστηρίου:

http://www.corelab.ntua.gr/people.html

Ιδιαίτερα θα θέλαμε να ευχαριστήσουμε τους παρακάτω πρώην και νυν σπουδαστές που συμμετείχαν στην προετοιμασία διδακτικών σημειώσεων, πάνω στις οποίες στηρίχθηκε το παρόν σύγγραμμα: Γιάννη Κασσιό, Κατερίνα Ποτίκα, Παναγιώτη Χείλαρη, Βαγγέλη Μπαμπά, Αντώνη Αχιλλέως, Γεωργία Καούρη, και Αντώνη Αντωνόπουλο.

Ευχόμαστε καλή ανάγνωση, και παρακαλούμε για τις παρατηρήσεις σας και τις υποδείξεις σας, τις οποίες μπορείτε να στείλετε στα προσωπικά email των συγγραφέων: zachos@cs.ntua.gr, pagour@cs.ntua.gr, dsouliou@mail.ntua.gr

Στάθης Ζάχος – Άρης Παγουρτζής – Δώρα Σούλιου, 2015

# Κεφάλαιο 1

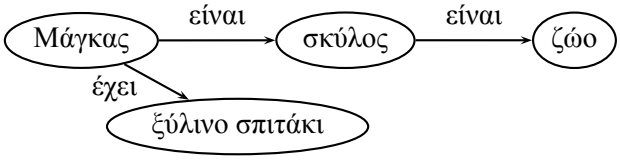
# Εισαγωγή

Ο όρος *Επιστήμη των Υπολογιστών* (Computer Science) χρησιμοποιείται περισσότερο στην Αμερική. Στην Ευρώπη χρησιμοποιούμε τον όρο *Πληροφορική* (Informatics), ενώ στην Μεγάλη Βρετανία λένε συχνά *Επιστήμη των Υπολογισμών* (Computing Science).

Όπως είπε και ο διαπρεπής E. Dijkstra, η Επιστήμη των Υπολογιστών έχει την ίδια σχέση με τους υπολογιστές που έχει η Αστρονομία με τα τηλεσκόπια.

Σήμερα η Επιστήμη των Υπολογιστών άπτεται σχεδόν οποιασδήποτε ανθρώπινης δραστηριότητας. Η Πληροφορική είναι η συστηματική μελέτη αλγοριθμικών διαδικασιών που περιγράφουν και επεξεργάζονται πληροφορίες. Πιο συγκεκριμένα, η Επιστήμη των Υπολογισμών περιλαμβάνει: θεωρία, ανάλυση, σχεδίαση, αποδοτικότητα, υλοποίηση και εφαρμογή αλγοριθμικών διαδικασιών. Το βασικό ερώτημα με το οποίο ασχολείται η Επιστήμη των Υπολογιστών είναι τι μπορεί να *μηχανοποιηθεί* και μάλιστα με αποδοτικό τρόπο. Είναι κατά βάση *αφαιρετική* επιστήμη: Κατασκευάζουμε το σωστό μοντέλο για ένα πρόβλημα και επινοούμε τις κατάλληλες μηχανιστικές τεχνικές για την επίλυσή του.

Συχνά η εύρεση μίας καλής αφαίρεσης για ένα πρόβλημα μπορεί να είναι δύσκολη, λόγω των θεμελιωδών περιορισμών ως προς τις διαδικασίες (tasks) που μπορούν να επιτελέσουν οι υπολογιστές και την ταχύτητα με την οποία εκτελούν οι υπολογιστές αυτές τις διαδικασίες. Ένα δύσκολο τέτοιο πρόβλημα είναι η *αναπαράσταση γνώσης* (knowledge representation). Παρ' όλα αυτά έχουμε κάνει πρόοδο επινοώντας διάφορες αφαιρέσεις οι οποίες μας βοηθούν να κατασκευάσουμε προγράμματα που κάνουν ορισμένα είδη συλλογισμών. Μία τέτοια αφαίρεση είναι ο κατευθυνόμενος γράφος (directed graph· βλέπε σχήμα 1.1).




Σχήμα 1.1: Γράφος για την αναπαράσταση γνώσης.

'Αλλη μία χρήσιμη αφαίρεση είναι η *τυπική λογική*, η οποία μας επιτρέπει να χειριζόμαστε *γεγονότα* (facts) μέσω της εφαρμογής *συμπερασματικών κανόνων* (rules of inference).

Για την επίλυση προβλημάτων χρησιμοποιούνται τα παρακάτω εργαλεία:

1. *μοντέλα δεδομένων (data models)*, που είναι οι αφαιρέσεις που περιγράφουν το πρόβλημα, για παράδειγμα, όπως αναφέραμε προηγουμένως, η λογική και οι γράφοι·
2. *δομές δεδομένων (data structures)*, που είναι οι δομές των γλωσσών προγραμματισμού που παριστάνουν τα μοντέλα δεδομένων. Για παράδειγμα στην Pascal, οι πίνακες, οι εγγραφές, οι δείκτες μας επιτρέπουν να κατασκευάσουμε δομές δεδομένων που παριστάνουν πιο πολύπλοκες αφαιρέσεις, όπως οι γράφοι·
3. *αλγόριθμοι*, που είναι οι τεχνικές με τις οποίες παίρνουμε λύσεις για τα προβλήματα μέσω της επεξεργασίας των αντίστοιχων δομών δεδομένων.

## 1.1 Κλάδοι Επιστήμης των Υπολογιστών

Ακολουθεί μία πρόχειρη απαρίθμηση διαφόρων κλάδων της Επιστήμης των Υπολογιστών. Κοινό χαρακτηριστικό όλων τους είναι το περιεχόμενό τους: το θεωρητικό κομμάτι, η δημιουργία μοντέλων και το σχεδιαστικό – κατασκευαστικό τους κομμάτι:

1. Υπολογισιμότητα και Πολυπλοκότητα – Μοντέλα Υπολογισμού
2. Θεωρία Αυτομάτων και Τυπικών Γλωσσών
3. Αλγόριθμοι και Δομές Δεδομένων
4. Γλώσσες Προγραμματισμού και Μεταγλωττιστές
5. Αρχιτεκτονική Υπολογιστών και Δικτύων (hardware)
6. Αριθμητικοί και Συμβολικοί Υπολογισμοί
7. Λειτουργικά - Παράλληλα - Κατανεμημένα Συστήματα
8. Μεθοδολογία - Τεχνολογία Λογισμικού (software)
9. Βάσεις Δεδομένων και Διαχείριση Πληροφοριών
10. Τεχνητή Νοημοσύνη και Ρομποτική
11. Επικοινωνία ανθρώπου - υπολογιστή. Πολυμέσα
12. Κρυπτογραφία και ασφάλεια
13. Δίκτυα Επικοινωνιών - Ευφυή Δίκτυα - Διαδίκτυο
14. Υπολογιστική βιολογία

## 1.2 Επανάληψη, επαγωγή, αναδρομή

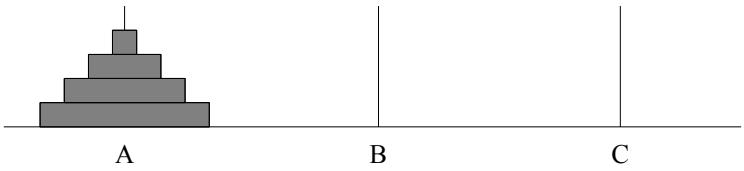
Θα συζητήσουμε διάφορες αρχές πάνω στις οποίες βασίζονται πολλές κατασκευές: π.χ. αλγεβρικές δομές όπως ’Αλγεβρες Boole και η Αναδρομική Μέθοδος.

### 1.2.1 Επανάληψη (Iteration)

Η ισχύς των υπολογιστών έγκειται στην δυνατότητά τους να εκτελούν με μεγάλη ταχύτητα *επαναληπτικά* την ίδια διαδικασία (πιθανώς παραμετροποιημένη). Ο απλούστερος τρόπος για να εκτελέσουμε μία διαδικασία επαναληπτικά είναι με μία δομή ελέγχου όπως η **for** της Pascal.

### 1.2.2 Αναδρομή (Recursion)

Αρκετά προβλήματα λύνονται εύκολα με την βοήθεια *αναδρομικών* διαδικασιών ή συναρτήσεων. Αναδρομικές είναι οι συναρτήσεις ή διαδικασίες που καλούν τον εαυτό τους μία ή περισσότερες φορές προκειμένου να λύσουν σχετικά υποπροβλήματα. Στον ορισμό της αναδρομικής διαδικασίας διαπιστώνουμε ότι το αρχικό πρόβλημα διασπάται σε μικρότερα προβλήματα του ίδιου τύπου με το αρχικό.




Σχήμα 1.2: Πύργοι του Ανόι ($n = 4$).

Ένα πρόβλημα για την λύση του οποίου είναι πιο εύκολο να κατασκευάσουμε έναν αναδρομικό από ότι έναν επαναληπτικό αλγόριθμο είναι οι “Πύργοι του Ανόι”: Έχουμε τρεις πασάλους, έστω A, B, C, και $n$ δίσκους, που είναι όλοι διαφορετικοί σε μέγεθος μεταξύ τους. Αρχικά όλοι οι δίσκοι, οι οποίοι έχουν μία τρύπα στην μέση, ούτως ώστε να περνάνε στους πασάλους, βρίσκονται περασμένοι στον πάσαλο A, έτσι που να μην υπάρχει μικρότερος δίσκος κάτω από κάποιον μεγαλύτερό του (βλέπε σχήμα 1.2). Σκοπός είναι να μετακινήσουμε όλους τους δίσκους από τον πάσαλο A στον πάσαλο B, έναν κάθε φορά έτσι ώστε ποτέ να μην βρεθεί (στον ίδιο πάσαλο) μικρότερος δίσκος κάτω από μεγαλύτερο, χρησιμοποιώντας τον (βοηθητικό) πάσαλο C.

Ένας αναδρομικός αλγόριθμος:

```
procedure move_anoi(n from X to Y using Z)
begin
   if n = 1 then move top disk from X to Y
   else begin
            move_anoi(n−1 from X to Z using Y);
            move top disk from X to Y;
            move_anoi(n−1 from Z to Y using X)
```

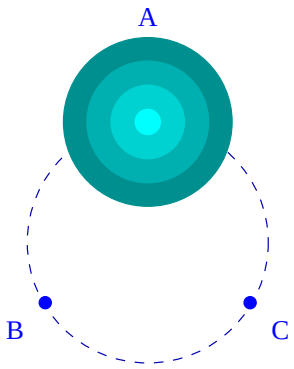
**end**
**end**

Για να λύσουμε το πρόβλημα του σχήματος 1.2 αρκεί να εκτελέσουμε την διαδικασία ως εξής: move_anoi(4 from A to B using C)

**'Ασκηση:** Δείξτε ότι ο ακόλουθος επαναληπτικός αλγόριθμος λύνει το πρόβλημα των πύργων του Ανόι:
Θεωρούμε ότι οι πάσαλοι είναι τοποθετημένοι επί κύκλου όπως στο σχήμα 1.3. Ο επαναληπτικός αλγόριθμος λειτουργεί ως εξής:

*Επαναλάμβανε συνέχεια τα παρακάτω βήματα μέχρι να μην μπορείς να εκτελέσεις το δεύτερο βήμα:*

1. *μετακίνησε κατά την θετική φορά τον μικρότερο δίσκο·*
2. *κάνε την μοναδική επιτρεπτή κίνηση που δεν αφορά τον μικρότερο δίσκο.*




Σχήμα 1.3: Πύργοι του Ανόι ($n = 4$): για τον επαναληπτικό αλγόριθμο.

### 1.2.3 Επαγωγή (Induction)

Η *επαγωγή* είναι μία σημαντική μέθοδος για την απόδειξη διαφόρων προτάσεων. Είναι πολύ χρήσιμη κατά την απόδειξη της ορθότητας των προγραμμάτων. Επίσης, η επαγωγή χρησιμοποιείται στον λεγόμενο επαγωγικό τρόπο ορισμού, π.χ.: "Μία λίστα είναι είτε η κενή λίστα είτε ένα στοιχείο ακολουθούμενο από μία λίστα".

### 1.2.4 Μερική και ολική ορθότητα

Στα προγράμματα συνήθως διακρίνουμε τρία επίπεδα ορθότητας: **συντακτική** ορθότητα (αντιστοιχεί στο context free κομμάτι της γλώσσας), **νοηματική** ορθότητα (αντιστοιχεί στο context sensitive κομμάτι της γλώσσας) και **σημασιολογική** ορθότητα. Στα προγράμματα ο compiler

ελέγχει τα 2 πρώτα επίπεδα ορθότητας. Η σημασιολογική ορθότητα δεν μπορεί φυσικά να ελεγχθεί από τον compiler και άρα ελέγχεται με δοκιμές (testing) ή με μαθηματική επαλήθευση (verification) σε σχέση με κάποιες προδιαγραφές (specifications). Γενικά, επιδιώκουμε ένα πρόγραμμα να έχει απλή δομή, δηλαδή να αποτελείται από δομικά στοιχεία (building blocks), επαναχρησιμοποιούμενες ενότητες (modules) για τις οποίες έχει ήδη γίνει αυστηρή επαλήθευση ορθότητας. Υπάρχουν πολλών ειδών σημασιολογίες προγραμμάτων (program semantics):

- **Λειτουργική σημασιολογία** (operational semantics). Περιγράφει την υπολογιστική ακολουθία που εκτελείται.
- **Δηλωτική σημασιολογία** (denotational semantics). Ορίζει μόνο τη συνάρτηση εισόδου-εξόδου.
- **Αξιωματική σημασιολογία** (axiomatic semantics). Περιγράφει τις σχετικές ιδιότητες που πρέπει απαραίτητα να ικανοποιούνται από την είσοδο και την έξοδο.

Θα περιοριστούμε σε μια ελάχιστη περιγραφή της τρίτης. Διάφορες ιδιότητες βεβαιώνονται (are asserted) σε ορισμένα σημεία της ροής του προγράμματος. Αυτές οι συνθήκες που βεβαιώνονται, και πρέπει να αποδειχθεί ότι ικανοποιούνται, λέγονται **βεβαιώσεις** (assertions). Οι βεβαιώσεις βρόχου λέγονται **αναλλοίωτες του βρόχου** (loop invariants).

Δεν αρκεί να αποδείξουμε την ορθότητα σε περίπτωση τερματισμού. Η απόδειξη ορθότητας σε περίπτωση τερματισμού λέγεται **μερική ορθότητα** (partial correctness). Για την **ολική ορθότητα** (total correctness) χρειάζεται και απόδειξη τερματισμού. **Συνθήκη τερματισμού** (termination condition) λέγεται μια γνησίως φθίνουσα θετική συνάρτηση $f(t)$ που εγγυάται τον τερματισμό όταν $f(t) = 0$ (όπου $t$ ο χρόνος, ο αριθμός των βημάτων που έχουν εκτελεστεί).

Στην περίπτωση που, αντί για κατηγορηματικό λογισμό και φυσικούς αριθμούς, χρησιμοποιούμε πράξεις και ιδιότητες (αξιώματα) κάποιας άλλης συγκεκριμένης αλγεβρικής δομής η αξιωματική σημασιολογία ονομάζεται συνήθως **αλγεβρική σημασιολογία** (algebraic semantics).

## 1.3 Δομημένος προγραμματισμός και modularity

Η ιδεολογία του δομημένου προγραμματισμού υποστηρίζει ότι για κάθε ανεξάρτητη συγκεκριμένη λειτουργία πρέπει να γράφεται μια ανεξάρτητη διαδικασία. Θα πρέπει λοιπόν να αναλύεται όσο γίνεται το πρόβλημα σε ανεξάρτητα υποπροβλήματα και για το καθένα από αυτά να γράφεται κάποιο υποπρόγραμμα (διαδικασία ή συνάρτηση). Επίσης, πρέπει να αποφεύγεται η χρήση παρενεργειών. Αυτό σημαίνει ότι κάθε υποπρόγραμμα πρέπει να κάνει μια συγκεκριμένη λειτουργία, χωρίς να επηρεάζει το κυρίως πρόγραμμα ή άλλα υποπρογράμματα. Η επικοινωνία των επιμέρους μονάδων (πέρασμα τιμών κ.τ.λ.) γίνεται με χρήση παραμέτρων (βλέπε παρακάτω). Έτσι, το πρόγραμμα γίνεται ευανάγνωστο και είναι εύκολη η μεταφορά και η χρήση των υποπρογραμμάτων σε άλλα προγράμματα .

- Είναι ευκολότερο να γράψουμε ένα δομημένο πρόγραμμα, επειδή πολύπλοκα προβλήματα διασπώνται σε έναν αριθμό μικρότερων, απλούστερων εργασιών.

- Είναι ευκολότερο να ανιχνεύουμε λάθη σε δομημένο πρόγραμμα.
- Ένα σχετικό πλεονέκτημα είναι ο χρόνος που εξοικονομείται με δομημένα προγράμματα. Μπορούν να διορθωθούν ή να τροποποιηθούν πιο εύκολα. Όταν έχουμε κατασκευάσει μια διαδικασία που κάνει κάτι συγκεκριμένο, θα μπορούμε να τη χρησιμοποιούμε σε άλλα προγράμματα δίχως να σπαταλάμε χρόνο σε συγγραφή νέων υποπρογραμμάτων.

Σε ορισμένες γλώσσες, όπως η Modula-2, αλλά και στις περισσότερες σύγχρονες γλώσσες, υποστηρίζεται η διάσπαση του κώδικα σε διαφορετικές αλληλοσυνεργαζόμενες **ενότητες** οι οποίες ονομάζονται **modules**. Κάθε μια από τις ενότητες αυτές περιέχει τα δικά της αντικείμενα (σταθερές, τύπους, μεταβλητές, υποπρογράμματα). Μια τέτοια ενότητα μπορεί να αναφέρεται σε (ή να καλεί) αντικείμενα που είτε είναι εσωτερικά σ' αυτήν είτε έχουν εισαχθεί, με αυστηρό τρόπο, από άλλες ενότητες. Βέβαια για να γίνει δυνατή η εισαγωγή τέτοιων αντικειμένων θα πρέπει να έχει γίνει επιτρεπτή η εξαγωγή τους στην ενότητα που έχουν οριστεί. Τελικά, ένα πρόγραμμα αποτελείται από ένα σύνολο ενοτήτων, εντελώς ανεξαρτήτων ως προς την υλοποίηση, οι οποίες όμως επικοινωνούν μεταξύ τους με αυστηρά καθορισμένο τρόπο. Έτσι είναι δυνατή η δημιουργία μιας "**βιβλιοθήκης**" **προγραμμάτων**, οριζόμενων από το χρήστη ώστε κώδικας που έχει γραφτεί μια φορά να μπορεί να χρησιμοποιείται από πολλά προγράμματα που δυνατόν να χρειάζονται τις ίδιες λειτουργίες. Ο βασικός λόγος για αυτή τη δόμηση *είναι η επιθυμία μας για την υλοποίηση μιας ιεραρχίας αφαίρεσης (abstraction) στην οποία modules που βρίσκονται σε υψηλότερο επίπεδο από κάποια άλλα να μπορούν να χρησιμοποιούν αντικείμενα που έχουν οριστεί σε αυτά χωρίς να χρειάζονται να γνωρίζουν τίποτα για τον τρόπο υλοποίησής τους (implementation hiding)*.

Απαγορεύοντας έτσι την πρόσβαση στο εσωτερικό της κάθε ενότητας μπορούμε να φτιάξουμε ενότητες οι οποίες (είμαστε σίγουροι ότι) λειτουργούν σωστά και έτσι, σε μετέπειτα στάδιο κατά την χρησιμοποίησή τους από άλλη ενότητα, θα έχουμε μικρύνει κατά πολύ την περιοχή αναζήτησης λαθών.

Εφόσον όμως κάθε ενότητα μπορεί να εισαγάγει αντικείμενα από διάφορες άλλες, ο μεταφραστής (compiler) θα πρέπει να έχει πρόσβαση στην περιγραφή της δομής των δεδομένων έτσι ώστε να εξασφαλίζεται η σωστή επικοινωνία. Οδηγούμαστε έτσι στην *περαιτέρω διάσπαση ενότητας σε μέρος ορισμού (definition part) και μέρος υλοποίησης (implementation part) τα οποία μπορούν να δημιουργηθούν ανεξάρτητα*.

## 1.4 Παράλληλες, ταυτόχρονες, κατανεμημένες διεργασίες

Γενικά, οι αλγόριθμοι στους οποίους αναφερθήκαμε ως τώρα εκτελούνται *σειριακά*, όπως λέμε, σε σε έναν υπολογιστή. Παρ' όλα αυτά υπάρχουν ορισμένα προβλήματα, τα οποία μπορούμε να λύσουμε, κατανέμοντας το υπολογιστικό φορτίο σε περισσότερους από έναν υπολογιστές/επεξεργαστές που λειτουργούν *παράλληλα*. Χάρη στον *ταυτοχρονισμό* (concurrency), μειώνεται σημαντικά ο χρόνος που παίρνουμε απάντηση στο πρόβλημα.

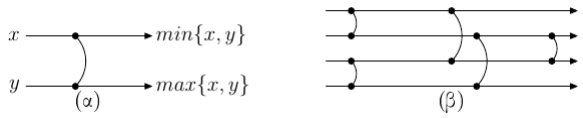
Πολλά προβλήματα σε αυτήν την περιοχή προκύπτουν από τις λεγόμενες κατανεμημένες (distributed) διεργασίες. Αυτές είναι διεργασίες που από την φύση τους εκτελούνται παράλληλα, για παράδειγμα οι συνδιαλέξεις σε ένα τηλεφωνικό δίκτυο.

Το μεγάλο πρόβλημα, στα παράλληλα συστήματα είναι πώς θα *παραλληλοποιήσουμε* έναν σειριακό αλγόριθμο. Για πολλούς αλγορίθμους, αυτό είναι εφικτό με σημαντική επιτυχία.

Παρ'όλα αυτά υπάρχουν και διεργασίες που είναι όπως λέμε *εγγενώς σειριακές* (inherently sequential). Σε αυτές, είναι τόσο έντονη η εξάρτηση ενός βήματος από τα προηγούμενα που δεν είναι δυνατόν τα βήματα να εκτελεστούν παράλληλα και ανεξάρτητα μεταξύ τους.

### 1.4.1 Δίκτυα ταξινόμησης

Σκοπός των δικτύων ταξινόμησης είναι η ταξινόμηση μίας ακολουθίας αριθμών χρησιμοποιώντας μόνο συγκρίσεις, με όσο το δυνατό μεγαλύτερη παραλληλία. Τα δίκτυα ταξινόμησης αναπαρίστανται με οριζόντιες γραμμές, μία για κάθε είσοδο. Οι συγκρίσεις μεταξύ δύο αριθμών αναπαρίστανται με κατακόρυφες συνδέσεις δύο γραμμών. Κάθε σύγκριση θεωρείται ότι "κοστίζει" μία χρονική μονάδα και η συνολική καθυστέρηση αντιστοιχεί στον αριθμό των κόμβων σε μια οριζόντια γραμμή. Η έξοδος είναι ταξινομημένη από "πάνω" προς τα "κάτω". Στο σχήμα 1.4(α) παρουσιάζεται ένας συγκριτής ενώ στο σχήμα 1.4(β) δίνεται ένα παράδειγμα δικτύου ταξινόμησης τεσσάρων εισόδων.




Σχήμα 1.4: (α) Συγκριτής (β) Δίκτυο ταξινόμησης 4 εισόδων

Η ποιότητα των δικτύων ταξινόμησης χαρακτηρίζεται από το χρόνο που χρειάζονται για να ταξινομήσουν τις εισόδους τους (*βάθος* δικτύου) και το πλήθος των συγκριτών που χρησιμοποιούν (*μέγεθος* δικτύου). Το βάθος του δικτύου ορίζεται ως το μέγιστο βάθος των γραμμών του, ενώ το βάθος μίας γραμμής είναι το πλήθος των συγκρίσεων που πραγματοποιούνται σ' αυτή. Για παράδειγμα, το βάθος και το μέγεθος του δικτύου του σχήματος 1.4(β) είναι 3 και 5 αντίστοιχα. Το μέγεθος αντιστοιχεί στον σειριακό χρόνο ταξινόμησης και το βάθος στον παράλληλο.

## 1.5 Ταξινόμηση treesort με δένδρο δυαδικής αναζήτησης

Θα δώσουμε έναν αλγόριθμο ταξινόμησης που βασίζεται στα δένδρα δυαδικής αναζήτησης (binary search tree).

Δίνεται μία λίστα αριθμών.

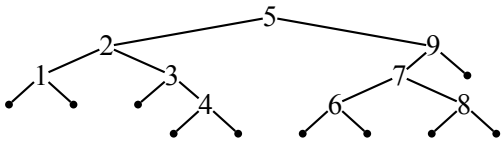
Παίρνουμε ένα ένα τα στοιχεία της λίστας με την σειρά και κατασκευάζουμε ένα δυαδικό δένδρο αναζήτησης. Αυτό κατασκευάζεται ως εξής: Το πρώτο στοιχείο της λίστας τοποθετείται στην ρίζα του δένδρου. Τα υπόλοιπα στοιχεία διασχίζουν το δένδρο μέσω των κόμβων μέχρι να βρουν την θέση τους σε κάποιο φύλλο του δένδρου ως εξής: αν το ήδη τοποθετημένο στοιχείο στον κόμβο είναι μεγαλύτερο από το στοιχείο που επιδιώκουμε να τοποθετήσουμε το προωθούμε στο δεξιό υποδένδρο του κόμβου, αλλιώς στο αριστερό. Επίσης, σε κάθε νέο στοιχείο

που τοποθετούμε στο δένδρο φροντίζουμε να δημιουργούμε δύο κόμβους παιδιά οι οποίοι είναι κενοί (empty).

Μετά διασχίζουμε το δένδρο με την λεγόμενη ενδοδιατεταγμένη (inorder) σειρά, οπότε προκύπτουν τα στοιχεία ταξινομημένα. Μία αναδρομική διαδικασία που τυπώνει τα στοιχεία του δένδρου σε ενδοδιατεταγμένη σειρά είναι η εξής:

```
procedure inorder(t: treenode)
begin
    if t is not empty then
    begin
        inorder(left branch of t);
        write(element at t);
        inorder(right branch of t)
    end
end
```

Το δυαδικό δένδρο που προκύπτει από την εκτέλεση του αλγορίθμου για είσοδο: 5, 2, 3, 9, 7, 1, 8, 4, 6 φαίνεται στο σχήμα 1.5. Εκτελέστε ενδοδιατεταγμένη διάσχιση για να δείτε ότι όντως τα στοιχεία τυπώνονται ταξινομημένα.

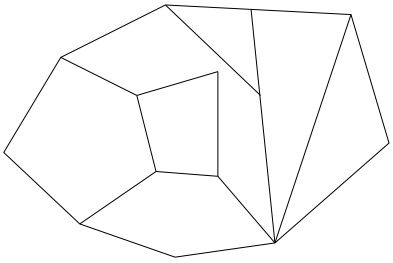



Σχήμα 1.5: Δένδρο αναζήτησης

Ο αλγόριθμος δεν είναι βέλτιστος. Η κατασκευή του δυαδικού δένδρου, αν αυτό δεν προκύψει ισορροπημένο, μπορεί να πάρει $O(n^2)$ βήματα.

## 1.6 Το θεώρημα τεσσάρων χρωμάτων (four color theorem)

Αναφερόμαστε στους γνωστούς επίπεδους χάρτες, όπως αυτός του σχήματος 1.6, όπου οι φραγμένες περιοχές αντιπροσωπεύουν χώρες.



Σχήμα 1.6: Επίπεδος χάρτης

Το ερώτημα είναι (δεδομένου ενός τέτοιου χάρτη):

*Πόσα χρώματα αρκούν για τον χρωματισμό όλων των χωρών, ούτως ώστε χώρες που συνορεύουν (δηλαδή έχουν γραμμή, όχι απλώς σημείο για κοινό σύνορο) να έχουν διαφορετικό χρώμα;*

Με την πρώτη ματιά, φαίνεται ότι το πλήθος των απαιτούμενων χρωμάτων μπορεί να γίνει πολύ μεγάλο αν δοθεί κάποιος ιδιαίτερα περίπλοκος χάρτης. Παρ' όλα αυτά, *είναι αρκετά τέσσερα χρώματα, σε κάθε περίπτωση*. Η παραπάνω πρόταση είναι γνωστή ως *θεώρημα τεσσάρων χρωμάτων*. Δοκιμάστε, για παράδειγμα, να χρωματίσετε τον χάρτη του σχήματος 1.6 με τέσσερα χρώματα.

Το πρόβλημα (αν αρκούν τέσσερα χρώματα) τέθηκε για πρώτη φορά το 1852 από τον Guthrie και αργότερα από τον Cayley το 1878. Ένα χρόνο μετά, ο Kempe παρουσίασε μία απόδειξη, η οποία όμως ήταν λανθασμένη. Σημειωτέον ότι η απόδειξη ότι αρκούν πέντε χρώματα είναι σχετικά εύκολη και την έδωσε ο Heawood το 1890, μεταβάλλοντας κάπως την λανθασμένη απόδειξη του Kempe. Από τότε, πολλοί επιδίωξαν να αποδείξουν το θεώρημα των τεσσάρων χρωμάτων.

Η πρώτη γενικά αποδεκτή απόδειξη δημοσιεύτηκε από τους Appel και Haken το 1977. Σε γενικές γραμμές, η απόδειξη λέει ότι αν είναι δυνατόν να χρωματιστεί κάθε ένας χάρτης από ένα πεπερασμένο σύνολο περιπτώσεων, τότε μπορεί να χρωματιστεί οποιοσδήποτε χάρτης. Το πρόβλημα είναι ότι πρέπει να αποδειχθεί για κάθε χάρτη από το πεπερασμένο σύνολο περιπτώσεων ότι αρκούν τέσσερα χρώματα και στην απόδειξη των Appel και Haken οι περιπτώσεις αυτές είναι περίπου 1700 το πλήθος και ορισμένες από αυτές αρκετά περίπλοκες. Λόγω του μεγάλου πλήθους των περιπτώσεων, ο χρωματισμός των περιπτώσεων γίνεται με την βοήθεια υπολογιστή. Επιπλέον και οι 1700 περίπου περιπτώσεις έχουν προκύψει με την βοήθεια υπολογιστή. Με λίγα λόγια, δεν φαίνεται να είναι δυνατόν να ελέγξει την απόδειξη κάποιος άνθρωπος χωρίς την βοήθεια του υπολογιστή και για αυτόν τον λόγο η απόδειξη των Appel και Haken δέχθηκε αρκετή κριτική. Η απόδειξη βασίζεται στην ορθότητα (βλέπε ενότητα 1.2.4) των χρησιμοποιηθέντων προγραμμάτων. Όμως, λόγω της περιπλοκότητας των προγραμμάτων, η ορθότητα τους δεν έχει πλήρως αποδειχθεί. Επιπλέον, αν θέλουμε να είμαστε απόλυτα αυστηροί, ούτε για τους μεταγλωττιστές με τους οποίους μεταφράστηκαν τα προγράμματα έχουμε απόδειξη ορθότητας.

Πάντως, η απόδειξη τελικά έγινε αποδεκτή, ειδικά επειδή επιπλέον έχουν εμφανιστεί και νεότερες αποδείξεις, όπως για παράδειγμα αυτή των Robertson, Sanders, Seymour, Thomas, οι οποίες μειώνουν κάπως τον αριθμό των εξεταζόμενων περιπτώσεων, αλλά και πάλι ο έλεγχος αν αρκούν τέσσερα χρώματα βασίζεται σε υπολογιστή.

## 1.7 Διαδραστικό υλικό - Σύνδεσμοι

- Η ιστοσελίδα http://towersofhanoi.info περιέχει animations και προσομοιώσεις για τον πρόβλημα των Πύργων του Hanoi.

## 1.8 Ασκήσεις

1. Αναδρομή – Επανάληψη – Επαγωγή:

   (α) Εκφράστε τον αριθμό μετακινήσεων δίσκων που κάνει ο αναδρομικός αλγόριθμος για τους πύργους του Ανόι, σαν συνάρτηση του αριθμού των δίσκων $n$.

   (β) Βρείτε τη σχέση ανάμεσα στον αριθμό των μετακινήσεων του αναδρομικού και τον αριθμό των μετακινήσεων του επαναληπτικού αλγορίθμου.

   (γ) Δείξτε ότι ο αριθμός των μετακινήσεων του αναδρομικού αλγορίθμου είναι ο ελάχιστος μεταξύ όλων των δυνατών αλγορίθμων για το πρόβλημα αυτό.

2. Δυαδική αναζήτηση:

   (α) Περιγράψτε (σε ψευδοκώδικα) διαδικασία / συνάρτηση treeinsert($T$, $k$), που να δέχεται ένα δένδρο δυαδικής αναζήτησης $T$ και έναν αριθμό $k$, και να εισάγει τον αριθμό $k$ στην σωστή θέση του δένδρου, ώστε αυτό να διατηρεί την ιδιότητα του δένδρου δυαδικής αναζήτησης. Υποθέστε ότι, δεδομένου του $T$, έχετε πρόσβαση στο 'αριστερό' / 'δεξί' παιδί του $T$ (π.χ. μέσω κάποιας συνάρτησης leftchild($T$) / rightchild($T$)).

   (β) Ποια είναι η πολυπλοκότητα της συνάρτησής σας, όταν εκτελεστεί διαδοχικά για $n$ αριθμούς; Μελετήστε την πολυπλοκότητα χειρότερης περίπτωσης και, προαιρετικά, την πολυπλοκότητα της μέσης περίπτωσης (θεωρώντας κάθε διάταξη των $n$ αριθμών εισόδου ισοπίθανη).

   (γ) Υλοποιήστε την συνάρτησή σας σε γλώσσα προγραμματισμού της επιλογής σας, και συνδυάστε την με κατάλληλη διάσχιση ώστε να φτιάξετε μια συνάρτηση treesort που να δέχεται λίστα ακεραίων και να την επιστρέφει ταξινομημένη.

3. Θεώρημα τεσσάρων χρωμάτων:

   (α) Αποδείξτε ότι υπάρχουν περιπτώσεις χαρτών που δεν μπορούν να χρωματιστούν με 3 χρώματα.

   (β) Αποδείξτε ότι 5 χρώματα αρκούν για χρωματισμό οποιουδήποτε χάρτη.

4. Δίκτυα ταξινόμησης:

   (α) Σχεδιάστε ένα δίκτυο ταξινόμησης οκτώ εισόδων. Τι βάθος και τι μέγεθος έχει το δίκτυό σας; Τι σημαίνει αυτό για τον χρόνο σειριακής και τι για τον χρόνο παράλληλης εκτέλεσης του αντίστοιχου αλγορίθμου ή κυκλώματος;

   (β) Μπορείτε να σχεδιάσετε καλύτερη μέθοδο ταξινόμησης οκτώ αριθμών με συγκρίσεις; Ποιος είναι ο αριθμός συγκρίσεων της μεθόδου σας; Είναι ελάχιστος; Αποδείξτε τους ισχυρισμούς σας.

# Βιβλιογραφία

[1] Thomas Cormen, Charles Leiserson, Ronald Rivest and Cliff Stein, “Introduction to Algorithms”, 3rd edition, MIT Press, 2009.

[2] J. Edmonds, “How to Think About Algorithms”, Cambridge University Press, 2008.

# Κεφάλαιο 2

# Σύνολα, Σχέσεις, Συναρτήσεις

Τα σύνολα, οι σχέσεις και οι συναρτήσεις χρησιμοποιούνται ευρύτατα σε κάθε είδους μαθηματικές αναπαραστάσεις και μοντελοποιήσεις. Στη θεωρία υπολογισμού χρησιμεύουν, εκτός των άλλων, και για την αναπαράσταση υπολογιστικών προβλημάτων: μια συνάρτηση περιγράφει ένα υπολογιστικό πρόβλημα όπου σε κάθε έγκυρη είσοδο αντιστοιχεί ακριβώς μία έγκυρη έξοδος. Αν έχουμε περισσότερες έγκυρες εξόδους (για παράδειγμα στο πρόβλημα των 4 χρωμάτων, μπορεί να υπάρχουν περισσότεροι του ενός χρωματισμοί) τότε μπορούμε να περιγράψουμε το πρόβλημα με δυαδική σχέση. Τέλος, υπολογιστικά προβλήματα όπου η απάντηση είναι απλώς 'ναι' ή 'όχι' (προβλήματα απόφασης) μπορούν να περιγραφούν μέσω ενός συνόλου που αποτελείται από όλες τις έγκυρες εισόδους για τις οποίες η απάντηση είναι 'ναι'.

Στην ενότητα αυτή θα δώσουμε τους βασικούς ορισμούς που χρειαζόμαστε για να μπορούμε να συζητάμε για τις έννοιες αυτές.

## 2.1 Σύνολα

Η έννοια του συνόλου είναι θεμελιώδης στα μαθηματικά, και είναι δύσκολο να οριστεί επακριβώς. Ορίζουμε άτυπα ως σύνολο μια συλλογή διακεκριμένων στοιχείων. Συμβολίζουμε τα σύνολα με κεφαλαία γράμματα π.χ. $A, B, C$ κλπ.

*Στοιχεία* ή *μέλη* ενός συνόλου ονομάζουμε τα αντικείμενα που το συνιστούν, τα οποία συμβολίζουμε με μικρά γράμματα. Όταν αντικείμενο $a$ ανήκει σε ένα σύνολο $A$ γράφουμε $a \in A$.

*Παράδειγμα* 2.1. Το σύνολο των φυσικών αριθμών:
$\mathbb{N} = \{0, 1, 2, 3, \dots\}$ π.χ. $1 \in \mathbb{N}$, $0 \in \mathbb{N}$, $\frac{1}{2} \notin \mathbb{N}$.
Το σύνολο των ακέραιων αριθμών: $\mathbb{Z} = \{\dots, -3, -2, -1, 0, 1, 2, 3, \dots\}$.

Τα στοιχεία ενός συνόλου καταγράφονται είτε με *αναγραφή*, δηλαδή παραθέτοντας όλα τα στοιχεία μεταξύ αγκυλών χωρισμένα με κόμματα, και στην περίπτωση συνόλων με άπειρα στοιχεία, συμβολίζοντας με τελείες μία σαφώς εννοούμενη συνέχεια όπως στο παραπάνω παράδειγμα, είτε με *περιγραφή*, ορίζοντας μία ιδιότητα $P$, και θεωρώντας το σύνολο των στοιχείων που ικανοποιούν αυτή την ιδιότητα:

*Παράδειγμα* 2.2. Έστω $P(x) : x^2 - 2x + 1 = 0$. Τότε γράφοντας:
$A = \{x : \ x \in \mathbb{N} \wedge P(x)\}$, [1] εννοούμε ότι το $A$ αποτελείται από τους φυσικούς αριθμούς που είναι ρίζες της εξίσωσης $x^2 - 2x + 1 = 0$. Επομένως, $A = \{1\}$. Το σύμβολο $\wedge$ είναι το λογικό "και" και το σύμβολο $\vee$ είναι το λογικό "ή".
Το σύνολο των ρητών αριθμών: $\mathbb{Q} = \{\frac{m}{n} : \ m \in \mathbb{Z} \wedge n \in \mathbb{N} \wedge n \geq 1\}$.

## Ορισμοί

Ένα σύνολο $A$ είναι *υποσύνολο* ενός συνόλου $B$ αν και μόνο αν κάθε στοιχείο του $A$ ανήκει και στο $B$, δηλαδή συμβολικά $A \subseteq B : \forall x \ (x \in A \Rightarrow x \in B)$.
Δύο σύνολα είναι *ίσα* αν και μόνο αν το ένα είναι υποσύνολο του άλλου.
Συμβολικά: $A = B \Leftrightarrow A \subseteq B \ \wedge \ B \subseteq A$. *Πράξεις* μεταξύ συνόλων:

- *Ένωση* (Union): $A \cup B = \{x : \ x \in A \vee x \in B\}$
- *Τομή* (Intersection): $A \cap B = \{x : \ x \in A \wedge x \in B\}$
- *Διαφορά* (Difference): $A \setminus B = \{x : \ x \in A \wedge x \notin B\}$

Δύο σύνολα με ιδιαίτερη σημασία είναι το *κενό σύνολο*, το οποίο δεν περιέχει κανένα στοιχείο και συμβολίζεται με $\emptyset$, και το *δυναμοσύνολο* (powerset) ενός συνόλου $A$ (συμβ. $Pow(A)$ ή $2^A$), που είναι το σύνολο όλων των υποσυνόλων του:

$$Pow(A) = 2^A = \{B : \ B \subseteq A\}$$

Το *συμπλήρωμα* (complement) ενός συνόλου $A$ ως προς ένα σύνολο-σύμπαν $U$, ορίζεται ως

$$A^c = \overline{A} = \{x \in U : \ x \notin A\}$$

*Παράδειγμα* 2.3. Έστω $A = \{1, 2, a, b\}$, $B = \{2, 1, c, d\}$. Τότε:
$A \cup B = \{1, 2, a, b, c, d\}, \quad A \cap B = \{1, 2\}$.
$A \setminus B = \{a, b\}, \quad Pow(A \cap B) = \{\emptyset, \{1\}, \{2\}, \{1, 2\}\}$.
$C = \{x : \ x \in \mathbb{N} \wedge x < 0\} = \emptyset$
$\overline{\mathbb{Q}} = \{x : \ x \in \mathbb{R} \ \wedge \ x \notin \mathbb{Q}\} = \mathbb{R} \setminus \mathbb{Q}$: σύνολο των άρρητων αριθμών.
Ισχύουν τα παρακάτω:
$\mathbb{N} \subseteq \mathbb{Z} \subseteq \mathbb{Q} \subseteq \mathbb{R}$, και
$\mathbb{Q} \cup \overline{\mathbb{Q}} = \mathbb{R}$.

Τα σύνολα $A$ και $B$ λέγονται *ξένα* ανν $A \cap B = \emptyset$.

**Θεώρημα 2.4.** *Ισχύουν οι ακόλουθες ιδιότητες για οποιαδήποτε σύνολα $A, B$:*

- $A \cup A = A$

[1] Συχνά χρησιμοποιείται και ο συμβολισμός '|' αντί για ':', για παράδειγμα $A = \{x \mid x \in \mathbb{N} \wedge P(x)\}$.

- $A \cap A = A$
- $A \cup \emptyset = A$
- $A \cap \emptyset = \emptyset$
- $A \subseteq B \Rightarrow A \cap B = A$
- $A \subseteq B \Rightarrow A \cup B = B$
- $A \subseteq B \Rightarrow \overline{B} \subseteq \overline{A}$ *(όπου τα συμπληρώματα ορίζονται ως προς το ίδιο σύμπαν U)*
- $A \subseteq B \Rightarrow A \cup (B \setminus A) = B$

Ο αριθμός των στοιχείων ενός συνόλου ονομάζεται *πληθικότητα* (cardinality) του συνόλου και συμβολίζεται με $|A|$. Όταν το σύνολο έχει πεπερασμένο αριθμό στοιχείων τότε $|A| \in \mathbb{N}$. Αν το σύνολο έχει άπειρα στοιχεία, τότε γράφουμε συμβολικά $|A| = \infty$.

*Παράδειγμα* 2.5. $|\{a, b, c\}| = |\{2, 3, 4\}| = 3$, $|\mathbb{N}| = \infty$

Ισχύουν: $|A \cup B| \leq |A| + |B|$, $|A \cap B| \leq |A|$, $|A \setminus B| \leq |A|$,
$|A \setminus B| \geq |A| - |B|$, $|Pow(A)| = 2^{|A|}$.
Στον ορισμό του συνόλου που δώσαμε, δεν υπάρχει διάταξη μεταξύ των στοιχείων του, αφού $A = \{a, b\} = \{b, a\}$. Για να ενσωματώσουμε την σειρά των στοιχείων στο σύνολο $A$, ορίζουμε το *διατεταγμένο ζεύγος* των $a, b$ ως:

$$(a, b) = \{\{a\}, \{a, b\}\}$$

Παρατηρήστε ότι $(a, b) \neq (b, a)$, αφού $\{\{a\}, \{a, b\}\} \neq \{\{b\}, \{a, b\}\}$.
Το *καρτεσιανό γινόμενο* δύο συνόλων $A, B$ ορίζεται ως

$$A \times B = \{(a, b) : \ a \in A \ \wedge \ b \in B\}$$

Παρατηρούμε ότι $|A \times B| = |A| \cdot |B|$. Το καρτεσιανό γινόμενο του $A$ με τον εαυτό του: $A^2 = A \times A$.

*Παράδειγμα* 2.6. Έστω $A = \{1, 2\}$ και $B = \{a, b, c, \}$. Τότε:
$A \times B = \{(1, a), (1, b), (1, c), (2, a), (2, b), (2, c)\}$. Βλέπουμε εύκολα ότι $|A \times B| = 6 = |A| \cdot |B|$.

Το διατεταγμένο ζεύγος και το καρτεσιανό γινόμενο εύκολα γενικεύονται:
$A_1 \times \cdots \times A_n = \{(a_1, \ldots, a_n) : \ a_i \in A_i, \text{ για κάθε } 1 \leq i \leq n\}$,
$A^n = \underbrace{A \times \cdots \times A}_{n \text{ φορές}}$

## 2.2 Σχέσεις

Συχνά είναι απαραίτητο να συνδέσουμε τα στοιχεία κάποιων συνόλων, ή ακόμα και τα στοιχεία ενός συνόλου μεταξύ τους. Για παράδειγμα, μεταξύ του συνόλου των μαθητών μιας τάξης και του συνόλου των θρανίων της, σε κάθε μαθητή αντιστοιχεί μία (ή περισσότερες) θέσεις.

**Ορισμός 2.1.** *Δυαδική σχέση* (binary relation) $R$ από το σύνολο $A$ στο σύνολο $B$ είναι ένα υποσύνολο του $A \times B$. Αν $(a, b) \in R$, γράφουμε $aRb$.

Στον παραπάνω ορισμό, για $A = B$ έχουμε μια σχέση μεταξύ των στοιχείων του συνόλου $A$: $R \subseteq A^2$.
Αν η $R$ είναι σχέση, τότε ορίζουμε την αντίστροφή της $R^{-1}$ ως εξής:

$$R^{-1} = \{(b, a) : \ (a, b) \in R\}$$

Προφανώς $xRy \Leftrightarrow yR^{-1}x$.
Μια σχέση $R$ ονομάζεται:

- *Ανακλαστική* (reflexive), αν
  $\forall x \in A \ (xRx)$ .
- *Συμμετρική* (symmetric), αν $\forall x, y \in A \ (xRy \Rightarrow yRx)$.
- *Αντισυμμετρική* (antisymmetric), αν $\forall x, y \in A (xRy \ \wedge \ yRx \Rightarrow x = y)$.
- *Μεταβατική* (transitive), αν $\forall x, y, z \in A \ (xRy \ \wedge \ yRz \Rightarrow xRz)$.

**Ορισμός 2.2.** *Κλείσιμο* (closure) ή *κλειστότητα* μιας σχέσης $R$ ως προς μια ιδιότητα $P$ είναι η ελάχιστη σχέση $R'$ που περιέχει την $R$ ($R \subseteq R'$) και για την οποία ισχύει η ιδιότητα $P$. Με τον όρο ελάχιστη εννοούμε ότι η $R'$ περιέχεται σε οποιαδήποτε άλλη σχέση $R'' \supseteq R$ για την οποία ισχύει η $P$.

Το ανακλαστικό κλείσιμο μιας σχέσης $R \subseteq A^2$ είναι το κλείσιμο της $R$ ως προς την ανακλαστική ιδιότητα. Αντίστοιχα ορίζονται το συμμετρικό και το μεταβατικό κλείσιμο.

*Παράδειγμα* 2.7. Το ανακλαστικό κλείσιμο της $R = \{(a, b), (a, c), (c, a)\}$ είναι το $R_r = \{(a, b), (a, c), (c, a), (a,$
ενώ το συμμετρικό κλείσιμο είναι η $R_s = \{(a, b), (a, c), (c, a), (b, a)\}$.
Για κάθε δυαδική σχέση $R$, το συμμετρικό της κλείσιμο είναι $R_s = R \cup R^{-1}$.

**Ορισμός 2.3.** Μια σχέση $R$ ονομάζεται *σχέση ισοδυναμίας* (equivalence relation) αν είναι:

1. Ανακλαστική
2. Συμμετρική
3. Μεταβατική

*Παράδειγμα* 2.8. 'Εστω $R$ η σχέση της ισότητας σε ένα σύνολο $A$, δηλαδή $xRy$ αν και μόνο αν $x = y\ \forall x, y \in A$. Η ισότητα είναι σχέση ισοδυναμίας, αφού είναι:

1. Ανακλαστική: $\forall x \in A\ (x = x)$.
2. Συμμετρική: $\forall x, y \in A\ (x = y \Rightarrow y = x)$.
3. Μεταβατική: $\forall x, y, z \in A\ (x = y \wedge y = z \Rightarrow x = z)$.

Έστω $A$ το σύνολο των φοιτητών του ΕΜΠ, και η σχέση $R$ στο $A$, που συνδέει δύο φοιτητές αν φοιτούν στην ίδια σχολή του ΕΜΠ. Εύκολα βλέπουμε ότι η $R$ είναι σχέση ισοδυναμίας. Έστω $a \in A$. Μπορούμε να θεωρήσουμε το σύνολο όλων των φοιτητών που συνδέονται με τον $a$ μέσω της $R$, δηλαδή όλων των φοιτητών που βρίσκονται στην ίδια σχολή με τον $a$. Το σύνολο αυτό ονομάζεται *κλάση ισοδυναμίας* του $a$, και συμβολίζεται, για μια δεδομένη σχέση $R$, ως $[a] = \{b :\ b \in A\ \wedge\ aRb\}$.
Παρατηρούμε ότι η σχέση $R$ χωρίζει το σύνολο των φοιτητών του ΕΜΠ σε υποσύνολα φοιτητών κάθε σχολής, τα οποία είναι ξένα μεταξύ τους, αφού δεν μπορεί ένας φοιτητής να ανήκει σε δύο σχολές. Το επόμενο θεώρημα μας δείχνει ότι μια σχέση ισοδυναμίας $R$ *διαμερίζει* το $A$ σε ξένες μεταξύ τους κλάσεις ισοδυναμίας:

**Θεώρημα 2.9.** *Έστω $R$ σχέση ισοδυναμίας στο $A$. Τότε τα επόμενα είναι ισοδύναμα:*

1. $aRb$
2. $[a] = [b]$
3. $a \in [b]$
4. $[a] \cap [b] \neq \emptyset$

*Σημείωση* 2.1. Η ελάχιστη σχέση ισοδυναμίας που περιέχει μια σχέση $R$ είναι το ανακλαστικό, συμμετρικό, και μεταβατικό κλείσιμο της $R$.

**Ορισμός 2.4.** Μια σχέση $R$ ονομάζεται *μερική διάταξη* (partial order) αν είναι:

1. Ανακλαστική
2. Αντισυμμετρική
3. Μεταβατική

*Παράδειγμα* 2.10. Η σχέση $\leq$ στο $\mathbb{N}$ είναι μερική διάταξη, αφού είναι:

1. Ανακλαστική: $\forall x \in \mathbb{N}\ (x \leq x)$.
2. Αντισυμμετρική: $\forall x, y \in \mathbb{N}\ (x \leq y \wedge y \leq x \Rightarrow x = y)$.
3. Μεταβατική: $\forall x, y, z \in \mathbb{N}\ (x \leq y \wedge y \leq z \Rightarrow x \leq z)$.

*Σημείωση* 2.2. Σε μια μερική διάταξη δεν είναι απαραίτητο κάθε δύο στοιχεία $x, y$ να είναι *συγκρίσιμα*, δηλαδή να ισχύει $x \leq y \vee y \leq x$.

Αν συμβαίνει αυτό, δηλαδή όλα τα στοιχεία να είναι συγκρίσιμα ανά δύο, τότε μιλάμε για *ολική διάταξη* (total order).

*Παράδειγμα* 2.11. Η σχέση $\leq$ στο $\mathbb{Q}$ είναι ολική διάταξη.

Η σχέση $\subseteq$ στο $Pow(A)$, για κάθε σύνολο είναι μερική διάταξη, αλλά όχι ολική.

Η διαιρετότητα στο σύνολο των φυσικών αριθμών είναι μερική διάταξη, αλλά όχι ολική.

## 2.3 Συναρτήσεις

Μία σχέση $f \subseteq A \times B$ ονομάζεται *μερική συνάρτηση* (partial function), αν για κάθε $a \in A$ υπάρχει το πολύ ένα $b \in B$ ώστε $(a, b) \in f$. Κάθε μερική συνάρτηση $f$ έχει πεδίο ορισμού $domain(f) = \{a : (a, b) \in f \text{ για κάποιο } b \in B\}$ και πεδίο τιμών $range(f) = \{b : (a, b) \in f \text{ για κάποιο } a \in A\}$. Γράφουμε $f(a) = b$ συνήθως, αντί $(a, b) \in f$.

Αν $domain(f) = A$, τότε η $f$ ονομάζεται *ολική* (total), και αν $range(f) = B$, τότε η $f$ ονομάζεται *επί* (surjection).

Επίσης, μια συνάρτηση $f$ ονομάζεται "ένα-προς ένα" (συμβ 1-1,injection) αν: $a_1 \neq a_2 \Rightarrow f(a_1) \neq f(a_2)$ (για κάθε $b \in B$ υπάρχει το πολύ ένα $a \in A$ ώστε $f(a) = b$).

Αν η $f$ είναι 1-1, τότε ορίζεται η *αντίστροφή* της $f^{-1}$: $f^{-1}(b) = a$ ανν $f(a) = b$. Εύκολα βλέπουμε ότι η $f^{-1}$ είναι μερική συνάρτηση.

Τέλος, μία συνάρτηση $f$ ονομάζεται *αμφιμονοσήμαντη* (bijection), αν είναι 1-1 και επί.

## 2.4 Διαδραστικό υλικό - Σύνδεσμοι

1. Στην ιστοσελίδα http://www.settheory.net/ θα βρείτε εφαρμογές της Θεωρίας Συνόλων στα Θεμέλια των Μαθηματικών.

## 2.5 Ασκήσεις

1. Καταγράψτε με αναγραφή τα παρακάτω σύνολα:
   (α) $\{x \mid x$ ακέραιος τέτοιος ώστε $x^2 = 2\}$
   (β) $\{x \mid x$ άρτιος φυσικός μικρότερος του $10\}$
   (γ) $\{x \mid x$ τετράγωνο ακεραίου και $x < 100\}$
2. Τι μπορείτε να αποφανθείτε για τα σύνολα $A$ και $B$ αν:
   (α) $A \cup B = B$;
   (β) $A \cap B = B$;
   (γ) $A \cap B = A \cup B$;

(δ) $A \setminus B = B$;

3. Δείξτε ότι για τα σύνολα $A, B, C$ ισχύει

$$(A \setminus B) \setminus C = A \setminus (B \setminus C)$$

4. Ποιά είναι η πληθικότητα των ακόλουθων συνόλων:

(α) $\emptyset$

(β) $\{\emptyset, \{\emptyset\}\}$

(γ) $Pow(\{x, y, \{x, y\}\})$

5. Δείξτε ότι αν για σύνολα $A, B, C$ ισχύει ότι $A \subseteq B$ και $B \subseteq C$, τότε και $A \subseteq C$.

6. (*Παράδοξο του Russell*) Έστω $X$ ένα σύνολο συνόλων, το οποίο περιέχει ένα σύνολο $x$ αν το $x$ δεν ανήκει στον εαυτό του, δηλαδή: $X = \{x \mid x \notin x\}$. Δείξτε ότι αν $X \in X$ ή αν $X \notin X$ οδηγούμαστε σε άτοπο. (Παρατηρήστε ότι το σύνολο $X$ δεν μπορεί να οριστεί με αυτό τον τρόπο. Το παράδοξο αυτό οδήγησε στην ανάπτυξη αξιωμάτων για την Συνολοθεωρία.)

7. Έστω $A$ το σύνολο των ανθρώπων. Ορίζουμε μια διμελή σχέση $R$ στο $A$:" $(a, b) \in R$ αν ο/η $a$ είναι αδερφός/ή του $b$". Είναι η σχέση $R$ σχέση ισοδυναμίας; Απαντήστε το ερώτημα και για την σχέση " $(a, b) \in R$ αν ο $a$ είναι πατέρας του/της $b$".

8. Έστω $A$ ένα μη-κενό σύνολο, και $f$ μία συνάρτηση με πεδίο ορισμού το $A$. Έστω $R$ η σχέση: "$(x, y) \in R$ αν $f(x) = f(y)$". Δείξτε ότι η $R$ είναι σχέση ισοδυναμίας, και βρείτε τις κλάσεις ισοδυναμίας της.

9. Έστω μια σχέση $R \subseteq A \times A$. Ποιό είναι το ανακλαστικό κλείσιμο της $R$; Δώστε το με περιγραφικό τρόπο.

10. Βρείτε την μικρότερη σχέση ισοδυναμίας στο σύνολο $A = \{0, 1, 2, 3, 4\}$ που να περιέχει την σχέση $\{(0, 1), (0, 2), (3, 4)\}$.

# Βιβλιογραφία

[1] C.L. Liu. Στοιχεία Διακριτών Μαθηματικών (απόδοση στα Ελληνικά: Κ. Μπους και Δ. Γραμμένος). Πανεπιστημιακές Εκδόσεις Κρήτης, 2003.

[2] K.H. Rosen. Discrete Mathematics and its Applications (6th Edition). McGraw-Hill, 2007.

# Κεφάλαιο 3

# Γραφήματα

## 3.1 Εισαγωγικές έννοιες – Ορισμός

**Ορισμός 3.1.** Γράφος (ή γράφημα) $G$, ονομάζεται ένα διατεταγμένο ζεύγος συνόλων $(V, E)$, όπου $V$ είναι μη κενό σύνολο στοιχείων και $E$ ένα σύνολο μη διατεταγμένων ζευγών του $V$, δηλαδή

$$E \subseteq \begin{pmatrix} V \\ 2 \end{pmatrix}$$

*Παράδειγμα* 3.2. Αν $V = \{v_1, v_2, v_3, v_4, v_5\}$ είναι ένα μη κενό σύνολο στοιχείων και $E = \{\{v_1, v_2\}, \{v_1, v_3\}, \{v_4, v_5\}\}$ τότε το διατεταγμένο ζεύγος $G = (V, E)$ είναι ένας γράφος.

Τα στοιχεία του μη κενού συνόλου $V$ λέγονται **κορυφές** ή **κόμβοι** (*vertices, nodes*) του γράφου. Τα στοιχεία του συνόλου $E$ λέγονται **ακμές** (*edges*) και μπορούν να συμβολιστούν και με ένα γράμμα, π.χ. $e$, όπου $e = \{x, y\}, x, y \in V$, $x \neq y$. Καμιά φορά, καταχρηστικώς, θεωρούμε και ακμές-βρόχους, δηλαδή $e = \{x, x\}$.

Αν $e = \{v_1, v_2\}$ είναι ακμή ενός γράφου $G$, αυτή ενώνει ή συνδέει τις κορυφές $v_1, v_2$ του $G$ και μπορεί να συμβολιστεί επίσης ως $v_1v_2$ ή $v_2v_1$. Οι κορυφές $v_1, v_2$ λέγονται **άκρα** (*endpoints*) της ακμής $e$. Επειδή δε η ακμή $e$ τις συνδέει, λέγονται **γειτονικές** (*adjacent*) κορυφές στο $G$.

Αν τώρα $v_1, v_2$ είναι γειτονικές κορυφές στο $G$, τότε η ακμή $v_1v_2$ προσπίπτει (*incident*) στις $v_1$ και $v_2$. Δύο ακμές που προσπίπτουν στην ίδια κορυφή είναι **γειτονικές** ακμές στο $G$.

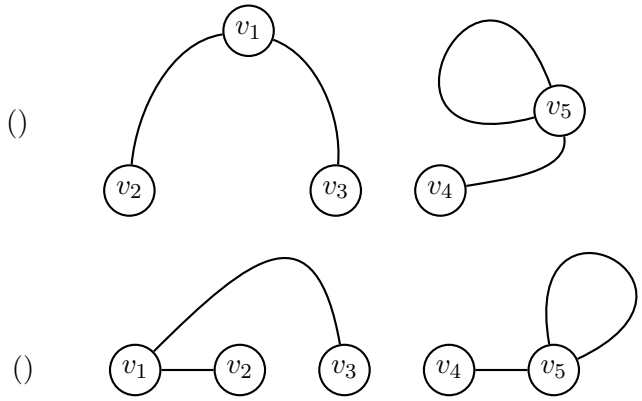
Ο ορισμός του γράφου όπως δόθηκε παραπάνω, δεν διευκολύνει την εποπτική αντίληψη του όρου. Είναι δυνατόν και πολλές φορές επιβάλλεται, για την αναγνώριση και τη μελέτη ιδιοτήτων των γράφων, η απεικόνιση αυτών με τη βοήθεια διαγράμματος.

Για την κατασκευή του διαγράμματος, κάθε κορυφή του γράφου τη σχεδιάζουμε με ένα σημείο, μία κουκίδα και κάθε ακμή με ένα τμήμα καμπύλης γραμμής. Από τον τρόπο κατασκευής του διαγράμματος, είναι φανερό πως δεν υπάρχει μοναδικός τρόπος σχεδίασης ενός γράφου.

*Παράδειγμα* 3.3. Το διάγραμμα του γραφήματος $G$ με

$$E = \{\{v_1, v_2\}, \{v_1, v_3\}, \{v_4, v_5\}, \{v_5, v_5\}\}$$

μπορεί να είναι αυτό που φαίνεται στο σχήμα 3.1(α) ή αυτό που φαίνεται στο σχήμα 3.1(β). Οι κορυφές $v_1, v_2$ είναι γειτονικές στο $G$ ενώ οι $v_3, v_4$ δεν είναι. Οι ακμές $v_1v_2, v_1v_3$ είναι γειτονικές στο $G$ ενώ οι $v_4v_5, v_1v_2$ δεν είναι.




Σχήμα 3.1: Δύο διαφορετικά διαγράμματα για τον γράφο $G$

Ο αριθμός των κορυφών ενός γράφου $G(V, E)$ ονομάζεται **τάξη** (*order*) του $G$ και συμβολίζεται με $|V|$ και ο αριθμός των ακμών του, **μέγεθος** (*size*) του $G$ και συμβολίζεται με $|E|$. Στην Πληροφορική όμως, συνήθως ονομάζουμε μέγεθος το $n = |V|$.

*Παράδειγμα* 3.4. Στο γράφο $G$ του Παραδείγματος 2 η τάξη του ισούται με 5 και το μέγεθος με 4. Ο γράφος $G$ μπορεί επίσης να συμβολιστεί και με $G(5, 4)$.

*Παρατήρηση* 3.5. Από τον ορισμό του γράφου προκύπτει ότι μία ακμή δεν μπορεί να έχει ως άκρα την ίδια κορυφή. Συχνά όμως στην Πληροφορική, όπως αναφέραμε και πιο πάνω, χρειαζόμαστε μια τέτοια ακμή. Η ακμή τότε λέγεται **βρόχος** (*loop*). Επίσης από τον ορισμό του γράφου προκύπτει ότι δεν είναι δυνατή η ύπαρξη περισσοτέρων ακμών με ίδια άκρα, δηλαδή δεν είναι δυνατή η ύπαρξη παράλληλων ακμών. Στο γράφο του Παραδείγματος 2, εφόσον υπάρχει η ακμή $v_1v_2$ η ύπαρξη μιας παράλληλης της π.χ. $v_2v_1$, αποκλείεται απ' τον ορισμό.

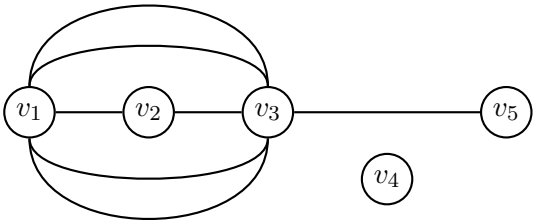
Ένας γράφος ο οποίος δεν έχει βρόχους, λέγεται στην Πληροφορική **απλός γράφος**. Επειδή σε ότι θα αναφερθεί παρακάτω, δεν επηρεάζει η ύπαρξη ή μη βρόχων στους γράφους, χωρίς βλάβη της γενικότητας, θα θεωρούμε στο εξής μόνο απλούς γράφους.

Υπάρχουν όμως και **πολυγραφήματα** (*multigraphs*). Το διάγραμμα ενός πολυγραφήματος μπορεί να περιέχει πολλές ακμές που συνδέουν τις ίδιες κορυφές.

*Παράδειγμα* 3.6. Στο σχήμα 3.2 φαίνεται ένα πολυγράφημα.

## 3.2 Υπογράφος

**Ορισμός 3.7.** Ένας γράφος $G' = (V', E')$ είναι **υπογράφος** (*subgraph*) ενός άλλου γράφου $G = (V, E)$, αν ισχύει $V' \subseteq V$ και $E' \subseteq E$.




Σχήμα 3.2: Πολυγράφημα

*Παράδειγμα* 3.8. Στο σχήμα 3.3 ο $G'$ είναι ένας υπογράφος του $G$.

**Ορισμός 3.9.** Έστω $G' = (V', E')$ υπογράφος ενός γράφου $G = (V, E)$. Αν ισχύει $V' = V$ τότε ο υπογράφος λέγεται **παράγων υπογράφος** (*spanning subgraph*) του γράφου $G$.

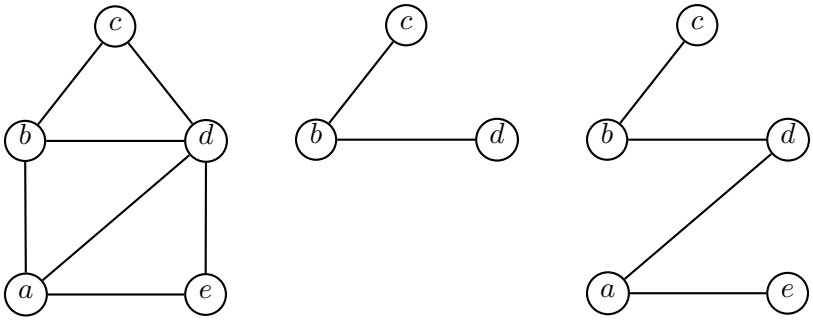
*Παράδειγμα* 3.10. Στο σχήμα 3.3 ο $G''$ είναι παράγων υπογράφος του $G$.

## 3.3 Βαθμός κορυφής

**Ορισμός 3.11.** Έστω ένας γράφος $G = (V, E)$. **Βαθμός** (*degree, valence*) μιας κορυφής $v \in V$ ονομάζεται ο αριθμός των ακμών του $G$ που προσπίπτουν στην $v$ και συμβολίζεται με $d_G(v)$ ή $d(v)$. Ένας γράφος $G(V, E)$, για τον οποίο ισχύει $d(v) = k$ για κάθε κορυφή του, λέγεται $k$**-κανονικός** γράφος.

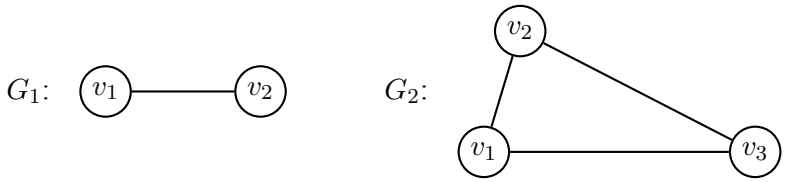
Αποδεικνύεται εύκολα ότι το άθροισμα των βαθμών όλων των κορυφών ενός γράφου, ισούται αριθμητικά με το διπλάσιο του αριθμού των ακμών του. Δηλαδή σε ένα γράφο $G = (V, E)$ έχουμε ότι

$$\sum_{v \in V} d(v) = 2|E|$$




Γράφος $G$ Γράφος $G'$ Γράφος $G''$

Σχήμα 3.3: Ο γράφος $G$ και δύο υπογράφοι αυτού




Σχήμα 3.4: $G_1$: 1–κανονικός και $G_2$: 2–κανονικός γράφος

*Παράδειγμα* 3.12. Στο γράφο $G$ στο σχήμα 3.3 έχουμε $d(b) = 3$, $d(d) = 4$, $d(c) = d(e) = 2$. Στο σχήμα 3.4 ο $G_1$ είναι $1-$κανονικός γράφος και ο $G_2$ είναι $2-$κανονικός γράφος.

## 3.4 Δρόμος - Μονοπάτι - Κύκλος

**Ορισμός 3.13.** Σε ένα γράφο $G$, μια πεπερασμένη ακολουθία εναλλάξ κορυφών και ακμών του $G$ που αρχίζει και τελειώνει σε κορυφή και που κάθε ακμή που περιέχεται στην ακολουθία προσπίπτει στην κορυφή που προηγείται και σε αυτήν που έπεται, λέγεται **δρόμος** ή **διαδρομή** (*walk*) στο $G$.

*Παράδειγμα* 3.14. Στο γράφο $G$ στο σχήμα 3.3 η ακολουθία κορυφών και ακμών του γράφου

$$c\{c,d\}d\{d,b\}b\{b,a\}a\{a,d\}d\{d,b\}b$$

είναι δρόμος στο $G$.

**Ορισμός 3.15.** Αν σε έναν δρόμο ενός γράφου κάθε ακμή του δρόμου εμφανίζεται μόνο μία φορά, ο δρόμος λέγεται **δρομίσκος** ή **μονοπάτι** (*trail*).

*Παράδειγμα* 3.16. Στο γράφο $G$ στο σχήμα 3.3 ο δρόμος

$$d\{d,b\}b\{b,a\}a\{a,d\}d\{d,e\}e$$

είναι δρομίσκος.

**Ορισμός 3.17.** Ένας δρόμος στον οποίο κάθε κορυφή και κάθε ακμή του εμφανίζονται ακριβώς μία φορά, λέγεται **απλό μονοπάτι** (*path*).

*Παράδειγμα* 3.18. Στο γράφο $G$ στο σχήμα 3.3 ο δρόμος

$$a\{a,b\}b\{b,c\}c\{c,d\}d\{d,e\}e$$

είναι απλό μονοπάτι.

**Ορισμός 3.19.** Ένας δρόμος με αρχή και τέλος την ίδια κορυφή, λέγεται **κλειστός δρόμος**, αλλιώς λέγεται **ανοικτός**.

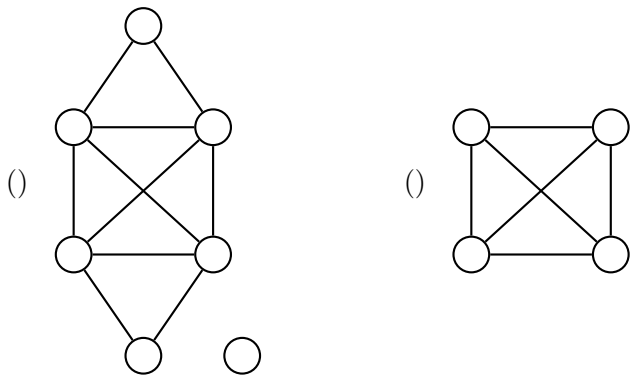
**Ορισμός 3.20.** Ένας δρόμος που είναι κλειστό μονοπάτι λέγεται **κύκλος** (*cycle*). Ένας δρόμος που είναι απλό κλειστό μονοπάτι λέγεται **απλός κύκλος** (*simple cycle*).

*Παράδειγμα* 3.21. Στο γράφο $G$ στο σχήμα 3.3

- ο δρόμος $c\{c,b\}b\{b,d\}$ είναι ανοικτός δρόμος
- ο δρόμος $c\{c,b\}b\{b,d\}d\{d,a\}a\{a,b\}b\{b,c\}c$ είναι κλειστός δρόμος
- ο δρόμος $c\{c,b\}b\{b,d\}d\{d,c\}c$ είναι κύκλος

**Ορισμός 3.22.** Ένας κύκλος που περνά ακριβώς μια φορά από κάθε ακμή ενός γράφου $G$ (χωρίς απαραίτητα να περνά ακριβώς μια φορά και από κάθε κορυφή) ονομάζεται **κύκλος Euler**. Ένας γράφος που έχει κύκλο Euler ονομάζεται **γράφος Euler**. Αποδεικνύεται εύκολα ότι ένας γράφος έχει κύκλο Euler ανν όλες οι κορυφές έχουν άρτιο βαθμό (σχήμα 3.5(α)).

**Ορισμός 3.23.** Ένας κύκλος που περνά ακριβώς μια φορά από κάθε κορυφή ενός γράφου $G$ (χωρίς απαραίτητα να περνά και από όλες τις ακμές) ονομάζεται **κύκλος Hamilton**. Ένας γράφος που έχει κύκλο Hamilton ονομάζεται **γράφος Hamilton** (σχήμα 3.5(β)).



Σχήμα 3.5: (α) Γράφος Euler, (β) Γράφος Hamilton

Σε ένα γράφο $G$, ένας δρόμος μεταξύ δύο κορυφών $u$ και $v$ του $G$ λέγεται και $(u,v)-$δρόμος ή απλούστερα $uv-$δρόμος. Ο αριθμός των ακμών ενός γράφου που εμφανίζονται σε έναν δρόμο του γράφου, λέγεται **μήκος** του δρόμου.

*Παράδειγμα* 3.24. Στο παράδειγμα 3.21 τα μήκη των δρόμων με τη σειρά που εμφανίζονται είναι 2, 5 και 3.

## 3.5 Παράσταση Γράφου

Ένας γράφος μπορεί να παρασταθεί με τη βοήθεια του **πίνακα γειτνίασης** (*adjacency matrix*) ή του **πίνακα πρόσπτωσης** (*incidence matrix*) ή των **λιστών γειτνίασης** (*adjacency lists*).

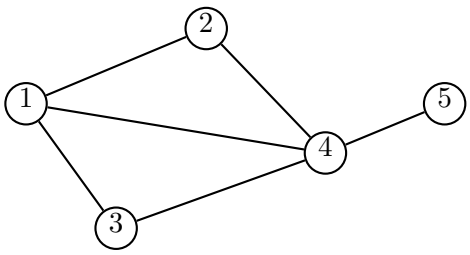
**Ορισμός 3.25** (Πίνακας γειτνίασης)**.** Έστω ένας γράφος $G = (V, E)$ με $V = \{v_1, v_2, ..., v_n\}$. Τότε ο γράφος μπορεί να παρασταθεί με τη βοήθεια ενός $n \times n$ πίνακα $A(G)$, όπου

$$A(G) = [a_{ij}], \quad a_{ij} = \begin{cases} 1, & \text{αν } \{v_i, v_j\} \in E \\ 0, & \text{αλλιώς} \end{cases}$$

Ο πίνακας $A(G)$ λέγεται **πίνακας γειτνίασης** (*adjacency matrix*), και είναι συμμετρικός ($a_{i,j} = a_{j,i}$).

Μια άλλη παράσταση είναι με τις **λίστες γειτνίασης** (*adjacency lists*). Η λίστα γειτνίασης μιας κορυφής $v$ περιέχει όλες τις γειτονικές κορυφές της $v$. Η παράσταση αυτή σε H/Y είναι πιο αποδοτική για **αραιούς** γράφους.

**Ορισμός 3.26.** Οι αραιοί γράφοι έχουν $O(n)$ ακμές ενώ οι πυκνοί γράφοι έχουν $\Omega(n^2)$.




Σχήμα 3.6: Γράφος

*Παράδειγμα* 3.27. Η αναπαράσταση του γράφου που φαίνεται στο σχήμα 3.6 με τον πίνακα γειτνίασης είναι η παρακάτω:

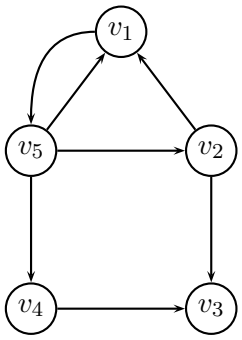
$$A(G) = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Η αναπαράσταση με τις λίστες γειτνίασης είναι η παρακάτω:

$$\begin{array}{lllll} [1]\rightarrow & 2 & 3 & 4 & \\ [2]\rightarrow & 1 & 4 & & \\ [3]\rightarrow & 1 & 4 & & \\ [4]\rightarrow & 1 & 2 & 3 & 5 \\ [5]\rightarrow & 4 & & & \end{array}$$

## 3.6 Προσανατολισμένος Γράφος

Αν στον ορισμό του γράφου αντικαταστήσουμε τα στοιχεία του $E$ με διατεταγμένα ζεύγη στοιχείων του $V$, παίρνουμε ένα **προσανατολισμένο** ή **κατευθυνόμενο** γράφο (*directed graph, digraph*). Δηλαδή $E \subseteq V \times V$.




Σχήμα 3.7: Κατευθυνόμενος γράφος

*Παράδειγμα* 3.28. Ο γράφος στο σχήμα 3.7 είναι ένας προσανατολισμένος γράφος. Αν ο γράφος είναι ο $G = (V, E)$ τότε έχουμε:

$$V = \{v_1, v_2, v_3, v_4, v_5\}$$
$$E = \{(v_5, v_1), (v_1, v_5), (v_2, v_1), (v_5, v_2), (v_2, v_3), (v_5, v_4), (v_4, v_3)\}$$

**Ορισμός 3.29.** Έστω $x$ μια κορυφή ενός προσανατολισμένου γράφου. Ο αριθμός των ακμών που προσπίπτουν (εισέρχονται) σε αυτήν την κορυφή ονομάζεται **προς-βαθμός** (*in-degree*) της κορυφής $x$ και συμβολίζεται με $deg^-(x)$:

$$deg^-(x) = |\{(y, x) : y \in V \text{ και } (y, x) \in E\}|$$

Ο αριθμός των ακμών που ξεκινούν (εξέρχονται) από την κορυφή $x$ ονομάζεται από-βαθμός (*out-degree*) και συμβολίζεται με $deg^+(x)$:

$$deg^+(x) = |\{(x, y) : y \in V \text{ και } (x, y) \in E\}|$$

*Παράδειγμα* 3.30. Στον προσανατολισμένο γράφο στο σχήμα 3.7 έχουμε:

$$deg^-(v_2) = |\{(v_5, v_2)\}| = 1$$
$$deg^+(v_2) = |\{(v_2, v_3), (v_2, v_1)\}| = 2$$

## 3.7 Συνεκτικός Γράφος

**Ορισμός 3.31.** Έστω ένας γράφος $G = (V, E)$. Δύο κορυφές $u$, $v$ του $G$ είναι **συνδεδεμένες** (*connected*) αν υπάρχει τουλάχιστον ένα $uv$-μονοπάτι στο $G$. Η σχέση "σύνδεση δύο κορυφών

στο $G$", είναι μια σχέση ισοδυναμίας στο σύνολο V του $G$, η οποία δημιουργεί μια διαμέριση (*partition*) σε κλάσεις ισοδυναμίας π.χ. τις $V_1, V_2, \ldots, V_k$. Για τις κλάσεις αυτές ισχύει ότι:

$$\begin{array}{ll} V_i \subseteq V & , 1 \leq i \leq k \\ V_i \bigcap V_j = \emptyset & , \forall i, j \\ V_1 \bigcup V_2 \bigcup \ldots \bigcup V_k = V & \end{array}$$

Προφανώς κάθε ζεύγος κορυφών $u, v$ συνδέονται αν και μόνο αν οι κορυφές $u, v$ ανήκουν στην ίδια κλάση ισοδυναμίας $V_i$.

**Ορισμός 3.32.** Έστω ένας γράφος $G = (V, E)$ και $V' \subseteq V$. Ο υπογράφος που έχει σύνολο κορυφών το $V'$ και σύνολο ακμών όλες τις ακμές του $G$, των οποίων και τα δύο άκρα ανήκουν στο $V'$, λέγεται **παραγόμενος υπογράφος** (*induced subgraph*) από τον $V'$ και συμβολίζεται $G[V']$.

**Ορισμός 3.33.** Έστω ένα γράφος $G = (V, E)$ και $V_1, V_2, \ldots, V_k$ οι κλάσεις ισοδυναμίας του V που δημιουργούνται απ' τη σχέση "σύνδεση δύο κορυφών". Τα υπογραφήματα $G[V_1], G[V_2], \ldots, G[V_k]$ λέγονται συνεκτικές συνιστώσες (connected components) του γράφου $G$. Ο αριθμός των συνιστωσών ενός γράφου $G$ συμβολίζεται με $(G)$.

**Ορισμός 3.34.** Ένας γράφος λέγεται **συνεκτικός** αν αποτελείται από μία μόνο συνιστώσα. Αν ο αριθμός των συνιστωσών ενός γράφου είναι μεγαλύτερος από το 1, ο γράφος λέγεται μη συνεκτικός. Είναι φανερό πως ένας γράφος είναι συνεκτικός, αν για κάθε ζεύγος κορυφών του γράφου υπάρχει ένα μονοπάτι τουλάχιστον, που τις συνδέει.

*Παράδειγμα* 3.35. Ο γράφος του σχήματος 3.5α είναι μη συνεκτικός με $(G) = 2$. Ο γράφος του σχήματος 3.5β είναι συνεκτικός με $(G) = 1$.

*Παρατήρηση* 3.36. Στην περίπτωση προσανατολισμένου γράφου οι ακμές σε μονοπάτια (άρα και σε κύκλο) πρέπει να έχουν όλες τον ίδιο προσανατολισμό.

**Ορισμός 3.37.** Ένας προσανατολισμένος γράφος λέγεται **ισχυρά συνεκτικός** (*strongly connected*) αν για κάθε ζεύγος $(u, v)$ υπάρχει μονοπάτι από το $u$ στο $v$. Ο γράφος λέγεται **ασθενώς συνεκτικός** (*weakly connected*) αν για κάθε ζεύγος $(u, v)$ υπάρχει μονοπάτι από το $u$ στο $v$ αν αγνοήσουμε τον προσανατολισμό των ακμών.

**Ορισμός 3.38.** Ένας απλός γράφος $G = (V, E)$ (χωρίς βρόχους και παράλληλες ακμές ) ονομάζεται **πλήρης** όταν δύο οποιεσδήποτε κορυφές του είναι γειτονικές. Για ένα πλήρη γράφο προφανώς ισχύει ότι:

$$E = \begin{pmatrix} V \\ 2 \end{pmatrix} \text{ άρα: } |E| = \begin{pmatrix} |V| \\ 2 \end{pmatrix} = \frac{|V|(|V|-1)}{2}$$

Ο πλήρης γράφος με $n$ κορυφές συμβολίζεται με $K_n$.

**Ορισμός 3.39.** Ένας γράφος $G(V, E)$ ονομάζεται **διμερής** (*bipartite*) αν το σύνολο των κόμβων $V$ μπορεί να διαμεριστεί σε δύο μη κενά υποσύνολα $X$ και $Y$ έτσι ώστε όλες οι ακμές στο $E$ να ενώνουν έναν κόμβο του $X$ με έναν κόμβο του $Y$. Ένας πλήρης διμερής γράφος (δηλαδή ο διμερής γράφος στον οποίο κάθε κορυφή του $X$ ενώνεται με κάθε κορυφή του $Y$) συμβολίζεται με $K_{n,m}$, όπου $n = |X|$ και $m = |Y|$.

**Ορισμός 3.40.** Ένας γράφος ονομάζεται **επίπεδος** (planar) αν μπορεί να σχεδιαστεί στο επίπεδο έτσι ώστε όλες οι ακμές του να μην διασταυρώνονται, φυσικά εκτός από τις κοινές κορυφές τους.

Έχει αποδειχθεί ότι ικανή και αναγκαία συνθήκη για να είναι ένας γράφος επίπεδος είναι να μην περιέχει υπογράφο ομοιομορφικό με τον $K_5$ ή τον $K_{3,3}$ (Θεώρημα Kuratowski).

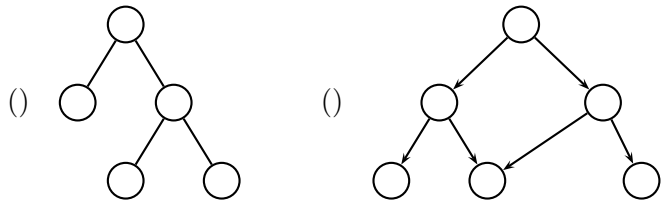
## 3.8 Δέντρα

**Ορισμός 3.41.** **Δένδρο** λέγεται ένας συνεκτικός γράφος που δεν περιέχει κύκλους. **Δάσος** λέγεται κάθε γράφος που δεν περιέχει κύκλους. Οι συνεκτικές συνιστώσες ενός δάσους, είναι δέντρα. Ένα δέντρο στο οποίο ξεχωρίζουμε μια κορυφή, την οποία ονομάζουμε **ρίζα**, λέγεται δέντρο με ρίζα (*rooted tree*).

**Πρόταση 3.42.** *Ένας γράφος είναι δάσος αν και μόνο αν είναι υπογράφος ενός δέντρου.*

Ανάλογοι είναι και οι ορισμοί για προσανατολισμένα δέντρα (δάση). Ο προσανατολισμός θεωρείται από τη ρίζα προς τα φύλλα.

Ένας κατευθυνόμενος γράφος χωρίς κύκλους δεν είναι πάντα δέντρο. Ένας τέτοιος **κατευθυνόμενος ακυκλικός γράφος** (*DAG=Directed Acyclic Graph*) είναι χρήσιμος στην κωδικοποίηση της συντακτικής δομής αριθμητικών εκφράσεων, στην αναπαράσταση μερικών διατάξεων, κ.α. Ένα δέντρο και ένας κατευθυνόμενος ακυκλικός γράφος φαίνονται στο σχήμα 3.8.



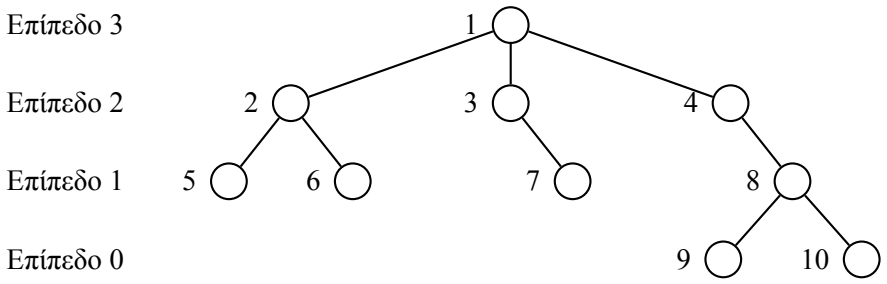

Σχήμα 3.8: (α) Δένδρο, (β) Κατευθυνόμενος ακυκλικός γράφος

**Ορισμός 3.43.** Έστω ένα δέντρο με ρίζα. Αν $x$ και $y$ είναι κορυφές τέτοιες ώστε η $x$ να βρίσκεται στο μονοπάτι που συνδέει τη ρίζα με την $y$, τότε η $x$ ονομάζεται **πρόγονος** (*ancestor*) της $y$ και η $y$ **απόγονος** (*descendant*) της $x$. Αν επιπλέον $x \neq y$ τότε η $x$ είναι **γνήσιος πρόγονος** της $y$ και η $y$ **γνήσιος απόγονος** της $x$. Αν $x$ είναι γνήσιος πρόγονος της $y$ και $\{x, y\}$ είναι ακμή του δέντρου, τότε η $x$ είναι **γονέας** (*parent*)της $y$ και η $y$ **παιδί** (*child*) της $x$. Οι κορυφές με τον ίδιο γονέα λέγονται **αδέλφια** (*siblings*). Οι κορυφές που δεν έχουν απογόνους ονομάζονται **τερματικές ή φύλλα** (*terminals, leaves*). Οι κορυφές που δεν είναι τερματικές, λέγονται **εσωτερικές ή μη τερματικές ή κορυφές κλάδων** (*internals, non-terminals, branch nodes*).

*Παράδειγμα* 3.44. Στο σχήμα 3.9 έχουμε:

- Η κορυφή 1 είναι η ρίζα του δέντρου.

- Η κορυφή 4 είναι πρόγονος της κορυφής 10.
- Η κορυφή 9 είναι απόγονος της κορυφής 4.
- Η κορυφή 3 είναι γονέας της κορυφής 7.
- Οι κορυφές 5 και 6 είναι αδέλφια με γονέα την κορυφή 2.
- Οι κορυφές 2, 3, 4, 8 είναι εσωτερικές κορυφές.
- Οι κορυφές 5, 6, 7, 9, 10 είναι φύλλα.




Σχήμα 3.9: Δένδρο με αριθμημένες κορυφές

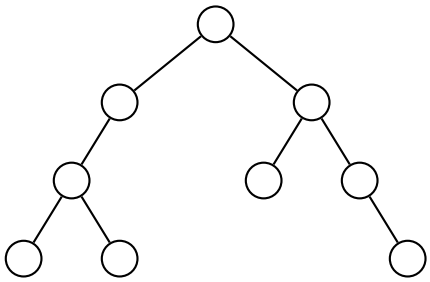
**Ορισμός 3.45.** **Ύψος** (*height*) ενός κόμβου είναι η μέγιστη απόστασή του από απόγονό του. Ύψος δέντρου είναι το ύψος της ρίζας. **Βάθος** (*depth*) ενός κόμβου είναι η απόστασή του από τη ρίζα. **Επίπεδο** (*level*) κόμβου είναι το ύψος του δέντρου πλην το βάθος του κόμβου.

**Ορισμός 3.46.** **Δυαδικό δέντρο** (*binary tree*) είναι ένα δέντρο με ρίζα στο οποίο κάθε κορυφή έχει το πολύ δύο παιδιά (σχήμα 3.10). Ισοδύναμα, δυαδικό δέντρο είναι ένα πεπερασμένο σύνολο κορυφών που είναι ή κενό, ή αποτελείται από τη ρίζα και δύο ξένα μεταξύ τους δυαδικά δέντρα που ονομάζονται το δεξί και το αριστερό υποδέντρο.

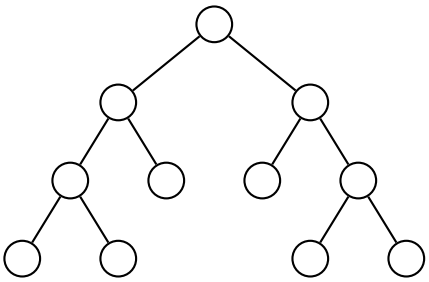
Σε ένα δυαδικό δέντρο στο οποίο κάθε κορυφή του είναι είτε τερματική είτε έχει ακριβώς δύο παιδιά, δεν υπάρχουν **εκφυλισμένοι εσωτερικοί κόμβοι** (*no degenerate branch nodes*) (σχήμα 3.10).

Ένα δυαδικό δέντρο ύψους $k$ το οποίο έχει $2^k$ φύλλα, όλα στο επίπεδο 0, ονομάζεται εντελώς πλήρες δυαδικό δέντρο (fully complete binary). Πλήρες δ.δ.(ή καμιά φορά σχεδόν πλήρες) ονομάζουμε ένα ε.π.δ.δ. που όμως του λείπουν μερικά ακροδεξιά φύλλα. (σχήμα 3.10).

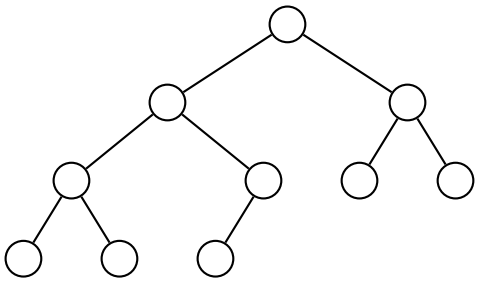
Μια κωδικοποίηση ενός δυαδικού δέντρου στον υπολογιστή μπορεί να γίνει πολύ εύκολα ως εξής: χρησιμοποιούμε ένα μονοδιάστατο πίνακα $A$ και στη θέση $A[1]$ βάζουμε τη ρίζα του δέντρου. Τις θέσεις $A[2]$, $A[3]$ καταλαμβάνουν (αν υπάρχουν) τα δύο παιδιά της ρίζας. Γενικά στις θέσεις $A[2 * k]$, $A[2 * k + 1]$ βρίσκονται τα παιδιά της κορυφής $k$. Συνεπώς ο γονέας της κορυφής $n$ βρίσκεται στη θέση $A[n\ div\ 2]$. Σημειώνουμε ότι η αναπαράσταση αυτή χρησιμοποιείται συνήθως όταν το δέντρο είναι (σχεδόν) πλήρες.



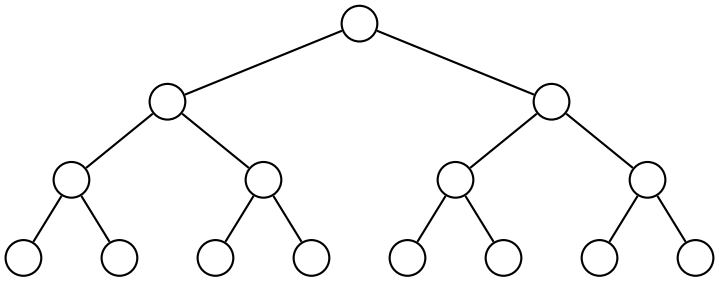
Δυαδικό δένδρο (Binary Tree)



Χωρίς εκφυλισμένους κόμβους



Πλήρες δυαδικό δένδρο
(Complete Binary Tree)



Εντελώς πλήρες δυαδικό δένδρο
(Fully Complete Binary Tree)

Σχήμα 3.10: Δυαδικά δένδρα

## 3.9 Διαδραστικό Υλικό – Σύνδεσμοι

- Διαδραστικό εργαλείο σχεδίασης γράφων Free Graph Theory Software: http://www.free-graph-theory-software.org/
- Ηλεκτρονικό σύγγραμμα για Θεωρία Γραφημάτων του R. Diestel: http://diestel-graph-theory.com/index.html
- Ηλεκτρονικό σύγγραμμα για Θεωρία Γραφημάτων των J.A. Bondy, U.S.R. Murty. http:

//www.iro.umontreal.ca/~hahn/IFT3545/GTWA.pdf

## 3.10 Ασκήσεις

1. Δείξτε ότι ένας ακυκλικός συνεκτικός γράφος με $n$ κόμβους έχει $n-1$ ακμές.

2. Δείξτε ότι ένας συνεκτικός γράφος με $n$ κόμβους και $n-1$ ακμές είναι δέντρο.

3. Δείξτε ότι αν ένας γράφος δεν έχει κύκλους αλλά η πρόσθεση οποιασδήποτε νέας ακμής δημιουργεί κύκλο τότε ο γράφος είναι δέντρο. Δείξτε ότι η αφαίρεση οποιασδήποτε ακμής του κύκλου που σχηματίστηκε κάνει πάλι τον γράφο δέντρο.

4. Σε ένα δένδρο ένας κόμβος λέγεται $1/k$ separator αν μετά την αφαίρεσή του, οι συνεκτικές συνιστώσες που απομένουν έχουν μέγεθος το πολύ $n/k$, όπου $n$ ο αριθμός των κόμβων του δένδρου.

   (α) Δείξτε ότι σε κάθε δένδρο υπάρχει $1/2$ separator.

   (β) Δείξτε ότι αν σε ένα δένδρο υπάρχει $1/k$ separator (υποθέτοντας ότι $k < n$) τότε υπάρχει κόμβος με βαθμό τουλάχιστον $k$. Εξετάστε αν ισχύει και το αντίστροφο.

   (γ) Βρείτε αλγόριθμο που αποφαίνεται αν ένα δένδρο έχει $1/(x+3)$ separator, όπου $x$ το τελευταίο ψηφίο του αριθμού ταυτότητάς σας. Αποδείξτε την ορθότητα του αλγορίθμου σας και υπολογίστε την πολυπλοκότητά του.

5. Ολική καταβόθρα (sink) σε ένα κατευθυνόμενο γράφο λέγεται μια κορυφή που δεν έχει ακμή προς άλλες κορυφές και υπάρχει ακμή από κάθε άλλη κορυφή προς αυτή. Για αναπαραστάση του γράφου με πίνακα γειτνίασης σχεδιάστε όσο το δυνατόν πιο αποδοτικό αλγόριθμο που βρίσκει μια ολική καταβόθρα ή αποφαίνεται ότι δεν υπάρχει. Ποια είναι η πολυπλοκότητα του αλγορίθμου σας;

   Μπορείτε να βρείτε αλγόριθμο με πολυπλοκότητα $O(n)$;

# Βιβλιογραφία


[1] Reinhard Diestel, Graph Theory (3rd edition), Springer 2005.

[2] Frank Harary, Graph Theory. Addison-Wesley Series in Mathematics, 1972.

[3] J.A. Bondy, U.S.R. Murty. Graph Theory with Applications. North Holland, 1976. Διατίθεται ελεύθερα στο διαδίκτυο.

[4] Γιάννης Μανωλόπουλος, Μαθήματα Θεωρίας Γράφων: Θεμελιώσεις - Θεωρία - Εφαρμογές. Εκδοσεις Νεων Τεχνολογιων, ISBN 960-7235-87-8, Εκδοση 2η (2000).

[5] Ronald Graham, Donald Knuth, Oren Patashnik. Συνκριτά Μαθηματικά (δεύτερη έκδοση, μτφ. Χ. Καπούτσης, επιμ. Ε. Ζάχος) Εκδόσεις Κλειδάριθμος, 2011.

[6] S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani. Algorithms, MacGraw-Hill, 2006. Αλγόριθμοι, ελληνική έκδοση, Εκδόσεις Κλειδάριθμος, 2008.

[7] Ε. Ζάχος, Α. Παγουρτζής, Θεμελιώδη Θέματα Επιστήμης Υπολογιστών, Σημειώσεις, ΕΜΠ, 2015.

[8] C.L. Liu. Στοιχεία Διακριτών Μαθηματικών (απόδοση στα Ελληνικά: Κ. Μπους και Δ. Γραμμένος). Πανεπιστημιακές Εκδόσεις Κρήτης, 2003.

[9] K.H. Rosen. Discrete Mathematics and its Applications (6th Edition). McGraw-Hill, 2007.

[10] Η. Κουτσουπιάς. Μαθηματικά της Πληροφορικής. ΕΚΠΑ, 2008.

[11] J.E. Hopcroft and J.D. Ullman. “Introduction to Automata Theory, Languages and Computation”, Addison Wesley Longman, 2007.

[12] M. Sipser. “Introduction to the Theory of Computation”, International Thomson Publishing, 1996.

# Κεφάλαιο 4

# Δομές Δεδομένων

## 4.1 Εισαγωγή

Όπως είναι γνωστό, για να εκτελεστεί ένας αλγόριθμος με Η/Υ θα πρέπει να γραφτεί σε μία αυστηρά ορισμένη γλώσσα Η/Υ. Αυτή η υλοποίηση του αλγόριθμου σε γλώσσα προγραμματισμού Η/Υ, λέγεται **πρόγραμμα**. Ένα πρόγραμμα Η/Υ επενεργεί σε σύνολα δεδομένων που αποθηκεύονται στη μνήμη του Η/Υ. Είναι λοιπόν φανερό πως η επιλογή των δομών με τις οποίες θα οργανωθούν τα δεδομένα στη μνήμη του Η/Υ επηρεάζουν την απόδοση του προγράμματος, άρα την απόδοση γενικότερα, του αλγορίθμου. **Δομές δεδομένων**, ονομάζουμε τις διάφορες διμελείς σχέσεις μεταξύ των στοιχείων ενός συνόλου δεδομένων. Οι σχέσεις αυτές μπορεί να είναι γραμμικές ή μη γραμμικές.

Έστω ένα μη κενό σύνολο δεδομένων και μια διμελής σχέση που διατάσσει τα στοιχεία του έτσι ώστε ένα στοιχείο που ονομάζεται αρχή να έχει ένα επόμενο, ένα στοιχείο που ονομάζεται τέλος ένα προηγούμενο και κάθε άλλο στοιχείο να έχει ένα μόνο προηγούμενο και ένα μόνο επόμενο. Τότε λέμε ότι τα στοιχεία του συνόλου αυτού των δεδομένων, είναι **ολικώς ή γραμμικώς διατεταγμένα** (*totally or linearly ordered*) και η δομή που ορίζεται απ' αυτή τη σχέση ονομάζεται γραμμική δομή δεδομένων (*linear data structure*). Κάθε άλλη δομή δεδομένων που δεν είναι γραμμική, ονομάζεται **μη γραμμική δομή δεδομένων** (*non-linear*). Σημειωτέον ότι η διάταξη αυτή δεν έχει σχέση με τυχόν άλλη διάταξη των τιμών των κόμβων π.χ. λεξικογραφική ή αριθμητική.

Στις γραμμικές δομές δεδομένων, ανήκουν οι πίνακες, οι εγγραφές, τα σύνολα, τα αρχεία και οι γραμμικές λίστες. Από τις γραμμικές λίστες μπορούμε να ξεχωρίσουμε τις **στοίβες** (*stacks*) και τις **ουρές** (*queues*). Στις μη γραμμικές δομές δεδομένων ανήκουν οι **γράφοι** (ή **γραφήματα**) και τα δέντρα.

Στο κεφάλαιο αυτό θα συζητήσουμε τις πιο γνωστές δομές δεδομένων, καθώς και τους αλγορίθμους υλοποίησης των βασικών τους πράξεων. Το υλικό που παρουσιάζεται έχει εν μέρει στηριχθεί στα διδακτικά συγγράμματα [5, 2, 3, 4], όπου και μπορούν να αναζητηθούν περισσότερες λεπτομέρειες για τα θέματα που αναπτύσσονται εδώ.

## 4.2 Σωροί-Ουρά Προτεραιότητας-Heapsort

Πριν προχωρήσουμε στις ουρές προτεραιότητας, θα αναφέρουμε συνοπτικά τι είναι οι λεγόμενες αφηρημένες δομές δεδομένων (ADT: abstract data types). Οι δομές αυτές αποτελούν ουσιαστικά ένα μοντέλο που περιγράφει ένα σύνολο συγκεκριμένων δομών δεδομένων που έχουν παρόμοια συμπεριφορά. Με άλλα λόγια, δεν ορίζουμε με σαφή τρόπο τη δομή μας, αλλά ορίζουμε μόνο τις πράξεις που θέλουμε να εκτελούμε στα στοιχεία μας, και τι ιδιότητες πρέπει να έχουν αυτές οι πράξεις.

Ορίζουμε λοιπόν ένα σύνολο λειτουργιών (μεθόδους) επί των στοιχείων μας που χαρακτηρίζει την ADT, αλλά ο ακριβής τρόπος με τον οποίο θα αναπαραστήσουμε τα δεδομένα μας και θα υλοποιήσουμε τις πράξεις μας μπορεί να διαφέρει.

Η υλοποίηση μιας ADT από μια (συγκεκριμένη) δομή δεδομένων αποτελείται από

- Αναπαράσταση: οργάνωση στιγμιοτύπων και υλοποίηση λειτουργιών με κατάλληλους αλγόριθμους.
- Διατύπωση: ορισμός αναπαράστασης και περιγραφή υλοποίησης λειτουργιών (ψευδοκώδικας).
- Ανάλυση: προσδιορισμός απαιτήσεων σε χώρο αποθήκευσης και χρόνο εκτέλεσης για κάθε (βασική) λειτουργία.

Είμαστε πλέον έτοιμοι να προχωρήσουμε στον ορισμό μιας ADT που ονομάζεται ουρά προτεραιότητας (priority queue).

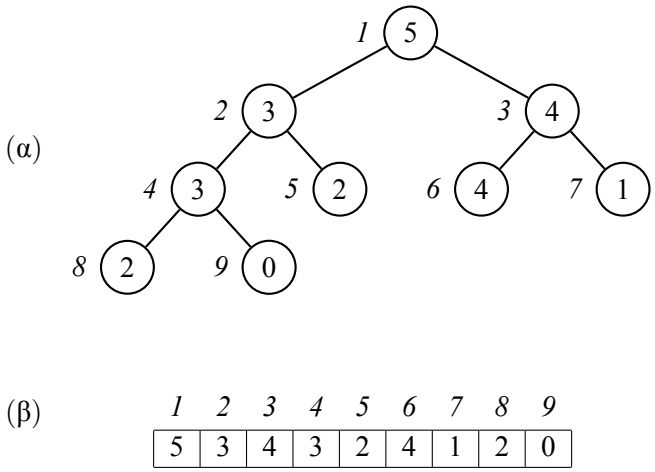
**Ορισμός 4.1.** Έστω ότι μας δίνονται κάποια στοιχεία για τα οποία ισχύει μια ολική σχέση διάταξης ($<$). Μια δομή δεδομένων η οποία μας επιτρέπει να κάνουμε με αποδοτικό τρόπο εισαγωγή ενός καινούργιου στοιχείου και εξαγωγή (δηλαδή εύρεση και διαγραφή) του μεγαλύτερου στοιχείου, λέγεται **ουρά με προτεραιότητα** (*priority queue*).

Με άλλα λόγια, η ουρά προτεραιότητας είναι μια ουρά όπου η σειρά διαγραφής καθορίζεται από μια προτεραιότητα (μεγαλύτερη – μικρότερη). Το κάθε στοιχείο λοιπόν έχει μια προτεραιότητα. Οι λειτουργίες που μας ενδιαφέρουν είναι:

- insert($x$): εισαγωγή $x$
- deleteMax(): διαγραφή και επιστροφή στοιχείου μέγιστης προτεραιότητας
- max(): επιστροφή στοιχείου μέγιστης προτεραιότητας (χωρίς διαγραφή)
- changePriority($k$): αλλαγή προτεραιότητας θέσης $k$
- isEmpty(), size(): βοηθητικές λειτουργίες.

Ένας τρόπος για να παραστήσουμε τις ουρές με προτεραιότητα είναι η υλοποίηση με σωρό.

**Ορισμός 4.2. Σωρός** (*heap tree*) είναι ένα πλήρες δυαδικό δέντρο στο οποίο η τιμή της ρίζας είναι μεγαλύτερη ή ίση (ή μικρότερη ή ίση) από τις τιμές των υπολοίπων κόμβων και αυτό ισχύει αναδρομικά για την ρίζα κάθε υποδέντρου (σχήμα 4.1).




Σχήμα 4.1: (α) Σωρός (Heap tree) και (β) ο πίνακας που αντιστοιχεί σε αυτόν

Έστω ότι έχουμε $n$ ακεραίους αποθηκευμένους με τυχαία σειρά σε ένα μονοδιάστατο πίνακα $a[n]$. Θα περιγράψουμε δύο στρατηγικές με τις οποίες μπορούμε να διατάξουμε τα στοιχεία του πίνακα $a$ σε σωρό.

Η πρώτη στρατηγική είναι η παρακάτω: Θεωρούμε το πρώτο στοιχείο του πίνακα ($a[1]$) σαν σωρό και εισάγουμε ένα προς ένα τα υπόλοιπα στοιχεία του πίνακα διατηρώντας κάθε φορά τη δομή του σωρού. Δηλαδή, όπως φαίνεται στον αλγόριθμο 4.1, κάθε στοιχείο ξεκινά από το τέλος του μέχρι στιγμής δέντρου και βρίσκει τη σωστή θέση του ανεβαίνοντας προς την ρίζα. Η χρονική πολυπλοκότητα του αλγορίθμου είναι $O(n \log n)$. Αυτό συμβαίνει διότι στη χειρότερη περίπτωση (δηλαδή όταν τα στοιχεία βρίσκονται αποθηκευμένα στον πίνακα με αύξουσα σειρά), κάθε στοιχείο πρέπει να διανύσει όλη την απόσταση ως τη ρίζα, συνεπώς αρκεί χρόνος $O(\log n)$.

*Παρατήρηση* 4.3. Η μέση χρονική πολυπλοκότητα είναι $O(n)$ (Άσκηση).

**Αλγόριθμος 4.1** Κατασκευή σωρού (insert)

```
procedure insert (var a:array; n:integer);
var item:integer; k:integer;
begin
  item:=a[n]; k:=n div 2; (* εε  *)
  while ((k>0) and (a[k]<item)) do
    begin a[n]:=a[k]; n:=k; k:=k div 2; end;
  a[n]:=item
end

procedure ConstructHeapInsert (var a:array; n:integer);
var i:integer;
begin
  for i:=2 to n do insert(a,i)
end
```

Η δεύτερη στρατηγική κατασκευής ενός σωρού είναι η εξής: Θεωρούμε τον πίνακα $a$ σαν δέντρο και εξετάζουμε ένα προς ένα όλα τα υποδέντρα του αρχίζοντας από το τέλος. Κάθε υποδέντρο το μετατρέπουμε σε σωρό και το ενώνουμε με την τελική δομή. Στη χειρότερη περίπτωση, ο χρόνος που χρειάζεται για την κατασκευή ενός σωρού με τη βοήθεια του αλγορίθμου 4.2, είναι $O(n)$ [1].

Όμως η χρήση της διαδικασίας *CostructHeapCombine* απαιτεί, όλα τα στοιχεία που θα φτιάξουν το σωρό να είναι διαθέσιμα από την αρχή της διαδικασίας, σε αντίθεση με τη χρήση της διαδικασίας *ConstructHeapInsert* όπου ένα καινούργιο στοιχείο μπορεί να εισαχθεί στο δέντρο οποιαδήποτε χρονική στιγμή.

*Παρατήρηση* 4.4. Η διαδικασία *combine* μπορεί να χρησιμοποιηθεί κι όταν θέλουμε να διαγράψουμε οποιοδήποτε στοιχείο ενός σωρού (όχι μόνο τη ρίζα) χωρίς να χαλάσουμε την ιδιότητα

[1] Αυτό προκύπτει ως εξής: Για κάθε $i$ (του αλγορίθμου) ο αριθμός επαναλήψεων είναι $k - i$, όπου $k = \lceil \log n \rceil$. Οι κόμβοι που υπάρχουν για κάθε $i$ είναι το πολύ $2^{i-1}$. Συνεπώς :

$$\sum_{i=1}^{k} 2^{i-1}(k-i) \leq n \sum_{i} \frac{i}{2^i} = O(n)$$

**Αλγόριθμος 4.2** Κατασκευή σωρού (combine)

```
procedure combine (var a:array; i,n:integer);
(* Μετατρέπει το υποδένδρο με ρίζα a[i] σε σωρό, υπό την προϋπόθεση
 ότι τα υποδένδρα με ρίζες τα παιδιά του a[i] είναι σωροί *)
var left, right, largest_child:integer;

begin
  while 2·i ≤ n do (* ο κόμβος i έχει τουλάχιστον ένα παιδί *)
  begin
    left:=2·i; (* ε *)
    right:=2·i+1; (* ε *)
    largest_child:=left;
    if right ≤ n and a[right]>a[left] then largest_child:=right;
    (* largest_child: η θέση του παιδιού με τη μεγαλύτερη τιμή *)
    if a[i]<a[largest_child] then
    begin
      swap(a[i],a[largest_child]);
      i:=largest_child
    end
    else i:= n div 2 + 1 (*exit while*)
  end
end

procedure CostructHeapCombine (var a:array; n:integer);
var i:integer;
begin
  for i:=n div 2 downto 1 do combine(a,i,n)
end
```

σωρού.

Μια από τις εφαρμογές του σωρού είναι η ταξινόμηση (*sorting*). Στην ταξινόμηση η απλή στρατηγική επιβάλλει συνεχώς να διαλέγουμε από τα στοιχεία που απομένουν, το μεγαλύτερο (ή το μικρότερο). Ένας αλγόριθμος που θα χρησιμοποιούσε αυτή τη στρατηγική όπως είναι, χωρίς καμία βελτίωση της αρχικής σκέψης, θα απαιτούσε στη χειρότερη περίπτωση χρόνο $O(n^2)$ ($n-1$ συγκρίσεις για κάθε στοιχείο). Η χρήση σωρού επιτρέπει την εύρεση του μεγαλύτερου στοιχείου και τη διαγραφή του από τα υπόλοιπα σε χρόνο τάξης $O(\log n)$. Έτσι επιτυγχάνεται για όλη τη διαδικασία της ταξινόμησης χρόνος στη χειρότερη περίπτωση της τάξης $O(n \log n)$.

Η μέθοδος που κάνει ταξινόμηση είναι η παρακάτω (αλγόριθμος 4.3): Κατασκευάζουμε από τον δοσμένο πίνακα $a[1..n]$, ένα σωρό με κάποια από τις μεθόδους που ήδη περιγράφηκαν. Έπειτα ανταλλάσσουμε (*swap*) το πρώτο στοιχείο της δομής ($a[1]$) με το τελευταίο ($a[n]$). Έτσι το τελευταίο στοιχείο του πίνακα τώρα είναι το μεγαλύτερο. Στη συνέχεια κάνουμε σωρό τον πίνακα

$a[1..n-1]$, παίρνουμε πάλι το πρώτο στοιχείο και το βάζουμε στη θέση $a[n-1]$, κ.ο.κ. Τελικά, και μετά από χρόνο $O(n \log n)$, ο πίνακας $a$ είναι ταξινομημένος σε αύξουσα σειρά (ascending order).

**Αλγόριθμος 4.3** Ταξινόμηση HeapSort

```
procedure HeapSort (var a:array; n:integer);
var i:integer;
begin
  ConstructHeap(a,n);  (* Αλγόριθμος 1 ή Αλγόριθμος 2 *)
  for i:=n downto 2 do
    begin swap(a[1],a[i]); combine(a,1,i−1) end
end
```

Ένα εύλογο ερώτημα είναι το εξής: Υπάρχει αλγόριθμος ο οποίος να ταξινομεί με **συγκρίσεις** στοιχεία σε χρόνο μικρότερο από αυτόν της τάξης $O(n \log n)$ (μιλώντας πάντα για τη χειρότερη περίπτωση); Η απάντηση είναι πως δεν είναι δυνατόν να υπάρξει τέτοιος αλγόριθμος.

*Παρατήρηση* 4.5. Πολυπλοκότητα δέντρου απόφασης λέγεται το ύψος του, αφού το ύψος καθορίζει τον αριθμό συγκρίσεων που θα χρειαστεί στη χειρότερη περίπτωση ένας αλγόριθμος που υλοποιεί τις συγκρίσεις του δέντρου.

**Θεώρημα 4.6.** *Κάθε δέντρο απόφασης για ταξινόμηση $n$ στοιχείων, έχει πολυπλοκότητα $\Omega(n \log n)$.*

*Απόδειξη.* Κάθε αλγόριθμος που ταξινομεί με συγκρίσεις, μπορεί να παρασταθεί με τη βοήθεια ενός **δέντρου απόφασης** (*decision tree*), δηλαδή ενός δυαδικού δέντρου του οποίου οι εσωτερικές κορυφές αντιπροσωπεύουν μία σύγκριση (μία απόφαση). Το δέντρο που θα χρησιμοποιηθεί για την ταξινόμηση $n$ στοιχείων θα έχει οπωσδήποτε $n!$ φύλλα (όλες οι δυνατές μεταθέσεις των $n$ στοιχείων). Το μήκος του δρόμου απ' τη ρίζα στο φύλλο ενός δέντρου απόφασης μας δίνει τον αριθμό των συγκρίσεων που χρειάστηκαν για την ταξινόμηση που παριστάνει το φύλλο. Το μήκος του μεγαλύτερου απ' τα μονοπάτια, δηλαδή το ύψος του δέντρου μας δίνει το αριθμό των συγκρίσεων στη χειρότερη περίπτωση. Μπορούμε να προσδιορίσουμε το ελάχιστο ύψος $h$ ενός δέντρου απόφασης $n$ στοιχείων το οποίο έχει $n!$ φύλλα ως εξής. Ακόμη και αν το δέντρο είναι εντελώς πλήρες δυαδικό θα έχει το πολύ $2^h$ φύλλα. Επομένως θα πρέπει να ισχύει:

$$2^h \geq n! \Leftrightarrow h \geq \log(n!) \\ n! \geq (\tfrac{n}{2})^{\frac{n}{2}} \Big\} \Rightarrow h \geq \frac{n}{2} \log(\frac{n}{2})$$

□

Σαν συνέπεια έχουμε το ακόλουθο:

**Πόρισμα 4.7.** *Η ταξινόμηση $n$ αριθμών με συγκρίσεις έχει χρονική πολυπλοκότητα $\Theta(n \log n)$.*

*Απόδειξη.* Από το πιο πάνω θεώρημα είδαμε ότι χρειαζόμαστε για την ταξινόμηση $n$ στοιχείων με συγκρίσεις, χρόνο $\Omega(n \log n)$. Όπως είδαμε ο αλγόριθμος 4.3 έχει πολυπλοκότητα $O(n \log n)$.

Συνεπώς το πρόβλημα της ταξινόμησης με συγκρίσεις λύνεται σε χρόνο $\Theta(n \log n)$, άρα ο αλγόριθμος 4.3 είναι *βέλτιστος*. □

## 4.3 Σύνολα - Συστήματα Εισαγωγής και Ανάκτησης Πληροφοριών - Πράξεις σε Σύνολα

### 4.3.1 Γενικά

Στην σχεδίαση αλγορίθμων τα σύνολα είναι η βάση πολλών σπουδαίων και χρήσιμων **γενικευμένων δομών δεδομένων** (*abstract data types*). Έχουν αναπτυχθεί πολλές τεχνικές υλοποίησης τέτοιων γενικευμένων δομών δεδομένων που βασίζονται σε σύνολα.

Η δομή του συνόλου είναι η βάση πολλών προβλημάτων στα οποία μας ενδιαφέρει η γρήγορη ανάκτηση πληροφοριών (*information retrieval*). Σε αυτά τα προβλήματα, συνήθως έχουμε ένα σύμπαν, καθολικό σύνολο (*universe*) από το οποίο μπορούν να πάρουν στοιχεία όλα τα σύνολα (*sets*) που χρησιμοποιούνται. Οι περιπτώσεις που συναντάμε είναι:

- | sets | $\sim$ | universe |, δηλαδή οι πληθικοί αριθμοί των συνόλων που χρησιμοποιούνται είναι της τάξης του πληθικού αριθμού (cardinality) του καθολικού συνόλου.
- | sets | $<<$ |universe|, # operations $\sim$ | universe |, δηλαδή οι πληθικοί αριθμοί των συνόλων που χρησιμοποιούνται είναι πολύ μικρότεροι από τον πληθικό αριθμό του καθολικού συνόλου και επιπλέον ο αριθμός των πράξεων ανάμεσα στα σύνολα είναι πολύ μεγάλος.
- | sets | $<<$ | universe |, # operations $<<$ |universe|, όμοια με την προηγούμενη περίπτωση, με τη διαφορά ότι ο αριθμός των πράξεων με τα σύνολα είναι μικρός.

Έστω ότι έχουμε ένα σύνολο αναφοράς U με $n$ στοιχεία από το οποίο μπορούμε να κατασκευάσουμε άλλα σύνολα S, τα οποία είναι υποσύνολα του U.

Ένας τρόπος να παραστήσουμε τα σύνολα αυτά S, είναι με τη βοήθεια ενός διανύσματος μήκους $n$, $S[1..n]$ τέτοιου ώστε $S[i] = 1$ αν το i-οστό στοιχείο του U ανήκει στο S και $S[i] = 0$ σε άλλη περίπτωση.

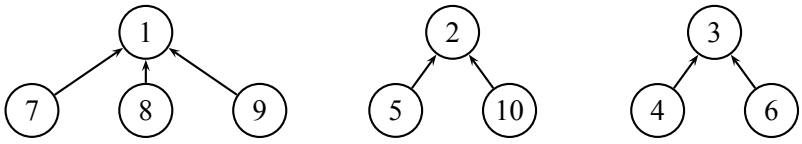
Άλλος τρόπος για να παραστήσουμε σύνολα S, ξένα μεταξύ τους, είναι με τη βοήθεια των δέντρων.

*Παράδειγμα* 4.8. Έστω

$$U = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\},$$
$$S_1 = \{1, 7, 8, 9\}, S_2 = \{2, 5, 10\}, S_3 = \{3, 4, 6\}.$$

Τα $S_1, S_2, S_3$ μπορούμε να τα παραστήσουμε όπως φαίνεται στο σχήμα 4.2. Δηλαδή κάθε κορυφή συνδέεται με τον γονέα της. Π.χ.

| | | |
|---|---|---|
| `Parent`[7] | $= 1,$ | στοιχείο του ίδιου συνόλου |
| `Parent`[9] | $= 1,$ | στοιχείο του ίδιου συνόλου |
| `Parent`[10] | $= 2,$ | στοιχείο του ίδιου συνόλου |
| `Parent`[1] | $= 0,$ | το 1 είναι ρίζα του δέντρου και χρησιμεύει σαν όνομα του συνόλου |




$S_1 = \{1, 7, 8, 9\}$ $S_2 = \{2, 5, 10\}$ $S_3 = \{3, 4, 6\}$

Σχήμα 4.2: Παράσταση συνόλων ξένων μεταξύ τους με χρήση δένδρων

Έστω στοιχείο $a \in U$ (universe) και σύνολα $S, S_1, S_2 \subseteq U$. Οι πιο χαρακτηριστικές πράξεις μεταξύ συνόλων είναι οι παρακάτω:

- Member($a, S$): Ελέγχει αν το στοιχείο a ανήκει στο σύνολο S και αν αυτό ισχύει επιστρέφει true αλλιώς false.
- Search($a, S$): Επιστρέφει ένα δείκτη στο στοιχείο $a$ αν $a \in S$ αλλιώς επιστρέφει nil.
- Insert($a, S$): Εισάγει το στοιχείο $a$ στο σύνολο $S$ και επιστρέφει το σύνολο $S \bigcup \{a\}$.
- Delete($a, S$): Διαγράφει το στοιχείο $a$ από το σύνολο $S$ και επιστρέφει το σύνολο $S \setminus \{a\}$.
- Union($S_1, S_2$): Επιστρέφει το σύνολο $S_1 \bigcup S_2$. Υποθέτουμε ότι τα σύνολα $S_1, S_2$ είναι ξένα (disjoint) για να αποφύγουμε τον έλεγχο για διπλά στοιχεία.
- Find($a$): Επιστρέφει το όνομα του συνόλου στο οποίο ανήκει το $a$. Υποθέτουμε ότι έχουμε διαμέριση σε ξένα σύνολα, άρα το στοιχείο $a$, θα ανήκει σε ένα ακριβώς σύνολο.
- Split($a, S$): Χωρίζει το σύνολο $S$ σε δύο σύνολα $S_1, S_2$ τέτοια ώστε:

  $$S_1 = \{b \mid b \leq a \bigwedge b \in S\} \text{ και } S_2 = \{b \mid b > a \bigwedge b \in S\}$$

  Εδώ υποθέτουμε ότι το σύνολο $S$ είναι ένα σύνολο του οποίου τα στοιχεία έχουν μια γραμμική διάταξη ($\leq$).
- Max($S$), Min($S$): Επιστρέφει το μεγαλύτερο ή το μικρότερο των στοιχείων του $S$. Υποθέτουμε πάλι γραμμική διάταξη των στοιχείων του $S$.
- Successor($a, S$): Επιστρέφει το μικρότερο από τα στοιχεία του $S$ που είναι μεγαλύτερο του $a$ (επόμενο στοιχείο). Predecessor($a, S$): Ομοίως επιστρέφει το μεγαλύτερο από τα στοιχεία του $S$ που είναι μικρότερο του $a$ (προηγούμενο στοιχείο).

Ανάλογα λοιπόν με το ποιες λειτουργίες από τις παραπάνω χρησιμοποιούμε συχνά, φτιάχνουμε και τις κατάλληλες δομές, υλοποιώντας τα σύνολα με διάφορες τεχνικές.

### 4.3.2 Δομή λεξικού (Dictionary)

**Ορισμός 4.9.** Ας υποθέσουμε ότι έχουμε ένα σύνολο S και θέλουμε να εκτελούνται γρήγορα οι λειτουργίες της εισαγωγής καινούργιου στοιχείου, διαγραφής παλαιού στοιχείου και ελέγχου για την ύπαρξη κάποιου στοιχείου στο S. Θέλουμε δηλαδή να εκτελείται αποδοτικά μία ακολουθία από διαδικασίες *Insert, Delete* και *Member*. Αυτή τη γενικευμένη αφηρημένη δομή δεδομένων (ADT) την ονομάζουμε **λεξικό** (*Dictionary*).

Η υλοποίηση ενός λεξικού μπορεί να γίνει π.χ. με μια συνάρτηση

$$h\colon universe \to \{0, \ldots, m-1\},$$

που ονομάζεται **συνάρτηση κατακερματισμού** (*hashing function*). Φροντίζουμε ώστε η συνάρτηση $h$ που διαλέγουμε να υπολογίζεται γρήγορα για οποιοδήποτε στοιχείο του καθολικού συνόλου, σε χρόνο δηλαδή $O(1)$. Θεωρούμε ένα πίνακα $A$ (hash table). Κάθε στοιχείο $A[i]$ του πίνακα δείχνει σε μια λίστα η οποία περιέχει εκείνα τα στοιχεία a του καθολικού συνόλου για τα οποία ισχύει $h(a) = i$. Συνεπώς για να κάνουμε *Insert, Delete* και *Member* ένα στοιχείο $a$, αρκεί να ψάξουμε μόνο τη λίστα στην οποία δείχνει το στοιχείο $A[h(a)]$.

Βέβαια στη χειρότερη περίπτωση, είναι πιθανόν μετά από $n$ εισαγωγές στοιχείων, να έχουμε μια λίστα μήκους σχεδόν $n$ (δηλαδή σχεδόν όλα τα στοιχεία να έχουν πάει στην ίδια λίστα). Σε αυτή την περίπτωση, αν χρειαστεί να εκτελέσουμε $n$ φορές τις διαδικασίες *Delete* ή *Member*, ο χρόνος που απαιτείται είναι $O(n^2)$. Αν όμως φροντίσουμε ώστε η εκλογή της συνάρτησης $h$ να εξασφαλίζει μια όσο το δυνατό ομοιόμορφη κατανομή των στοιχείων στις λίστες, έτσι ώστε να μην υπάρχει συσσώρευση στοιχείων σε μια λίστα, τότε ο χρόνος αναζήτησης μπορεί να καλυτερεύσει σημαντικά. Για παράδειγμα, αν πρόκειται να εισαχθούν περίπου $n \in O(m)$ στοιχεία, τότε τη στιγμή που εισάγεται το i-οστό στοιχείο, η λίστα στην οποία θα πρέπει να μπει θα έχει αναμενόμενο μήκος $\frac{i-1}{m} < 1$ συνεπώς η κάθε διαδικασία που θα πρέπει να διατρέξει κάποια λίστα θα χρειάζεται περίπου σταθερό χρόνο $O(1)$ και έτσι $n$ διαδικασίες θα χρειάζονται $O(n)$ χρόνο περίπου.

Είναι συνηθισμένο να μην γνωρίζουμε από πριν τον πληθικό αριθμό που μπορεί να έχει το σύνολό μας. Στην περίπτωση αυτή διαλέγουμε μια τιμή $m$ για τον πίνακα (*hash table, bucket table*) και όταν ο αριθμός των στοιχείων γίνει μεγαλύτερος από $m$, δημιουργούμε ένα καινούργιο πίνακα-στήλη μεγέθους $2m$ και με rehashing (δηλαδή ορίζοντας μια νέα συνάρτηση με πεδίο τιμών αυτή τη φορά το $[0, 2m-1]$), βάζουμε τα στοιχεία στον καινούργιο πίνακα, καταστρέφοντας τον παλιό. Όταν τα στοιχεία γίνουν περισσότερα από $2m$, δημιουργούμε ένα άλλο πίνακα μεγέθους $4m$ κ.ο.κ. Είναι σαφές ότι κάθε φορά η κατάλληλη επιλογή της συνάρτησης κατακερματισμού παίζει σπουδαίο ρόλο προκειμένου να διατηρήσουμε τους χρόνους προσπέλασης μικρούς.

*Παράδειγμα* 4.10. Έστω ότι το σύνολό μας αποτελείται από ακεραίους που μπορούν να πάρουν τιμές στο διάστημα $[0, r], r > n$. Τότε αν χρησιμοποιήσουμε τη συνάρτηση $h(a) = a \bmod m$, όπου $m$ το μέγεθος του τρέχοντα πίνακα-στήλη έχουμε τα παρακάτω: Έστω ότι εισάγουμε τους αριθμούς $1, 5, 8, 3, 9, 6$. Αρχικά επιλέγουμε $m = 2$ και έχουμε :

$$\begin{array}{ll} 0: & \\ 1: & 1, 5 \end{array}$$

Σε αυτό το σημείο επιλέγουμε $m = 4$:

$$\begin{array}{ll} 0: & 8 \\ 1: & 1, 5 \\ 2: & \\ 3: & 3 \end{array}$$

Τέλος με $m = 8$ προκύπτει το παρακάτω:

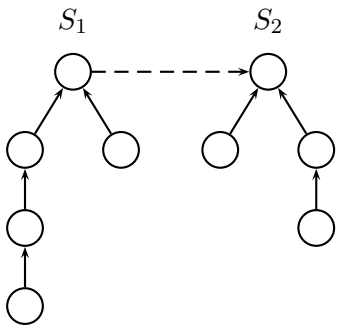
$$\begin{array}{ll} 0: & 8 \\ 1: & 1, 9 \\ 2: & \\ 3: & 3 \\ 4: & \\ 5: & 5 \\ 6: & 6 \\ 7: & \end{array}$$

### 4.3.3 Δομή Union–Find

**Ορισμός 4.11.** Έστω ότι έχουμε διάφορα ξένα μεταξύ τους σύνολα και μας ενδιαφέρει η αποδοτική υλοποίηση μιας **ακολουθίας** από διαδικασίες ένωσης (*Union*) συνόλων και εύρεσης του συνόλου στο οποίο ανήκει κάποιο στοιχείο (*Find*). Μια τέτοια δομή δεδομένων ονομάζεται δομή Union-Find.

Η αναπαράσταση των συνόλων μπορεί να γίνει π.χ. με δέντρα, όπου κάθε κόμβος περιέχει ένα στοιχείο και έχει ένα pointer προς τον γονέα του. Κατά σύμβαση το όνομα του συνόλου είναι το στοιχείο που τυχαίνει να βρίσκεται στη ρίζα.

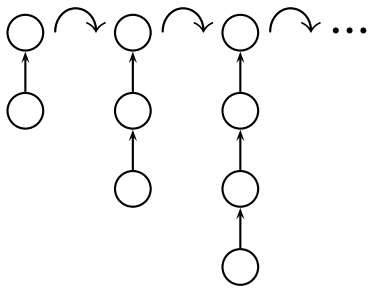
Η array Parent[i], μας δίνει τον γονέα του κόμβου $i$ και θέτουμε Parent[root]:=0. Για να βρούμε την ένωση δύο συνόλων $S_1, S_2$ αρκεί να μεταβάλλουμε την εγγραφή της Parent σε μια απ' τις δύο ρίζες των $S_1, S_2$, έτσι ώστε αντί να δείχνει 0, να δείχνει στη ρίζα του άλλου δέντρου (σχήμα 4.3). Η υλοποίηση της συνάρτησης Union φαίνεται στον αλγόριθμο 4.4.




Σχήμα 4.3: Η ένωση των συνόλων $S_1$ και $S_2$

Η εκτέλεση της συνάρτησης Union γίνεται σε σταθερό χρόνο $O(1)$, ενώ για την εύρεση του συνόλου S στο οποίο ανήκει το στοιχείο i, αρκεί να βρούμε τη ρίζα του δέντρου που αναπαριστά το σύνολο S. Η υλοποίηση της συνάρτησης Find φαίνεται στον αλγόριθμο 4.5.

Στη χειρότερη περίπτωση, ένα εκφυλισμένο (*degenarate*) δέντρο μπορεί να προκύψει από την ένωση πολλών συνόλων όπως στο σχήμα 4.4. Έτσι λοιπόν ο χρόνος που χρειάζεται η Find για να διανύσει ένα μονοπάτι του συνόλου-δέντρου με $n$-στοιχεία-κορυφές είναι στην χειρότερη περίπτωση $O(n)$. Συνεπώς $n$ εκτελέσεις της Find χρειάζονται χρόνο $O(n^2)$.



Σχήμα 4.4: Εκφυλισμένο δένδρο μετά από την ένωση πολλών συνόλων

Μπορούμε να βελτιώσουμε αυτό το χρόνο αν λάβουμε υπόψη μας τον αριθμό των στοιχείων που έχει κάθε σύνολο και συνδέουμε κάθε φορά το σύνολο-δέντρο με τα λιγότερα στοιχεία-κόμβους σε εκείνο με τα περισσότερα, αλλάζοντας τη συνάρτηση Union (Balancing).

Θέτουμε την πληροφορία του πληθικού αριθμού κάθε συνόλου σαν τιμή του γονέα της ρίζας του ($Parent[root] := -\#elements$). Ο αρνητικός αριθμός ξεχωρίζει την τιμή του γονέα της ρίζας από τα υπόλοιπα στοιχεία του συνόλου. Η νέα υλοποίηση της συνάρτησης Union φαίνεται στον αλγόριθμο 4.6.

**Λήμμα 4.12.** *Ένα δέντρο που κατασκευάστηκε με τη βοήθεια του αλγόριθμου 4.6 έχει ύψος μικρότερο από* $\lfloor \log n \rfloor + 1$.

*Απόδειξη.* Επαγωγική βάση: $n = 1$. Προφανές.

Επαγωγικό βήμα: Έστω αληθές για όλα τα δέντρα με αριθμό κόμβων $\leq n - 1$. Έστω ότι η τελευταία εφαρμογή της διαδικασίας ήταν $union(k, j)$ και ότι το δέντρο $j$ είχε $m$ κόμβους. Χωρίς βλάβη της γενικότητας $1 \leq m \leq \frac{n}{2}$.

Περίπτωση 1: Νέο ύψος = ύψος του δέντρου $k$:

$$\text{ύψος} \leq \lfloor \log(n - m) \rfloor + 1 \leq \lfloor \log n \rfloor + 1.$$

**Αλγόριθμος 4.4** Διαδικασία ένωσης (Union)

```
function Union (i, j: integer (*set*)): integer (*set*);
begin
  Parent[i]:=j; return j
end
```

**Αλγόριθμος 4.5** Διαδικασία εύρεσης (Find)

```
function Find (i: integer (*element*)): integer (*set*);
begin
  while Parent[i]>0 do i := Parent[i];
  return i
end
```

**Αλγόριθμος 4.6** Βελτιωμένη διαδικασία ένωσης (Balancing)

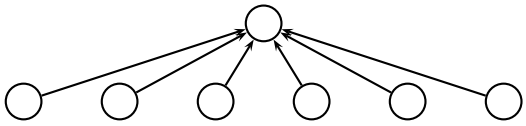
```
function Union (i, j: integer (*set*)): integer (*set*);
var x: integer; (*αρνητικός αριθμός στοιχείων του νέου συνόλου*)
begin
  x:=Parent[i]+Parent[j];
  if |Parent[i]| < |Parent[j]| then
    begin Parent[i]:=j; Parent[j]:=x; return j end
  else begin Parent[j]:=i; Parent[i]:=x; return i end
end
```

Περίπτωση 2: Νέο ύψος = ύψος του δέντρου $j+1$:

$$\text{ύψος} \leq \lfloor \log m \rfloor + 2 \leq \lfloor \log \frac{n}{2} \rfloor + 2 \leq \lfloor \log n \rfloor + 1.$$

□

Σαν αποτέλεσμα έχουμε ότι ο χρόνος που χρειάζεται μια εκτέλεση της συνάρτησης *Find* είναι $O(\log n)$, δηλαδή $n$ εκτελέσεις χρειάζονται $O(n \log n)$ χρόνο. Μπορούμε να πετύχουμε μια ακόμη καλυτέρευση του χρόνου, αλλάζοντας αυτή τη φορά τη συνάρτηση *Find* με προσθήκη της διαδικασίας *Path Compression* : ένα προς ένα τα στοιχεία-κορυφές που συναντά ο αλγόριθμος στο δρόμο για τη ρίζα, τα «ξεκρεμά» από τη θέση τους και τα «κρεμάει» από τη ρίζα, με αποτέλεσμα το σύνολο-δέντρο να τείνει προοδευτικά να μετασχηματιστεί όπως στο σχήμα 4.5. Η νέα υλοποίηση της συνάρτησης *Find* φαίνεται στον αλγόριθμο 4.7.



Σχήμα 4.5: Path compression

Είναι προφανές ότι αυτός ο τρόπος συμφέρει όταν πρόκειται να εκτελεστούν πολύ περισσότερα του ενός *Find*. Αποδεικνύεται μάλιστα ότι $n$ εκτελέσεις της συνάρτησης *Find* χρειάζονται χρόνο $O(n\alpha(n))$, όπου $\alpha(n)$ είναι ψευδοαντίστροφη της συνάρτησης Ackermann(σχεδόν σταθερά). Η ιδέα αυτή είναι παράδειγμα **αποσβεστικής** αποδοτικότητας (*amortization*). Ένα βήμα της *Path*

*Compression* κοστίζει αλλά έτσι εξοικονομείται χρόνος για κατοπινές εφαρμογές της *Find*. Η βελτίωση αυτή οφείλεται στον Tarjan [**?**].

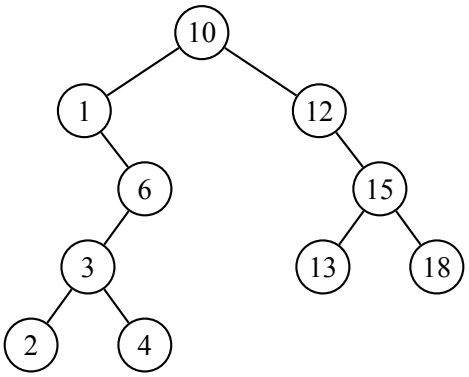
**Αλγόριθμος 4.7** Βελτιωμένη διαδικασία εύρεσης (Path compression)

```
function Find(i: integer (*element*)): integer (*set*);
var j, t: integer (*element*);
begin
  j := i;
  (*Εύρεση της ρίζας*)
  while Parent[j] > 0 do j := Parent[j];
  (*Ξεκρέμασμα των φύλλων και κρέμασμα από τη ρίζα*)
  while (i ≠ j) do
    begin t := Parent[i]; Parent[i] := j; i := t end;
  return j
end
```

### 4.3.4 Δυαδικά δέντρα αναζήτησης (binary search trees)

Ένας άλλος τρόπος αναπαράστασης συνόλων με μια γραμμική διάταξη, είναι με **δυαδικά δέντρα**. Αυτή η δομή είναι χρήσιμη όταν έχουμε μεγάλα σύνολα και είναι ασύμφορη η χρησιμοποίηση των προηγούμενων μεθόδων. Ένα **δυαδικό δέντρο αναζήτησης** (*binary search tree*) μπορεί να υποστηρίξει αποδοτικά πράξεις όπως *Insert, Delete, Member* και *Min* απαιτώντας κατά μέσο όρο $O(\log n)$ χρόνο για κάθε διαδικασία (όπου $n$ ο πληθάριθμος του συνόλου).

Τα στοιχεία του συνόλου διατάσσονται στο δυαδικό δέντρο ως εξής: Όλα τα στοιχεία-κορυφές που βρίσκονται στο αριστερό υποδέντρο μιας κορυφής $x$, είναι μικρότερα από το στοιχείο που βρίσκεται στην κορυφή $x$ ενώ τα στοιχεία του δεξιού υποδέντρου είναι μεγαλύτερα. Αυτή η συνθήκη (*binary search tree property*) ισχύει για όλες τις κορυφές του δέντρου (σχήμα 4.6).




Σχήμα 4.6: Δυαδικό δένδρο αναζήτησης

Ο έλεγχος για το αν ένα στοιχείο $x$ ανήκει στο σύνολο γίνεται όπως παρακάτω:

Συγκρίνουμε το στοιχείο $x$ με τη ρίζα του δέντρου

- Αν τα στοιχεία είναι ίσα η διαδικασία τελειώνει.
- Αν το $x$ είναι μικρότερο προχωράμε στο αριστερό υποδέντρο του $x$ και επαναλαμβάνουμε τη διαδικασία.
- Αν το $x$ είναι μεγαλύτερο προχωράμε στο δεξιό υποδέντρο του $x$ και επαναλαμβάνουμε τη διαδικασία.

Η συνάρτηση Member φαίνεται στο αλγόριθμο 4.8. Κάθε κόμβος περιέχει κάποιο στοιχείο του συνόλου και δύο δείκτες, ένα στο αριστερό παιδί και ένα στο δεξιό. Οι διαδικασίες Insert, RetrieveMin και Delete είναι εντελώς ανάλογες και φαίνονται στους αλγόριθμους 4.9, 4.10 και 4.11.

**Αλγόριθμος 4.8** Συνάρτηση Member σε δυαδικό δένδρο αναζήτησης

```
function Member(x:elementtype; A:^node):boolean;
begin
  if A=Nil then return(false)
  else if x=A^.element then return true
  else if x<A^.element then return Member(x,A^.leftchild)
  else return Member(x,A^.rightchild)
end
```

**Αλγόριθμος 4.9** Συνάρτηση Insert σε δυαδικό δένδρο αναζήτησης

```
procedure Insert(x:elementtype; var A:^node);
begin
  if A=Nil then
  begin
    new(A); A^.element:=x;
    A^.leftchild:=Nil; A^.rightchild:=Nil
  end
  else if x<A^.element then Insert(x,A^.leftchild)
  else if x>A^.element then Insert(x,A^.rightchild)
  (* Αν x=A^.element, τότε ήδη x είναι στο A *)
end
```

### 4.3.5 Ισοζυγισμένα Δέντρα (Balanced Trees)

Όπως είδαμε στην προηγούμενη παράγραφο, χρησιμοποιώντας ένα δυαδικό δέντρο αναζήτησης σαν αναπαράσταση ενός συνόλου, έχουμε έναν αναμενόμενο χρόνο (*average-case complexity*)

**Αλγόριθμος 4.10** Συνάρτηση RetrieveMin σε δυαδικό δένδρο αναζήτησης

```
function RetrieveMin(var A:^node):elementttype; (* Επιστρέφει,
διαγράφοντας από το σύνολο A το μικρότερο στοιχείο του *)
begin
  if A^.leftchild=Nil then
    (∗ Το A δείχνει στο μικρότερο στοιχείο ∗)
    begin return A^.element; A:=A^.rightchild end
  else return RetrieveMin(A^.leftchild)
end
```

**Αλγόριθμος 4.11** Συνάρτηση Delete σε δυαδικό δένδρο αναζήτησης

```
procedure Delete(x:elementtype; var A:^node);
begin
  if A<>Nil then
    if x<A^.element then Delete(x, A^.leftchild)
    else if x>A^.element then Delete(x,A^.rightchild)
    else (* x=A^.element *)
          if A^.leftchild=Nil then A:=A^.rightchild
          else if A^.rightchild=Nil then A:=A^.leftchild
          else (∗ ο κόμβος με το x έχει 2 παιδιά ∗)
              A^.element := RetrieveMin(A^.rightchild)
end
```

$O(\log n)$ για κάθε προσπέλαση. Στη χειρότερη περίπτωση όμως, αν εισάγουμε συνεχώς στοιχεία στο σύνολό μας, μπορεί να καταλήξουμε σε εκφυλισμένο δέντρο του οποίου το ύψος προσεγγίζει τον αριθμό των κορυφών του. Έτσι μια προσπέλαση σε αυτό το δέντρο χρειάζεται χρόνο $O(n)$.

Έχουν αναπτυχθεί διάφορες τεχνικές που φροντίζουν να διατηρούν το δέντρο **ισοζυγισμένο** (*balanced*) κατά τη διάρκεια διαφόρων διαδικασιών που το μεταβάλλουν (Insert, Delete κ.α.). Δύο από αυτές τις τεχνικές είναι τα 2-3 trees και τα AVL trees[2].

**Ορισμός 4.13.** Το 2-3 tree είναι ένα δέντρο στο οποίο κάθε κορυφή που δεν είναι φύλλο έχει δύο ή τρία παιδιά και κάθε μονοπάτι από τη ρίζα σε ένα φύλλο έχει το ίδιο μήκος.

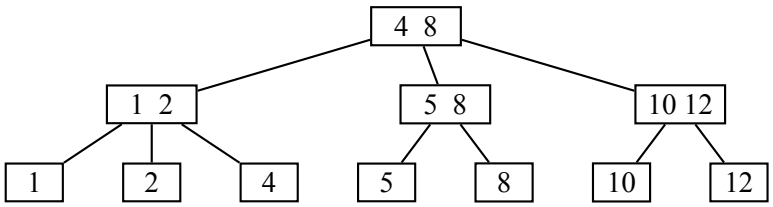
Τα στοιχεία εισάγονται σε ένα 2-3 tree συνήθως ως εξής: Κάθε εσωτερική κορυφή έχει δύο θέσεις.

- Στην αριστερή θέση εισάγεται το μεγαλύτερο από τα στοιχεία του υποδέντρου που έχει ρίζα το αριστερό παιδί της κορυφής αυτής, στοιχείο που (πρέπει να) είναι και μικρότερο ή ίσο από τα στοιχεία των υπόλοιπων παιδιών.

[2] Το όνομα AVL προέρχεται από τα αρχικά των δημιουργών του Adelson-Velskii-Landis (1962)

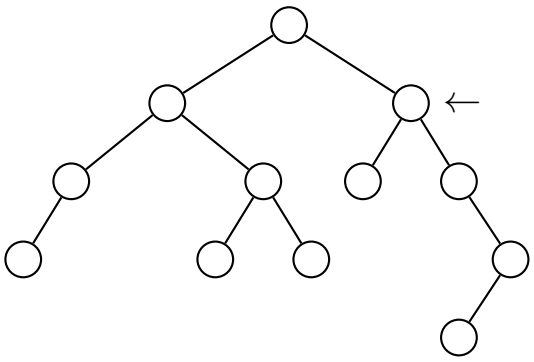
- Στη δεξιά θέση εισάγεται το μεγαλύτερο από τα στοιχεία του μεσαίου υποδέντρου, που (πρέπει να) είναι και μικρότερο ή ίσο από τα στοιχεία δεξιού υποδέντρου.

Τα φύλλα του δέντρου περιέχουν τα στοιχεία του συνόλου (σχήμα 4.7).




Σχήμα 4.7: 2-3 δένδρο

**Ορισμός 4.14.** Το **AVL tree** είναι ένα δυαδικό δέντρο αναζήτησης με την εξής ιδιότητα, το ύψος του αριστερού και του δεξιού υποδέντρου κάθε κορυφής, διαφέρουν το πολύ κατά 1.



Σχήμα 4.8: Το δένδρο δεν είναι AVL εξαιτίας της σημειωμένης κορυφής

*Παράδειγμα* 4.15. Το δέντρο στο σχήμα 4.8 δεν είναι AVL tree διότι η σημειωμένη κορυφή έχει ένα αριστερό υποδέντρο ύψους 0 και ένα δεξιό υποδέντρο ύψους 2. Παρ' όλα αυτά η ιδιότητα του AVL tree ισχύει σε κάθε άλλη κορυφή.

Οι διαδικασίες αναπτύσσονται έτσι ώστε να διατηρούν τη μία ή την άλλη δομή, με συνέπεια η προσπέλαση στα δέντρα να γίνεται στη χειρότερη περίπτωση (*worst-case complexity*) σε χρόνο $O(\log n)$.

## 4.4 Διαδραστικό υλικό - Σύνδεσμοι

- Στην σελίδα http://visualgo.net/ θα βρείτε οπτικοποιήσεις δομών δεδομένων με γραφικά.
- Στην σελίδα http://www.algolist.net/ υπάρχουν υλοποιήσεις δομών δεδομένων σε C++ και Java.

- Η σελίδα http://www.graphtheory.com/ είναι κεντρικός κόμβος για πληροφορίες, νέα και βιβλιογραφία στην Θεωρία Γραφημάτων.
- Εδώ μπορείτε να βρείτε υλοποιήσεις σε Python αναπαραστάσεων και βασικών λειτουργιών γραφημάτων.

## 4.5 Ασκήσεις

1. Για το παρακάτω σύνολο αριθμών, εκτελέστε τα στάδια κατασκευής σωρού για τους δύο αλγορίθμους heap combine και heap insert:

   $$\{x, y, z, w, 14, 5, 6, 15, 8, 7, 6, 1, 3\}$$

   όπου $x$, $y$, $z$, $w$ είναι τα τέσσερα τελευταία ψηφία του αριθμού ταυτότητάς σας. Δείξτε τα στάδια εκτέλεσης με σχήματα.

2. Πόσα βήματα χρειάζεται ο αλγόριθμος heapsort για να ταξινομήσει ένα διάνυσμα με:

   - ήδη ταξινομημένα στοιχεία κατά αύξουσα σειρά;
   - ήδη ταξινομημένα στοιχεία κατα φθίνουσα σειρά;

3. Δείξτε ότι στη χειρότερη περίπτωση ο αλγόριθμος heapsort έχει πολυπλοκότητα $\Omega(n \log n)$.

4. Σε έναν κατευθυνόμενο γράφο, ένας κόμβος με indegree μηδέν λέγεται πηγή.

   (α) Αποδείξτε ότι σε κάθε ακυκλικό κατευθυνόμενο γράφο υπάρχει τουλάχιστον μία πηγή.

   (β) Δείξτε ότι ένας κατευθυνόμενος γράφος με $n$ κόμβους είναι ακυκλικός αν και μόνο αν μπορούμε να τοποθετήσουμε ετικέτες $1, 2, \ldots, n$ στους κόμβους ώστε όλες οι ακμές να κατευθύνονται από κόμβο με μικρότερη ετικέτα σε κόμβο με μεγαλύτερη ετικέτα.

   (γ) Περιγράψτε πολυωνυμικό αλγόριθμο που να αποφαίνεται αν ένας κατευθυνόμενος γράφος είναι ακυκλικός.

5. Σχεδιάστε συνάρτηση που με είσοδο ένα δυαδικό δέντρο αναζήτησης και αριθμό $x$, επιστρέφει δείκτη στον μεγαλύτερο αριθμό στο δέντρο που είναι μικρότερος του $x$ ή $nil$ αν δεν υπάρχει τέτοιος αριθμός.

6. Σχεδιάστε συνάρτηση που να συνδυάζει σωστά δύο δυαδικά δέντρα αναζήτησης.

7. Σχεδιάστε τις συναρτήσεις Insert και RetrieveMin για 2-3 tree.

8. Σχεδιάστε τη συνάρτηση Delete για AVL tree.

# Βιβλιογραφία


[1] Thomas Cormen, Charles Leiserson, Ronald Rivest and Cliff Stein, “Introduction to Algorithms”, 3rd edition, MIT Press, 2009.

[2] C.L. Liu. Στοιχεία Διακριτών Μαθηματικών (απόδοση στα Ελληνικά: Κ. Μπους και Δ. Γραμμένος). Πανεπιστημιακές Εκδόσεις Κρήτης, 2003.

[3] Γεώργιος Γεωργακόπουλος, Δομές Δεδομένων, Πανεπιστημιακές Εκδόσεις Κρήτης, 2002, 960-524-125-0

[4] Παναγιώτης Μποζάνης, Δομές δεδομένων, 960-418-010-X, Τζιόλας, 2003

# Κεφάλαιο 5

# Αλγόριθμοι

## 5.1 Αλγόριθμοι και Πολυπλοκότητα

Η ονομασία Αλγόριθμος προέρχεται από το όνομα του ’Αραβα Μαθηματικού Al-Khowârizmi (με καταγωγή από το Ουζμπεκιστάν, που έζησε στη Βαγδάτη τον 9ο αιώνα μ.χ). Ήταν ο πρώτος που διατύπωσε τους κανόνες για τις 4 βασικές αριθμητικές πράξεις (από δικό του βιβλίο προέρχεται και η Άλγεβρα).

Υπενθυμίζουμε ότι σκοπός ενός αλγορίθμου είναι η επίλυση ενός προβλήματος. Πριν ξεκινήσουμε ας θέσουμε ορισμένες ερωτήσεις για τα προβλήματα: Ποιος ορίζει πρόβλημα; Τι είναι πρόβλημα; Πώς μοντελοποιούμε ένα πρόβλημα; Τι είναι λύση; Πώς αναπαριστούμε μια λύση (γλώσσα); Είναι πράγματι μια προτεινόμενη λύση, λύση του προβλήματος (ορθότητα); Ποια είναι η αποδοτικότητα της μεθόδου εξεύρεσης λύσης (πολυπλοκότητα); Πότε είναι μια λύση καλή; Καλύτερη; Βέλτιστη; Ποία είναι τα όρια εφαρμοσιμότητας μιας οποιασδήποτε λύσης; Είναι ένα πρόβλημα δύσκολο; Πώς χειριζόμαστε τα δύσκολα προβλήματα; Και μια απάντηση: Λύνουμε προβλήματα με συνδυασμό ευφυΐας, διαίσθησης, τύχης, πείρας και σκληρής δουλειάς.

Σε έναν αλγόριθμο μας ενδιαφέρουν: ορθότητα, πολυπλοκότητα, αν εφαρμόζεται γενικά (δηλαδή για όλα τα πιθανά στιγμιότυπα εισόδου), αν είναι βέλτιστος, ακριβής ή προσεγγιστικός, πιθανοτικός κ.τ.λ.

### 5.1.1 Τι είναι αλγόριθμος

Η έννοια αλγόριθμος είναι πρωταρχική έννοια της θεωρίας αυτής. Γι’ αυτό δεν ορίζεται. Εδώ δίνουμε μία άτυπη εξήγηση.

**Αλγόριθμος** είναι ένα πεπερασμένο σύνολο κανόνων, οι οποίοι περιγράφουν μία μέθοδο (που αποτελείται από μία σειρά υπολογιστικών διεργασιών) για να λυθεί ένα συγκεκριμένο πρόβλημα. Τα αντικείμενα πάνω στα οποία επενεργούν αυτές οι διεργασίες λέγονται **δεδομένα** (*data*).

Ο αλγόριθμος χαρακτηρίζεται από τα παρακάτω πέντε στοιχεία:

- Κάθε εκτέλεση είναι *πεπερασμένη*, δηλαδή τελειώνει ύστερα από έναν πεπερασμένο αριθμό

διεργασιών ή βημάτων (*finiteness*).

- Κάθε κανόνας του ορίζεται επακριβώς και η αντίστοιχη διεργασία είναι συγκεκριμένη (*definiteness*).
- Έχει μηδέν ή περισσότερα μεγέθη *εισόδου* που δίδονται εξαρχής, πριν αρχίσει να εκτελείται ο αλγόριθμος (*input*).
- Δίδει τουλάχιστον ένα μέγεθος σαν αποτέλεσμα (*έξοδο-output*) που εξαρτάται κατά κάποιο τρόπο από τις αρχικές εισόδους.
- Είναι *μηχανιστικά* αποτελεσματικός, δηλαδή όλες οι διαδικασίες που περιλαμβάνει μπορούν να πραγματοποιηθούν με ακρίβεια και σε πεπερασμένο χρόνο "με μολύβι και χαρτί" (*effectiveness*).

### 5.1.2 Πολυπλοκότητα αλγορίθμων και προβλημάτων

Στην πράξη, το ενδιαφέρον δεν σταματά στο να βρεθεί ένας αλγόριθμος που επιλύει ένα πρόβλημα, αλλά προχωρά στη μελέτη των μετρήσιμων ιδιοτήτων που χαρακτηρίζουν την **αποδοτικότητα** μιας υπολογιστικής μεθόδου. Αυτά τα μεγέθη (αγαθά-*resources*) είναι π.χ. ο χρόνος υπολογισμού, ο χώρος σε μνήμη υπολογιστή, ο αριθμός προκαταρκτικών διαδικασιών που προαπαιτούνται και είναι αυτά που ορίζουν την **πολυπλοκότητα** (*complexity*) του αλγορίθμου. Ονομάζουμε **πολυπλοκότητα ενός προβλήματος** την πολυπλοκότητα ενός **βέλτιστου** (*optimal* ) αλγορίθμου που λύνει το πρόβλημα.

Η συμπεριφορά του αλγορίθμου μελετάται κυρίως σε δύο περιπτώσεις. Στην **χειρότερη** (*worst case*) και στην **μέση** (*average case*), μιας δεδομένης κατανομής πιθανών **στιγμιοτύπων** (*instances*) του προβλήματος. Μια άλλη ανάλυση ενδιαφέρεται για την **μακροπρόθεσμη απόσβεση** (*amortization*) επαναληπτικής χρήσης ενός αλγορίθμου. Η μελέτη της πολυπλοκότητας ενός αλγορίθμου μας επιτρέπει πολλές φορές να αποφανθούμε αν αυτός είναι **βέλτιστος** (*optimal*) για το συγκεκριμένο πρόβλημα. Αυτό προϋποθέτει ότι έχουμε **τα άνω (με αλγόριθμο)** και **κάτω (με απόδειξη) φράγματα του χρόνου** (ή και **του χώρου)** που επαρκούν και απαιτούνται για την επίλυση ενός προβλήματος και επίσης προϋποθέτει ότι αυτά ταυτίζονται.

Το κόστος ενός αλγορίθμου εξαρτάται φυσικά και από την είσοδο (input). Λογικά το κόστος αυξάνεται με την αύξηση του μεγέθους της εισόδου. Από την άλλη μεριά το κόστος μπορεί να διαφέρει για διαφορετικές εισόδους ίδιου μεγέθους.

Αντιστοιχούμε λοιπόν σ'έναν αλγόριθμο μία συνάρτηση, η οποία μας δίδει την ποσότητα χώρου ή χρόνου που απαιτείται για την επίλυση ενός προβλήματος με δεδομένα διαφόρου μεγέθους.

Το κόστος ενός αλγορίθμου ορίζεται με τη βοήθεια της παρακάτω συνάρτησης:

$$\text{κόστος αλγορίθμου}(n) = \max_{\substack{\text{για όλες τις δυνατές εισόδους } x \text{ μεγέθους } n}} \{\text{κόστος αλγορίθμου για είσοδο } x\}$$

Και το κόστος ενός προβλήματος, με τη βοήθεια της συνάρτησης:

$$\text{κόστος προβλήματος } (n) = \min_{\substack{\text{για όλους τους} \\ \text{αλγόριθμους A που} \\ \text{επιλύουν το πρόβλημα}}} \{\text{κόστος του αλγορίθμου } A(n)\}$$

**Αριθμητική Πολυπλοκότητα – Πολυπλοκότητα Ψηφιοπράξεων (Arithmetic *vs.* Bit Complexity)**

Η μέτρηση πολυπλοκότητας γίνεται συνήθως μετρώντας το πλήθος των στοιχειωδών αριθμητικών πράξεων ή εντολών (κάθε απλή αριθμητική πράξη, και κάθε απλή εντολή μιας γλώσσας προγραμματισμού θεωρούνται ότι έχουν μοναδιαίο κόστος) και τότε λέγεται *αριθμητική πολυπλοκότητα* (*arithmetic complexity*). Η πιο ακριβής μέτρηση πολυπλοκότητας που λαμβάνει υπ'όψη της το πλήθος των στοιχειωδών ψηφιοπράξεων (π.χ. η πρόσθεση και η σύγκριση δύο αριθμών $b$ ψηφίων απαιτούν $b$ ψηφιοπράξεις, ο πολλαπλασιασμός τους απαιτεί, με τον σχολικό αλγόριθμο, $b^2$ ψηφιοπράξεις, κ.ο.κ.) λέγεται *πολυπλοκότητα ψηφιοπράξεων* (*bit complexity*) και είναι απαραίτητη όταν στην είσοδο εμφανίζονται 'μεγάλοι' αριθμοί, των οποίων το μέγεθος επηρεάζει σημαντικά το πλήθος ή/και το κόστος των αριθμητικών πράξεων που θα εκτελέσει ο αλγόριθμος. [1]

### 5.1.3 Ντετερμινιστικοί και μη ντετερμινιστικοί αλγόριθμοι

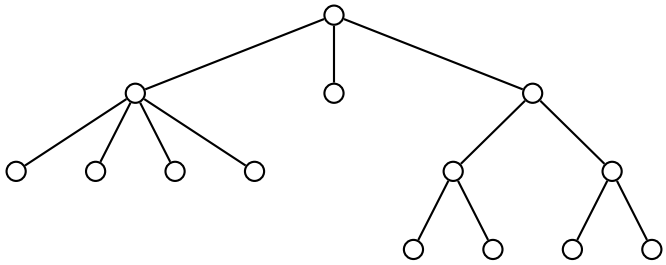
Ένας αλγόριθμος μπορεί να είναι **ντετερμινιστικός** (*deterministic*) ή **μη ντετερμινιστικός** (*non-deterministic*). Ο ντετερμινιστικός αλγόριθμος διακρίνεται από τα παρακάτω στοιχεία:

- Ο υπολογισμός που προτείνει είναι γραμμικός. Σε κάθε υπολογιστικό βήμα υπάρχει μία μοναδική επιτρεπτή επόμενη κατάσταση (configuration, διαμόρφωση).
- Η υπολογιστική διαδικασία προχωρεί βήμα προς βήμα και πάντοτε σταματάει για οποιαδήποτε είσοδο.

Σχήμα 5.1: Ντετερμινιστικός αλγόριθμος

Σχηματικά ο ντετερμινιστικός αλγόριθμος φαίνεται στο σχήμα 5.1. Ο μη ντετερμινιστικός αλγόριθμος παριστάνεται στο σχήμα 5.2. Όπως φαίνεται στο σχήμα 5.2 ο μη ντετερμινιστικός υπολογισμός είναι δέντρο, μη γραμμικός: για κάθε υπολογιστική διαμόρφωση (κόμβο του δέντρου) μπορεί να υπάρχουν πολλές, μία ή και καμία νόμιμες επόμενες υπολογιστικές διαμορφώσεις. Συνεπώς ο ντετερμινιστικός αλγόριθμος είναι ειδική περίπτωση μη ντετερμινιστικού αλγορίθμου. Σε μια άλλη ισοδύναμη περιγραφή ο μη ντετερμινιστικός αλγόριθμος αποτελείται από δύο διαφορετικές φάσεις. Στην πρώτη φάση ο μη ντετερμινιστικός αλγόριθμος επιλέγει μια επιτρεπτή ακολουθία από υπολογιστικές διαμορφώσεις (δηλαδή ένα μονοπάτι στο δέντρο) και στη

[1] Για τις ανάγκες ανάλυσης των περισσότερων αλγορίθμων που θα δούμε στο σύγγραμμα αυτό αρκεί η εκτίμηση της αριθμητικής πολυπλοκότητας. Όπου θα χρειαστούμε την πολυπλοκότητα ψηφιοπράξεων αυτό θα αναφέρεται ρητά.



Σχήμα 5.2: Μη ντετερμινιστικός αλγόριθμος

δεύτερη φάση, με μια καθαρά ντετερμινιστική διαδικασία ελέγχει αν το αποτέλεσμα που δίνει η πρώτη φάση αποτελεί λύση του προβλήματος. Η έννοια του μη ντετερμινιστικού αλγορίθμου δεν είναι κατ' ανάγκη πρακτικά χρήσιμη για την επίλυση προβλημάτων. Χρησιμεύει όμως στην θεωρητική ταξινόμηση της δυσκολίας των προβλημάτων.

Θέλοντας να τυποποιήσουμε δηλαδή να ορίσουμε αυστηρά την έννοια του αλγορίθμου, είναι απαραίτητο να ορίσουμε ένα συγκεκριμένο υπολογιστικό μοντέλο. Πολλοί επιστήμονες, όπως οι *A. Turing, A. Church, S. Kleene, E. Post, A. Markov* κ.α., ασχολήθηκαν με το θέμα αυτό και όρισαν διάφορα υπολογιστικά μοντέλα.

Το υπολογιστικό μοντέλο που αντιστοιχεί στον πιο "φυσικό" και διαισθητικό ορισμό του αλγορίθμου είναι η **μηχανή Turing**. Σύμφωνα με την αξιωματική **"θέση των Church-Turing"**:

> *"Κάθε αλγόριθμος μπορεί να περιγραφεί με τη βοήθεια μιας μηχανής Turing"*

ή διατυπωμένη αλλιώς:

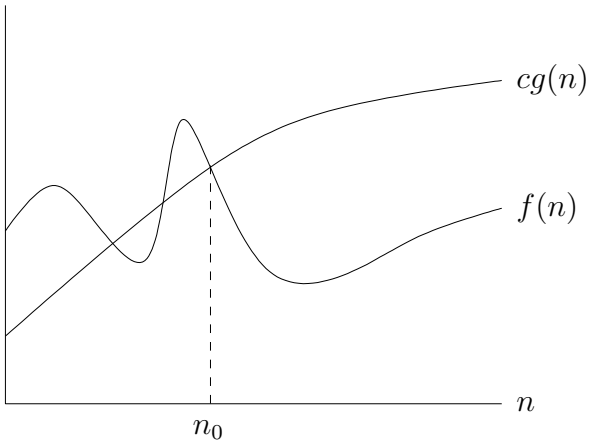
> *"Όλα τα γνωστά και άγνωστα υπολογιστικά μοντέλα είναι μηχανιστικά ισοδύναμα"*

δηλαδή:

> *"Για μια συγκεκριμένη συνάρτηση f, δοθέντος ενός αλγορίθμου σ'ένα υπολογιστικό μοντέλο μπορούμε με τη βοήθεια μηχανής (ή προγράμματος: compiler) να κατασκευάσουμε, για την ίδια συνάρτηση f, αλγόριθμο σ'ένα άλλο υπολογιστικό μοντέλο".*

## 5.2 Μαθηματικοί Συμβολισμοί

Συνήθως δεν μας ενδιαφέρει να μετρήσουμε ακριβώς το κόστος εκτέλεσης ενός αλγόριθμου αλλά να βρούμε μόνο την τάξη μεγέθους του κόστους. Μας ενδιαφέρει η ασυμπτωτική συμπεριφορά του αλγόριθμου. Με άλλα λόγια αναζητούμε την οριακή αυξητική τάση της συνάρτησης που εκφράζει την πολυπλοκότητα του αλγόριθμου καθώς αυξάνεται το μέγεθος της εισόδου (input). Έτσι λοιπόν ένας αλγόριθμος που έχει καλύτερη ασυμπτωτική συμπεριφορά (μικρότερη τάξη μεγέθους ή, ισοδύναμα, μικρότερο ρυθμό αύξησης) από έναν άλλο για το ίδιο πρόβλημα, θα είναι και η καλύτερη επιλογή για όλα τα μεγέθη της εισόδου (εκτός ίσως από τα πολύ μικρά).




Σχήμα 5.3: $f = O(g)$

Ορίζουμε τώρα κάποια σύμβολα που χρησιμοποιούνται στη μελέτη της **αυξητικής τάσης** (*rate of growth*) μιας συνάρτησης:

Έστω $f : \mathbb{N}\backslash\{0\} \to \mathbb{N}\backslash\{0\}$ και $g : \mathbb{N}\backslash\{0\} \to \mathbb{N}\backslash\{0\}$, όπου $\mathbb{N}$ οι φυσικοί αριθμοί. Να σημειώσουμε εδώ ότι μας ενδιαφέρουν μόνο μονότονα αύξουσες συναρτήσεις (η συνάρτηση $\frac{1000}{n}$ π.χ. δεν εμφανίζεται ως πολυπλοκότητα αλγορίθμου).

$$O(g) = \{f \mid \exists c > 0,\ \exists n_0 : \forall n > n_0 \ \ f(n) \leq cg(n)\}$$

Το $O(g)$ περιλαμβάνει "χοντρικά" τις συναρτήσεις μικρότερης ή ίδιας τάξης μεγέθους με την $g$.

$$\Omega(g) = \{f \mid \exists c > 0,\ \exists n_0 : \forall n > n_0 \ \ f(n) \geq cg(n)\}$$

Το $\Omega(g)$ περιλαμβάνει "χοντρικά" τις συναρτήσεις μεγαλύτερης ή ίδιας τάξης μεγέθους με την $g$.

$$\Theta(g) = \{f \mid \exists c_1 > 0,\ \exists c_2 > 0,\ \exists n_0 : \forall n > n_0 \ \ c_1 \leq \frac{f(n)}{g(n)} \leq c_2\}$$

Το $\Theta(g)$ περιλαμβάνει "χοντρικά" τις συναρτήσεις της ίδιας τάξης μεγέθους με την $g$.

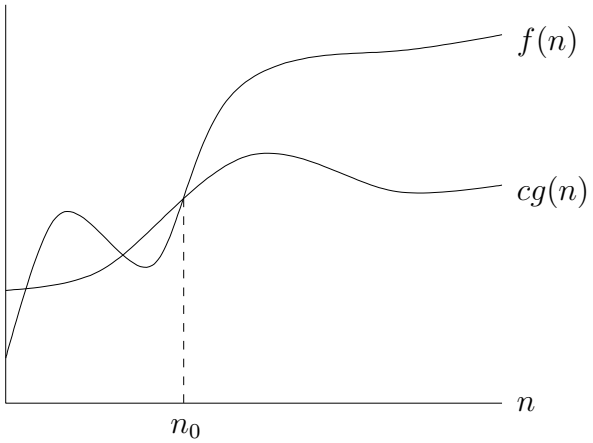
Αν $g \in O(f)$ συνήθως γράφουμε $g(n) = O(f(n))$ και λέμε ότι συνάρτηση $g$ είναι τάξης μεγέθους $f$.

Ισχύουν: $\Theta(f) = O(f) \bigcap \Omega(f)$ και $f \in \Theta(g) \iff (f \in O(g)$ και $\ g \in O(f))$.

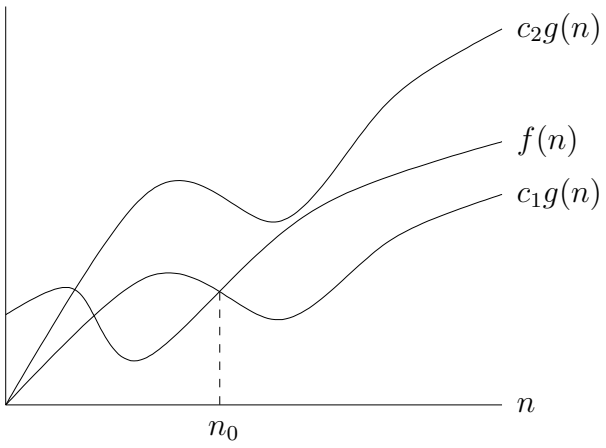
Αν $p = c_k n^k + c_{k-1} n^{k-1} + \cdots + c_0$, δηλαδή πολυώνυμο βαθμού $k$, τότε $p \in \mathrm{O}(n^k)$ ή $p(n) = \mathrm{O}(n^k)$. Επίσης $p \in \Omega(n^k)$ ή $p(n) = \Omega(n^k)$. Συνεπώς $p(n) = \Theta(n^k)$.

Ορίζουμε $\mathrm{O}(poly) = \bigcup \mathrm{O}(n^k)$.

Γενικά ισχύει ότι: $O(1) < O(\log^* n) < O(\log(n)) < O(\sqrt{n}) < O(n) < O(n \log(n)) <$




Σχήμα 5.4: $f = \Omega(g)$




Σχήμα 5.5: $f = \Theta(g)$

$O(n^2) < \ldots < O(poly) < O(2^n) < O(n!) < O(n^n) < O(2^{\overset{\overset{2}{\cdot^{\cdot^{\cdot}}}}{\}n}})$
Σημείωση: γράφουμε "$<$" αντί "$\subset$".

**Ορισμός 5.1.** $\log^* n$ δίνει πόσες φορές πρέπει να λογαριθμήσουμε το $n$ για να πάρουμε 1.

## 5.3 Εύρεση μέγιστου κοινού διαιρέτη

Δίνονται δύο θετικοί ακέραιοι $a$ και $b$. Αναζητούμε τον μέγιστο κοινό διαιρέτη τους— ΜΚΔ (greatest common divisor — gcd):

$$z := gcd(a, b)$$

Μία πρώτη ιδέα είναι να παραγοντοποιήσουμε τους αριθμούς σε γινόμενο πρώτων (από το Θεώρημα μοναδικής παραγοντοποίησης σε πρώτους, για κάθε θετικό ακέραιο αυτό το γινόμενο είναι μοναδικό) και να εντοπίσουμε τους κοινούς παράγοντες, δηλαδή να ανάγουμε το πρόβλημα σε αυτό της εύρεσης πρώτων παραγόντων. Δυστυχώς, ο αλγόριθμος δεν είναι ιδιαίτερα αποδοτικός. Μάλιστα, δεν γνωρίζουμε καν αν υπάρχει αλγόριθμος πολυωνυμικού χρόνου για το πρόβλημα εύρεσης πρώτων παραγόντων.

### 5.3.1 Ένας απλός αλγόριθμος για το gcd

$z := min(a, b)$;
**while** $(a$ **mod** $z \neq 0)$ **or** $(b$ **mod** $z \neq 0)$ **do** $z := z - 1$;
**return** $(z)$

Η ορθότητα του παραπάνω αλγορίθμου είναι προφανής: Δεν υπάρχει αριθμός $w > z$ τέτοιος ώστε να διαιρεί και τον $a$ και τον $b$.

Η πολυπλοκότητα του αλγορίθμου είναι $O(\min(a, b))$ στην χειρότερη περίπτωση και αυτό συμβαίνει όταν οι αριθμοί $a$, $b$ είναι πρώτοι μεταξύ τους. Ακόμη όμως και στην μέση περίπτωση ο αλγόριθμος δεν αποδίδει καλά, δεδομένου ότι αν το $\min(a, b)$ δεν διαιρεί ακριβώς το $\max(a, b)$, τότε ο βρόχος επαναλαμβάνεται τουλάχιστον $\min(a, b)/2$ φορές.

### 5.3.2 Αλγόριθμος με αφαιρέσεις για το gcd

$i := a$; $j := b$;
**while** $i \neq j$ **do if** $i > j$ **then** $i := i - j$ **else** $j := j - i$;
**return** $(i)$

Η απόδειξη της ορθότητας βασίζεται στο εξής: Αν $w$ διαιρεί το $a$ και το $b$ (με $a > b$) τότε διαιρεί και το $a - b$.

Όσο για την πολυπλοκότητα, αυτή είναι $O(\max(a, b))$ για την χειρότερη περίπτωση, όταν για παράδειγμα το $b = 1$. Πάντως, στην μέση περίπτωση ο αλγόριθμος με τις αφαιρέσεις είναι καλύτερος από τον προηγούμενο αλγόριθμο.

### 5.3.3 Αλγόριθμος του Ευκλείδη

$i := a$; $j := b$;
**while** $(i > 0)$ **and** $(j > 0)$ **do**
  **if** $i > j$ **then** $i := i$ **mod** $j$ **else** $j := j$ **mod** $i$;
**return** $(i + j)$

Η απόδειξη της ορθότητας βασίζεται στο εξής: Αν $w$ διαιρεί το $a$ και το $b$ (με $a > b$) τότε διαιρεί και το $a$ **mod** $b$.

Ο αλγόριθμος έχει πολυπλοκότητα χειρότερης περίπτωσης $O(\log(a+b))$, αλλά και $O(\log(\min(a, b)))$ [2]. Συνήθως, όμως, τερματίζει πιο γρήγορα.

Μάλιστα, την χειρότερη απόδοση ο αλγόριθμος του Ευκλείδη την παρουσιάζει αν του δοθούν ως είσοδος δύο διαδοχικοί αριθμοί της ακολουθίας Fibonacci:

$$F_k = \begin{cases} 0 & \text{για } k = 0 \\ 1 & \text{για } k = 1 \\ F_{k-1} + F_{k-2} & \text{για } k \geq 2 \end{cases}$$

Ισχύει $F_k = (\phi^k - \hat{\phi}^k)/\sqrt{5}$, όπου $\phi = (1 + \sqrt{5})/2 \approx 1.618$ (γνωστή και ως η *χρυσή τομή*) και $\hat{\phi} = (1 - \sqrt{5})/2$. Ας σημειώσουμε ότι επειδή $|\hat{\phi}| < 1$, το $F_k$ είναι ίσο με το $\phi^k/\sqrt{5}$ στρογγυλοποιημένο στον πλησιέστερο ακέραιο, οπότε προκύπτει το παρακάτω πόρισμα:

**Πόρισμα 5.2.** $\log_\phi(F_k) + 1 \leq k \leq \log_\phi(F_k) + 2$

Έχουμε τα παρακάτω αποτελέσματα σχετικά με την πολυπλοκότητα:

**Πρόταση 5.3.** *Ο αλγόριθμος του Ευκλείδη για $a = F_{k+1}$ και $b = F_k$ έχει χρονική πολυπλοκότητα $\Theta(k)$.*

**Πόρισμα 5.4.** *Η χρονική πολυπλοκότητα του αλγορίθμου του Ευκλείδη είναι $\Omega(\log(a + b))$.*

**Λήμμα 5.5.** *Η χρονική πολυπλοκότητα του αλγορίθμου του Ευκλείδη είναι $O(\log(a + b))$.*

*Απόδειξη.* Σε δύο επαναλήψεις, ο μεγαλύτερος από τους δύο αριθμούς τουλάχιστον υποδιπλασιάζεται. Οι λεπτομέρειες αφήνονται ως άσκηση. □

**Θεώρημα 5.6.** *Η χρονική πολυπλοκότητα του αλγορίθμου του Ευκλείδη είναι $\Theta(\log(a + b))$.*

*Απόδειξη.* Προκύπτει από το παραπάνω λήμμα ότι ο αλγόριθμος θα εκτελέσει το πολύ $2\log(a + b)$ επαναλήψεις. Το πλήθος βημάτων κάθε επανάληψης φράσσεται από μία μικρή σταθερά. □

[2] Αυτό μοιάζει κάπως παράδοξο, θυμηθείτε όμως ότι αναφερόμαστε στην αριθμητική πολυπλοκότητα, δηλαδή στο πλήθος απλών εντολών που εκτελεί ο αλγόριθμος.

## 5.4 Ύψωση σε δύναμη με επαναλαμβανόμενο τετραγωνισμό (repeated squaring)

Για να υπολογίσουμε το $a^n$, ο απλοϊκός αλγόριθμος χρειάζεται $n$ πολλαπλασιασμούς, επομένως έχει (αριθμητική) πολυπλοκότητα $O(n)$. Ο παρακάτω αλγόριθμος επιτυγχάνει σημαντική βελτίωση.

**function** fastpower($a$, $n$)
  result := 1;
  **while** $n$>0 **do**
    **if** odd($n$) **then** result:=result*$a$;
    $n$ := $n$ **div** 2;
    $a$ := $a$*$a$
  **return** result

*Παράδειγμα* 5.7. $a^{13} = a^{1\cdot 2^3+1\cdot 2^2+0\cdot 2^1+1\cdot 2^0}$

Η (αριθμητική) πολυπλοκότητα του αλγορίθμου είναι $O(\log n)$, δηλαδή πολυωνυμική ως προς το μήκος της εισόδου.

## 5.5 Αριθμοί Fibonacci (Υπολογισμός)

Παρακάτω δίνονται 3 αλγόριθμοι για τον υπολογισμό του $n$-οστού αριθμού Fibonacci (βλ. και πιο πάνω για ορισμό), και συγκρίνεται η πολυπλοκότητά τους.

### 5.5.1 Αναδρομικός αλγόριθμος

**function** $F(n)$
  **if** ($n$<2) **then return** $n$
  **else return** $F(n-1)$+$F(n-2)$;

Πολυπλοκότητα: $T(n) = T(n-1) + T(n-2) + c$, δηλαδή η $T(n)$ ορίζεται όπως η $F(n)$ (συν κάτι μικρό), οπότε (μπορούμε να αποδείξουμε ότι):

$$(n) > F(n) = \Omega(1.62^n)$$

### 5.5.2 Επαναληπτικός αλγόριθμος

**function** $F(n)$
  $a$:=0; $b$:=1;
  **for** i:=2 **to** $n$ **do**
    $c$:=$b$; $b$:=$a$+$b$; $a$:=$c$;
  **return** $b$;

Η πολυπλοκότητα είναι γραμμική, $O(n)$.

### 5.5.3 Αλγόριθμος με πίνακα

Μπορούμε να γράψουμε τον υπολογισμό σε μορφή πινάκων:

$$\begin{pmatrix} F(n) \\ F(n-1) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} F(n-1) \\ F(n-2) \end{pmatrix}$$

Από το παραπάνω συμπεραίνουμε ότι:

$$\begin{pmatrix} F(n) \\ F(n-1) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-2} \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Αν χρησιμοποιηθεί ο αλγόριθμος επαναλαμβανόμενου τετραγωνισμού για την ύψωση σε δύναμη του πίνακα, τότε ο αριθμός των αριθμητικών πράξεων μειώνεται στο $O(\log n)$. Έτσι, ο αλγόριθμος αυτός έχει αριθμητική πολυπλοκότητα που είναι πολυωνυμική ως προς το μέγεθος της εισόδου. Δεν συμβαίνει όμως το ίδιο για την πολυπλοκότητα ψηφιοπράξεων του αλγορίθμου και την χωρική πολυπλοκότητα. Η εκτίμηση αυτών των πολυπλοκοτήτων, και η σύγκριση με τις αντίστοιχες του επαναληπτικού αλγορίθμου αφήνεται ως άσκηση.

### 5.5.4 Σύγκριση αλγορίθμων

Θεωρήστε τέσσερα προγράμματα με αριθμό βημάτων $O(2^n)$, $O(n^2)$, $O(n)$, και $O(\log n)$ που το καθένα χρειάζεται 1 δευτερόλεπτο για να υπολογίσει το $F(100)$ (πιθανόν σε διαφορετικούς υπολογιστές). Στον παρακάτω πίνακα, φαίνεται πόσα δευτερόλεπτα θα χρειαστούν για να υπολογιστεί το $F(n)$.

| | $c_1 \cdot 2^n$ | $c_2 \cdot n^2$ | $c_3 \cdot n$ | $c_4 \cdot \log n$ |
|---|---|---|---|---|
| $F(100)$ | 1 | 1 | 1 | 1 |
| $F(101)$ | 2 | 1.02 | 1.01 | 1.002 |
| $F(110)$ | 1024 | 1.21 | 1.1 | 1.02 |
| $F(200)$ | $2^{100}$ | 4 | 2 | 1.15 |

## 5.6 Δυαδική αναζήτηση (Binary Search)

Η δυαδική αναζήτηση είναι ίσως το πιο απλό παράδειγμα της τεχνικής **Divide and Conquer**. Στην δυαδική αναζήτηση, για να διαπιστώσουμε αν ένα δοσμένο κλειδί είναι στοιχείο ενός ήδη ταξινομημένου πίνακα (έστω μη φθίνουσα διάταξη), συγκρίνουμε το κλειδί με το στοιχείο που βρίσκεται στη μεσαία θέση του πίνακα. Αν το κλειδί είναι μικρότερο τότε θα βρίσκεται (αν υπάρχει) στο πρώτο μισό του πίνακα, αλλιώς στο δεύτερο μισό. Συνεχίζοντας αναδρομικά την εφαρμογή της παραπάνω διαδικασίας, καταλήγουμε στο να εντοπίσουμε, αν υπάρχει, το δοσμένο κλειδί. Μια υλοποίηση της δυαδικής αναζήτησης είναι αυτή που φαίνεται στον αλγόριθμο 5.1.

**Αλγόριθμος 5.1** Δυαδική αναζήτηση (Binary Search)

```
function BinSearch (a:array[p..q] of item; search: item): info;
  var first, last, mid: index; found: boolean;
begin
  first:=p; last:=q; found:=(search=a[first]) or (search=a[last]);
  while not found and (first<last) do
  begin
    mid := (first+last) div 2;
    if search < a[mid] then begin last:=mid−1; first:=first+1 end
                       else begin first:=mid; last:=last−1 end
    found:=(search = a[first]) or (search = a[last])
  end
  if found then if search=a[first] then return(first) else return(last)
           else return('not found')
end
```

Αν $T(n)$ είναι η συνάρτηση που δίνει την χρονική πολυπλοκότητα του αλγορίθμου 5.1, τότε εύκολα καταλαβαίνουμε ότι ισχύει:

$$T(n) = \begin{cases} a & , \text{για } n = 1 \\ T(\frac{n}{2}) + c & , \text{για } n > 1 \end{cases}$$

Στην περίπτωση που $n > 1$ έχουμε:

$$T(n) \quad = \quad T(\frac{n}{2}) + c = T(\frac{n}{4}) + 2c = T(\frac{n}{8}) + 3c = \ldots = T(\frac{n}{2^k}) + kc$$

Αν ο πίνακας έχει $2^k$ στοιχεία ($n = q - p + 1 = 2^k$), τότε έχουμε ότι $T(n) = T(1) + c \log n$. Άρα ο αλγόριθμος 5.1, σε ταξινομημένο πίνακα στοιχείων χρειάζεται χρόνο τάξης $O(\log n)$.

*Σημείωση* 5.1. Η τεχνική της δυαδικής αναζήτησης εφαρμόζεται άμεσα και σε περιπτώσεις που αναζητούμε ρίζες μονότονων συναρτήσεων σε συγκεκριμένο διάστημα ακέραιων ή και πραγματικών αριθμών (κατά προσέγγιση). Η πολυπλοκότητά της αποδεικνύεται ότι είναι $O(\log n + k)$ όπου $n$ το εύρος του διαστήματος και $k$ η ακρίβεια σε δεκαδικά ψηφία. Η απόδειξη αφήνεται ως άσκηση.

## 5.7 Εύρεση του μεγαλύτερου και του μικρότερου στοιχείου

Έστω ένας πίνακας $S_n$ που τα στοιχεία του είναι ακέραιοι αριθμοί μη ταξινομημένοι. Ζητάμε να βρούμε το μεγαλύτερο και το μικρότερο στοιχείο του πίνακα. Ο απλός αλγόριθμος δουλεύει ως εξής: με $n - 1$ συγκρίσεις (για $n \geq 2$) βρίσκουμε το μεγαλύτερο από τα στοιχεία του πίνακα και έπειτα με $n - 2$ βρίσκουμε το μικρότερο από τα $n - 1$ στοιχεία. Δηλαδή συνολικά έχουμε $2n - 3$ συγκρίσεις. Η τεχνική **Divide and Conquer** χωρίζει την ακολουθία των $n$ ακεραίων σε

δύο υποακολουθίες μήκους $\frac{n}{2}$ και υπολογίζει το μεγαλύτερο και το μικρότερο στοιχείο αυτών των υποακολουθιών. Στη συνέχεια με δύο συγκρίσεις, μια ανάμεσα στα μεγαλύτερα και μια ανάμεσα στα μικρότερα στοιχεία, βρίσκει το μεγαλύτερο και το μικρότερο στοιχείο της αρχικής ακολουθίας. Τα μεγαλύτερα και τα μικρότερα στοιχεία των υποακολουθιών υπολογίζονται αναδρομικά. Αν $T(n)$ είναι η συνάρτηση που δίνει τον αριθμό των συγκρίσεων, είναι εύκολο να δούμε ότι ισχύει:

$$T(n) = \begin{cases} 0 & , \text{για } n = 1 \\ 1 & , \text{για } n = 2 \\ 2T(\frac{n}{2}) + 2 & , \text{για } n > 2 \end{cases}$$

Για $n > 2$ έχουμε:

$$\begin{aligned} T(n) &= 2T(\frac{n}{2}) + 2 = \\ &= 4T(\frac{n}{4}) + 4 + 2 = \\ &= 8T(\frac{n}{8}) + 8 + 4 + 2 = \\ &= \ldots = \\ &= 2^k T(\frac{n}{2^k}) + 2\sum_{i=0}^{k-1} 2^i = \\ &= 2^k T(\frac{n}{2^k}) + 2\frac{2^k - 1}{2 - 1} = \\ &= 2^k T(\frac{n}{2^k}) + 2^{k+1} - 2 \end{aligned}$$

Όταν ο πίνακας έχει $n = 2^{k+1}$ στοιχεία ($\frac{n}{2} = 2^k$) τότε έχουμε:

$$T(n) = \frac{n}{2}T(2) + 2^{k+1} - 2 = \frac{3}{2}n - 2 \tag{5.1}$$

## 5.8 Πολλαπλασιασμός ακεραίων (integer multiplication)

Το πρόβλημα του κλασσικού πολλαπλασιασμού δύο $n$-bit ακεραίων, έστω των $X$ και $Y$, περιλαμβάνει τον υπολογισμό $n$ μερικών γινομένων μεγέθους $n$, είναι δηλαδή μια πράξη με πολυπλοκότητα της τάξης $O(n^2)$. Η τεχνική Divide and Conquer χωρίζει τον καθένα από τους $X$ και $Y$ σε δύο ακεραίους που έχουν μήκος $\frac{n}{2}$ bits ο καθένας. Αν υποθέσουμε ότι το $n$ είναι μια δύναμη του 2 τότε έχουμε:

$$X : \boxed{a}\boxed{b} \quad = a \cdot 2^{\frac{n}{2}} + b$$

$$Y : \boxed{c}\boxed{d} \quad = c \cdot 2^{\frac{n}{2}} + d$$

Τότε το γινόμενο των $X$ και $Y$ εκφράζεται ως εξής:

$$X\ Y = ac \cdot 2^n + (ad + bc) \cdot 2^{\frac{n}{2}} + bd \tag{5.2}$$

Για να υπολογίσουμε αυτό το γινόμενο χρειάζεται να κάνουμε τέσσερις πολλαπλασιασμούς $\frac{n}{2}$-bit ακεραίων, τρεις προσθέσεις, το πολύ $2n$-bit, ακεραίων και δύο ολισθήσεις. Αφού οι προσθέσεις και οι ολισθήσεις είναι πράξεις πολυπλοκότητας $O(n)$ μπορούμε να γράψουμε την παρακάτω αναδρομική σχέση για να περιγράψουμε τη συνάρτηση χρονικής πολυπλοκότητας του πολλαπλασιασμού:

$$T(n) = \begin{cases} a & \text{, για } n = 1 \\ 4T(\frac{n}{2}) + cn & \text{, για } n > 1 \end{cases}$$

Κάνοντας πράξεις όπως και στις προηγούμενες περιπτώσεις καταλήγουμε στο:

$$T(n) = (c+1)n^2 - cn = O(n^2)$$

Δηλαδή η πράξη του πολλαπλασιασμού αν γίνει με τη χρήση της 5.1, έχει την ίδια πολυπλοκότητα με τον κλασικό πολλαπλασιασμό. Μπορούμε να βελτιώσουμε την κατάσταση αν μειώσουμε τον αριθμό των υποπροβλημάτων και αυτό γίνεται με την παρακάτω παρατήρηση:

$$(ad + bc) \;=\; [(a-b)(d-c) + ac + bd] \tag{5.3}$$

Συνδυάζοντας τις 5.1 και 5.2 έχουμε:

$$X\,Y = ac \cdot 2^n + [(a-b)(d-c) + ac + bd]\; 2^{\frac{n}{2}} + bd \tag{5.4}$$

Για τον υπολογισμό της 5.3 χρειάζονται μόνο τρεις πολλαπλασιασμοί $\frac{n}{2}$-bit ακεραίων $(ac,\ (a-b)(d-c), bd)$ και οι υπόλοιπες πράξεις έχουν τάξη πολυπλοκότητας $O(n)$. Η συνάρτηση $T(n)$ αυτή τη φορά έχει ως εξής (a σταθερά):

$$T(n) = \begin{cases} a & \text{, για } n = 1 \\ 3T(\frac{n}{2}) + cn & \text{, για } n > 1 \end{cases}$$

Τελικά έχουμε ότι:

$$T(n) \;=\; O(n^{\log_2 3}) \;=\; O(n^{1.59})$$

## 5.9 Πολλαπλασιασμός πινάκων (matrix multiplication)

Παρόμοιο με την προηγούμενη εφαρμογή είναι και το πρόβλημα πολλαπλασιασμού δύο πινάκων. Αν $A$ και $B$ είναι δύο πίνακες $n \times n$, το γινόμενό τους $A \times B$ είναι ένας $n \times n$ πίνακας $C$, του οποίου τα στοιχεία δίνονται από τον τύπο:

$$c_{ij} \;=\; \sum_{k=1}^{n} a_{ik} b_{kj}$$

Η κλασσική μέθοδος πολλαπλασιασμού, δηλαδή η άμεση εφαρμογή του παραπάνω τύπου, έχει πολυπλοκότητα χρόνου $O(n^3)$. Η μέθοδος DIVIDE AND CONQUER, προτείνει τον διαχωρισμό κάθε πίνακα σε τέσσερις υποπίνακες διαστάσεων $\frac{n}{2} \times \frac{n}{2}$ ο καθένας. Αν θεωρήσουμε ότι $n = 2^k$, τότε το γινόμενο των πινάκων $A \times B$ θα δίνεται από τον τύπο:

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

όπου

$$\begin{aligned} C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\ C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\ C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\ C_{22} &= A_{21}B_{12} + A_{22}B_{22} \end{aligned}$$

και η πολυπλοκότητα του αλγορίθμου ως προς το χρόνο θα είναι:

$$T(n) \;=\; 8T(\frac{n}{2}) + 4c(\frac{n}{2})^2 \;=\; O(n^3)$$

($c$ σταθερά) ενώ η πολυπλοκότητα χώρου θα είναι $S(n) = O(1)$. Σημειωτέον ότι για τον πολλαπλασιασμό αυτό θα χρειαστούν οκτώ πολλαπλασιασμοί $\frac{n}{2} \times \frac{n}{2}$ πινάκων και τέσσερις προσθέσεις $\frac{n}{2} \times \frac{n}{2}$ πινάκων. Η πολυπλοκότητα $O(n^3)$ προκύπτει με προσεκτικές αντικαστάσεις ή με τη μέθοδο του Κυρίαρχου Όρου (Master Theorem).

Το 1969 ο *Volker Strassen* απέδειξε ότι για να προσδιορίσουμε τα $C_{ij}$, αρκεί να χρησιμοποιήσουμε μόνον επτά πολλαπλασιασμούς $\frac{n}{2} \times \frac{n}{2}$ πινάκων και δεκαοκτώ προσθέσεις $\frac{n}{2} \times \frac{n}{2}$ πινάκων. Σύμφωνα με τη μέθοδο του, πρώτα υπολογίζονται οι επτά $\frac{n}{2} \times \frac{n}{2}$ πίνακες $P,\ Q,\ R,\ S,\ T,\ U,\ V$ και μετά υπολογίζονται τα $C_{ij}$:

$$\begin{aligned} P &= (A_{11} + A_{22})(B_{11} + B_{22}) \\ Q &= (A_{21} + A_{22})B_{11} \\ R &= A_{11}(B_{12} - B_{22}) \\ S &= A_{22}(-B_{11} + B_{12}) \\ T &= (A_{11} + A_{12})B_{22} \\ U &= (A_{21} - A_{11})(B_{11} + B_{12}) \\ V &= (A_{12} - A_{22})(B_{21} + B_{22}) \end{aligned}$$

Τα $C_{ij}$ υπολογίζονται ως εξής:

$$\begin{aligned} C_{11} &= P + S - T + V \\ C_{12} &= R + T \\ C_{21} &= Q + S \\ C_{22} &= P + R - Q + U \end{aligned}$$

Η πολυπλοκότητα χρόνου σε αυτήν την περίπτωση είναι:

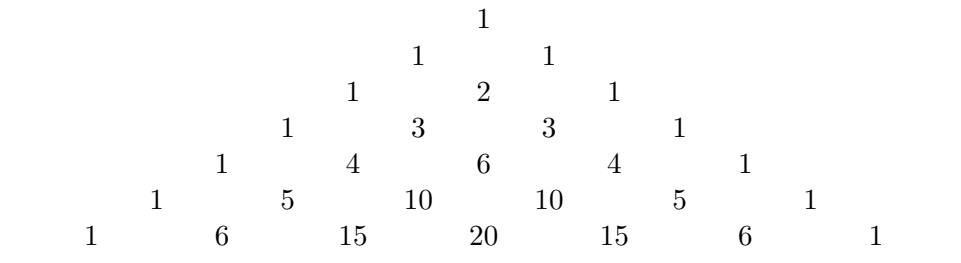
$$T(n) \;=\; 7T(\frac{n}{2}) + 18(\frac{n}{2})^2 \;\leq\; 43n^{\log 7}$$

Άρα:

$$T(n) \;=\; O(n^{\log 7}) \;\approx\; O(n^{2.81})$$

όπως προκύπτει με προσεκτικές αντικαστάσεις ή με τη μέθοδο του Κυρίαρχου Όρου (Master Theorem).

Ο *Strassen* απέδειξε επίσης ότι ο αριθμός 7 (πολλαπλασιασμοί υποπινάκων) δεν μπορεί να βελτιωθεί. Στα τελευταία είκοσι χρόνια έχουν βρεθεί καλύτεροι αλγόριθμοι (*Schonhage, Bini, Pan*)




Σχήμα 5.6: Το τρίγωνο του Pascal

με πολυπλοκότητα $T(n) \approx O(n^{2.38})$. Από την άλλη μεριά το καλύτερο γνωστό κάτω φράγμα είναι $\Omega(n^2)$. Υπάρχει δηλαδή ένα κενό μεταξύ άνω και κάτω φράγματος. Συνεπώς αυτή η βελτίωση πολυπλοκότητας πολλαπλασιασμού πινάκων είναι ένα πεδίο έρευνας με έντονη δραστηριότητα.

*Σημείωση* 5.2. Πολυπλοκότητα χρόνου ίδια με αυτή του πολλαπλασιασμού δύο πινάκων $(M(n))$, έχει και ο αλγόριθμος εύρεσης αντιστρόφου πίνακα (όπως και ο αλγόριθμος εύρεσης της ορίζουσας του πίνακα, κ.α.), γιατί μπορεί να αναχθεί σε πολλαπλασιασμό πινάκων.

## 5.10 Τρίγωνο Pascal

Το τρίγωνο του Pascal είναι το (άπειρο) τρίγωνο που φαίνεται στο σχήμα 5.10. Κάθε όρος προκύπτει προσθέτοντας τους ακριβώς από επάνω όρους. Επίσης, ο $k-$οστός όρος της $n-$οστής γραμμής του τριγώνου, $a_{n,k}$ προκύπτει και από τον τύπο:

$$a_{n,k} = \left( \begin{array}{c} n \\ k \end{array} \right) = \frac{n!}{k! \cdot (n-k)!}$$

Οι όροι του τριγώνου του Pascal λέγονται και διωνυμικοί συντελεστές, γιατί ο $a_{n,k}$ είναι ο $k-$οστός συντελεστής του αναπτύγματος του διωνύμου $(a+b)^{n'}$:

$$(a+b)^n = \left( \begin{array}{c} n \\ 0 \end{array} \right) a^n + \left( \begin{array}{c} n \\ 1 \end{array} \right) a^{n-1}b + \ldots + \left( \begin{array}{c} n \\ n \end{array} \right) a^0 b^n$$

Ένας αλγόριθμος για την εύρεση της $n-$οστής γραμμής του τριγώνου είναι ο αλγόριθμος 5.2. Χρησημοποιεί έναν πίνακα $1 \times n$.

$$a_{n+1,k+2} = a_{n,k+1} + a_{n,k+2}$$

Ο αλγόριθμος αυτός χρειάζεται $\frac{n^2}{2}$ βήματα για να υπολογίσει τη $n-$οστή γραμμή. Υπάρχει όμως και πιο αποδοτικός αλγόριθμος. Αυτός υπολογίζει κάθε όρο της $n-$οστής γραμμής του τριγώνου

**Αλγόριθμος 5.2** Αλγόριθμος για εύρεση της $n-$οστής γραμμής του τριγώνου του Pascal. Στηρίζεται στην παραπάνω σχέση.

```
program pasctri (n);
 var i, j:integer;
     a: array [0..n] of integer;
begin
  for i := 0 to n do
  begin
    a[i] := 1;
    for k := i downto 1 do a[k] := a[k − 1] + a[k]
  end
  output(a)
end.
```

και τον χρησιμοποιεί για να υπολογίσει τον επόμενο. Στηρίζεται στη σχέση

$$\left( \begin{array}{c} n \\ k+1 \end{array} \right) = \left( \begin{array}{c} n \\ k \end{array} \right) \cdot \frac{n-k}{k+1},$$

που είναι προφανής από τον ορισμό των $\left( \begin{array}{c} n \\ k \end{array} \right)$.

**Αλγόριθμος 5.3** Αλγόριθμος σε $n+1$ βήματα για την εύρεση της $n-$οστής γραμμής του τριγώνου του Pascal.

```
program pasctri2 (n);
 var i:integer;
     a: array [0..n] of integer;
begin
  a[0] := 1
  for i := 1 to n do
    a[i] := a[i − 1] ∗ (n − i)div(i + 1);
  output(a)
end.
```

## 5.11 Πρώτοι αριθμοί – Παραγοντοποίηση – Κρυπτοσύστημα RSA

Ίσως τα σπουδαιότερα, και σίγουρα τα πλέον "ιστορικά" προβλήματα της Θεωρίας Αριθμών είναι το Primality (έλεγχος αν ένας αριθμός είναι πρώτος) και το Factoring (παραγοντοποίηση). Στην ενότητα αυτή θα συζητήσουμε την πολυπλοκότητά τους και το ρόλο τους στην κρυπτογραφία δημοσίου κλειδιού.

**Primality ('Ελεγχος αν ένας αριθμός είναι πρώτος).** Το πρόβλημα Primality ορίζεται ως εξής: "Δίνεται ακέραιος $n$. Είναι πρώτος;". Το πρόβλημα αυτό είναι υπολογιστικά ευεπίλυτο, αν και αυτό δεν είναι καθόλου προφανές. Συγκεκριμένα, ανήκει στην κλάση **P** όπως έδειξαν πρόσφατα οι Agrawal, Kayal και Saxena, που επινόησαν τον αλγόριθμο AKS το 2002 (δες 11.3 για ορισμούς). Για πολλά χρόνια δεν ήταν γνωστό αν το πρόβλημα ήταν στο **P**, αλλά είχαν επινοηθεί ήδη από την δεκαετία του '70 αποδοτικοί *πιθανοτικοί* αλγόριθμοι από τους Miller-Rabin και Solovay-Strassen. Οι αλγόριθμοι αυτοί χρησιμοποιούνται ακόμη και σήμερα καθώς είναι καλύτερης πολυπλοκότητας από τον αλγόριθμο AKS και επίσης πιο εύκολοι στην υλοποίησή τους.

Παρακάτω παρουσιάζουμε τον έλεγχο Miller-Rabin. Στην αρχή θα περιγράψουμε έναν ακόμη απλούστερο πιθανοτικό αλγόριθμο ελέγχου πρώτων, τον έλεγχο Fermat (που όμως αποτυγχάνει σε ελάχιστες περιπτώσεις όπως θα εξηγήσουμε).

*Θεώρημα* 5.1 (Μικρό Θεώρημα Fermat). *Αν $n$ πρώτος τότε για κάθε $a, 1 \leq a < n-1$, ισχύει*

$$a^{n-1} \bmod n = 1$$

Το θεώρημα αποδεικνύεται με χρήση στοιχειώδους Θεωρίας Αριθμών και η απόδειξη μπορεί να βρεθεί στα περισσότερα σχετικά εγχειρίδια ή και στο διαδίκτυο (WikiPedia).

Από το μ. Θεώρημα Fermat προκύπτει άμεσα ο παρακάτω απλός αλγόριθμο ελέγχου πρώτων αριθμών, που όμως δεν δουλεύει σωστά για κάποιους (πολύ λίγους) αριθμούς, συγκεκριμένα για τους αριθμούς Carmichael (περισσότερα παρακάτω).

**Έλεγχος (test) του Fermat**:

1. Input: $n$
2. randomly choose $a, 1 < a < n-1$
3. **if** $a^{n-1} \bmod n \neq 1$ **then output** 'Composite' (* με βεβαιότητα *)
4. **else output** 'Prime' (* ορθό με μεγάλη πιθανότητα εκτός αν $n$ αριθμός Carmichael *)

Ο αλγόριθμος συμπεριφέρεται ως εξής: Αν ο $n$ είναι πρώτος, τότε η ισότητα ισχύει και πάντα το τεστ θα δώσει σίγουρα τη σωστή απάντηση. Αν ο $n$ είναι σύνθετος, τότε αποδεικνύεται ότι για τουλάχιστον τα μισά $a$ η ισότητα δεν ισχύει, οπότε θα πάρουμε τη σωστή απάντηση με πιθανότητα $\geq 1/2$. Για να αυξήσουμε σημαντικά την πιθανότητα μπορούμε να επαναλάβουμε μερικές φορές (τυπικά 30-40 φορές) με διαφορετικό $a$. Αν όλες τις φορές βρεθεί να ισχύει η παραπάνω ισότητα τότε λέμε ότι το $n$ "περνάει το test" και ανακηρύσσουμε το $n$ πρώτο αριθμό°αν έστω και μία φορά αποτύχει ο έλεγχος, τότε ο αριθμός είναι σύνθετος.

*Σημείωση*: Υπάρχουν (λίγοι) σύνθετοι που έχουν την ιδιότητα να περνούν τον έλεγχο Fermat για κάθε $a$ που είναι σχετικά πρώτο με το $n$, οπότε για αυτούς το test θα αποτύχει όσες δοκιμές και αν γίνουν (εκτός αν πετύχει κανείς $a$ που δεν είναι σχετικά πρώτο με το $n$, πράγμα πολύ απίθανο). Αυτοί οι αριθμοί λέγονται *Carmichael*.

Ένας έλεγχος πρώτων αριθμών που δεν καλύπτει και την περίπτωση που ο $n$ είναι αριθμός Carmichael, είναι ο έλεγχος Miller-Rabin που αποτελεί βελτίωση του ελέγχου του Fermat και δίνει σωστή απάντηση με πιθανότητα τουλάχιστον 1/2 για *κάθε* φυσικό αριθμό (οπότε με π.χ. 40 επαναλήψεις έχουμε αμελητέα πιθανότητα λάθους για κάθε αριθμό εισόδου). Ο έλεγχος Miller-Rabin περιγράφεται παρακάτω:

**Έλεγχος (test) Miller-Rabin**:

1. Input: $n$
2. **repeat** $k$ times (* παράμετρος ακριβείας, τυπική τιμή $k = 40$ *)
3. randomly choose $a, 1 < a < n - 1$
4. **if** $a^{n-1} \bmod n \neq 1$ **then output** 'Composite' (* με βεβαιότητα *)
5. **else** compute $s, t : n - 1 = 2^s \cdot t;\ y := a^t;\ z := y^2 \bmod n$
6. **while** $(y \neq \pm 1 \pmod n \wedge z \neq \pm 1 \pmod n)$ **do**
7. $y := z;\ z := y^2 \bmod n$
8. **end while**
9. **if** $y \equiv -1 \pmod n \wedge z \equiv 1 \pmod n$ **then output** 'Composite' (* με βεβαιότητα *)
10. **end repeat**
11. **output** 'Prime' (* ορθό με πολύ μεγάλη πιθανότητα *)

Η ορθότητα του ελέγχου βασίζεται στο ότι αν $a^{n-1} \bmod n \neq 1$ τότε ο αριθμός είναι οπωσδήποτε σύνθετος, λόγω του (μικρού) Θεωρήματος του Fermat. Αν όμως $a^{n-1} \bmod n = 1$, τότε αν γράψουμε $n - 1 = 2^s \cdot t$, με $t$ περιττό, η ακολουθία των αριθμών

$$a^t, a^{2t}, \ldots, a^{2^i t}, \ldots a^{2^s t}$$

που προκύπτει με διαδοχικούς τετραγωνισμούς, εμφανίζει σίγουρα σε κάποια θέση ισοτιμία με $1 \pmod n$, δηλαδή $\exists i, 0 \leq i \leq s, a^{2^i t} \bmod n = 1$ (γιατί;). Αν για τον προηγούμενο αριθμό στη σειρά, δηλαδή τον $x = a^{2^{i-1} t}$, ισχύει $x \bmod n \neq \pm 1$, τότε έχουμε βρεί μια μη τετριμμένη τετραγωνική ρίζα της μονάδας (modulo $n$), κάτι που όπως μπορεί να αποδειχθεί οδηγεί άμεσα σε παραγοντοποίηση του $n$: $\mathbf{gcd}(n, x - 1)$ και $\mathbf{gcd}(n, x + 1)$ είναι παράγοντες του αριθμού (άσκηση: αποδείξτε γιατί).

Αποδεικνύεται με χρήση θεωρίας ομάδων ότι, αν ο $n$ είναι σύνθετος, τουλάχιστον οι μισές επιλογές για το $a$ παράγουν τέτοιες *ακολουθίες παραγοντοποίησης*. Επομένως η πιθανότητα λάθος απάντησης είναι το πολύ $\frac{1}{2}$. Επαναλαμβάνοντας $k$ φορές μειώνουμε την πιθανότητα αυτή στο $\frac{1}{2^k}$.

**Factoring (παραγοντοποίηση σε πρώτους αριθμούς).** Το πρόβλημα Factoring ορίζεται ως εξής: "Δίνεται ακέραιος $n$. Να βρεθούν οι πρώτοι παράγοντές του." Δεν ξέρουμε αν είναι εύκολο ή δύσκολο. Πιστεύουμε ότι δεν είναι στο **P** (άρα μάλλον δύσκολο), αλλά δεν είναι τόσο δύσκολο όσο τα **NP**-πλήρη προβλήματα. Για κβαντικούς υπολογιστές (που δεν έχουμε καταφέρει ακόμα να κατασκευάσουμε) ανήκει στο **P**. Παρ'όλα αυτά, θεωρείται από όλους τους ειδικούς ότι έχει αρκετή δυσκολία ώστε να μην είναι πιθανός ένας πρακτικός αλγόριθμος παραγοντοποίησης ενός αριθμού που διαθέτει χιλιάδες ψηφία.

**Factoring και κρυπτοσύστημα RSA.** Το RSA είναι ένα κρυπτογραφικό σχήμα δημοσίου κλειδιού για να στείλει η Α (Alice) στον Β (Bob) ένα μήνυμα $m$ (κωδικοποιημένο κείμενο). Δηλαδή, ο Β έχει δύο κλειδιά, ένα δημόσιο, που μπορεί να το γνωρίζει ο καθένας και ένα ιδιωτικό που το ξέρει μόνο αυτός και το χρησιμοποιεί για την αποκρυπτογράφηση.

- Ο Β βρίσκει 2 μεγάλους πρώτους αριθμούς $p$ και $q$, υπολογίζει το γινόμενο $n = pq$, και ακέραιο $e$ σχετικά πρώτο με το $\phi(n) = (p-1)(q-1)$[3]. Ο $e$ μαζί με τον $n$ αποτελούν το *δημόσιο κλειδί* του Β.

- Ο Β στέλνει στην Α τα $n$ και $e$. Ή μπορεί να τα έχει μονίμως δημοσιοποιημένα για να τα χρησιμοποιεί οποιοσδήποτε θέλει να του στείλει κρυπτογραφημένο κείμενο.

- Η Α στέλνει στον Β τον αριθμό $c = m^e \bmod n$. Ο $c$ είναι η κρυπτογράφηση του $m$.

- Ο Β υπολογίζει το $m$: $m = c^d \bmod n$,
όπου το $d \equiv e^{-1} \pmod{\phi(n)}$, δηλαδή $d \cdot e \bmod \phi(n) = 1$. Το $d$ είναι το *ιδιωτικό κλειδί* του Β.

Αποδεικνύεται ότι αν $m < n$, τότε η αποκρυπτογράφηση είναι σωστή, δηλαδή στο τελευταίο βήμα ισχύει πράγματι ότι $m = c^d \bmod n$.

Παράδειγμα: $p = 11$, $q = 17$, $n = 187$, $e = 21$, $d = 61$, $m = 42$, $c = 9$.

Η ασφάλεια του RSA στηρίζεται στην (εκτιμώμενη) δυσκολία του Factoring: αν μπορούσαμε να κάνουμε παραγοντοποίηση γρήγορα, τότε θα υπολογίζαμε τα $p, q$, και στη συνέχεια τα $\phi(n), d$. Από την άλλη, το να υπολογίσει κάποιος τον ιδιωτικό εκθέτη $d$, έχοντας στη διάθεσή του μόνο τα $e$, $n$ (χωρίς δηλαδή να ξέρει τους παράγοντες $p, q$ του $n$) είναι πρακτικά το ίδιο δύσκολο με το να παραγοντοποιήσει το $n$, δηλαδή να λύσει το Factoring: υπάρχει αλγόριθμος που παραγοντοποιεί το $n$ με πολύ μεγάλη πιθανότητα, αν κάποιος γνωρίζει το $d$ (μαζί βέβαια με το $e$ που είναι δημόσιο).

*Σημείωση* 5.3. Παρ'όλα αυτά, δεν έχει αποδειχθεί με απόλυτη αυστηρότητα η υπολογιστική ισοδυναμία του "σπασίματος" του RSA με το Factoring. Συγκεκριμένα, είναι πολύ ενδιαφέρον ανοιχτό ερώτημα, κατά πόσον ένας πολυωνυμικού χρόνου αλγόριθμος που μπορεί να υπολογίσει το $m$, με είσοδο $n, e, c$, θα μπορούσε να δώσει έναν αλγόριθμο για την παραγοντοποίηση του $n$. Δεν γνωρίζουμε δηλαδή κάποια *αναγωγή* (βλ. και σε επόμενο κεφάλαιο) από το πρόβλημα Factoring στο πρόβλημα του "σπασίματος" του RSA, ενώ γνωρίζουμε αναγωγή για το αντίστροφο.

[3] Η $\phi(n)$ είναι γνωστή ως συνάρτηση του Euler και εκφράζει το πλήθος των αριθμών $< n$ που είναι σχετικά πρώτοι με τον $n$

**Υλοποίηση του RSA** Για την υλοποίηση του RSA χρησιμοποιούνται, μεταξύ άλλων: ένας έλεγχος πρώτων αριθμών (χρησιμεύει για την εύρεση των $p, q$), ο αλγόριθμος επαναλαμβανόμενου τετραγωνισμού (για την ύψωση σε δύναμη) και ο επεκτεταμένος Ευκλείδειος αλγόριθμος (που επιπλέον εκφράζει τον $\gcd(a, b)$ σαν γραμμικό συνδυασμό των $a$ και $b$°χρησιμεύει για την εύρεση του $d$ ως αντιστρόφου του $e$ modulo $\phi(n)$: $1 = ke + l\phi(n) \Rightarrow ke \bmod \phi(n) = 1$, οπότε ο $k$ είναι ο αντίστροφος του $e$ modulo $\phi(n)$, δηλαδή ο $d$).

Ιδιαίτερη προσοχή χρειάζεται στην επιλογή των παραμέτρων. Διάφορα κριτήρια έχουν προταθεί στη σχετική βιβλιογραφία προς αποφυγή περιπτώσεων που μπορούν να οδηγήσουν σε παραγοντοποίηση του $n$.

## 5.12 Διαδραστικό Υλικό – Σύνδεσμοι

- Διαδραστικό site οπτικοποιημένων αλγορίθμων και δομών δεδομένων: http://visualgo.net/
- Διαδραστικό site με animated παρουσίαση βασικών αλγορίθμων: http://www.algomation.com/
- Διαδραστικό site σύγκρισης αλγορίθμων ταξινόμησης με χρήση animation: http://www.sorting-algorithms.com/
- Ηλεκτρονικό σύγγραμμα για Foundations of Computer Science, με περιγραφή και ανάλυση πλήθους βασικών αλγορίθμων. Al Aho and Jeff Ullman, "Foundations of Computer Science": http://infolab.stanford.edu/~ullman/focs.html
- Ηλεκτρονικό σύγγραμμα για Αλγόριθμους, βασικούς και προχωρημένους. S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani, "Algorithms": http://code.google.com/p/eclipselu/downloads/detail?name=algorithms.pdf
- Λογισμικό LEDA για ανάπτυξη αλγορίθμων (η βασική έκδοση διατίθεται ελεύθερα): http://algorithmic-solutions.com/leda/ledak/index.htm

## 5.13 Ασκήσεις

1. Ασυμπτωτικός συμβολισμός (i):

   Βάλτε σε φθίνουσα σειρά (αν $f = O(g)$ η $f$ θα βρίσκεται δεξιά της $g$) τα ακόλουθα: $O(5\sqrt[15]{n})$, $O(n^2 \log n + n!)$, $O(\frac{\log^3 n}{100!})$, $O(5n^5 - 4n^4 + 1)$, $O(n^3 \log^{37} n)$, και $O(3^n)$.

2. Ασυμπτωτικός συμβολισμός (ii):

   Σχεδιάστε σε διάγραμμα Venn τα εξής σύνολα συναρτήσεων:
   $O(\frac{n^2}{\log(100!)})$, $O(2^n)$, $O(n^3 + 15n)$, $\Omega(2^{2 \log n})$, $O(\log \log n)$, $\Omega(1)$, $\Theta(n^2)$, $\Theta(n!)$, $O(n^{\frac{n}{2}})$.

3. Πύργοι Hanoi:

   Θεωρήστε την παραλλαγή του προβλήματος με 4 πασάλους. Περιγράψτε σε ψευδοκώδικα έναν αλγόριθμο για την μετακίνηση των δίσκων από τον πάσαλο 1 στον πάσαλο 4, με αριθμό κινήσεων σημαντικά μικρότερο από την λύση που χρησιμοποιεί 3 πασάλους μόνο. Εκφράστε τον αριθμό κινήσεων του αλγορίθμου σας σαν συνάρτηση του $n$.

   *Υπόδειξη: σκεφτείτε την μέθοδο "διαίρει και κυρίευε".*

4. Αριθμοί Fibonacci (i):

   Υλοποιήστε σε γλώσσα της επιλογής σας τους τρεις αλγορίθμους για υπολογισμό αριθμών Fibonacci και συγκρίνετέ τους πειραματικά (οι δύο ταχύτεροι αλγόριθμοι θα πρέπει να υπολογίζουν τουλάχιστον τον $10^6$-οστό όρο). Εξηγήστε τα αποτελέσματα των δοκιμών σας.

5. Αριθμοί Fibonacci (ii):

   Βρείτε την πολυπλοκότητα σε ψηφιοπράξεις (bit complexity) των δύο ταχύτερων αλγορίθμων για υπολογισμό Fibonacci (του επαναληπτικού και του αλγορίθμου με χρήση πίνακα). Τι παρατηρείτε; Πώς επηρεάζεται η πολυπλοκότητα αν χρησιμοποιήσουμε πιο αποδοτικό αλγόριθμο πολλαπλασιασμού (π.χ. Gauss-Karatsuba);.

   Τι συμβαίνει με την χωρική πολυπλοκότητα (κόστος σε χώρο μνήμης);

6. Υπολογισμός κυβικής ρίζας

   (α) Σχεδιάστε απλό αλγόριθμο που να υπολογίζει το ακέραιο μέρος της κυβικής ρίζας ενός φυσικού αριθμού $n$ με διαδοχικές δοκιμές – επιθυμητή πολυπλοκότητα $O(n^{1/3})$.

   (β) Βελτιώστε τον αλγόριθμό σας ώστε να γίνει σημαντικά ταχύτερος – επιθυμητή πολυπλοκότητα $O(\log n)$.

   (γ) Τροποποιήστε τον αλγόριθμο ώστε να υπολογίζει την κυβική ρίζα με ακρίβεια δεκαδικού ψηφίου που θα δίνεται ως παράμετρος (δηλαδή δίνεται επιπλέον φυσικός αριθμός $k$ και ζητείται η κυβική ρίζα του $n$ με ακρίβεια $k$-οστού δεκαδικού ψηφίου). Δώστε την πολυπλοκότητα του αλγορίθμου σας εκφρασμένη σαν συνάρτηση των $n$ και $k$.

   *Σημείωση:* για την πολυπλοκότητα θεωρήστε ότι οι βασικές αριθμητικές πράξεις (πολ/σμός, διαίρεση) κοστίζουν ένα βήμα υπολογισμού (αριθμητική πολυπλοκότητα).

7. Έλεγχος πρώτων αριθμών Fermat

   (α) Υλοποιήστε σε γλώσσα προγραμματισμού της επιλογής σας (που να υποστηρίζει υπολογισμούς με πολύ μεγάλους αριθμούς, εκατοντάδων ψηφίων) τους αλγορίθμους ύψωσης σε δύναμη που μάθαμε (τον απλό και του επαναλαμβανόμενου τετραγωνισμού). Τροποποιήστε τους ώστε να δέχονται και μία 3η παράμετρο, έστω $m$, και να βρίσκουν το αποτέλεσμα $a^n \bmod m$. Διαπιστώστε με δοκιμές μέχρι ποια περίπου δύναμη του 17 ($\bmod 2^{100}$) μπορείτε να υπολογίσετε με τον κάθε αλγόριθμο σε χρόνο 1 λεπτού. Σχολιάστε.

   (β) Χρησιμοποιήστε τη συνάρτησή σας για να ελέξγετε αν ένας αριθμός είναι πρώτος με τον έλεγχο (test) του Fermat:

Αν $n$ πρώτος τότε για κάθε $a$ τ.ώ. $1 < a < n - 1$, ισχύει

$$a^{n-1} \bmod n = 1$$

Ένας πρώτος αριθμός ικανοποιεί τη συνθήκη για οποιαδήποτε επιλογή του $a$ (περνάει πάντα τον έλεγχο Fermat). Όπως είπαμε ήδη, ισχύει (σχεδόν για κάθε αριθμό $n$) ότι αν ο $n$ είναι σύνθετος, τότε η παραπάνω συνθήκη δεν ικανοποιείται για τουλάχιστον τα μισά $a$ στο διάστημα $(0, n - 1]$. Επομένως ένας σύνθετος αριθμός, με πιθανότητα $\geq \frac{1}{2}$ δεν περνάει τον έλεγχο Fermat.

Πώς μπορούμε να αυξήσουμε την πιθανότητα ορθής απάντησης στην περίπτωση που ο αριθμός είναι σύνθετος; Χρησιμοποιήστε τις παραπάνω ιδέες για να ελέγξετε αν $n$ πρώτος για $n = 2^i - 1, i \in \{100, 200, \ldots, 1000\}$.

8. Έλεγχος πρώτων αριθμών Miller-Rabin

   Επεκτείνετε τη συνάρτηση της προηγούμενης άσκησης ώστε να υλοποιήσετε τον έλεγχο Miller-Rabin. Δοκιμάστε τη συνάρτησή σας με είσοδο αριθμούς Carmichael: http://en.wikipedia.org/wiki/Carmichael_number

# Βιβλιογραφία


[1] S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani. “Algorithms”, MacGraw-Hill, 2006. “Αλγόριθμοι”, ελληνική έκδοση, Εκδόσεις Κλειδάριθμος, 2008.

[2] Thomas Cormen, Charles Leiserson, Ronald Rivest and Cliff Stein: “Introduction to Algorithms”, 3rd edition, MIT Press, 2009. Ελληνική έκδοση: “Εισαγωγή στους Αλγόριθμους”, Πανεπιστημιακές Εκδόσεις Κρήτης, 2012.

[3] D. Harel: “Algorithmics: The Spirit of Computing”, Addison-Wesley, Reading, MA, 1st edition, 1987; 2nd edition, 1992. 3rd edition (with Y. Feldman), 2004.

[4] A.V. Aho, J.E. Hopcroft, and J.D. Ullman: “The Design and Analysis of Computer Algorithms”, Addison-Wesley Series in Computer Science andInformation Processing, 1974.

[5] A. Levitin: “Ανάλυση και Σχεδίαση Αλγορίθμων”, Εκδόσεις Τζιόλα, 2007.

[6] H.R. Lewis and C.H. Papadimitriou: “Elements of the Theory of Computation”, 2nd edition, Prentice Hall, 1997. Μεταφρασμένη έκδοση: ”Στοιχεία Θεωρίας Υπολογισμού”, Εκδόσεις Κριτική, 2005.

[7] Ronald Graham, Donald Knuth, Oren Patashnik. “Συνκριτά Μαθηματικά” (δεύτερη έκδοση, μτφ. Χ. Καπούτσης, επιμ. Ε. Ζάχος) Εκδόσεις Κλειδάριθμος, 2011.

[8] M. Sipser: “Introduction to the Theory of Computation”, 2nd edition, Course Technology, 2005. Μεταφρασμένη έκδοση: “Εισαγωγή στη Θεωρία Υπολογισμού”, Παν. Εκδόσεις Κρήτης, 2007.

# Κεφάλαιο 6

# Αλγόριθμοι Γράφων

## 6.1 Διάσχιση γράφων

### 6.1.1 Γενικά

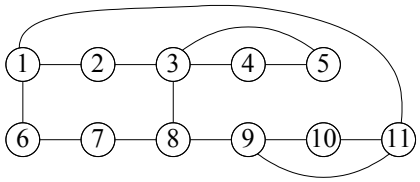
Οι τεχνικές διάσχισης γράφων μας βοηθούν στο να επισκεπτόμαστε συστηματικά τους κόμβους ενός γράφου G(V,E) έτσι ώστε να δίνουμε γρήγορα απαντήσεις σε προβλήματα όπως τα παρακάτω (*G.A.P., Graph Accessibility Problem*):

- Στο γράφο $G$ υπάρχει μονοπάτι από τον κόμβο v στον κόμβο u;
- Ο γράφος $G$ είναι ακυκλικός;
- Ποιες είναι οι συνεκτικές συνιστώσες (*strong components*) του γράφου $G$; Δηλαδή ζητάμε να βρεθούν όλα τα υποσύνολα του συνόλου $V$ που είναι τέτοια ώστε όλοι οι κόμβοι που ανήκουν σε αυτά να είναι μεταξύ τους συνδεδεμένοι.
- Ποια είναι τα σημεία σύνδεσης (*articulation points*) του γράφου $G$; Δηλαδή ζητάμε όλους εκείνους τους κόμβους του γράφου, που αν αφαιρεθούν μαζί με τις προσπίπτουσες ακμές τους, χωρίζουν το γράφο σε δύο ή περισσότερες συνεκτικές συνιστώσες.

### 6.1.2 Αναζήτηση κατά πλάτος (Breadth First Search)

Στην αναζήτηση κατά πλάτος από κάθε κόμβο $v$ που επισκεπτόμαστε, αναζητούμε κόμβους όσο το δυνατόν κατά πλάτος, δηλαδή αμέσως μετά την επίσκεψη του κόμβου $v$ επισκεπτόμαστε όλους τους γειτονικούς του κόμβους. Έτσι λοιπόν, κατά τη διάρκεια της διαδικασίας αυτής μπορούμε να ξεχωρίσουμε δύο κατηγορίες κόμβων:

- κόμβους που έχουμε επισκεφθεί (*visited nodes*)
- κόμβους που έχουμε επισκεφθεί αυτούς και όλους τους γειτονικούς τους (*explored nodes*)




Σχήμα 6.1: Παράδειγμα γράφου στον οποίο θα γίνει αναζήτηση

Μετά το τέλος της διαδικασίας της αναζήτησης κατά πλάτος στο γράφο $G$, προκύπτει το συνδετικό δέντρο με προτεραιότητα πλάτους (*breadth first spanning tree*) του γράφου $G$. Η υλοποίηση της διαδικασίας φαίνεται στον αλγόριθμο 6.1.

**Αλγόριθμος 6.1** Αναζήτηση κατά πλάτος

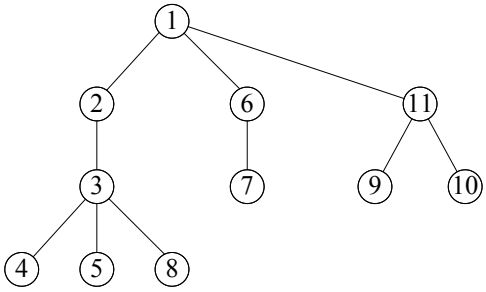
```
procedure bfs(v:vertex);
begin
  initialize queue with v; visited[v]:=true;
  repeat dequeue(u);
    for all vertices w adjacent to u do
      if not visited[w] then
        begin visited[w] := true; enqueue(w) end
  until queue is empty
end
```

*Παράδειγμα* 6.1. Έστω ο γράφος που φαίνεται στο σχήμα 6.1. Αν καλέσουμε τη διαδικασία **bfs** με **bfs**(1) έχουμε τα βήματα που φαίνονται στο πίνακα 6.1. Το συνδετικό δέντρο με προτεραιότητα πλάτους, είναι αυτό που φαίνεται στο σχήμα 6.2. Οποιαδήποτε υλοποίηση και να χρησιμοποιηθεί, ο αλγόριθμος 6.1 χρειάζεται τον επιπλέον χώρο μιας ουράς μήκους $n$ και ένα πίνακα επίσης μήκους $n$. Συνεπώς ο χώρος που χρειάζεται είναι της τάξης $O(n)$. Αν ο γράφος υλοποιηθεί με πίνακα γειτονίας, ο χρόνος που χρειάζεται για να διατρέξουμε τον πίνακα είναι $O(n^2)$, ενώ αν έχουμε λίστες γειτονικών κορυφών ο χρόνος είναι $O(n + e)$.

Για να βρούμε όλες τις συνεκτικές συνιστώσες του γράφου $G$ διασχίζουμε το γράφο με προτεραιότητα πλάτους (*breadth first traversing*). Η υλοποίηση αυτής της διάσχισης φαίνεται στον αλγόριθμο 6.2. Στον αλγόριθμο αυτό είναι φανερό ότι αν ο γράφος είναι συνεκτικός, τότε με την πρώτη κλήση της bfs θα επισκεφθούμε όλους τους κόμβους του.

### 6.1.3 Αναζήτηση κατά βάθος (Depth First Search)

Στην αναζήτηση κατά βάθος από κάθε κόμβο $v$ που επισκεπτόμαστε αναζητάμε άλλους κόμβους όσο το δυνατό βαθύτερα. Μετά το τέλος της διαδικασίας προκύπτει το συνδετικό δέντρο με προτεραιότητα βάθους (*depth first spanning tree*). Η υλοποίηση της διαδικασίας φαίνεται στον αλγόριθμο 6.3, ενώ το συνδετικό δέντρο με προτεραιότητα βάθους είναι αυτό που φαίνεται στο σχήμα 6.3.




Σχήμα 6.2: Αναζήτηση κατά πλάτος στον γράφο

| visited nodes | Queue |
|---|---|
| 1 | 1 |
| 2 | 2 |
| 6 | 2-6 |
| 11 | 2-6-11 |
| 3 | 6-11-3 |
| 7 | 11-3-7 |
| 9 | 3-7-9 |
| 10 | 3-7-9-10 |
| 4 | 7-9-10-4 |
| 5 | 7-9-10-4-5 |
| 8 | 7-9-10-4-5-8 |
| - | 9-10-4-5-8 |
| - | 10-4-5-8 |
| - | 4-5-8 |
| - | 5-8 |
| - | 8 |
| - | Ø |

Πίνακας 6.1: Εκτέλεση αναζήτησης κατά πλάτος

Ο Αλγόριθμος **dft**, που βρίσκει όλες τις συνεκτικές συνιστώσες ενός γράφου, είναι ίδιος με τον **bft**, με την διαφορά ότι αντικαθιστούμε την κλήση της **bfs** με την κλήση της **dfs**.

**D-Search** Ένας ακόμη αλγόριθμος διάσχισης είναι ο **D-search** που περιγράφεται ακριβώς όπως ο **bfs** εκτός από το ότι χρησιμοποιεί μια στοίβα (*stack*) αντί ουράς. Έτσι συμπεριφέρεται σαν ένα κράμα **bfs** και **dfs** (θυμηθείτε σε αυτό το σημείο ότι ο **dfs** μπορεί να υλοποιηθεί και με επαναληπτικό αλγόριθμο, που χρησιμοποιεί στοίβα).

## 6.2 Το πρόβλημα των συντομότερων μονοπατιών, single source shortest paths problem

Μας δίνεται ένας κατευθυνόμενος γράφος με (θετικά) βάρη και ένας κόμβος του σαν πηγή (*source*). Ζητάμε να βρούμε τα συντομότερα μονοπάτια από την πηγή προς όλες τις άλλες κορυφές του γράφου. Το πρόβλημα της εύρεσης του συντομότερου μονοπατιού από την πηγή προς ένα συγκεκριμένο κόμβο του γράφου είναι εξίσου δύσκολο.

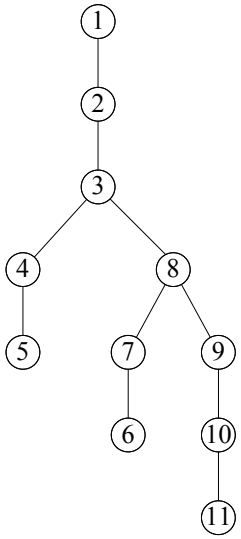
Ο αλγόριθμος που θα περιγράψουμε στηρίζεται στη μέθοδο που αναπτύχθηκε από τον *Dijkstra* (1959). Αριθμούμε τις κορυφές του γράφου και σαν πηγή λαμβάνουμε την κορυφή 1. Έστω $V$ το σύνολο των κορυφών του γράφου και $S$ το σύνολο των κορυφών για τις οποίες το συντομότερο μονοπάτι από την πηγή είναι ήδη γνωστό. Χρησιμοποιούμε τους πίνακες $C, D, P$. Με $C[i, j]$ συμβολίζουμε το κόστος μετάβασης από την κορυφή $i$ στην κορυφή $j$, δηλαδή το κόστος μετάβασης της ακμής $i \rightarrow j$. Αν η ακμή $i \rightarrow j$ δεν υπάρχει, τότε $C[i, j] = \infty$, δηλαδή μια τιμή μεγαλύτερη από οποιοδήποτε πραγματικό κόστος. Το $D[v]$ περιέχει σε κάθε βήμα το κό-

**Αλγόριθμος 6.2** Εύρεση συνεκτικών συνιστωσών

```
procedure bft(G);
begin
  initialize visited with false;
  for i:=1 to n do
    if not visited[i] then bfs(i)
end
```

**Αλγόριθμος 6.3** Αναζήτηση κατά βάθος

```
procedure dfs(v:vertex);
begin
  visited[v]:=true;
  for all vertices u adjacent to v do
    if not visited[u] then dfs(u)
end
```




Σχήμα 6.3: Αναζήτηση κατά βάθος στον γράφο

στος του τρέχοντος συντομότερου από την πηγή στον κόμβο $v$, ενώ το $P[v]$ περιέχει την αμέσως προηγούμενη κορυφή από την $v$ στο συντομότερο μονοπάτι. Ο αλγόριθμος ξεκινάει με $S = 1$ και προχωράει σε βήματα κατά τα οποία επιλέγει μια κορυφή $w$ έτσι ώστε το $D[w]$ να είναι το ελάχιστο. Στη συνέχεια, αφαιρεί την κορυφή $w$ από το $V$ και την προσθέτει στο $S$. Έπειτα επαναϋπολογίζει τον πίνακα $D$ για κάθε κορυφή $v$ που ανήκει στο $V - S$ ως εξής:

$$D[v] := min(D[v], D[w] + C[w, v])$$

Αν ισχύει ότι $D[v] > D[w]+C[w, v]$ τότε θέτει $P[v] = w$. Ο αλγόριθμος σταματά όταν $V-S = \emptyset$. Η υλοποίηση της μεθόδου του *Dijkstra* φαίνεται στον αλγόριθμο 6.4.

Από τον αλγόριθμο 6.4 φαίνεται ότι αρχικά το σύνολο $V - S$ έχει $n - 1$ στοιχεία άρα για να βρεθεί το ελάχιστο θα πρέπει να γίνουν $n - 2$ συγκρίσεις. Ακολούθως το $V - S$ θα έχει $n - 2$ στοιχεία άρα θα χρειάζονται $n - 3$ συγκρίσεις κ.ο.κ. Συνολικά λοιπόν οι συγκρίσεις είναι:

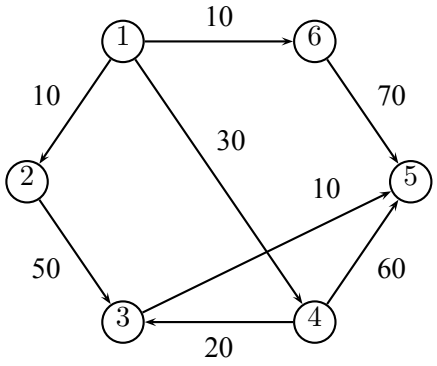
$$\sum_{i=1}^{n-2} i = \frac{(n-1)(n-2)}{2} = O(n^2)$$

*Παράδειγμα* 6.2. Έστω ότι μας δίνεται ο γράφος που φαίνεται στο σχήμα 6.4. Η εκτέλεση του αλγορίθμου φαίνεται στον πίνακα 6.2. Για να τυπώσουμε το συντομότερο μονοπάτι από ένα κόμβο σε ένα άλλο ανιχνεύουμε τους προκατόχους του τελευταίου κόμβου χρησιμοποιώντας τον πίνακα $P$. Αν για παράδειγμα θέλουμε το συντομότερο μονοπάτι από τον κόμβο 1 στον κόμβο 5 βλέπουμε ότι $P[5] = 3$, δηλαδή ο 3 είναι ο προκάτοχος του 5, $P[3] = 4$ δηλαδή ο 4 είναι προκάτοχος του 3. 'Αρα το συντομότερο μονοπάτι από τον κόμβο 1 στον 5 είναι το $1 \to 4 \to 3 \to 5$.

**Αλγόριθμος 6.4** Εύρεση συντομότερων μονοπατιών (single source shortest paths) με τη μεθόδο του Dijkstra

```
procedure Dijkstra;
begin (* Αρχικοποίηση*)
  for i:=2 to n do begin D[i]:=cost[1,i]; P[i]:=1 end;
  S := {1};

  for i:=2 to n − 1 do
  begin
    Select w from V − S such that D[w] is minimum;
    S := S + {w};
    for all v in V − S do if D[v] > D[w] + C[w,v] then
        begin P[v] := w; D[v] := D[w] + C[w,v] end
  end
end
```




Σχήμα 6.4: Κατευθυνόμενος γράφος με βάρη

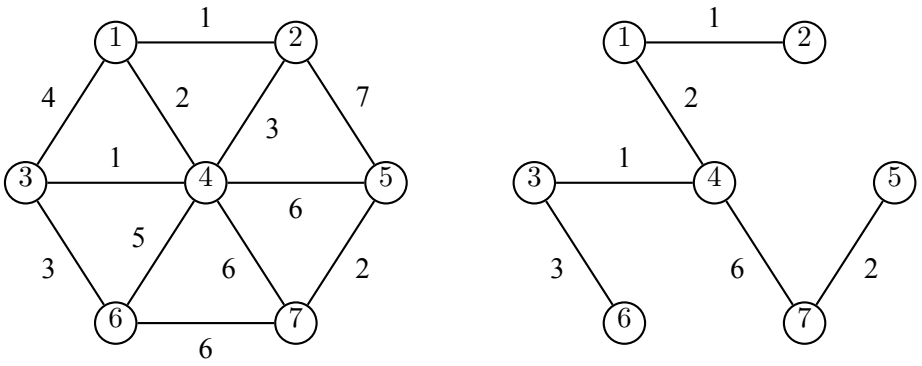
## 6.3 Ελάχιστο συνδετικό δέντρο (MST)

Στο πρόβλημα αυτό έχουμε ένα συνεκτικό γράφο $G(V, E)$ και ένα πίνακα κόστους $c_{|V|\times|V|}$ έτσι ώστε για κάθε ακμή $(u, v) \in E$ να υπάρχει το αντίστοιχο (βάρος) κόστος $c[v, u] > 0$. Ζητάμε ένα δέντρο $T(V, E')$ για το οποίο ισχύει:

$$\begin{cases} E' & \subseteq E \\ \& & \sum_{(u,v)\in E'} c[u,v] = MINIMUM \end{cases}$$

δηλαδή ένα δέντρο-υπογράφο που συνδέει όλες τις κορυφές του $V$ και το συνολικό κόστος των ακμών του είναι το ελάχιστο δυνατό. Το δέντρο αυτό λέγεται ελαχίστου κόστους συνδετικό δέντρο του γράφου $G$ (σχήμα 6.5).

Τα ελάχιστα συνδετικά δέντρα βρίσκουν εφαρμογή στο σχεδιασμό τηλεπικοινωνιακών δικτύων. Οι κόμβοι του γράφου αναπαριστούν πόλεις και οι ακμές αναπαριστούν πιθανούς τηλεπικοινωνιακούς συνδέσμους μεταξύ των πόλεων. Το κόστος κάθε ακμής είναι το κόστος κατασκευής του




Σχήμα 6.5: Ο γράφος G και ένα ελαχίστου κόστους συνδετικό δένδρο αυτού

συγκεκριμένου συνδέσμου για το τελικό δίκτυο. Το ελάχιστο συνδετικό δέντρο αναπαριστά ένα τηλεπικοινωνιακό δίκτυο που ενώνει όλες τις πόλεις και έχει το ελάχιστο κόστος κατασκευής. Η άπληστη μέθοδος αρχίζει με την κενή λύση και κατασκευάζει το δέντρο ακμή-ακμή (*edge-by-edge*) ακολουθώντας είτε το κριτήριο του *Prim* είτε το κριτήριο του *Kruskal*, είτε άλλα κριτήρια.

**Κριτήριο του Prim:** Προσθέτουμε κάθε φορά την ακμή που έχει το ελάχιστο κόστος έτσι ώστε ο νέος υπογράφος να παραμένει δέντρο. Αρχικοποιούμε με ένα δέντρο που αποτελείται από έναν μόνο κόμβο (π.χ. τον κόμβο 1). Η υλοποίηση που χρησιμοποιεί αυτό το κριτήριο φαίνεται στον αλγόριθμο 6.5 ενώ στο σχήμα 6.6 φαίνεται η ακολουθία των ακμών που προστίθενται στο ελάχιστο συνδετικό δέντρο μέχρι αυτό να πάρει την τελική μορφή του.

**Κριτήριο του Kruskal:** Προσθέτουμε κάθε φορά την ακμή ελαχίστου κόστους έτσι ώστε ο νέος υπογράφος να μην έχει κύκλους. Σχηματίζουμε έτσι ένα δάσος το οποίο τελικά γίνεται δέντρο. Η υλοποίηση που χρησιμοποιεί αυτό το κριτήριο φαίνεται στον αλγόριθμο 6.6 ενώ στο σχήμα 6.7 φαίνεται η ακολουθία των ακμών που προστίθενται στο ελάχιστο συνδετικό δέντρο μέχρι αυτό να πάρει την τελική μορφή του.

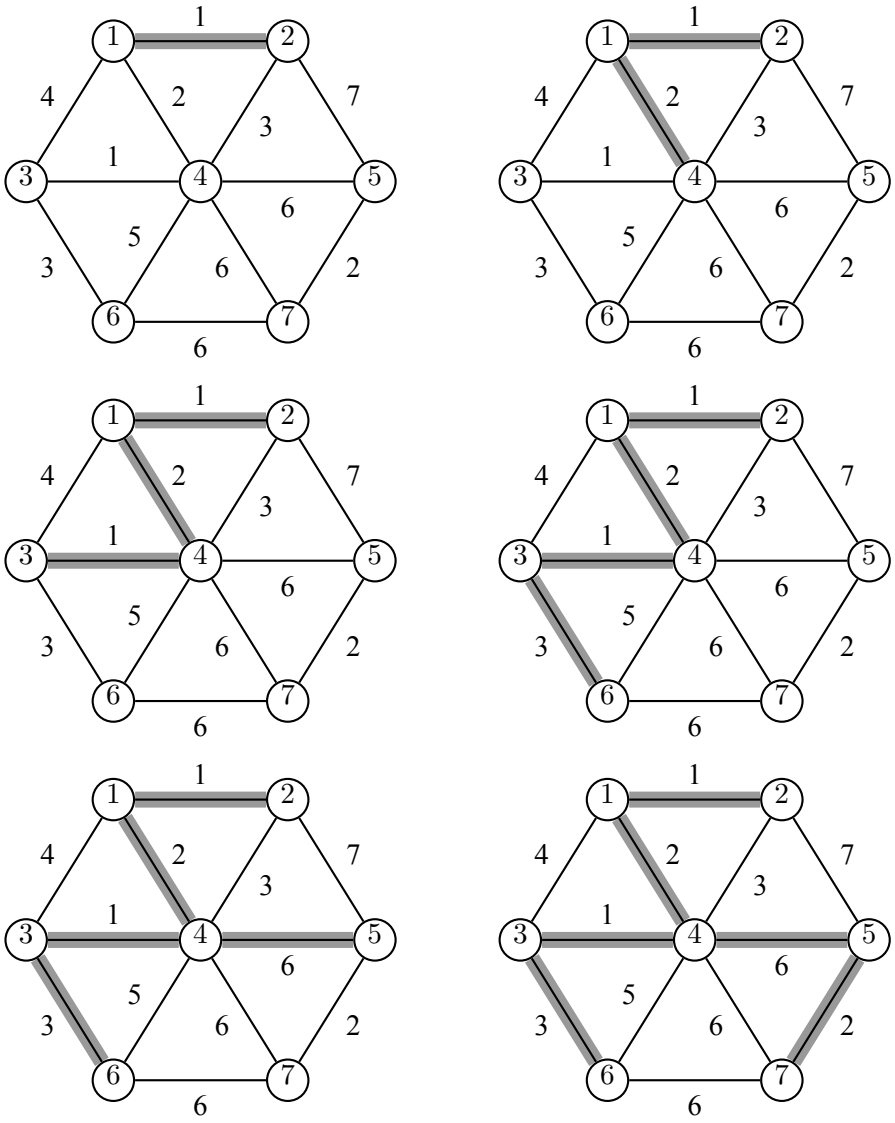
Αν $n = |V|$ και $e = |E|$ τότε ο αλγόριθμος 6.5 (με το κριτήριο του Prim) έχει πολυπλοκότητα $O(n^2)$ ενώ ο αλγόριθμος 6.6 (με το κριτήριο του Kruskal) έχει πολυπλοκότητα $O(e \log e)$. Για

| Βήμα | $S$ | $w$ | $D$ | | | | | $P$ | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 2 | 3 | 4 | 5 | 6 | 2 | 3 | 4 | 5 | 6 |
| - | {1} | - | 10 | $\infty$ | 30 | $\infty$ | 10 | 1 | 1 | 1 | 1 | 1 |
| 2 | {1,2} | 2 | | 60 | 30 | $\infty$ | 10 | | 2 | | | |
| 3 | {1,2,6} | 6 | | 60 | 30 | 80 | | | | | 6 | |
| 4 | {1,2,6,4} | 4 | | 50 | | 80 | | | 4 | | | |
| 5 | {1,2,6,4,3} | 3 | | | | 60 | | | | | 3 | |
| 6 | {1,2,6,4,3,5} | 5 | | | | | | | | | | |

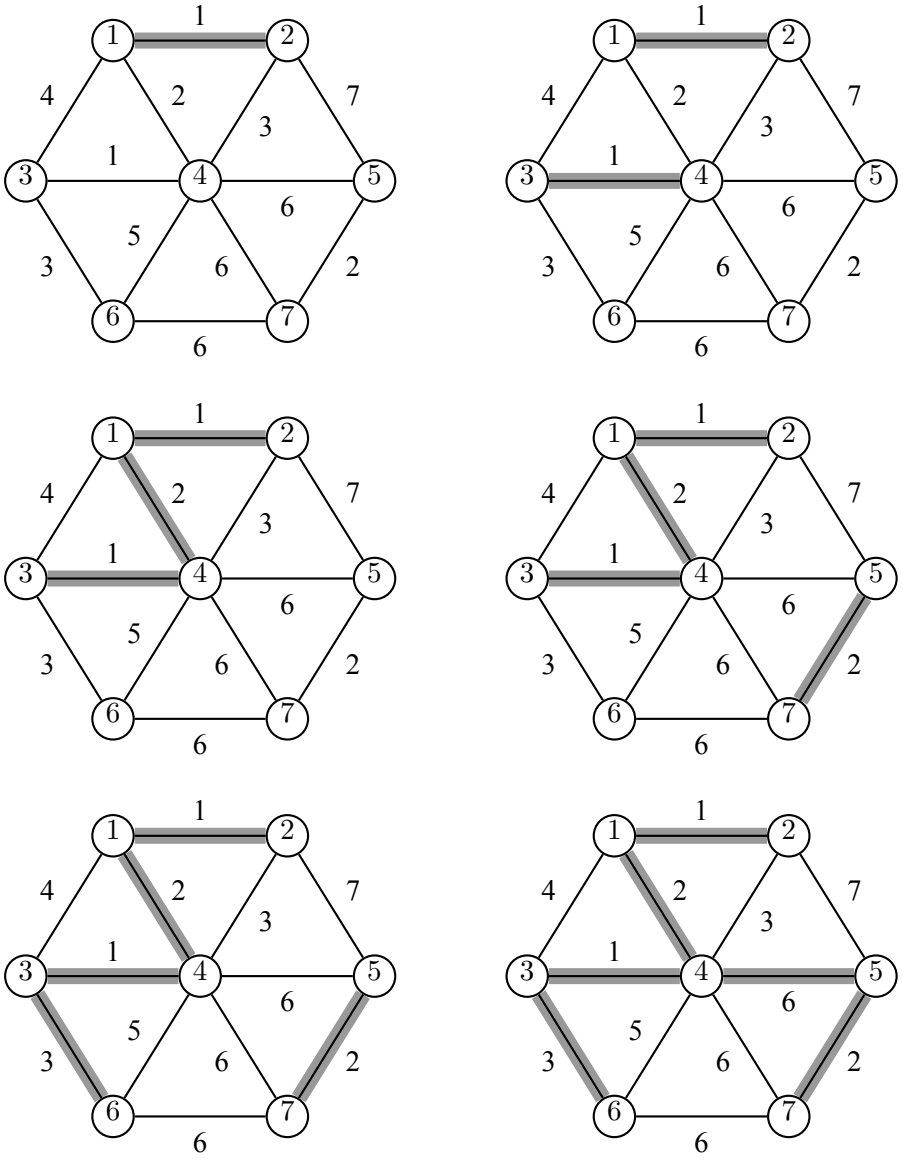
Πίνακας 6.2: Εφαρμογή της μεθόδου του Dijkstra για το γράφο G

**Αλγόριθμος 6.5** Άπληστη μέθοδος εύρεσης ελαχίστου κόστους συνδετικού δέντρου (minimum cost spanning tree) με το κριτήριο του Prim

```
procedure Prim (Input: graph G(V, E) with costs on edges;
                Output: MST T of minimum total cost);
  var DistFromTree: array[1..n] of real;
      Edge: array[1..n] of edges; i, j, k, l: integer;
  (* DistFromTree[v] : το ελάχιστο από τα κόστη των ακμών
   που συνδέουν τον κόμβο v με το μέχρι στιγμής κατασκευασμένο
   συνδετικό δέντρο *)
  (* Edge[v] : η ακμή με αυτό το κόστος *)
begin
  (* αρχικοποίηση με βάση τον κόμβο 1 *)
  V_T := {1}
  for v:=1 to n do
  begin
    DistFromTree[v]:=cost[(1, v)]; Edge[v] := (1, v)   (* ενημέρωση απόστασης από δέντρο *)
  end;
  for i:=1 to n − 1 do
  begin
    select v ∈ V \ V_T such that DistFromTree[v] is minimum;
    V_T := V_T ∪ {v}; T := T ∪ Edge[v];
    for u := 1 to n do
    begin
      if cost[(v, u)] < DistFromTree[u] then
      begin
        DistFromTree[u]:=cost[(v, u)]; Edge[u]:=(v, u)
      end
    end
  end
end
```




Σχήμα 6.6: Εφαρμογή του κριτηρίου του Prim για την εύρεση ελαχίστου κόστους συνδετικού δένδρου




Σχήμα 6.7: Εφαρμογή του κριτηρίου του Kruskal για την εύρεση ελαχίστου κόστους συνδετικού δένδρου

**Αλγόριθμος 6.6** Απληστη μέθοδος εύρεσης ελαχίστου κόστους συνδετικού δέντρου (minimum cost spanning tree) με το κριτήριο του Kruskal

```
procedure Kruskal (Input: graph G(V, E) with costs on edges;
                   Output: MST T of minimum total cost);
                   (* θεωρεί ακμές ταξινομημένες κατά βάρος *)
begin
  forest := ∅;
  while |forest| < n − 1 do
  begin
    select minimum cost edge (u, w) and delete it from E;
    if (u, w) does not create a cycle in forest then
      add it to forest
    else discard it
  end
end
```

συνεκτικούς γράφους ισχύει:

$$n - 1 \leq e \leq \frac{n(n-1)}{2}$$

Βλέπουμε λοιπόν ότι ο αλγόριθμος του *Kruskal* έχει καλύτερη απόδοση στους αραιούς (*sparse*) γράφους όπου η πολυπλοκότητά του είναι $O(n \log n)$, σε αντίθεση με τον αλγόριθμο του *Prim* που έχει πολυπλοκότητα $O(n^2)$.

## 6.4 Μέγιστη ροή – Ελάχιστη τομή (Max Flow – Min Cut)

Ένα πρόβλημα που, όπως θα δούμε, ανάγεται στο πρόβλημα της διάσχισης είναι το *Πρόβλημα Μέγιστης Ροής* (Max Flow Problem): Δοθέντος ενός *δικτύου*, δηλαδή ενός κατευθυνόμενου γράφου με βάρη που αντιπροσωπεύουν *χωρητικότητες* και δύο κόμβων $s$ (source, πηγή), $t$ (target, στόχος), ζητείται να δρομολογηθεί όσο το δυνατόν μεγαλύτερη ροή από τον $s$ στον $t$ έτσι ώστε να ισχύει η αρχή διατήρησης της ροής και να μην παραβιάζονται οι χωρητικότητες των ακμών.

Ας σημειωθεί ότι η *ροή* (όπως και η χωρητικότητα) ορίζεται τυπικά ως μια συνάρτηση πάνω στις ακμές που παίρνει μη αρνητικές πραγματικές τιμές $f : E \to \mathbf{R}_+$ (για την χωρητικότητα συμβολίζουμε με $c : E \to \mathbf{R}_+$).

Η αρχή διατήρησης της ροής απαιτεί σε κάθε κόμβο εκτός των $s, t$ το άθροισμα της $f$ στις εισερχόμενες ακμές να είναι ίδιο με το άθροισμα της $f$ στις εξερχόμενες ακμές (πρβλ. νόμο Kirchhoff). Η *τιμή* της ροής ορίζεται ως η καθαρή ροή που εξέρχεται από τον κόμβο $s$ (ή ισοδύναμα, που εισέρχεται στον κόμβο $t$), δηλαδή το άθροισμα της $f$ στις εξερχόμενες ακμές του κόμβου $s$ μείον το άθροισμα της $f$ στις εισερχόμενες ακμές του κόμβου $s$.[1]

[1] Στη βιβλιογραφία χρησιμοποιούνται και ορισμοί του προβλήματος που επιτρέπουν αρνητική ροή, ή/και απαιτούν διατήρηση της ροής και στους κόμβους $s, t$ (με προσθήκη μιας ακμής $(t, s)$ εάν δεν υπάρχει). Μπορεί να δειχθεί ότι

Ισχύει το παρακάτω σημαντικό θεώρημα:

*Θεώρημα* 6.3. *(Max Flow – Min Cut) Η μέγιστη ροή ισούται με την ελάχιστη (ως προς χωρητικότητα) τομή (σύνολο ακμών) που διαχωρίζει τον $s$ από τον $t$.*

Η απόδειξη του θεωρήματος αυτού, μαζί με τον αλγόριθμο που υπολογίζει τη μέγιστη ροή, δόθηκε από τους Ford και Fulkerson(1962).

---

**Αλγόριθμος 6.7** Αλγόριθμος Ford-Fulkerson

---

Επαναλαμβάνονται τα παρακάτω βήματα στο δίκτυο για όσο υπάρχει μονοπάτι από τον $s$ στον $t$ (μετά την 1η επανάληψη χρησιμοποιείται το παραμένον δίκτυο):

1. Επιλογή μονοπατιού από τον $s$ στον $t$. Δρομολόγηση ροής ίσης με την ελάχιστη χωρητικότητα ακμής στο μονοπάτι.
2. Υπολογισμός της *παραμένουσας χωρητικότητας* σε κάθε ακμή του μονοπατιού $p$ (καθώς και στις αντίθετές τους): κατασκευή του *παραμένοντος δικτύου (residual network)*

---

Ας εξηγήσουμε λίγο περισσότερο το βήμα 2. Έστω $f_i$ η ροή που προστίθεται στην $i$-οστή επανάληψη (δηλαδή $f_i = \min_{e \in p} c_{i-1}(e)$, όπου $c_{i-1}$ είναι η συνάρτηση χωρητικότητας στο τέλος της $(i-1)$-οστής επανάληψης). Σε κάθε μία από τις ακμές του μονοπατιού $p$ αφαιρούμε χωρητικότητα ίση με $f_i$ και στις αντίθετες ακμές προσθέτουμε χωρητικότητα ίση με $f_i$ (αν δεν υπάρχουν κάποιες από τις αντίθετες ακμές, τις προσθέτουμε). Το δίκτυο που προκύπτει είναι το παραμένον δίκτυο.

Η τελική ροή είναι το άθροισμα όλων των $f_i$. Αν σε κάποιο ζεύγος αντίθετων ακμών υπάρχει ροή και στις δύο ακμές, τότε η διαφορά τους αποδίδεται στην ακμή που είχε τη μεγαλύτερη ροή, ενώ η ροή της αντίθετης ακμής μηδενίζεται. Ένα παράδειγμα της εκτέλεσης του αλγορίθμου δίνεται στα Σχήματα 6.8, 6.9, 6.10.

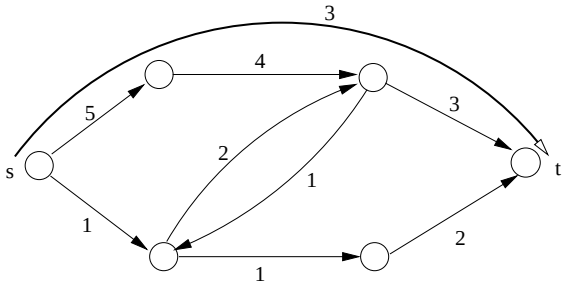
Τα μονοπάτια που χρησιμοποιεί ο αλγόριθμος λέγονται *μονοπάτια επαύξησης (augmenting paths)*.

Πολυπλοκότητα του αλγορίθμου: για ακέραιες χωρητικότητες είναι $O(|f^*||E|)$, όπου $f^*$ η τιμή της μέγιστης ροής, καθώς σε κάθε επανάληψη η ροή αυξάνεται τουλάχιστον κατά 1 μονάδα, και η εύρεση μονοπατιού από τον $s$ στον $t$ γίνεται σε χρόνο $(|E|)$. Επομένως, ο αλγόριθμος αυτός δεν είναι πολυωνυμικού χρόνου στη χειρότερη περίπτωση (γιατί;).

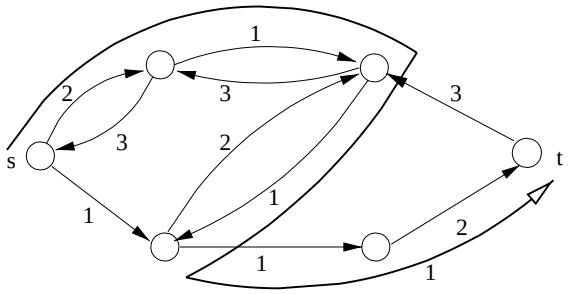
Εάν όμως επιλέγεται κάθε φορά το συντομότερο μονοπάτι, τότε ο αριθμός των επαναλήψεων γίνεται πολυωνυμικός, όπως απέδειξαν οι Edmonds-Karp (1972) - ο αντίστοιχος αλγόριθμος έχει πολυπλοκότητα $O(|V||E|^2)$. Ακόμη ταχύτερος είναι ο αλγόριθμος του Goldberg (1986-87) με πολυπλοκότητα $O(|V|^2|E|)$ και $O(|V|^3)$ (μέθοδος preflow-push).

---

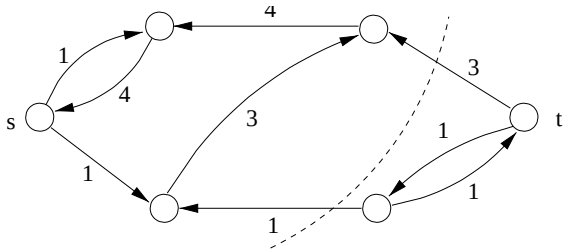
όλοι αυτοί οι ορισμοί είναι ισοδύναμοι με αυτόν που δίνουμε εδώ.




Σχήμα 6.8: Το αρχικό δίκτυο και η πρώτη ροή, μεγέθους 3, από τον κόμβο $s$ στον $t$.




Σχήμα 6.9: Το παραμένον δίκτυο και η δεύτερη ροή, μεγέθους 1, από τον κόμβο $s$ στον $t$.




Σχήμα 6.10: Το παραμένον δίκτυο. Δεν υπάρχει μονοπάτι από τον κόμβο $s$ στον $t$. Η συνολική ροή (δεν απεικονίζεται) προκύπτει από τη διαφορά της χωρητικότητας σε σχέση με το αρχικό δίκτυο. Η ελάχιστη τομή (με διακεκομμένη γραμμή) έχει συνολική χωρητικότητα 4, ίση με τη μέγιστη ροή.

**Πρόβλημα ταιριάσματος (Matching)** Αξίζει να σημειωθεί ότι ένα άλλο γνωστό πρόβλημα, το πρόβλημα Maximum Matching (Μέγιστο Ταίριασμα), και επομένως και η ειδική του περίπτωση Perfect Matching (Τέλειο Ταίριασμα), λύνονται, όταν η είσοδος είναι διμερής γράφος, με αναγωγή στο πρόβλημα Maximum Flow.

## 6.5 Πολυπλοκότητα γραφοθεωρητικών προβλημάτων

Υπάρχουν πολυωνυμικοί (ντετερμινιστικοί) αλγόριθμοι για τα κάτωθι προβλήματα: κύκλος Euler, reachability και διάσχιση γράφων, συνεκτικές συνιστώσες, συντομότερα μονοπάτια, ελάχιστο συνδετικό δένδρο (minimum spanning tree), μέγιστη ροή, ταίριασμα (matching), χρωματισμός ακμών σε διμερείς γράφους (bipartite graph edge coloring).

Αντιθέτως, για τα κάτωθι προβλήματα (που είναι όλα ισοδύναμα από άποψη πολυπλοκότητας, NP-πλήρη) είναι γνωστοί μόνο μη ντετερμινιστικοί αλγόριθμοι πολυωνυμικού χρόνου και η ακριβής πολυπλοκότητά τους είναι ανοιχτό πρόβλημα: κάλυψη με κορυφές (vertex cover), κλίκα (clique), κύκλος Hamilton (Hamilton circuit), πρόβλημα πλανόδιου πωλητή (traveling salesperson problem), 3-χρωματισμός (3-colorability), ισομορφισμός υπογράφων (subgraph isomorphism), 3-διάστατο ταίριασμα (3-dimensional matching, 3DM).

## 6.6 Διαδραστικό Υλικό – Σύνδεσμοι

- Διαδραστικό site οπτικοποιημένων αλγορίθμων και δομών δεδομένων: http://visualgo.net/
- Διαδραστικό site με animated παρουσίαση βασικών αλγορίθμων: http://www.algomation.com/
- Ηλεκτρονικό σύγγραμμα για Foundations of Computer Science, με περιγραφή και ανάλυση πλήθους βασικών αλγορίθμων. Al Aho and Jeff Ullman, "Foundations of Computer Science": http://infolab.stanford.edu/~ullman/focs.html
- Ηλεκτρονικό σύγγραμμα για Αλγόριθμους, βασικούς και προχωρημένους. S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani, "Algorithms": http://code.google.com/p/eclipselu/downloads/detail?name=algorithms.pdf
- Λογισμικό LEDA για ανάπτυξη αλγορίθμων (η βασική έκδοση διατίθεται ελεύθερα): http://algorithmic-solutions.com/leda/ledak/index.htm

## 6.7 Ασκήσεις

1. Σε έναν κατευθυνόμενο γράφο, ένας κόμβος με indegree (πλήθος εισερχομένων ακμών) μηδέν λέγεται πηγή. Θεωρήστε ότι ο γράφος δίνεται σε μορφή πίνακα γειτνίασης.

   (α□) Αποδείξτε ότι σε κάθε ακυκλικό κατευθυνόμενο γράφο υπάρχει τουλάχιστον μία πηγή.

   (β□) Δείξτε ότι ένας κατευθυνόμενος γράφος με $n$ κόμβους είναι ακυκλικός αν και μόνο αν μπορούμε να τοποθετήσουμε ετικέτες $1, 2, \ldots, n$ στους κόμβους ώστε όλες οι ακμές να κατευθύνονται από κόμβο με μικρότερη ετικέτα σε κόμβο με μεγαλύτερη ετικέτα.

(γ□) Περιγράψτε αποδοτικό αλγόριθμο που να αποφαίνεται αν ένας κατευθυνόμενος γράφος είναι ακυκλικός.

2. (α□) Στην όχθη ενός ποταμού βρίσκονται ένας λύκος, ένα πρόβατο κι ένα καφάσι με μαρούλια. Υπάρχει μόνο μια βάρκα, η οποία εκτός από το βαρκάρη μπορεί να μεταφέρει μόνο ένα από τα προηγούμενα κάθε φορά. Όταν ο βαρκάρης είναι παρών τότε επικρατεί ασφάλεια στο σύστημα. Παρόλα αυτά όταν απουσιάζει, κάποια από τα παραπάνω μπορούν το ένα να φάει το άλλο. Συγκεκριμένα ισχύουν τα εξής:

   - Αν ο λύκος και το πρόβατο μείνουν αφύλακτα στην όχθη όσο ο βαρκάρης μεταφέρει το καφάσι με τα μαρούλια, ο λύκος μπορεί να φάει το πρόβατο.
   - Αν το πρόβατο μείνει αφύλακτο μαζί με τα μαρούλια στην όχθη όσο ο βαρκάρης μεταφέρει το λύκο, το πρόβατο θα φάει τα μαρούλια.

   Βρείτε έναν τρόπο ώστε να καταφέρει ο βαρκάρης να τα μεταφέρει και τα τρία άθικτα στην απέναντι όχθη.

   (β□) Γενικεύοντας, έστω ότι υπάρχουν $n$ αντικείμενα $\{x_1, x_2, \ldots x_n\}$, τα οποία ο βαρκάρης επιθυμεί να περάσει στην απέναντι όχθη. Δίνεται γι' αυτά ένας γράφος ασυμβατοτήτων, του οποίου οι κορυφές είναι τα $n$ αντικείμενα και το $x_i$ συνδέεται με το $x_j$ όταν τα $x_i$ και $x_j$ δεν επιτρέπεται να μείνουν αφύλακτα μαζί στην ίδια όχθη (δηλαδή όταν κάποιο από τα δύο μπορεί να φάει το άλλο).

   Βρείτε ποιά πρέπει να είναι τουλάχιστον η χωρητικότητα της βάρκας ώστε το πρόβλημα να λύνεται όταν ο γράφος ασυμβατοτήτων είναι:

   - αλυσίδα (chain) $P_n$,
   - δακτύλιος (ring) $C_n$,
   - αστέρι (star) $S_n$,
   - ο πλήρης γράφος $K_n$,
   - πλέγμα (grid) διαστάσεων $\sqrt{n} \times \sqrt{n}$,
     Πλέγμα (grid) διαστάσεων $m \times n$ είναι ο γράφος $G(V, E)$, με $V = \{(i, j) : i = 1 \ldots m, j = 1 \ldots n\}$ και $E = \{((i, j), (i', j')) : |i - i'| + |j - j'| = 1\}$

   Πόσες φορές πρέπει ο βαρκάρης να διασχίσει τον ποταμό σε κάθε μια από τις παραπάνω περιπτώσεις; Δώστε μέθοδο επίλυσης.

   (γ□) Περιγράψτε έναν αλγόριθμο για τη γενική περίπτωση του προβλήματος: είσοδος η χωρητικότητα της βάρκας και ο γράφος ασυμβατοτήτων.

3. Ένας οδηγός αποφασίζει να κάνει ένα ταξίδι από την πόλη $X$ μέχρι την πόλη $Y$ με το αυτοκίνητό του, το οποίο έχει αυτονομία κίνησης ως προς τη βενζίνη που μπορεί να αποθηκεύσει, $k$ χιλιόμετρα. Αν ο οδηγός διαθέτει χάρτη στον οποίο αναφέρονται όλα τα βενζινάδικα της διαδρομής με τις αποστάσεις τους και επιπλέον επιθυμεί να πραγματοποιήσει όσο το δυνατό λιγότερες στάσεις, πώς πρέπει να προγραμματίσει τον ανεφοδιασμό καυσίμων; Αποδείξτε την ορθότητα του αλγορίθμου που επινοήσατε.

   Σημείωση: δεν υπάρχει ζεύγος διαδοχικών βενζινάδικων που να απέχουν μεταξύ τους πάνω από $k$ χιλιόμετρα.

4. Κάποιος ισχυρίζεται οτι παρόλο που ο αλγόριθμος Dijkstra δε δουλεύει για κατευθυνόμενους γράφους με αρνητικά βάρη, μπορεί να τροποποιηθεί ώστε να δίνει σωστά αποτελέσματα. Πιο συγκεκριμένα αν $v$ είναι η ακμή με το ελάχιστο βάρος (αρνητικό), προτείνει να προσθέσουμε σε όλες τις ακμές βάρος $|w(v)|$ ώστε να αποκτήσουν όλες οι ακμές θετικό βάρος. Στη συνέχεια προτείνει να εφαρμόσουμε τον αλγόριθμο του Dijkstra ως έχει. Εξηγήστε αν η παραπάνω σκέψη λύνει το single-source πρόβλημα σωστά, δίνοντας απόδειξη ή αντιπαράδειγμα.

# Βιβλιογραφία


[1] S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani. “Algorithms”, MacGraw-Hill, 2006. “Αλγόριθμοι”, ελληνική έκδοση, Εκδόσεις Κλειδάριθμος, 2008.

[2] Thomas Cormen, Charles Leiserson, Ronald Rivest and Cliff Stein: “Introduction to Algorithms”, 3rd edition, MIT Press, 2009. Ελληνική έκδοση: “Εισαγωγή στους Αλγόριθμους”, Πανεπιστημιακές Εκδόσεις Κρήτης, 2012.

[3] D. Harel: “Algorithmics: The Spirit of Computing”, Addison-Wesley, Reading, MA, 1st edition, 1987; 2nd edition, 1992. 3rd edition (with Y. Feldman), 2004.

[4] A.V. Aho, J.E. Hopcroft, and J.D. Ullman: “The Design and Analysis of Computer Algorithms”, Addison-Wesley Series in Computer Science andInformation Processing, 1974.

[5] A. Levitin: “Ανάλυση και Σχεδίαση Αλγορίθμων”, Εκδόσεις Τζιόλα, 2007.

[6] Reinhard Diestel, “Graph Theory” (3rd edition), Springer 2005.

[7] H.R. Lewis and C.H. Papadimitriou: “Elements of the Theory of Computation”, 2nd edition, Prentice Hall, 1997. Μεταφρασμένη έκδοση: ”Στοιχεία Θεωρίας Υπολογισμού”, Εκδόσεις Κριτική, 2005.

[8] J.A. Bondy, U.S.R. Murty. “Graph Theory with Applications”. North Holland, 1976. Διατίθεται ελεύθερα στο διαδίκτυο.

[9] Ronald Graham, Donald Knuth, Oren Patashnik. “Συνκριτά Μαθηματικά” (δεύτερη έκδοση, μτφ. Χ. Καπούτσης, επιμ. Ε. Ζάχος) Εκδόσεις Κλειδάριθμος, 2011.

[10] M. Sipser: “Introduction to the Theory of Computation”, 2nd edition, Course Technology, 2005. Μεταφρασμένη έκδοση: “Εισαγωγή στη Θεωρία Υπολογισμού”, Παν. Εκδόσεις Κρήτης, 2007.

# Κεφάλαιο 7

# Πεπερασμένα Αυτόματα και Κανονικές Παραστάσεις

## 7.1 Εισαγωγή

Στο κεφάλαιο αυτό θα ασχοληθούμε με τυπικές γλώσσες που μπορούν να περιγράψουν υπολογιστικά προβλήματα και επίσης χρησιμεύουν στον προγραμματισμό. Επίσης θα επιδιώξουμε να ταξινομήσουμε γλώσσες ανάλογα με το είδος του **αυτομάτου** (*αφηρημένης υπολογιστικής συσκευής*) που μπορεί να τις αναγνωρίσει.

Ένα αυτόματο έχει μερικές εσωτερικές καταστάσεις: $q_0, q_1, q_{16}, q_{23}, \ldots$ και μία συνάρτηση $\delta$ που καθορίζει την επόμενη κατάσταση του αυτομάτου.

Στις πιο απλές συσκευές, που ονομάζονται **μηχανισμοί**, η συνάρτηση μετάβασης (*transition function*) $\delta$ έχει ως όρισμα μία κατάσταση $q_i$ και ως τιμή μια άλλη κατάσταση $q_j$. Δεν υπάρχει δηλαδή είσοδος και έξοδος. Υπολογιστική ακολουθία σε αυτήν την περίπτωση είναι μια ακολουθία από καταστάσεις:

$$q_i \to q_j \to q_k \to q_l \cdots$$

Αν επιπλέον η συσκευή διαβάζει μια συμβολοσειρά εισόδου, σύμβολο προς σύμβολο, από αριστερά προς τα δεξιά και ανάλογα αλλάζει καταστάσεις, τότε ονομάζεται **αναγνωριστής πεπερασμένων καταστάσεων** (*FSA: finite state acceptor*). Η συνάρτηση μετάβασης είναι της μορφής

$$\delta\colon (q_i, a) \to q_j,$$

όπου $\alpha \in \Sigma$ και το $\Sigma$ το αλφάβητο εισόδου. Στο τέλος το FSA αποδέχεται ή απορρίπτει την είσοδό του.

Αν προσθέσουμε έξοδο σε ένα *FSA*, δηλαδή $\delta\colon (q_i, a) \to (q_j, b)$, όπου $b \in \Delta$ και $\Delta$ το αλφάβητο εξόδου, τότε έχουμε τη λεγόμενη μηχανή πεπερασμένων καταστάσεων (*FSM: finite state machine*). Οι FSA και FSM εμφανίζονται συχνά ως χρήσιμα εργαλεία στο λογικό, ψηφιακό σχεδιασμό. Προσθέτοντας στο αυτόματό μας μνήμη υπό μορφή στοίβας (*stack*) έχουμε πολύ περισσότερες δυνατότητες: το αυτόματο ονομάζεται τότε **αυτόματο στοίβας** (*PDA: push-down automaton*). Αν αντί στοίβας έχουμε απεριόριστη δυνατότητα μνήμης υπό μορφή ταινίας, τότε

έχουμε τη γνωστή **μηχανή Turing** (*TM*). **Γραμμικά περιορισμένο αυτόματο** (*LBA: linearly bounded automaton*) είναι μια μηχανή Turing που όμως μπορεί να χρησιμοποιήσει ταινία που το μήκος της είναι μια γραμμική συνάρτηση του μήκους της εισόδου.

Τα διαφορετικά αυτά αυτόματα αναγνωρίζουν κλάσεις γλωσσών που όμως μπορούν να περιγραφούν και με άλλους τρόπους π.χ. με αλγεβρικό τρόπο, ή με **τυπικές γραμματικές**.

Οι γλώσσες προγραμματισμού είναι προφανώς **τυπικές γλώσσες** (*formal languages*) που έχουν αυστηρό συντακτικό ορισμό. Οι τυπικές γλώσσες μπορούν να ταξινομηθούν ανάλογα με τις τυπικές γραμματικές που τις παράγουν. Στη θεωρία τυπικών γλωσσών οι πρωταρχικές έννοιες είναι τα **σύμβολα** (ως αντικείμενα) και η **παράθεση** (ως πράξη).

Ένα **αλφάβητο** $\Sigma$ είναι ένα πεπερασμένο σύνολο συμβόλων. Μια λέξη ή πρόταση ή συμβολοσειρά ή string είναι μια πεπερασμένου μήκους ακολουθία συμβόλων. Το μήκος του string $w$ συμβολίζεται με $|w|$. Το κενό string συμβολίζεται με $\varepsilon$. Η παράθεση των strings $x$ και $y$ συμβολίζεται με $xy$. 'Αλλοι χρήσιμοι όροι είναι **πρόθεμα** (*prefix*), **υποσυμβολοσειρά** (*substring*), **υπακολουθία** (*subsequence*), **κατάληξη** (*suffix*), **αντίστροφη** (*reversal*), **παλινδρομική** ή **καρκινική** (*palindrome*).

Ισχύουν $x\varepsilon = \varepsilon x = x$ για όλα τα strings $x$ και $|\varepsilon| = 0$. Το string $x^k$ μπορεί να οριστεί με πρωταρχική αναδρομή:

$$\begin{cases} x^0 = \varepsilon \\ x^{k+1} = x^k x \end{cases}$$

*Ορισμός* 7.1. Αν $\Sigma$ είναι ένα αλφάβητο τότε $\Sigma^*$ είναι το σύνολο όλων των strings από το $\Sigma$. Μια γλώσσα $L$ από το $\Sigma$ δεν είναι παρά κάποιο υποσύνολο του $\Sigma^*$.

Παραδείγματα (με αλφάβητο $\Sigma = \{a, b\}$):

- $L_1 = \{w \in \Sigma^* \mid w \text{ αρχίζει με } a\}$
- $L_2 = \{w \in \Sigma^* \mid w \text{ περιέχει ζυγό αριθμό από } a\}$
- $L_3 = \{w \in \Sigma^* \mid w \text{ είναι παλινδρομική }\}$

## 7.2 Ντετερμινιστικά πεπερασμένα αυτόματα

Θα περιγράψουμε πρώτα τα *ντετερμινιστικά πεπερασμένα αυτόματα* (deterministic finite automata - DFA). Ένα DFA είναι ένα αυτόματο που *αναγνωρίζει* (δηλαδή *αποδέχεται* ή *απορρίπτει*) συμβολοακολουθίες που εμφανίζονται στην ταινία εισόδου του. Η κεφαλή[1] του DFA βρίσκεται αρχικά στο αριστερότερο σημείο της εισόδου. Μετακινείται δε προς τα δεξιά κατά ένα σύμβολο σε κάθε βήμα μέχρι να σταματήσει στο τέλος της ταινίας εισόδου. Από σύμβαση, η αρχική κατάσταση (initial state) ονομάζεται $q_0$, κάποιες από τις καταστάσεις αποδέχονται και ονομάζονται τελικές (accepting, final states), και οι υπόλοιπες καταστάσεις απορρίπτουν (rejecting, nonfinal states). Αν το DFA βρίσκεται σε μια κατάσταση που αποδέχεται αφότου η κεφαλή έχει διαβάσει όλη την συμβολοακολουθία είσοδου, λέμε ότι το DFA *αποδέχεται* την είσοδο. Η γλώσσα $L$, την οποία αναγνωρίζει ένα DFA, είναι το σύνολο των συμβολοακολουθιών που αποδέχεται.

[1] ώς *κεφαλή* ορίζουμε εκείνο το στοιχείο της ταινίας εισόδου που βρίσκεται την παρούσα στιγμή υπό εξέταση

*Ορισμός* 7.2. Τυπικά ένα DFA είναι μία πεντάδα $M = (Q, \Sigma, \delta, q_0, F)$, όπου:

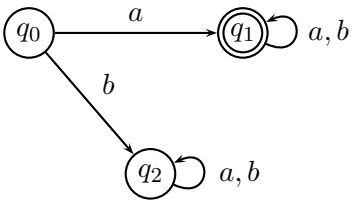
- $Q$: ένα πεπερασμένο σύνολο από καταστάσεις,
- $\Sigma$: ένα αλφάβητο εισόδου ($\Sigma \cap Q = \emptyset$),
- $\delta\colon Q \times \Sigma \to Q$: η συνάρτηση μετάβασης,
- $q_0 \in Q$: η αρχική κατάσταση,
- $F \subseteq Q$: το σύνολο των τελικών καταστάσεων.

Ενα DFA μπορεί επίσης να περιγραφεί (όπως και μια TM) από ένα *πίνακα καταστάσεων* ή από ένα *διάγραμμα καταστάσεων*

Στα παρακάτω παραδείγματα θεωρούμε αλφάβητο $\Sigma = \{a, b\}$.

*Παράδειγμα* 7.1. $L_1 = \{w \in \Sigma^* \mid w$ αρχίζει από $a\}$

Διάγραμμα:

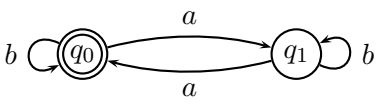



Πίνακας:

| | $a$ | $b$ |
|---|---|---|
| $q_0$ | $q_1$ | $q_2$ |
| $(q_1)$ | $q_1$ | $q_1$ |
| $q_2$ | $q_2$ | $q_2$ |

Χρησιμοποιούμε έναν επιπλέον κύκλο για να δείξουμε τις τελικές καταστάσεις.

*Παράδειγμα* 7.2. $L_2 = \{w \in \Sigma^* \mid w$ περιέχει άρτιο πλήθος από $a\}$




| | $a$ | $b$ |
|---|---|---|
| $(q_0)$ | $q_1$ | $q_0$ |
| $q_1$ | $q_0$ | $q_1$ |

*Παράδειγμα* 7.3. $L_3' = \{w \in \Sigma^* \mid w$ παλίνδρομη αρτίου μήκους$\}$, δηλαδή $L_3' = \{ww^R \mid w \in \Sigma^*, w^R =$ αντίστροφη της $w\}$. Δεν υπάρχει DFA που αποδέχεται την $L_3'$.

Για να συντομεύσουμε την περιγραφή της συμπεριφοράς του DFA, επεκτείνουμε την συνάρτηση $\delta$ ώστε να δέχεται ως όρισμα συμβολοακολουθίες αντί για απλά σύμβολα. Η $\delta$ επεκτείνεται λοιπόν ως εξής:

*Ορισμός* 7.3. $\delta : Q \times \Sigma^* \to Q$ όπου

$$\begin{cases} \delta(q, \varepsilon) = q \\ \delta(q, wa) = \delta(\delta(q, w), a) \end{cases}$$

Ο πιο πάνω ορισμός είναι αναδρομικός, ή, πιο συγκεκριμένα, είναι ορισμός σύμφωνα με το σχήμα της *πρωταρχικής αναδρομής*.

*Ορισμός* 7.4. Έχουμε:

- Ένα DFA αποδέχεται το string $w \in \Sigma^*$ ανν $\delta(q_0, w) \in F$
- Ένα DFA αποδέχεται τη γλώσσα $L(M) = \{w \mid \delta(q_0, w) \in F\}$
- Η γλώσσα $L$ λέγεται *κανονική* (*regular*) ανν $\exists$ FA $M$: $L = L(M)$

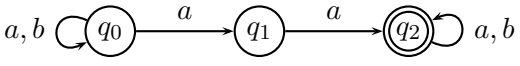
**Άσκηση:** Δείξτε ότι $\delta(q, uv) = \delta(\delta(q, u), v)$, όπου $u, v \in \Sigma^*$.

Υπάρχουν διάφορες γενικεύσεις και διαφοροποιήσεις ως προς τα DFA:

- NFA: *μη ντετερμινιστικό πεπερασμένο αυτόματο*. Σε κάθε μετάβαση υπάρχει επιλογή της επόμενης κατάστασης από ένα σύνολο πιθανών νομίμων καταστάσεων.
- NFA$_\varepsilon$: *μη ντετερμινιστικό πεπερασμένο αυτόματο με $\varepsilon$-κινήσεις*. Το πεπερασμένο αυτόματο ενδέχεται να αλλάζει την κατάστασή του χωρίς να μετακινείται η κεφαλή στην ταινία εισόδου.

## 7.3 Μη ντετερμινιστικά πεπερασμένα αυτόματα

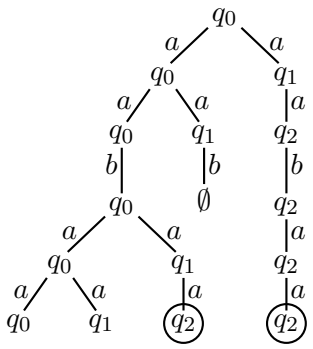
*Παράδειγμα* 7.4. NFA για $L_4 := \{w \in \Sigma^* \mid w$ περιέχει δύο συνεχόμενα $a\}$




| | $a$ | $b$ |
|---|---|---|
| $q_0$ | $\{q_0, q_1\}$ | $\{q_0\}$ |
| $q_1$ | $\{q_2\}$ | $\emptyset$ |
| $(q_2)$ | $\{q_2\}$ | $\{q_2\}$ |

Ένας υπολογισμός σε ένα NFA δεν είναι απλώς μία γραμμική (νόμιμη) ακολουθία καταστάσεων, αλλά ένα *υπολογιστικό δένδρο* (κάθε κλάδος είναι μία νόμιμη ακολουθία καταστάσεων).

Το δένδρο υπολογισμού για το παραπάνω παράδειγμα για είσοδο $aabaa$:



Η συμβολοσειρά $aabaa$ γίνεται αποδεκτή, επειδή υπάρχει τουλάχιστον ένα νόμιμο μονοπάτι που την αποδέχεται.

Στα μη ντετερμινιστικά αυτόματα, για κάθε είσοδο και κατάσταση, μπορεί να υπάρχει καμμία, μία ή πολλές πιθανές επόμενες καταστάσεις. Αυτό εκφράζεται στον ορισμό ενός NFA από το γεγονός ότι η συνάρτηση μετάβασης $\delta$ έχει ως πεδίο τιμών το *δυναμοσύνολο* του $Q$ ($\text{Pow}(Q)$).

Ένα NFA λοιπόν αποτελείται από μια πεντάδα $M = (Q, \Sigma, \delta, q_0, F)$, όπου:

- $Q$: ένα πεπερασμένο σύνολο από καταστάσεις,
- $\Sigma$: ένα πεπερασμένο αλφάβητο εισόδου,
- $\delta : Q \times \Sigma \to \text{Pow}(Q)$: η συνάρτηση μετάβασης
- $q_0 \in Q$: η αρχική κατάσταση και
- $F \subseteq Q$: το σύνολο των τελικών καταστάσεων.

Όπως και πριν, επεκτείνουμε την συνάρτηση μετάβασης $\delta$ για να συμπεριλάβουμε στο πεδίου ορισμού της τις συμβολοακολουθίες. Η επεκτεταμένη συνάρτηση μετάβασης $\delta(q, w)$ θα περιέχει το σύνολο όλων των καταστάσεων που μπορούμε να φτάσουμε από το $q$ με είσοδο το $w$.

*Ορισμός* 7.5. $\delta : Q \times \Sigma^* \to \text{Pow}(Q)$ όπου

$$\begin{cases} \delta(q, \varepsilon) = \{q\} \\ \delta(q, wa) = \{p \in \delta(r, a) | r \in \delta(q, w)\} = \bigcup\limits_{r \in \delta(q,w)} \delta(r, a) \end{cases}$$

Επεκτείνουμε περαιτέρω την $\delta$, συμπεριλαμβάνοντας στο πεδίο ορισμού το δυναμοσύνολο του $Q$. Η $\delta(P, w)$ θα περιέχει όλες τις καταστάσεις που μπορούμε να φτάσουμε από οποιαδήποτε κατάσταση στο $P$ με είσοδο $w$.

*Ορισμός* 7.6. $\delta : \text{Pow}(Q) \times \Sigma^* \to \text{Pow}(Q)$ όπου

$$\delta(P, w) = \{p \in \delta(q, w) | q \in P\} = \bigcup_{q \in P} \delta(q, w)$$

*Παρατήρηση* 7.1. Ισχύει $\delta(q, wa) = \delta(\delta(q, w), a)$

*Παρατήρηση* 7.2. Ισχύει $\delta(P, wa) = \delta(\delta(P, w), a)$

*Ορισμός* 7.7. Έχουμε:

- Ένα NFA αποδέχεται το string $w \in \Sigma^*$ ανν $\delta(q_0, w) \cap F \neq \emptyset$
- Ένα NFA $M$ αποδέχεται τη γλώσσα $L(M) = \{w | \delta(q_0, w) \cap F \neq \emptyset\}$

**Ισοδυναμία DFA και NFA.** Όπως φαίνεται από τον ορισμό της $\delta$ ενός NFA, ένα DFA είναι μια "υποπερίπτωση" ενός NFA. Παρ' όλα αυτά, τα NFA δεν μας παρέχουν περισσότερες δυνατότητες υπολογισμού από ότι τα DFA. Αυτό αποδεικνύει το παρακάτω θεώρημα:

*Θεώρημα* 7.5. *(Rabin - Scott)*

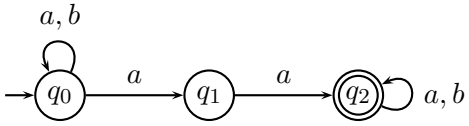
$$\textit{Έστω } M \textit{ ένα NFA. Τότε } \exists \text{ DFA } M' : L(M) = L(M')$$

*Απόδειξη.* Ορίζουμε το DFA $M' = (Q', \Sigma', q_0', F', \delta')$ όπου

- $Q' = \text{Pow}(Q)$,
- $\Sigma' = \Sigma$, $q_0' = \{q_0\}$,
- $F' = \{R \in Q' | R \cap F \neq \emptyset\} = \bigcup_{R \cap F \neq \emptyset} (R \in Q')$ και τέλος
- $\delta'(R, a) = \delta(R, a)$, όπου $R \in Q'$.

Μένει να αποδειχθεί ότι $L(M) = L(M')$ με την βοήθεια του ισχυρισμού: $\delta'(q_0', w) = \delta(q_0, w)$. (Αφήνεται ως άσκηση· χρησιμοποιήστε επαγωγή.) □

*Παράδειγμα* 7.6.

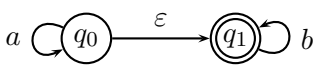



| | $a$ | $b$ |
|---|---|---|
| $\emptyset$ | $\emptyset$ | $\emptyset$ |
| $\{q_0\}$ | $\{q_0, q_1\}$ | $\{q_0\}$ |
| $\{q_1\}$ | $\{q_2\}$ | $\emptyset$ |
| $\{q_2\}$ | $\{q_2\}$ | $\{q_2\}$ |
| $\{q_0, q_1\}$ | $\{q_0, q_1, q_2\}$ | $\{q_0\}$ |
| $\{q_0, q_2\}$ | $\{q_0, q_1, q_2\}$ | $\{q_0, q_2\}$ |
| $\{q_1, q_2\}$ | $\{q_2\}$ | $\{q_2\}$ |
| $\{q_0, q_1, q_2\}$ | $\{q_0, q_1, q_2\}$ | $\{q_0, q_2\}$ |

Το DFA φαίνεται στο παράδειγμα 7.9.

**Μη ντετερμινιστικά αυτόματα με $\varepsilon$-κινήσεις - NFA$_\varepsilon$**

*Παράδειγμα* 7.7. NFA$_\varepsilon$ για $L_5 := \{a^*b^*\} = \{a^n b^m \mid n, m \in \mathbb{N}\}$




| | $a$ | $b$ | $\varepsilon$ |
|---|---|---|---|
| $q_0$ | $\{q_0\}$ | $\emptyset$ | $\{q_1\}$ |
| $q_1$ | $\emptyset$ | $\{q_1\}$ | $\emptyset$ |

Μπορούμε να γενικεύσουμε το μη ντετερμινιστικό αυτόματο που ορίσαμε ώστε να συμπεριλάβουμε στο πεδίο ορισμού της $\delta$ και το $\varepsilon$. Με άλλα λόγια, το αυτόματο μπορεί να αλλάζει κατάσταση χωρίς να διαβάζει κάποια είσοδο, με μια $\varepsilon$-κίνηση. Ένα NFA$_\varepsilon$ λοιπόν είναι μια πεντάδα $M = (Q, \Sigma, \delta, q_0, F)$ όπου

- $Q$: ένα πεπερασμένο σύνολο από καταστάσεις,
- $\Sigma$: ένα πεπερασμένο αλφάβητο εισόδου,
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to \text{Pow}(Q)$: η συνάρτηση μετάβασης
- $q_0 \in Q$: η αρχική κατάσταση και
- $F \subseteq Q$: το σύνολο των τελικών καταστάσεων.

Για την ανάλυσή μας, είναι χρήσιμο να ορίσουμε το *ε-κλείσιμο* μιας κατάστασης.

*Ορισμός* 7.8. Ως $\varepsilon$-κλείσιμο: $Q \to \text{Pow}(Q)$ ορίζουμε το

$$\varepsilon\text{-κλείσιμο}(q) = \{p| \text{ τα } p \text{ προσβάσιμα από το } q \text{ μόνο με } \varepsilon\text{-κινήσεις}\}$$

Παρατηρούμε ότι πάντα $q \in \varepsilon$-κλείσιμο$(q)$. Επεκτείνουμε τον ορισμό αυτό:

*Ορισμός* 7.9. Ως $\varepsilon$-κλείσιμο: $\text{Pow}(Q) \to \text{Pow}(Q)$ ορίζουμε το

$$\varepsilon\text{-κλείσιμο}(P) = \bigcup_{q \in P} \varepsilon\text{-κλείσιμο}(q)$$

Παρατηρούμε επιπλέον ότι $\varepsilon$-κλείσιμο($\varepsilon$-κλείσιμο$(P)) = \varepsilon$-κλείσιμο$(P)$. Η συνάρτηση μετάβασης $\delta$ μπορεί να επεκταθεί. Η επεκτεταμένη συνάρτηση μετάβασης $\delta(q, w)$ θα περιέχει το σύνολο όλων των καταστάσεων που μπορούμε να φτάσουμε από το $q$ με είσοδο το $w$, καθώς και με οσεσδήποτε αρχικές, ενδιάμεσες και τελικές $\varepsilon$-κινήσεις.

*Ορισμός* 7.10. $\delta : Q \times \Sigma^* \to \text{Pow}(Q)$ όπου

$$\begin{cases} \delta(q, \varepsilon) = \varepsilon\text{-κλείσιμο}(q) \\ \delta(q, wa) = \varepsilon\text{-κλείσιμο}(\{p \in \delta(r, a) \mid r \in \delta(q, w)\}) = \varepsilon\text{-κλείσιμο}(\bigcup_{r \in \delta(q,w)} \delta(r, a)) \end{cases}$$

Επιπλέον, επεκτείνουμε την $\delta$, συμπεριλαμβάνοντας στο πεδίο ορισμού το δυναμοσύνολο του $Q$.

*Ορισμός* 7.11. $\delta : \text{Pow}(Q) \times \Sigma^* \to \text{Pow}(Q)$ όπου

$$\delta(P, w) = \{p \in \delta(q, w) | q \in P\} = \bigcup_{q \in P} \delta(q, w)$$

*Ορισμός* 7.12. Έχουμε:

- Ένα NFA$_\varepsilon$ αποδέχεται το string $w \in \Sigma^*$ ανν $\delta(q_0, w) \cap F \neq \emptyset$
- Ένα NFA$_\varepsilon$ αποδέχεται τη γλώσσα $L(M) = \{w | \delta(q_0, w) \cap F \neq \emptyset\}$

**Ισοδυναμία NFA και NFA$_\varepsilon$.** Και πάλι, μπορεί να θεωρήσει κανείς ότι τα NFA είναι μια "υποπερίπτωση" των NFA$_\varepsilon$. Όμως, όπως και πριν, τα NFA$_\varepsilon$ δεν έχουν περισσότερες δυνατότητες υπολογισμού από τα NFA, όπως αποδεικνύει το επόμενο θεώρημα, και άρα και από τα DFA.

*Θεώρημα* 7.8. *Έστω $M$ ένα NFA$_\varepsilon$ τότε* $\exists$ NFA $M' : L(M) = L(M')$

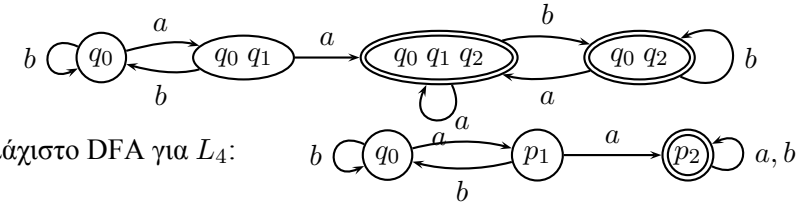
*Απόδειξη.* Ορίζουμε το NFA $M' = (Q, \Sigma, q_0, F', \delta')$ όπου

$$F' = \begin{cases} F \cup \{q_0\}, & \text{αν } \varepsilon\text{-κλείσιμο}(q_0) \cap F \neq \emptyset \\ F, & \text{ειδάλλως} \end{cases}, \qquad \delta'(q,a) = \delta(q,a)$$

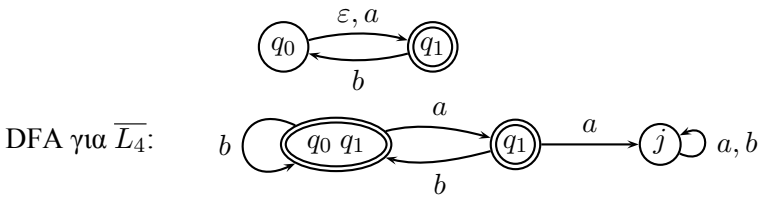
Πλέον, προκειμένου να ισχύει $L(M) = L(M')$, αρκεί να αποδειχθεί ο ισχυρισμός: $\forall w \in \Sigma^* - \{\varepsilon\}$: $\delta'(q_0, w) = \delta(q_0, w)$. ('Ασκηση.) □

Ακολουθούν μερικά παραδείγματα με μη ντετερμινιστικά αυτόματα και ισοδύναμά τους ντετερμινιστικά αυτόματα.

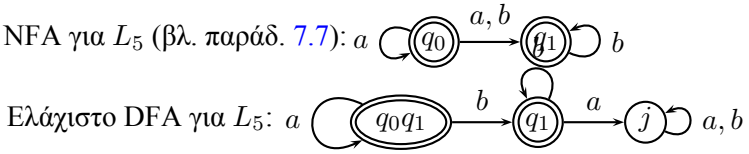
*Παράδειγμα* 7.9. DFA για $L_4$ (βλ. παραδ. 7.4):




*Παράδειγμα* 7.10. NFA$_\varepsilon$ για $\overline{L_4}$ (δηλαδή "όχι δύο συνεχόμενα $a$"):




*Παράδειγμα* 7.11. NFA για $L_5$ (βλ. παράδ. 7.7):




## 7.4 Κανονικές παραστάσεις

Παρακάτω, ορίζουμε κάποιες πράξεις επί των γλωσσών.

*Ορισμός* 7.13. Έστω $L$, $L_1$, $L_2$ γλώσσες επί του ίδιου αλφαβήτου $\Sigma$.

- $L_1 L_2 := \{uv \mid u \in L_1 \wedge v \in L_2\}$: παράθεση
- $L_1 \cup L_2 := \{w \mid w \in L_1 \vee w \in L_2\}$: ένωση
- $L_1 \cap L_2 := \{w \mid w \in L_1 \wedge w \in L_2\}$: τομή
- $L^0 := \{\varepsilon\}$, $L^{n+1} := LL^n$
- $L^* := \bigcup_{n=0}^{\infty} L^n$: άστρο του Kleene
- $L^+ := \bigcup_{n=1}^{\infty} L^n$

Τώρα πλέον μπορούμε να εισάγουμε επαγωγικά έναν συμβολισμό για τις κανονικές γλώσσες, τις λεγόμενες *κανονικές παραστάσεις*.

*Ορισμός* 7.14. Κανονική παράσταση είναι:

$\emptyset$**:** παριστάνει το κενό σύνολο·

$\varepsilon$**:** παριστάνει το $\{\varepsilon\}$·

$a$**:** παριστάνει το $\{a\}$, όπου $a \in \Sigma$·

$(r + s)$**:** παριστάνει το $R \cup S$, όπου $r$, $s$ κανονικές παραστάσεις που παριστάνουν τα $R$, $S$ αντιστοίχως·

$(rs)$**:** παριστάνει το $RS$, όπου $r$, $s$ κανονικές παραστάσεις που παριστάνουν τα $R$, $S$ αντιστοίχως·

$(r^*)$**:** παριστάνει το $R^*$, όπου $r$ κανονική παράσταση που παριστάνει το $R$.

**Σύμβαση:** Μπορούμε να περιορίσουμε τις παρενθέσεις αν χρησιμοποιήσουμε την ακόλουθη προτεραιότητα των τελεστών: $*$, παράθεση, ένωση.

*Παράδειγμα* 7.12.

$L_1 = a(a + b)^*$
$L_2 = (b^*ab^*a)^*b^* = (b + ab^*a)^*$
$L_3$ δεν είναι δυνατόν να παρασταθεί με κανονική παράσταση
$L_4 = (a + b)^*aa(a + b)^*$ (τουλάχιστον δύο συνεχόμενα $a$)
$\overline{L_4} = (a + \varepsilon)(ba + b)^*$ (όχι συνεχόμενα $a$)
$L_5 = a^*b^*$

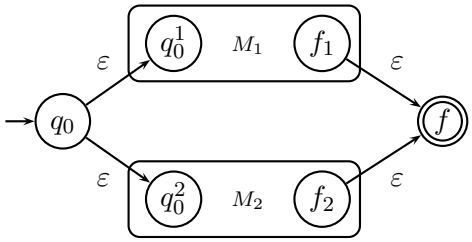
*Θεώρημα* 7.13. *Μία γλώσσα μπορεί να παρασταθεί με κανονική παράσταση ανν $L = L(M)$ για κάποιο πεπερασμένο αυτόματο $M$.*

*Ιδέα απόδειξης.*

$\Rightarrow$ Επαγωγή στην δομή της κανονικής παράστασης. Έστω $r$ η κανονική παράσταση.

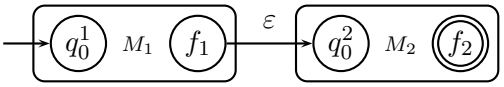
1. Επαγωγική Βάση:


2. Επαγωγικό Βήμα: Υποθέτουμε ότι οι $L(r_1)$ και $L(r_2)$ αναγνωρίζονται από NFA$_\varepsilon$ $M_1$, $M_2$ αντιστοίχως με τελική κατάσταση $f_1$, $f_2$ αντιστοίχως.
   Περίπτωση α: $r = r_1 + r_2$



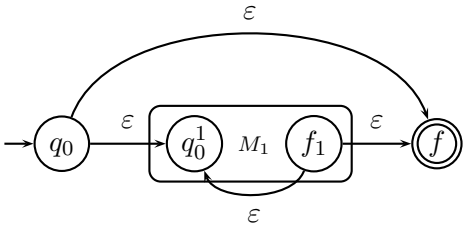

$$L(M) = L(M_1) \cup L(M_2)$$

   Περίπτωση β: $r = r_1 r_2$




$$L(M) = L(M_1)L(M_2)$$

   Περίπτωση γ: $r = r_1^*$




$$L(M) = L(M_1)^*$$

$\Leftarrow$ Θεωρούμε ότι το DFA είναι της μορφής $= (Q, \Sigma, \delta, q_1, F)$, όπου τα στοιχεία του $Q$ είναι αριθμημένα κατά αύξουσα σειρά και η αρχική κατάσταση είναι η $q_1$, δηλαδή $Q = \{q_1, \ldots, q_n\}$, όπου $n = |Q|$. Ορίζουμε:

$$R_{ij}^k = \{w \mid \tilde{\delta}(q_i, w) = q_j \text{ και } \forall x \text{ πρόθεμα του } w \text{ με } x \neq w, \varepsilon\colon \tilde{\delta}(q_i, x) = q_l \Rightarrow l \leq k\}$$

Δηλαδή, $R_{ij}^k$ είναι το σύνολο των συμβολοακολουθιών που οδηγούν από την κατάσταση $q_i$ στην $q_j$ χωρίς να περνούν από οποιαδήποτε κατάσταση με δείκτη μεγαλύτερο από $k$. Προφανώς, αφού δεν υπάρχει κατάσταση με δείκτη μεγαλύτερο από $n$, το $R_{ij}^n$ περιέχει όλες τις συμβολοακολουθίες από την $q_i$ στην $q_j$. Μπορούμε να υπολογίσουμε το $R_{ij}^k$ αναδρομικά, σύμφωνα με μια ιδέα των Floyd και Warshall, ως εξής:

**Αλγόριθμος Floyd-Warshall**

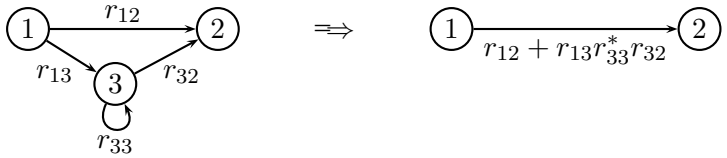
$$R_{ij}^0 = \begin{cases} \{a|\delta(q_i,a)=q_j\}, & \text{αν } i \neq j \\ \{a|\delta(q_i,a)=q_j\} \cup \{\varepsilon\}, & \text{αν } i = j \end{cases}$$
$$R_{ij}^k = R_{ik}^{k-1}(R_{kk}^{k-1})^* R_{kj}^{k-1} \cup R_{ij}^{k-1}$$

Τέλος, αρκεί να παρατηρήσουμε ότι $L(M) = \bigcup_{q_j \in F} R_{1j}^n$. Η ιδέα των Floyd και Warshall είναι ότι το $R_{ij}^k$ αποτελείται από συμβολοακολουθίες που οδηγούν από την κατάσταση $q_i$ στην $q_j$ είτε χωρίς να περνούν από την κατάσταση $q_k$ (οπότε περιέχονται και στο $R_{ij}^{k-1}$, είτε περνούν από την κατάσταση $q_k$ μία ή περισσότερες φορές (οι λεπτομέρειες της απόδειξης αφήνονται ως άσκηση.) □

**Κατασκευή κανονικής παράστασης από FA** Το 2ο σκέλος της πιο πάνω απόδειξης δίνει έναν πλήρη, συστηματικό, αλλά συχνά χρονοβόρο, τρόπο κατασκευής της κανονικής παράστασης που αντιστοιχεί σε ένα FA. Παρακάτω δίνεται ένας πιο γρήγορος τρόπος που βασίζεται στην έννοια του GNFA: γενικευμένο αυτόματο, του οποίου η συνάρτηση μετάβασης δέχεται και κανονικές παραστάσεις (στα βέλη του μπορούμε να γράφουμε κανονικές παραστάσεις αντί για σκέτα σύμβολα).

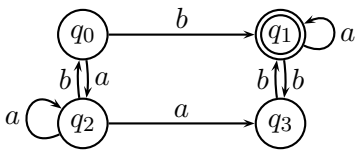
Κατ' αρχάς, θεωρούμε ότι το FA έχει μια αρχική και μια τελική κατάσταση, διαφορετικές μεταξύ τους. Αν υπάρχουν πάνω από μια τελικές καταστάσεις, προσθέτουμε μια νέα κατάσταση στην οποία βαίνουν όλες με $\varepsilon$-κινήσεις και στη συνέχεια τις κάνουμε μη τελικές. Αντίστοιχα μετατρέπουμε και την αρχική κατάσταση σε μη τελική, αν χρειάζεται.

Φέρνοντας το FA μας σε αυτή τη μορφή, μπορούμε να απαλείψουμε μία μία τις ενδιάμεσες καταστάσεις σύμφωνα με το πιο κάτω σχήμα:



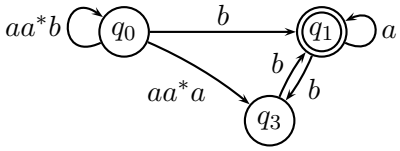

Με διαδοχικές απαλοιφές οδηγούμαστε τελικά στο επιθυμητό GNFA (γενικευμένο αυτόματο, του οποίου η συνάρτηση μετάβασης δέχεται και κανονικές παραστάσεις). Στη συνέχεια θα δώσουμε ένα παράδειγμα με όλα τα ενδιάμεσα βήματα υπολογισμού.

*Παράδειγμα* 7.14. 'Εστω πως έχουμε το ακόλουθο FA:

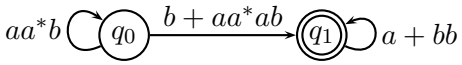



Επιλέγουμε να διαγράψουμε τη κατάσταση $q_2$, οπότε ενημερώνουμε τις μεταβάσεις των άλλων

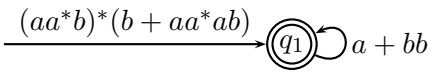
καταστάσεων:




Επιλέγουμε να διαγράψουμε τη κατάσταση $q_3$, οπότε ενημερώνουμε τα υπόλοιπα:




Επιλέγουμε να διαγράψουμε τη κατάσταση $q_0$:




Τελικά, η κανονική έκφραση που προκύπτει είναι:

$$(aa^*b)^*(b + aa^*ab)(a + bb)^*$$

## 7.5 Ελαχιστοποίηση DFA

Σε πολλές περιπτώσεις μπορούμε να μειώσουμε τον αριθμό καταστάσεων ενός DFA ως εξής: Δύο καταστάσεις μπορούν να συγχωνευτούν σε μία αν είναι και οι δύο τελικές ή και οι δύο μη τελικές και αν ξεκινώντας από οποιαδήποτε από αυτές το αυτόματο θα έχει το ίδιο αποτέλεσμα (ως προς την αποδοχή ή μη) για οποιαδήποτε συμβολοσειρά διαβαστεί στη συνέχεια. Για παράδειγμα, δύο καταστάσεις που απορρίπτουν και με κάθε σύμβολο πηγαίνουν στον εαυτό τους ('junk' states) μπορούν να συγχωνευτούν σε μία.

Πιο αυστηρά, οι καταστάσεις $q_i, q_j$ μπορούν να συγχωνευτούν αν:

$$\forall w \in \Sigma^* : \delta(q_i, w) \in F \Leftrightarrow \delta(q_j, w) \in F$$

Προσέξτε ότι ο παραπάνω ορισμός περιλαμβάνει και την περίπτωση $w = \varepsilon$: αυτό ισοδυναμεί με την απαίτηση οι δύο καταστάσεις να είναι του ίδιου είδους ως προς την αποδοχή (δηλ. τελικές ή μη τελικές).

Παρατηρήστε ακόμη ότι δεν έχει καμμία σημασία με ποιες συμβολοσειρές μπορεί το αυτόματο να φτάσει στις δύο καταστάσεις. Αυτό που έχει σημασία είναι τι κάνει *από εκεί και μετά*.

Αποδεικνύεται (συνέπεια του Θεωρήματος Myhill-Nerode) ότι σε κάθε κανονικό σύνολο αντιστοιχεί ένα μοναδικό DFA (εκτός ισομορφισμού, δηλαδή εκτός μετονομασίας των καταστάσεων) με ελάχιστο αριθμό καταστάσεων και επιπλέον υπάρχει αλγόριθμος που κατασκευάζει το μοναδικό αυτό DFA. Ο αλγόριθμος αυτός βρίσκει συστηματικά όλα τα ζεύγη διακρίσιμων καταστάσεων όπως περιγράφεται παρακάτω:

**Αλγόριθμος 7.1** Αλγόριθμος ελαχιστοποίησης DFA

1. Εξαλείφουμε όλες τις απρόσιτες καταστάσεις.
2. Φτιάχνουμε ένα πίνακα για να συγκρίνουμε κάθε ζεύγος καταστάσεων. Αρχικά σημειώνουμε $X_0$ σε όλα τα ζεύγη όπου η μία κατάσταση είναι τελική ενώ η άλλη δεν είναι.
3. Επαναλαμβάνουμε το παρακάτω για $i := 1, 2, \ldots$ μέχρι που σε κάποια επανάληψη να μην σημειωθεί κανένα νέο ζεύγος στον πίνακα:

   για κάθε ζεύγος $q_k, q_j$ που δεν έχει ήδη σημειωθεί:

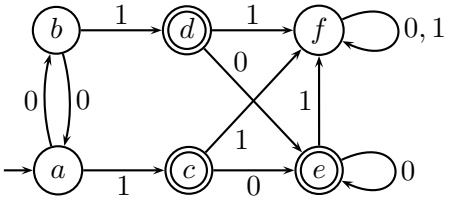
   για κάθε σύμβολο $\sigma \in \Sigma$:

   αν το ζεύγος $(\delta(q_k, \sigma), \delta(q_j, \sigma))$ είναι σημειωμένο με $X_{i-1}$

   τότε σημειώνουμε το ζεύγος $(q_k, q_j)$ με $X_i$.
4. Ζεύγη που δεν έχουν σημειωθεί συγχωνεύονται.

Αποδεικνύεται (η απόδειξη παραλείπεται) ότι ο παραπάνω αλγόριθμος δεν οδηγεί σε αντιφάσεις, δηλαδή αν τα ζεύγη $(q_k, q_j)$ και $(q_k, q_m)$ δεν έχουν σημειωθεί (είναι συγχωνεύσιμα) τότε ούτε το ζεύγος $(q_m, q_j)$ έχει σημειωθεί (επομένως και οι τρεις καταστάσεις συγχωνεύονται σε μία.

*Παράδειγμα* 7.15. Έστω το αυτόματο $M$ που φαίνεται στο σχήμα, το οποίο αποδέχεται την γλώσσα $L = 0^*10^*$.

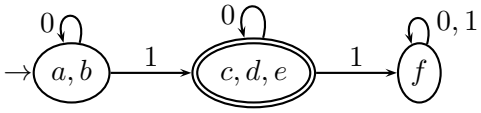



Στον πίνακα οι δείκτες του $X$ δείχνουν σε ποια επανάληψη εγγράφουμε το (οι δείκτες υποδηλώνουν επίσης με πόσα σύμβολα διακρίνονται οι καταστάσεις).

| | | | | | |
|---|---|---|---|---|---|
| $b$ | | | | | |
| ⓒ | $X_0$ | $X_0$ | | | |
| ⓓ | $X_0$ | $X_0$ | | | |
| ⓔ | $X_0$ | $X_0$ | | | |
| $f$ | $X_1$ | $X_1$ | $X_0$ | $X_0$ | $X_0$ |
| | $a$ | $b$ | ⓒ | ⓓ | ⓔ |

Τελικά οι ισοδύναμες καταστάσεις είναι $a \equiv b$, $c \equiv d \equiv e$.

Το ελάχιστο αυτόματο φαίνεται στο παρακάτω σχήμα.



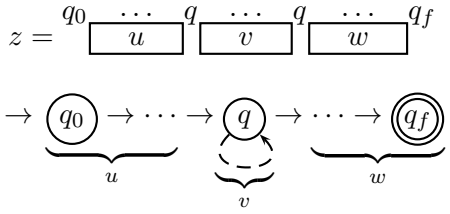

**Επεξηγήσεις για τον παραπάνω αλγόριθμος** Ο παραπάνω αλγόριθμος βασίζεται στην εξής παρατήρηση: αν δύο καταστάσεις $q_i, q_j$ διακρίνονται με $k$ σύμβολα, δηλαδή υπάρχει συμβολοσειρά $w$ μήκους $k$ που τις οδηγεί σε διαφορετικό αποτέλεσμα, τότε αν δύο καταστάσεις έστω $q_m, q_n$ οδηγούν με κάποιο σύμβολο στις $q_i, q_j$, τότε οι $q_m, q_n$ διακρίνονται με $k+1$ σύμβολα. Αυτό ισχύει και αντίστροφα, και επομένως αρκεί ο αλγόριθμος να εξετάζει ένα σύμβολο κάθε φορά: αν υπάρχουν δύο καταστάσεις που διακρίνονται με $k+1$ σύμβολα, τότε υπάρχει ένα σύμβολο που τις οδηγεί σε $k$-διακρίσιμες καταστάσεις.

## 7.6 Pumping Lemma για κανονικά σύνολα

Ένα σημαντικό αποτέλεσμα είναι το Pumping Lemma που χρησιμοποιείται κυρίως για να αποδεικνύουμε ότι συγκεκριμένες γλώσσες δεν είναι κανονικές αλλά και σε αλγορίθμους για να απαντάμε ερωτήσεις σχετικά με πεπερασμένα αυτόματα, όπως αν μια γλώσσα που γίνεται αποδεκτή από ένα πεπερασμένο αυτόματο είναι πεπερασμένη ή άπειρη.

**Pumping Lemma.** Αν μία γλώσσα είναι κανονική τότε την αποδέχεται ένα DFA $M = \{Q, \Sigma, \delta, q_0, F\}$ με κάποιο συγκεκριμένο αριθμό από καταστάσεις, έστω $n$, δηλαδή $|Q| = n$. Έστω μία λέξη $z$ που γίνεται αποδεκτή από το αυτόματο $M$ και η οποία έχει μήκος μεγαλύτερο από $n$.

Καθώς επεξεργαζόμαστε το $z$, το αυτόματο μας $M$ πρέπει να περάσει ξανά από μία κατάσταση, γιατί δεν υπάρχουν περισσότερες από $n$ καταστάσεις (αρχή του περιστερώνα, pigeonhole principle). Έχουμε ότι ένα μονοπάτι που αποδέχεται το $z$ είναι το ακόλουθο:




Το $uvw$ γίνεται αποδεκτό, όπως επίσης και το $uw$, ή το $uvvw$ ή γενικά το $uv^iw$ για οποιοδήποτε $i \in \mathbb{N}$. Δηλαδή το $v$ μπορεί να επαναληφθεί όσες φορές θέλουμε.

*Λήμμα* 7.16 (Pumping Lemma). *Εάν L είναι regular τότε:*
$\exists n \in \mathbb{N}, \forall z \in L$ *με* $|z| \geq n, \exists u, v, w \in \Sigma^*$:
$[z = uvw \ \wedge |uv| \leq n \ \wedge |v| \geq 1 \wedge \forall i \in \mathbb{N} \ (uv^iw \in L)]$

Χρησιμοποιώντας το λήμμα δείχνουμε ότι ένα δοσμένο σύνολο δεν είναι regular.
Η μέθοδος είναι η ακόλουθη:

1. Διαλέγεις τη γλώσσα που θέλεις να αποδείξεις πως δεν είναι regular.
2. Ο αντίπαλος (PL) επιλέγει ένα $n$. Θα πρέπει να μπορείς για οποιοδήποτε πεπερασμένο ακέραιο $n$ διαλέξει, να αποδείξεις ότι η $L$ δεν είναι regular, αλλά από τη στιγμή που ο αντίπαλος έχει διαλέξει ένα $n$ αυτό είναι σταθερό στην απόδειξη.
3. Διαλέγεις ένα string $z$ της $L$ έτσι ώστε $\mid z \mid \geq n$.
4. Ο αντίπαλος (PL) σπάει το $z$ σε $u, v$ και $w$ που ικανοποιούν τους περιορισμούς $|uv| \leq n$ και $|v| \geq 1$.
5. Φτάνεις σε αντίφαση δείχνοντας ότι για κάθε $u, v, w$ που καθορίζονται από τον αντίπαλο, υπάρχει ένα $i$ για το οποίο $uv^iw$ δεν ανήκει στην $L$. Τότε μπορούμε να συμπεράνουμε ότι η $L$ δεν είναι regular. Η επιλογή του $i$ μπορεί να εξαρτάται από τα $n, u, v, w$.

*Παράδειγμα* 7.17. $L = \{a^k b^k \mid k \in \mathbb{N}\} = \{\varepsilon, ab, aabb, aaabbb, \ldots\}$ δεν είναι regular.

1. Υποθέτουμε ότι $L$ είναι regular και χρησιμοποιούμε το Pumping lemma.
2. PL: $\exists n \in \mathbb{N}$
3. Διαλέγουμε $z = a^n b^n$. Εντάξει επιλογή, διότι $z \in L$ , $|z| = 2n \geq n$.
4. PL: $z$ μπορεί να γραφεί: $z = uvw$ με $|uv| \leq n \ \wedge |v| \geq 1$, οπότε αναγκαστικά $v = a^l$ με $l \geq 1$.
5. Διαλέγουμε $i = 2$: $uvvw = a^{n+l} b^n \in L$.

'Ατοπο

Εναλλακτικά:

1. Υποθέτουμε πάλι ότι η $L$ είναι regular.
2. Συνεπώς, αναγνωρίζεται από ένα DFA $M$. Αυτό θα έχει κάποιον πεπερασμένο αριθμό καταστάσεων, έστω $n$.
3. Θέτουμε $z = a^n b^n \in L$.
4. Το DFA $M$ αποδέχεται τη λέξη $z$. Καθώς τη διατρέχει, μέχρι να φτάσει στο $n-$οστό $a$, αναγκαστικά τουλάχιστον μία κατάσταση του επαναλαμβάνεται, έστω στο $i-$οστό και στο $j-$οστό $a$, $i < j$.
5. Έστω $l = j - i$. Τότε, για κάθε $k \geq -1$, το DFA $M$ θα αποδέχεται το string $a^{n+k \cdot l} b^n$. Το substring $a^{k \cdot l}$ θα αντιστοιχεί σε $k$ επαναλήψεις της ίδιας κυκλικής ακολουθίας καταστάσεων του $M$.

'Ατοπο

## 7.7 Διαδραστικό υλικό - Σύνδεσμοι

- Στην σελίδα http://automatonsimulator.com/ μπορείτε να βρείτε έναν online προσομοιωτή αυτομάτων.
- Επίσης, έναν παρόμοιο προσομοιωτή για μια πολύ ενδιαφέρουσα κατηγορία αυτομάτων, τα κυψελωτά αυτόματα (Cellular Automata) μπορείτε να βρείτε στην σελίδα: http://www.jcasim.de/.
- Διαφάνειες από το μάθημα "Introduction to Automata Theory, Languages, and Computation" του Jeff Ullman.
- Το βιβλίο των Al Aho και Jeff Ullman "Foundations of Computer Science" (διαθέσιμο ηλεκτρονικά).
- Παραδείγματα απλών αυτομάτων βρίσκονται εδώ (διαφάνειες 3-8).

## 7.8 Ασκήσεις

1. Κατασκευάστε DFA που να αποδέχεται την γλώσσα όλων των δυαδικών συμβολοσειρών που το τρίτο από δεξιά ψηφίο τους είναι 0. Δείξτε ότι το αυτόματό σας είναι ελάχιστο ή κατασκευάστε ένα ελάχιστο.
2. Δείξτε ότι $\delta(q, xy) = \delta(\delta(q, x), y)$ για κάθε $x$, $y$.
3. Κατασκευάστε DFA που να αναγνωρίζει όλες τις συμβολοσειρές από το αλφάβητο $\Sigma = \{a, b\}$ που τελειώνουν σε $aa$ ή $ab$ ή $ba$.
4. Δώστε DFA που αποδέχονται τις ακόλουθες γλώσσες επί του αλφαβήτού $\{0, 1\}$:
   - $\{w \mid$ κάθε υποακολουθία από 5 διαδοχικά σύμβολα περιέχει τουλάχιστον δύο «0»$\}$
   - $\{w \mid w$: δυαδική αναπαράσταση φυσικού αριθμού $\overline{w}$, $w$ αρχίζει με 1, $\overline{w} \equiv 0 \pmod 5 \}$
   - $\{w \mid$ το 3ο σύμβολο πριν το τελευταίο της συμβολοσειράς είναι «1»$\}$
   - $\{w \mid |w|$ διαιρείται από το 2 ή το 3 ή από αμφότερα$\}$
   - $\{w \in \{1, 2, 3\}^* \mid$ άθροισμα ψηφίων της $w$ είναι $\equiv 0 \pmod 4\}$
5. Για τις ακόλουθες κανονικές εκφράσεις, κατασκευάστε DFA που να αποδέχεται την ίδια γλώσσα:

   (α) $(a + ba)^*(b + ab)^*$

   (β) $0(0 + 1)^*0 + 1(0 + 1)^*1$
6. Δώστε NFA για την $\{w \in \{0, 1\}^* \mid$ υπάρχουν δύο «0» που χωρίζονται από μία συμβολοσειρά μήκους $4i$ για κάποιο $i \geq 0 \}$.

7. Κατασκευάστε DFA ισοδύναμα με τα NFA:

   (α) $(\{p,q,r,s\},\{0,1\},\delta_1,p,\{s\})$

| $\delta_1$ | 0 | 1 |
|---|---|---|
| $p$ | $p,q$ | $p$ |
| $q$ | $r$ | $r$ |
| $r$ | $s$ | $-$ |
| Ⓢ | $s$ | $s$ |

   (β) $(\{p,q,r,s\},\{0,1\},\delta_2,p,\{q,s\})$

| $\delta_2$ | 0 | 1 |
|---|---|---|
| $p$ | $q,s$ | $q$ |
| Ⓠ | $r$ | $q,r$ |
| $r$ | $s$ | $p$ |
| Ⓢ | $-$ | $p$ |

8. Δώστε διάγραμμα καταστάσεων για DFA που αποδέχεται την γλώσσα

$$L = \left\{ w \in \{a,b\}^* \mid \text{κάθε } a \left\langle \begin{array}{l} \text{ακολουθείται αμέσως από ένα } b \\ \text{και έπεται αμέσως ενός } b \end{array} \right. \right\}$$

   Για παράδειγμα, $bababb \in L$, $bbb \in L$, αλλά $ab \notin L$.
   Το DFA σας δεν θα πρέπει να έχει πάνω από τέσσερις καταστάσεις.

9. Κατασκευάστε κανονικές εκφράσεις για κάθε μία από τις γλώσσες:
   (α) Σύνολο συμβολοσειρών του $\{a,b\}^*$ των οποίων το πλήθος των $a$ δεν είναι πολλαπλάσιο του 3.
   (β) Σύνολο συμβολοσειρών του $\{a,b\}^*$ που έχουν άρτιο πλήθος $a$ και τελειώνουν σε $b$.
   (γ) Σύνολο συμβολοσειρών του $\{a,b\}^*$ που δεν έχουν δύο συνεχόμενα $a$.

10. Δύο κανονικές εκφράσεις ονομάζονται ισοδύναμες όταν παράγουν την ίδια γλώσσα. Χρησιμοποιούμε το σύμβολο της ισότητας για να δείξουμε την ισοδυναμία. Έστω $r$ και $s$ κανονικές εκφράσεις. Θεωρήστε την 'εξίσωση' $X = rX + s$. Με την προϋπόθεση ότι η γλώσσα που παράγεται από την $r$, δηλαδή η $L(r)$, δεν περιέχει την κενή συμβολοσειρά $\varepsilon$, βρείτε την λύση για το $X$ και αποδείξτε ότι είναι μοναδική (εκτός ισοδυναμίας). Ποια είναι η λύση αν η $L(r)$ περιέχει την $\varepsilon$; *Υπόδειξη*: Στην τελευταία περίπτωση η λύση δεν είναι μοναδική. Για παράδειγμα, αμφότερες οι $X = 10^*$ και $X = (1+11)0^*$, που αναπαριστούν διαφορετικές γλώσσες, είναι λύσεις της $X = 1 + X0^*$.

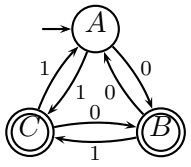
11. Κατασκευάστε FA ισοδύναμα με:

    (α) $10 + (0+11)0^*1$

    (β) $01[((10)^* + 111)^* + 0]^*1$

    (γ) $((0+1)(0+1))^* + ((0+1)(0+1)(0+1))^*$

12. Κατασκευάστε κανονική έκφραση για το:




13. Γράψτε κανονικές εκφράσεις για τις παρακάτω γλώσσες επί του $\{0,1\}$. Δικαιολογήστε την ορθότητα των κανονικών σας εκφράσεων.

(α) Το σύνολο των strings που δεν περιέχουν το 101 ως substring.

(β) Το σύνολο των strings με ίσο πλήθος εμφανίσεων «0» και «1», έτσι ώστε δεν υπάρχει πρόθεμα που να περιέχει δύο περισσότερα «0» από ότι «1», ούτε δύο περισσότερα «1» από ότι «0».

14. Δείξτε τις ακόλουθες ταυτότητες για κανονικές εκφράσεις. Εδώ $r = s$ σημαίνει $L(r) = L(s)$.

    - $r + s = s + r$
    - $(r + s) + t = r + (s + t)$
    - $(rs)t = r(st)$
    - $r(s + t) = rs + rt$
    - $(r + s)t = rt + st$
    - $\emptyset^* = \varepsilon$,
    - $(r^*)^* = r^*$
    - $(\varepsilon + r)^* = r^*$
    - $(r^*s^*)^* = (r + s)^*$

15. Ποια από τα παρακάτω σύνολα είναι κανονικά; Αποδείξτε!

    - $\{a^{3k} \mid k \in \mathbb{N}\}$
    - σύνολο εξισορροπημένων παρενθέσεων
    - $\{a^i b^j \mid 1 \leq i < j\}$
    - $\{w \mid$ η $w$ περιέχει λιγότερα $a$ από $b\}$
    - $\{a^i b^j a^j \mid i, j \geq 1\}$
    - $\{w \in \{1\}^* \mid |w|$ είναι πρώτος αριθμός$\}$
    - $\{uvw \in \{a, b\}^* \mid$ η $uv$ είναι παλίνδρομη αρτίου μήκους$\}$
    - $\{w \in \{a, b\}^* \mid$ η $w$ δεν έχει 4 συνεχόμενα $b\}$
    - $\{uvw \in \{a, b\}^* \mid$ η $uw$ είναι παλίνδρομη αρτίου μήκους$\}$
    - $\{w \in \{a, b\}^* \mid$ η $w$ είναι παλίνδρομη$\}$

16. Δείξτε ότι η γλώσσα $\{0^p \mid p$ είναι πρώτος $\}$ δεν είναι κανονική.

# Βιβλιογραφία


[1] Martin D. Davis, Ron Sigal, Elaine J. Wayuker, "Computability, Complexity, and Languages", 2nd edition, Academic Press Professional, Inc. San Diego, CA, USA, 1994.

[2] M. Harrison. "Introduction to Switching and Automata Theory", McGraw-Hill Book Company, New York, 1965.

[3] J.E. Hopcroft and J.D. Ullman. "Introduction to Automata Theory, Languages and Computation", Addison Wesley Longman, 2007.

[4] D. C. Kozen. "Automata and Computability" (Undergraduate Texts in Computer Science), Springer-Verlag New York, Inc. Secaucus, NJ, USA, 1997.

[5] M. Sipser. "Introduction to the Theory of Computation", International Thomson Publishing, 1996.

# Κεφάλαιο 8

# Τυπικές Γραμματικές και Άλλα Αυτόματα

## 8.1 Εισαγωγή

Στο κεφάλαιο αυτό θα ασχοληθούμε με μη κανονικές γλώσσες και γραμματικές, καθώς και τα αντίστοιχα αυτόματα που τις αναγνωρίζουν. Πιο συγκεκριμένα για: γλώσσες και γραμματικές *χωρίς συμφραζόμενα (context free)*, γλώσσες και γραμματικές *με συμφραζόμενα (context sensitive)*, και *αναδρομικά αριθμήσιμες (recursively enumerable)* γλώσσες και *γενικές γραμματικές*.

## 8.2 Κανονικές Γραμματικές

Στις *κανονικές γραμματικές* όλοι οι κανόνες παραγωγής είναι της μορφής:

1. δεξιογραμμικοί (rightlinear): $A \to wB$, $A \to w$, όπου $w \in T^*$, ή
2. αριστερογραμμικοί (leftlinear): $A \to Bw$, $A \to w$, όπου $w \in T^*$.

*Θεώρημα* 8.1. *Τα κανονικά σύνολα παράγονται από κανονικές (δεξιο- ή αριστερογραμμικές γραμματικές)*

*Ιδέα απόδειξης.* Περιοριζόμαστε σε δεξιογραμμικές γραμματικές:

- Έστω $G_1$ δεξιογραμμική. Κατασκευάζουμε ισοδύναμη γραμματική $G_2$ σε *μεταβατική μορφή*, δηλαδή σε μορφή κανόνων $A \to aB$ ή $A \to a$ (ή $\varepsilon$). Ιδέα για μετατροπή: $A \to a_1 \dots a_n B$ αντικαθίσταται από τους κανόνες: $A \to a_1 A_1$, $A_1 \to a_2 A_2$, $\dots$, $A_{n-1} \to a_n B$, όπου $_1, \dots, A_{n-1}$ νέα μη τερματικά σύμβολα.

Μετά κατασκευάζουμε ένα NFA$_\varepsilon$ με $Q = V \cup \{\varepsilon'\}$, $\Sigma =$ , $q_0 = S$, $F = \{\varepsilon'\}$ και:

$$B \in \delta(A, a) \text{ ανν } (A \to aB) \in P$$
$$\varepsilon' \in \delta(A, a) \text{ ανν } (A \to a) \in P$$
$$\varepsilon' \in \delta(A, \varepsilon) \text{ ανν } (A \to \varepsilon) \in P$$

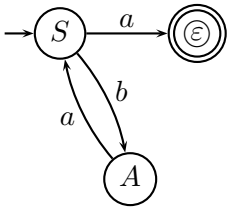
- Έστω DFA. Κατασκευάζουμε γραμματική με $V = Q$, $T = \Sigma$, $S = q_0$ και παραγωγές:

$$(q_i \to aq_j) \in P \text{ ανν } \delta(q_i, a) = q_j, \qquad (q_i \to a) \in P \text{ ανν } \delta(q_i, a) \in F$$

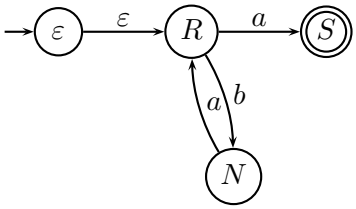
□

*Παράδειγμα* 8.2. $(ba)^*a$ δεξιογραμμικά $S \to baS \mid a$
αριστερογραμμικά $S \to Ra,\ R \to Rba \mid \varepsilon$

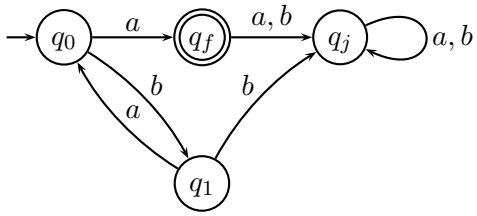
• Για να κατασκευάσουμε F.A. που αποδέχεται την κανονική γλώσσα που παράγεται από τη δεξιογραμμική γραμματική ($S \to baS \mid a$) πρώτα την απλοποιούμε σε *μεταβατική μορφή* ($S \to bA \mid a, A \to aS$) και μετά:




• Για να κατασκευάσουμε F.A. που αποδέχεται την κανονική γλώσσα που παράγεται από την αριστερογραμμική γραμματική ($S \to Ra,\ R \to Rba \mid \varepsilon$) πρώτα απλοποιείται σε μεταβατική μορφή ($S \to Ra, R \to Na \mid \varepsilon, N \to Rb$) και μετά:



• Κατασκευή δεξιογραμμικής γραμματικής από αυτόματο.
Έστω $M$:




$$\begin{cases} q_0 \to a \mid bq_1 \mid aq_f \\ q_1 \to aq_0 \mid bq_j \\ q_f \to aq_j \mid bq_j \\ q_j \to aq_j \mid bq_j \end{cases}$$

Απαλοιφή σύμβολων που δεν παράγουν τίποτα (non yielding) $q_j, q_f$ και παραγωγών που σχετίζονται με αυτά. Τότε έχουμε:

$$\begin{cases} q_0 \to a \mid bq_1 \\ q_1 \to aq_0 \end{cases}$$

• Κατασκευή αριστερογραμμικής γραμματικής από αυτόματο. ($S := q_f$)

$$\begin{cases} q_f \to q_0a \\ q_0 \to q_1a \mid \varepsilon \\ q_1 \to q_0b \\ q_j \to q_jb \mid q_ja \mid q_fb \mid q_fa \mid q_1b \end{cases}$$

Απαλοιφή συμβόλων στα οποία δεν φθάνουμε ποτέ (unreachable) $q_j$ καθώς και παραγωγών που σχετίζονται με αυτά. Τότε έχουμε:

$$\begin{cases} q_f \to q_0a \\ q_0 \to q_1a \mid \varepsilon \\ q_1 \to q_0b \end{cases}$$

## 8.3 Γραμματικές χωρίς συμφραζόμενα και αυτόματα στοίβας

Όπως είδαμε στην ενότητα περί ιεραρχίας Chomsky, μια γραμματική $G = (V, T, P, S)$ λέγεται *χωρίς συμφραζόμενα (context free - c.f.)* αν οι κανόνες παραγωγής στο $P$ έχουν τη μορφή $A \to \beta$ όπου $\beta \in (V \cup T)^*$ και $A \in V$. Παρακάτω δίνουμε μερικά παραδείγματα γραμματικών χωρίς συμφραζόμενα:

*Παράδειγμα* 8.3. Έστω η γραμματική

$$G_1: \quad V = \{S\},\ T = \{a, b\},\ P = \{S \to \varepsilon, S \to aSb\}$$

Μια πιθανή ακολουθία παραγωγών είναι η:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$$

Η γλώσσα που παράγεται από την $G_1$ είναι η $L(G_1) = \{a^n b^n | n \in \mathbb{N}^*\}$.

*Παράδειγμα* 8.4. $G_2 = (V, T, P, S)$ με $T = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, *\}$, $V = \{S\}$ και το $P$ περιέχει τους κανόνες:

$$S \to S + S, \quad S \to S * S, \quad S \to 0|1|2|3|4|5|6|7|8|9$$

Τόσο στο πιο πάνω παράδειγμα, όσο και στο παράδειγμα 8.3, είναι εύκολο να καταλάβουμε ποιες είναι οι γλώσσες που παράγονται από τις γραμματικές $G_1, G_2$. Εν γένει, το να βρούμε την $L(G)$ δεν είναι πάντοτε τόσο προφανές, όπως φαίνεται από το επόμενο παράδειγμα.

*Παράδειγμα* 8.5. $G_3 = (V, T, P, S), V = \{S, A, B\}, T = \{a, b\}$ και το $P$ περιέχει τους κανόνες:

$$S \to aB \mid bA, \quad A \to a \mid aS \mid bAA, \quad B \to b \mid bS \mid aBB$$

Μια πιθανή ακολουθία παραγωγών της πιο πάνω γραμματική είναι:

$$S \Rightarrow aB \Rightarrow abS \Rightarrow abbA \Rightarrow abba$$

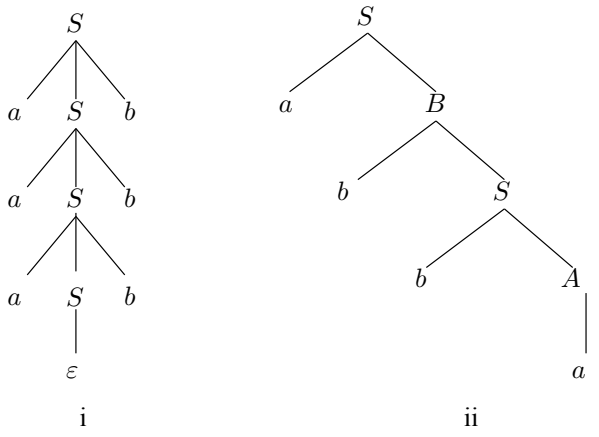
Αν και δεν είναι προφανές, $L(G_3) = \{w \in T^+ \mid w$ έχει ίσο αριθμό $a$ και $b\}$

### 8.3.1 Συντακτικά δένδρα

Ένας άλλος τρόπος να να περιγράψουμε πως με μια γραμματική προκύπτει μια συμβολοακολουθία, εκτός από τις ακολουθίες παραγωγών μέσω των "⇒", είναι τα *συντακτικά δένδρα* (parse trees).

*Ορισμός* 8.6. Έστω $G = \{V, T, P, S\}$ μια c.f. γραμματική. Ένα δένδρο είναι συντακτικό δένδρο της $G$ αν

1. Κάθε κόμβος του δένδρου έχει μια *επιγραφή*, η οποία είναι ένα σύμβολο στο $V \cup T \cup \{\varepsilon\}$.
2. Η επιγραφή της ρίζας είναι το $S$.
3. Αν ένας κόμβος είναι εσωτερικός και έχει επιγραφή $A$, τότε το πρέπει να είναι στοιχείο του $V$.
4. Αν ο κόμβος $n$ έχει επιγραφή $A$ και οι κόμβοι $n_1, n_2, \ldots, n_k$ είναι παιδιά του $n$, σε διάταξη από αριστερά προς τα δεξιά, με επιγραφές $X_1, X_2, \ldots, X_k$ αντίστοιχα, τότε ο $A \to X_1 X_2 \ldots X_k$ πρέπει να είναι κανόνας παραγωγής στο $P$.




Σχήμα 8.1: Παραδείγματα συντακτικών δένδρων

5. Αν ένας κόμβος έχει επιγραφή $\varepsilon$, τότε είναι φύλλο και είναι το μοναδικό παιδί του γονέα του.

Για παράδειγμα, τα συντακτικά δένδρα των ακολουθιών παραγωγών των παραδειγμάτων 8.3 και 8.5 φαίνονται στο σχήμα 8.1.

Ονομάζουμε *φύλλωμα (leaf string)* τη συμβολοακολουθία που προκύπτει από τα φύλλα ενός δένδρου, αν τα διατρέξουμε από το αριστερότερο προς το δεξιότερο. Ορίζουμε ως *$A$-δένδρο* ενός συντακτικού δέντρου ένα υποδέντρο το οποίο περιλαμβάνει ως ρίζα έναν κόμβο με επιγραφή $A$ καθώς και όλους τους απογόνους αυτού του κόμβου και τις μεταξύ τους ακμές.

*Θεώρημα* 8.7. *Έστω $G(V, T, P, S)$ μια c.f. γραμματική. Τότε $S \overset{*}{\Rightarrow} \alpha$ ανν υπάρχει συντακτικό δένδρο με φύλλωμα το $\alpha$.*

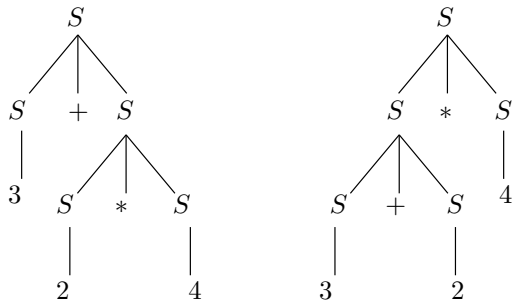
Η απόδειξη της κατεύθυνσης "$\Leftarrow$" γίνεται με επαγωγή ως προς τον αριθμό των εσωτερικών κόμβων, ενώ της "$\Rightarrow$" με επαγωγή ως προς των αριθμό των βημάτων της ακολουθίας παραγωγών (άσκηση).

**Διφορούμενες γραμματικές (ambiguous grammars)**

*Ορισμός* 8.8. Μια γραμματική $G$ ονομάζεται *διφορούμενη (ambiguous)* αν υπάρχουν δύο συντακτικά δένδρα που να έχουν ως φύλλωμα ένα $w \in L(G)$.

Για παράδειγμα, η συμβολοακολουθία $3 + 2 * 4$ ανήκει στην $L(G_2)$, του παραδείγματος 8.4, σελίδα 122 και υπάρχουν δύο συντακτικά δένδρα που αντιστοιχούν σε αυτήν, όπως φαίνεται στο σχήμα 8.2. Παρ'όλα αυτά, μπορούμε να κατασκευάσουμε μια γραμματική που είναι ισοδύναμη με την $G_2$ και η οποία *δεν* είναι διφορούμενη (άσκηση).

Υπάρχουν όμως και γλώσσες χωρίς συμφραζόμενα, για τις οποίες όλες οι γραμματικές που τις παράγουν είναι αναγκαστικά διφορούμενες:




Σχήμα 8.2: Συντακτικά δένδρα διφορούμενης γραμματικής

*Ορισμός* 8.9. Μια γλώσσα χωρίς συμφραζόμενα ονομάζεται *εγγενώς διφορούμενη* (inherently ambiguous) αν όλες οι γραμματικές που την παράγουν είναι διφορούμενες.

*Παράδειγμα* 8.10. Η γλώσσα χωρίς συμφραζόμενα $\{a^i b^j c^k \mid i = j \vee j = k\}$ είναι εγγενώς διφορούμενη.

### 8.3.2 Απλοποίηση και κανονικές μορφές

Είναι δυνατόν να απλοποιήσουμε μία γραμματική χωρίς συμφραζόμενα ως εξής: Κρατάμε μόνον τα σύμβολα που χρειάζονται (για παράδειγμα εξαλείφουμε όλα τα μη τερματικά που δεν παράγουν συμβολοακολουθίες με τερματικά και όλα τα σύμβολα που δεν μπορούν να εμφανισθούν σε καμία παραγωγή που ξεκινάει από το αρχικό σύμβολο). Επίσης, μπορούμε να εξαλείψουμε κανόνες του τύπου $A \to B$ (unit productions).

Επιπλέον, αν το $\varepsilon$ δεν ανήκει στην $L$, δεν χρειάζονται κανόνες παραγωγής της μορφής $A \to \varepsilon$ ($\varepsilon$-productions). Μάλιστα, αν το $\varepsilon$ δεν ανήκει στην $L$, μπορούμε να κατασκευάσουμε την $G$ με τέτοιο τρόπο ούτως ώστε κάθε κανόνας παραγωγής να έχει είτε τη μορφή $A \to BC$ είτε την $A \to a$, όπου οι $A$, $B$ και $C$ είναι μεταβλητές και το $a$ είναι τερματικό σύμβολο. Εναλλακτικά, μπορούμε να μετατρέψουμε κάθε κανόνα παραγωγής του $G$ στη μορφή $A \to a$, όπου το $\alpha$ είναι μια συμβολοακολουθία από μεταβλητές (μπορεί και κενή). Αυτές οι δύο ειδικές μορφές ονομάζονται *κανονική μορφή Chomsky* και *κανονική μορφή Greibach*, οπότε έχουμε τα παρακάτω δύο θεωρήματα κανονικών μορφών:

*Θεώρημα* 8.11 (Κανονικής μορφής Chomsky, CNF). *Κάθε c.f. γλώσσα χωρίς το $\varepsilon$ παράγεται από μια γραμματική στην οποία όλες οι παραγωγές είναι της μορφής $A \to BC$ ή $A \to a$. όπου , , $C$ μεταβλητές και $a$ τερματικό.*

*Θεώρημα* 8.12 (Κανονικής μορφής Greibach, GNF). *Κάθε c.f. γλώσσα χωρίς το $\varepsilon$ παράγεται από μια γραμματική στην οποία όλες οι παραγωγές είναι της μορφής $A \to a$, όπου $\alpha \in V^*$, $a \in T$.*

Από γραμματικές στις παραπάνω κανονικές μορφές, προκύπτουν σχετικά απλούστερα συντακτικά δένδρα: για την CNF τα συντακτικά δένδρα έχουν πάντοτε διακλάδωση βαθμού δύο αν προκύπτουν μεταβλητές, αλλιώς από μία μεταβλητή προκύπτει ένα μόνον φύλλο (τερματικό σύμβολο), ενώ για την GNF, τα αριστερά παιδιά κάθε κόμβου είναι πάντοτε τερματικά σύμβολα.

Μπορούμε να εκμεταλλευτούμε αυτές τις ιδιότητες των συντακτικών δένδρων για να επιλύσουμε ταχύτερα ορισμένα προβλήματα όπως αν κάποια συμβολοσειρά $x$ ανήκει στην γλώσσα που παράγει μία γραμματική: Δεδομένης γραμματικής χωρίς συμφραζόμενα $G$, όχι απαραίτητα σε κανονική μορφή, υπάρχει μηχανιστικός αλγόριθμος ο οποίος για οποιαδήποτε συμβολοσειρά $x$ αποκρίνεται αν $x \in L(G)$ ή όχι. Π.χ. αν συστηματικά κατασκευάσουμε όλες τις παραγόμενες συμβολοσειρές κατά αύξουσα σειρά μήκους, τότε μπορούμε να αποφασίσουμε εάν $x \in L(G)$. Ο αλγόριθμος όμως είναι εκθετικού χρόνου ως προς το μήκος της συμβολοσειράς εισόδου. Αν όμως η γραμματική δίνεται σε κανονική μορφή Chomsky, τότε υπάρχει ταχύτερος αλγόριθμος, πολυπλοκότητας $O(|x|^3)$, ο λεγόμενος αλγόριθμος CYK (από τους Cocke, Younger, Kasami).

```
function CYK(x: string): boolean  (* assumes Chomsky n.f.  *)
begin  n := |x|
   for i := 1 to n do
      V_i^1 := {A | (A → a) ∈ P ∧ (x)_i = a};
   for j := 2 to n do
      for i := 1 to n − j + 1 do
```

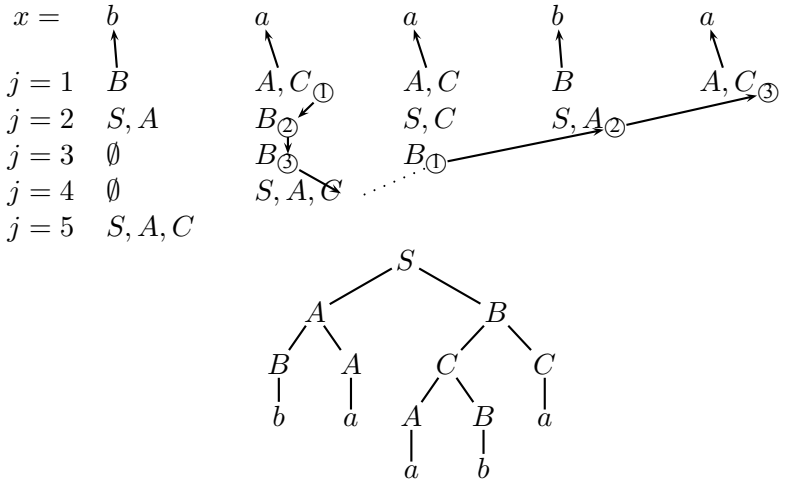
**begin** $V_i^j := \emptyset$;
**for** $k := 1$ **to** $j - 1$ **do**
$V_i^j := V_i^j \cup \{A \mid (A \to BC) \in P \wedge B \in V_i^k \wedge C \in V_{i+k}^{j-k}\}$
**end** ;
CYK := $S \in V_1^n$
**end**

Ο αλγόριθμος ουσιαστικά ελέγχει όλα τα δυνατά συντακτικά δένδρα, αρχίζοντας από τα τερματικά σύμβολα στα φύλλα και αποδίδοντας σε αυτά πιθανά μη τερματικά σύμβολα που τα παραγάγουν. Ο αλγόριθμος συνεχίζει αποδίδοντας πιθανά μη τερματικά σύμβολα σε κάθε υποσυμβολοσειρά της συμβολοσειράς εισόδου και τέλος ελέγχει αν στην συμβολοσειρά εισόδου έχει αποδοθεί το αξίωμα $S$.

*Παράδειγμα* 8.13. Έστω γραμματική σε κανονική μορφή Chomsky:

$$S \to AB \mid BC, \quad A \to BA \mid a, \quad B \to CC \mid b, \quad C \to AB \mid a$$

Στο σχήμα 8.3 δίνουμε την εκτέλεση του αλγορίθμου και το αντίστοιχο συντακτικό δένδρο για είσοδο $x = baaba$. Με τα βέλη, δείχνουμε την ακολουθία υπολογισμού για $j = 4$ (μήκος substring) και για το substring $(x)_{2..5}$ (αρχίζει, δηλαδή, από την θέση $i = 2$): Μέσα σε κύκλο και με τον ίδιο αριθμό συμβολίζουμε τα δύο substrings (συνολικού μήκους 4) που συνδυάζονται για να δώσουν το $(x)_{2..5}$.




Σχήμα 8.3: Εκτέλεση αλγορίθμου CYK

### 8.3.3 Αυτόματα στοίβας

Ένα *αυτόματο στοίβας* (push down automaton ή για συντομία PDA) αποτελείται από μία ταινία εισόδου, αλλά επιπλέον, σε σχέση με τα FA, έχει και μία *στοίβα* (μη φραγμένη σε μέγεθος μνήμη,

αλλά με περιορισμένες δυνατότητες πρόσβασης σε αυτήν). Η πρόσβαση στην στοίβα γίνεται μόνον στην κορυφή αυτής με τις εξής δύο λειτουργίες:

1. push: Τοποθετεί ένα στοιχείο που δίνεται στην κορυφή της στοίβας.
2. pop: Αφαιρεί ένα στοιχείο για χρήση από την κορυφή της στοίβας.

Είναι προφανές ότι αν κάποιο στοιχείο βρίσκεται βαθιά μέσα στην στοίβα, δηλαδή αφού έχει μπει στην στοίβα με push, έχουν επίσης μπει με push άλλα στοιχεία πάνω από αυτό, προκειμένου να έχουμε πρόσβαση σε αυτό θα πρέπει να κάνουμε pop σε όλα τα στοιχεία που είναι από πάνω του.
Θεωρήστε την γλώσσα $L = \{wcw^R \mid w \in (0+1)^*\}$. Για παράδειγμα $110c011 \in L$. Να πώς μπορούμε να αναγνωρίσουμε την παραπάνω γλώσσα με ένα PDA:

1. push($a$) στην στοίβα για κάθε 0 που συναντάς στην είσοδο,
   push($b$) στην στοίβα για κάθε 1 που συναντάς στην είσοδο,
   μέχρι να συναντήσεις το $c$
2. διάβασε το $c$,
   κάνε pop τα στοιχεία της στοίβας και διάβαζε παράλληλα την είσοδο,
   αποδέξου αν υπάρχει συμφωνία των στοιχείων εισόδου με τα στοιχεία της στοίβας ($a$ με 0, $b$ με 1).

Δίνουμε παρακάτω τον τυπικό ορισμό:

*Ορισμός* 8.14. Ένα αυτόματο στοίβας (push down automaton ή για συντομία PDA) είναι μία πλειάδα $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, όπου:

- $Q$: πεπερασμένο σύνολο καταστάσεων,
- $\Sigma$: αλφάβητο εισόδου,
- $\Gamma$: αλφάβητο στοίβας,
- $\delta$: $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to Pow(Q \times \Gamma^*)$ (πεπερασμένα υποσύνολα), η συνάρτηση μετάβασης (επιτρέπονται $\varepsilon$-κινήσεις και μη ντετερμινισμός),
- $q_0 \in Q$: αρχική κατάσταση,
- $Z_0 \in \Gamma$: αρχικό σύμβολο στην στοίβα,
- $F \subseteq Q$: τελικές καταστάσεις.

Υπάρχουν δύο είδη PDA ως προς το αποδέχεσθαι:

1. αποδέξου όταν βρίσκεσαι σε τελική κατάσταση αφού έχεις διαβάσει όλη την ταινία εισόδου, ανεξάρτητα από το τι υπάρχει στην στοίβα, ή,

2. αποδέξου όταν η στοίβα είναι άδεια αφού έχεις διαβάσει όλη την ταινία εισόδου, ανεξάρτητα από την κατάσταση στην οποία ευρίσκεσαι (σύμβαση: $F = \emptyset$).

Αντίστοιχα ορίζουμε και την γλώσσα που αποδέχεται ένα PDA:

1. Γλώσσα που αποδέχεται σε τελική κατάσταση $L_f(M)$
2. Γλώσσα που αποδέχεται με κενή στοίβα $L_e(M)$

Προκειμένου να γίνει αποδεκτή η $L_1 = \{ww^R \mid w \in (0+1)^*\}$ χωρίς το σημάδι $c$ στην μέση της συμβολοσειράς χρειαζόμαστε απαραιτήτως ένα μη ντετερμινιστικό PDA. Τα μη ντετερμινιστικά PDA είναι γνησίως πιο ισχυρά από τα ντετερμινιστικά.

Τελικά ισχύει το παρακάτω θεώρημα:

*Θεώρημα* 8.15. *Τα παρακάτω είναι ισοδύναμα για μία γλώσσα $L$:*

1. *$L = L_f(M_2)$, $M_2$ είναι PDA.*
2. *$L = L_e(M_1)$, $M_1$ είναι PDA.*
3. *Η $L$ είναι γλώσσα χωρίς συμφραζόμενα (context free).*

Παραλείπεται η λεπτομερής απόδειξη.

## 8.4 Γενικές γραμματικές

Τύπου 0: *γενικές γραμματικές (general or unrestricted grammars)*, semi-Thue, phrase structure
Παραγωγές: $\alpha \to \beta$, με $\alpha \neq \varepsilon$

*Παράδειγμα* 8.16. $L = \{a^{2^n} \mid n \in N\}$
$S \to AaCB$
$CB \to E \mid DB$
$aE \to Ea$
$AE \to \varepsilon$
$aD \to Da$
$AD \to AC$
$Ca \to aaC$

*Θεώρημα* 8.17. *Τα ακόλουθα είναι ισοδύναμα:*

1. *η $L$ γίνεται αποδεκτή από μία Turing μηχανή (βλέπε κεφάλαιο* **??***, ενότητα* **??***)*
2. *$L = L(G)$, όπου $G$ είναι γενική γραμματική*

Χωρίς απόδειξη. Μία τέτοια γλώσσα λέγεται και *αναδρομικά αριθμήσιμη* (recursively enumerable)· βλέπε ορισμό 11.3 στην σελίδα 163.

## 8.5 Γραμματικές με συμφραζόμενα

Τύπου 1: *γραμματικές με συμφραζόμενα (context sensitive grammars, c.s.)*
Παραγωγές: $\alpha \to \beta$, με $\alpha \neq \varepsilon$, $|\alpha| \leq |\beta|$ (noncontracting grammar, non-decreasing, not $\varepsilon$ string)
Η ονομασία context sensitive οφείλεται στην παρακάτω εναλλακτική περιγραφή αυτών των γραμματικών: Κάθε c.s. γραμματική μπορεί να τεθεί σε κανονική μορφή στην οποία όλοι κανόνες παραγωγής είναι της μορφής:

$$\underset{\substack{\nwarrow \quad \nearrow \\ context}}{\alpha_1 A \alpha_2} \to \alpha_1 \beta \alpha_2, \qquad \text{όπου } A\text{: μη τερματικό και } \beta \neq \varepsilon$$

*Παράδειγμα* 8.18. $1^n 0^n 1^n$. C.s. γραμματική:
$S \to 1Z1$
$Z \to 0 \mid 1Z0A$
$A0 \to 0A$
$A1 \to 11$

'Αλλα παραδείγματα τέτοιων γλωσσών: $\{a^n b^n c^n\}$, $\{a^i b^j c^k \mid 0 \leq i \leq j \leq k\}$, $\{ww \mid w \in \Sigma^*\}$, $0^n 1^n 0^n 1^n$.

*Γραμμικά φραγμένο αυτόματο (Linearly bounded automaton (LBA))*: Είναι μία μη-ντετερμινιστική μηχανή Turing (T.M.) της οποίας η κεφαλή είναι περιορισμένη να κινείται μόνον στο τμήμα της ταινίας που περιέχει την είσοδο.

*Θεώρημα* 8.19. *Τα ακόλουθα είναι ισοδύναμα ($L$ χωρίς $\varepsilon$):*

1. *η $L$ γίνεται αποδεκτή από LBA*
2. *η $L$ είναι c.s.*

Χωρίς απόδειξη.

Ένα ανοιχτό ερώτημα είναι το εξής: Είναι ισοδύναμη η ντετερμινιστική και η μη ντετερμινιστική εκδοχή του LBA;

## 8.6 Διαδραστικό υλικό - Σύνδεσμοι

- Το JFLAP, που μπορείτε να βρείτε εδώ: http://www.jflap.org/ είναι ένα πλήρες πακέτο για την προσομοίωση και υπολογισμό αυτομάτων και γραμματικών. Μπορεί να εκτελέσει όλους τους αλγόριθμους που αναφέρθηκαν στα Κεφάλαια 8 και 9, όπως μετατροπή NFA σε DFA, ελαχιστοποίηση DFA, μετατροπή DFA σε κανονική έκφραση/γραμματική κλπ.

## 8.7 Ασκήσεις

1. Έστω η γραμματική χωρίς συμφραζόμενα $G = (\{S, A, B\}, \{a, b\}, P, S)$, με κανόνες παραγωγής $P$:
   $S \to ABA$ $A \to a|bb$ $B \to bS|\varepsilon$
   Ποια γλώσσα αναγνωρίζει η γραμματική $G$;

2. Αποδείξτε ή ανταποδείξτε τις ιδιότητες κλειστότητας της κλάσης των γλωσσών με συμφραζόμενα (context-sensitive) ως προς ένωση, παράθεση, τομή και άστρο του Kleene. Τι συμπέρασμα προκύπτει για την πράξη του συμπληρώματος;

3. Βρείτε γραμματική χωρίς συμφραζόμενα που να υπολογίζει την γλώσσα
   $$L = \{x \in \{a, b\}^* \mid x = x^R\}$$

4. Κατατάξτε τις παρακάτω γλώσσες στην Ιεραρχία Chomsky με όσο μεγαλύτερη ακρίβεια μπορείτε (με απόδειξη ή εξήγηση, όσο καλύτερα μπορείτε):
   (i) $\{0^k 1^{k+1} 0^k \mid k \in \mathbb{N}\}$
   (i) $\{0^{3k+1} \mid k \in \mathbb{N}\}$
   (ii) $\{a^k b^m c^n \mid k, m, n \in \mathbb{N}, k < m \leq n\}$
   (i) $\{w \in \{a, b\}^* \mid w$περιέχει‘$1101'\}$

5. Δείξτε ότι κάθε συμβολοσειρά στην γλώσσα $L$ έχει περισσότερα $a$ από $b$, όπου $L$ η γλώσσα που παράγεται από την γραμματική:
   $$S \to Sa|bSS|SSb|SbS|a$$

6. Δώστε περιγραφή υψηλού επιπέδου (όσο καλύτερα μπορείτε) ενός LBA για τις παρακάτω γλώσσες:
   (i) $L = \{a^p : p$ είναι δύναμη του 2 $\}$.

(ii) $L = \{a^p : p \text{ είναι πρώτος αριθμός}\}$.

7. Δείξτε ότι η γραμματική χωρίς συμφραζόμενα $G$ που παράγεται από τους κανόνες:
$S \to aA$ $A \to BA|a$ $B \to bS|cS$
είναι μη-διφορούμενη.

8. Κατατάξτε τις παρακάτω γλώσσες σε κλάσεις της Ιεραρχίας Chomsky (στην μικρότερη κλάση που ανήκουν) και αποδείξτε τον ισχυρισμό σας δίνοντας κατάλληλα αυτόματα και γραμματικές (όσο καλύτερα μπορείτε). Για όσες θεωρείτε ότι δεν είναι κανονικές, αποδείξτε το γεγονός αυτό με χρήση του Pumping Lemma.

$L_1 = \{w \in \{0,1\}^* \mid w = u100u^R, \quad u \in \{0,1\}^*\}$, όπου $u^R$ η ανάστροφη συμβολοσειρά της $u$.

$L_2 = \{w \in \{0,1\}^* \mid w$ παριστάνει δυαδικό αριθμό που είναι πολλαπλάσιο του 2 αλλά όχι του 4 $\}$

$L_3 = \{w \in \{a,b,c\}^* \mid w = a^k b^m c^l, \quad k,m,l \in \mathbb{N}, k < m < l\}$

9. Κατασκευάστε αυτόματο στοίβας (PDA) που να αποδέχεται την γλώσσα:

$$L = \{a^i b^j \mid 2i \neq 3j\}$$

10. Κατασκευάστε αυτόματο στοίβας (PDA) που να αποδέχεται την γλώσσα που παράγεται από την γραμματική $G$ με κανόνες παραγωγής:

$$S \to \varepsilon|SS|aSb$$

# Βιβλιογραφία


[1] Martin D. Davis, Ron Sigal, Elaine J. Wayuker, "Computability, Complexity, and Languages", 2nd edition, Academic Press Professional, Inc. San Diego, CA, USA, 1994.

[2] M. Harrison. "Introduction to Switching and Automata Theory", McGraw-Hill Book Company, New York, 1965.

[3] J.E. Hopcroft and J.D. Ullman. "Introduction to Automata Theory, Languages and Computation", Addison Wesley Longman, 2007.

[4] D. C. Kozen. "Automata and Computability" (Undergraduate Texts in Computer Science), Springer-Verlag New York, Inc. Secaucus, NJ, USA, 1997.

[5] M. Sipser. "Introduction to the Theory of Computation", International Thomson Publishing, 1996.

# Κεφάλαιο 9

# Αλγεβρικές Δομές και Αριθμοθεωρία

## 9.1 Εισαγωγή

Θα παρουσιάσουμε κάποια στοιχεία από Θεωρία Αριθμών και ελάχιστα από Θεωρία Ομάδων. Οι γνώσεις αυτές είναι οι ελάχιστες απαραίτητες για την κατανόηση των κρυπτογραφικών συστημάτων. Η Θεωρία Αριθμών ασχολείται κυρίως με τις ιδιότητες των φυσικών αριθμών (ή αλλιώς θετικών ακεραίων): $1, 2, 3, \ldots$

Θα αρχίσουμε παρουσιάζοντας τα βασικά (εισαγωγικά) στοιχεία της θεωρίας Αριθμών, δηλαδή την έννοια της διαιρετότητας, των πρώτων αριθμών, του μεγίστου κοινού διαιρέτη (μκδ) και της συνάρτησης Euler ($\phi$).

## 9.2 Διαιρετότητα

### 9.2.1 Διαιρετότητα

*Ορισμός* 9.1. Αν $a, b \in \mathbf{Z}$ (ακέραιοι) λέμε ο "a διαιρεί τον b" αν $\exists c \in \mathbf{Z} : b = ca$ και συμβολίζουμε με $a \mid b$.

**Παρατηρήσεις:**

- Η άρνηση του $a \mid b$ συμβολίζεται με $a \nmid b$.
- Ο $a$ λέγεται διαιρέτης του $b$ και ο $b$ πολλαπλάσιο του $a$.
- $a$ γνήσιος διαιρέτης του $b$: $a \mid b \ \wedge \ 0 < a < b$.
- $a$ μη τετριμένος διαιρέτης του $b$: $a \mid b \ \wedge \ a \neq 1 \ \wedge \ a \neq b$.

*Πρόταση* 9.1. Έστω $a, b, c, x, y \in \mathbf{Z}$

1. $a \mid 0$.

2. Κάθε αριθμός μεγαλύτερος του 1 έχει τουλάχιστον δύο διαιρέτες : το 1 και τον εαυτό του.
3. $a \mid b \ \Rightarrow \ a \mid (bc)$.
4. $a \mid b \ \wedge \ b \mid c \ \Rightarrow \ a \mid c$.
5. $a \mid b \wedge b \mid a \ \Rightarrow \ |a| = |b|$.
6. $a \mid b \wedge a \mid c \ \Rightarrow \ a \mid (b + c)$.
7. $a \mid b \wedge a \mid c \ \Rightarrow \ a \mid (bx + cy)$
8. $a \mid b \wedge b > 0 \ \Rightarrow \ a \leq b$

### 9.2.2 Πρώτοι Αριθμοί

*Ορισμός* 9.2. Πρώτος αριθμός λέγεται ένας ακέραιος μεγαλύτερος του 1 που δεν έχει άλλους διαιρέτες εκτός από το 1 και τον εαυτό του, ειδάλλως λέγεται σύνθετος π.χ.

- $2, 3, 5, \ldots, 1997, \ldots, 6469, \ldots$
- $(333+10^{793})10^{791}+1$ (με 1585 ψηφία, παλίνδρομος βρέθηκε το 1987 από τον H. Dubner)
- $2^{1257787} - 1$ (με 378632 ψηφία βρέθηκε το 1996 από τους Slovinski, Gage από τους μεγαλύτερους πρώτους που είναι γνωστοί)
- $2^{13466917} - 1$ (με 4053946 ψηφία βρέθηκε το 2001)

*Πρόταση* 9.2. Κάθε ακέραιος μεγαλύτερος του 1 είναι είτε πρώτος είτε γινόμενο πρώτων αριθμών.

*Άσκηση* 9.3. Αποδείξτε με επαγωγή την προηγούμενη πρόταση.

*Θεώρημα* 9.4 (Ευκλείδη). *Οι πρώτοι είναι άπειροι σε πλήθος.*

*Απόδειξη.* Εστω ότι οι πρώτοι είναι πεπερασμένοι σε πλήθος — συγκεκριμένα $p_1, p_2, \ldots, p_n$ — τότε ο αριθμός $p_1 p_2 \ldots p_n + 1$ δε διαιρείται από κανένα πρώτο παρά μόνο από το 1 και τον εαυτό του, άρα είναι πρώτος, κάτι που είναι άτοπο. □

*Θεώρημα* 9.5 (Θεμελιώδες Θεώρημα Αριθμητικής). *Κάθε αριθμός μπορεί να γραφεί με μοναδικό τρόπο σε γινόμενο πρώτων αριθμών (όχι απαραίτητα διάφορετικών ανά δύο).*

*Άσκηση* 9.6. Αν p πρώτος και $p \mid ab \Rightarrow p \mid a \vee p \mid b$

### 9.2.3 Μέγιστος Κοινός Διαιρέτης - Ακέραια Διαίρεση

*Ορισμός* 9.3. Ο μέγιστος κοινός διαιρέτης των a, b είναι ο μεγαλύτερος ακέραιος που διαιρεί τον a και τον b. συμβολίζουμε με $(a, b)$, ή με μκδ$(a, b)$, ή με gcd$(a, b)$.

*Πρόταση* 9.7. Διαίρεση: Για $a, b \in \mathbf{Z}, b \neq 0$ τότε $\exists q, r \in \mathbf{Z}$ έτσι ώστε $a = qb+r$ και $0 \leq r < |b|$. Τα $q, r$ είναι μοναδικά (για $r = 0$ έχουμε ότι $b \mid a$).

*Απόδειξη.* Ορίζουμε $r$ τον μικρότερο μη αρνητικό όρο της εξής αριθμητικής προόδου: $\ldots, a - 3b, a - 2b, a - b, a, a + b, a + 2b, a + 3b, \ldots$

- Το $r$ ικανοποιεί τις συνθήκες της πρότασης ($r$ είναι της μορφής $a - qb$).
- Μοναδικότητα: Έστω $r, r'(q, q')$.
  Τότε: $0 = (q - q')b + (r - r') \Rightarrow r' - r = (q - q')b$
  αλλά: $|r' - r| < b$, άρα $r = r'$ και $q = q'$.

□

**Εύρεση μκδ - Αλγόριθμος Ευκλείδη**: Η εύρεση του μκδ όταν είναι γνωστοί οι πρώτοι παράγοντες των $a, b$ είναι απλή. Η παραγοντοποίηση μεγάλων αριθμών είναι δύσκολο πρόβλημα, όμως για τον μκδ υπάρχει ένας αλγόριθμος που δίνει αποτέλεσμα με $O(\log a)$ διαιρέσεις ($O(\log^3 a)$) bit operations), όπου $a$ ο μεγαλύτερος από τους δύο αριθμούς. Με διαδοχικές διαιρέσεις έχουμε:

$$\begin{array}{rcll} a & = & bq_1 + r_1 & 0 < r_1 < b \\ b & = & r_1q_2 + r_2 & 0 < r_2 < r_1 \\ r_1 & = & r_2q_3 + r_3 & 0 < r_3 < r_2 \\ & \vdots & & \vdots \\ r_{j-2} & = & r_{j-1}q_j + r_j & 0 < r_j < r_{j-1} \\ r_{j-1} & = & r_jq_{j+1} & \end{array}$$

Μέχρι να βρούμε ένα $r_j$ που να διαιρεί ακριβώς το προηγούμενο υπόλοιπο $r_{j-1}$. Τότε το $r_j$ είναι ο μκδ. Π.χ.

$$\begin{array}{rcl@{\qquad}rcl} 1742 & = & 3 \cdot 494 + 260 & 132 & = & 3 \cdot 35 + 27 \\ 494 & = & 1 \cdot 260 + 234 & 35 & = & 1 \cdot 27 + 8 \\ 260 & = & 1 \cdot 234 + 26 & 27 & = & 3 \cdot 8 + 3 \\ 234 & = & 9 \cdot 26 & 8 & = & 2 \cdot 3 + 2 \\ & & & 3 & = & 1 \cdot 2 + 1 \\ & & & 2 & = & 2 \cdot 1 \end{array}$$

Επομένως $(1742, 494) = 26$ και $(132, 35) = 1$.

Ο αλγόριθμος του Ευκλείδη μπορεί να γραφεί πολύ συνοπτικά με χρήση αναδρομής:

**Αλγόριθμος Ευκλείδη**
**function** gcd($a, b$: natural);
  **if** $b \mid a$ **then** gcd := $b$ **else** gcd := gcd$(b, a \bmod b)$

Η ορθότητα του αλγόριθμου του Ευκλείδη στηρίζεται στην παρακάτω πρόταση:

*Άσκηση* 9.8. Αν $a, b \in \mathbf{Z}$ τότε ισχύει ότι $(a, b) = (b, a \mod b)$.

Η πολυπλοκότητα του αλγόριθμου του Ευκλείδη στηρίζεται στην παρακάτω πρόταση:

*Πρόταση* 9.9. Έστω $a, b$ η είσοδος στον αλγόριθμο του Ευκλείδη και $i, j$ οι παράμετροι της 2ης αναδρομικής κλήσης της gcd (εφ'όσον ο αλγόριθμος δεν σταματήσει νωρίτερα). Ισχύει:

$$\max(i, j) \leq \max(a, b)$$

*Απόδειξη.* Άσκηση. □

Με απλά λόγια, κάθε 2 το πολύ κλήσεις, ο μεγαλύτερος από τους δύο αριθμούς υποδιπλασιάζεται. Επομένως εκτελούνται $(\log(\max(a, b)))$ επαναλήψεις, και $O(\log^3(\max(a, b)))$ bit operations.

*Ορισμός* 9.4. *Σχετικά πρώτοι (coprime)* λέγονται οι $a, b$ αν ισχύει $(a, b) = 1$.

Είναι φανερό ότι η απόφαση για το αν δύο αριθμοί είναι σχετικά πρώτοι δίνεται σε πολυωνυμικό χρόνο από τον αλγόριθμο του Ευκλείδη.

*Πρόταση* 9.10. Αν $d = (a, b)$ τότε $\exists x, y \in \mathbf{Z} : d = xa + yb$

Δηλαδή ο μκδ μπορεί να γραφεί ως γραμμικός συνδυασμός των a, b. Μάλιστα αυτό μπορεί να γίνει σε πολυωνυμικό χρόνο ($O(\log^3(a)$ bit operations). Ο αλγόριθμος που υπολογίζει τα x, y στηρίζεται στον αλγόριθμο του Ευκλείδη κάνοντας την αντίστροφή διαδικασία π.χ. στο προηγούμενο παράδειγμα για να υπολογίσουμε το 26 (το 1) ως γραμμικό συνδυασμό των 1742 και 494 (των 132 και 35) κάνουμε τα εξής:

$$\begin{aligned} 26 &= 260 - 234 \\ &= 260 - (494 - 260) \\ &= 2 \cdot 260 - 494 \\ &= 2(1742 - 3 \cdot 494) - 494 \\ &= 2 \cdot 1742 - 7 \cdot 494 \end{aligned}$$

$$\begin{aligned} 1 &= 3 - 2 \\ &= 3 - (8 - 2 \cdot 3) \\ &= 3 \cdot 3 - 8 \\ &= 3(27 - 3 \cdot 8) - 8 \\ &= 3 \cdot 27 - 10 \cdot 8 \\ &= 3 \cdot 27 - 10(35 - 27) \\ &= 13 \cdot 27 - 10 \cdot 35 \\ &= 13(132 - 3 \cdot 35) - 10 \cdot 35 \\ &= 13 \cdot 132 - 49 \cdot 35 \end{aligned}$$

**Παρατηρήσεις:**

- $(ma, mb) = m(a, b)$
- $(a, m) = (b, m) = 1 \Rightarrow (ab, m) = 1$
- $(a, b) = (a, b + ax)$
- $p$ πρώτος $\wedge\ p \mid ab \Rightarrow p \mid a \vee p \mid b$
- $m \mid a \wedge n \mid a \wedge (m, n) = 1 \Rightarrow mn \mid a$

### 9.2.4 Η συνάρτηση Euler

*Ορισμός* 9.5. *Η συνάρτηση $\phi$ του Euler*. Ορίζουμε $\phi(n)$ να είναι το πλήθος των αριθμών από το 1 μέχρι και το n που είναι σχετικά πρώτοι με τον n.

*Πρόταση* 9.11. Ιδιότητες της συνάρτησης $\phi$:

1. $\phi(p) = p - 1$ όπου $p$ πρώτος.
2. $\phi(p^a) = p^a(1 - \frac{1}{p})$ όπου $p$ πρώτος.
3. $\phi(mn) = \phi(m)\phi(n)$ για m,n σχετικά πρώτους.

*Άσκηση* 9.12. Να αποδειχθεί το (2) της παραπάνω πρότασης.

**Παρατήρηση:** Από τα (2) (3) είναι φανερό ότι αν είναι γνωστή η ανάλυση ενός αριθμού n σε πρώτους παράγοντες τότε είναι εύκολο να υπολογίσουμε το $\phi(n)$.

*Πόρισμα* 9.13. Αν $n \in N$ τότε $\phi(n) = n \prod_{p|n}(1 - \frac{1}{p})$.

*Πρόταση* 9.14. Έστω $n = pq$, $p, q$ πρώτοι, $p \neq q$. Γνώση των δύο πρώτων $p, q$ είναι ισοδύναμη με γνώση του $\phi(n)$ (οι αντίστοιχοι υπολογισμοί χρειάζονται $(\log^3 n)$ bit operations).

*Θεώρημα* 9.15 (Euler). $\forall n \geq 1; \sum_{d|n} \phi(d) = n$

## 9.3 Ομάδες, Δακτύλιοι

*Ορισμός* 9.6. Αντιμεταθετική ομάδα λέγεται ένα ζεύγος $(G, *)$ όπου G είναι ένα σύνολο και $*$ πράξη: $* : G \times G \to G$ ώστε να ισχύει για $a, b, c \in G$:

1. $a * (b * c) = (a * b) * c$ (προσεταιριστική ιδιότητα της *)
2. $a * b = b * a$ (αντιμεταθετική ιδιότητα της *)
3. $\exists e \in G$ τέτοιο ώστε $\forall a$ ισχύει $a * e = a$ (μοναδιαίο στοιχείο)
4. $\forall a \; \exists a^{-1} \in G : a * a^{-1} = e$ (αντίστροφο στοιχείο)

*Παραδείγματα:* (Q, +) (R, +) $(Q - \{0\}, \cdot)$.

*Ορισμός* 9.7. Έστω $(A, \oplus)$ και $(B, \otimes)$ δύο αλγεβρικές δομές. Μια συνάρτηση $f : A \to B$ λέγεται *ομομορφισμός* αν $f(a \oplus b) = f(a) \otimes f(b)$, *μονομορφισμός* εάν επιπλέον είναι *1-1*, *επιμορφισμός* αν είναι *επί*. *Ισομορφισμός* λέγεται ένας ομομορφισμός που είναι 1-1 και επί.

*Ορισμός* 9.8. Δακτύλιος λέγεται μια τριάδα $(R, +, *)$ όπου το $(R, +)$ είναι αντιμεταθετική ομάδα και ισχύει η προσεταιριστική ιδιότητα για την πράξη $*$ και η επιμεριστική ιδιότητα της $*$ ως προς την $+$:

$$\forall a, b, c \in R \begin{cases} a * (b + c) = a * b + a * c, \\ (b + c) * a = b * a + c * a \end{cases}$$

Ο δακτύλιος λέγεται "αντιμεταθετικός δακτύλιος" αν η πράξη $*$ έχει την αντιμεταθετική ιδιότητα ή "δακτύλιος με μονάδα" αν υπάρχει μοναδιαίο στοιχείο ως προς την $*$.

*Ορισμός* 9.9. Σώμα λέγεται μια τριάδα $(F, +, *)$ όπου το $(F, +)$ και $(F - \{e_+\}, *)$ είναι αντιμεταθετικές ομάδες και ισχύει η επιμεριστική ιδιότητα της * ως προς την +. Με $e_+$ συμβολίζουμε το ουδέτερο ως προς την πράξη +.

*Ορισμός* 9.10. Υποομάδα μιας ομάδας $(G, *)$ λέγεται το ζεύγος $(S, *)$ όπου ισχύει ότι $S \subseteq G$, S κλειστό ως προς $*$ και $(S, *)$ είναι ομάδα.

*Ορισμός* 9.11. Μια ομάδα $(G, *)$ λέγεται κυκλική αν υπάρχει στοιχείο $g \in (G, *)$ με την ιδιότητα $\forall x \in G \; \exists y : x = g^y$. Το στοιχείο αυτό το ονομάζουμε και γεννήτορα της $(G, *)$.

*Ορισμός* 9.12. Τάξη ενός στοιχείου $a$ μιας ομάδας $G$ είναι το μικρότερο $y$ έτσι ώστε $a^y = e$.

*Πρόταση* 9.16. Αν ένα στοιχείο είναι γεννήτορας της $(G, *)$ τότε ισχύει ότι η τάξη του είναι $|G|$.

*Ορισμός* 9.13. Έστω $H$ υποομάδα της $(G, *)$ και $a \in G$. Το $H * a = \{h * a | h \in H\}$ λέγεται δεξί σύμπλοκο (coset) της $H$ στη $G$.

*Πρόταση* 9.17. Το $G/H$ (= σύνολο των δεξιών cosets της $H$ στην $G$) είναι ομάδα και λέγεται *quotient group* με πράξη $(H * a) * (H * b) = H * (a * b)$.

*Θεώρημα* 9.18 (Lagrange). *Αν $G$ είναι πεπερασμένη ομάδα και $H$ είναι υποομάδα της $G$, τότε $|G| = |G/H| \cdot |H|$, δηλαδή το $|H|$ διαιρεί το $|G|$.*

## 9.4 Ισοτιμίες, Ο δακτύλιος $\mathbf{Z}_m$

### 9.4.1 Εισαγωγή

Εδώ παρουσιάζονται οι έννοιες της σχέσης ισοτιμίας και του δακτυλίου $\mathbf{Z}_m$. Μετά παρουσιάζονται το μικρό θεώρημα του Fermat και το Κινέζικο Θεώρημα Υπολοίπων.

### 9.4.2 Σχέση ισοτιμίας (congruence)

*Ορισμός* 9.14. Δύο αριθμοί a, b λέγονται *ισότιμοι modulo m* αν $m \mid (a - b)$ και συμβολίζουμε με $a \equiv b \pmod{m}$. Ισοδύναμα μπορούμε να πούμε ότι οι a και b αν διαιρεθούν με το m δίνουν το ίδιο υπόλοιπο.

*Παρατήρηση* 9.19.

i. $a = b \mod m$ αν και μόνο αν $a \equiv b \pmod{m}$ και $a \in \mathbb{Z}_m$. Το $b \bmod m$ (χωρίς παρένθεση) συμβολίζει το υπόλοιπο της ευκλείδιας διαίρεσης του ακεραίου $b$ με το $m$.

ii. Στη βιβλιογραφία θα βρείτε και άλλους συμβολισμούς για την ισοτιμία. π.χ. $a = b \pmod{m}$ ή και $a \equiv b \, (m)$.

*Άσκηση* 9.20. Η σχέση της ισοτιμίας $modulo \; m$ είναι σχέση ισοδυναμίας για τους ακεραίους και τους χωρίζει σε $m$ κλάσεις $C_0, C_1, C_2, \ldots, C_{m-1}$.

Κάθε κλάση $C_k$ περιέχει τους ακεραίους που αφήνουν υπόλοιπο $k$ αν διαιρεθούν με το $m$. π.χ. για $m = 5$ έχουμε τις εξής κλάσεις:

$$\begin{aligned} C_0 &= \{0, 5, 10, 15, 20, 25, \ldots, 5k, \ldots\} \\ C_1 &= \{1, 6, 11, 16, 21, 26, \ldots, 5k+1, \ldots\} \\ C_2 &= \{2, 7, 12, 17, 22, 27, \ldots, 5k+2, \ldots\} \\ C_3 &= \{3, 8, 13, 18, 23, 28, \ldots, 5k+3, \ldots\} \\ C_4 &= \{4, 9, 14, 19, 24, 29, \ldots, 5k+4, \ldots\} \end{aligned}$$

### 9.4.3 Πράξεις και ισοτιμίες

*Ορισμός* 9.15. Οι πράξεις που ορίζονται μεταξύ των κλάσεων ορίζονται με τον εξής τρόπο: $C_k + C_j = C_{(k+j) \mod m}$ και $C_k \cdot C_j = C_{kj \mod m}$

Το σύνολο των κλάσεων $\{C_0, C_1, C_2, \ldots, C_{m-1}\}$ που για απλότητα θα γράφουμε ώς $\{0, 1, 2, \ldots, m-1\}$, θα το συμβολίζουμε με $\mathbf{Z}_m$ και το ονομάζουμε *σύνολο ακεραίων modulo m*.

*Άσκηση* 9.21. Με τις πράξεις $+, \cdot$ (όπως ορίσθηκαν παραπάνω) να δειχθεί ότι ο $(\mathbf{Z}_m, +, \cdot)$ είναι αντιμεταθετικός δακτύλιος με μοναδιαίο στοιχείο.

Ενας ισοδύναμος τρόπος να ορίσουμε το $\mathbf{Z}_m$ είναι $\mathbf{Z}_m = \{a \mod m \mid a \in \mathbf{Z}\}$ και τότε οι πράξεις είναι ακριβώς η πρόσθεση και ο πολλαπλασιασμός modulo $m$ μεταξύ ακεραίων,

$$(a \bmod m + b \bmod m) \bmod m = (a + b) \bmod m \,,$$

$$\big((a \bmod m) \cdot (b \bmod m)\big) \bmod m = (a \cdot b) \bmod m \,,$$

έχουμε δηλαδή στην ισοτιμία εκτός της ισοδυναμίας και ιδιότητες ομομορφισμού.

Γενικά μπορούμε να εφαρμόσουμε στους ακέραιους modulo $m$ πρόσθεση, αφαίρεση, πολλαπλασιασμό θεωρώντας τους απλούς ακέραιους αρκει πάντα στο τέλος να παίρνουμε σαν αποτέλεσμα το υπόλοιπο της διαίρεσης με το $m$. Βέβαια όπως θα δούμε παρακάτω αυτός ο τρόπος δεν είναι πάντα ο καλύτερος για υλοποίηση σε υπολογιστή.

Ενα παράδειγμα του ισχυρισμού αυτού είναι ο υπολογισμός του $a^m \bmod n$. Αν χρησιμοποιήσουμε διαδοχικούς πολλαπλασιασμούς του στοιχείου $a$ τότε είναι σαφές ότι ο αλγόριθμος αυτός θα χρειαστεί εκθετικά πολλούς πολλαπλασιασμούς σε σχέση με το μήκος του m. 'Ενας πιο "καλός" αλγόριθμος για τον υπολογισμό αυτό είναι ο εξής:

**Αλγόριθμος Επαναλαμβανόμενου Τετραγωνισμού (Repeated Squaring)**

Θέλουμε να υπολογίσουμε το $a^m \mod n$, έστω ότι $m = b_{k-1}2^{k-1} + \ldots b_1 2 + b_0$.

**input**: $a, b_0, b_1, \ldots, b_{k-1}$
  $x := a; y := 1;$
  **for** i:=0 **to** k-1 **do**
  **begin**
    **if** $b_i = 1$ **then** $y := (y \cdot x) \mod n;$

$x := x^2 \mod n$
**end**
**output**: $y$

Ο παραπάνω αλγόριθμος πραγματοποιεί το πολύ $2\lceil \log_2 m \rceil$ πολλαπλασιασμούς.

### 9.4.4 Ιδιότητες ισοτιμιών

*Πρόταση* 9.22. Ισχύουν:

1. Αν $a \equiv b \pmod m$ τότε $a \equiv b \pmod d$ για κάθε $d \mid m$.
2. Αν $a \equiv b \pmod m$ και $a \equiv b \pmod n$ με $(m,n) = 1$ τότε $a \equiv b \pmod{mn}$.

*Άσκηση* 9.23. Αποδείξτε την προηγούμενη πρόταση.

Οι ακέραιοι $modulo\ m$ δεν έχουν όλοι αντίστροφο ως προς τον πολλαπλασιασμό. Μπορεί να δειχθεί ότι ικανή και αναγκαία συνθήκη για να συμβαίνει αυτό είναι να ισχύει ότι $(a,m) = 1$.

*Πρόταση* 9.24. $(a,m) = 1$ αν και μόνο αν $\exists c \in \mathbf{Z}_m$ τέτοιο ώστε $a \cdot c \equiv 1 \pmod m$.

*Απόδειξη.* Αν ισχύει $(a,m) = 1$ τότε $\exists x, y : 1 = xa + ym$. Παίρνοντας το υπολοιπο της διαίρεσης με το $m$ των δύο μελών έχουμε ότι

$$1 \mod m = xa \mod m + ym \mod m \Rightarrow$$

$$(x \mod m)(a \mod m) = 1 \mod m\ .$$

Αρα το $a \mod m$ έχει αντίστροφο το $x \mod m$. Και όπως είδαμε το $x$ μπορεί να υπολογιστεί σε πολυωνυμικό χρόνο με την επέκταση του αλγορίθμου του Ευκλείδη.

Αντίστροφα αν το $a \mod m$ έχει αντίστροφο το $x \mod m$ τότε ισχύει ότι το $ax \equiv 1 \pmod m \Rightarrow m \mid (ax-1)$. Αν $(a,m) = d > 1$ τότε $d \mid (ax-1) \Rightarrow d \mid 1$, άτοπο αφού $d > 1$. □

*Ορισμός* 9.16. Το υποσύνολο του $\mathbf{Z}_m$ που κάθε στοιχείο του είναι σχετικά πρώτο με το $m$ το συμβολίζουμε με $U(\mathbf{Z}_m)$ (units). Δηλαδή $U(\mathbf{Z}_m) = \{a \in \mathbf{Z}_m : (a,m) = 1\}$.

*Άσκηση* 9.25. Το $(U(\mathbf{Z}_m), \cdot)$ είναι αντιμεταθετική ομάδα. Αν $m = p$ πρώτος τότε $U(\mathbf{Z}_p) = \mathbf{Z}_p - \{0\} = \mathbb{Z}_p^*$ και σε αυτήν την περίπτωση το $\mathbf{Z}_p$ είναι σώμα (τότε συμβ. με $GF(p)$ ή $F_p$). Ο πληθικός αριθμός του $U(\mathbf{Z}_m)$ είναι $\phi(m)$.

### 9.4.5 Μικρό Θεώρημα Fermat

*Θεώρημα* 9.26 (Μικρό Θεώρημα Fermat).

$$\forall a \in \mathbf{Z},\ \forall \text{prime } p \not| a : \quad a^{p-1} \equiv 1 \pmod p$$

*Απόδειξη.* Έστω ένα $a \in \mathbf{Z}$ με $p \not| a$. Τότε το $a$ αντιστοιχεί σε ένα στοιχείο του $\mathbf{Z}_p$ διαφορετικό του 0. Το στοιχείο αυτό το συμβολίζουμε με $a$. Τότε τα στοιχεία,

$$a \cdot 1, a \cdot 2, \ldots, a \cdot (p-1)$$

είναι διαφορετικά ανά δύο στην ομάδα $\mathbf{Z}_p - \{0\}$. Πράγματι αν $i \cdot a \equiv j \cdot a \pmod{p}$ τότε έχουμε

$$p \mid (a \cdot i - a \cdot j) \Rightarrow p \mid a(i-j) \Rightarrow p \mid a \vee p \mid (i-j)$$

'Ομως αφού το $a$ είναι διαφορετικό του 0 στον $\mathbf{Z}_p$ δεν ισχύει ότι $p \mid a$ άρα $p \mid (i-j)$ συνεπώς $i \equiv j \pmod{p}$.

Από τα παραπάνω έχουμε ότι αφού τα στοιχεία $a \cdot 1, a \cdot 2, \ldots, a \cdot (p-1)$ είναι διαφορετικά ανά δύο θα αποτελούν μια μετάθεση των στοιχείων της ομάδας $(\mathbf{Z}_p - \{0\}, \cdot)$. Άρα,

$$a1 \cdot a2 \cdot \ldots \cdot a(p-1) \equiv 1 \cdot 2 \cdot \ldots \cdot (p-1) \pmod{p}$$

$$a^{p-1} \cdot 1 \cdot 2 \cdot \ldots \cdot (p-1) \equiv 1 \cdot 2 \cdot \ldots \cdot (p-1) \pmod{p}$$

Τα στοιχεία $1, 2, \ldots, (p-1)$ έχουν αντίστροφο στο $\mathbf{Z}_p$ επομένως έχουμε ότι $a^{p-1} \equiv 1 \pmod{p}$. □

Το μικρό Θεώρημα του Fermat είναι ειδική περίπτωση ενός πολύ πιο γενικού θεωρήματος από τη θεωρία ομάδων,

*Θεώρημα* 9.27. *Αν $(G, \cdot)$ πεπερασμένη ομάδα τότε $\forall a \in G$ ισχύει $a^{|G|} = e$ όπου $e$ το ουδέτερο στοιχείο της ομάδας.*

Σαν πόρισμα προκύπτει το μικρό θεώρημα Fermat, εφαρμόζοντας το παραπάνω θεώρημα για την ομάδα $(\mathbf{Z}_p - \{0\}, \cdot)$. Επίσης εφαρμόζοντας για την ομάδα $(U(\mathbf{Z}_m), \cdot)$ έχουμε το εξής,

*Πόρισμα* 9.28. (Euler) $\forall a \in \mathbf{Z}, (a, m) = 1$ ισχύει $a^{\phi(m)} \equiv 1 \pmod{m}$.

*Άσκηση* 9.29. Αν ο πρώτος p δε διαιρεί τον a και $n \equiv m \pmod{p-1}$ τότε $a^n \equiv a^m \pmod{p}$.

*Θεώρημα* 9.30 (Wilson). *$p$ πρώτος* $\Rightarrow$ $(p-1)! \equiv -1 \pmod{p}$

### 9.4.6 Κινέζικο Θεώρημα Υπολοίπων

*Θεώρημα* 9.31 (Κινέζικο Θεώρημα Υπολοίπων). *Εστω ένα σύστημα εξισώσεων*

$$\begin{aligned} x &\equiv a_1 \pmod{m_1} \\ x &\equiv a_2 \pmod{m_2} \\ &\vdots \\ x &\equiv a_k \pmod{m_k} \end{aligned}$$

*ώστε $(m_i, m_j) = 1$ για $i \neq j$. Τότε το σύστημα έχει μοναδική λύση στο δακτύλιο $\mathbf{Z}_M$ όπου $M = m_1 m_2 \ldots m_k$. Ισοδύναμα μπορούμε να πούμε ότι το σύστημα έχει άπειρες λύσεις και αν $s_1, s_2$ δύο λύσεις ισχύει $s_1 \equiv s_2 \pmod{M}$.*

*Απόδειξη.* Εστω $M_i = \frac{M}{m_i}$ τότε έχουμε ότι $(M_i, m_i) = 1$ και αν θεωρήσουμε το $M_i$ στοιχείο του $\mathbf{Z}_{m_i}$ τότε είναι αντιστρέψιμο. Επομένως υπάρχει ένα στοιχείο $N_i$ με $N_i \cdot M_i \equiv 1 \pmod{m_i}$ για κάθε $i$. Παρατηρούμε επίσης ότι για κάθε $i \neq j$ $M_i \cdot N_i \equiv 0 \pmod{m_j}$. 'Εστω

$$y = \sum_{i=1}^{k} N_i \cdot M_i \cdot a_i$$

Η $y$ είναι η ζητούμενη λύση. Πράγματι

$$\forall i: \quad y = (M_1 N_1 a_1 + \ldots + M_i N_i a_i + \ldots + M_k N_k a_k) \equiv a_i \pmod{m_i}$$

Αν $s_1, s_2$ δύο διαφορετικές λύσεις τότε έχουμε ότι για κάθε $i$,

$$s_1 \equiv a_i \pmod{m_i} \qquad s_2 \equiv a_i \pmod{m_i}$$

Από την πρόταση 9.22 έχουμε ότι $s_1 \equiv s_2 \pmod{M}$. Άρα το σύστημα έχει μοναδική λύση στο $\mathbf{Z}_M$ την $y \mod M$. □

Είναι φανερό από την παραπάνω απόδειξη ότι η λύση ενός τέτοιου συστήματος μπορεί να βρεθεί σε πολυωνυμικό χρόνο.

## 9.5 Τετραγωνικά Υπόλοιπα, Ο νόμος Τετραγωνικής Αμοιβαιότητας

Εδώ θα παρουσιάσουμε την έννοια των τετραγωνικών υπολοίπων και το σύμβολο Legende.

### 9.5.1 Τετραγωνικά Υπόλοιπα

Το κεντρικό ενδιαφέρον της ενότητας αυτής είναι η εξίσωση $x^2 \equiv a \pmod{p}$ που στα πρώτα εδάφια την εξετάζουμε για συγκεκριμένο $p$. Η μελέτη της συγκεκριμένης εξίσωσης δεν παρουσιάζει ιδιαίτερες δυσκολίες, παρ'όλ'αυτά η μελέτη των λύσεων της εξίσωσης αυτής αν κρατήσουμε σταθερό το $a$ και εξετάσουμε ποιοι πρώτοι την επαληθεύουν παρουσιασε μεγάλο ενδιαφέρον στο παρελθόν. Σε αυτήν την ενότητα θα εξετάσουμε διάφορα "εργαλεία" που έπαιξαν ρόλο στα αποτελέσματα αυτά, τα οποία έχουν σημαντικές εφαρμογές στην μοντέρνα κρυπτογραφία.

*Πρόταση* 9.32. Έστω $p, q$ πρώτοι και $a \not\equiv 0 \pmod{p}$, τότε:

1. Η εξίσωση $x^2 \equiv a \pmod{p}$ έχει είτε 0 είτε 2 λύσεις στο $\mathbf{Z}_p$.
2. Η εξίσωση $x^2 \equiv a \pmod{pq}$ έχει είτε 0 είτε 4 λύσεις στο $\mathbf{Z}_{pq}$.

*Απόδειξη.* 1. 'Εστω ότι η $x$ είναι λύση της εξίσωσης, τότε και η $-x$ είναι λύση. Επίσης αν $x \equiv -x \pmod{p}$ τότε $2x \equiv 0 \pmod{p} \Rightarrow x \equiv 0 \pmod{p}$, όποτε για τιμές διαφορες του 0 έχουμε ότι $x \not\equiv -x \pmod{p}$. Αν $x, y$ λύσεις της εξίσωσης τότε $x^2 \equiv y^2 \pmod{p}$ άρα

$$p \mid (x^2 - y^2) \Rightarrow p \mid (x-y)(x+y) \Rightarrow$$
$$p \mid (x-y) \vee p \mid (x+y) \Rightarrow x \equiv y \pmod{p} \vee x \equiv -y \pmod{p}$$

2. Η λύση της εξίσωσης είναι ισοδύναμη με τη λύση των δύο εξισώσεων $x^2 \equiv a \pmod{p}$, $x^2 \equiv a \pmod{q}$. Εστω ότι η πρώτη έχει λύσεις τις $x_1, -x_1$ και η δεύτερη τις $x_2, -x_2$. Για καθε ένα από τους συνδυασμούς των λύσεων αυτών (που είναι 4) προκύπτει μια διαφορετική λύση για την εξίσωση, από το σύστημα $x \equiv \pm x_1 \pmod{p}$, $x \equiv \pm x_2 \pmod{q}$. Η ύπαρξη μοναδικής λύσης στο $\mathbf{Z}_{pq}$ αυτού του συστήματος προκύπτει από το κινέζικο θεώρημα υπολοίπων.

□

*Σημείωση* 9.1. Η παραπάνω πρόταση μπορεί να γενικευτεί και για τυχαίο $n = p_1^{a_1} p_2^{a_2} \dots p_k^{a_k}$ όπου η αντίστοιχη εξίσωση έχει είτε 0 είτε $2^k$ λύσεις.

Στην επόμενη πρόταση θα δούμε ένα κριτήριο που θα μας επιτρέπει να ελέγχουμε αν ένας αριθμός στο $\mathbf{Z}_p$ έχει τετραγωνικές ρίζες.

*Πρόταση* 9.33. Η εξίσωση $x^2 \equiv a \pmod{p}$ έχει λύση αν και μόνο αν $a^{\frac{p-1}{2}} \equiv 1 \pmod{p}$.

*Απόδειξη.* 'Εστω $g$ είναι γεννήτορας της $\mathbf{Z}_p$ και $y, b$ με $x \equiv g^y \pmod{p}$ και $a \equiv g^b \pmod{p}$. Τότε,

$$x^2 \equiv a \pmod{p} \Rightarrow g^{2y} \equiv g^b \pmod{p} \Rightarrow 2y \equiv b \pmod{p-1}$$

Η τελευταία συνεπαγωγή προκύπτει από την άσκηση 9.29. Η εξίσωση $2y \equiv b \pmod{p-1}$ έχει λύση αν και μόνο αν $(2, p-1) \mid b \Rightarrow 2 \mid b$.

Αν η $x^2 \equiv a \pmod{p}$ έχει λύση τότε θα έχει και η $2y \equiv b \pmod{p-1}$ οπότε $2 \mid b$ και $a^{\frac{p-1}{2}} \equiv g^{b\frac{p-1}{2}} \equiv 1 \pmod{p}$ (από την πρόταση 9.26).

Αν ισχύει ότι $a^{\frac{p-1}{2}} \equiv 1 \pmod{p}$ τότε $g^{b\frac{p-1}{2}} \equiv 1 \pmod{p} \Rightarrow (p-1) \mid \frac{b(p-1)}{2} \Rightarrow 2 \mid b$. Άρα η $2y \equiv b \pmod{p-1}$ θα έχει λύση και επομένως θα έχει και η $x^2 \equiv a \pmod{p}$. □

*Σημείωση* 9.2. Η προηγούμενη πρόταση γενικεύεται ως εξής: Αν $(a, m) = 1$ τότε $\exists x : x^n \equiv a \pmod{m}$ ανν $a^{\frac{\phi(m)}{d}} \equiv 1 \pmod{m}$ όπου $d = (n, \phi(m))$.

*Πρόταση* 9.34. Ακριβώς τα μισά στοιχεία του $U(\mathbf{Z}_p)$ είναι τετραγωνικά υπόλοιπα ενώ τα υπόλοιπα είναι τετραγωνικά μη υπόλοιπα.

*Απόδειξη.* Οι λύσεις της εξίσωσης $x^2 \equiv a \pmod{p}$ είναι πάντα στη μορφή $x, -x$. Χωρίζουμε τα στοιχεία του $U(\mathbf{Z}_p)$ σε 'θετικά' $\{1, 2, \dots, \frac{p-1}{2}\}$ και 'αρνητικά' $\{\frac{p-1}{2} + 1, \dots, p-1\}$. Είναι προφανές ότι κάθε ζευγάρι λύσεων μιας εξίσωσης $x^2 \equiv a \pmod{p}$ θα έχει το ένα μέλος στο ένα σύνολο των 'θετικών' στοιχείων και το άλλο στο σύνολο των 'αρνητικών'. 'Ετσι οι αριθμοί που θα προκύψουν αν τετραγωνίσουμε modulo $p$ τα στοιχεία του ενός υποσυνόλου θα είναι ίδιοι με τους αριθμούς που θα προκύψουν αν τετραγωνίσουμε τα στοιχεία του άλλου υποσυνόλου. Επομένως το πλήθος των τετραγωνικών υπολοίπων θα είναι ακριβώς $\frac{p-1}{2}$. □

### 9.5.2 Σύμβολο Legendre

Ενας ιδιαίτερα βολικός τρόπος να εκφράσουμε το αν ένας αριθμός $a \in \mathbf{Z}_p$ έχει τετραγωνικές ρίζες $modulo\ p$ είναι με το σύμβολο Legendre.

*Ορισμός* 9.17. Το σύμβολο Legendre ορίζεται ως εξής,

$$\left(\frac{a}{p}\right) = \begin{cases} 1, & \text{if } \exists x : x^2 \equiv a \pmod{p} \\ -1, & \text{if } \nexists x : x^2 \equiv a \pmod{p} \\ 0, & \text{if } p \mid a \end{cases}$$

Αν $(\frac{a}{p}) = 1$ τότε το $a$ θα το ονομάζουμε και τετραγωνικό υπόλοιπο modulo $p$. Αν $(\frac{a}{p}) = -1$ τότε το $a$ θα το ονομάσουμε τετραγωνικό μη υπόλοιπο modulo $p$.

Τα στοιχεία του $U(\mathbf{Z}_p)$ που δίνουν 1 και -1 στο σύμβολο Legendre είναι ίσα σε πλήθος, όπως φαίνεται από την προηγούμενη πρόταση.

*Πρόταση* 9.35. Μερικές ιδιότητες του συμβόλου Legendre,

1. $m \equiv n \pmod{p} \Rightarrow (\frac{m}{p}) = (\frac{n}{p})$
2. $(\frac{a}{p}) \equiv a^{\frac{p-1}{2}} \pmod{p}$
3. $(\frac{ab}{p}) = (\frac{a}{p})(\frac{b}{p})$

*Απόδειξη.* Το 1. είναι άμεσο από τον ορισμό. Για το 2. έχουμε ότι από την πρόταση 9.33 αν η εξίσωση $x^2 \equiv a \pmod{p}$ έχει λύσεις τότε $a^{\frac{p-1}{2}} \equiv 1 \equiv (\frac{a}{p}) \pmod{p}$. Για την περίπτωση που η εξίσωση δεν έχει λύσεις έχουμε γενικά ότι,

$$a^{p-1} \equiv 1 \pmod{p} \Rightarrow p \mid (a^{p-1} - 1) \Rightarrow$$

$$p \mid (a^{\frac{p-1}{2}} - 1)(a^{\frac{p-1}{2}} + 1) \Rightarrow a^{\frac{p-1}{2}} \equiv \pm 1 \pmod{p}\,.$$

Άρα αναγκαστικά έχουμε το 2. Το 3. προκύπτει εύκολα εφαρμόζοντας την ιδιότητα 2. □

Πρέπει να παρατηρηθεί ότι η ιδιότητα 2. μας δίνει επίσης έναν εύκολο αλγόριθμο υπολογισμού του συμβόλου Legendre, χρησιμοποιώντας τον αλγόριθμο επαναλαμβανόμενου τετραγωνισμού. Άλλη μία σημαντική ιδιότητα για τον υπολογισμό του συμβόλου Legendre είναι το εξής:

*Λήμμα* 9.36. *(Gauss) Αν το πλήθος των στοιχείων του συνόλου*
$\{a \mod p, 2a \mod p, \ldots, \frac{p-1}{2}a \mod p\}$ *που είναι μεγαλύτερα του* $\frac{p}{2}$ *το συμβολίσουμε με* $\mu$ *τότε ισχύει ότι* $(\frac{a}{p}) = (-1)^{\mu}$.

Με βάση το λήμμα αυτό μπορούμε να δείξουμε την επόμενη σημαντική πρόταση.

*Πρόταση* 9.37.

1. $(\frac{-1}{p}) = (-1)^{\frac{p-1}{2}} = \begin{cases} 1, & \text{if } p \equiv 1 \pmod{4} \\ -1, & \text{if } p \equiv 3 \pmod{4} \end{cases}$
2. $\left(\frac{2}{p}\right) = \begin{cases} 1, & \text{if } p \equiv 1 \pmod{8} \vee p \equiv 7 \pmod{8} \\ -1, & \text{if } p \equiv 3 \pmod{8} \vee p \equiv 5 \pmod{8} \end{cases}$

*Απόδειξη.* Το *1.* προκύπτει άμεσα από την ιδιότητα 2 της πρότασης 9.35. Για το *2* έχουμε ότι για το σύνολο $2 \cdot 1, 2 \cdot 2, \ldots, 2\frac{p-1}{2}$, αν $m$ είναι τέτοιο ώστε, $2m \leq \frac{p-1}{2}$ και $2(m+1) > \frac{p-1}{2}$ τότε το πλήθος των στοιχείων του συνόλου αυτού μεγαλύτερων του $\frac{p-1}{2}$ είναι $\frac{p-1}{2} - m$.

Αν $p \equiv 1 \pmod 8$ τότε από τα παραπάνω έχουμε ότι το $\mu$ του λήμματος 9.36 είναι ίσο με $\frac{p-1}{2} - m$ και επειδή $\frac{p-1}{2} = 4k$ έχουμε $m = 2k$. Άρα $\mu = 4k - 2k = 2k$ και συνεπώς $(\frac{2}{p}) = (-1)^{2k} = 1$. Με όμοιο τρόπο αποδεικνύονται και οι υπόλοιπες περιπτώσεις. □

Το πιο σημαντικό θεώρημα της ενότητας είναι το ακόλουθο που αποδείχθηκε για πρώτη φορά από τον Gauss το 1796. Η απόδειξη παραλείπεται.

*Θεώρημα* 9.38 (Νόμος Τετραγωνικής Αντιστροφής).

$$\left(\frac{p}{q}\right) = \begin{cases} -(\frac{q}{p}), & \text{αν } p \equiv q \equiv 3 \pmod 4 \\ (\frac{q}{p}), & \text{αλλιώς.} \end{cases}$$

Μια χρήσιμη γενίκευση του συμβόλου Legendre είναι το *σύμβολο Jacobi* που ορίζεται πάνω σε όλους τους αριθμούς.

*Ορισμός* 9.18 (Σύμβολο Jacobi). Αν $n = p_1^{a_1} p_2^{a_2} \ldots p_k^{a_k}$ τότε ορίζουμε το σύμβολο Jacobi

$$\left(\frac{m}{n}\right) = \prod_{i=1}^{k} \left(\frac{m}{p_i}\right)^{a_i}.$$

Μπορεί να επαληθευθεί με βάση τον ορισμο ότι το σύμβολο Jacobi ικανοποιεί τις προτάσεις 9.35, 9.37 και 9.38.

*Σημείωση* 9.3. Πρέπει να σημειωθεί ότι το σύμβολο Jacobi δεν χαρακτηρίζει απολύτως την ύπαρξη λύσεων της αντίστοιχης εξίσωσης $x^2 \equiv a \pmod n$. Πράγματι είναι εύκολο να δούμε ότι αν αυτή η εξίσωση έχει λύσεις το σύμβολο Jacobi $(\frac{a}{n}) = 1$ αλλά δεν ισχύει το αντίστροφο.

Απο πλευράς πολυπλοκότητας θα δούμε επιγραμματικά κάποια συνηθισμένα προβλήματα που προκύπτουν στο σχεδιασμό κρυπτογραφικών αλγορίθμων.

### 9.5.3 Αλγόριθμοι και Τετραγωνικά Υπόλοιπα

*Πρόταση* 9.39. Για τα παρακάτω υπάρχουν "αποδοτικοί" αλγόριθμοι.

1. Δίνεται $m, n$, να υπολογιστεί το $(\frac{m}{n})$.
2. Δίνεται $p$, να βρεθεί $a$ με $(\frac{a}{p}) = 1$.
3. Δίνεται $a, p$ με $(\frac{a}{p}) = 1$, να βρεθεί $x$ με $x^2 \equiv a \pmod p$.
4. Δίνεται $a, p, q$, να βρεθούν οι τετραγωνικές ρίζες του $a \pmod n$, όπου $n = pq$.

*Απόδειξη.* Για το 1. μπορούμε είτε να χρησιμοποιήσουμε την πρόταση 9.35 αν το $n$ είναι πρώτος, είτε πιο γενικά μπορεί να βρεθεί αλγόριθμος που χρησιμοποιεί το νόμο της τετραγωνικής αντιστροφής. Και στις δύο περιπτώσεις η πολυπλοκότητα είναι $O(\log_2(n)^3)$. Για το 2. έχουμε από την πρόταση 9.34 ότι η πιθανότητα ένας τυχαίος αριθμός στο $U(\mathbf{Z}_p)$ να είναι τετραγωνικό υπόλοιπο είναι $\frac{1}{2}$. 'Ετσι μπορούμε να κατασκευάσουμε ένα πιθανοτικό αλγόριθμο που να δίνει ένα τετραγωνικό υπόλοιπο modulo $p$ κάνοντας δοκιμές. Για το 3. υπάρχει αποδοτικός (πιθανοτικός) αλγόριθμος που σε πολυωνυμικό χρόνο απαντά σε αυτό το πρόβλημα, βλέπε [Koblitz] σελίδες 48-50. Για το 4. μπορούμε να χρησιμοποιήσουμε τον αλγόριθμο του 3. και να συνδυάσουμε τις λύσεις με το Κινέζικο Θεώρημα Υπολοίπων όπως στην πρόταση 9.32. Προσοχή, αν οι $p, q$ δεν είναι γνωστοί τότε το πρόβλημα είναι δύσκολο. □

## 9.6 Διαδραστικό Υλικό – Σύνδεσμοι

- Βιντεο-παρουσίαση Θεωρήματος Fermat: https://www.khanacademy.org/math/recreational-math/math-warmup/number-theory-warmups/v/fermat-s-little-
- Βίντεο σχετικά με συνάρτηση του Euler: https://www.youtube.com/watch?v=Rp8-ZAHPbg4
- Online Υπολογισμός GCD: http://www.mathportal.org/calculators/numbers-calcul gcd-lcm-calculator.php
- Online Υπολογισμός Συνάρτησης του Euler σε Javascript
- Βιβλιοθήκες Θεωρίας Αριθμών σε C++ από τον Victor Shoup: http://www.shoup.net/ntl/
- Βιβλιοθήκες Θεωρίας Αριθμών σε Python: http://userpages.umbc.edu/~rcampbel/Computers/Python/numbthy.html
- Παραγοντοποίηση με χρήση Javascript
- Βίντεο για έλεγχο Πρώτων Αριθμών
- Διαδραστικός Υπολογιστής Αλγόριθμου Ευκλείδη και χρήση του για εύρεση αντιστρόφου

## 9.7 Ασκήσεις

1. Υπολογίστε τον ΜΚΔ των αριθμών $5^{56} - 1$ και $5^{72} - 1$
2. Αποδείξτε ότι αν $a > 1$ και $m, n$ ακέραιοι, τότε $gcd(a^m - 1, a^n - 1) = a^{gcd(m,n)} - 1$.
3. Αποδείξτε ότι για κάθε $n > 2$ τουλάχιστον ένας από τους $2^n - 1$ και $2^n + 1$ είναι σύνθετος.
4. Να αποδειχθεί ότι $\phi(mn) = \phi(m)\phi(n)$ για $m, n$ σχετικά πρώτους.

5. Να αποδειχθεί η ιδιότητα της συνάρτησης $\phi$ του Euler: $\phi(p^a) = p^a(1-\frac{1}{p})$ όπου $p$ πρώτος.

6. Υπολογίστε τις τετραγωνικές ρίζες του 119 modulo 209. Χρησιμοποιήστε μεθόδους της θεωρίας αριθμών αλλά και εμπειρικές παρατηρήσεις. Υπόδειξη: θα σας βοηθήσει το Κινέζικο Θεώρημα Υπολοίπων.

7. Αποδείξτε ότι αν $p, q$ διαφορετικοί πρώτοι, τότε $p^{q-1} + q^{p-1} \equiv \pmod{pq}$.

8. Αποδείξτε ότι $N$ πρώτος αν και μόνο αν $(N-1)! \equiv -1 \pmod{N}$. (Υπόδειξη: για το ευθύ θεωρήστε ζεύγη αντιστρόφων στο $\mathbb{Z}_p^*$, για το αντίστροφο εξετάστε τον $gcd(N, (N-1)!)$ για $N$ σύνθετο.)

9. Αποδείξτε ότι αν $p$ πρώτος, τότε $a^p + b^p \equiv (a+b)^p \pmod{p}$.

10. Υλοποιήστε τον αλγόριθμο του Κινέζικου Θεωρήματος Υπολοίπων, δηλαδή με δεδομένες τις ισοτιμίες modulo δοσμένους πρώτους να υπολογίζει τον μοναδικό αριθμό modulo το γινόμενο των πρώτων.

11. Υπολογίστε τα $(\frac{19}{61}), (\frac{17}{31}), (\frac{107}{117})$ χρησιμοποιώντας μόνο το θεώρημα τετραγωνικής αντιστροφής καθώς και άλλες ιδιότητες των συμβόλων, χωρίς χρήση παραγοντοποίησης (εκτός με το 2)

12. Από το κινέζικο θεώρημα υπολοίπων γνωρίζουμε, πώς να λύσουμε το σύστημα

$$x \equiv a_1 \pmod{m_1}$$

$$x \equiv a_2 \pmod{m_2},$$

στην περίπτωση που $gcd(m_1, m_2) = 1$. Τι μπορούμε να πούμε όταν $gcd(m_1, m_2) > 1$; Πότε έχει λύση το σύστημα; Πως λύνουμε ένα τέτοιο σύστημα όταν δίνονται περισσότερες από δύο εξισώσεις;

# Βιβλιογραφία


[1] V. Shoup: “A Computational Introduction to Number Theory and Algebra”: http://shoup.net/ntb/

[2] D. Stinson: “Cryptography: Theory and Practice”, 3rd edition, CRC Press, 2005.

[3] A. Menezes, P. van Oorschot, and C. Vanstone: “Handbook of Applied Cryptography”: http://www.cacr.math.uwaterloo.ca/hac

[4] Neal Koblitz.“ A Course in Number Theory and Cryptography”. Second Edition. Springer-Verlag, 1994.

[5] Ε. Ζάχος, Α. Παγουρτζής, “Θεωρία Αριθμών και Εφαρμογές στην Κρυπτογραφία”, Σημειώσεις, ΕΜΠ, 2014.

# Κεφάλαιο 10

# Μαθηματική Λογική

## 10.1 Προτασιακή Λογική

Η γλώσσα της μαθηματικής λογικής στηρίζεται βασικά στις εργασίες του Boole και του Frege. Ο Προτασιακός Λογισμός περιλαμβάνει στο αλφάβητό του, εκτός από τα σύμβολα προτασιακών μεταβλητών, τα λογικά σημάδια ζεύξης: $\wedge$ (and), $\vee$ (or), $\neg$ (not), $\rightarrow$ (implies), $\leftrightarrow$ (equivalent), …

Στον προτασιακό λογισμό ονομάζουμε ατομικούς τύπους τις σταθερές TRUE και FALSE καθώς και τις προτασιακές μεταβλητές π.χ. $x_1, x_2, \ldots$ Οι **προτασιακοί τύποι** ορίζονται **επαγωγικά**:

1. Οι ατομικοί τύποι είναι τύποι.
2. Αν $\Phi$ είναι τύπος τότε και ο $\neg\Phi$ είναι τύπος.
3. Αν οι $\Phi$ και $\Psi$ είναι τύποι τότε και οι $(\Phi \wedge \Psi)$ και $(\Phi \vee \Psi)$ είναι τύποι.
4. Ο,τιδήποτε δεν ορίζεται με βάση τα (1)–(3) δεν είναι προτασιακός τύπος.

Παρατηρήσεις:

- Μερικές φορές παραλείπουμε παρενθέσεις και υποθέτουμε αριστερό προσεταιρισμό π.χ. $x_1 \wedge x_2 \wedge \neg x_3$
- Μπορούμε να ορίσουμε νέους τύπους ως **συντομογραφία** άλλων γνωστών π.χ.:

$$(\Phi \rightarrow \Psi) :\equiv (\neg\Phi \vee \Psi)$$
$$(\Phi \leftrightarrow \Psi) :\equiv (\Phi \rightarrow \Psi) \wedge (\Psi \rightarrow \Phi)$$

Οι προτασιακοί τύποι είναι συντακτικές συμβολοσειρές που όμως έχουν κάποια σημασία (**σημασιολογία**) δηλαδή είναι αληθείς ή ψευδείς ανάλογα με τις αληθοτιμές που έχουν απονεμηθεί στις προτασιακές μεταβλητές. Πιο συγκεκριμένα αληθοτιμές των τυπών $\neg\Phi$, $(\Phi \wedge \Psi)$ και $(\Phi \vee \Psi)$ ορίζονται από τις αληθοτιμές των $\Phi$, $\Psi$ όπως φαίνεται στον παρακάτω πίνακα αληθείας (truth table):

| $\Phi$ | $\Psi$ | $\neg\Phi$ | $(\Phi \wedge \Psi)$ | $(\Phi \vee \Psi)$ |
|---|---|---|---|---|
| TRUE | TRUE | FALSE | TRUE | TRUE |
| TRUE | FALSE | | FALSE | TRUE |
| FALSE | TRUE | TRUE | FALSE | TRUE |
| FALSE | FALSE | | FALSE | FALSE |

Ένας τύπος λέγεται **έγκυρος** (valid) ή **ταυτολογία** αν είναι αληθής για κάθε απονομή αληθοτιμών στις μεταβλητές. Ένας τύπος λέγεται **ικανοποιήσιμος** (satisfiable) αν υπάρχει απονομή αληθοτιμών που τον καθιστά αληθή. 'Αρα $\Phi$ είναι ικανοποιήσιμος **εάν και μόνο εάν** ο $\neg\Phi$ δεν είναι ταυτολογία.

Μια προτασιακή μεταβλητή ή άρνηση προτασιακής μεταβλητής ονομάζεται **λέκτημα** (literal). Μια **φράση** (clause) είναι μια διάζευξη από literals (π.χ. $x_1 \vee \neg x_2 \vee x_3 \vee \neg x_4$).

Κάθε τύπος της προτασιακής λογικής είναι ισοδύναμος με κάποιον που βρίσκεται σε **συζευκτική κανονική μορφή** (conjunctive normal form) δηλαδή είναι μια σύζευξη από διαζευκτικές φράσεις. Αντίστοιχα είναι επίσης ισοδύναμος με τύπο που βρίσκεται σε **διαζευκτική** κανονική μορφή (disjunctive normal form) δηλαδή είναι μια διάζευξη από συζευκτικές φράσεις. Μια φράση λέγεται **φράση Horn** αν έχει το πολύ ένα **θετικό** literal δηλαδή είναι της μορφής:

$$(x_0 \vee \neg x_1 \vee \neg x_2 \vee ... \vee \neg x_n) \text{ ή } (x_0) \text{ ή } (\neg x_1 \vee \neg x_2 \vee ... \vee \neg x_n)$$

που γράφεται ισοδύναμα:

$$(x_1 \wedge x_2 \wedge \cdots \wedge x_n \rightarrow x_0), (\text{TRUE} \rightarrow x_0), (x_1 \wedge x_2 \wedge ... \wedge x_n \rightarrow \text{FALSE}),$$

αντίστοιχα.

## 10.2 Κατηγορηματικός Λογισμός

Ο κατηγορηματικός λογισμός έχει επί πλέον σύμβολα γιά μεταβλητές, σταθερές, κατηγορήματα, συναρτήσεις και τους ποσοδείκτες: $\exists$ (υπαρξιακός) και $\forall$ (καθολικός).

Για να ορίσουμε συντακτικά τις προτάσεις είναι αναγκαίο πρώτα να ορίσουμε τους **όρους** (οι οποίοι όταν τους αποδοθεί σημασία θα ερμηνεύονται σαν **αντικείμενα** από κάποιο σύνολο). Οι όροι ορίζονται επαγωγικά:

1. Οι μεταβλητές και οι σταθερές είναι όροι.
2. Αν $t_1, t_2, ..., t_n$ είναι όροι και $f$ είναι σύμβολο συνάρτησης $n$ θέσεων τότε f($t_1, t_2, ..., t_n$) είναι επίσης όρος.
3. Τίποτε άλλο δεν είναι όρος.

Οι **προτάσεις** του κατηγορηματικού λογισμού ορίζονται επαγωγικά:

1. Αν $t_1, t_2, ..., t_n$ είναι όροι και $P$ είναι σύμβολο κατηγορήματος $n$ θέσεων τότε $P(t_1, t_2, ..., t_n)$ είναι πρόταση (ατομική πρόταση).

2. Αν οι Φ και Ψ είναι προτάσεις και x είναι μεταβλητή τότε και οι $\neg\Phi$, $(\Phi \wedge \Psi)$, $(\Phi \vee \Psi)$, $\forall x\Phi$, $\exists x\Phi$ είναι προτάσεις.

3. Τίποτε άλλο δεν είναι πρόταση.

Οι σταθερές και οι μεταβλητές **ερμηνεύονται** σαν στοιχεία ενός συνόλου $A$. Τα συναρτησιακά σύμβολα ερμηνεύονται σαν συναρτήσεις: $A^n \to A$. Έτσι κάθε όρος ερμηνεύεται σαν ένα στοιχείο του $A$.

Τα κατηγορήματα ερμηνεύονται σαν υποσύνολα του $A^n$. Η πρόταση
$P(t_1, t_2, ..., t_n)$ είναι αληθής ανν $(s_1, s_2, ..., s_n) \in R$ όπου $s_1, s_2, ..., s_n$ είναι τα στοιχεία του Α με τα οποία ερμηνεύονται οι όροι $t_1, t_2, ..., t_n$ και $R$ το υποσύνολο με το οποίο ερμηνεύεται το $P$. Οι αληθοτιμές των $\neg\Phi$, $(\Phi \wedge \Psi)(\Phi \vee \Psi)$ ορίζονται από τις αληθοτιμές των Φ και Ψ όπως και στην προτασιακή λογική. Η πρόταση $\forall x\Phi$ είναι αληθής αν η πρόταση Φ είναι αληθής για **οποιαδήποτε** ερμηνεία της μεταβλητής $x$, ενώ η πρόταση $\exists x\Phi$ είναι αληθής αν η Φ αληθεύει για **κάποια** ερμηνεία της $x$.

Οι φράσεις Horn για τον κατηγορηματικό λογισμό ορίζονται όπως και στην προτασιακή λογική αν αντί για προτασιακές μεταβλητές χρησιμοποιούμε ατομικές προτάσεις. Ένα πρόγραμμα $Prolog$ είναι βασικά μία σύξευξη από φράσεις Horn.

## 10.3 Πρωτοβάθμια Λογική

Όπως είπαμε και προηγουμένως η γλώσσα της πρωτοβάθμιας λογικής (ή αλλιώς κατηγορηματικού λογισμού) περιέχει:

- όλα τα σύμβολα που περιέχει ο προτασιακός λογισμός
- επιπλέον σύμβολα για συναρτήσεις και σταθερές, π.χ. $f, g, h, c_1, c_2, ...,$
- σύμβολα για κατηγορήματα π.χ. $P, Q, =, ...$
- και τους ποσοδείκτες : καθολικό $\forall$ και υπαρξιακό $\exists$.

Οι μεταβλητές εδώ ερμηνεύονται σαν στοιχεία κάποιου συνόλου, όχι σαν αληθοτιμές.

Θα ορίσουμε **επαγωγικά** τους **όρους** και τους **τύπους** της πρωτοβάθμιας λογικής.

**Όροι**:

1. Οι μεταβλητές και οι σταθερές είναι όροι.
2. Αν $f$ είναι σύμβολο συνάρτησης $n$ θέσεων και $t_1, ..., t_n$ είναι όροι τότε όρος είναι και ο $f(t_1, ..., t_n)$.
3. Τίποτα άλλο.

**Τύποι**:

1. Αν $P$ είναι σύμβολο κατηγορήματος $n$ θέσεων και $t_1, ..., t_n$ είναι όροι τότε $P(t_1, ..., t_n)$ και $t_1 = t_2$ είναι ατομικοί τύποι.
2. Αν οι $\Phi$ και $\Psi$ είναι τύποι και $x$ μεταβλητή τότε τύποι είναι και οι: $\neg\Phi$, $(\Phi \vee \Psi)$, $(\Phi \wedge \Psi)$, $\forall x\Phi$, $\exists x\Phi$.
3. Τίποτα άλλο.

Σημειώση: μια σταθερά $c$ μπορεί να θεωρηθεί συνάρτηση 0 θέσεων.

Η **εμβέλεια** του $\forall x$ (ή $\exists x$) στον τύπο $\forall x\Phi$ (ή αντίστοιχα $\exists x\Phi$ ) είναι ο **υποτύπος** $\Phi$.

**Ελεύθερη εμφάνιση** της μεταβλητής $x$ στον τύπο $\Phi$ λέγεται μια εμφάνιση της μεταβλητής $x$ που δεν είναι μέσα στην εμβέλεια ενός ποσοδείκτη $\forall x$ ή $\exists x$.
**Δεσμευμένη εμφάνιση** της $x$ είναι μέσα στην εμβέλεια ενός ποσοδείκτη ή και ακριβώς δεξιά του συμβόλου $\forall$ (ή $\exists$).

Ένας τύπος λέγεται **κλειστός** αν δεν περιέχει ελεύθερες εμφανίσεις μεταβλητών.

Η **σημασιολογία** τύπων του κατηγορηματικού λογισμού δίνεται με την βοήθεια των αλγεβρικών δομών $A$ που ονομάζουμε **μοντέλα**. Στην περίπτωση του προτασιακού λογισμού το πεδίο $A$ είναι $\{\text{True}, \text{False}\}$, εδώ όμως μπορεί να είναι οποιοδήποτε μή κενό, πεπερασμένο ή και άπειρο, σύνολο. Εδώ λοιπόν δεν έχουμε **απονομή** αληθοτιμών αλλά **ερμηνεία** (interpretation) των μεν σταθερών και μεταβλητών με στοιχεία του πεδίου $A$, των δε συναρτησιακών και κατηγορηματικών συμβόλων με πραγματικές **απεικονίσεις** και **σχέσεις** μεταξύ των στοιχείων του πεδίου $A$. Με τέτοια σημασιολογία κάθε όρος ερμηνεύεται με στοιχείο του $A$ και κάθε κλειστός τύπος αληθεύει (ή όχι) στο μοντέλο $A$.

Συμβολίζουμε $\Gamma \vdash \Phi$ το γεγονός ότι ο τύπος $\Phi$ αποδεικνύεται **συντακτικά** από τους τύπους του συνόλου $\Gamma$.
Συμβολίζουμε $\Gamma \models \Phi$ το γεγονός ότι ο τύπος $\Phi$ αληθεύει σε όλα τα μοντέλα όπου αληθεύουν και οι τύποι του συνόλου $\Gamma$.

Το περίφημο **θεώρημα πληρότητας** του Gödel λέει:

$$\Gamma \vdash \Phi \quad \text{ανν} \quad \Gamma \models \Phi$$

Αφ' ετέρου το **θεώρημα μη πληρότητας** του Gödel λέει:

Δεν μπορεί να υπάρξει **συνεπής και πλήρης αξιωματικοποίηση** όλων των αληθών τύπων της Αριθμητικής.

## 10.4 Διαδραστικό υλικό - Σύνδεσμοι

- Στις σελίδες http://world.logic.at/ και http://settheory.net/world θα βρείτε σύνδεσμους για βιβλιογραφικές πηγές και συνέδρια Λογικής.
- Εδώ θα βρείτε ένα "κομπιουτεράκι" λογικής.
- Στην σελίδα http://www.ee.umd.edu/ yavuz/logiccalc.html θα βρείτε μια γεννήτρια πινάκων αλήθειας για λογικούς τύπους.

## 10.5 Ασκήσεις

1. Χρησιμοποιώντας πίνακες αληθείας ελέξτε αν είναι ισοδύναμες οι $A$ και $B$.
   $A$: Θα βρέξει ή ο ήλιος θα λάμπει.
   $B$: Το εξής είναι λάθος: Δε θα βρέξει και ο ήλιος δε θα λάμπει.

2. Χρησιμοποιήστε πίνακες αλήθειας για να ελέγξετε τις ισοδυναμίες:

   (α) $\phi \vee \psi \Leftrightarrow \psi \vee \phi$

   (β) $(\phi_1 \vee \phi_2) \vee \phi_3 \Leftrightarrow \phi_1 \vee (\phi_2 \vee \phi_3)$

3. Ποιοι από τους παρακάτω είναι σωστά σχηματισμένοι προτασιακοί τύποι; Εξηγήστε.
   (α) $(\phi \rightarrow \psi) \wedge (\neg\phi)$
   (β) $\phi \rightarrow \wedge(\neg\phi \vee \psi)$
   (γ) $\neg p \vee (p \vee q)$
   (δ) $(\neg(A \vee B)\neg C)$

4. Θα ορίσουμε τον λογικό σύνδεσμο $NAND$: Η πρόταση $\phi\ NAND\ \psi$ είναι αληθής αν τουλάχιστον ένας από τους $\phi, \psi$ είναι ψευδής, και είναι ψευδής όταν οι $\phi$ και $\psi$ είναι αληθείς.

   (α) Κατασκευάστε έναν πίνακα αλήθειας για τον σύνδεσμο $NAND$.

   (β) Δείξτε ότι η πρόταση $\phi\ NAND\ \psi$ είναι λογικά ισοδύναμη με την $\neg(\phi \wedge \psi)$.

5. Υποθέστε ότι η εξής πρόταση είναι αληθής: "Αν διαβάζω πολύ, θα πάρω 10."
   Προσδιορίστε για καθεμία από τις παρακάτω προτάσεις αν είναι αληθής, ψευδής, ή δεν ξέρουμε.
   $p$: Αν πάρω 20, τότε διαβάζω πολύ.
   $q$: Αν δεν διαβάζω πολύ, δεν θα πάρω 10.
   $r$: Αν δεν πάρω 10, τότε δεν διαβάζω πολύ.

# Βιβλιογραφία

[1] Martin Davis. “Μηχανές της Λογικής: Οι Μαθηματικοί και οι Απαρχές του Υπολογιστή”. Μετάφραση: Στάθης Ζάχος. Εκδόσεις Εκκρεμμές, 2007.

[2] Herbert Enderton. “A Mathematical Introduction to Logic”. Academic Press; 2 edition (January 5, 2001)

# Κεφάλαιο 11

# Υπολογισιμότητα και Πολυπλοκότητα

## 11.1 Ιστορία - Εισαγωγή

Η Συλλογιστική του Αριστοτέλη αποτέλεσε την πρώτη προσπάθεια θεμελίωσης της λογικής και των μαθηματικών. Ο *Leibni(t)z* πρότεινε το εξής πρόγραμμα:

1. Να δημιουργηθεί μια τυπική γλώσσα (*formal language*), με την οποία να μπορούμε να περιγράψουμε όλες τις μαθηματικές έννοιες και προτάσεις.
2. Να δημιουργηθεί μια μαθηματική θεωρία (δηλαδή ένα σύνολο από αξιώματα και συμπερασματικούς κανόνες συνεπαγωγής), με την οποία να μπορούμε να αποδεικνύουμε όλες τις ορθές μαθηματικές προτάσεις.
3. Να αποδειχθεί ότι αυτή η θεωρία είναι συνεπής (*consistent*), (δηλαδή ότι η πρόταση "Α και όχι Α" ($A \wedge \neg A$) δεν είναι δυνατόν να αποδειχθεί σ' αυτή τη θεωρία).

Η πραγμάτωση αυτού του προγράμματος άρχισε πολύ αργότερα, προς το τέλος του 19ου αιώνα. Πολλοί επιστήμονες ασχολήθηκαν με τον ορισμό της ενιαίας γλώσσας της μαθηματικής (ή συμβολικής) λογικής (*Boole, Frege*, κ.α). 'Αλλοι ασχολήθηκαν με τον ορισμό της ενιαίας θεωρίας των συνόλων (*Cantor*, κ.α.) και άλλοι με την παραγωγή (*derivation*) όλων των αληθών μαθηματικών προτάσεων με χρήση της Συνολοθεωρίας (*Russel, Whitehead*, κ.α.).

Στην αρχή αυτού του αιώνα ο *Hilbert* βάλθηκε να πραγματοποιήσει το 3ο μέρος του προγράμματος του *Leibni(t)z*, δηλαδή να βρει έναν αλγόριθμο που να αποκρίνεται (*decides*) για την ορθότητα κάθε μαθηματικής πρότασης. Τελικά, όμως, το 1931 ο *Gödel* απέδειξε ότι:

- Δεν υπάρχει τέτοιος αλγόριθμος.
- Είναι αδύνατον να αποδειχθεί η συνέπεια της Συνολοθεωρίας.
- Επιπλέον, οποιαδήποτε (δηλαδή όχι μόνο η Συνολοθεωρία) αξιωματική θεωρία των Μαθηματικών, που περιλαμβάνει τουλάχιστον την Αριθμοθεωρία, θα περιλαμβάνει και μη αποκρίσιμες (*undecidable*) προτάσεις.

- Κωδικοποιώντας προτάσεις με φυσικούς αριθμούς (αυτή η κωδικοποίηση λέγεται σήμερα "Γκεντελοποίηση": *Gödelization*) μπόρεσε να παρουσιάσει μια συγκεκριμένη πρόταση που είναι μη αποκρίσιμη.

Το αποτέλεσμα αυτό του *Gödel* ήταν η αιτία μιας σημαντικής κρίσης στα κλασσικά μαθηματικά, μα συγχρόνως και η απαρχή των μοντέρνων δυναμικών μαθηματικών. Το κεντρικό ερώτημα δεν είναι πια απλά αν μια πρόταση είναι αληθής η ψευδής, αλλά αν είναι "αποκρίσιμη ή μη αποκρίσιμη", δηλαδή αν είναι "υπολογιστή (*computable*) ή όχι". Αυτό ακριβώς είναι και το αντικείμενο της **Θεωρίας της Υπολογιστότητας** (*computability*). Αν δοθεί ότι μια συνάρτηση $f$ είναι υπολογιστή, ποιο είναι το κόστος ή τα αγαθά (*resources*) που χρειάζονται για να υπολογίσουμε την $f$; Αυτό είναι το βασικό ερώτημα της **Θεωρίας της Πολυπλοκότητας** (complexity)[1].

Διάφοροι επιστήμονες (*Turing, Church, Kleene, Post, Markov*, κ.α.) βάλθηκαν να ξεκαθαρίσουν τις έννοιες: υπολογιστό ή επιλύσιμο (*solvable*) με αλγόριθμο, υπολογιστή συνάρτηση και αποκρίσιμο πρόβλημα. Κατέληξαν, λοιπόν, σε διαφορετικά υπολογιστικά μοντέλα, τα οποία όμως αποδείχθηκαν όλα ισοδύναμα μεταξύ τους.

Η περίφημη **Θέση (thesis) των Church-Turing** λέει λοιπόν απλουστευμένα: "Όλα τα γνωστά και τα "άγνωστα" μοντέλα της έννοιας "υπολογιστός" είναι μηχανιστικά ισοδύναμα (*effectively equivalent*)". Δηλαδή δοθέντος ενός αλγορίθμου σε ένα μοντέλο για μια συγκεκριμένη συνάρτηση $f$, μπορούμε μηχανιστικά (με τη βοήθεια μηχανής) να κατασκευάσουμε αλγόριθμο σε ένα άλλο μοντέλο για την ίδια συνάρτηση $f$.

Ας χρησιμοποιήσουμε εδώ ένα γνωστό μοντέλο υπολογισμού, μια γλώσσα προγραμματισμού υψηλού επιπέδου. Μια συνάρτηση[2] τότε, θα λέγεται υπολογιστή, αν υπάρχει πρόγραμμα που υπολογίζει την τιμή της για κάθε όρισμα. Είναι προφανές ότι υπάρχουν συναρτήσεις μη υπολογιστές, γιατί:

- Υπάρχουν άπειρα μεν, αλλά μόνο αριθμήσιμα (*countable*) διαφορετικά προγράμματα. Εκτός αυτού μπορούμε χρησιμοποιώντας κωδικοποίηση να τα απαριθμήσουμε μηχανιστικά (*effectively enumerate*)[3]

[1] Συχνά χρησιμοποιείται και ο όρος υπολογίσιμος αντί για υπολογιστός

[2] θα πρέπει εδώ να διευκρινίσουμε ότι αναφερόμαστε σε συναρτήσεις $f : \mathbb{N}^n \to \mathbb{N}$, αφού τα δεδομένα σε ένα πραγματικό υπολογιστή κωδικοποιούνται με φυσικούς αριθμούς (ακολουθίες από 0 και 1). Γενικότερα, αναφερόμαστε σε συναρτήσεις από αριθμήσιμα σύνολα σε αριθμήσιμα σύνολα

[3] Απόδειξη: Κάθε πρόγραμμα μιας γλώσσας προγραμματισμού είναι στοιχείο του $\Sigma^*$, όπου $\Sigma = \{a_1, a_2, \ldots, a_m\}$ το αλφάβητο της γλώσσας. Το $\Sigma^*$ όμως αποτελεί την ένωση $\bigcup_{n=0}^{\infty} \Sigma_n$, όπου $\Sigma_n$ το σύνολο των συμβολοσειρών του αλφάβητου $\Sigma$ που έχουν μήκος $n$. Κάθε σύνολο $\Sigma_n$ είναι πεπερασμένο και έτσι αν διατάξουμε τα στοιχεία του αλφαβητικά μπορούμε να θεωρήσουμε την ακόλουθη αρίθμηση για το $\Sigma^*$:

$$\begin{array}{ll} \Sigma_0: & \{\varepsilon\} \\ \Sigma_1: & \{a_1, a_2, \ldots, a_m\} \\ \Sigma_2: & \{a_1a_1, a_1a_2, \ldots, a_1a_m, \ldots, a_ma_m\} \\ \vdots & \end{array}$$

Η παραπάνω αρίθμηση του $\Sigma^*$ είναι μηχανιστική, δηλαδή μπορεί να γίνει με πρόγραμμα. Επομένως, με κατάλληλη χρήση compiler για τον έλεγχο ορθότητας μπορούμε να κατασκευάσουμε μηχανιστική αρίθμηση των συντακτικά ορθών $n$ προγραμμάτων

- Από την άλλη μεριά όμως, ξέρουμε ότι υπάρχουν μη αριθμήσιμες άπειρες (*uncountable*) διαφορετικές συναρτήσεις. Αυτό αποδεικνύεται με διαγωνιοποίηση (*diagonalization*), ανάλογη με αυτή που χρησιμοποιούμε για να δείξουμε ότι το σύνολο $\mathbb{R}$ είναι μη αριθμήσιμο[4]

Ας στραφούμε σε ένα συγκεκριμένο πρόβλημα που είναι μη αποκρίσιμο. Όπως ξέρουμε ο μεταγλωττιστής (*compiler*) μιας γλώσσας προγραμματισμού μπορεί να ελέγξει αν ένα πρόγραμμα είναι συντακτικά ορθό ή όχι. Τι γίνεται όμως με τα λάθη χρόνου εκτέλεσης (*run time errors*); Θα ήταν ωραίο να είχαμε ένα πρόγραμμα που θα μπορούσε να ελέγχει αν ένα συντακτικά ορθό πρόγραμμα θα σταματήσει κάποτε ή αν θα τρέχει για πάντα. Δυστυχώς τέτοιο πρόγραμμα δεν υπάρχει.

*Θεώρημα* 11.1. *Το **halting problem (HP)** είναι μη αποκρίσιμο.*

*Απόδειξη.* Έστω ότι $\pi_0, \pi_1, \pi_2, \ldots$ είναι μια μηχανιστική απαρίθμηση (*effective enumeration*) όλων των προγραμμάτων. Ας υποθέσουμε ότι το **HP** είναι επιλύσιμο. Τότε κατασκευάζουμε ένα πρόγραμμα $\pi$, που ελέγχει αν το πρόγραμμα $\pi_n$ με είσοδο $n$ σταματάει ή όχι και ανάλογα με την απάντηση σε αυτόν τον έλεγχο, το πρόγραμμα $\pi$ σταματάει αν το $\pi_n(n)$ δεν σταματάει, και αντιστρόφως:

$$\pi\colon \text{read}(n); \textbf{if } \pi_n(n) \text{ terminates } \textbf{then } \text{loop\_forever } \textbf{else } \text{halt}$$

Φυσικά αυτό το πρόγραμμα $\pi$ κάπου θα εμφανίζεται στην παραπάνω αρίθμηση. Ας πούμε ότι ο δείκτης για το $\pi$ είναι $i$, δηλαδή $\pi = \pi_i$. Η ιδέα της διαγωνιοποίησης είναι να δώσουμε το δείκτη $i$ για input στο $\pi_i$. Τότε το $\pi_i(i)$ σταματάει αν και μόνο αν το $\pi(i)$ σταματάει και αυτό συμβαίνει αν και μόνο αν το $\pi_i(i)$ δεν σταματάει. Αντίφαση. □

Τελικά πολλά άλλα προβλήματα είναι επίσης μη επιλύσιμα. Αν και το HP δεν είναι επιλύσιμο, μπορούμε να κατασκευάσουμε με μηχανιστικό τρόπο μια άπειρη λίστα όλων των προγραμμάτων, με την αντίστοιχη είσοδο για την οποία σταματούν. Αυτό δεν σημαίνει φυσικά ότι μπορούμε να επιλύσουμε το HP, γιατί αν για παράδειγμα το $\pi_k(n)$ δεν έχει εμφανισθεί στη λίστα μας, δεν ξέρουμε αν θα προστεθεί στη λίστα αργότερα ή αν δε θα εμφανισθεί ποτέ στη λίστα. Για να ακριβολογούμε λίγο περισσότερο δίνουμε τους παρακάτω ορισμούς.

*Ορισμός* 11.2. Ένα σύνολο $S$ λέγεται αποκρίσιμο ή υπολογιστό ή επιλύσιμο (*decidable, computable, solvable*) αν και μόνο αν υπάρχει ένας αλγόριθμος που σταματάει ή μια υπολογιστική μηχανή που δίνει έξοδο "ναι" για κάθε είσοδο $a \in S$ και έξοδο "όχι" για κάθε είσοδο $a \notin S$.

*Ορισμός* 11.3. Ένα σύνολο $S$ λέγεται καταγράψιμο (με μηχανιστική γεννήτρια) (*listable, effectively generatable*) αν και μόνο αν υπάρχει μια γεννήτρια διαδικασία ή μηχανή που καταγράφει όλα τα στοιχεία του $S$. Στην, πιθανώς άπειρη, λίστα εξόδου επιτρέπονται οι επαναλήψεις και δεν υπάρχει περιορισμός για την διάταξη των στοιχείων.

[4] Απόδειξη: Ας θεωρήσουμε το σύνολο των ολικών συναρτήσεων $\phi\colon \mathbb{N} \to \mathbb{N}$ και έστω $\phi_0, \phi_1, \phi_2, \ldots$ μια αρίθμηση τους (ολικές ονομάζονται οι συναρτήσεις που ορίζονται για κάθε $x \in \mathbb{N}$). Ορίζουμε μια συνάρτηση $f$ ως εξής: $f(x) = \phi_x(x) + 1, \forall x \in \mathbb{N}$. Η $f$ είναι προφανώς ολική συνάρτηση και επομένως θα αντιστοιχίζεται σε κάποιο δείκτη $y$ στην παραπάνω αρίθμηση μας, δηλαδή $f = \phi_y$. Τότε όμως θα ισχύει ότι $\phi_y(y) = f(y) = \phi_y(y) + 1$ που είναι άτοπο. Επομένως το σύνολο των ολικών συναρτήσεων δεν είναι αριθμήσιμο.

Μερικές απλές ιδιότητες:

- Αν το $S$ είναι αποκρίσιμο τότε και το $\overline{S}$ είναι αποκρίσιμο.
- Αν το $S$ είναι αποκρίσιμο τότε το $S$ είναι και καταγράψιμο.
- Αν το $S$ και το $\overline{S}$ είναι καταγράψιμα τότε το $S$ είναι αποκρίσιμο.
- Αν το $S$ είναι καταγράψιμο με γνησίως αύξουσα διάταξη τότε το $S$ είναι αποκρίσιμο.

## 11.2 Υπολογιστικά μοντέλα

Λόγω της θέσης των Church-Turing δεν χρειάζεται να καθορίσουμε ένα συγκεκριμένο υπολογιστικό μοντέλο για τη λύση κάποιου προβλήματος: όλα τα ντετερμινιστικά υπολογιστικά μοντέλα είναι ισοδύναμα μεταξύ τους, με την έννοια ότι αν ένα πρόβλημα λύνεται από κάποιο υπολογιστικό μοντέλο, τότε θα λύνεται και από οποιοδήποτε άλλο, με το πολύ πολυωνυμική απώλεια χρόνου. Μερικά υπολογιστικά μοντέλα είναι τα εξής:

- προγράμματα Pascal
- προγράμματα Pascal χωρίς αναδρομή (αφαίρεση αναδρομής με χρήση στοίβας)
- προγράμματα Pascal χωρίς αναδρομή και χωρίς άλλους τύπους δεδομένων εκτός από τους φυσικούς αριθμούς (επιτυγχάνεται με κωδικοποιήσεις)
- προγράμματα WHILE ( μόνη δομή ελέγχου το WHILE )
- προγράμματα GOTO και IF
- Assembler-like RAM (random access machine), URM (universal register machine)
- SRM (single register machine) ένας καταχωρητής
- Μηχανή Turing (πρόσβαση μόνο σε μια κυψέλη ”cell” της ταινίας κάθε φορά)

Τα χαρακτηριστικά των παραπάνω μοντέλων είναι:

- ντετερμινιστική πολυπλοκότητα σε διακριτά βήματα
- πεπερασμένο σύνολο εντολών που εκτελούνται από επεξεργαστή
- απεριόριστη μνήμη

’Αλλα μοντέλα είναι:

- παραλλαγές από μηχανές Turing
- Thue: κανόνες επανεγγραφής (re-writing rules)

- Post: κανονικά συστήματα (normal systems)
- Church: λογισμός λ (λ-calculus)
- Curry: συνδυαστική λογική (combinatory logic)
- Markov: M. αλγόριθμοι
- Kleene: γενικά αναδρομικά σχήματα (general recursive schemes)
- Shepherdson-Sturgis, Elgott: URM, SRM, RAM, RASP
- Σχήματα McCarthy (if ... then ... else ... $\Rightarrow$ LISP)

*Θεώρημα* 11.4. *$f$ είναι TM υπολογιστή ανν*

- *$f$ είναι WHILE-υπολογιστή*
- *$f$ είναι GOTO-υπολογιστή*
- *$f$ είναι PASCAL-υπολογιστή*
- *$f$ είναι μερικά αναδρομική (partial recursive)*

Παραλλαγές Μηχανών Turing που έχουν την ίδια υπολογιστική δυνατότητα, όχι όμως και αποδοτικότητα (*efficiency*) είναι:

- πολλές ταινίες, μνήμη πλέγματος (grid memory), μνήμη περισσοτέρων διαστάσεων
- μεγαλύτερο $\Sigma$
- πολλές παράλληλες κεφαλές
- μη ντετερμινιστικές μεταβάσεις
- μίας κατευθύνσεως, απείρου μήκους ταινία
- εγγραφή και κίνηση της κεφαλής σε κάθε βήμα

## 11.3 Υπολογιστική Πολυπλοκότητα

Μια άλλη (πιο μοντέρνα) ταξινόμηση προβλημάτων (γλωσσών, συνόλων) σε κλάσεις μπορεί να γίνει με κριτήριο το **ποσόν** των αγαθών (χρόνος, χώρος, επεξεργαστές, κ.ο.κ.) που χρειάζεται ένας βέλτιστος αλγόριθμος για να τα επιλύσει (αναγνωρίσει).

Για να χρησιμοποιήσουμε όμως το χρόνο, χώρο, κ.τ.λ. για μέτρο πολυπλοκότητας χρειαζόμαστε επακριβείς ορισμούς του υπολογιστικού μοντέλου καθώς και του μεγέθους που μετράμε. Το κόστος ενός αλγορίθμου δεν είναι φυσικά μια σταθερά τιμή αλλά είναι συνάρτηση του μεγέθους

$n$ της εισόδου. Συνήθως μετράμε το κόστος της χειρότερης περίπτωσης για είσοδο μεγέθους $n$. Έτσι το κόστος του αλγορίθμου $(n)$ είναι το max του κόστους του αλγορίθμου από όλες τις δυνατές εισόδους μεγέθους $n$.

Το κόστος $C(n)$, τώρα, ενός προβλήματος $\Pi(n)$ είναι το $\min((n))$ από όλους τους αλγορίθμους που λύνουν το πρόβλημα Π. Συνεπώς για να προσδιορίσουμε το κόστος $C(n)$ ενός προβλήματος $\Pi(n)$ χρειαζόμαστε ένα **άνω όριο** (upper bound) δηλαδή έναν αλγόριθμο που έχει κόστος $C(n)$ αλλά και ένα **κάτω όριο** (lower bound), δηλαδή μια απόδειξη ότι το καλύτερο δυνατό κόστος με το τρέχον μοντέλο είναι $C(n)$. Έτσι, π.χ., η χρονική πολυπλοκότητα ταξινόμησης με συγκρίσεις (όπου μοντέλο είναι πρόγραμμα Pascal, και μετράμε τον αριθμό συγκρίσεων) είναι $\Theta(n \log n)$.

Συνήθως ονομάζουμε αποδοτικό ένα αλγόριθμο αν ο χρόνος του είναι πολυωνυμικός ($n^{O(1)}$) ως προς το μέγεθος της εισόδου. **P** λέγεται η κλάση των προβλημάτων που λύνονται με ντετερμινιστικό αλγόριθμο σε χρόνο πολυωνυμικό. **NP** λέγεται η κλάση των προβλημάτων που λύνονται με μη ντετερμινιστικό αλγόριθμο σε χρόνο πολυωνυμικό. Λέμε ότι ένας μη ντετερμινιστικός αλγόριθμος λύνει ένα πρόβλημα Π εάν υπάρχει τουλάχιστον μια από τις δυνατές εκτελέσεις του Α που λύνει το Π. Προφανώς ισχύει **P** $\subseteq$ **NP** αλλά εδώ και τριάντα-πέντε χρόνια παραμένει άλυτο το πρόβλημα "**P** $\neq$ **NP**;". **NP**$-$πλήρη λέγονται τα δυσκολότερα προβλήματα της κλάσης **NP**. Αν πράγματι ισχύει **P** $\neq$ **NP**, τότε για τα **NP**$-$πλήρη προβλήματα δεν υπάρχει αλγόριθμος πολυωνυμικού χρόνου. **PSPACE** λέγεται η κλάση των προβλημάτων που λύνονται

με (ντετερμινιστικό ή μη ντετερμινιστικό αλγόριθμο) σε πολυωνυμικό χώρο (μνήμη).

Τέλος **NC** λέγεται η κλάση των προβλημάτων που λύνονται με αλγόριθμο που χρησιμοποιεί πολυλογαριθμικό χρόνο ($\log^{O(1)} n$) και πολυωνυμικό αριθμό επεξεργαστών.

Μερικά γνωστά προβλήματα σε αυτές τις κλάσεις:

- Στην κλάση **NP**: το πρόβλημα (SAT) ικανοποιησιμότητας τύπων της προτασιακής λογικής, το πρόβλημα (TSP) του πλανόδιου πωλητή, κ.τ.λ.
- Στην κλάση **PSPACE**: το πρόβλημα (QBF) αποτίμησης τύπων της κατηγορηματικής (boolean) λογικής, το πρόβλημα στρατηγικής σε διάφορα παιχνίδια, κ.τ.λ.
- Στην κλάση **NC**: το πρόβλημα (GAP) πρόσβασης (δηλαδή ύπαρξης μονοπατιού) σε ένα γράφο $G$ μεταξύ δυο κόμβων.
- Το πρόβλημα ικανοποιησιμότητας τύπων της κατηγορηματικής λογικής είναι μη επιλύσιμο αλλά καταγράψιμο.

Ισχύει:

$$\textbf{NC} \subseteq \textbf{P} \subseteq \textbf{NP} \subseteq \textbf{PSPACE} \subsetneq \textbf{REC} \subsetneq \textbf{R.E.}$$

*Παρατήρηση:*
**REC** = recursive = decidable
**R.E.** = recursively enumerable = listable

## 11.4 Αναγωγές μεταξύ προβλημάτων

Με τον όρο *αναγωγή* εννοούμε την μετατροπή ενός υπολογιστικού προβλήματος $\Pi_A$ σε ένα άλλο πρόβλημα $\Pi_B$ έτσι ώστε αν μπορούμε να λύσουμε το $\Pi_B$ να μπορούμε να λύσουμε και το $\Pi_A$. Παρά το ότι υπάρχουν διάφορα είδη αναγωγών, εμείς θα ασχοληθούμε με μια απλή μορφή, κατάλληλη για προβλήματα απόφασης (όπου η απάντηση είναι "ναι'' ή "όχι''):

**Ορισμός**. Λέμε ότι ένα πρόβλημα απόφασης $\Pi_A$ ανάγεται σε ένα πρόβλημα απόφασης $\Pi$, και γράφουμε $\Pi_A \leq \Pi_B$ αν υπάρχει μία υπολογίσιμη (computable) συνάρτηση $h$ που απεικονίζει ένα (οποιοδήποτε) στιγμιότυπο (έγκυρη είσοδο) $I_A$ του $\Pi$ σε ένα στιγμιότυπο (έγκυρη είσοδο) $I_B$ του $\Pi$ τέτοια ώστε:

$$\text{answer}_{\Pi_A}(I_A) = \text{"yes''} \iff \text{answer}_{\Pi_B}(I_B) = \text{"yes''}$$

Δηλαδή, η απάντηση για το πρόβλημα $\Pi_A$ με είσοδο $I_A$ είναι "ναι'' *αν και μόνο αν* η απάντηση για το πρόβλημα $\Pi_B$ με είσοδο $I_B = h(I_A)$ είναι "ναι''. [5]

Συχνά με τον όρο αναγωγή αναφερόμαστε τόσο στη συνάρτηση $h$ όσο και στον *αλγόριθμο* που την υπολογίζει.

### 11.4.1 Αναγωγές πολυωνυμικού χρόνου

Εάν η συνάρτηση $h$ μπορεί να υπολογιστεί αποδοτικά, δηλαδή σε πολυωνυμικό χρόνο, τότε λέγεται *αναγωγή πολυωνυμικού χρόνου*. Η ύπαρξη αναγωγής πολυωνυμικού χρόνου του $\Pi_A$ στο $\Pi_B$ συμβολίζεται ως εξής:

$$\Pi_A \leq^p \Pi_B$$

Συχνά χρησιμοποιούνται και οι συμβολισμοί $\Pi_A \leq_P \Pi_B$ και $\Pi_A \leq_m^p \Pi_B$.

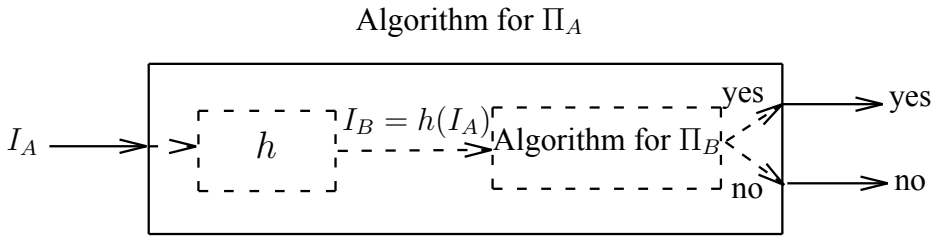
**Παράδειγμα: Hamilton Cycle $\leq^p$ TSP** Θα περιγράψουμε μια αναγωγή από το πρόβλημα Hamilton Cycle στο TSP.

Η ζητούμενη αναγωγή είναι, όπως αναφέρεται και πιο πάνω, μια συνάρτηση που απεικονίζει οποιοδήποτε στιγμιότυπο του Hamilton Cycle σε ένα στιγμιότυπο του TSP, ώστε η απάντηση για τα δύο στιγμιότυπα να είναι ίδια.

Για ευκολία, θα περιγράψουμε αναγωγή από το πρόβλημα Hamilton Cycle στο πρόβλημα U-TSP (Undirected TSP). [6] Θα δώσουμε τη διαδικασία μετατροπής ενός στιγμιοτύπου του Hamilton Cycle, που είναι απλά ένας μη κατευθυνόμενος γράφος $G$, σε ένα στιγμιότυπο του U-TSP, που

[5] Προσοχή στο 'αν και μόνο αν': το συνηθέστερο λάθος είναι να δείχνει κανείς την μία μόνο κατεύθυνση, δηλαδή το 'μόνο αν'. Αυτό όμως δεν μας εγγυάται ότι γνωρίζοντας την απάντηση για το $I_B$ θα γνωρίζουμε και την απάντηση για το $I_A$. Πιο συγκεκριμένα, μας εξασφαλίζει μόνο ότι αν η απάντηση για το $I_B$ είναι αρνητική τότε και η απάντηση για το $I_A$ είναι αρνητική. Αν όμως η απάντηση για το $I_B$ είναι καταφατική, μπορεί η απάντηση για το $I_A$ να είναι είτε καταφατική είτε αρνητική.

[6] Αυτό δεν δημιουργεί πρόβλημα αφού οι μη κατευθυνόμενοι γράφοι μπορούν να θεωρηθούν συμμετρικοί κατευθυνόμενοι. Επομένως, το πρόβλημα U-TSP είναι ειδική περίπτωση του προβλήματος TSP.




Σχήμα 11.1: Αναγωγή από $\Pi_A$ σε $\Pi_B$ + αλγόριθμος για $\Pi_B \Rightarrow$ αλγόριθμος για $\Pi_A$.

είναι ένας μη κατευθυνόμενος γράφος $G'$ με βάρη στις ακμές του, και ένα όριο κόστους (θυμηθείτε ότι μιλάμε για προβλήματα απόφασης: το ερώτημα στο πρόβλημα U-TSP είναι αν υπάρχει κύκλος Hamilton στον $G'$ με κόστος $\leq B$).

**Κατασκευή:** Ο γράφος $G'$ έχει το ίδιο πλήθος κόμβων με τον $G$, έστω $n$, αλλά είναι πλήρης. Σε κάθε ακμή του $G'$ που υπήρχε και στον αρχικό γράφο $G$ δίνουμε βάρος 1, ενώ στις υπόλοιπες δίνουμε βάρος 2. Θέτουμε σαν όριο κόστους $B = n$.

**Απόδειξη ορθότητας:** Θα δείξουμε ότι υπάρχει κύκλος Hamilton στον αρχικό γράφο $G$ **αν και μόνο αν** υπάρχει κύκλος Hamilton στον $G'$ με κόστος $\leq n$:
'μόνο αν': αν υπάρχει κύκλος Hamilton στον $G$ τότε ο ίδιος κύκλος στον $G'$ αποτελείται από ακμές βάρους 1, οπότε το συνολικό κόστος του κύκλου είναι $n$.
'αν': αν υπάρχει κύκλος Hamilton στον $G'$ με κόστος $\leq n$, αυτός θα πρέπει να αποτελείται αποκλειστικά από ακμές βάρους 1, αφού κάθε κύκλος Hamilton έχει $n$ ακμές. Επομένως αυτός ο κύκλος υπάρχει και στον $G$.

Χρειάζεται επιπλέον να δείξουμε ότι η αναγωγή γίνεται σε πολυωνυμικό χρόνο. Πράγματι, ένα πέρασμα του πίνακα γειτνίασης (ή των λιστών γειτνίασης) και αντικατάσταση των (μη διαγωνίων) 0 με 2 αρκεί, επομένως ο απαιτούμενος χρόνος είναι $O(n^2)$ ($O(m)$ με λίστες), δηλαδή πολυωνυμικός ως προς το μέγεθος της εισόδου.

### 11.4.2 Αναγωγές: δύο τρόποι χρήσης

Όπως είδαμε πιο πάνω, μια ορθή αναγωγή έχει την ιδιότητα οι απαντήσεις για τα δύο στιγμιότυπα ($I_A$ του προβλήματος $\Pi_A$ και $I_B$ του προβλήματος $\Pi_B$) να ταυτίζονται. Αυτό σημαίνει ότι αν έχουμε έναν αποδοτικό αλγόριθμο επίλυσης του $\Pi_B$ και μια αποδοτική αναγωγή από το $\Pi_A$ στο $\Pi_B$ τότε παίρνουμε έναν αποδοτικό αλγόριθμο επίλυσης του $\Pi_A$, όπως φαίνεται και στο Σχ. 11.1.

Η ιδιότητα αυτή των αναγωγών μπορεί να χρησιμοποιηθεί με δύο τρόπους, είτε με "θετικό" τρόπο, **προς επίλυση προβλήματος**, είτε με "αρνητικό", προς **απόδειξη δυσκολίας προβλήματος**.

Πιο συγκεκριμένα, μια αναγωγή πολυωνυμικού χρόνου του $\Pi_A$ στο $\Pi_B$ μπορεί να χρησιμοποιηθεί είτε για την εύρεση αποδοτικού αλγορίθμου για το πρόβλημα $\Pi_A$ (εάν γνωρίζουμε αλγόριθμο για το $\Pi_B$) είτε για την απόδειξη δυσκολίας του $\Pi_B$ (εάν διαθέτουμε απόδειξη δυσκολίας για το $\Pi_A$). Ο λόγος για το δεύτερο είναι ότι αν γνωρίζουμε (με βεβαιότητα, ή κάτω από βάσιμες

υποθέσεις) ότι το $\Pi_A$ δεν διαθέτει πολυωνυμικό αλγόριθμο, τότε συνάγεται (με (με βεβαιότητα, ή κάτω από τις ίδιες υποθέσεις) ότι και το $\Pi_B$ δεν διαθέτει πολυωνυμικό αλγόριθμο (αλλιώς θα διέθετε και το $\Pi_A$ (αντίφαση).

Παραδείγματα:

- Η επίλυση του Προβλήματος του Βαρκάρη με αναγωγή στο πρόβλημα Reachability (βλ. Ασκήσεις) υπάγεται στην πρώτη περίπτωση.
- Οι αποδείξεις **NP**-πληρότητας υπάγονται στη δεύτερη περίπτωση και παρουσιάζονται πιο αναλυτικά παρακάτω.

**Αποδείξεις NP-πληρότητας με χρήση αναγωγής πολυωνυμικού χρόνου**

**Ορισμός**. Ένα πρόβλημα απόφασης Π λέγεται **NP-δύσκολο (NP-hard)** αν για κάθε πρόβλημα Π' $\in$ **NP** ισχύει Π' $\leq^p$ Π. Αν επιπλέον Π $\in$ **NP** τότε το Π λέγεται **NP-πλήρες (NP-complete)**.

Ισχύει ότι αν ένα πρόβλημα Π είναι **NP**-δύσκολο τότε αν υπήρχε πολυωνυμικός αλγόριθμος $S$ για το πρόβλημα αυτό θα συνεπαγόταν ότι **P**=**NP** (γιατί θα μπορούσαμε να 'συνθέσουμε' τον αλγόριθμο της αναγωγής Π' $\leq^p$ Π με τον υποθετικό αλγόριθμο $S$ και να πάρουμε πολυωνυμικό αλγόριθμο για το Π', όπου Π' οποιοδήποτε πρόβλημα στην κλάση **NP**). Παρ'ότι αυτό είναι ακόμη ανοιχτό πρόβλημα, θεωρείται εξαιρετικά απίθανο να συμβαίνει, οπότε αν γνωρίζουμε ότι ένα πρόβλημα είναι **NP**-δύσκολο συμπεραίνουμε ότι είναι μάλλον αδύνατο να έχει πολυωνυμικό αλγόριθμο. Τα ίδια ισχύουν βέβαια αν ένα πρόβλημα είναι **NP**-πλήρες (αφού **NP**-πλήρες είναι ειδική περίπτωση **NP**-δύσκολου).

Το παρακάτω θεώρημα δίνει τον (συνήθη) τρόπο απόδειξης ότι ένα πρόβλημα είναι **NP**-δύσκολο (ή και **NP**-πλήρες).

**Θεώρημα**. Αν $\Pi_A \leq^p \Pi_B$ και το $\Pi_A$ είναι **NP**-πλήρες τότε το $\Pi_B$ είναι **NP**-δύσκολο. Αν επιπλέον $\Pi_B \in$ **NP** τότε το $\Pi_B$ είναι **NP**-πλήρες.

Η ιδέα της απόδειξης (δεν θα την παρουσιάσουμε αναλυτικά εδώ) είναι ότι η σύνθεση αναγωγών πολυωνυμικού χρόνου είναι αναγωγή πολυωνυμικού χρόνου, επομένως ισχύει ότι:

$$\Pi \leq^p \Pi_A \text{ και } \Pi_A \leq^p \Pi_B \Longrightarrow \Pi \leq^p \Pi_B$$

**Παράδειγμα:** η αναγωγή που δώσαμε παραπάνω Hamilton Cycle $\leq^p$ TSP, επειδή γνωρίζουμε ότι το Hamilton Cycle είναι **NP**-πλήρες οδηγεί στο συμπέρασμα ότι και το πρόβλημα απόφασης TSP είναι **NP**-δύσκολο. Επειδή επιπλέον το πρόβλημα απόφασης TSP ανήκει στην κλάση **NP** (αποδεικνύεται σχετικά εύκολα: μια πιθανή λύση επαληθεύεται σε πολυωνυμικό χρόνο) συμπεραίνουμε ότι το πρόβλημα απόφασης TSP είναι **NP**-πλήρες.

## 11.5 Διαδραστικό υλικό - Σύνδεσμοι

- Οι κλάσεις πολυπλοκότητας που υπάρχουν σήμερα, είναι πλέον τόσο πολλές που χρειάζονται μια γενικευμένη κατηγοριοποίηση. Εδώ μπορείτε να βρείτε έναν "ζωολογικό κήπο"

με τις περισσότερες κλάσεις πολυπλοκότητας που υπάρχουν.

## 11.6 Ασκήσεις

1. *Κάλυψη κόμβων* σε έναν γράφο $G(V, E)$ λέγεται ένα υποσύνολο $V'$ των κόμβων του γράφου τέτοιο ώστε κάθε ακμή του γράφου έχει έναν τουλάχιστον κόμβο της στο σύνολο, δηλαδή $\forall\{u, v\} \in E : u \in V' \vee v \in V'$. *Ανεξάρτητο σύνολο* σε έναν γράφο $G(V, E)$ λέγεται ένα υποσύνολο κόμβων του γράφου $V'$ που δεν έχουν καμμία ακμή μεταξύ τους, δηλαδή $\forall u, v \in V' : \{u, v\} \notin E$.

   (α) Αποδείξτε ότι ένα υποσύνολο $V'$ των κόμβων του γράφου είναι κάλυψη κόμβων αν και μόνο αν το σύνολο $V \setminus V'$ είναι ανεξάρτητο.

   (β) *Κλίκα* σε έναν γράφο $G(V, E)$ λέγεται ένα υποσύνολο κόμβων $V'$ του γράφου που συνδέονται όλοι ανά δύο μεταξύ τους, δηλαδή $\forall u, v \in V' : (u, v) \in E$.

   Περιγράψτε αναγωγή από το πρόβλημα Vertex Cover (δίνεται γράφος και αριθμός $k$, υπάρχει κάλυψη κόμβων μεγέθους το πολύ $k$ στο γράφο;) στο πρόβλημα Clique (δίνεται γράφος και αριθμός $m$, υπάρχει κλίκα μεγέθους τουλάχιστον $m$ στο γράφο;). Τι συμπέρασμα προκύπτει εάν γνωρίζουμε ότι το πρόβλημα Vertex Cover είναι NP-πλήρες; Τι έχετε να πείτε για το πρόβλημα Independent Set (δίνεται γράφος και αριθμός $t$, υπάρχει ανεξάρτητο σύνολο μεγέθους τουλάχιστον $t$ στο γράφο;);

2. Δώστε αναγωγή από το Vertex Cover (βλ. άσκηση 1) στο Set Cover: "Δίνεται ένα πεπερασμένο σύνολο $U$ και ένα σύνολο $S = \{S_1, \ldots, S_m\}$ υποσυνόλων του $U$. Ζητείται το μικρότερο δυνατό υποσύνολο $C$ του $S$, τέτοιο ώστε το $C$ να καλύπτει όλα τα στοιχεία του $U$, δηλαδή $\cup_{S_i \in C} S_i = U$.

3. Γενικεύοντας το Πρόβλημα του Βαρκάρη (βλ. και ασκήσεις γράφων) μας δίνονται $n$ αντικείμενα και χωρητικότητα βάρκας $k$ (εκτός του βαρκάρη). Οι ασυμβατότητες μεταξύ των αντικειμένων δίνονται από μια σχέση $R$: $R(a_i, a_j)$ σημαίνει ότι απαγορεύεται τα $a_i$ και $a_j$ να βρίσκονται στην ίδια όχθη αφύλακτα (επιτρέπεται όμως να είναι στην ίδια όχθη με τον βαρκάρη παρόντα, ή μέσα στη βάρκα πηγαίνοντας από την μια όχθη στην άλλη).

   (α) Έχει πάντοτε λύση το Πρόβλημα του Βαρκάρη;

   (β) Επεκτείνετε την αναγωγή που περιγράψαμε στο μάθημα σε μια αναγωγή του Γενικευμένου Προβλήματος του Βαρκάρη στο Πρόβλημα Προσβασιμότητας σε γράφο (Reachability).

   (γ) Διατυπώστε έναν αλγόριθμο επίλυσης του Γενικευμένου Προβλήματος του Βαρκάρη. Ποια είναι η πολυπλοκότητα του αλγορίθμου σας;

4. Δείξτε ότι η αναγωγή πολυωνυμικού χρόνου είναι μεταβατική σχέση, δηλαδή αν $\Pi_A \leq^p \Pi_B$ και $\Pi_B \leq^p \Pi_\Gamma$, τότε και $\Pi_A \leq^p \Pi_\Gamma$.

5. Ορίζουμε το πρόβλημα της Διαδρομής Ίππου ως εξής: είσοδος είναι οι διαστάσεις ορθογώνιας σκακιέρας $n, m$, καθώς και κάποια απαγορευμένα τετράγωνα που δίνονται σαν ζεύγη $(i, j)$. Θέλουμε να βρούμε αν ένα άλογο (ίππος) που ξεκινά από το τετράγωνο $(x_1, y_1)$

μπορεί να φτάσει στο τετράγωνο $(x_2, y_2)$ χωρίς να πατήσει πάνω σε απαγορευμένα τετράγωνα, και αν ναι με ποιον τρόπο.

*Σημείωση*: η κίνηση του αλόγου είναι 2 τετράγωνα προς μία κατεύθυνση, οριζόντια ή κάθετα, και μετά 1 τετράγωνο αριστερά ή δεξιά, κάθετα στην αρχική κατεύθυνση. Το άλογο επιτρέπεται να περάσει πάνω από απαγορευμένα τετράγωνα, αλλά όχι να σταματήσει σε αυτά.

(α) Έχει λύση το πρόβλημα για τη σκακιέρα $4 \times 5$ με απαγορευμένο τετράγωνο το $(2, 3)$, αρχικό τετράγωνο $(1, 1)$ και τελικό τετράγωνο $(3, 1)$; Αν ναι, ζωγραφίστε τη διαδρομή στο παρακάτω σχήμα. Αν όχι εξηγήστε γιατί.

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | ♞ | | | | |
| 2 | | | × | | |
| 3 | | | | | |
| 4 | | | | | |

(β) Περιγράψτε με σαφήνεια έναν όσο το δυνατόν πιο αποδοτικό αλγόριθμο που να επιλύει το πρόβλημα της Διαδρομής Ίππου στη γενική περίπτωση (δηλ. για οποιαδήποτε σκακιέρα δοθεί). Ποια είναι η πολυπλοκότητα του αλγορίθμου σας σε σχέση με τα $n, m$ και το πλήθος $k$ των απαγορευμένων τετραγώνων;

(γ) Θεωρήστε επιπλέον την παραλλαγή όπου δίνεται επιπλέον ένας ακέραιος $k$ και ζητείται να βρείτε εάν υπάρχει διαδρομή από το αρχικό στο τελικό τετράγωνο με $k$ το πολύ κινήσεις. Βρείτε την απάντηση για την παραπάνω σκακιέρα, για $k = 3$.

Πώς πρέπει να τροποποιήσετε / συμπληρώσετε τον αλγόριθμό σας ώστε να επιλύει αυτή την εκδοχή στη γενική περίπτωση; Αλλάζει η πολυπλοκότητα του αλγορίθμου σας; Εξηγήστε.

6. Δείξτε ότι **P** $\subseteq$ **PSPACE**.

7. (α) Ορίστε τις κλάσεις Reg (Regular), P (Polynomial Time), CF (Context Free), NP (Nondeterministic Polynomial Time), REC (Recursive), RE (Recursively Enumerable).

   (β) Σχεδιάστε τις παραπάνω κλάσεις σε διάγραμμα Hasse, με σύντομη αιτιολόγηση.

   (γ) Ορίστε το πρόβλημα Κύκλου Hamilton. Σε ποια από τις παραπάνω κλάσεις ανήκει και γιατί;

   (δ) Αντιστοιχήστε προβλήματα με κλάσεις πολυπλοκότητας (με σύντομη αιτιολόγηση):

   (i) Προβλήματα:

   - Ελάχιστο Συνδετικό Δένδρο
   - $L_1 = \{w \in \{a, b\}^* \mid w$ περιέχει ίσο αριθμό $a$ και $b\}$
   - Ισοδυναμία αυτομάτων

- $L_2 = \{ww \mid w \in \{a, b, c, d\}^*\}$
- Satisfiability (SAT)
- $L_3 = \{w \in \{0, 1\}^* \mid w$ περιέχει ακριβώς τρία $'0'\}$
- Halting Problem

(ii) Κλάσεις: NP, REC, Context Free, Regular, RE, Context Sensitive, P

# Βιβλιογραφία


[1] Christos Papadimitriou. “Computational Complexity”. Addison Wesley, 1994

[2] Sanjeev Arora, Boaz Barak. “Computational Complexity: A Modern Approach”. Cambridge University Press, 2009

[3] Oded Goldreich. “Computational Complexity: A Conceptual Perspective”, Cambridge University Press, 2008

[4] Martin D. Davis, Ron Sigal, Elaine J. Wayuker, “Computability, Complexity, and Languages”, 2nd edition, Academic Press Professional, Inc. San Diego, CA, USA, 1994.

[5] J.E. Hopcroft and J.D. Ullman. “Introduction to Automata Theory, Languages and Computation”, Addison Wesley Longman, 2007.

[6] M. Sipser. “Introduction to the Theory of Computation”, International Thomson Publishing, 1996.

[7] D. C. Kozen. “Automata and Computability” (Undergraduate Texts in Computer Science), Springer-Verlag New York, Inc. Secaucus, NJ, USA, 1997.

[8]

# Κεφάλαιο 12

# Προσεγγιστικοί Αλγόριθμοι

## 12.1 Προβλήματα Βελτιστοποίησης

Σε ένα πρόβλημα βελτιστοποίησης σε κάθε στιγμιότυπο του προβλήματος αντιστοιχούν κάποιες εφικτές (feasible) -δηλαδή επιτρεπτές- λύσεις, που κάθε μια τους έχει μία τιμή ως προς μια αντικειμενική συνάρτηση. Ζητάμε μια βέλτιστη λύση, δηλαδή μια **εφικτή** λύση που έχει βέλτιστη τιμή.

Τα προβλήματα βελτιστοποίησης είναι ή μεγιστοποίησης (οπότε ζητάμε τη λύση που μεγιστοποιεί την αντικειμενική συνάρτηση ανάμεσα σε όλες τις εφικτές λύσεις) ή ελαχιστοποίησης (οπότε ζητάμε την εφικτή λύση που ελαχιστοποιεί την αντικειμενική συνάρτηση).

*Παράδειγμα* 12.1. Το πρόβλημα Vertex Cover. Θυμηθείτε ότι στο πρόβλημα αυτό, αναζητούμε ένα σύνολο από κόμβους του γράφου με ελάχιστο πληθάριθμο, έτσι ώστε κάθε ακμή του γράφου να καλύπτεται από τουλάχιστον έναν κόμβο. Είναι αντίστοιχο με το να θεωρήσουμε το σύνολο των ακμών του γράφου σαν ένα σύνολο από διαδρόμους σε ένα μουσείο και να προσπαθήσουμε να τοποθετήσουμε φύλακες στις διασταυρώσεις των διαδρόμων (κορυφές του γράφου) κατά τέτοιο τρόπο, ώστε να φρουρούνται όλοι οι διάδρομοι, ενώ συγχρόνως προσπαθούμε να ελαχιστοποιήσουμε τον αριθμό των φυλάκων. Για το πρόβλημα αυτό έχουμε:

- Είναι πρόβλημα ελαχιστοποίησης
- Στιγμιότυπο είναι ένας γράφος
- Κάθε Vertex Cover είναι εφικτή λύση (π.χ. $V(G)$)
- Αντικειμενική Συνάρτηση: Ο πληθάριθμος του Vertex Cover
- Βέλτιστη λύση: Ένα Vertex Cover με ελάχιστο πληθάριθμο

## 12.2 Κλάσεις προσεγγιστικών προβλημάτων

### 12.2.1 Η κλάση PO

Η κλάση PO περιέχει όλα τα προβλήματα βελτιστοποίησης των οποίων το αντίστοιχο πρόβλημα απόφασης είναι στο P. Αυτό σημαίνει ότι περιέχει όλα τα προβλήματα βελτιστοποίησης τα οποία μπορούν να επιλυθούν σε πολυωνυμικό χρόνο.

Ας δούμε όμως τι εννοούμε με τη φράση «αντίστοιχο πρόβλημα απόφασης»: εννοούμε ένα πρόβλημα στο οποίο μπορούμε να αναγάγουμε το πρόβλημα βελτιστοποίησης -σε πολυωνυμικό πάντα χρόνο-, έτσι ώστε λύνοντας το πρόβλημα απόφασης (για κατάλληλα επιλεγμένες εισόδους που θα δούμε παρακάτω) να μπορούμε να βρούμε την τιμή μίας βέλτιστης λύσης στο πρόβλημα βελτιστοποίησης. Ο λόγος που αναφερόμαστε σε αντίστοιχο πρόβλημα απόφασης, είναι ότι ως τώρα η πολυπλοκότητα και τα διάφορα υπολογιστικά μοντέλα που έχουμε, αναφέρονται αποκλειστικά σε προβλήματα απόφασης. Ως εκ τούτου, για να μελετήσουμε τα προβλήματα βελτιστοποίησης με πιο τυπικό τρόπο, προσπαθούμε να τα αντιστοιχίσουμε σε προβλήματα απόφασης.

Στην προκείμενη περίπτωση, τα αντίστοιχα προβλήματα είναι:

**Πρόβλημα Βελτιστοποίησης (έστω ελαχιστοποίησης):** Δοθέντος ενός στιγμιοτύπου μεγέθους $n$, βρες μία εφικτή λύση με ελάχιστη τιμή της αντικειμενικής συνάρτησης.

**Πρόβλημα απόφασης:** Δοθέντος ενός στιγμιοτύπου μεγέθους $n$ και ενός αριθμού $k$, υπάρχει εφικτή λύση με τιμή $\leq k$;

Αν μπορούμε να λύσουμε το πρόβλημα βελτιστοποίησης τότε τετριμμένα μπορούμε να απαντήσουμε και στο πρόβλημα απόφασης.

Ισχύει όμως και το αντίστροφο: αν το πρόβλημα απόφασης επιλύεται σε χρόνο πολυωνυμικό ως προς $n$ και $\log k$ (δηλαδή ως προς το μέγεθος αναπαράστασης του αριθμού $k$), τότε μπορούμε να λύσουμε και το αντίστοιχο πρόβλημα βελτιστοποίησης σε πολυωνυμικό ως προς $n$ χρόνο.

Μία πρώτη ιδέα είναι να τρέξουμε τον αλγόριθμο για το πρόβλημα απόφασης (έστω ) $K$ φορές, για $k = 1, \ldots, K$ (όπου $K$ ένας κατάλληλα επιλεγμένος μεγάλος αριθμός, τέτοιος ώστε να υπάρχει εφικτή λύση με κόστος λιγότερο από ) και να επιστρέψουμε την πρώτη λύση για την οποία ο επιστρέφει yes. Δυστυχώς μία τέτοια προσέγγιση, θα απαιτούσε να τρέξουμε τον αλγόριθμο στη χειρότερη περίπτωση φορές, δηλαδή θα απαιτούσε χρόνο εκθετικό ως προς το μέγεθος αναπαράστασης του .

Αντ' αυτού επιλέγουμε την εξής πιο γρήγορη λύση: ψάχνουμε να βρούμε κάποιο $k$, τέτοιο ώστε να υπάρχει μία εφικτή λύση με κόστος $\leq k$ αλλά όχι με κόστος $\leq k/2$. Το $k$ αυτό, μπορεί να βρεθεί εύκολα σε $\log k$ βήματα (δοκιμάζοντας διαδοχικά τις τιμές $2^0, 2^1, \ldots$ ). Αφού βρούμε αυτό το $k$ εφαρμόζουμε δυαδική αναζήτηση στο διάστημα $[k, k/2]$, ώστε να βρούμε τη μικρότερη τιμή $k$ για την οποία ο επιστρέφει yes. Η δυαδική αναζήτηση κοστίζει επίσης $\log \frac{k}{2}$ χρόνο, οπότε συνολικά, τρέχοντας $O(\log k)$ φορές τον αλγόριθμο (δηλαδή πολυωνυμικό αριθμό φορών ως προς

το μέγεθος της εισόδου) μπορούμε να λύσουμε και το αντίστοιχο πρόβλημα βελτιστοποίησης.

**Παρατηρήσεις:** Προσέξτε ότι με την παραπάνω μέθοδο, μπορούμε να χρησιμοποιήσουμε το πρόβλημα απόφασης για να βρούμε την τιμή μίας βέλτιστης λύσης του προβλήματος βελτιστοποίησης, όχι για να βρούμε μία βέλτιστη λύση αυτή καθεαυτή (πχ αν το πρόβλημα είναι το Vertex Cover, με βάση την παραπάνω μέθοδο θα μπορέσουμε να βρούμε το μέγεθος ενός ελάχιστου Vertex Cover, αλλά όχι από ποιους κόμβους αποτελείται ένα ελάχιστο VC). Συνήθως όμως, όταν μπορούμε να βρούμε την τιμή μίας βέλτιστης λύσης, μπορούμε να βρούμε και μία βέλτιστη λύση καθεαυτή, εξετάζοντας πιθανά στοιχεία της λύσης ένα-ένα (’σκηση: σκεφτείτε πώς μπορεί να γίνει αυτό για το Vertex Cover).

## 12.3 Η κλάση NPO και οι προσεγγιστικοί αλγόριθμοι

Προβλήματα στην PO μπορούν να επιλυθούν σε πολυωνυμικό χρόνο. Τώρα θα συζητήσουμε για προβλήματα βελτιστοποίησης στο NP, δηλαδή τέτοια που το αντίστοιχο πρόβλημα απόφασης είναι στο NP και μάλιστα είναι NP-complete. Εκτός αν P=NP δεν μπορούμε να υπολογίσουμε σε πολυωνυμικό χρόνο τη βέλτιστη τιμή ενός προβλήματος βελτιστοποίησης που αντιστοιχεί σε ένα NP-hard πρόβλημα απόφασης, άρα

- είτε επιλύουμε **σε πολυωνυμικό χρόνο** το πρόβλημα **ακριβώς**, **όχι** όμως **για όλα τα στιγμιότυπα**,
- είτε επιλύουμε το πρόβλημα **ακριβώς**, **για όλα τα στιγμιότυπα**, με χρήση κάποιου **εκθετικού αλγορίθμου** με χαμηλή exponentiality (βάση της δύναμης) όμως (π.χ. με πολυπλοκότητα $(1.01^n)$),
- είτε βρίσκουμε **προσεγγιστικούς** αλγορίθμους **πολυωνυμικού χρόνου**, για **όλα τα στιγμιότυπα**.

Στο υπόλοιπο μέρος του κεφαλαίου θα παρουσιάσουμε κάποιες βασικές έννοιες των προσεγγιστικών αλγορίθμων, καθώς και συγκεκριμένους αλγόριθμους, για μερικά πολύ γνωστά NP-hard προβλήματα.

Συμβολισμοί[1]:

- $\Pi$: το πρόβλημα
- $I$: στιγμιότυπο
- $SOL_A(\Pi, I)$ : η τιμή της λύσης για το στιγμιότυπο $I$ του προβλήματος $\Pi$ που επιστρέφει ο αλγόριθμος $A$
- $OPT(\Pi, I)$: η βέλτιστη τιμή λύσης του προβλήματος $\Pi$ για το στιγμιότυπο $I$

[1] συνήθως παραλείπουμε τα $\Pi$, $I$ και $A$ από τους παρακάτω συμβολισμούς

Με βάση τα παραπάνω μπορούμε να ορίσουμε το λόγο προσέγγισης (approximation ratio ή approximation factor) ενός προσεγγιστικού αλγορίθμου ως εξής:

*Ορισμός* 12.2. Ένας αλγόριθμος Α για ένα πρόβλημα ελαχιστοποίησης Π, έχει λόγο προσέγγισης $(n)$, αν για κάθε στιγμιότυπο του προβλήματος Ι (με $|| = n$) ισχύει:

$$\frac{SOL_A(I)}{OPT(I)} \leq \rho(n)$$

Ο παραπάνω ορισμός μπορεί επίσης να διατυπωθεί ως εξής:

$$OPT(I) \leq SOL_A(I) \leq \rho(n) \cdot OPT(I)$$

Η παραπάνω μορφή καθιστά πιο σαφές το νόημα του παράγοντα προσέγγισης σε έναν προσεγγιστικό αλγόριθμο Α: δοθέντος ενός στιγμιοτύπου του προβλήματος μεγέθους $n$, ο αλγόριθμος επιστρέφει μία λύση $SOL_A(I)$ η οποία είναι χειρότερη (μεγαλύτερη) από τη βέλτιστη (ελάχιστη) λύση, αλλά το πολύ $\rho(n)$ φορές χειρότερη. Με άλλα λόγια έχουμε ένα είδος «εγγύησης», ότι για οποιοδήποτε input ο αλγόριθμος αυτός δεν θα μας δώσει μία λύση που θα απέχει πολύ από τη βέλτιστη.

Αντίστοιχα μπορούμε να διατυπώσουμε και τον ορισμό για προβλήματα μεγιστοποίησης:

*Ορισμός* 12.3. Ένας αλγόριθμος Α για ένα πρόβλημα μεγιστοποίησης Π, έχει λόγο προσέγγισης $\rho(n)$, αν για κάθε στιγμιότυπο του προβλήματος Ι (με $|| = n$) ισχύει:

$$\frac{SOL_A(I)}{OPT(I)} \geq \rho(n)$$

Τον αλγόριθμο αυτό τον λέμε $\rho(n)$ - προσεγγιστικό αλγόριθμο.

Το επόμενο φυσικό ερώτημα είναι, τι ακριβώς κρύβεται πίσω από το $\rho(n)$; Έχουμε λοιπόν τις εξής περιπτώσεις:

- $\rho(n) = \cup_i O(n^i)$, δηλαδή ο παράγοντας προσέγγισης είναι μία πολυωνυμική συνάρτηση του μεγέθους της εισόδου. Τα προβλήματα που έχουν τέτοιο αλγόριθμο προσέγγισης ανήκουν στην κλάση poly-APX.
- $\rho(n) = O(\log n)$, δηλαδή ο παράγοντας προσέγγισης είναι μία λογαριθμική συνάρτηση του μεγέθους της εισόδου. Τα προβλήματα που έχουν τέτοιο αλγόριθμο προσέγγισης ανήκουν στην κλάση log-APX.
- $\rho(n) = \rho$, οπότε ο παράγοντας προσέγγισης είναι σταθερός (ανεξάρτητος του μεγέθους της εισόδου). Τα προβλήματα που έχουν τέτοιο αλγόριθμο προσέγγισης ανήκουν στην κλάση APX.

Από τις τρεις παραπάνω περιπτώσεις, η APX είναι η πιο επιθυμητή, αφού μας εξασφαλίζει ότι για οποιαδήποτε (οσοδήποτε μεγάλη) είσοδο, μπορούμε να προσεγγίσουμε μία λύση που διαφέρει

κατά ένα συγκεκριμένο παράγοντα από τη βέλτιστη. Αντίστοιχα η log-APX περίπτωση είναι καλύτερη από την poly-APX, αφού -για δεδομένο μέγεθος εισόδου- η λογαριθμική συνάρτηση μας δίνει καλύτερο παράγοντα (μικρότερο) προσέγγισης από μία πολυωνυμική.

Προσέξτε τώρα, ότι η κλάση APX περιέχει προβλήματα βελτιστοποίησης που επιδέχονται σταθερό παράγοντα προσέγγισης. Έτσι, λέγοντας για παράδειγμα ότι το Vertex Cover επιδέχεται έναν 2-προσεγγιστικό αλγόριθμο, εννοούμε ότι επιδέχεται έναν αλγόριθμο που επιστρέφει ένα VC με το πολύ το διπλάσιο αριθμό κόμβων από ένα ελάχιστο VC. Ιδανικά θα θέλαμε για κάθε παράγοντα προσέγγισης $\rho$ να μπορούμε να βρούμε έναν $\rho$-προσεγγιστικό αλγόριθμο. Με άλλα λόγια θα ήταν πολύ χρήσιμο αν μπορούσαμε να κατασκευάσουμε μία οικογένεια προσεγγιστικών αλγορίθμων, έναν για κάθε παράγοντα προσέγγισης $\rho$, για το πρόβλημα.

Πιο τυπικά θα ορίσουμε την οικογένεια αυτή των αλγορίθμων σαν ένα προσεγγιστικό σχήμα (approximation scheme) ως εξής:

*Ορισμός* 12.4. Ο $A$ είναι ένα προσεγγιστικό σχήμα (approximation scheme) για το πρόβλημα Π, αν για είσοδο $(I, \varepsilon)$, όπου $I$ είναι ένα στιγμιότυπο και $\varepsilon > 0$ η παράμετρος λάθους έχουμε:

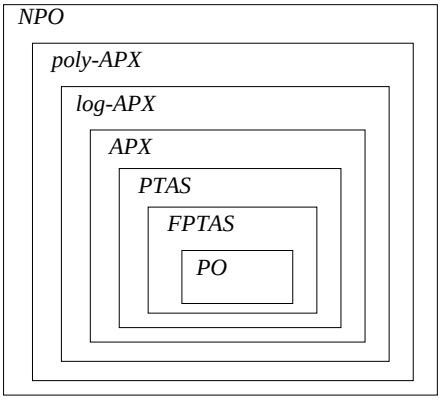
- $OPT(I) \leq SOL_A(I, \varepsilon) \leq (1 + \varepsilon) \cdot OPT(I)$, για πρόβλημα ελαχιστοποίησης
- $OPT(I) \geq SOL_A(I, \varepsilon) \geq (1 - \varepsilon) \cdot OPT(I)$, για πρόβλημα μεγιστοποίησης

Επίσης λέμε ότι ο $A$ είναι PTAS (Polynomial Time Approximation Scheme) αν για κάθε $\varepsilon > 0$ χρειάζεται πολυωνυμικό χρόνο ως προς το μέγεθος του $I$.
Ο $A$ είναι FPTAS (Fully PTAS) αν για κάθε $\varepsilon > 0$ χρειάζεται πολυωνυμικό χρόνο ως προς το μέγεθος του $I$ και ως προς την τιμή $1/\varepsilon$.

Τα προσεγγιστικά σχήματα PTAS και FPTAS λοιπόν είναι αλγόριθμοι που δέχονται ως είσοδο, πέρα από το στιγμιότυπο του προβλήματος, και έναν φυσικό αριθμό $\varepsilon$ που είναι η παράμετρος λάθους και μπορείτε να δείτε πώς σχετίζεται με τον παράγοντα προσέγγισης $\rho$ σε κάθε είδος προβλήματος βελτιστοποίησης. Αν ο αλγόριθμος αυτός τρέχει σε πολυωνυμικό χρόνο ως προς το $||$ έχουμε την περίπτωση του PTAS. Παρόλα αυτά, αυτό μπορεί να μην είναι αρκετό: για παράδειγμα μπορεί να έχουμε βρει έναν PTAS που επιλύει το πρόβλημα με πολύ μικρό σφάλμα $\varepsilon$, αλλά ο χρόνος εκτέλεσής του αυξάνει εκθετικά όσο μειώνεται το $\varepsilon$ (δηλαδή όσο αυξάνεται το $1/\varepsilon$). Για αυτό το λόγο θεωρούμε την οικογένεια FPTAS όπου προσθέτουμε την απαίτηση ο αλγόριθμος να είναι πολυωνυμικός και ως προς $1/\varepsilon$. Έτσι, ναι μεν χρειαζόμαστε παραπάνω χρόνο για περισσότερη ακρίβεια προσέγγισης, αλλά μόνο πολυωνυμικά περισσότερο.

Από την παραπάνω συζήτηση πρέπει να έχει γίνει κατανοητό ότι τα FPTAS συνιστούν (μάλλον) την καλύτερη κατηγορία προσεγγιστικών αλγορίθμων. Δυστυχώς αρκετά προβλήματα δεν επιδέχονται FPTAS (όπως για παράδειγμα το Traveling Salesperson Problem), ενώ για αρκέτα άλλα δε γνωρίζουμε αν επιδέχονται ή όχι (όπως για παράδειγμα το Vertex Cover).

**Παρατηρήσεις:** Προσέξτε ότι στα FPTAS πρέπει ο αλγόριθμος να είναι πολυωνυμικός ως προς το $1/\varepsilon$. Αυτό είναι πολύ σημαντικό, διότι αν ο αλγόριθμος ήταν πολυωνυμικός ως προς το μήκος της αναπαράστασης του $1/\varepsilon$, θα μπορούσαμε να επιλέξουμε κατάλληλα το $\varepsilon$ ώστε η προσεγγιστική λύση να είναι μεταξύ του OPT και του OPT+1. Αυτό όμως πρακτικά θα σήμαινε (για




Σχήμα 12.1: Κλάσεις προβλημάτων βελτιστοποίησης.

προβλήματα με output ακέραιες τιμές) ότι μπορούμε να λύσουμε ακριβώς το πρόβλημα (δηλαδή ότι το πρόβλημα είναι στην PO). Περισσότερα για αυτό το θέμα στην ενότητα που μελετά το πρόβλημα Discrete Knapsack.

Οι σχέσεις αυτές μεταξύ των διαφόρων κατηγοριών προσεγγιστικών αλγορίθμων απεικονίζονται υπό τη μορφή ιεραρχίας στο Σχήμα 12.1.

Τέλος, μερικά χαρακτηριστικά προβλήματα για κάθε κλάση είναι τα εξής:

- NPO: Traveling Salesman Problem
- poly-APX: Clique
- log-APX: Set Cover
- APX: Vertex Cover
- PTAS: Bin Packing
- FPTAS: Discrete Knapsack
- PO: Matching

Τα παραπάνω προβλήματα έχουν τοποθετηθεί στη μικρότερη από τις κλάσεις που είναι μέχρι στιγμής γνωστό ότι ανήκουν. Για αρκετά από αυτά έχουμε ενδείξεις (ή και αποδείξεις με την προϋπόθεση φυσικά ότι $P \neq NP$) ότι δεν είναι πιο κάτω στην ιεραρχία.

## 12.4 Αντιπροσωπευτικοί προσεγγιστικοί αλγόριθμοι

Προχωράμε τώρα στη μελέτη τριών γνωστών NP-hard προβλημάτων βελτιστοποίησης και εξετάζουμε σε ποιο βαθμό επιδέχονται προσεγγιστικό αλγόριθμο.

### 12.4.1 Vertex Cover Problem

Όπως αναφέραμε και πιο πριν το πρόβλημα του Vertex Cover επιδέχεται έναν 2-προσεγγιστικό αλγόριθμο.

Πριν προχωρήσουμε στην παρουσίαση του προσεγγιστικού αλγορίθμου, θα παρουσιάσουμε μία γενική στρατηγική για τη σχεδίαση προσεγγιστικών αλγορίθμων. Η στρατηγική αυτή έχει δύο βασικά βήματα:

1. Βρες ένα κάτω φράγμα $L$ για τη βέλτιστη λύση ($L \leq OPT$)
2. Σχεδίασε έναν αλγόριθμο που εκμεταλλεύεται το φράγμα αυτό, επιστρέφοντας μία λύση κόστους $SOL \leq \rho \cdot L \leq \rho \cdot OPT$

Στην περίπτωση του VC το κάτω φράγμα αυτό θα μας προκύψει από ένα οποιοδήποτε maximal matching. Ας θυμηθούμε λοιπόν κάποια πράγματα σχετικά με matchings:

*Ορισμός* 12.5. Δίνεται ένας γράφος $G = (V, E)$. Ένα ταίριασμα (matching) είναι ένα υποσύνολο των ακμών του $M \subseteq E$ τέτοιο ώστε ποτέ δύο ακμές να μην έχουν κοινό άκρο.

- Maximal matching είναι ένα matching στο οποίο δεν μπορούμε να προσθέσουμε καμία επιπλέον ακμή. Δηλαδή αν προσθέσουμε οποιαδήποτε άλλη ακμή, παύει να είναι matching. Ένα τέτοιο matching είναι πολύ εύκολο να βρεθεί σε πολυωνυμικό χρόνο με έναν greedy αλγόριθμο ο οποίος προσθέτει ακμές στο M, ώσπου να μην μπορούν να προστεθούν άλλες.
- Maximum matching είναι ένα mathcing το οποίο έχει μέγιστη πληθικότητα, δηλαδή περιέχει τις περισσότερες δυνατές ακμές. Ένα maximum matching είναι και maximal, ενώ το αντίστροφο δεν ισχύει αναγκαστικά. Ένα maximum matching μπορεί επίσης να βρεθεί σε πολυωνυμικό χρόνο, μέσω αναγωγής στο πρόβλημα max-flow, που μελετήσαμε σε προηγούμενο κεφάλαιο.

Ο προσεγγιστικός αλγόριθμος για το VC είναι πολύ απλός και έχει ως εξής:

**Αλγόριθμος 12.1** 2-προσεγγιστικός αλγόριθμος για VC

Βρες ένα maximal matching $M$ και επίστρεψε το σύνολο $V'$ των κορυφών όλων των ακμών του.

*Θεώρημα* 12.6. *Το σύνολο $V'$ καλύπτει όλες τις ακμές του γράφου, είναι δηλαδή όντως VC.*

*Απόδειξη.* Προφανώς οι κορυφές του $V'$ καλύπτουν όλες τις ακμές του matching . ’ρα χρειάζεται να εξετάσουμε μόνο τις ακμές εκτός του . Έστω λοιπόν ότι υπάρχει κάποια ακμή (που δεν ανήκει στο matching $M$) που δεν καλύπτεται από κανέναν κόμβο του $V'$. Αυτό όμως σημαίνει ότι η ακμή αυτή μπορεί να προστεθεί στο $M$, μιας και δε θα έχει κοινό άκρο με καμία από τις ακμές του matching, δημιουργώντας ένα νέο matching με μεγαλύτερο μέγεθος. Αυτό μας οδηγεί σε άτοπο, αφού το  είναι maximal matching. □

Είδαμε λοιπόν ότι όντως ο παραπάνω αλγόριθμος μας επιστρέφει ένα VC. Ας επαληθεύσουμε τώρα ότι το VC αυτό έχει μέγεθος το πολύ 2 φορές μεγαλύτερο από το βέλτιστο (ελάχιστο).

*Θεώρημα* 12.7. *Ο παραπάνω αλγόριθμος είναι 2-προσεγγιστικός.*

*Απόδειξη.* Όπως αναφέραμε και πιο πριν, σκοπός μας είναι να χρησιμοποιήσουμε το μέγεθος του maximal matching $|M|$ ως ένα κάτω φράγμα για το OPT του VC. Όντως ισχύει $|M| \leq OPT$, αφού κάθε VC πρέπει να διαλέξει τουλάχιστον ένα άκρο από κάθε ακμή του $M$: αν για κάποια ακμή δε διαλέξει κανένα άκρο της, τότε αυτή η ακμή δε θα καλύπτεται[2]. Παρατηρήστε τώρα ότι $|V'| = 2 \cdot |M|$, οπότε εύκολα προκύπτει:

$$SOL = |V'| = 2 \cdot |M| \leq 2 \cdot OPT \Rightarrow SOL \leq 2 \cdot OPT$$

□

Τέλος είναι εύκολο να δούμε ότι ο παραπάνω αλγόριθμος εκτελεί μία φορά τον αλγόριθμο για το maximal matching και είναι επομένως πολυωνυμικού χρόνου.

**Γίνεται καλύτερα;**

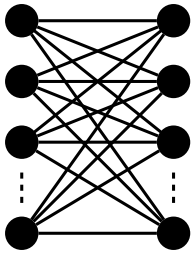
Αφού σχεδιάσουμε έναν οποιοδήποτε αλγόριθμο και μελετήσουμε την ορθότητα και την αποδοτικότητά του (και στην περίπτωση των προσεγγιστικών αλγορίθμων και το λόγο προσέγγισής τους), το επόμενο βήμα είναι να αναρωτηθούμε κατά πόσον μπορούμε να πετύχουμε κάτι καλύτερο. Στην περίπτωση των προσεγγιστικών αλγορίθμων αυτό μεταφράζεται στο «κατά πόσον μπορούμε να πετύχουμε καλύτερη προσέγγιση», που με τη σειρά του αντιστοιχεί σε τρία διακριτά ερωτήματα:

1. Είναι όντως ο λόγος προσέγγισης του αλγορίθμου τόσο μεγάλος, ή μήπως η ανάλυσή μας (το Θεώρημα 12.7 δηλαδή) δεν είναι τόσο ακριβής (tight); Με άλλα λόγια, υπάρχει είσοδος για την οποία ο αλγόριθμος επιστρέφει μία λύση που είναι 2 φορές χειρότερη από τη βέλτιστη; (κρατάμε το κάτω φράγμα του OPT και τον αλγόριθμο και επανεξετάζουμε την ανάλυσή του)

2. Είναι καλός ο αλγόριθμός μας, ή μήπως, χρησιμοποιώντας το ίδιο κάτω φράγμα για το OPT, θα μπορούσαμε να σχεδιάσουμε έναν πιο έξυπνο αλγόριθμο. Στο παράδειγμά μας, μήπως, ας πούμε, θα μπορούσαμε να μην επιλέξουμε όλες τις κορυφές του matching αλλά κάποιες μόνο από αυτές; (κρατάμε το κάτω φράγμα του OPT και επανεξετάζουμε τον αλγόριθμο)

3. Μήπως τελικά το κάτω φράγμα που χρησιμοποιήσαμε δεν είναι και τόσο καλό και πρέπει να επανεξετάσουμε την όλη προσέγγισή μας στο πρόβλημα;

[2] αυτό ισχύει για κάθε matching, άρα και για το maximal, και για κάθε VC άρα και για το ελάχιστο



Σχήμα 12.2: Ένας πλήρης διμερής γράφος της οικογένειας $K_{n,n}$

Από τα δύο παραπάνω ερωτήματα τα δύο πρώτα είναι συνήθως εύκολα να απαντηθούν, ενώ το τρίτο όχι και τόσο.

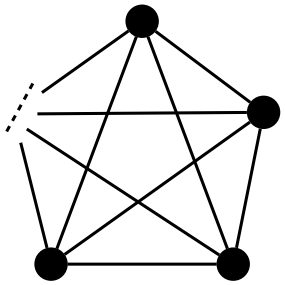
Ας εστιάσουμε όμως στο πρόβλημα του VC και ας προσπαθήσουμε να απαντήσουμε στο πρώτο ερώτημα: είναι δυνατόν με μία καλύτερη ανάλυση να έχουμε καλύτερο λόγο προσέγγισης;

Η απάντηση είναι όχι και αυτό το δείχνουμε ως εξής: βρίσκουμε μία άπειρη οικογένεια στιγμιοτύπων του προβλήματος (εδώ μία οικογένεια γράφων) για την οποία η λύση που επιστρέφει ο αλγόριθμος είναι όντως 2 φορές χειρότερη από τη βέλτιστη. Μία τέτοια οικογένεια παραδειγμάτων ονομάζεται tight και η σημασία τους για την ανάλυση των προσεγγιστικών αλγορίθμων είναι τεράστια. Είναι δε αναγκαίο να είναι άπειρη η οικογένεια (ή τουλάχιστον να είναι δύσκολο να εντοπίσουμε κάθε μέλος της), αλλιώς ο προσεγγιστικός αλγόριθμος θα μπορούσε ανάλογα με την είσοδο να επιλέγει τη συμπεριφορά του.

*Παράδειγμα* 12.8. Θεωρήστε την άπειρη οικογένεια από στιγμιότυπα της μορφής του σχήματος (12.2), δηλαδή από πλήρεις διμερείς γράφους $_{n,n}$. Μιας και οποιοδήποτε maximal matching στο γράφο αυτό έχει μέγεθος $n$, ο αλγόριθμος θα επιστρέψει πάντα ένα VC μεγέθους $2n$. Όμως το βέλτιστο VC έχει μέγεθος $n$ (επιλέγοντας για παράδειγμα όλες τις κορυφές του ενός μέρους του γράφου) και έτσι η λύση που επιστρέφει ο αλγόριθμος είναι όντως 2 φορές χειρότερη από τη βέλτιστη.

Ας προχωρήσουμε τώρα στο δεύτερο ερώτημα: μήπως είναι υπερβολικό να βάλουμε τον αλγόριθμό μας να επιστρέψει όλους τους κόμβους των ακμών ενός maximal matching; Και πάλι η απάντηση είναι όχι, όπως προκύπτει συνήθως με χρήση ενός ακόμα κατάλληλα επιλεγμένου παραδείγματος.

*Παράδειγμα* 12.9. Θεωρήστε την άπειρη οικογένεια $K_{2n+1}$ πλήρων γράφων με περιττό αριθμό κόμβων, όπως αυτός του Σχήματος (12.3). Στην περίπτωση αυτή ένα maximal matching έχει μέγεθος $n$ και έτσι το μέγεθος του επιστρεφόμενου VC θα είναι $2n$. Όμως είναι εύκολο να επαληθεύσουμε ότι και το ελάχιστο VC έχει μέγεθος $2n$. Έτσι ο αλγόριθμός μας θα επιστρέψει μία βέλτιστη λύση. Το παράδειγμα αυτό καθιστά σαφές ότι δεν μπορούμε να βελτιώσουμε τον αλγόριθμο κάνοντας μία πιο «έξυπνη» επιλογή κορυφών (για παράδειγμα μόνο τις μισές κορυφές του ), αφού υπάρχουν περιπτώσεις εισόδων (όπως ο $_{2n+1}$) όπου χρειάζεται να πάρουμε όλες τις κορυφές. Με άλλα λόγια, εφόσον ο αλγόριθμος αυτός για κάποιες εισόδους επιστρέφει το ελάχιστο VC, αν προσπαθούσαμε να ελαττώσουμε τον αριθμό των επιστρεφόμενων κορυφών, τότε για τις συγκεκριμένες αυτές εισόδους η επιστρεφόμενη λύση δε θα ήταν καν VC. Αν π.χ.



Σχήμα 12.3: Ένας πλήρης γράφος της οικογένειας $K_{2n+1}$

προσπαθούσαμε να βρούμε αλγόριθμο που να επιστρέφει λύση κόστους $SOL \leq 3/2|M|$ αυτό δε θα ήταν δυνατό.

Το τελευταίο ερώτημα, αν και πολύ σημαντικό, συνήθως είναι δύσκολο να απαντηθεί. Για παράδειγμα για το VC, ξέρουμε ότι υπάρχει αλγόριθμος με λόγο προσέγγισης $2 - \Theta(\frac{1}{\sqrt{\log n}})$ (χάρη σε μία πρόσφατη εργασία του Καρακώστα (George Karakostas) που βασίζεται σε τεχνικές γραμμικού προγραμματισμού), αλλά δεν ξέρουμε αν το πρόβλημα επιδέχεται προσεγγιστικό αλγόριθμο με σταθερό λόγο προσέγγισης $< 2$ ή PTAS.

**Παρατηρήσεις:** Παρόλο που τα προβλήματα VC, Clique και Independent Set μπορούν να αναχθούν πολυωνυμικά το ένα στο άλλο, αυτό δε σημαίνει ότι ένας πολυωνυμικός προσεγγιστικός αλγόριθμος για το VC μας δίνει έναν αντίστοιχο αλγόριθμο για κάποιο από τα άλλα δύο (θυμηθείτε για παράδειγμα ότι το Clique είναι στην κλάση poly-APX).

### 12.4.2 Discrete Knapsack Problem

Στο κεφάλαιο του Δυναμικού Προγραμματισμού είχαμε αναφερθεί στο πρόβλημα Discrete Knapsack, στο οποίο πρακτικά θέλουμε να επιλέξουμε από ένα σύνολο από αντικείμενα αυτά που χωράνε σε ένα σακίδιο πεπερασμένης χωρητικότητας, έτσι ώστε να μεγιστοποιήσουμε τη συνολική αξία του περιεχομένου του σακιδίου (κάθε αντικείμενο μπορεί να έχει διαφορετική αξία). Είχαμε επίσης αναφέρει ότι το πρόβλημα αυτό επιδέχεται έναν ψευδοπολυωνυμικό αλγόριθμο και είχαμε πει ότι τέτοια προβλήματα συνήθως επιδέχονται και καλό προσεγγιστικό αλγόριθμο. Στο μέρος αυτό θα αναφερθούμε με περισσότερη λεπτομέρεια στη σχέση αυτή των προσεγγιστικών αλγορίθμων με τους ψευδοπολυωνυμικούς, χρησιμοποιώντας ως αντιπροσωπευτικό πρόβλημα το Discrete Knapsack.

#### Ψευδοπολυωνυμικοί αλγόριθμοι

Για να καταλάβουμε καλύτερα τι είναι ένας ψευδοπολυωνυμικός αλγόριθμος θα χρειαστεί να κάνουμε μία διάκριση ανάμεσα στα συστατικά στοιχεία ενός στιγμιοτύπου εισόδου σε έναν αλγόριθμο. Η είσοδος μπορεί να αποτελείται από αντικείμενα (όπως για παράδειγμα σύνολα, γράφους) πλήθους $n$ και από αριθμούς, π.χ. $m$.

Στην πρώτη περίπτωση είναι σαφές τι εννοούμε «μέγεθος εισόδου $n$»: τον πληθάριθμο του συνόλου, ή το μέγεθος του συνόλου κορυφών ή ακμών του γράφου, αντίστοιχα, που φυσικά είναι πολυωνυμικά συσχετισμένο με το μήκος της αναπαράστασης του συνόλου.

Τι γίνεται όμως όταν η επιπλέον είσοδος περιέχει και αριθμούς; Σε αυτή την περίπτωση θεωρούμε ως μέγεθος της εισόδου το μέγεθος της δυαδικής αναπαράστασης των αριθμών. Για παράδειγμα, αν δοθεί ως είσοδος ένας αριθμός $m$, τότε λέμε ότι ο αλγόριθμος είναι πολυωνυμικός αν χρειάζεται χρόνο $O(p(|m|)) = O(p(\log m))$, όπου $p$ ένα πολυώνυμο. Αυτό σημαίνει ότι αν γράψουμε ένα πρόγραμμα το οποίο διαβάζει (δέχεται ως είσοδο) έναν αριθμό $m$ και τρέχει ένα for loop (με σώμα πολυωνυμικού χρόνου ως προς $n$) $m$ φορές, δεν μπορούμε να πούμε ότι ο χρόνος εκτέλεσής του είναι πολυωνυμικός: λέμε ότι είναι ψευδοπολυωνυμικός, δηλαδή τρέχει σε χρόνο $p(n \cdot m)$. Αν εκφράσουμε το χρόνο εκτέλεσης ως συνάρτηση του μεγέθους της εισόδου **σε unary αναπαράσταση του** $m$ (ο αριθμός 42 για παράδειγμα αναπαρίσταται με 43 άσσους[3]) τότε λέμε ότι ένας ψευδοπολυωνυμικός αλγόριθμος είναι πολυωνυμικός ως προς το μέγεθος της unary αναπαράστασης του $m$ (που είναι το ίδιο με πολυωνυμικός ως προς την τιμή του $m$). Ένας ψευδοπολυωνυμικός αλγόριθμος έχει ικανοποιητική απόδοση για «μικρά» $m$. Συγκεκριμένα, για στιγμιότυπα όπου $m = O(p(n))$, ο αλγόριθμος είναι πραγματικά πολυωνυμικός ως προς το μήκος της εισόδου.

Ο λόγος που για πολλά προβλήματα Δυναμικού Προγραμματισμού έχουμε ψευδοπολυωνυμικούς αλγορίθμους είναι ότι η επίλυσή τους βασίζεται συνήθως στη μέθοδο του πίνακα (που γεμίζει με for loops) του οποίου το μέγεθος συχνά καθορίζεται από την τιμή των αριθμών που δίνονται στην είσοδο.

### Strong NP-hardness

Είδαμε στα προηγούμενα κεφάλαια πως στη Θεωρητική Πληροφορική θεωρούμε εν γένει εύκολα τα προβλήματα που λύνονται σε πολυωνυμικό χρόνο και δύσκολα τα NP-hard προβλήματα.

Οι ψευδοπολυωνυμικοί αλγόριθμοι είναι πρακτικά αποδοτικοί για μικρές τιμές των εισόδων, αλλά γενικά συνιστούν μία σαφώς ασθενέστερη (ως προς την αποδοτικότητα) κλάση αλγορίθμων σε σχέση με τους πολυωνυμικούς. Σε πλήρη αντιστοιχία λοιπόν με αυτή την ασθενέστερη έννοια αποδοτικότητας, θα ορίσουμε και μία πιο ισχυρή έννοια «δυσκολίας» (intractability), την strong NP-hardness.

*Ορισμός* 12.10. Ένα πρόβλημα καλείται strongly NP-hard αν κάθε πρόβλημα του NP μπορεί να αναχθεί σε αυτό σε πολυωνυμικό χρόνο, με τους αριθμούς στην είσοδό του να είναι γραμμένα σε unary αναπαράσταση.

Διαισθητικά ο παραπάνω ορισμός μας λέει ότι τα strongly NP-hard προβλήματα παραμένουν δύσκολα να λυθούν ακόμα και αν αφήσουμε τον αλγόριθμο να εκτελεστεί για πολυωνυμικό ως προς $m$ χρόνο, δηλαδή αρκετά (εκθετικά) περισσότερο από πολυωνυμικό ως προς $\log m$ χρόνο.

Παρατηρήστε τώρα ότι αν μπορούσαμε να λύσουμε κάποιο strongly NP-hard πρόβλημα σε ψευδοπολυωνυμικό χρόνο, τότε θα μπορούσαμε να λύσουμε κάθε πρόβλημα του NP σε πολυωνυμικό χρόνο. Με άλλα λόγια, εκτός και αν P=NP, κανένα strongly NP-hard πρόβλημα δεν επιδέ-

[3] λόγω της σύμβασης ότι 1 σε unary είναι το 0 σε δεκαδικό

χεται ψευδοπολυωνυμικό αλγόριθμο.

Πριν προχωρήσουμε στη μελέτη του προσεγγιστικού αλγορίθμου για το Discrete Knapsack θα διατυπώσουμε ένα θεώρημα που καταδεικνύει τη σχέση μεταξύ των FPTAS και των ψευδοπολυωνυμικών αλγορίθμων.

*Θεώρημα* 12.11. *Για ένα πρόβλημα ελαχιστοποίησης* Π*, αν για κάθε στιγμιότυπο εισόδου* $I$*,*

- $OPT < p(|I_u|)$ *για κάποιο πολυώνυμο* $p$*, όπου* $_u$ *είναι η unary αναπαράσταση του* $I$ *και*
- *η αντικειμενική συνάρτηση παίρνει ακέραιες τιμές*

*τότε, αν το* Π *επιδέχεται FPTAS, τότε το* Π *επιδέχεται και ψευδοπολυωνυμικό αλγόριθμο.*

*Πόρισμα* 12.1. Ένα strongly NP-hard πρόβλημα βελτιστοποίησης με τις παραπάνω ιδιότητες δεν επιδέχεται FPTAS, εκτός αν P=NP.

Το πόρισμα προκύπτει άμεσα από το Θεώρημα 12.11 και από την παραπάνω παρατήρηση ότι τα strongly NP-hard προβλήματα δεν επιδέχονται ψευδοπολυωνυμικό αλγόριθμο.

Ας δούμε τώρα τι συμβαίνει με το πρόβλημα Discrete Knapsack. Για τους τυπικούς ορισμούς μπορείτε να ανατρέξετε στο κεφάλαιο του Δυναμικού Προγραμματισμού. Εδώ θα παρουσιάσουμε έναν άλλο αλγόριθμο Δυναμικού Προγραμματισμού που χρησιμοποιεί τη μέθοδο του πίνακα, αλλά διαφέρει από τον αλγόριθμο που παρουσιάσαμε στο προηγούμενο κεφάλαιο στο εξής: αντί να διατηρούμε έναν πίνακα $: n \times W$ όπου το στοιχείο $[i, w]$ περιέχει το υποσύνολο του του $\{x_1, \ldots, x_i\}$ βάρους $w$ με τη μέγιστη αξία, διατηρούμε τον πίνακα $W : n \times n \cdot P$, όπου $P = \max_i p_i$ και η θέση $W[i, p]$ περιέχει το υποσύνολο του $\{x_1, \ldots, x_i\}$ αξίας $p$ με το ελάχιστο βάρος[4].
Ο νέος αλγόριθμος είναι ο 12.2. Παρατηρούμε ότι ο χρόνος εκτέλεσής του είναι $(n^2P)$, όπου $P$ είναι ο μεγαλύτερος από τους αριθμούς που δέχεται ο αλγόριθμος ως είσοδο, δηλαδή ο αλγόριθμος είναι ψευδοπολυωνυμικός.

[4] προφανώς η αξία μίας οποιαδήποτε λύσης δεν μπορεί να υπερβαίνει το $n \cdot P$

**Αλγόριθμος 12.2** Δυναμικός Προγραμματισμός για Discrete Knapsack

**function** DKnap($n$:**integer**; profit: **array**; weight: **array**) :**integer**;

**var** $P$: **integer**; *(*$P = \max_i\{$profits[i]$\}$*)*
$W$: **array**$[1..n][1..n * P]$;
$i, j$: **integer**;

**begin**
**for** $i := 1$ **to** $P$ **do**
**if** $i =$ profit$[i]$ **then** $W[1, i] =$ weight$[i]$;
**else** $W[1, i] := \infty$;

**for** $i := 1$ **to** $n$ **do**
**for** $j := 1$ **to** $n * P$ **do**
$W[i + 1, j] := \min\{W[i, j],$ weight$[i + 1] + W[i, j -$ profit$[i + 1]]$;

DKnap $:= \max_j\{W[n, j] \leq \sum_i$ weight$[i]\}$;
**end**.

Ας δούμε τώρα πώς μπορούμε να αλλάξουμε λίγο τον παραπάνω αλγόριθμο προκειμένου να πάρουμε ένα FPTAS. Η ιδέα είναι να αγνοήσουμε μερικά λιγότερο σημαντικά ψηφία των $p_i$, έτσι ώστε τελικά τα $p_i$, άρα και το $P$, να είναι πολυωνυμικά ως προς το μέγεθος της εισόδου ($n$) και ως προς την παράμετρο σφάλματος $1/\varepsilon$. Οπότε εργαζόμαστε όπως φαίνεται στον αλγόριθμο 12.3. Μπορεί να αποδειχθεί ότι για τη λύση $SOL$ που επιστρέφει ο αλγόριθμος αυτός ισχύει:

$$SOL \geq (1 - \varepsilon)OPT$$

Η πολυπλοκότητα του αλγορίθμου είναι $O(n^2 \cdot P') = O\left(\left\lceil \frac{n^3}{\varepsilon} \right\rceil\right)$, καθώς $P' = \lceil \frac{P}{K} \rceil = \lceil \frac{n}{\varepsilon} \rceil$. Επομένως ο αλγόριθμος είναι FPTAS.

**Αλγόριθμος 12.3** FPTAS για Discrete Knapsack

1. Δοθέντος του $\varepsilon$ όρισε το $= \frac{\varepsilon \cdot P}{n}$
2. Θέσε τα νέα κέρδη στις τιμές $p'_i = \lceil \frac{p_i}{K} \rceil$
3. Εκτέλεσε τον αλγόριθμο 12.2 με κέρδη τα $p'_i$

### 12.4.3 Traveling Salesman Problem

Στο τελευταίο αυτό μέρος θα εστιάσουμε στο πρόβλημα του πλανόδιου πωλητή, το οποίο θα δείξουμε ότι δεν επιδέχεται κανένα χρήσιμο παράγοντα προσέγγισης. Για να δείξουμε ότι ένα

πρόβλημα βελτιστοποίησης Π είναι δύσκολο να προσεγγιστεί γενικά υπάρχουν δύο μέθοδοι:

- Οι gap-introducing reductions, στις οποίες ανάγουμε ένα NP-complete πρόβλημα απόφασης Π′ στο πρόβλημα Π.
- Οι gap-preserving reductions, στις οποίες ανάγουμε ένα άλλο δύσκολα προσεγγίσιμο πρόβλημα Π′ στο πρόβλημα Π.

Ας δούμε τώρα την εφαρμογή μίας gap-introducing reduction από το πρόβλημα Hamilton Cycle στο TSP (θυμηθείτε ότι με αντίστοιχη αναγωγή δείξαμε ότι το TSP είναι NP-complete).

*Θεώρημα* 12.12. *Για κάθε πολυωνυμικά υπολογίσιμη συνάρτηση $\rho(n)$, το TSP δεν μπορεί να προσεγγιστεί με παράγοντα προσέγγισης $\rho(n)$, εκτός αν P=NP (όπου $n$ το πλήθος των κόμβων του γράφου).*

*Απόδειξη.* Ας υποθέσουμε ότι υπάρχει ένας $\rho(n)$-προσεγγιστικός αλγόριθμος για το TSP. Θα δείξουμε πώς ο αλγόριθμος αυτός μπορεί να χρησιμοποιηθεί για να επιλύσουμε το πρόβλημα Hamilton Cycle. Η κεντρική ιδέα είναι μία αναγωγή από ένα Hamilton Cycle στιγμιότυπο σε ένα TSP στιγμιότυπο, που μετασχηματίζει έναν γράφο $G$ με $n$ κορυφές, σε έναν πλήρη γράφο $G'$ με $n$ κορυφές, έτσι ώστε:

- αν ο $G$ έχει κύκλο Hamilton, τότε το κόστος της βέλτιστης λύσης στον TSP είναι $n$ και
- αν ο $G$ δεν έχει κύκλο Hamilton, τότε το κόστος της βέλτιστης λύσης στον TSP είναι $> \rho(n) \cdot n$

Παρατηρήστε τώρα ότι μπορούμε να αποφανθούμε για το αν ο $G$ έχει κύκλο ή όχι, εκτελώντας τον $\rho(n)$-προσεγγιστικό αλγόριθμο για το TSP: στην πρώτη περίπτωση, ο προσεγγιστικός αλγόριθμος θα επιστρέψει μία λύση που θα είναι $\leq \rho(n) \cdot OPT = \rho(n) \cdot n$, ενώ στη δεύτερη θα επιστρέψει μία λύση που θα είναι $\geq OPT > \rho(n) \cdot n$. Ανάλογα λοιπόν με την τιμή που επιστρέφεται για τον $G'$ μπορούμε να καταλάβουμε τι ισχύει για τον $G$.

Η αναγωγή τώρα προκειμένου να πετύχουμε τα ως άνω είναι πολύ απλή: αναθέτουμε βάρος 1 σε κάθε ακμή του αρχικού γράφου $G$ και βάρος $\rho(n) \cdot n$ σε όλες τις υπόλοιπες ακμές, ώστε να αποκτήσουμε τον πλήρη γράφο $G'$. Η αναγωγή γίνεται προφανώς σε πολυωνυμικό χρόνο. Παρατηρήστε τώρα ότι, αν ο $G$ έχει κύκλο Hamilton, τότε η βέλτιστη λύση του TSP είναι προφανώς αυτός ο κύκλος, κόστους $n$. Αντίθετα, σε περίπτωση που ο $G$ δεν έχει κύκλο Hamilton, η βέλτιστη λύση στο TSP θα περνά από τουλάχιστον μία ακμή βάρους $\rho(n) \cdot n$ και έτσι το συνολικό της κόστος θα είναι $OPT > \rho(n) \cdot n$.

## 12.5 Διαδραστικό Υλικό – Σύνδεσμοι

- Βιβλίο για σχεδίαση προσεγγιστικών αλγορίθμων από τους D. Williamson, D. Shmoys, Cambridge University Press:

- διαθέσιμη ηλεκτρονική έκδοση.
- Οπτικοποίηση αλγορίθμων για το πρόβλημα του Πλανόδιου Πωλητή (βίντεο): https://www.youtube.com/watch?v=q6fPk0--eHY
- Ο αντίστοιχος πηγαίος κώδικας: https://github.com/gumpu/TSP_Animation

  Διαφάνειες για προσεγγιστικούς αλγορίθμους απο τον Shmuel Safra: http://www.tau.ac.il/~safra/Complexity/Approximation%20Problems.ppsx

## 12.6 Ασκήσεις

1. Σκεφτείτε τον εξής προσεγγιστικό αλγόριθμο για το Traveling Salesman Problem: βρες πρώτα ένα ελάχιστο συνδετικό δένδρο του γράφου, μετά κάνε τις ακμές του "διπλές", και βρες έναν κύκλο Euler πάνω στο διπλασιασμένο δένδρο. Στη συνέχεια, πάρε τον κύκλο που προκύπτει από τον κύκλο Euler που βρήκες, διαγράφοντας κάθε κορυφή που έχεις ήδη συναντήσει στον κύκλο.

   (α) Εκτελέστε τον αλγόριθμο σε ένα παράδειγμα της επιλογής σας.

   (β) Αποδείξτε ότι για γράφους όπου ισχύει η τριγωνική ανισότητα, ο αλγόριθμος αυτός είναι 2-προσεγγιστικός.

   (γ) Αποδείξτε ότι για γενικούς γράφους δεν μπορεί να επιτύχει 2-προσέγγιση. Μπορεί να πετύχει κάποιον άλλο σταθερό λόγο προσέγγισης;

2. Θεωρήστε τον εξής αλγόριθμο για το πρόβλημα Vertex Cover: βρές ένα δένδρο με αναζήτηση κατά βάθος (DFS) στον δοσμένο γράφο $G$ και δώσε σαν απάντηση το σύνολο των κόμβων $S$ που δεν είναι φύλλα. Αποδείξτε ότι το $S$ είναι πράγματι vertex cover του $G$ και ότι $|S| <= 2OPT$. (Υπόδειξη: δείξτε ότι υπάρχει matching στον $G$ με τουλάχιστον $|S|/2$ ακμές.)

3. Διαβάστε τον 2-προσεγγιστικό αλγόριθμο για το πρόβλημα $k$-Center (ορισμός εδώ), που αναφέρεται και στο βιβλίο των Williamson-Shmoys (δείτε σύνδεσμο), σελ. 38.

   Εκτελέστε τον αλγόριθμο σε παράδειγμα της επιλογής σας. Εξετάστε αν ισχύει ο λόγος προσέγγισης στην περίπτωση που δεν ισχύει η τριγωνική ανισότητα.

4. Το πρόβλημα 2-Color Bounded Matching ορίζεται ως εξής: δίνεται γράφος όπου κάθε ακμή του έχει ένα χρώμα (μπλε ή κόκκινο) και δίνεται επιπλέον ένας ακέραιος $W$. Ζητείται να βρεθεί το μέγιστο (σε πληθικότητα) matching που να έχει το πολύ $W$ ακμές από κάθε χρώμα.

   (α) Βρείτε έναν αποδοτικό 2-προσεγγιστικό αλγόριθμο για το πρόβλημα. Θυμηθείτε ότι μπορούμε να λύνουμε το πρόβλημα του μέγιστου ταιριάσματος βέλτιστα με αποδοτικό αλγόριθμο.

   (β) Μπορείτε να προσαρμόσετε τον αλγόριθμό σας για $k$ χρώματα; Τι λόγο προσέγγισης επιτυγχάνει;

5. Θεωρήστε το πρόβλημα Partition: δίνεται ένα σύνολο $A = \{a_1, \ldots, a_n\}$ φυσικών αριθμών και ζητείται αν υπάρχει υποσύνολο $A' \subseteq A$ με άθροισμα στοιχείων ακριβώς ίσο με $A/2$.

   Μπορείτε να βρείτε αλγόριθμο δυναμικού προγραμματισμού που να λύνει το πρόβλημα Partition ακριβώς; Ποιά είναι η πολυπλοκότητα του αλγορίθμου; Είναι πολυωνυμικός; Τι μπορούμε να πούμε για το πρόβλημα;

6. Δώστε έναν $f$–προσεγγιστικό αλγόριθμο για το πρόβλημα Set Cover (δείτε ορισμό στο [1] ή [2], όπου $f$ είναι το μέγιστο πλήθος συνόλων στα οποία μπορεί να ανήκει κάποιο στοιχείο. Αποδείξτε τον λόγο προσέγγισης.

   Υπόδειξη: δείτε το Set Cover σαν γενίκευση του Vertex Cover και προσπαθήστε να γενικεύσετε την ιδέα του 2-προσεγγιστικού αλγορίθμου για το Vertex Cover.

# Βιβλιογραφία


[1] V. V. Vazirani. Approximation Algorithms. Springer, 2001.

[2] D. B. Shmoys and D. P. Williamson. The Design of Approximation Algorithms. Cambridge University Press, 2011.

[3] R. Motwani and P. Raghavan. Randomized Algorithms. Cambridge University Press, 1995.

[4] S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani. "Algorithms", MacGraw-Hill, 2006. "Αλγόριθμοι", ελληνική έκδοση, Εκδόσεις Κλειδάριθμος, 2008.

[5] Thomas Cormen, Charles Leiserson, Ronald Rivest and Cliff Stein: "Introduction to Algorithms", 3rd edition, MIT Press, 2009. Ελληνική έκδοση: "Εισαγωγή στους Αλγόριθμους", Πανεπιστημιακές Εκδόσεις Κρήτης, 2012.

# Κεφάλαιο 13

# Λειτουργικά Συστήματα

## 13.1 Εισαγωγή

Κάθε υπολογιστικό σύστημα αποτελείται από το υλικό (hardware) και το λογισμικό (software). Με τον όρο υλικό αναφερόμαστε στο σύνολο των συσκευών όπως Κεντρική Μονάδα επεξεργασίας, μονάδες αποθήκευσης, μονάδες εισόδου-εξόδου (εκτυπωτές, πληκτρολόγιο, οθόνη). Με τον όρο λογισμικό αναφερόμαστε τόσο στο λειτουργικό σύστημα όσο και στα προγράμματα εφαρμογής που γράφει ο χρήστης προκειμένου να δώσει εντολές στον υπολογιστή για τον τρόπο χρησιμοποίησης των συσκευών.

Λειτουργικό σύστημα είναι το σύνολο των προγραμμάτων που ασχολούνται με τη διαχείριση και τον συντονισμό των εργασιών, καθώς και την κατανομή των διαθέσιμων πόρων. Το λειτουργικό σύστημα ουσιαστικά αποτελεί το σύνδεσμο μεταξύ λογισμικού και υλικού του υπολογιστικού συστήματος. Μια από τις βασικές αρμοδιότητες του λειτουργικού συστήματος είναι η διαχείριση του υλικού. Με αυτό τον τρόπο οι εφαρμογές του χρήστη απλουστεύονται εφόσσον απαλλάσονται από τον άμεσο και επίπονο χειρισμό του υπολογιστή και παρέχεται έτσι η δυνατότητα σε χρήστες με περιορισμένες γνώσεις υπολογιστών να αναπτύσουν τις εφαρμογές τους. Αν για παράδειγμα δεν υπήρχε το λειτουργικό σύστημα θα όφειλε ο προγραμματιστής να γράφει πρόσθετο κώδικα ο οποίος θα παρακολουθούσε τακτικά αν υπάρχει κάποια είσοδος από το πληκτρολόγιο, θα έγραφε κώδικα για να στείλει κάτι προς εκτύπωση και ουτω καθεξής. Αρμοδιότητα επίσης του λειτουργικού συστήματος αποτελεί η βέλτιστη χρήση του υλικού έτσι ώστε να αυξάνεται η αποδοτικότητα του υπολογιστικού συστήματος.

## 13.2 Ιστορική Αναδρομή

Τη δεκαετία του ’40 παρουσιάστηκαν οι πρώτοι Η/Υ οι οποίοι βασίζονταν στις λυχνίες κενού και δεν διέθεταν λειτουργικό σύστημα (λειτουργικά συστήματα μηδενικής γενιάς). Οι χρήστες των πρώτων Η/Υ ήταν μόνο έμπειροι προγραμματιστές οι οποίοι έδιναν εντολές χειριζόμενοι τους διακόπτες και τα σήματα ελέγχου είχαν άμεση προσπέλαση στη γλώσσα μηχανής και προγραμμάτιζαν τα πάντα διότι δεν υπήρχε λειτουργικό σύστημα να αναλάβει τη διαχείριση του

υλικού. Την περίοδο εκείνη ένας άνθρωπος ή μία ομάδα ανθρώπων σχεδίαζε, υλοποιούσε, προγραμμάτιζε και συντηρούσε τον υπολογιστή. Αυτό που επι το πλείστον έκαναν αυτοί οι πρώτοι υπολογιστές ήταν αριθμητικοί υπολογισμοί (ημίτονο, συνημίτονο, λογάριθμος) και ο τρόπος που γινόταν ήταν με καλωδίωση των πινάκων συνδέσεων και σε γλώσσα μηχανής δεδομένου ότι δεν υπήρχαν οι γλώσσες προγραμματισμού. Το να καεί δε μία λυχνία κατά τη διάρκεια της εκτέλεσης σήμαινε επανάληψη της διαδικασίας από την αρχή.

Στα μέσα της δεκαετίας του ’50 η χρήση των τρανζίστορ κάνει τους υπολογιστές πιο αξιόπιστους. Δεν υπήρχε σοβαρός κίνδυνος να διακοπεί η εκτέλεση μιας εργασίας και να χρειαστεί να ξεκινήσει από την αρχή. Αυτοί οι υπολογιστές που είναι γνωστοί ως mainframes τοποθετούνταν σε μεγάλους κλιματιζόμενους χώρους και προσωπικό διαφορετικών ειδικοτήτων απασχολούνταν με αυτούς. Οι πελάτες που αγόραζαν αυτούς τους υπολογιστές ήταν μεγάλες εταιρείες, κρατικές υπηρεσίες και πανεπιστήμια. Οι λόγοι μεταξύ άλλων ήταν το μέγεθός τους και το κόστος τους. Ο τρόπος που γίνονταν οι εργασίες διέφερε αρκετά από την προηγούμενη γενιά. Καταρχάς υπήρχε εξειδίκευση στο προσωπικό που χρησιμοποιούσε τους υπολογιστές και έτσι είχαμε σχεδιαστές, κατασκευαστές, χειριστές, προγραμματιστές και προσωπικό συντήρησης.Έτσι αρχικά κάποιος έγραφε σε χαρτί το πρόγραμμα που επίλυε κάποιο πρόβλημα. Η γλώσσα που χρησιμοποιούσε ήταν assembly ή fortran. Στη συνέχεια ο κώδικας γραφόταν σε διάτρητες κάρτες. Οι χεριστές έπαιρναν τις κάρτες και τις εισήγαγαν στον υπολογιστή για ανάγνωση, ακολουθούσε η εκτέλεση του κώδικα και ο χειριστής έπαιρνε τα αποτελέσματα που τυπωνόταν σε κάποιον εκτυπωτή και συνήθως δεν βρισκόταν στον ίδιο χώρο. Επειδή υπήρχε μεγάλη καθυστέρηση σε αυτή τη διαδικασία υιοθετήθηκε το σύστημα δέσμης (batch system). Στο σύστημα αυτό ένα πλήθος καρτών με κώδικα δέσμης εργασιών συγκεντρωνόταν (σε χρονικό διάστημα μιας ώρας για παράδειγμα) και ένας υπολογιστής χαμηλού κόστους διάβαζε τις κάρτες και τις έγραφε σε μαγνητική ταινία. Η ταινία με τα προγράμματα καθώς και το πρόγραμμα που θα μπορούσε να χαρακτηριστεί ως κώδικας λειτουργικού συστήματος φορτωνόταν στον υπολογιστή που εκτελούσε τις εργασίες σειριακά και στη συνέχεια έγραφε τα αποτελέσματα σε ταινία και όχι στον εκτυπωτή. Όταν ολοκληρωνόταν και η τελευταία εργασία ο χειριστής τοποθετούσε την ταινία με τα αποτελέσματα σε έναν εκτυπωτή ο οποίος δεν ήταν συνδεδεμένος με τον υπολογιστή και έτσι ο υπολογιστής έπαιρνε την επόμενη δέσμη εργασιών. Έτσι έχουμε τα λειτουργικά συστήματα της πρώτης γενιάς που ονομάζονται Λειτουργικά Συστήματα ομαδικής επεξεργασίας (batch processing).

Τα λειτουργικά συστήματα πρώτης γενιάς δέχονταν τη δέσμη με τα προγράμματα που υπέβαλαν οι χρήστες και τα επεξεργαζόταν το ένα μετά το άλλο με τη σειρά. Βασικό μειονέκτημα αυτών των λειτουργικών ήταν η υποαπασχόληση των συσκευών. Τα προγράμματα μπορούσαν να εκτελεστούν σειριακά και καμία ταυτόχρονη εκτέλεση δεν ήταν εφικτή. Έτσι δεδομένου ότι ένα πρόγραμμα δεν θα μπορούσε να απασχολεί ταυτόχρονα όλες τις συσκευές υλικού αναγκαστικά όλες οι υπόλοιπες συσκευές παρέμεναν ανενεργές. Ο χρόνος ανακύκλωσης ενός προγράμματος σε αυτά τα λειτουργικά συστήματα θα μπορούσε να είναι 50-60 φορές μεγαλύτερος από τον χρόνο εκτέλεσης του συγκεκριμένου προγράματος. Ως χρόνος ανακύκλωσης ορίζεται ο χρόνος από τη στιγμή της υποβολής ενός προγράμματος έως τη στιγμή λήψης των αποτελεσμάτων. Οι υπολογιστές της γενιάς αυτής χρησιμοποιήθηκαν κυρίως για επιστημονικούς υπολογισμούς όπως η επίλυση μερικών διαφορικών εξισώσεων. Η γλώσσα προγραμματισμού είναι η Fortran.

Στη δεκαετία του ’60 έχουμε τη δεύτερη γενιά λειτουργικών συστημάτων. Τα συστήματα αυτά ονομάζονται Λετουργικά Συστήματα Πολυπρογραμματισμού (multiprogramming). Στα συστή-

ματα αυτά μειώθηκε σημαντικά ο άεργος χρόνος των συσκευών του υλικού και αυτό είχε ως συνέπεια την μείωση του χρόνου ανακύκλωσης. Θα μπορούσε να ειπωθεί πως με τα συστήματα αυτά υπήρχε ένα είδος παράλληλης επεξεργασίας των προγραμμάτων που υποβάλλονται σε κάποιο υπολογιστή αλλά αυτή η παραλληλία είναι εικονική. Ο επεξεργαστής μία λειτουργία εκτελεί κάθε φορά αλλά διακόπτεται με τη χρήση σημάτων διακοπής (interrupts) προκειμένου να επιτευχθεί η βέλτιστη δυνατή χρήση των συσκευών. Έτσι ένα πρόγραμμα μπορεί να εκτελείται στην Κεντρική Μονάδα Επεξεργασίας, ένα άλλο μπορεί να διαβάζει από το σκληρό δίσκο, ένα άλλο να στέλνει δεδομένα στη συσκευή εξόδου και ουτω καθεξής. Το βασικό μειονέκτημα των λειτουργικών συστημάτων της δεύτερης γενιάς σε σχέση με αυτά της προηγούμενης είναι πως το λειτουργικό σύστημα γίνεται πλέον ένα πιο πολύπλοκο πρόγραμμα που απασχολεί και αυτό τον υπολογιστή εις βάρος των προς εκτέλεση προγραμμάτων.

Ακολουθούν τα λειτουργικά συστήματα καταμερισμού χρόνου. Σε αυτά κάθε χρήστης πληκτρολογεί τον κώδικά του σε ένα τερματικό και ένα υπολογιστικό σύστημα αναλαμβάνει την εκτέλεση των προγραμμάτων των χρηστών. Οι χρήστες έχουν την εντύπωση πως εξυπηρετούνται παράλληλα αλλά αυτό που γίνεται ουσιαστικά είναι ο καταμερισμός χρόνου του υπολογιστικού συστήματος. Το υπολογιστικό σύστημα δηλαδή καταμερίζει το χρόνο του εκ περιτροπής σε κάθε χρήστη. Το ρόλο του ρολογιού εδώ έχει ένα κύκλωμα που ονομάζεται timer (χρονιστής) και το οποίο σε τακτά χρονικά διαστήματα στέλνει σήμα διακοπής. Σε περίπτωση που κάποιο πρόγραμμα πρέπει να εκτελεστεί χωρίς καθυστέρηση δίνεται προτεραιότητα σε αυτό και εφόσσον ολοκληρωθεί η εκτέλεσή του συνεχίζεται ο καταμερισμός του χρόνου στα υπόλοιπα προγράμματα.Η συνεχής εξέλιξη του υλικού καθώς και η ανάγκη για μεγαλύτερες ταχύτητες οδήγησε στην ανάπτυξη πιο εξελιγμένων και πιο πολύπλοκων λειτουργικών συστημάτων. Οι παράλληλοι και κατανεμημένοι υπολογιστές καθώς και τα δίκτυα υπολογιστών ήταν εκείνα που δημιούργησαν τις νέες ανάγκες για αποδοτικότερα λειτουργικά συστήματα. Στα συστήματα αυτά γίνεται πραγματική και όχι εικονική παράλληλη επεξεργασία εφόσσον κάθε επεξεργαστής ασχολείται με διαφορετικό πρόγραμμα.

Στα παράλληλα συστήματα έχουμε πολλές κεντρικές μονάδες επεξεργασίας στο ίδιο μηχάνημα. Με αυτό τον τρόπο επιτυγχάνεται η παράλληλη (ταυτόχρονη) επεξεργασία πολλών προγραμμάτων εφόσσον δεν είναι αναγκαίο να εξυπηρετηθούν όλα από την ίδια ΚΜΕ. Στα κατανεμημένα συστήματα οδήγησε η δικτύωση και η διαδικτύωση. Μία εργασία μπορεί πλέον να διεκπεραιωθεί σε έναν υπολογιστή που βρίσκεται πολλά χιλιόμετρα μακριά. Υπάρχει επίσης η δυνατότητα να εκτελείται τμήμα του κώδικα σε έναν υπολογιστή, άλλο τμήμα του σε άλλο υπολογιστή αρκεί να υπάρχει δικτύωση μεταξύ τους. Τα σημερινά λειτουργικά συστήματα των προσωπικών υπολογιστών υποστηρίζουν πολυπρογραμματισμό και αυτό γίνεται φανερό από την εκκίνηση του υπολογιστή όπου πολλά προγράμματα τρέχουν ταυτόχρονα. Τα πιο γνωστά εξ'αυτών είναι τα Windows, Linux, Macintosh. Η συνεχής εξέλιξη του υλικού καθώς και η ανάγκη για μεγαλύτερες ταχύτητες οδήγησε στην ανάπτυξη πιο εξελιγμένων και πιο πολύπλοκων λειτουργικών συστημάτων. Οι παράλληλοι και κατανεμημένοι υπολογιστές καθώς και τα δίκτυα υπολογιστών ήταν εκείνα που δημιούργησαν τις νέες ανάγκες για αποδοτικότερα λειτουργικά συστήματα. Στα συστήματα αυτά γίνεται πραγματική και όχι εικονική παράλληλη επεξεργασία εφόσσον κάθε επεξεργαστής ασχολείται με διαφορετικό πρόγραμμα.

Λειτουργικά συστήματα σήμερα έχουμε επίσης και στους υπολογιστές χειρός. Τα λειτουργικά συστήματα που εκτελούνται στους υπολογιστές χειρός είναι αρκετά περίπλοκα δεδομένου ότι

υπάρχουν απαιτήσεις για χειρισμό τηλεφωνίας, ψηφιακής φωτογραφίας κ.λπ. Μια άλλη κατηγορία σημερινών λειτουργικών συστημάτων είναι αυτά που χρησιμοποιούνται σε συσκευές οι οποίες δεν είναι ακριβώς υπολογιστές όπως η τηλεόραση, ο φούρνος μικροκυμάτων κ.λπ. Αυτά είναι τα ενσωματωμένα λειτουργικά συστήματα και η βασική τους διαφορά από τα λειτουργικά συστήματα χειρός είναι πως δεν θα απαιτηθεί να δώσει κάποιος χρήστης ένα πρόγραμμα για εκτέλεση. Τέλος θα πρέπει να αναφερθούν και τα λετουργικά συστήματα πραγματκού χρόνου στα οποία ο χρόνος είναι πολύ σημαντικός. Για παράδειγμα στη βιομηχανία ένα ρομπότ θα πρέπει να λειτουργήσει σωστά κάποια συγκεκριμένη χρονική στιγμή. Σε περίπτωση που αυτό δεν γίνει όταν ακριβώς πρέπει πιθανόν να καταστραφεί ολόκληρη η παραγωγή. Λειτουργικά συστήματα πραγματικού χρόνου συναντούμε στην αυτοκινητοβιομηχανία, στην βιομηχανία αεροσκαφών κ.λπ. Στις έξυπνες κάρτες επίσης εκτελούνται κάποια λειτουργικά συστήματα.

## 13.3 Βασικές έννοιες των Λειτουργικών Συστημάτων

Μία από τις πιο βασικές έννοιες των λετουργικών συστημάτων είναι η διεργασία. Διεργασία ονομάζεται ένα πρόγραμμα ή ένα αυτόνομο τμήμα προγράμματος που εκτελείται. Η διαφορά της διεργασίας από το πρόγραμμα είναι πως το πρόγραμμα είναι ανενεργό, ουσιαστικά είναι ένα σύνολο εντολών αποθηκευμένων στη μνήμη. Η διεργασία είναι ένα πρόγραμμα που έχει ξεκινήσει να εκτελείται αλλά δεν έχει ολοκληρωθεί. Μια διεργασία μπορεί να βρίσκεται σε μία από τις παρακάτω τρεις καταστάσεις:

Να χρησιμοποιεί την ΚΜΕ.

Να έχει διακοπεί προσωρινά για να εκτελεστεί κάποια άλλη.

Να είναι μπλοκαρισμένη γιατί περιμένει κάτι για να συνεχίσει.

Στην πρώτη περίπτωση η διεργασία εκτελείται κανονικά. Στη δεύτερη η εκτέλεσή της έχει διακοπεί προκειμένου να παραχωρηθεί η ΚΜΕ σε κάποια άλλη διεργασία η οποία θα εκτελεστεί για κάποιο χρονικό διάστημα έως ότου ολοκληρωθεί ή διακοπεί για να εκτελεστεί κάποια άλλη. Στην τρίτη περίπτωση η διεργασία ενδέχεται να περιμένει κάποια δεδομένα για παράδειγμα που δεν είναι διαθέσιμα εκείνη τη στιγμή.Ας δούμε όμως πιο πρακτικά τι σημαίνει διεργασία. Ας υποθέσουμε ότι έχουμε γράψει μία εφαρμογή σε κάποια γλώσσα προγραμματσμού και απατείται κάποιο χρονικό διάστημα για να πάρουμε τα πειραματικά αποτελέσματα. Ώσπου να ολοκληρωθεί αυτή η διαδικασία κάνουμε μια αναζήτηση στο διαδίκτυο. Παράλληλα είναι ενεργοποιημένη μία διεργασία η οποία ελέγχει αν υπάρχουν εισερχόμενα μηνύματα ηλεκτρονικού ταχυδρομείου. Τι συμβαίνει με το λειτουργικό σύστημα σε αυτή την περίπτωση; Υπάρχουν τρεις διεργασίες που εκτελούνται ταυτόχρονα: εκείνη του προγράμματος που υπολογίζει τα πειραματικά αποτελέσματα, εκείνη της αναζήτησης στο διαδίκτυο και εκείνη που ελέγχει και ενημερώνει για την ύπαρξη νέων μηνυμάτων. Το λειτουργικό σύστημα διανέμει την ΚΜΕ στις τρεις διεργασίες. Εκείνο είναι που περιοδικά αποφασίζει πότε έχει ολοκληρωθεί ο χρόνος δέσμευσης της ΚΜΕ από τη μία διεργασία και αποφασίζει να συνεχιστεί η εκτέλεση της επόμενης διεργασίας. Κάθε διεργασία διακόπτεται και συνεχίζεται όταν έρθει ξανά η σειρά της για να εκτελστεί. Θα πρέπει

όμως κάθε φορά που διακόπτεται η εκτέλεση μιας διεργασίας να κρατούνται όλες οι πληροφορίες που απαιτούνται έτσι ώστε όταν έρθει πάλι η σειρά της να χρησιμοποιήσει την ΚΜΕ να μπορεί να συνεχιστεί η εκτέλεσή της. Ας υποθέσουμε για παράδειγμα πως η διεργασία έχει ανοίξει κάποια αρχεία για διάβασμα και κάποια για εγγραφή. Όταν διακοπεί για να εκτελεστεί μια άλλη διεργασία θα πρέπει να ξέρει για κάθε αρχείο τηντρέχουσα θέση στην οποία βρισκόταν πριν ανασταλεί η εκτέλεσή της έτσι ώστε να συνεχίσει την ανάγνωση ή την εγγραφή από τη σωστή θέση. Στα περισσότερα λειτουργικά συστήματα όλες αυτές οι πληροφορίες οι σχετικές με την αναστολή εκτέλεσης της διεργασίας αποθηκεύονται στον πίνακα διεργασιών. Στον πίνακα αυτό υπάρχουν όλες οι διεργασίες των οποίων η εκτέλεση έχει ανασταλεί με τις τιμές των καταχωρητών της κ.λπ σε ένα πίνακα ή μία δομή. Οι διεργασίες "ανταγωνίζονται" η μία την άλλη στην χρήση των πόρων του υπολογιστικού συστήματος. Για παράδειγμα ενώ μια διεργασία εκτελείται κάποια άλλη περιμένει να χρησιμοποιήσει την ΚΜΕ. Τον συγχρονισμό των εργασιών τον αναλαμβάνει ο διαχειριστής διεργασιών(ή χρονοπρογραμματιστής διεργασιών) ο οποίος χρησιμοποιεί ουρές αναμονής και αποφασίζει πότε θα εκτελεστεί μία διεργασία, πότε θα διακοπεί, ποιά θα είναι η επόμενη και για πόσο χρονικό διάστημα θα εκτελείται η τρέχουσα διεργασία. Οι πληροφορίες κάθε διεργασίας αποθηκεύονται στο τμήμα ελέγχου διεργασίας που συσχετίζεται άμεσα με την διεργασία. Ο διαχειριστής διεργασιών αποθηκεύει στις ουρές αναμονής το τμήμα ελέγχου κάθε διεργασίας και όχι την ίδια τη διεργασία διότι το μέγεθός τους είναι μεγάλο ώστε να μπουν ολόκληρες στη λίστα αναμονής. Έτσι το τμήμα ελέγχου διεργασίας παραμένει στην ουρά αναμονής και η ίδια η διεργασία είναι αποθηκευμένη στον δίσκο ή στη μνήμη. Η ουρά αναμονής για την ΚΜΕ δεν είναι η μοναδική, υπάρχουν ουρές για τις συσκευές εισόδου εξόδου και μάλιστα καθε τέτοια συσκευή έχει τη δική της λίστα αναμονής, λίστα για χρήση μνήμης κ.λπ. Η επιλογή της επόμενης διεργασίας σε κάθε ουρά αποφασίζεται από τον διαχειριστή διεργασιών ανάλογα με τον τρόπο που είναι προγραμματισμένος να επιλέξει την επόμενη εργασία. Υπάρχουν διάφοροι αλγόριθμοι που συντονίζουν τον χρονοπρογραμματισμό των εργασιών λαμβάνοντας υπόψη τόσο την αποδοτικότητα του συστήματος όσο και την απονομή "δικαιοσύνης" στο χρόνο απασχόλησης της ΚΜΕ. Για παράδειγμα το σχήμα First In First Out (FIFO) επιτρέπει στην διεργασία που μπήκε πρώτη στην ουρά αναμονής να εξυπηρετηθεί πρώτη. Σε άλλες υλοποιήσεις εξυπηρετείται πρώτα η πιο σύντομη ή αυτή που έχει μεγαλύτερη προτεραιότητα κ.λπ.

## 13.4 Κρίσιμα Τμήματα και Αμοιβαίος Αποκλεισμός Διαδιεργασιακή επικοινωνία

Οι διεργασίες κάποιες φορές επικοινωνούν. Για παράδειγμα η μία περιμένει κάποια δεδομένα εισόδου από την έξοδο κάποιας άλλης διεργασίας. Εδώ υπάρχει το πρόβλημα του πως θα μεταφερθούν τα δεδομένα από τη μία διεργασία στην άλλη καθώς και πως θα γίνει ο συγχρονισμός των διεργασιών με τέτοιο τρόπο που να μπορεί η διεργασία που περιμένει δεδομένα να μην εκτελεστεί εως ότου τα λάβει. Επίσης πολλές φορές συμβαίνει σε κάποιο υπολογιστικό σύστημα να υπάρχουν πόροι οι οποίοι μοιράζονται. Για παράδειγμα τι θα συνέβαινε αν δύο πελάτες προσπαθούσαν να κλείσουν ταυτόχρονα το τελευταίο αεροπορικό εισιτήριο που διαθέτει μία εταιρεία. Θα πρέπει με κάποιο τρόπο να εξασφαλιστεί πως ο ένας από τους δύο θα δεσμεύσει τη θέση.

Δεν πρέπει ούτε να την πάρουν και οι δύο ούτε να μην την πάρει κανείς. Μια άλλη περίπτωση είναι αυτή των ΑΤΜ μιας τράπεζας. Έστω πως δύο άνθρωποι έχουν κοινό λογαριασμό σε μία τράπεζα. Και έστω πως πηγαίνουν και οι δύο την ίδια χρονική στιγμή να πάρουν χρήματα από δύο διαφορετικά ΑΤΜ. Τι θα συμβεί; Θα πάρουν τα χρήματα και οι δύο και αν ναι θα πάρουν τη σωστή απόδειξη για το υπόλοιπο του λογαριασμού τους και πως αυτό θα εξασφαλιστεί; Υπάρχει περίπτωση να συμβεί το εξής: Τη χρονική στιγμή t1 ο πρώτος από τους δύο συνδικαιούχους ζητά να κάνει ανάληψη. Το ΑΤΜ του ζητά το ποσό το οποίο θέλει να πάρει και διαβάζει το ποσό που πληκτρολογείται. Έστω για παράδειγμα ζητά 1000 ευρώ. Ο δεύτερος συνδικαιούχος τη χρονική στιγμή t2 ζητά να εισπράξει το ποσό των 2000 ευρώ. Η πρώτη διεργασία που συνδέται με τον πρώτο συνδικαιούχο αναζητά το υπόλοιπο του λογαριασμού. Η δεύτερη διεργασία που συνδέεται με τον δεύτερο συνδικαιούχο κάνει το ίδιο με κάποια καθυστέρηση. Η πρώτη διεργασία υπολογίζει το υπόλοιπο που απομένει στον λογαριασμό αν αφαιρεθεί το ποσό των 1000 ευρώ. Αν υποθέσουμε ότι ο λογαριασμός είχε 10000 ευρώ το υπόλοιπο είναι 9000 ευρώ. Και την ώρα που γίνεται αυτός ο υπολογισμός η δεύτερη διεργασία ξεκινά να υπολογίζει ακριβώς το ίδιο έχοντας ως υπόλοιπο το αρχικό υπόλοιπο που είναι οι δέκα χιλιάδες ευρώ και όχι αυτό που απομένει όταν ο πρώτος συνδικαιούχος πάρει το ποσό που ζήτησε. Έτσι το υπόλοιπο που τυπώνεται στον πρώτο πελάτη είναι 9000 ευρώ και στον δεύτερο 8000 ευρώ. Το σωστό θα ήταν να τυπωθεί στον δεύτερο πελάτη το ποσό των 7000 ευρώ. Για να μην συμβαίνουν τέτοια λάθη θα πρέπει τη στιγμή που ξεκινά να εκτελείται αυτό το τμήμα της διεργασίας το οποίο ονομάζεται κρίσιμο τμήμα να μην επιτρέπεται σε καμιά άλλη διεργασία να διακόψει την εκτέλεση του κρισίμου τμήματος της πρώτης. Το φαινόμενο αυτό δεν παρουσιάζεται μόνο με την ΚΜΕ αλλά και με την χρήση άλλων πόρων του συστήματος. Για παράδειγμα η χρήση του εκτυπωτή. Όταν μία διεργασία πρόκειται να εκτυπώσει ένα αρχείο αποθηκεύει το όνομα του αρχείου στην ουρά εκτύπωσης έως ότου ο εκτυπωτής ελευθερωθεί και μπορεί να εκτυπώσει. Μια διεργασία ελέγχει κάθε φορά αν υπάρχουν αρχεία προς εκτύπωση και αν ναι τότε τα τυπώνει και διαγράφει το όνομά τους από την ουρά εκτύπωσης. Υπάρχουν δύο μεταβλητές οι in και η out οι οποίες δείχνουν αντίστοιχα στην επόμενη ελεύθερη θέση στην ουρά και στη θέση του επόμενου αρχείου που θα εκτυπωθεί. Οι τιμές των δύο αυτών μεταβλητών είναι διαθέσιμες σε κάθε διεργασία. Υποθέτουμε ότι δύο διεργασίες αποφασίζουν να ζητήσουν εκτύπωση σχεδόν ταυτόχρονα. Τι θα συμβεί σε αυτή την περίπτωση; Έστω πως υπάρχουν κάποια αρχεία στην ουρά εκτύπωσης και η επόμενη θέση για την προσθήκη αρχείου είναι η θέση 10. Η πρώτη διεργασία (με ελάχιστη διαφορά από τη δεύτερη) αποφασίζει να εκτυπώσει και αποθηκεύει σε μία μεταβλητή της την τιμή 10 για να γνωρίζει σε ποιά θέση θα αποθηκεύσει το αρχείο που θέλει να εκτυπώσει. Ακριβώς μόλις αποθηκευτεί η τιμή αυτή γίνεται μία διακοπή στην εκτέλεση της διεργασίας γιατί ολοκληρώθηκε ο χρόνος που της είχε δωθεί. Συνεχίζει τώρα η εκτέλεση της δεύτερης διεργασίας η οποία επίσης θέλει να εκτυπώσει. Διαβάζει και εκείνη την τιμή της μεταβλητής σχετικά με την επόμενη ελεύθερη θέση για την προσθήκη ονόματος αρχείου για εκτύπωση και διαπιστώνει πως η τιμή αυτή είναι 10. Άρα αυτή τη στιγμή και οι δύο διεργασίες θεωρούν πως η επόμενη θέση της ουράς είναι η δέκατη. Ο χρόνος εκτέλεσης της δεύτερης διεργασίας δεν έχει ολοκληρωθεί και επομένως αυτή προσθέτει το όνομα του προς εκτύπωση αρχείου στη θέση 10. Η τιμή της μεταβλητής στην οποία αποθηκεύεται η επόμενη ελεύθερη θέση στην ουρά εκτύπωσης γίνεται 11. Η δεύτερη διεργασία συνεχίζει να εκτελείται. Κάποια στιγμή διακόπτεται η εκτέλεση για να δωθεί χρόνος στην επόμενη διεργασία. Όταν συνεχιστεί η εκτέλεση της πρώτης διεργασίας αυτή σκοπεύει να

αποθηκεύσει το όνομα του αρχείου που θέλει να εκτυπώση στη θέση 10 διότι αυτή την τιμή είχε κρατήσει πρν διακοπεί η εκτέλεσή της. Πηγαίνει λοιπόν στη θέση 10, διαγράφει το αρχείο της διεργασίας Β και γράφει το όνομα του δικού της αρχείου. Το αρχεό που είχε αποθηκευτεί στην ουρά εκτύπωσης από τη δεύτερη διεργασία δεν θα εκτυπωθεί ποτέ διότι έχει διαγραφεί. Αυτές οι καταστάσεις που προσπαθούν δύο ή περισσότερες διεργασίες να χρησιμοποιήσουν ταυτόχρονα πόρους του συστήματος συνθήκες ανταγωνισμού. Και για να μην συμβαίνουν ανεξήγητα λάθη αυτές οι συνθήκες πρέπει να μην δημιουργούνται. Ο τρόπος για να γίνει αυτό ονομάζεται αμοιβαίος αποκλεισμός. Όταν μία διεργασία μπαίνει στο κρίσιμο τμήμα της θα πρέπει το λειτουργικό σύστημα να εμποδίζει οποιαδήποτε άλλη να μπει στο δικό της κρίσιμο τμήμα. Αυτό ονομάζεται αμοιβαίος αποκλεισμός και θα δούμε στη συνέχεια πως γίνεται. Όταν μία διεργασία μπαίνει στο κρίσιμο τμήμα της και εκτελείται οποιαδήποτε προσπάθεια άλλης διεργασίας να μπει στο δικό της κρίσιμο τμήμα δεν επιτρέπεται. Σε αυτή την περίπτωση η δεύτερη εργασία περιμένει να εκτελεστεί. Όταν ολοκληρωθεί η εκτέλεση του κρίσιμου τμήματος της πρώτης διεργασίας τότε διακόπτεται η εκτέλεσή της (δεδομένου ότι ο χρόνος της έχει ήδη εκπνεύσει) και συνεχίζεται η εκτέλεση του κρίσιμου τμήματος της δεύτερης διεργασίας. Ο Dijikstra το 1965 [1] έθεσε και τους περιορισμούς για την παράλληλη εκτέλεση διεργασιών και τα κρίσιμα τμήματα:

> Δύο διεργασίες δεν επιτρέπεται να βρίσκονται ταυτόχρονα στο κρίσιμο τμήμα τους.
>
> Όταν μία διεργασία δεν βρίσκεται στο κρίσιμο τμήμα της δεν επιτρέπεται να εμποδίσει καμία διεργασία από την εκτέλεση του δικού της κρίσιμου τμήματος.
>
> Δεν επιτρέπεται μία διεργασία να περιμένει επ'αόριστο να μπει στο κρίσιμο τμήμα της ή όταν χρειστεί να μπουν ταυτόχρονα δύο διεργασίες στο κρίσιμο τμήμα τους να περιμένει η μία την άλλη και να μην εκτελείται καμιά από τις δύο.
>
> Δεν επιτρέπονται παραδοχές σε ό,τι αφορά την ταχύτητα ή το πλήθος των επεξεργαστών.
>
> Δεν επιτρέπεται κάποιες διεργασίες να μονοπωλούν το σύστημα. Πρέπει να υπάρχει κατα κάποιο τρόπο δικειοσύνη στην επιλογή της διεργασίας που θα μπει στο κρίσιμο τμήμα της.

Υπάρχουν διάφοροι τρόποι για να επιτευχθεί ο αμοιβαίος αποκλεισμός. Ένας από αυτούς είναι η απενεργοποίηση των διακοπών. Κάθε φορά που ολοκληρώνεται ο χρόνος που έχει στη διάθεσή της μία εργασία για να εκτελεστεί ενεργοποιείται μία διακοπή ρολογιού και η διεργασία αυτή εφόσσον δεν έχει ολοκληρωθεί η εκτέλεσή της μπαίνει στην ουρά αναμονής για να εκτελεστεί η επόμενη διεργασία της λίστας αναμονής. Θα μπορούσε μία διεργασία όταν μπει στο κρίσιμο τμήμα της να απανεργοποιεί αυτό το σύστημα διακοπών και να μην υπάρξει διακοπή εκτέλεσης και ενεργοποίηση νέας διεργασίας για όσο χρονικό διάστημα η εργασία αυτή εκτελεί το κρίσιμο τμήμα της. Η λύση αυτή έχει το μειονέκτημα πως η χρήση των διακοπών δίνεται στις διεργασίας και αυτό μπορεί να έχει απρόβλεπτες συνέπειες. Όπως για παράδειγμα να διακόψει μια διεργασία τις διακοπές του ρολογιού και να μην τις επαναφέρει.Υπήρξαν διάφορες ιδέες υλοποίησης του αμοιβαίου αποκλεισμού με πιο γνωστή του Ολλανδού μαθηματικού Dekker ο οποίος έδωσε λύση μέσω λογισμικού. Αλλά το 1981 ο Peterson έδωσε μια πιο απλή και αποτελεσματική λύση. Όταν μία διεργασία θέλει να μπει στο κρίσιμο τμήμα της και κατά συνέπεια να χρησιμοποιήσει

τις κοινόχρηστες μεταβλητές καλεί την enter_region με παράμετρο 0 ή 1. Με την κλήση αυτή η διεργασία θα περιμένει έως ότου γίνει εφικτό να μπει στο κρίσιμο τμήμα της. Όταν ολοκληρωθεί το κρίσιμο τμήμα της καλεί την leave_region προκειμένου να δηλώσει πως ολοκλήρωσε το κρίσιμο τμήμα της και μπορεί κάποια άλλη διεργασία που περιμένει να μπει στο δικό της κρίσιμο τμήμα να αποκτήσει πρόσβαση στις κοινόχρηστες μεταβλητές.

Πιο αναλυτικά διακρίνουμε δύο περιπτώσεις:

1. Οι διεργασίες ζητούν σε διαφορετική χρονική στιγμή να μπουν στο κρίσιμο τμήμα τους. Τι θα συμβεί;

   Τη στιγμή που κάποια διεργασία ζητήσει να μπει στο κρίσιμο τμήμα της και δεδομένου ότι υπάρχει αυτή η δυνατότητα καλείται η enter_region και το στοιχείο του πίνακα "interested" που αντιστοιχεί σε αυτή τη διεργασία (έστω η διεργασία 0) γίνεται true και η μεταβλητή turn γίνεται 0. Όταν κάποια άλλη διεργασία ζητήσει να μπει στο κρίσιμο τμήμα της θα περιμένει έως ότου η τιμή του στοιχείου 0 του πίνακα "interested" γίνει false. Αυτό θα συμβεί όταν η πρώτη διεργασία (δηλ. η διεργασία 0) καλέσει την leave_region.

2. Οι διεργασίες ζητούν την ίδια χρονική στιγμή να μπουν στο κρίσιμο τμήμα τους. Τι θα συμβεί;

   Υποθέτουμε πως και οι δύο διεργασίες ζητούν ταυτόχρονα να μπουν στο κρίσιμο τμήμα τους. Η μεταβλητή turn θα πάρει ως τιμή τον αριθμό της μιας διεργασίας και κατόπιν θα διαγραφεί αυτή η τιμή και θα πάρει την τιμή της επόμενης. Όταν φτάσουν και οι δύο στο while η πρώτη δεν θα το εκτελέσει και θα μπεί στο κρίσιμο τμήμα της γιατί η τιμή της turn είναι αυτή της δεύτερης διεργασίας ενώ η δεύτερη θα μπεί στο βρόχο και θα εκτελεί το while έως ότου η πρώτη διεργασία καλέσει την leave_region.

Ακολουθεί ο κώδικας του αμοιβαίου αποκλεισμού.

```
define False 0
define True 1
define N 2
int turn;
int interested[N];

void enter_region(int process);
{
int other;
other=1-process;
interested[process]=TRUE;
turn=process;
while (turn==process&&interested[other]==True);
```

```
}
void leave_region(int process)
{
interested[process]=False;
}
```

Η λύση Peterson εκτελεί αναμονή με απασχόληση γιατί κάθε φορά που μία διεργασία θέλει να μπει στο κρίσιμο τμήμα της και δεν μπορεί εκτελεί ένα βρόχο while ο οποίος δεν κάνει τίποτα. Οι σηματοφορείς έρχονται να λύσουν αυτό το πρόβλημα.

## 13.5 Σηματοφορείς

Ο Dijkstra το 1965 [1] παρουσίασε τις λειτουργίες P και V. Ένας σηματοφορέας ορίζεται σαν μια ακέραιη μεταβλητή. Οι λειτουργίες P και V ορίζονται ως εξής:

P(s): while s<= 0 do skip;

s:=s-1

V(s): s:=s+1

Υποθέτουμε πως δύο διεργασίες δεν θα ζητήσουν να εκτελέσουν ταυτόχρονα τις λειτουργίες P ή V. Αν κάτι τέτοιο προκύψει τότε τυχαία καθορίζεται ποια διεργασία θα έκτελέσει πρώτη τη λειτουργία που ζητά. Όταν έχουμε n διεργασίες το πρόβλημα του κρίσιμου τμήματος λύνεται ως εξής:

mutex:=1;

parbegin

P1: repeat P(mutex); KT1; V(mutex); μη KT1; Until false ||

.

.

.

Pi: repeat P(mutex); KTi; V(mutex); μη KTi; Until false ||

.

.

.

Pn: repeat P(mutex); KTn; V(mutex); μη KTn; Until false ||

parend

όπου mutex είναι η ακέραια μεταβλητή η οποία αποτελεί τον σηματοφορέα. Η τιμή της μεταβλητής – σηματοφορέα είναι ίση με το μηδέν κάθε φορά που κάποια διεργασία μπαίνει στο κρίσιμο τμήμα της. Η τιμή αυτή γίνεται ένα όταν βγαίνει από το κρίσιμο τμήμα. Μία μόνο διεργασία μπορεί να αλλάξει την τιμή του σηματοφορέα και επομένως μόνο μία διεργασία μπαίνει στο κρίσιμο τμήμα της.

## 13.6 Δίκτυα

### 13.6.1 Κατηγορίες Δικτύων

Αν και δεν υπάρχει σαφής τρόπος διαχωρισμού των δικτύων υπάρχουν κάποιες βασικές ιδιότητες που μας επιτρέπουν να διακρίνουμε τα δίκτυα σε κατηγορίες. Οι πιο γνωστές εκ των οποίων είναι η τεχνολογία μετάδοσης, το μέσο διασύνδεσης και η άλλη η γεωγραφική κάλυψη (κλίμακα).

Ως προς την πρώτη ιδιότητα, αυτή της τεχνολογίας μετάδοσης, διακρίνουμε τα δίκτυα σε δίκτυα εκπομπής ( broadcast networks) και δίκτυα από σημείο σε σημείο (point to point).

Δίκτυα εκπομπής: όλες οι μηχανές του δικτύου μοιράζονται το ίδιο κανάλι επικοινωνίας. Η επικοινωνία γίνεται με την αποστολή πακέτου από κάποια μηχανή σε όλες τις υπόλοιπες. Για να παραληφθεί το πακέτο από τη σωστή μηχανή υπάρχει ένα πεδίο διεύθυνσης στο ίδιο το πακέτο στο οποίο καθορίζεται ο παραλήπτης. Ο παραλήπτης επεξεργάζεται αυτό το οποίο λαμβάνει με το πακέτο ενώ οι υπόλοιπες μηχανές απλά το αγνοούν. Υπάρχει επίσης η δυνατότητα να σταλεί ένα πακέτο προς όλους τους παραλήπτες χρησιμοποιώντας κάποια συγκεκριμένη τιμή στο πεδίο διεύθυνσης. Αυτός ο τρόπος ονομάζεται ευρείας μετάδοσης. Τέλος σε κάποια δίκτυα υπάρχει η δυνατότητα να σταλεί ένα πακέτο σε ένα υποσύνολο των μηχανών του δικτύου. Στην περίπτωση αυτή στο πεδίο της διεύθυνσης φαίνεται ότι πρόκειται για πολυδιανομή καθώς και η ομάδα (το υποσύνολο των μηχανών) την οποία αφορά το πακέτο. Όταν το πακέτο απευθύνεται σε συγκεκριμένη ομάδα το επεξεργάζονται όλες οι μηχανές που ανήκουν στην ομάδα.

Δίκτυα από σημείο σε σημείο: Στην κατηγορία αυτή ανήκουν τα δίκτυα στα οποία οι μηχανές συνδέονται με πολλές συνδέσεις μεταξύ τους. Και επειδή πολλές φορές συμβαίνει ένα πακέτο για να φτάσει στον προορισμό του να χρειαστεί να περάσει από πολλές ενδιάμεσες μηχανές ένα σημαντικό θέμα είναι να βρεθεί το καλύτερο δυνατό δρομολόγιο μεταξύ των μηχανών.

Ως προς την δεύτερη ιδιότητα, αυτή της τεχνολογίας μετάδοσης, τα δίκτυα διακρίνονται σε ενσύρματα και ασύρματα.

Ενσύρματο χαρακτηρίζεται ένα δίκτυο υπολογιστών στο οποίο η διασύνδεση των μηχανών για λόγους επικοινωνίας επιτυγχάνεται με τη χρησιμοποίηση κάποιου τύπου καλωδίου. Ασύρματο δίκτυο χαρακτηρίζεται το τηλεπικοινωνιακό δίκτυο το οποίο χρησιμοποιεί, ραδιοκύματα ως φορείς πληροφορίας. Τα δεδομένα μεταφέρονται μέσω ηλεκτρομαγνητικών κυμάτων και η συχνότητα εξαρτάται κάθε φορά από τον ρυθμό μετάδοσης δεδομένων που απαιτείται να υποστηρίζει το δίκτυο. Η ασύρματη επικοινωνία, σε αντίθεση με την ενσύρματη, δεν χρησιμοποιεί ως μέσο μετάδοσης κάποιον τύπο καλωδίου.

Ως προς την γεωγραφική κάλυψη τα δίκτυα χωρίζονται σε τοπικά, μητροπολιτικά, ευρείας περιοχής και διαδίκτυα.

Τα τοπικά δίκτυα τα οποία συνήθως αποκαλούνται LAN (Local Area Networks) είναι δίκτυα που συνδέουν υπολογιστές που βρίσκονται σε κοντινές μεταξύ τους αποστάσεις. Είναι αυτά που συνήθως χρησιμοποιούνται σε εταιρείες, πανεπιστήμια, εργοστάσια.

Τα μητροπολιτικά δίκτυα τα οποία συνήθως αποκαλούνται MAN (Metropolitan Area Network) καλύπτει μεγαλύτερες αποστάσεις όπως για παράδειγμα μία πόλη. Το πιο γνωστό παράδειγμα σε αυτή την κατηγορία είναι το δίκτυο καλωδιακής τηλεόρασης που μπορεί να υπάρχει σε μία πόλη.

Τα δίκτυα ευρείας περιοχής τα οποία συνήθως αποκαλούνται WAN (Wide Area Network) καλύπτουν μεγάλες γεωγραφικές περιοχές όπως μία χώρα ή μία ήπειρο και μπορούν να συνδέσουν ακόμη και περισσότερα από ένα τοπικά δίκτυα καθώς και ομάδες τοπικών δικτύων. Τα διαδίκτυα είναι ένα σύνολο διασυνδεδεμένων δικτύων. Είναι δίκτυα ευρείας περιοχής και καλύπτουν γεωγραφικές περιοχές μιάς ή περισσοτέρων ηπείρων διασυνδέοντας επιμέρους δίκτυα. Σε ένα διαδίκτυο μπορεί να συνυπάρχουν διασυνδεδεμένοι υπολογιστές και δίκτυα υπολογιστών που χρησιμοποιούν διαφορετικές τεχνολογίες και διαφορετικά λειτουργικά συστήματα. Το ”Διαδίκτυο” (Internet) είναι το μεγαλύτερο τέτοιου είδους δίκτυο.

### 13.6.2 Ιεραρχίες Πρωτοκόλλων

Τα δίκτυα οργανώνονται σε μια στοίβα επιπέδων στην οποία το ένα επίπεδο χτίζεται πάνω στο προηγούμενο. Κάθε επίπεδο προσφέρει υπηρεσίες στο ανώτερό του επίπεδο αποκρύπτοντας μέρος της πληροφορίας που αφορά στις λεπτομέρειες υλοποίησης της των υπηρεσιών που προσφέρει το κατώτερο στο ανώτερο επίπεδο. Ο τρόπος που πραγματοποιείται μία συνομιλία μεταξύ δύο μηχανών στο επίπεδο n ονομάζεται πρωτόκολλο επιπέδου n. Το πρωτόκολλο ουσιαστικά καθορίζει τους κανόνες με τους οποίους γίνεται η επικοινωνία. Δύο ευρέως γνωστές αρχιτεκτονικές δικτύων είναι το μοντέλο OSI και το μοντέλο TCP/IP.
Το μοντέλο OSI (Open Systems Interconnection) έχει 7 επίπεδα ιεραρχίας:

**Φυσικό επίπεδο**
Είναι το κατώτερο επίπεδο και ασχολείται με τη μετάδοση των bits στο κανάλι επικοινωνίας.

**Επίπεδο Διασύνδεσης Δεδομένων**
Ελέγχει τη ροή των δεδομένων και την αξιόπιστη μετάβαση των πακέτων. Παίρνει τα δεδομένα από το φυσικό επίπεδο και να τα προωθεί στο επίπεδο δικτύου αφού ανιχνεύσει και διορθώσει σφάλματα μετάδοσης.

**Επίπεδο Δικτύου**
Ελέγχει τη λειτουργία του υποδικτύου. Αν πολλά πακέτα βρίσκονται στο διαδίκτυο την ίδια χρονική στιγμή παρεμποδίζει το ένα το άλλο. Το επίπεδο δικτύου αναλαμβάνει τον έλεγχο της συμφόρησης. Επίσης αν το πακέτο πηγαίνει από ένα δίκτυο σε ένα άλλο μπορούν να υπάρξουν προβλήματα. Για παράδειγμα να διαφέρουν στην διευθυνσιοδότηση,

να μη γίνει αποδεκτό το πακέτο από το δίκτυο προορισμού λόγω μεγέθους, να διαφέρουν τα πρωτόκολλα.

**Επίπεδο μεταφοράς**
Δέχεται δεδομένα από το επίπεδο συνδιάλεξης (το ανώτερό του επίπεδο), τα διασπά αν χρειάζεται και τα μεταβιβάζει στο επίπεδο δικτύου.

**Επίπεδο Συνδιάλεξης**
Πραγματοποιεί τις κατάλληλες λειτουργίες ώστε διαφορετικές μηχανές να συνδιαλέγονται χωρίς προβλήματα μεταξύ τους μεσω του δικτύου. Σε περίπτωση προβλήματος στη σύνδεση, φροντίζει για την αποκατάστασή του.

**Επίπεδο Παρουσίασης**
Ασχολείται με τις κατάλληλες ενέργειες ώστε τα δεδομένα που μεταδίδονται να αναπαριστώνται με συμβατό τρόπο μεταξύ των διαφόρων υπολογιστών. Αυτό επιτυγχάνεται για παράδειγμα με τον ορισμό δομών δεδομένων με αφαιρετικό τρόπο.

**Επίπεδο Εφαρμογής**
Αποτελεί το ανώτερο επίπεδο. Περιέχει μια ποικιλία πρωτοκόλλων με εφαρμογές που εξυπηρετούν τον χρήστη. Τέτοια πρωτόκολλα είναι το πρωτόκολλο μεταφοράς υπερκειμένου, το ηλεκτρονικό ταχυδρομείο κ.λπ.

## 13.7 Διαδραστικό Υλικό – Σύνδεσμοι

- https://en.wikipedia.org/wiki/Operating_system
- https://en.wikipedia.org/wiki/Computer_network
- https://en.wikipedia.org/wiki/Process_management_%28computing%29
- https://el.wikipedia.org/wiki/%CE%9C%CE%BF%CE%BD%CF%84%CE%AD%CE%BB%CE%BF_%CE%B1%CE%BD%CE%B1%CF%86%CE%BF%CF%81%CE%AC%CF%82_OSI
- https://en.wikipedia.org/wiki/Internet_protocol_suite

## 13.8 Ασκήσεις

1. Τι είναι κρίσιμο τμήμα μιας διεργασίας και πως υλοποιείται ο αμοιβαίος αποκλεισμός.
2. Ποιά η διαφορά προγράμματος και διεργασίας και σε ποιές καταστάσεις μπορεί να βρίσκεται μία διεργασία;
3. Τι είναι πολυπρογραμματισμός;

4. Υποθέτουμε ότι 3 διεργασίες (δ1,δ2,δ3) με χρόνο εκτέλεσης 10, 12, 16 microsecs θέλουν να χρησιμοποιήσουν την ΚΜΕ. Η σειρά άφιξης είναι πρώτα η δ1 μετά η δ2 και τελευταία η δ3. Σε πόσο χρόνο θα ολοκληρωθεί κάθε μία από τις τρεις αν η ΚΜΕ εκτελεί για 4 microsecs μια διεργασία και στη συνέχεια εκτελεί την επόμενη. Αρχικά υποθέτουμε η σειρά άφιξης είναι και η σειρά εισόδου στην ΚΜΕ και κάθε φορά που τελειώνει ο χρόνος για μία διεργασία και ολοκληρωθεί μπαίνει τελευταία στην ουρά αναμονής. Αν υποθέσουμε πως η ΚΜΕ εκτελεί κάθε φορά την μικρότερη σε διάρκεια εκτέλεσης διεργασία και ότι όλες οι διεργασίες βρίσκονται ταυτόχρονα στην ουρά αναμονής πόσος χρόνος θα απαιτηθεί για κάθεμιά από τις δ1, δ2, δ3;

5. Υποθέτουμε πως έχουμε έναν επεξεργαστή και εκτελεί προγράμματα που κατά μέσο όρο απαιτούν 2 secs πρόσβαση στην ΚΜΕ και 10 secs στις μονάδες εισόδου εξόδου. Ποιό είναι το ποσοστό του χρόνου που η ΚΜΕ μένει ανενεργή;

# Βιβλιογραφία


[1] Al Aho and Jeff Ullman: ”Foundations of Computer Science”, W.H.Freeman, 1992, free online.

[2] E. Dijkstra “Concurrent Programming, Mutual Exclusion” 1965.

[3] Andrew S. Tanenbaum “Modern Operating Systems, third edition”

[4] Γ.Κ. Παπακωνσταντίνου, Ν.Α Μπιλάλης, Π.Δ. Τσανάκας “Λετουργικά Συστήματα”

[5] Andrew S. Tanenbaum “ Computer Networks, fourth edtion”

[6] Behrouz Forouzan “Εισαγωγή στην Επιστήμη των Υπολογιστών” τρίτη έκδοση 2014

# Κεφάλαιο 14

# Βάσεις Δεδομένων

## 14.1 Εισαγωγή

Το κεφάλαιο των Βάσεων Δεδομένων Η αποθήκευση και διαχείριση δεδομένων σε ένα υπολογιστικό σύστημα είναι μια από τις πιο σημαντικές εφαρμογές των υπολογιστών. Η οργανωμένη αυτή συλλογή δεδομένων καλείται βάση δεδομένων. Ειδικότερα, στην επιστήμη της πληροφορικής με τον όρο βάσεις δεδομένων αναφερόμαστε σε οργανωμένες, διακριτές συλλογές σχετιζόμενων αποθηκευμένων δεδομένων. Με τον όρο Σύστημα Διαχείρισης βάσεων δεδομένων αναφερόμαστε στο λογισμικό που χειρίζεται τέτοιες συλλογές. Ο τρόπος σχεδιασμού και ιεράρχησης των δεδομένων θα πρέπει να αποβλέπει στην γρήγορη άντληση πληροφορίας καθώς και στην εύκολη ανανέωση των δει σύνται όλες οι μαθηματικές έννοιες.

Με τον όρο μοντέλο δεδομένων αναφερόμαστε στο λογικό σχεδιασμό των δεδομένων καθώς και στις μεταξύ τους σχέσεις. Τα πιο διαδεδομένα μοντέλα είναι το ιεραρχικό, το διαδικτυακό και το σχεσιακό. Στο ιεραρχικό μοντέλο τα δεδομένα αποθηκεύονται σε μορφή δένδρου με τη ρίζα να αποτελεί την κορυφή της ιεραρχίας και τα παιδιά κάθε κόμβου να μην σχετίζονται μεταξύ τους. Στο δικτυακό μοντέλο οι οντότητες είναι οργανωμένες σε ένα γράφο και η προσπέλασή τους είναι εφικτή μέσω κάποιων διαδρομών. Τα δύο αυτά πρώτα μοντέλα είναι παρωχημένα και δεν θα αναλυθούν περαιτέρω. Το σχεσιακό μοντέλο το οποίο προτάθηκε για πρώτη φορά το 1970 από τον Edgar F. Codd οργανώνει τα δεδομένα σε διδιάστατους πίνακες οι οποίοι αποτελούν τον απλούστερο και πλέον ευέλικτο τρόπο οργάνωσης των δεδομένων. Οι πίνακες ονομάζονται σχέσεις και το σύνολο των σχέσεων αποτελεί το σχεσιακό σύστημα διαχείρισης δεδομένων (ΣΣΔΔ). Κάθε σχέση στο ΣΣΔΔ έχει όνομα, πεδία (ή ιδιότητες ή χαρακτηριστικά) και πλειάδες (εγγραφές).

Κάθε σχέση (πίνακας) πρέπει να έχει ένα όνομα και το οποίο θα είναι μοναδικό στη συγκεκριμένη Βάση Δεδομένων. Τα χαρακτηριστικά (οι ιδιότητες) κάθε σχέσης αποτελούν ουσιαστικά τις επικεφαλίδες των στηλών κάθε πίνακα. Τα ονόματα των χαρακτηριστικών είναι επίσης μοναδικά στο ΣΣΔΔ. Ως βαθμός μιας σχέσης ορίζεται το πλήθος των χαρακτηριστικών της.

Οι γραμμές μιας σχέσης αποτελούν τις πλειάδες. Κάθε πλειάδα είναι ένα σύνολο τιμών των χαρακτηριστικών της σχέσης. Ως πληθικότητα μιας σχέσεις ορίζεται το πλήθος των πλειάδων

της. Η πληθικότητα μιας σχέσης αλλάζει δυναμκά καθώς προστίθενται ή διαγράφονται πλειάδες. Υπάρχει επίσης η δυνατότητα να αλλαχθούν οι τιμές κάποιων χαρακτηριστικών μιας πλειάδας.

Ο όρος κλειδί για έναν πίνακα αναφέρεται στο σύνολο των χαρακτηριστικών των οποίων οι τιμές καθορίζουν μία και μόνο συγκεκριμένη πλειάδα. Καμία άλλη πλειάδα δεν έχει τις ίδιες τιμές στα αντίστοιχα χαρακτηριστικά. Να σημειωθεί πως δεν επιτρέπεται δύο πλειάδες μιας σχέσης να έχουν ακριβώς τις ίδιες τιμές σε όλα τα χαρακτηριστικά τους. Θα πρέπει να διαφέρουν σε ένα τουλάχιστον χαρακτηριστικό.

Οι δείκτες είναι δομές δεδομένων που μας επιτρέπουν να αντλήσουμε ή να τροποποιήσουμε αποτελεσματικά πληροφορία που είναι αποθηκευμένη σε έναν πίνακα.

Με τον όρο σχήμα μιας σχέσης αναφερόμαστε στο σύνολο των ονομάτων των χαρακτηριστικών της σχέσης (είναι ουσιαστικά το σύνολο των ονομάτων των πεδίων του πίνακα). Η σειρά εμφάνισης των χαρακτηριστικών της σχέσης είναι επουσιώδης αλλά πρέπει να γνωρίζουμε ακριβώς σε ποιά στήλη αποθηκεύεται η τιμή κάθε χαρακτηριστικού έτσι ώστε να γράφονται σωστά οι πλειάδες και να γίνεται σωστά η άντληση πληροφορίας. Συχνά και για λόγους ευκολίας στη χρήση το σχήμα είναι και το όνομα μιας σχέσης. Για παράδειγμα αν μία σχέση περιέχει τα χαρακτηριστικά: Όνομα υπαλλήλου, Τμήμα, Μισθός το όνομα της σχέσης θα μπορούσε να είναι Όνομα-Τμήμα-Μισθός ή συντομευμένα (ΟΤΜ). Αναπαράσταση σχέσεων.

Μία σχέση μπορεί να αναπαρασταθεί με μία δομή δεδομένων. Ένας πίνακας για παράδειγμα θα μπορούσε να είναι μία δομή με τα πεδία της δομής να αντιστοιχούν στα πεδία του πίνακα. Για παράδειγμα η σχέση ΟΤΜ θα μπορούσε να είναι:

```
struct OTM {
char Onoma[20];
char Tmhma [10];
int Misthos;
}
```

Και κατά συνέπεια μία σχέση θα μπορούσε να αναπαρασταθεί από έναν πίνακα με στοιχεία δομές του τύπου ΟΤΜ ή από μια συνδεδεμένη λίστα με στοιχεία δομές του τύπου ΟΤΜ. Βάση δεδομένων είναι μία συλλογή τέτοιων σχέσεων. Επομένως όταν θέλουμε να υλοποιήσουμε μία βάση δεδομένων για να αποθηκεύσουμε δεδομένα το πρώτο πράγμα που θα πρέπει να αποφασίσουμε είναι το πως θα πρέπει να καταχωρηθεί η πληροφορία σε πίνακες. Ο σχεδιασμός θα πρέπει να γίνεται με τρόπο που να επιτρέπει την άντληση πληροφορίας αποτελεσματικά. Αυτό σημαίνει πως πρέπει να γίνεται γρήγορα χωρίς να απαιτεί χρήση πολύπλοκων προγραμματιστικών τεχνικών. Ακολουθεί ένα παράδειγμα μιας βάσης δεδομένων που περιέχει πληροφορίες για τους φοιτητές κάποιου πανεπιστημίου. Οι σχέσεις που περιέχονται είναι οι ακόλουθες:

1. {Κωδικός_φοιτητή, Ονοματεπώνυμο, Διεύθυνση, Τηλέφωνο}
2. {Κωδικός_φοιτητή, Μάθημα, Βαθμός}

3. {Μάθημα, Ημέρα, Ώρα}
4. {Μάθημα, Αμφιθέατρο}

## 14.2 Λειτουργίες που εκτελούνται σε μία βάση δεδομένων

Οι πιο γνωστές λειτουργίες είναι η εισαγωγή, η διαγραφή, η ενημέρωση, η επιλογή, η προβολή, η σύνδεση, η ένωση, η τομή και η διαφορά. Η γλώσσα που ευρέως χρησιμοποιείται για τις σχεσιακές βάσεις δεδομένων είναι η SQL (Structured Query Language) η οποία τυποποιήθηκε από το Αμερικάνικο Ινστιτούτο Εθνικών Προτύπων (ANSI) το 1986 και τον Διεθνή Οργανισμό Προτυποποίησης (ISO)το 1987. Η SQL αναπτύχθηκε αρχικά στην IBM από τον Donald D. Chamberlin και τον Raymond F. Boyce στις αρχές του 1970.

- Εισαγωγή

  Η πιο γνωστή λειτουργία είναι αυτή της εισαγωγής. Η λειτουργία αυτή προσθέτει μά πλειάδα σε μία σχέση αν η πλειάδα αυτή δεν υπάρχει ήδη. Η μορφή της είναι η εξής:

  > insert into όνομα σχέσης values (τιμές χαρακτηριστικών χωρισμένες με κόμματα)

  για παράδειγμα:

  > insert into OTM values (“Papadopoulos”, “Logisthrio”, 1200)

- Διαγραφή

  Η λειτουργία αυτή διαγράφει τις πλειάδες που έχουν συγκεκριμένες τιμές στα χαρακτηριστικά τους. Η μορφή της είναι η εξής:

  > delete from όνομα σχέσης
  > where (τιμές χαρακτηριστικών χωρισμένες με κόμματα)

  Το κριτήριο διαγραφής εδώ είναι το τμήμα στο οποίο εργάζεται ο υπάλληλος να είναι το λογιστήριο.

- Ενημέρωση

  Η λειτουργία αυτή αλλάζει τις τιμές κάποιων χαρακτηριστικών κάποιων πλειάδων που ικανοποιούν μια συνθήκη. Η μορφή της είναι η εξής:

  > update Όνομα Σχέσης
  > set ανάθεση τιμών σε χαρακτηριστικά χωρισμένα με κόμματα
  > where κριτήρια

  για παράδειγμα:

```
update OTM
set misthos = 1500
where tmhma=”logisthrio”
```

- Επιλογή

  Η λειτουργία αυτή επιλέγει κάποιες πλειάδες από την αρχική σχέση που ικανοποιούν κάποια κριτήρια.

  Η μορφή της είναι η εξής:

  ```
  select *
  from όνομα σχέσης
  where κριτήρια
  ```

  για παράδειγμα:

  ```
  select *
  from OTM
  where misthos = 1000
  ```

  Η επιλογή αποτελεί μία νέα σχέση με πλειάδες υποσύνολο των πλειάδων της αρχικής σχέσης. Το * δηλώνει πως επιλέγονται όλα τα χαρακτηριστικά της σχέσης.

- Προβολή

  Η προβολή δημιουργεί μία νέα σχέση επίσης με την εξής διαφορά. Η νέα σχέση έχει το ίδιο πλήθος πλειάδων αλλά λιγότερα χαρακτηριστικά. Η μορφή της είναι η εξής:

  ```
  select λίστα χαρακτηριστικών χωρισμένα με κόμματα
  from όνομα σχέσης
  ```

  για παράδειγμα

  ```
  select Onoma, Misthos
  from OTM
  ```

- Σύνδεση σχέσεων με (τουλάχιστον ένα) κοινό χαρακτηριστικό Οι λειτουργίες που είδαμε έως τώρα εφαρμόζονται σε μία σχέση. Στη συνέχεια θα δούμε και λειτουργίες που εφαρμόζονται σε περισσότερες σχέσεις. Ακολουθεί ένα παράδειγμα βάσης δεδομένων με 2 σχέσεις. Η πρώτη έχει πληροφορίες με τα προσωπικά στοιχεία των φοιτητών κάποιας σχολής και η δεύτερη περιέχει τους μέσους όρους βαθμολογίας των φοιτητών.

  ```
  Σχέση: Students
  Χαρακτηριστικά: ID Name Address Phone
  ```

και

Σχέση: Vathmos
Χαρακτηριστικά: ID Mesos_Oros
Σύνδεση σχέσεων με (τουλάχιστον ένα) κοινό χαρακτηριστικό

Η λειτουργία join συνδέει δύο σχέσεις που έχουν κάποια κοινά χαρακτηριστικά. Η join δημιουργεί μία νέα σχέση με χαρακτηριστικά υποσύνολο των χαρακτηριστικών των δύο σχέσεων. Η μορφή της είναι η εξής:

```
select λίστα χαρακτηριστικών χωρισμένα με κόμματα
from όνομα σχέσης
where κριτήρια
```

για παράδειγμα

```
select ID, Name, Mesos_Oros
from Students, Vathmos
where Students.ID = Vathmos.ID
```

Η νέα σχέση που θα δημιουργηθεί θα περιέχει τα χαρακτηριστικά ID, Name, Mesos_Oros και θα ενώνει δύο πλειάδες (μία από κάθε σχέση)αν έχουν την ίδια τιμή στο χαρακτηριστικό ID. Σημειώνεται εδώ πως η λειτουργία join είναι περίπλοκη και εδώ παρουσιάζεται με την πιο απλή μορφή της.

- Ένωση σχέσεων με όλα τα χαρακτηριστικά τους κοινά

  Η λειτουργία union ενώνει δύο σχέσεις που έχουν ακριβώς τα ίδια χαρακτηριστικά. Για παράδειγμα αν έχω τη σχέση studentsA με τα προσωπικά στοιχεία των φοιτητών της σχολής Α και τη σχέση studentsB με τα προσωπικά στοιχεία των φοιτητών της σχολής Β η νέα σχέση θα περιέχει τα προσωπικά στοιχεία των φοιτητών που είναι εγγεγραμμένοι είτε στη μία σχολή είτε στην άλλη είτε και στις δύο σχολές. Ουσιαστικά η union κάνει ότι γίνεται και στην ένωση δύο συνόλων. Το σύνολο που προκύπτει περιέχει κοινά και μη κοινά στοιχεία των δύο συνόλων.

  Η μορφή της είναι η εξής:

```
select *
from όνομα πρώτης σχέσης
union
select *
from όνομα δεύτερης σχέσης
```

για παράδειγμα

```
select *
from Students1
union
select *
from Students2
```

Ο αστερίσκος δηλώνει όλα τα χαρακτηριστικά της σχέσης.

Οι σχέσεις Students1 και Students2 έχουν τα ίδια ακριβώς χαρακτηριστικά και η νέα σχέση που θα προκύψει από την εφαρμογή της λειτουργίας union στις δύο σχέσεις θα περιέχει όλους τους φοιτητές των δύο σχολών με τα προσωπικά τους στοιχεία.

- Τομή σχέσεων με όλα τα χαρακτηριστικά τους κοινά

  Η λειτουργία intersection προϋποθέτει όπως και η ένωση κοινά χαρακτηριστικά. Αυτό που επιτυγχάνεται με τη λειτουργία αυτή είναι η δημιουργία μιας νέας σχέσης με πλειάδες τις κοινές πλειάδες των δύο σχέσεων. Η μορφή της είναι η εξής:

```
select *
from όνομα πρώτης σχέσης
intersection
select *
from όνομα δεύτερης σχέσης
```

  για παράδειγμα

```
select *
from Students1
intersection
select *
from Students2
```

  Ο αστερίσκος δηλώνει όλα τα χαρακτηριστικά της σχέσης.

  Οι σχέσεις Students1 και Students2 έχουν τα ίδια ακριβώς χαρακτηριστικά και η νέα σχέση που θα προκύψει από την εφαρμογή της λειτουργίας intersection στις δύο σχέσεις θα περιέχει μόνο τους κοινούς φοιτητές των δύο σχολών με τα προσωπικά τους στοιχεία. Ουσιαστικά η intersection κάνει ότι γίνεται και στην τομή δύο συνόλων. Το σύνολο που προκύπτει περιέχει μόνο τα κοινά στοιχεία των δύο συνόλων.

- Διαφορά

  Η διαφορά εφαρμόζεται επίσης σε δύο σχέσεις που έχουν τα ίδια ακριβώς χαρακτηριστικά. Με τη λειτουργία αυτή επίσης δημιουργείται μία νέα σχέση από δύο αρχικές η οποία περιέχει όλες τις πλειάδες της πρώτης σχέσεις που δεν υπάρχουν στη δεύτερη. Η μορφή της είναι η εξής:

```
select *
from όνομα πρώτης σχέσης
minus
select *
from όνομα δεύτερης σχέσης
```

για παράδειγμα

```
select *
from Students1
minus
select *
from Students2
```

Οι σχέσεις Students1 και Students2 όπως προαναφέρθηκε έχουν τα ίδια ακριβώς χαρακτηριστικά και η νέα σχέση που θα προκύψει από την εφαρμογή της λειτουργίας minus στις δύο σχέσεις θα περιέχει μόνο τις πλειάδες με τους φοιτητές της πρώτης σχολής που δεν είναι εγγεγραμμένοι και στη δεύτερη σχολή. Ουσιαστικά η minus κάνει ότι γίνεται και στην διαφορά δύο συνόλων. Το σύνολο που προκύπτει περιέχει μόνο τα στοιχεία του πρώτου συνόλου που δεν περιέχονται και στο δεύτερο.

Η γλώσσα SQL παρέχει τη δυνατότητα συνδυασμού των παραπάνω εντολών έτσι ώστε ακόμη και αν τα ερωτήματα στη Βάση Δεδομένων είναι πολύ σύνθετα να μπορούμε να αντλούμε την πληροφορία που μας χρειάζεται.

## 14.3 Σχεδιασμός Βάσης Δεδομένων

Ένα σημαντικό θέμα στο σχεδιασμό μιας βάσης δεδομένων είναι το πως θα επιλέξουμε το σωστό σχήμα. Για παράδειγμα σε πόσες και ποιες σχέσεις θα αποθηκεύεται η πληροφορία. Ποια χαρακτηριστικά θα έχουν οι σχέσεις αυτές. Ποιά συσχέτιση πρέπει να υπάρχει μεταξύ των σχέσεων. Γιατί να χρησιμοποιήσουμε 2 σχέσεις με λίγα χαρακτηριστικά αντί μιας με πολλά χαρακτηριστικά ή αντίστροφα. Ας δούμε το παρακάτω παράδειγμα: Θέλουμε να αποθηκεύσουμε πληροφορίες για φοιτητές και μαθήματα. Θα μπορούσαμε να είχαμε μία σχέση με τα χαρακτηριστικά:

Mathima
Code_foiththth
Vathmos
Proapaitoumena
Hmera
Wra
Amfitheatro

Θα μπορούσαμε ωστόσο να έχουμε περισσότερες σχέσεις με λιγότερα χαρακτηριστικά που θα περιέχουν όλη την πληροφορία σε συμπιεσμένη μορφή. Όπως για παράδειγμα οι σχέσεις:

*Grades* με χαρακτηριστικά

Mathima
Code_foiththh
Vathmos

*Precourse* με χαρακτηριστικά

Mathima
Proapaitoumena

*Programma* με χαρακτηριστικά

Mathima
Hmera
Wra
Amfitheatro

Σε αυτή την περίπτωση δεν θα χρειαστεί να επαναλάβουμε σε κάθε φοιτητή που έχει εγγραφεί σε κάποιο μάθημα ποιά είναι τα προαπαιτούμενα καθώς και την ώρα και αίθουσα διεξαγωγής του μαθήματος. Επίσης τα προαπαιτούμενα ενός μαθήματος είναι ανεξάρτητα από την ώρα και το αμφιθέατρο επομένως δεν χρειάζεται σε κάθε πλειάδα που περιέχει ένα μάθημα και ένα από τα προαπαιτούμενα να υπάρχει η ώρα και το αμφιθέατρο. Θα είχαμε έτσι πολλές πλειάδες με ίδια τιμή στο μάθημα, ώρα και αμφιθέατρο και θα διέφεραν μόνο στην τιμή του χαρακτηριστικού προαπαιτούμενο. Συνεπώς αν κάποιο μάθημα είχε 5 προαπαιτούμενα θα είχε 5 πλειάδες με ίδια τιμή σε όλα τα υπόλοιπα χαρακτηριστικά.

Αρχικά ο σχεδιασμός μιας βάσης δεδομέων απαιτεί τη μελέτη και την ανάλυση των απαιτήσεων. Στη συνέχεια σχεδιάζεται το μοντέλο οντοτήτων – συσχετίσεων που καθορίζει το σχήμα της βάσης και τέλος υλοποιείται η βάση δεδομένων. Το μοντέλο οντοτήτων-συσχετίσεων περιλαμβάνει τις οντότητες, τις συσχετίσεις καθώς και τα χαρακτηριστικά τόσο των οντοτήτων όσο και των συσχετίσεων.

Ως οντότητες ορίζουμε ανεξάρτητα αντικείμενα του προβλήματος και ως συσχετίσεις τις μεταξύ τους σχέσεις. Για παράδειγμα αν θέλαμε να σχεδιάσουμε το φοιτητολόγιο μιας σχολής ενός πανεπιστημίου θα μπορούσαμε να ορίσουμε 3 οντότητες. Η πρώτη θα περιλαμβάνει προσωπικά στοιχεία του φοιτητή, η δεύτερη στοιχεία του διδάσκοντος και η τρίτη στοιχεία των μαθημάτων. Όσον αφορά τις συσχετίσεις θα είχαμε τη συσχέτιση φοιτητή-μαθήματος και τη συσχέτιση διδάσκων-μάθημα. Τα χαρακτηριστικά της οντότητας φοιτητής είναι κωδικός (foit_code), όνομα (foit_name), διεύθυνση (address), τηλέφωνο(phone). Εκείνα της οντότητας διδάσκων θα μπορούσαν να είναι κωδικός (prof_code), όνομα (prof_name), αριθμός γραφείου (room). Και τέλος τα χαρακτηριστικά της οντότητας μάθημα θα ήταν κωδικός (course_code), όνομα(coursei_name), τομέας (division).

Ως συσχετίσεις ορίζουμε τη συχέτιση δύο ή περισσοτέρων οντοτήτων και έχουν και αυτές επίσης

χαρακτηριστικά. Η συσχέτιση φοιτητή-μαθήματος για παράδειγμα μπορεί να έχει τα χαρακτηριστικά foit_code και course_code που δηλώνει πως ο φοτητής με τον κωδικό foit_code έχει δηλώσει το μάθημα με κωδικό course_code και η συσχέτιση διδάσκων-μάθημα μπορεί να έχει τα χαρακτηριστικά prof_code και course_code που δηλώνει πως ο καθηγητής με τον κωδικό profi_code διδάσκει το μάθημα με κωδικό course_code. Συσχετίσεις σε περισσότερες από δύο οντότητες είναι πιο περίπλοκες. Ως βαθμός μιας συσχέτισης ορίζεται το πλήθος των οντοτήτων που συνδέονται με αυτή.

Κλειδί μιας οντότητας ονομάζεται η τιμή ενός χαρακτηριστικού που έχει μία μοναδική τιμή στην οντότητα. Για παράδειγμα ο κωδικός του φοιτητή. Δεν μπορεί δύο ή περισσότεροι φοιτητές να έχουν τον ίδιο κωδικό. Σε κάποιες περιπτώσεις το κλειδί είναι σύνθετο δηλαδή περιλαμβάνει περισσότερα από ένα χαρακτηριστικά. Για παράδειγμα μία πολυεθνική εταιρεία κρατά στοιχεία για τους υπαλλήλους της σε κάθε χώρα. Οι κωδικοί είναι μοναδικοί σε κάθε χώρα αλλά ο ίδιος κωδικός μπορεί να δωθεί σε υπάλληλο που εργάζεται στην Ελλάδα και στην Γερμανία. Σε αυτή την περίπτωση κλειδί είναι τα χαρακτηριστικά κωδικός υπαλλήλου και κωδικός χώρας. Ισχύει επίσης και το αντίστροφο. Μία οντότητα να έχει περισσότερα από ένα κλειδιά. Για παράδειγμα σε μία βάση δεδομένων υπάρχει η οντότητα αυτοκίνητο και έχει μεταξύ των άλλων χαρακτηριστικών τον αριθμό κυκλοφορίας που είναι μοναδικός καθώς και τον αριθμό πλαισίου που επίσης είναι μοναδικός.

## 14.4 Κανονικές Μορφές

Κανονικοποίηση ονομάζεται η διαδικασία κατά την οποία ένα σχεσιακό σχήμα διασπάται ώστε να μετασχηματιστεί σε ένα σχήμα με δομή που θα ικανοποιεί ορισμένες απαιτήσεις. Υπάρχει μια ιεραρχία κανονικών μορφών η πρώτη, η δεύτερη, η Τρίτη, BCNF κ.λπ.

**1η κανονική μορφή** Έστω πως έχουμε μία σχέση με χαρακτηριστικά code_foit (κωδικός φοιτητή), όνομα (name), course (μάθημα). Ένας φοιτητής μπορεί να δηλώσει πολλά μαθήματα και σε αυτή την περίπτωση το χαρακτηριστικό αυτό έχει πολλές τιμές. Το ερώτημα που τίθεται εδώ είναι το κατα πόσο είναι εύκολο να αντλήσει κάποιος πληροφορίες της μορφής "ποιοί φοιτητές έχουν δηλώσει το μάθημα Βάσεις Δεδομένων;". Επίσης είναι εύκολο να προστεθεί ένα νέο μάθημα σε κάποιο φοιτητή ή να διαγράψει κάποιο μάθημα από κάποιον φοιτητή (λόγω απαλλαγής για παράδειγμα) ή απαιτούνται πολύπλοκοι χειρισμοί; Μία πιθανή λύση για να αποφύγουμε να δίνουμε πολλές τιμές σε ένα χαρακτηριστικό θα ήταν να γράψουμε όλα τα μαθήματα σαν νέα χαρακτηριστικά και να συμπληρώνουμε με ναι ή όχι ανάλογα με το αν ο φοιτητής δήλωσε το μάθημα ή όχι. Αυτό όμως θα απαιτούσε τόσα χαρακτηριστικά για τα πεδία όσα και τα μαθήματα της σχολής και θα μεγάλωνε υπερβολικά η σχέση. Αυτό που είναι αποδοτικότερο είναι να προσθέσουμε πολλές πλειάδες στη σχέση η οποία θα εξακολουθεί να έχει τα τρία αρχικά χαρακτηριστικά. Έτσι κάθε φοιτητής θα προστίθεται μία φορά για κάθε μάθημα που δηλώνει. Αυτή είναι η πρώτη κανονική μορφή και αν η βάση μας περιέχει σχέσεις που έχουν πολλές τιμές στο ίδιο χαρακτηριστικό θα πρέπει να κανονικοποιηθεί προσθέτοντας περισσότερες γραμμές. Ουσιαστικά προσθέτουμε μία γραμμή για κάθε τιμή του χαρακτηριστικού που έχει πολλές τιμές.

**2η κανονική μορφή** Στη δεύτερη κανονική μορφή θέλουμε σε κάθε σχέση να υπάρχει ένα κλειδί απλό ή σύνθετο από το οποίο να εξαρτώνται οι τιμές όλων των υπολοίπων χαρακτηριστικών. Ας υποθέσουμε ότι το πρωτεύον κλειδί μιας σχέσης αποτελείται από δύο ή περισσότερα χαρακτηριστικά. Στην περίπτωση αυτή θα πρέπει κάθε άλλο χαρακτηριστικό να εξαρτάται από το σύνολο των χαρακτηριστικών του πρωτεύοντος κλειδιού. Έστω για παράδειγμα η σχέση «υπάλληλος» με τα χαρακτηριστικά κωδικός υπαλλήλου(code), όνομα υπαλλήλου (name), αριθμός προγράμματος (project_ID) και ώρες που απασχολήθηκε στο πρόγραμμα. Το πρωτεύον κλειδί είναι ο υπάλληλος και ο αριθμός προγράμματος. Στην περίπτωση αυτή το όνομα του υπαλλήλου εξαρτάται μόνο από τον κωδικό του ενώ αντιθέτως οι ώρες απασχόλησης εξαρτώνται από τον κωδικό υπαλλήλου και από το πρόγραμμα στο οποίο συμμετέχει. Στην περίπτωση αυτή για να εγγράψουμε κάποιο υπάλληλο πρέπει να έχουμε πληροφορίες για τα προγράμματα στα οποία απασχολήθηκε καθώς και τις ώρες. Αν δεν έχουμε τις πληροφορίες αυτές και θέλουμε να έχουμε τα στοιχεία κάποιου υπαλλήλου θα έπρεπε να δημιουργήσουμε δύο σχέσεις. Η μία να έχει τον κωδικό και το όνομα του υπαλλήλου και η δεύτερη άλλη να έχει τον κωδικό του υπαλλήλου τον αριθμό προγράμματος καθώς και τις ώρες απασχόλησής του σε κάθε πρόγραμμα.

## 14.5 Κατανεμημένες Βάσεις Δεδομένων

Εκτός από το σχεσιακό μοντέλο δεδομένων υπάρχουν και άλλα γνωστά μοντέλα που χρησιμοποιούνται σήμερα. Ένα από αυτά είναι οι κατανεμημένες βάσεις δεδομένων. Στο μοντέλο αυτό τμήματα της Βάσης Δεδομένων αποθηκεύονται σε πολλούς υπολογιστές που βρίσκονται σε διαφορετικές τοποθεσίες και συνδέονται μεταξύ τους με το διαδίκτυο ή κάποιο άλλο δίκτυο. Οι χρήστες έχουν πρόσβαση στο τοπικό τμήμα της βάσης δεδομένων στην τοποθεσία τους και δεν επηρεάζονται από τις εργασίες άλλων χρηστών. Ένα κεντρικό κατανεμημένο σύστημα διαχείρισης βάσεων δεδομένων βλέπει τη βάση δεδομένων σαν να ήταν ολόκληρη αποθηκευμένη στον ίδιο υπολογιστή. Οι εισαγωγές, διαγραφές και ενημερώσεις των δεδομένων γίνονται με τέτοιο τρόπο ώστε να εξασφαλίζεται πως δεν θα αλλοιωθεί η αποθηκευμένη πληροφορία και δεν θα επηρεάσει κάποιο τοπικό ή απομακρυσμένο χρήστη.

## 14.6 Εξόρυξη Δεδομένων

Ονομάζουμε εξόρυξη δεδομένων την εξαγωγή χρήσιμης πληροφορίας από πολύ μεγάλες αποθήκες δεδομένων. Η εξόρυξη πληροφορίας ακολουθεί τα παρακάτω βήματα:

- Καταρχάς “καθαρίζουμε” τα δεδομένα από θόρυβο κ.λπ.
- Συγχώνευση των δεδομένα αν δεν προέρχονται από την ίδια πηγή.
- Επιλογή των δεδομένων (όπου δεδομένα σχετκά με την ανάλυση που λαμβάνει χώρα αντλούνται από τη βάση δεδομένων).

- Μετατροπή των δεδομένων (για να τα έχουμε σε μορφή που μπορεί να αντληθεί).
- Εξόρυξη δεδομένων.
- Αξιολόγηση των προτύπων (ώστε να κρατηθούν αυτά που πραγματικά μας ενδιαφέρουν).
- Αναπαράσταση της γνώσης (τεχνικές οπτικοποίησης, απεικόνισης και αναπαράστασης των δεδομένων σε μορφή που την αντιλαμβάνεται ο χρήστης).

Ορισμένα από τα γνωστά προβλήματα εξόρυξης πληροφορίας είναι:

- Εξόρυξη συχνών συνόλων αντικειμένων (frequent itemsets mining)
- Ομαδοποίηση (Clustering)
- Κατηγοριοποίηση και Πρόβλεψη (classification and prediction)

### 14.6.1 Εξόρυξη Συχνών Συνόλων αντικειμένων

Το πρόβλημα αφορά στα αντικείμενα που αγοράζει κανείς σε μία επίσκεψή του στο super market. Αναζητούμε τα αντικείμενα εκείνα τα οποία αγοράζονται συχνά στην ίδια επίσκεψη κάποιου πελάτη στο κατάστημα. Για παράδειγμα είναι δυνατό να διαπιστωθεί πως κάποιος που αγοράζει γάλα αγοράζει επίσης καφέ και φρυγανιές. Το πρόβλημα είναι γνωστό ως market basket. Τα καλάθια (baskets)είναι οι εγγραφές στη βάση δεδομένων και items είναι τα αντικείμενα που αγοράζει κάποιος πελάτης. Ο ρόλος της πληροφορικής είναι να αντλήσει την πληροφορία το πως θα αξιοποιηθεί αυτή η πληροφορία είναι αντικείμενο της ανάλυσης αγοράς(market analysis) και όχι της πληροφορικής. Κάποιος έμπορος για παράδειγμα μπορεί να αποφασίσει να τοποθετήσει τα αντικείμενα που αγοράζονται συχνά μαζί σε γειτονικά ράφια ώστε να μην ξεχνά ο πελάτης να τα αγοράσει όλα. Μία άλλη ιδέα θα ήταν ίσως να τα τοποθετήσει σε απομακρυσμένα ράφια ουτως ώστε πηγαίνοντας από το ένα ράφι στο άλλο να αγοράσει και άλλα προϊόντα που μεσολαβούν.

Το ίδιο πρόβλημα συναντάται και σε άλλες εφαρμογές. Για παράδειγμα στο web mining όπου καλάθια είναι οι σελίδες του web και αντικείμενα είναι οι σελίδες που κάνουν link σε αυτές. Η πληροφορία που αντλούμε είναι ποιές σελίδες έχουν παρόμοια links και πιθανόν αυτό δηλώνει πως οι σελίδες είναι γύρω από το ίδιο θέμα. Στα κείμενα και στις λέξεις έχουμε μία ακόμη εφαρμογή όπου καλάθια είναι τα κείμενα, και items οι λέξεις. Η πληροφορία που παίρνουμε είναι ότι κείμενα στα οποία εμφανίζονται συχνά ίδιες λέξεις θα είναι του ίδιου περιεχομένου.

Οι αλγόριθμοι εξόρυξης συχνών συνόλων αντικειμένων χωρίζονται σε δύο μεγάλες κατηγορίες. Σε αυτούς που έχουμε παραγωγή υποψηφίων συνδυασμών αντικειμένων και σε αυτούς που δεν έχουμε. Στην πρώτη κατηγορία ο πιο γνωστός αλγόριθμος είναι ο A-priori που παρουσιάστηκε το 1994 από τους Agrawal και Shrikant [6]. Στην δεύτερη κατηγορία ο πιο γνωστός είναι ο FP-Growth που παρουσιάστηκε από τους Han, Pei, Yin το 2000 [7]. Για να εξηγήσουμε τι σημαίνει παραγωγή υποσυνόλων θεωρούμε πως έχουμε ένα πίνακα με τις γραμμές του να αποτελούν τα προϊόντα που αγοράζει κάποιος καταναλωτής σε μία επίσκεψή του στο κατάστημα και οι στήλες του τα προϊόντα. Θεωρούμε πως τα ονόματα των προϊόντων θα είναι α, β, γ, ... και ξεκινούμε

την διαδικασία παραγωγής των υποσυνόλων. Στο πρώτο βήμα παράγουμε όλα τα μονοσύνολα {α}, {β}, {γ}, {δ}, ... . Στο δεύτερο βήμα έχουμε τα υποσύνολα με δύο στοιχεία {α,β}, {α,γ}, ... . Στο τρίτο βήμα έχουμε τα υποσύνολα με τρία στοιχεία {α,β,γ}, {α,β,δ}, ... και ούτω καθεξής.

Ένας άπλοϊκός αλγόριθμος για να βρούμε τα αντικείμενα που αγοράζονται συχνά μαζί είναι να δημιουργούμε κάθε φορά ένα υποσύνολο και να διασχίζουμε τη βάση δεδομένων προκειμένου να βρούμε σε πόσες εγγραφές εμφανίζονται τα στοιχεία του υποσυνόλου. Αυτό όμως απαιτεί πολλές διασχίσεις της βάσης δεδομένων και ο χρόνος εκτέλεσης είναι μεγάλος. Ένα άλλο σημαντικό πρόβλημα είναι το εκθετικό πλήθος υποσυνόλων που θα δημιουργηθεί. Η πρόταση των Agrawal και Shrikant ήταν η μείωση αυτού του πλήθους των υποσυνόλων και βασίζεται στην ιδέα ότι αν ένα υποσύνολο δεν εμφανίζεται συχνά στις εγγραφές του πίνακα κανένα υπερσύνολό του δεν θα εμφανίζεται συχνά. Για παράδειγμα αν το κατώφλι συχνότητας εμφάνισης έχει οριστεί στο 30 και τα προϊόντα γ,ε,κ εμφανίζονται μαζί σε 25 εγγραφές αποκλείεται να συναντήσουμε κάποιο σύνολο τεσσάρων ή περισσοτέρων στοιχείων με συχνότητα εμφάνισης μεγαλύτερη του 25. Έτσι ο αλγόριθμός τους έχει ως εξής:

Χωρίς βλάβη της γενικότητας τοποθετούμε τα αντικείμενα με λεξικογραφική διάταξη. π.χ. {a,b,c,d,e,f,g,h}

- Δημιουργία των μονοσυνόλων και διάσχιση της βάσης προκειμένου να διαπιστωθεί ποια ξεπερνούν το κατώφλι συχνότητας.
- Διαγράφονται τα μονοσύνολα (προϊόντα) με συχνότητα μικρότερη από το κατώφλι συχνότητας.
- Από αυτά που απομένουν δημιουργούνται τα δισύνολα. Για παράδειγμα να είχαν ξεπεράσει το κατώφλι συχνότητας τα b,c,e,g,h τα υποψήφια συχνά δισύνολα που προκύπτουν είναι bc,be,bg,bh,ce,cg,ch,eg,eh,gh. Δεν υπάρχει ενδεχόμενο να υπάρχει κάποιο άλλο δισύνολο με συχνότητα μεγαλύτερη από το κατώφλι συχνότητας.
- Ακολουθεί διάσχιση της βάσης δεδομένων προκειμένου να διαπιστωθεί ποια από αυτά τα δισύνολα είναι συχνά.
- Τα μη συχνά διαγράφονται.
- Στη συνέχεια δημιουργούνται από τα συχνά δισύνολα τα τρισύνολα. Δύο δισύνολα συσχετίζονται για να πάρουμε ένα τρισύνολο αν και μόνο αν διαφέρουν ως προς το τελευταίο στοιχείο. Σε διαφορετική περίπτωση είτε δεν θα είναι συχνό είτε θα έχει ήδη εμφανιστεί. υπολογίζεται η συχνότητα των υποψήφιων συχνών υποσυνόλων. Διασχίζεται η βάση προκειμένου να υπολογιστούν οι συχνότητες, διαγράφονται τα μη συχνά και ουτω κάθεξής.
- Ο Αλγόριθμος τερματίζει όταν για κάποια τιμή του n δεν σχηματίζεται κανένα σύνολο με n+1 στοιχεία.

Ο λόγος που χρησιμοποιήθηκε λεξικογραφική διάταξη είναι για να παραχθούν τα υποσύνολα με ένα στοιχείο παραπάνω πιο απλά. Για παράδειγμα αν έχουμε τα τρισύνολα {abd, abf, adf, bcf, bcg, bck, bdf, bdl, cdf} συσχετίζουμε το πρώτο στοιχείο “abd” με το επόμενο “abf” και προκύπτει το “abdf”. Ελέγχουμε αν όλα τα υποσύνολά του με τρία στοιχεία εμφανίζονται στη λίστα

με τα συχνά τρισύνολα. Και όντως το “adf” και το “bdf” υπάρχουν επομένως δεν μπορούμε να αποφανθούμε αν το “abdf” είναι συχνό ή όχι αν πριν διασχίσουμε τη βάση. Συνεχίζοντας πρέπει να συγκρίνουμε το “abd” και το “adf”. Αλλά από αυτά προκύπτει το “abdf” που έχει ήδη ελεχθεί. Επομένως αρκεί σε οποιόδηποτε επίπεδο (καθορίζεται από το πλήθος των προϊόντων που συνδυάζονται) να συσχετίζουμε μόνο αντικείμενα που διαφέρουν ως προς το τελευταίο στοιχείο. Στο συγκεκριμένο παράδειγμα ο έλεγχος για τετρασύνολα που περιέχουν τα αντικείμενα a και b έχει ολοκληρωθεί. Συνεχίζουμε με το “adf” το οποίο δεν μπορεί να συσχετιστεί με κανένα επόμενο στοιχείο. Ακολουθεί το “bcf” και το “bcg” από τα οποία προκύπτει το “bcfg”. Τα υποσύνολα με τρία στοιχεία του “bcfg” πέρα από τα “bcf” και το “bcg” είναι τα “bfg”, “cfg” τα οποία δεν συναντούμε στην υπόλοιπη λίστα των συχνών υποσυνόλων με τρία στοιχεία και επομένως το “bcfg” δεν είναι υποψήφιο συχνό υποσύνολο και δεν χρειάζεται να δασχίσουμε τη βάση προκειμένου να υπολογίσουμε τη συχνότητά του. Όταν από κάποιο επίπεδο δεν προκύψουν νέα συχνά υποσύνολα ο αλγόριθμος τερματίζει.

Το 2000 προτάθηκε από τους Han, Pei, Yin [7] ο αλγόριθμος FP-growth ο οποίος δεν παράγει υποψήφια συχνά σύνολα αντικειμένων αλλά χρησιμοποιεί δενδρική δομή αποθήκευσης της βάσης. Ο αλγόριθμος διασχίζει μία φορά τη βάση για να υπολογίσει τη συχνότητα εμφάνισης κάθε αντικειμένου. Στη συνέχεια αφαιρεί τα μη συχνά αντικείμενα και επαναδιατάσει τα συχνά με φθίνουσα σειρά συχνότητας. Τοποθετεί τις εγγραφές στο δέντρο και διασχίζει το δέντρο για να υπολογίσει τις συχνότητες των συνόλων αντικειμένων.

### 14.6.2 Ομαδοποίηση (Clustering)

Ομαδοποίηση λέγεται η οργάνωση μιας συλλογής από αντικείμενα-στοιχεία σε ομάδες με βάση κάποιο μέτρο ομοιότητας. Τα στοιχεία είναι σημεία ή διανύσματα σε πολυδιάστατο χώρο και τα στοιχεία της ίδιας ομάδας παρουσιάζουν ομοιότητα. Η ομοιότητα καθορίζεται από τη συνάρτηση απόστασης. Η συνάρτηση απόστασης όποια και αν είναι θα πρέπει να ικανοποεί την τριγωνική ανισότητα. (η απόσταση κάθε στοιχείου με τον ευατό του είναι 0, η απόσταση του αντικειμένου x με το αντικείμενο y είναι ίση με αυτή του y με το x και η απόσταση x και y είναι μικρότερη από αυτή του x και z συν αυτή του z και y για κάθε x,y,z). Υπάρχουν διάφορες συναρτήσεις όπως η ευκλείδια απόσταση, η απόσταση Manhattan το μέγιστο της διαφοράς σε κάθε διάσταση.

Οι αλγόριθμοι clustering χωρίζονται σε τρεις κατηγορίες Partitioning methods, Hierarchical methods, Density Based methods.

Το πρόβλημα της ομαδοποίησης έχει ως εξής: Δίνεται η βάση δεδομένων με τα αντικείμενα (n objects) και ο αριθμός των clusters k. Πρέπει να βρούμε τα αντικείμενα του κάθε cluster.

*Partitioning Methods*

Αρχικά γίνεται τυχαία ομαδοποίηση των δεδομένων σε k clusters σύμφωνα με τους παρακάτω κανόνες:

- Σε κάθε cluster υπάρχει τουλάχιστον ένα αντικείμενο.
- Κάθε αντικείμενο να ανήκει σε ένα και μόνο cluster.

- Ακολουθούν επαναληπτικές τεχνικές επαναταξινόμησης αντικειμένων σε clusters με υπολογισμό της απόστασης των αντικειμένων από τα κέντρα των clusters και υπολογισμό εκ νέου των κέντρων των clusters.

*Hierarchical methods*

Χωρίζονται σε δύο κατηγορίες τους bottom up (agglomerative) και τους top down (divisive). Η διαφορά τους έγγειται στον τρόπο κατανομής των αρχικών δεδομένων στους clusters. Στην πρώτη τεχνική κάθε cluster αποτελείται από ένα αντικείμενο και ενώνοντας κατάλληλα τους clusters φτάνουμε στον επιθυμητό αριθμό. Η δεύτερη θεωρεί πως όλα τα αντικείμενα ανήκουν στον ίδιο cluster ο οποίος διασπάται σε μικρότερους ώσπου να φτάσουμε στον επιθυμητό αριθμό.

Τέλος οι *Density Based* βασίζονται στην πυκνότητα γύρω από κάποιο σημείο που θεωρείται cluster. Αντιπροσωπευτικός αλγόριθμος ο DBScan.

## 14.7 Διαδραστικό Υλικό – Σύνδεσμοι

- http://www.w3schools.com/sql/trysql.asp?filename=trysql_select_all
- http://www.tutorialspoint.com/execute_sql_online.php

## 14.8 Ασκήσεις

Μπορείτε να χρησιμοποιήσετε τον πρώτο από τους παραπάνω free sql editors για τα προβλήματα που ακολουθούν. Οι πίνακες βρίσκονται στα δεξιά.

1. Στον πίνακα Products υπάρχουν τα πεδία ProductID, ProductName, SupplierID, CategoryID, Unit, Price. Να διατυπώσετε τη λειτουργία που θα σας δώσει όλα τα προϊόντα που έχουν CategoryID =8
2. Στον ίδιο πίνακα να βρεθούν τα προϊόντα με Price<30 και CategoryID=7
3. Στον πίνακα Employees να προσθέσετε έναν υπάλληλο με τιμές της επιλογής σας.
4. Από τον πίνακα Orders να αναζητήσετε όλα τα EmployeeID των υπαλλήλων που εξυπηρέτησαν τον πελάτη με CustomerID=34
5. Στον πίνακα Products να διαγραφούν όλα τα προϊόντα με τιμή>40
6. Στον πίνακα Orders να τροποποιήσετε τις τιμές CustomerID που είναι μικρότερες από 100 και να τις αντικαταστήσετε με τιμές πολλαπλασιασμένες με το 100.
7. Από τον πίνακα Customers δημιουργήστε έναν νέο πίνακα που δεν θα περιέχει το πεδίο ContactName.

# Βιβλιογραφία


[1] Al Aho and Jeff Ullman: ”Foundations of Computer Science”, W.H.Freeman, 1992, free online.

[2] Behrouz Forouzan “Εισαγωγή στην Επιστήμη των Υπολογιστών” τρίτη έκδοση 2014

[3] Wikipedia Data Base https://en.wikipedia.org/wiki/Database

[4] Wikipedia Relational Data Base https://en.wikipedia.org/wiki/Relational_database

[5] Wikipedia Distributed Data Base https://en.wikipedia.org/wiki/Distributed_database.

[6] Rakesh Agrawal and Ramakrishnan Srikant. Fast Algorithms for Mining Association Rules in Large Databases. VLDB’94, Proceedings of 20th International Conference on Very Large Data Bases, pp. 487–499, 1994.

[7] J. Han, H. Pei, and Y. Yin. Mining Frequent Patterns without Candidate Generation. In: Proc. Conf. on the Management of Data (SIGMOD’00, Dallas, TX). ACM Press, New York, NY, USA 2000.

# Κεφάλαιο 15

# Τεχνητή Νοημοσύνη

## 15.1 Εισαγωγή

Ο όρος "τεχνητή νοημοσύνη" (ΤΝ) αναφέρεται στον κλάδο εκείνο της πληροφορικής που ασχολείται με την ανάπτυξη ευφυών συστημάτων. Ο όρος αποδίδεται στον John McCarthy ο οποίος τον εισήγαγε για πρώτη φορά το 1956 στη διάσκεψη του Dartmouth. Ως ευφυή συστήματα θεωρούμε τα υπολογιστικά εκείνα συστήματα που σκέφτονται και ενεργούν όπως ο άνθρωπος. Η ΤΝ εφάπτεται πολλών επιστημών πέραν αυτής πληροφορικής όπως της ψυχολογίας, της φιλοσοφίας, της γλωσσολογίας, της λογικής κ.λπ. Ένα από τα βασικά ερωτήματα των επιστημόνων είναι το κατά πόσον είναι εφικτή η ΤΝ. Ο Alan Turing το 1950 πρότεινε μία δοκιμασία (ένα test) που εξετάζει την ικανότητα μιας μηχανής να επιδεικνύει ευφυή συμπεριφορά που ισοδυναμεί με αυτή ενός ανθρώπου. Για να περάσει αυτή τη δοκιμασία ένας υπολογιστής θα πρέπει να έχει διάφορες ικανότητες όπως επεξεργασία φυσικής γλώσσας, αναπαράσταση γνώσης, μηχανική όραση κ.λπ.

Ένα κλασσικό παράδειγμα της δοκιμασίας Turing είναι το εξής: Έχουμε δύο δωμάτια που δεν επικοινωνούν μεταξύ τους, δύο ανθρώπους και έναν υπολογιστή. Ο ένας άνθρωπος τοποθετείται στο πρώτο δωμάτιο ενώ ο άλλος στο δεύτερο μαζί με τον υπολογιστή. Ο άνθρωπος στο δεύτερο δωμάτιο ή ο υπολογιστής (η επιλογή είναι τυχαία) απαντούν σε μία ερώτηση. Οι απαντήσεις αποστέλλονται στον άνθρωπο του πρώτου δωματίου με γραπτό κείμενο κάθε φορά. Ο άνθρωπος αυτός πρέπει από την απάντηση που λαμβάνει να αποφασίσει αν αυτή έχει δωθεί από τον Η/Υ ή από τον άνθρωπο. Η δοκιμασία δεν εξετάζει αν οι απαντήσεις είναι ορθές αλλά αν προέρχονται από άνθρωπο ή Η/Υ. Αν από τις απαντήσεις δεν είναι δυνατό να διευκρινιστεί ποιά απάντηση δόθηκε από τον Η/Υ και ποια από τον άνθρωπο, τότε ο άνθρωπος και ο Η/Υ έχουν παρόμοια ευφυία.

## 15.2 Ιστορικά Στοιχεία

Οι απαρχές της τεχνητής νοημοσύνης ανάγονται στην αρχαιότητα:

- 550 π.Χ. Πυθαγόρας: Όλα τα υπαρκτά αντικείμενα μπορούν να αναχθούν σε αριθμητικές

σχέσεις.

- 400 π.Χ. Πλάτωνας: Η διαλεκτική ως δρόμος για την καθαρή γνώση. Αναζήτηση μεθόδων εύρεσης της γνώσης και της αλήθειας.
- 384-322 π.Χ. Αριστοτέλης: Οι "συλλογισμοί του" παρείχαν πρότυπα εκφράσεων που έδιναν πάντα σωστά συμπεράσματα από σωστές υποθέσεις (Αριστοτέλεια συλλογιστική). Αρχές της λογικής.
- 1630 μ.Χ. Καρτέσιος: Ο πρώτος μοντέρνος ορθολογιστής.
- 1640 μ.Χ. Hobbes: Η σοφία ως προϊόν λογικής.
- 1840 μ.Χ. Boole: Συσχετισμός μεταξύ αλγεβρικών και λογικών πράξεων.
- 1930 μ.Χ. Goedel: Απόδειξη του θεωρήματος της μη-πληρότητας Υπάρχουν αλήθειες που δεν μπορούν αποδεικνύονται.
- 1935 μ.Χ. Church και Turing: Ανάπτυξη της έννοιας της αποτελεσματικής διαδικασίας. Αναγωγή κάθε τυπικής συμβολικής επεξεργασίας σε απλές μηχανικές πράξεις.
- 1949 μ.Χ. Ο Shannon έθεσε τις βάσεις για τον προγραμματισμό του παιξίματος σκακιού.
- 1956 μ.Χ. Ο John McCarthy εισήγαγε τον όρο ΤΝ (Artificial Intelligence). 1950-μέσα 1970μ.Χ. Πρώτη φάση της ΤΝ (συμβολική/κλασσική ΤΝ). Κάποιες σημαντικές και επιτυχίες της ΤΝ αφορούσαν στη:
  (i) Μηχανική μετάφραση (language translation) τεχνικών κειμένων.
  (ii) Επίλυση προβλημάτων (problem solving) πχ. (i) Logic Theorist (Newell, Shaw και Simon, 1957) το οποίο αποδεικνύει θεωρήματα της λογικής και μάλιστα – κάποιες φορές – μέσω συντομότερων αποδείξεων από τις υπάρχουσες. (ii) General Problem Solver (Newell και Simon, 1972) το οποίο χρησιμοποιεί γενικές ευριστικές μεθόδους (heuristics) και εμπειρικούς κανόνες (rules of thumb) προκειμένου να επιλύσει προβλήματα.
  (iii) Αναγνώριση προτύπων (pattern recognition) πχ. κατανόηση κώδικα Morse ή αναγνώριση του γράμματος "Α" σε χειρόγραφο κείμενο.
- 1959 μ.Χ. Ο McCarthy ανέπτυξε τη γλώσσα λογικού προγραμματισμού LISP.
- 1965 μ.Χ. Ο Robinson ανέπτυξε τη γλώσσα λογικού προγραμματισμού PROLOG.
- 1972 μ.Χ. Ο Winograd εισήγαγε ένα σύστημα αναγνώρισης τεχνητού μικρόκοσμου κύβων (blocks world).
- Μέσα 1970-μέσα 1980 μ.Χ. Δεύτερη φάση της ΤΝ. Η ΤΝ επεκτείνεται και σε εμπορικές εφαρμογές πέρα από τις ερευνητικές). Δόθηκε έμφαση στην αναπαράσταση γνώσης και στη δημιουργία αποδοτικών τεχνικών.

- Μέσα 1980-σήμερα: Τρίτη φάση της ΤΝ. Πειράματα και εφαρμογές, οι οποίες δίνουν απαντήσεις σε δύσκολα προβλήματα. Δημιουργούνται παρακλάδια της ΤΝ, όπως τα τεχνητά νευρωνικά δίκτυα, η ασαφής λογική, τα έμπειρα συστήματα και οι γενετικοί αλγόριθμοι, οι κοινωνίες μυρμηγκιών κ.λπ.

## 15.3 Η Τεχνητή νοημοσύνη σήμερα

Επιτυχίες της ΤΝ περιλαμβάνουν:

- Σε θέματα καθημερινής γνώσης
  - Αντίληψη, πχ. όραση (ανάλυση και κατηγοριοποίηση οπτικών δεδομένων, επεξεργασία εικόνων), ομιλία (κατανόηση απλής γλώσσας, απάντηση σε απλές ερωτήσεις, συνομιλία, δυνατότητα άντλησης πληροφοριών από φυσική γλώσσα). Σημειώνεται ότι αυτά επιτυγχάνονται για καλά καθορισμένα και πολύ περιορισμένα περιβάλλοντα (περιορισμοί στον χώρο του προβλήματος).
  - Ρομποτική, πχ. κίνηση και δράση στο περιβάλλον, αλληλεπίδραση στα ερεθίσματα του περιβάλλοντος, χειρισμός αντικειμένων.
  - Οργάνωση δράσης ώστε να επιτευχθεί ένας σκοπός.
  - Σύνθεση πρωτότυπων καλλιτεχνικών δημιουργημάτων (ποιημάτων, μελωδιών, ζωγραφικής).
  - Εκμάθηση μέσα από παραδείγματα καθώς και μέσα από λάθη, με αποτέλεσμα την βελτίωση της απόδοσης.
- Σε θέματα έμπειρης γνώσης
  - Επίλυση δύσκολων προβλημάτων μαθηματικών και λογικής (λόγω της ταχύτητάς τους σε αριθμητικές πράξεις και της μεγάλης και ακριβούς μνήμης τους), πχ. ολοκληρώματα, εύρεση ριζών.
  - Απόδειξη θεωρημάτων των μαθηματικών και της λογικής (πχ. απόδειξη χρωματισμού χαρτών).
  - Ικανότητα νίκης με αντιπάλους ανθρώπους/πρωταθλητές σε ευρύ πεδίο πνευματικών παιχνιδιών (πχ. σκάκι).
  - Εφαρμογή έμπειρων αποφάσεων, πχ. διάγνωση ασθενειών (διαγνωστική ιατρική), ανάλυση και κατασκευή πολύπλοκων κυκλωμάτων, αυτόματος έλεγχο πολύπλοκων βιομηχανικών συστημάτων, παροχή συμβουλών (ειδικά σε μεγάλες βάσεις δεδομένων), δημιουργία νέων προϊόντων από συνδυασμούς υπαρχόντων, διάγνωση μηχανικών βλαβών (συλλογή γνώσης πολλών ειδικών).
- Σε άλλα θέματα
  - Ελεγχος ορθογραφίας/γραμματικής/σύνταξης κειμένων.

  - Τηλε-εκπαίδευση, προσαρμοζόμενη σε μεταβαλλόμενα επίπεδα γνώσης.
  - Εκτέλεση δύσκολων/βαρετών/επικίνδυνων λειτουργιών.
  - Διευκόλυνση ανθρώπου σε καθημερινές λειτουργίες (πχ. μαγείρεμα, έλεγχος ποιότητας).

Τι δεν μπορούν να κάνουν οι υπολογιστές (μέχρι σήμερα):

- Να περπατήσουν στο πάρκο και να περιγράψουν τι βλέπουν.
- Να ετοιμάσουν ένα απλό γεύμα στην κουζίνα.
- Να παίξουν ατομικά παιχνίδια, πχ. τέννις.
- Να παίξουν ομαδικά παιχνίδια, πχ. ποδόσφαιρο.
- Να κάνουν τους κηπουρούς.
- Να κυνηγήσουν έναν λαγό όπως ένας σκύλος.
- Να συζητήσουν χωρίς περιορισμούς για ένα θέμα.
- Να κατανοήσουν ένα έργο ή μια εκπομπή στην τηλεόραση.
- Να ζωγραφίσουν μία εικόνα ή μια απλή σκηνή που τους περιγράφεται.
- Να οδηγήσουν ένα αυτοκίνητο.
- Να καταλάβουν ένα ανέκδοτο.

## 15.4 Αναπαράσταση Προβλήματος

Η υλοποίηση ενός προβλήματος σε σύστημα Η/Υ που επιδεικνύει ΤΝ προωθεί:

- Τη σκέψη, ειδικότερα τον τρόπο κατάλληλης περιγραφής των δεδομένων του προβλήματος (αναπαράστασης δεδομένων) καθώς και τον τρόπο συνδυασμού διάφορων πράξεων (μετασχηματισμού δεδομένων), έτσι ώστε να επιτευχθεί η ορθή καθώς και οικονομική επίλυση του προβλήματος.
- Την ακρίβεια στη σκέψη. Διανοητικά λάθη και παραλείψεις όσον αφορά στην αναπαράσταση καθώς και στον χειρισμό αποκαλύπτονται.
- Την ποσοτικοποίηση των απαιτήσεων μνήμης και πράξεων. Με την επιτυχή λειτουργία της υλοποίησης τίθενται ανώτερα όρια σε αυτές τις απαιτήσεις.
- Την διακρίβωση των απαραίτητων στοιχείων για την επίλυση. Με την αφαίρεση τμήματος των δεδομένων ελέγχεται εφόσον αυτά αποτελούν απαραίτητα στοιχεία γνώσης.

### 15.4.1 Συμβολική αναπαράσταση

Κάθε αριθμός ή κομμάτι δεδομένων αναπαριστάται στον Η/Υ με δυαδικό τρόπο. Ο χειρισμός των δεδομένων πραγματοποιείται σε αυτή τη μορφή, και η αποθήκευσή τους γίνεται σε καθορισμένες θέσεις μνήμης του Η/Υ. Σημειώνεται ότι ο Η/Υ είναι ιδιαίτερα ισχυρός (ακριβής και ταχύς) σε αριθμητικές και λογικές πράξεις. Στα προγράμματα επεξεργασίας δεδομένων, συνήθως εκτελείται περιγραφή των δεδομένων με ανάθεση σε μεταβλητές του προγράμματος, δηλαδή μέσω της χρήσης συμβόλων τα οποία λαμβάνουν συγκεκριμένες τιμές από τη μνήμη ή/και μεταβάλλονται κατά τη διάρκεια του προγράμματος. Η συμβολική αυτή αναπαράσταση των δεδομένων της μνήμης επεκτείνεται στην ΤΝ με τη χρήση συμβόλων και συμβολικών δομών για την κωδικοποίηση των δεδομένων, των αρχικών συνθηκών καθώς και των περιορισμών του προβλήματος στον Η/Υ, με σκοπό να προγραμματιστεί η επίλυσή του προβλήματος με τεχνικές ΤΝ. Τα σύμβολα ορίζονται από τον προγραμματιστή της μεθόδου ΤΝ στον Η/Υ έτσι ώστε η αναπαράσταση του προβλήματος στις συμβολικές δομές (καταστάσεις) να έχει τα εξής χαρακτηριστικά:

- Γενικότητα και πληρότητα, δηλαδή να επιτρέπει την επίλυση όλων των παραδειγμάτων του προβλήματος με την ίδια τεχνική, ανεξαρτήτως μεγέθους προβλήματος.
- Απλότητα και αφαίρεση, ώστε να αναδεικνύει τα σημαντικά χαρακτηριστικά του προβλήματος και να αποκρύπτει τα μη σημαντικά. Ετσι, πλήθος καταστάσεων το οποίο εκφράζεται από τα ίδια σημαντικά χαρακτηριστικά αναπαριστάται από την ίδια συμβολική δομή (οικονομία χώρου και χρόνου επεξεργασίας, π.χ. για ενημέρωση καταστάσεων).

Ετσι η αναπαράσταση είναι κρίσιμη προκειμένου να αναδειχθούν τα στοιχεία τα οποία αποτελούν τα κλειδιά για την επίλυση του προβλήματος, ενώ αντίθετα να κρυφτούν/παραλειφθούν αυτά τα οποία δεν έχουν σχέση με την επίλυση. Και τα δύο προωθούν τον καλό ορισμό του χώρου του προβλήματος. Κατά την υλοποίηση της μεθόδου ΤΝ εκτελείται τυπική διαχείριση συμβόλων. Ετσι, ενώ η έννοια των συμβόλων αποδίδεται κατά την έκφραση/περιγραφή του προς επίλυση προβλήματος, τα σύμβολα δεν έχουν έννοια οπότε η επίλυση και τα ενδιάμεσα εξαγόμενα συμπεράσματα ανάγονται στον χειρισμό των συμβόλων που αναπαριστούν το πρόβλημα.

### 15.4.2 Επιλογή Συμβολικής Αναπαράστασης

Η επιλογή της αναπαράστασης στον Η/Υ, δηλαδή η επιλογή συμβόλων και συμβολικών δομών για την κωδικοποίηση του προβλήματος (επιθυμητής λύσης, δεδομένων και περιορισμών) στο πρόγραμμα με τεχνικές ΤΝ καθορίζει σε ένα μεγάλο βαθμό όχι μόνο τον τρόπο επίλυσης αλλά και την αποδοτικότητά (ταχύτητα, οικονομία μνήμης και ποσοστό επιτυχίας) της.

Ενα παράδειγμα της σημασίας της αναπαράστασης παρουσιάζεται στον τομέα της αναγνώρισης προτύπων, συγκεκριμένα στο πρόβλημα αναγνώρισης χαρακτήρων (optical character recognition) και ιδιαίτερα στο πρόβλημα του εάν οι εισαγόμενοι χαρακτήρες κατατάσσονται σε ένα από τα 10 ψηφία (0, 1, 2, 3, 4, 5, 6, 7, 8, 9).

### 15.4.3 Χειρισμός Συμβολικής Αναπαράστασης Λογική

Τα δεδομένα καθώς και η γνώση ενός προβλήματος απαρτίζουν γεγονότα (facts), δηλαδή το τι ισχύει και τι όχι στον χώρο του προβλήματος. Τα γεγονότα εκφράζονται μέσω δηλωτικών προτάσεων (declarative sentences) και αναπαριστώνται μέσω μεταβλητών. Για παράδειγμα, το γεγονός ότι η θερμοκρασία έξω είναι 0ο εκφράζεται μέσω της δηλωτικής πρότασης ”Κάνει κρύο” και αναπαριστάται μέσω της μεταβλητής ΚΡΥΟ ή - απλούστερα - ως x. Καθότι η αναπαράσταση στον Η/Υ πραγματοποιείται σε δυαδική μορφή (1 για αληθές γεγονός, 0 για ψευδές γεγονός), χρησιμοποιείται η δυαδική λογική (Boolean) για την αναπαράσταση και τον χειρισμό των γεγονότων, προτάσεων και αντίστοιχων μεταβλητών. Ετσι η μεταβλητή x λαμβάνει τιμή 1 ή 0 ανάλογα με την απάντηση ψευδές ή αληθές, αντίστοιχα, στο ερώτημα ”Είναι αλήθεια ότι x”; Για παράδειγμα, για θερμοκρασία έξω -10ο η μεταβλητή x παίρνει την τιμή 1, ενώ για θερμοκρασία έξω +40ο η μεταβλητή x παίρνει την τιμή 0. Τα γεγονότα (και οι αντίστοιχες προτάσεις και μεταβλητές) είναι δυνατόν να συνδυαστούν. Οι συνδυασμοί γεγονότων αποτελούν εκφράσεις. Η λογική (Hodges, 1977) αποτελεί τη μελέτη συνεπών συνόλων γεγονότων ή συνδυασμών τους (δηλαδή εκφράσεων). Επικεντρώνεται στην αλήθεια και την συμπερασματική διαδικασία (inference), με άλλα λόγια στην εύρεση των προϋποθέσεων (τιμές 1 ή 0 των γεγονότων) κάτω από τις οποίες ένα σύνολο γεγονότων ή μία έκφραση είναι αληθή.

#### Προτασιακή λογική (propositional logic)

Είναι η λογική την οποία χρησιμοποιούν οι Η/Υ. Δεδομένης μάς έκφρασης (ως συνδυασμός δεδομένων), εξάγεται ένα αποτέλεσμα το οποίο εκφράζει την αληθή/ψευδή κατάσταση της έκφρασης (1 για αληθή κατάσταση, 0 για ψευδή κατάσταση). Ο πίνακας αληθείας είναι το σύνηθες εργαλείο προκειμένου να εξαχθεί η αλήθεια ή όχι μίας έκφρασης: ο συνδυασμός τιμών των εμπλεκομένων λογικών μεταβλητών εφαρμόζεται εξαντλητικά και η εφαρμογή των τελεστών της έκφρασης για κάθε συνδυασμό δημιουργεί ένα αληθές ή ψευδές αποτέλεσμα. Για τον συνδυασμό περισσοτέρων από δύο λογικών μεταβλητών χρησιμοποιούνται παρενθέσεις, οι οποίες καθορίζουν την προτεραιότητα εφαρμογής των λογικών πράξεων. Σημειώνεται ότι αν δεν χρησιμοποιηθούν παρενθέσεις μεγαλύτερη προτεραιότητα έχει ο τελεστής $\neg$ μετά οι τελεστές $\vee$ και $\wedge$, και τέλος οι τελεστές $\rightarrow$ και $\leftrightarrow$. Ετσι, η προτασιακή λογική επιτρέπει την εύκολη και σαφή έκφραση σχέσεων λογικών μεταβλητών, όπου το αποτέλεσμα ανάγεται πάντα σε μία ακολουθία τελεστών $\neg$, $\vee$ και $\wedge$ (βασική μορφή συνδυασμού λογικών μεταβλητών). Είναι λοιπόν προτιμότερη η χρήση κάποιας συμπερασματικής διαδικασίας προκειμένου να επαληθευτεί ή να απορριφθεί το συμπέρασμα. Για παράδειγμα, η απαγωγή σε άτοπο υποθέτει ότι δεν ισχύει το συμπέρασμα $\neg(p \wedge q)$, άρα ότι ισχύει η έκφραση $p \wedge q$. Τότε όμως η έκφραση $(\neg p \vee r) \wedge (\neg p \vee t)$ ικανοποιείται μόνο εάν η έκφραση $r \wedge t$ είναι αληθής. Σε αυτή την περίπτωση όμως, το δεύτερο μέλος (άρα και ολόκληρη η έκφραση) $(\neg sV \neg q) \wedge (\neg tV \neg q)$ δεν ικανοποιείται, άρα και οι δύο εκφράσεις των γεγονότων του προβλήματος είναι αναγκαστικά ψευδείς. Κάτι τέτοιο όμως δεν ισχύει, οπότε συμπεραίνεται ότι το συμπέρασμα είναι αληθές.

**Κατηγορηματική λογική (predicate logic)**

Το ότι εκφράσεις της μορφής "Ο Γιώργος είναι άνθρωπος", "Η γάτα έχει τέσσερα πόδια", "Η Ελένη αγαπά το παγωτό" και "Η Ελένη τρώει το παγωτό" ή "Η Ελένη έφαγε τρία παγωτά" αναπαριστώνται στην προτασιακή λογική μόνο ως αδιαίρετες έννοιες συνεπάγεται ότι προτάσεις οι οποίες είναι φανερό ότι είναι ασύμβατες δεν μπορούν να ανιχνευτούν ως τέτοιες, πχ. οι εκφράσεις "Κάνει κρύο" και "Κάνει ζέστη". Επίσης εκφράσεις της μορφής "Όλοι οι άνθρωποι έχουν δύο πόδια" μπορούν να εκφραστούν μόνο ως σύνολο προτάσεων που αφορούν σε κάθε μεταβλητή-άνθρωπο του χώρου του προβλήματος ξεχωριστά, πχ. "Ο Γιώργος έχει δύο πόδια", "Η Ελένη έχει δύο πόδια" κλπ. Η κατηγορική λογική επιτρέπει την πιο δομημένη και κατανεμημένη έκφραση γεγονότων (ακόμη και αρκετά περίπλοκων) καθώς και την (συμπερασματική) παραγωγή νέων γεγονότων που απορρέουν από υπάρχοντα γεγονότα. Για το σκοπό αυτό έχει καλά ορισμένη σύνταξη και συγκεκριμένους κανόνες παραγωγής. Η σύνταξη εκφράσεων της κατηγορικής λογικής στηρίζεται στους όρους και τα κατηγορήματα.

- Ο όρος (term) αποτελεί είτε μία σταθερά (αδιάσπαστο γεγονός του χώρου του προβλήματος, object), είτε μία μεταβλητή (variable) η οποία λαμβάνει τιμές μέσα από κάποιο σύνολο σταθερών.
- Το κατηγόρημα (predicate) εκφράζει κάποια σχέση μεταξύ των όρων. Συντάσσεται ως συνάρτηση (function) με αντικείμενά της (arguments) όρους ή οι οποίοι βρίσκονται μέσα στις παρενθέσεις (σε καθορισμένες θέσεις ανάλογα με τον ρόλο που παίζουν για το κατηγόρημα). Με αυτή τη σύνταξη δημιουργείται μία έκφραση (expression) της κατηγορικής λογικής, δηλαδή ένα γεγονός του χώρου του προβλήματος το οποίο όμως, σε αντίθεση με τους όρους, διασπάται.

Μία σχέση η οποία ισχύει μεταξύ των όρων του κατηγορήματος καθιστά την έκφραση αληθή, ενώ μία σχέση η οποία δεν ισχύει μεταξύ των όρων του κατηγορήματος καθιστά την έκφραση ψευδή. Η άρνηση μίας έκφρασης την καθιστά ψευδή αν η ίδια είναι αληθής και αληθή αν η ίδια είναι ψευδής. Παρατηρείται ότι, στην περίπτωση χρήσης της κατηγορικής λογικής, είναι απαραίτητη η ανάθεση όρων-σταθερών στα κατηγορήματα προκειμένου να δημιουργηθούν συγκεκριμένες εκφράσεις (χωρίς μεταβλητές ή σύμβολα $\forall$ και $\exists$).

Λόγω της μεγαλύτερης πολυπλοκότητας της κατηγορικής λογικής, εάν η αναπαράσταση και επίλυση του προβλήματος είναι δυνατό να πραγματοποιηθεί μέσω της προτασιακής λογικής τότε είναι προτιμότερη. Εάν πάλι είναι απαραίτητο να διασπαστούν/ποσοτικοποιηθούν/γενικευτούν/συγκεκριμενοποιηθούν οι εκφράσεις, τότε προτιμάται η κατηγορική λογική.

Σημειώνεται πάντως ότι, αν και η χρήση πινάκων αληθείας καθώς και οι διάφορες τεχνικές συμπερασματικής διαδικασίας οδηγούν πάντα σε σωστές διαπιστώσεις, είναι δυνατόν μία απόδειξη να είναι τόσο χρονοβόρα που να μην είναι χρήσιμη. Επιπλέον είναι δυνατόν να μην μπορεί να αποδειχτεί κάτι το οποίο είναι αληθές (μη πληρότητα κατηγορικής λογικής, incompleteness) ή η συμπερασματική διαδικασία να μην τερματίζει στην προσπάθειά της να αποδείξει κάτι το οποίο είναι ψευδές (μη επιδεκτικότητα απόφασης κατηγορικής λογικής, undecidability). Είναι γεγονός όμως ότι η δυνατότητα και αποδοτικότητα της απόδειξης εξαρτάται κατά πολύ από την αναπαράσταση του προβλήματος καθώς και την κατάλληλη επιλογή τεχνικής επίλυσης.

**Ασαφής Λογική (fuzzy logic)**

Η δίτιμη λογική (και μέσω αυτής η προτασιακή και κατηγορική λογική), αν και κατάλληλη για την αναπαράσταση και τον χειρισμό γεγονότων, προτάσεων και αντίστοιχων μεταβλητών στον Η/Υ, δεν είναι πάντα επαρκώς ευέλικτη ώστε να εκφράζει επιτυχώς ορισμένες έννοιες, ιδιαίτερα γλωσσικές, υποκειμενικές ή εμπειρικές. Είναι προφανές ότι η δίτιμη λογική δεν επιτρέπει:

(i) Την εις βάθος έκφραση της σχέσης

(ii) Την ακριβή περιγραφή του βαθμού *membership**

(iii) Την αποδοτική έκφραση της σχέσης

Η ασαφής λογική έχει δημιουργηθεί ώστε να καλύψει την αδυναμία της δίτιμης λογικής στην έκφραση μερικού χαρακτηρισμού γεγονότων, προτάσεων και αντίστοιχων μεταβλητών.

Κάθε χαρακτηριστικό της ασαφούς λογικής απεικονίζεται σε μία γραφική παράσταση όπου ο άξονας Χ αναπαριστά το εύρος δυνατών τιμών της μεταβλητής και ο άξονας Υ το ποσοστό συμμετοχής (degree of membership) των τιμών της μεταβλητής στο χαρακτηριστικό. Αντί των τιμών 1 και 0 (δίτιμη λογική) το ποσοστό συμμετοχής μπορεί να λαμβάνει συνεχείς τιμές στο διάστημα [0,1] οι οποίες εκφράζουν το κατά πόσο το χαρακτηριστικό εκφράζει τη συγκεκριμένη τιμή της μεταβλητής. Η καμπύλη που δημιουργείται από όλα τα ζεύγη (x,y) ονομάζεται συνάρτηση συμμετοχής (membership function) του χαρακτηριστικού. Αυτή μπορεί να πάρει πρακτικά οποιαδήποτε μορφή, δεν είναι αναγκαστικά ευθεία ή συνεχής, και συνήθως εμφανίζει ποσοστά συμμετοχής 0 στο ένα ή και στα δύο άκρα του εύρους τιμών της μεταβλητής και ποσοστά που αυξάνονται ανάμεσα. Επιπλέον, διαφορετικά αλλά γλωσσικά, υποκειμενικά ή εμπειρικά σχετιζόμενα χαρακτηριστικά μπορούν να συνδυαστούν στην ίδια γραφική παράσταση εκφράζοντας συνολικά τα διαφορετικά χαρακτηριστικά τα οποία αλληλοσυμπληρώνονται και εκφράζουν πληρέστερα τη μεταβλητή.

## 15.5 Τεχνικές για επίλυση προβλημάτων

Η ΤΝ απαιτεί:

- κατάλληλα σχήματα διαχείρισης συμβόλων για την αναπαράσταση του περιβάλλοντος, δηλαδή για την σύλληψη της γνώσης της σχετικής με το περιβάλλον,
- αποτελεσματικές διαδικασίες για την εξαγωγή και χρήση συμβολικά εκφρασμένης γνώσης προκειμένου να εκτελέσει ευφυείς λειτουργίες (επίλυση προβλημάτων).

Ετσι, αφενός ένα πρόγραμμα ΤΝ χειρίζεται συμβολικές δομές και όχι κατευθείαν αυτό που αναπαριστούν, και αφετέρου η ερμηνεία των καταστάσεων γίνεται από τον χρήστη, μέσα από την αναπαράστασή τους. Καθότι τα περισσότερα προβλήματα της ΤΝ δεν έχουν απευθείας λύση (δεν είναι τετριμμένα) είναι απαραίτητη η εύρεση μίας μεθοδολογίας που (α) να οδηγεί στην λύση πιο αποτελεσματικά από ότι η τύχη (guesswork) και (β) να επιτρέπει την οπισθοδρόμηση (backtracking) εφόσον η πορεία δεν οδηγεί προς τη λύση. Στα επόμενα αναλύονται διάφορες τεχνικές ΤΝ με τα δύο αυτά χαρακτηριστικά.

### 15.5.1 Αναζήτηση

**Επίλυση ως αναζήτηση στον χώρο του προβλήματος**

Ως χώρος του προβλήματος χαρακτηρίζεται το σύνολο των καταστάσεων, δηλαδή όλοι οι συνδυασμοί των δυνατών τιμών των σημαντικών χαρακτηριστικών που αναπαριστούν το πρόβλημα. Λόγω της υπερπληθώρας (έκρηξης) των συνδυασμών αυτών (combinatorial explosion) για όλα εκτός από τα πιο περιορισμένα προβλήματα, η απαρίθμηση όλων των καταστάσεων (states) δεν είναι αποδοτική και μερικές φορές δεν είναι ούτε εφικτή.

Η αναζήτηση (search) αποτελεί ένα ταξίδι στον χώρο του προβλήματος: ξεκινώντας από την αρχική κατάσταση πραγματοποιείται μετάβαση (state transition) σε μία νέα κατάσταση, από εκεί – μέσω μετάβασης – σε μία νεώτερη κατάσταση και ου τω καθεξής. Η επίλυση ανάγεται στην αναζήτηση που περιγράφει μία διαδρομή (path), δηλαδή ένα ταξίδι μέσω μεταβάσεων από την αρχική σε μία από τις τελικές καταστάσεις του προβλήματος.

Οι καταστάσεις αναπαριστώνται από συμβολικές δομές, οι οποίες όχι μόνο είναι επεξεργάσιμες από τον Η/Υ αλλά και είναι κατάλληλες για χειρισμό από την τεχνική ΤΝ, επί του προκειμένου από την αναζήτηση στον χώρο του προβλήματος. Είναι απαραίτητο να καθοριστούν τα εξής:

- Οι συμβολικές δομές (symbolic structures) οι οποίες αναπαριστούν επακριβώς την πλευρά του προβλήματος τη σχετική με την επίλυσή του.
- Οι διαδικασίες επεξεργασίας (processing procedures) οι οποίες αναγνωρίζουν, δημιουργούν, μετατρέπουν και διαγράφουν συμβολικές δομές. Σημειώνεται ότι οι διαδικασίες επεξεργασίας πρέπει να εκφράζουν και προωθούν αποκλειστικά και μόνο τις επιτρεπτές μεταβάσεις.
- Οι διαδικασίες ελέγχου (control procedures) οι οποίες οργανώνουν την εφαρμογή των διαδικασιών επεξεργασίας ώστε να είναι δυνατή η μετάβαση στην τελική κατάσταση.

Τα τρία αυτά χαρακτηριστικά μεγέθη είναι αλληλεξαρτώμενα.

Τα σύμβολα, οι συμβολικές δομές και οι διαδικασίες επεξεργασίας αποτελούν τα απαραίτητα στοιχεία για τη έκφραση του προβλήματος και της επίλυσης. Για κάθε μετάβαση, πρέπει να πραγματοποιείται έλεγχος σχετικά με το αν η νέα κατάσταση είναι επιτρεπτή. Επιπλέον, προκειμένου να δοθεί μία ευφυής λύση στο πρόβλημα (μικρότερος αριθμός μετακινήσεων από την αρχική μέχρι την τελική κατάσταση), πρέπει όχι μόνο να επιτυγχάνεται μετάβαση σε νέα κατάσταση αλλά και να αποφεύγονται οι κύκλοι, δηλαδή ακολουθίες μετακινήσεων με την πρώτη και τελευταία κατάσταση στην ακολουθία να ταυτίζονται.

Το πρόβλημα είναι τώρα έτσι εκφρασμένο ώστε να είναι κατανοητό, αλλά όχι αναγκαστικά κατάλληλο για προγραμματισμό. Είναι φανερό ότι η επεξεργασία συνόλων με μεταβλητά μεγέθη δεν είναι η προτιμότερη συμβολική δομή. Μια συμβολική δομή σταθερού μήκους είναι προτιμότερη, για παράδειγμα συμβολικές δομές με ακριβώς 5 σύμβολα, όπου το σύμβολο  (ποτάμι) έχει εισαχθεί προκειμένου να ξεχωρίσει τα σύμβολα στη Ο1 από αυτά στην Ο2. Αυτή η αναπαράσταση είναι:

- πλήρης (εμπεριέχει όλα τα απαραίτητα στοιχεία του προβλήματος), διαφανής (κατανοητή από τον χρήστη),
- περιεκτική και αφαιρετική (καθιστά σαφή τα σημαντικά στοιχεία του προβλήματος, - π.χ. την όχθη στην οποία βρίσκεται κάθε σύμβολο - ενώ παραλείπει τα μη σημαντικά στοιχεία του προβλήματος - πχ. το χρώμα της χήνας ή την ποσότητα/ποιότητα του σταριού -), οπότε επιτρέπει την οικονομική έκφραση του προβλήματος),
- ευνοεί τον προγραμματισμό (οι συμβολικές δομές αποθηκεύονται, προσπελαύνονται και μετατρέπονται με ευκολία), και
- είναι εύκολα υλοποιήσιμη.

Καθίσταται λοιπόν και πάλι προφανές πώς η σωστή επιλογή αναπαράστασης επιτρέπει την ρητή έκφραση των περιορισμών που ενυπάρχουν στο πρόβλημα, προκειμένου να διευκολύνεται η επίλυση.

Δεδομένης της συμβολικής δομής που έχει επιλεγεί, το δίκτυο (graph) των καταστάσεων του χώρου του προβλήματος περιέχει έναν κόμβο για κάθε κατάσταση και μία γραμμή για κάθε δυνατή μετάβαση από μία κατάσταση σε μία άλλη, η οποία ενώνει τους αντίστοιχους κόμβους.

Η πορεία προς τη λύση, χρησιμοποιώντας τις διαδικασίες επεξεργασίας και ελέγχου για την μετάβαση σε κάθε δυνατή νέα κατάσταση και τον έλεγχο των μη επιτρεπτών καταστάσεων και των κύκλων. Η πορεία προς τη λύση παρουσιάζει το σύνολο όλων των δυνατών μεταβάσεων και περιλαμβάνει όλες τις δυνατές διαδρομές. Οι μεταβάσεις είναι μη αμφίδρομες, οπότε η πορεία προς τη λύση αποκτά τη μορφή δένδρου (αντί για τη μορφή γράφου): ξεκινώντας από την αρχική κατάσταση (κορυφή του δένδρου), δημιουργούνται παρακλάδια για όλες τις δυνατές μεταβάσεις, κάθε μία από τις οποίες επιτυγχάνεται με την εφαρμογή μίας διαδικασίας επεξεργασίας. Οι διαδικασίες επεξεργασίας εφαρμόζονται κάθε φορά στην όχθη που περιέχει τον καλλιεργητή, άρα στην πλευρά του  που περιέχει το σύμβολο K. Επιπλέον, ο αριθμός των διαδικασιών επεξεργασίας που μπορούν να εφαρμοστούν είναι ανάλογος με τον αριθμό των συμβόλων που βρίσκονται στην ίδια πλευρά με το σύμβολο K όσον αφορά στο σύμβολο  . Σε αυτή την περίπτωση, για κάθε κατάσταση δημιουργούνται τόσες νέες καταστάσεις (παρακλάδια) όσες είναι και οι εφαρμόσιμες διαδικασίες επεξεργασίας. Μέσω των διαδικασιών ελέγχου, από τις νέες καταστάσεις αποκλείονται αυτές που δημιουργούν απαγορευμένες καταστάσεις (κόκκινα τετράγωνα) ή κύκλους (μπλε τετράγωνα). Για κάθε μία από τις αποδεκτές καταστάσεις δημιουργούνται νέα παρακλάδια (μέσω των διαδικασιών επεξεργασίας) τα οποία είτε επεκτείνονται είτε σταματούν (μέσω των διαδικασιών ελέγχου) και η όλη διαδικασία επαναλαμβάνεται μέχρι την μετάβαση στην τελική κατάσταση.

Η εξαντλητική (πλήρης) δημιουργία όλων των δυνατών μεταβάσεων από την αρχική κατάσταση είτε στην τελική κατάσταση είτε σε μία απαγορευμένη κατάσταση ή κύκλο αποτελεί μία τυπική έκφραση της αναζήτησης, η οποία επιβάλλει την εύρεση λύσης αν κάτι τέτοιο είναι εφικτό. Αυτή η μέθοδος εκφράζεται ως εξής:

**Βήμα 1.** Εκκίνηση από την αρχική κατάσταση.
**Βήμα 2.** Εφαρμογή των εφαρμόσιμων διαδικασιών επεξεργασίας στην αρχική κατάσταση. Δημιουργία παρακλαδιών (ένα για κάθε εφαρμόσιμη διαδικασία επεξεργασίας), το καθένα από τα

οποία εκφράζει μία νέα κατάσταση.
**Βήμα 3.** Εφαρμογή των διαδικασιών ελέγχου σε κάθε μία από τις νέες καταστάσεις. Διαγραφή νέων καταστάσεων οι οποίες δεν ικανοποιούν τις διαδικασίες ελέγχου (αποτελούν απαγορευμένες καταστάσεις ή κύκλους).
**Βήμα 4.** Επιλογή μίας από τις νέες καταστάσεις, την τρέχουσα κατάσταση.
**Βήμα 5.** Διαπίστωση ύπαρξης τρέχουσας κατάστασης.

Εάν ναι,

> **Βήμα 5.1.** Διαπίστωση του εάν η τρέχουσα κατάσταση ταυτίζεται με την τελική κατάσταση.
> Εάν ναι,
> Μετάβαση στο **Βήμα 4** (προκειμένου να διαπιστωθεί αν υπάρχει και άλλος τρόπος μετάβασης στην τελική κατάσταση).
> Εάν όχι,
>
> > **Βήμα 5.1.1.** Εφαρμογή των εφαρμόσιμων διαδικασιών επεξεργασίας στην τρέχουσα κατάσταση. Δημιουργία παρακλαδιών (ένα για κάθε εφαρμόσιμη διαδικασία επεξεργασίας), το καθένα από τα οποία εκφράζει μία νέα κατάσταση.
> >
> > **Βήμα 5.1.2.** Εφαρμογή των διαδικασιών ελέγχου σε κάθε μία από τις νέες καταστάσεις. Διαγραφή νέων καταστάσεων οι οποίες δεν ικανοποιούν τις διαδικασίες ελέγχου. Μετάβαση στο **Βήμα 4.**

Εάν όχι,

> **Βήμα 5.2.** Τέλος μεθόδου.

Το δένδρο αναζήτησης δημιουργείται από την πορεία προς τη λύση αφού παραληφθούν όλες οι μεταβάσεις που οδηγούν σε κύκλους ή μη επιτρεπτές καταστάσεις, δηλαδή διατηρώντας μόνο όσες ακολουθίες μεταβάσεων αποτελούν διαδρομές (δηλαδή αποτελούν επιλύσεις του προβλήματος). Επίσης μπορεί να δημιουργηθεί από το δίκτυο του προβλήματος, ξεκινώντας από την αρχική κατάσταση και προχωρώντας προς συνδεδεμένους κόμβους που δεν οδηγούν σε απαγορευμένη κατάσταση. Σημειώνεται ότι μόνο μία διέλευση από γραμμή του γράφου επιτρέπεται. Με αυτόν τον τρόπο περιγράφεται μία μέθοδος επίλυσης προβλημάτων στην ΤΝ, η αναζήτηση, ως ακολουθία μεταβάσεων. Σημειώνεται πάντως ότι αυτή η περιγραφή δεν αποτελεί από μόνη της πρόγραμμα επίλυσης. Για τον προγραμματισμό στον Η/Υ απαιτούνται ακόμη:

> (α) ένας τρόπος αποθήκευσης/χαρακτηρισμού καθώς και διαγραφής/αποχαρακτηρισμού μίας κατάστασης ως νέας,
>
> (β) ένας τρόπος αποθήκευσης καταστάσεων ώστε να αναγνωρίζονται οι κύκλοι,
>
> (γ) ένας τρόπος ελέγχου για ύπαρξη νέων καταστάσεων προκειμένου να εφαρμοστεί το **Βήμα 4**,

(δ) ένας τρόπος διατήρησης των παρακλαδιών που οδηγούν στην τελική κατάσταση (μια και ενδιαφέρει εξίσου η ακολουθία μεταβάσεων από την αρχική στην τελική λύση εξίσου με το αν υπάρχει λύση).

Τα παραπάνω εκφράζονται από συμπληρωματικές διαδικασίες ελέγχου. Σημειώνεται τέλος ότι κάθε διαδρομή μπορεί να χαρακτηρίζεται από ένα κόστος (cost). Αυτό μπορεί να αποδίδει τον αριθμό μεταβάσεων μέχρι την επίλυση (άρα πιο σύντομες διαδρομές είναι και πιο οικονομικές, δηλαδή έχουν μικρότερο κόστος) ή και τη δυνατότητα κάποιων μεταβάσεων να είναι πιο ακριβές από κάποιες άλλες. Στην περίπτωση ύπαρξης κόστους, ο σκοπός της αναζήτησης δεν είναι να προσδιοριστεί μία οποιαδήποτε επίλυση αλλά να βρεθεί η πιο οικονομική επίλυση (πχ. αυτή με τον μικρότερο αριθμό μεταβάσεων ή με το μικρότερο συνολικό κόστος).

### 15.5.2 Μέθοδοι αναζήτησης

Υπενθυμίζεται ότι ο στόχος της ΤΝ είναι (ιδιαίτερα σε πολύπλοκα προβλήματα) να ελαχιστοποιηθεί το κόστος της διαδρομής-επίλυσης. Ετσι, έχουν αναπτυχθεί διάφορες μέθοδοι αναζήτησης οι οποίες διαφέρουν στον τρόπο επιλογής της τρέχουσας κατάστασης στο **Βήμα 4** της προηγούμενης παραγράφου (και άρα στον τρόπο μετάβασης σε νέα κατάσταση). Διακριτικά χαρακτηριστικά των διάφορων μεθόδων αναζήτησης αποτελούν:

(i) Εάν επιτυγχάνεται η εύρεση της πιο οικονομικής (βέλτιστης) λύσης (optimal search) ή απλά αναζητείται κάποια διαδρομή (non-optimal search).

(ii) Εάν η αναζήτηση πραγματοποιείται τυφλά (blind search) ή είναι καθοδηγούμενη (informed search), δηλαδή εάν η κάθε μετάβαση πραγματοποιείται ανεξάρτητα ή βάσει της ποιότητας της νέας κατάστασης (πόσο κοντύτερα εκτιμάται ότι η νέα κατάσταση πλησιάζει στην τελική κατάσταση) αντίστοιχα. Η τυφλή αναζήτηση είναι η πιο απλή: δεν εμπεριέχεται πληροφορία για τον χώρο του προβλήματος οπότε η μετάβαση γίνεται βάσει κάποιου συστηματικού κριτηρίου. Η καθοδηγούμενη αναζήτηση χρησιμοποιεί ευριστικές μεθόδους (heuristics) που αφορούν στην ποιότητα της μετάβασης.

(iii) Εάν η μέθοδος είναι πλήρης (complete), δηλαδή εγγυάται ότι θα βρεθεί μία λύση, εφόσον υπάρχει.

(iv) Εάν χρησιμοποιείται οπισθοδρόμηση σε περίπτωση μη εύρεσης λύσης.

Ο Πίνακας 15.1 κατηγοριοποιεί τις διάφορες μεθόδους αναζήτησης βάσει των παραπάνω χαρακτηριστικών. Σημειώνεται ότι οι απαιτήσεις μνήμης και χρόνου έχουν υπολογιστεί για δένδρα αναζήτησης με b τον μέσο αριθμό παρακλαδιών (branching factor) ανά κόμβο-κατάσταση, d το μέσο βάθος (depth) των παρακλαδιών, και e το μέσο βάθος από την αρχική στην τελική κατάσταση ($e \leq d$).

| ΜΕΘΟΔΟΣ | ΒΕΛΤΙΣΤΗ | ΤΥΦΛΗ | ΠΛΗΡΗΣ | ΟΠΙΣΘΟΔΡΟΜΗΣΗ | ΜΝΗΜΗ | ΧΡΟΝΟΣ |
|---|---|---|---|---|---|---|
| Εξαντλητική | ναι | ναι | ναι | ναι | $O(bd)$ ή $O(b^d)$ | $O(b^d)$ |
| Σε βάθος | όχι | ναι | ναι | ναι | $O(be)$ | $O(b^e)$ |
| Σε πλάτος | όχι | ναι | ναι | ναι | $O(b^e)$ | $O(b^e)$ |
| Καλύτερη-πρώτα | όχι | όχι | ναι | ναι | - | - |
| Ακτινωτή | όχι | όχι | όχι | όχι | $O(bd)$ | $O(bd)$ |
| Αναρρίχηση | όχι | όχι | ναι | ναι | - | - |
| Περιορισμός παρακλαδιών | ναι | όχι | ναι | ναι | - | - |
| Περιορ. παρακλ. με υποεκτίμηση | ναι | όχι | ναι | ναι | - | - |
| Δυναμικός προγραμματισμός | ναι | όχι | ναι | ναι | | |
| A* | ναι | όχι | ναι | ναι | - | - |

Πίνακας 15.1: Χαρακτηριστικά μεθόδων αναζήτησης.

### 15.5.3 Επίλυση ως ανάλυση και ικανοποίηση περιορισμών

Κάθε σύστημα επιδεικνύει περιορισμούς: τα τμήματά του σχετίζονται (συναρμολογούνται) μέσω καλά καθορισμένων σχέσεων. Ετσι, αν μέσω της ανάλυσης ενός συστήματος είναι δυνατό να καταστεί γνωστό ακριβώς το πώς τα τμήματά του επιτρέπεται να σχετίζονται, τότε αυτή η γνώση μπορεί να χρησιμοποιηθεί ώστε να επιτρέψει αφενός τη συναρμολόγηση ενός νέου συστήματος και αφετέρου την διαπίστωση του εάν ένα συγκεκριμένο σύστημα είναι κατασκευασμένο βάσει των επιτρεπτών σχέσεων (και άρα ανήκει στην κατηγορία συστημάτων που διέπονται από τους ίδιους περιορισμούς).

Για συστήματα που συναρμολογούνται σύμφωνα με καλά καθορισμένες όσο περισσότερα τμήματα συνδυάζονται τόσο αυξάνονται οι περιορισμοί σχετικά με το πώς μπορούν να συναρμολογηθούν τα υπόλοιπα τμήματα. Μάλιστα, αυτά ενδέχεται να μην μπορούν να συναρμολογηθούν, οπότε πρέπει να αποσυναρμολογηθούν κάποια ήδη συναρμολογημένα τμήματα και να δοκιμαστεί εκ νέου η συναρμολόγηση. Με άλλα λόγια, ακόμα και εάν και η λύση επιμέρους προβλημάτων είναι εύκολη, ο συνδυασμός των επιλύσεων μπορεί να μην είναι εφικτός ως λύση του συνολικού προβλήματος.

Ετσι, ενώ μία συλλογή τμημάτων φαινομενικά προσφέρει άπειρες δυνατότητες για τη δημιουργία μίας κατηγορίας συστημάτων, οι περιορισμοί μειώνουν σημαντικά τον αριθμό συστημάτων που δύνανται να συναρμολογηθούν. Παραδείγματα ικανοποίησης περιορισμών αποτελούν:

- Η συναρμολόγηση ενός ρολογιού.

- Το κόλλημα ενός σπασμένου πιάτου.
- Η συμπλήρωση ενός σταυρολέξου.
- Η αναγνώριση σχεδιαγραμμάτων τα οποία περιγράφουν γενικές όψεις αδιαφανών στερεών με επίπεδες πλευρές λευκού χρώματος, όπου ακριβώς τρεις έδρες και τρεις ακμές συναντώνται ώστε να δημιουργήσουν μία κορυφή. Ο φωτισμός είναι ομοιόμορφος ώστε να μην δημιουργούνται σκιές, ενώ δεν επιτρέπονται ρωγμές ή άλλες χαρακτηριστικές γραμμώσεις στις επιφάνειες.

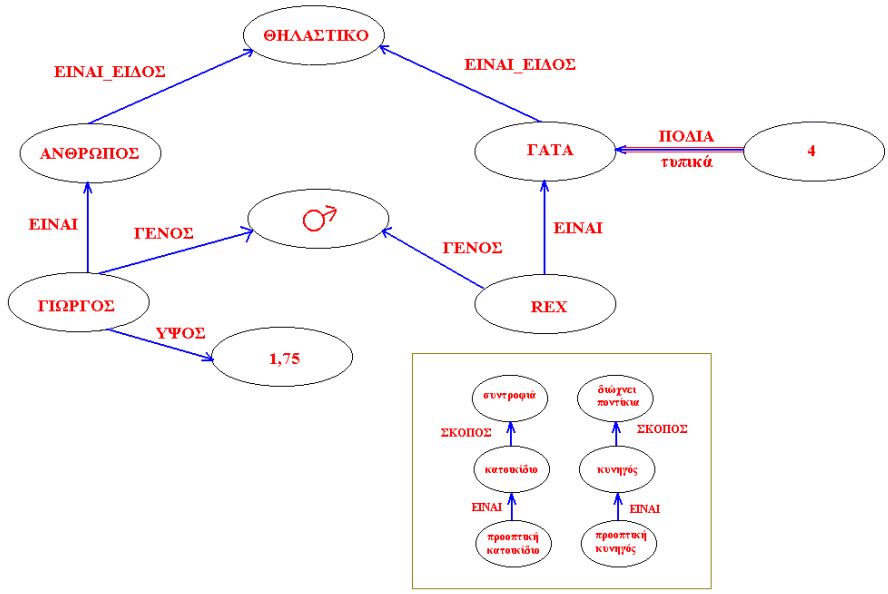
## 15.6 Χρήση Λογικής για την επίλυση προβλημάτων

Οι μορφές της λογικής οι οποίες περιγράφονται στην παράγραφο 2.3, και ιδιαίτερα η κατηγορική λογική, έχουν σημαντικές εφαρμογές στην υλοποίηση της καθημερινής γνώσης καθώς και στη συλλογιστική (reasoning). Για τον ίδιο σκοπό έχουν μάλιστα αναπτυχθεί γλώσσες λογικού προγραμματισμού (logic programming) οι οποίες είναι ιδιαίτερα κατάλληλες για την υλοποίηση αυτών των συστημάτων, υιοθετώντας ως δομή τους το τρόπο αναπαράστασης και σύνταξης (όρους και κατηγορήματα) της κατηγορικής λογικής. Οι δύο βασικότερες από αυτές της γλώσσες είναι η LISP και η PROLOG, οι οποίες λειτουργούν βασικά μέσω συμπλήρωσης/ταύτισης όρων στα κατηγορήματά τους.

### 15.6.1 Καθημερινή γνώση Σημασιολογικά δίκτυα (semantic nets)

Τα δίκτυα αυτά αποτελούν σχηματικό τρόπο έκφρασης των σχέσεων μεταξύ ομάδων αντικειμένων, αντικειμένων και χαρακτηριστικών τους. Οι ομάδες, τα αντικείμενα και τα χαρακτηριστικά τους αναπαριστώνται από κόμβους, ενώ οι σχέσεις μεταξύ τους εκφράζονται από κατευθυνόμενες ενώσεις μεταξύ των αντίστοιχων κόμβων. Ενα παράδειγμα δίνεται στο 15.1, για τις σχέσεις που περιγράφονται από το κείμενο: ”Ο άνθρωπος είναι θηλαστικό. Ο Γιώργος είναι άνδρας και έχει ύψος 1,75. Η γάτα είναι θηλαστικό και τυπικά έχει τέσσερα πόδια. Ο Rex είναι γάτος. Ως κατοικίδιο, σκοπός του Rex είναι η συντροφιά, ενώ ως κυνηγός, σκοπός του είναι να διώχνει τα ποντίκια.

Σχήμα 15.1: Σημασιολογικό Δίκτυο



Κλασσικοί μηχανισμοί αναπαράστασης σχέσεων στα σημασιολογικά δίκτυα αποτελούν:

(i) Κληρονομικότητα. Η γνώση η σχετική με τα χαρακτηριστικά μίας ομάδας αντικειμένων χρησιμοποιείται για τη γνώση σχετικά με τα αντικείμενα της ομάδας πχ Η κληρονομικότητα επιτυγχάνεται μέσω σχέσεων της μορφής ”είναι”, ”είναι είδος” κλπ.

(ii) Ανάθεση τυπικών (default) τιμών. Αποτελεί έκφραση της πιο πιθανής τιμής ενός χαρακτηριστικού ενός αντικειμένου. Πχ η τυπική τιμή για τον αριθμό ποδιών της γάτας είναι τέσσερα, άρα ο Rex μάλλον έχει τέσσερα πόδια, χωρίς αυτό να χρειαστεί να δηλωθεί ειδικά για τον Rex.

(iii) Χρήση δαιμόνων. Οι δαίμονες αποτελούν ρουτίνες παραγωγής πληροφορίας/χαρακτηριστικού αντικειμένων ή ομάδων αντικειμένων μέσα από άλλες πληροφορίες/χαρακτηριστικά τους. Οι δαίμονες χρησιμοποιούνται μόνο αν είναι αναγκαίο να συλλεχθεί αυτή η πληροφορία).

(iv) Προοπτική. Περιλαμβάνει τις διαφορετικές έννοιες κάτω από τις οποίες θεωρείται ένα αντικείμενο ή μία ομάδα αντικειμένων. Εφόσον ενεργοποιηθεί μία προοπτική αγνοούνται οι υπόλοιπες, ούτως ώστε να αποσαφηνιστεί η έννοια του αντικειμένου. Για παράδειγμα, στο Σχήμα 2 φαίνονται οι ”σκοποί” του Rex ως κατοικίδιο και ως κυνηγός.

**Σχήματα**

Τα σχήματα (schemata, frames, scenarios, scripts, Minsky (1975)) αποτελούν έναν από τους πιο συνήθεις τρόπους έκφρασης γνώσης σχετικής με αντικείμενα, ενέργειες ή γεγονότα. Αφορούν στην αναγνώριση και κατανόηση του περιβάλλοντος με δομημένο τρόπο. Εχουν ως πηγή έμπνευσής τους την ευελιξία της ΦΝ αφενός όταν αντιμετωπίζει μία νέα κατάσταση (η οποία μοιάζει με κάποια γνωστή κατάσταση) και αφετέρου όταν προσαρμόζεται στην αλλαγή μίας κατάστασης.

Τα σχήματα αποτελούν παραλλαγή των σημασιολογικών δικτύων: αντιπροσωπεύουν πραγματικές καταστάσεις εκφρασμένες συνολικά ως κόμβοι/θέσεις (slots) που περιέχουν/συμπληρώνονται από συγκεκριμένες τιμές και αντίστοιχες κατευθυνόμενες συνδέσεις. Ισχύουν οι κλασσικοί μηχανισμοί αναπαράστασης σχέσεων των σημασιολογικών δικτύων (κληρονομικότητα, τυπικές τιμές, δαίμονες και προοπτική), οπότε είναι δυνατό να μεταβάλλονται οι θέσεις-λεπτομέρειες όπως χρειάζεται, καθώς και να εμπεριέχονται (οπότε και να επεξηγούνται) λεπτομέρειες οι οποίες είναι χαρακτηριστικές αλλά παραλείπονται λόγω του ότι είναι αυτονόητες. Με τον τρόπο αυτό είναι δυνατό να υλοποιηθεί ο ελλειπτικός, καθημερινός, λόγος και - μέσω των σχημάτων - να αποσαφηνίζονται/συμπληρώνονται τα δεδομένα ώστε να δίνεται σωστή ερμηνεία σε περίπτωση ελλιπών δεδομένων ή πολλαπλών και αμφίσημων εκδοχών. Επιπλέον σχήματα μπορούν να περιέχουν επιμέρους (πιο εξειδικευμένα) σχήματα. Παραδείγματα σχημάτων αποτελούν η αναγνώριση μίας καρέκλας (μαζί με την ανάσυρση των χαρακτηριστικών της), το παιδικό πάρτυ γενεθλίων, μία βόλτα παίρνοντας το λεωφορείο, ή μία έξοδος πηγαίνοντας στο εστιατόριο. Μία υλοποίηση του σχήματος του εστιατορίου μπορεί να εκφραστεί με το εξής κείμενο:

Ο Γιάννης πήγε στο εστιατόριο.

Κάθησε.

Διάβασε το μενού.

Παρήγγειλε κοτόπουλο.

Δεν ήθελε ποτό.

Εφαγε το κοτόπουλο.

Αφησε μεγάλο φιλοδώρημα.

Εφυγε από το εστιατόριο.

Σχήμα 15.2: Σχήμα εστιατορίου. Χαρακτηριστικά και Θέσεις


Από το σχήμα 15.2 είναι φανερό ότι υπήρχε μενού και όχι μπουφές για την επιλογή του γεύματος, ότι έφαγε κοτόπουλο και ότι δεν ήπιε κρασί. Καθίσταται πιθανό ότι ο Γιάννης ήταν μόνος, ενώ συμπεραίνεται ότι του άρεσε το κοτόπουλο (αφού άφησε μεγάλο φιλοδώρημα). Το τελευταίο δίνεται από ένα ξεχωριστό, επιμέρους σχήμα, το σχήμα του φιλοδωρήματος που καθορίζει το ποσοστό του φιλοδωρήματος ανάλογο της ποιότητας του γεύματος. Παρ' όλο που δεν γίνεται

φανερό πώς ήρθε και πώς έφυγε ο Γιάννης, αν είχε κλείσει τραπέζι ή με ποιόν τρόπο πλήρωσε το λογαριασμό, η κατανόηση του ελλειπτικού κειμένου επιτυγχάνεται με ευκολία.

**Επεξεργασία φυσικής γλώσσας (natural language processing)**

Ορμώμενοι από την εξέταση του Turing, δημιουργούνται ολοένα και πιο εξελιγμένα προγράμματα (chatbots ή chatterbots ή crash dummies of communication) τα οποία επικοινωνούν με τον άνθρωπο σε φυσική γλώσσα με τέτοιον τρόπο ώστε να προσομοιώνουν επιτυχώς - για κάποιο χρονικό διάστημα - την ανθρώπινη συμπεριφορά και επικοινωνία. Αποτελούν σχετικά μικρές βάσεις δεδομένων (λέξεις της αγγλικής – συνήθως – γλώσσας) συνδυασμένα με σειρά κανόνων για τη δημιουργία ”ευφυών” προτάσεων. Τα προγράμματα αυτά έχουν απλή δομή συνήθως με τη μορφή κανόνων εκφρασμένων μέσω κατηγορικής λογικής, σημασιολογικών δικτύων και υλοποιημένα σε κάποια γλώσσα λογικού προγραμματισμού. Μπορούν να ξεγελάσουν τον συνομιλητή για περιορισμένο χρονικό διάστημα, αλλά - όσο εξεζητημένα και αν είναι - αποτυγχάνουν εύκολα (και συχνά) όταν αντιμετωπίζουν το ευρύτατο πεδίο ερωτήσεων/απαντήσεων του συνομιλητή τους (ΦΝ). Το ενδιαφέρον, ο συναγωνισμός, αλλά και η δυσκολία δημιουργίας ενός προγράμματος επικοινωνίας σε φυσική γλώσσα είναι τέτοια που κάθε χρόνο διεξάγεται ο διαγωνισμός Loebner για το πιο πετυχημένο πρόγραμμα.
Το κλασσικό πρόγραμμα επικοινωνίας σε φυσική γλώσσα είναι η ELIZA
(Weizenbaum, 1966), η οποία προσομοιώνει πειστικά (για περιορισμένο χρονικό διάστημα) έναν ψυχοθεραπευτή: διεισδυτική και ήρεμη, θέτει σύντομες αλλά κατάλληλες ερωτήσεις προκειμένου να κάνει τον ασθενή να ξανοιχτεί (πχ. ”Πώς σε κάνει αυτό να αισθάνεσαι;” ή ”Πες μου περισσότερα”). Χωρίς ουσιαστικό ίχνος εξυπνάδας (πχ. ευριστικών μεθόδων) στον πρόγραμμα, λειτουργεί ανιχνεύοντας λέξεις κλειδιά (keyword matching), πχ. ”μητέρα” ή ”καταπιεσμένος/η” προκειμένου να θέσει τις ανάλογες ερωτήσεις μέσα από μία βάση δεδομένων-ερωτήσεων. Αν η ανίχνευση λέξεων-κλειδιών αποτύχει, η ELIZA θέτει γενικές ερωτήσεις προκειμένου να συλλέξει περισσότερες πληροφορίες (και λέξεις-κλειδιά). Παρ’ όλα αυτά, εύκολα αποκαλύπτεται η περιορισμένη ικανότητά της για διάλογο.

Η ELIZABETH (Peter Millican), απόγονος της ELIZA με καλύτερη βάση δεδομένων και εκτεταμένο τρόπο αναζήτησης απαντήσεων. Η ELIZABETH μπορεί να χειρίζεται πιο πολύπλοκες εκφράσεις και μετασχηματισμούς. Ξεκινά τη συζήτηση με έναν τυχαία επιλεγμένο χαιρετισμό (από τους αποθηκευμένους) και οι απαντήσεις της βασίζονται αποκλειστικά στις προτάσεις του συνομιλητή της μέσω της παρακάτω διαδικασίας:

- Ανάλυση της πρότασης του συνομιλητή.
- Αναγνώριση λέξεων-κλειδιών στην πρόταση και τυχαία επιλογή ενός από τους υπάρχοντες μετασχηματισμούς τους.
- Δημιουργία απάντησης (ένα παράδειγμα δίνεται παρακάτω).

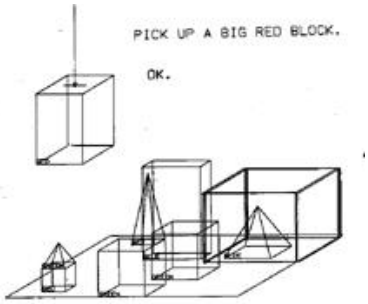
Ο SANTA (Rich Waugh and Rey Barry) για τη δημιουργία ενός προσομοιωτή του Αη Βασίλη που δέχεται παραγγελίες για πρωτοχρονιάτικα δώρα.

**Συστήματα γνώσης (knowledge-based systems)**

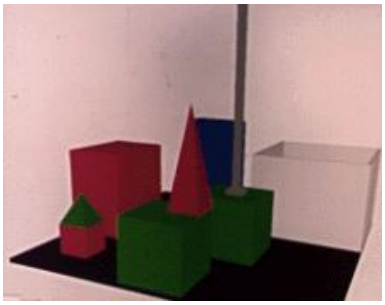
Δεδομένου του προβλήματος της καθημερινής γνώσης (όπως φάνηκε παραπάνω) αντί για την παραγωγή συστημάτων ΤΝ γενικής χρήσης, αναπτύχθηκε η εναλλακτική μεθοδολογία μίμησης της ΦΝ σε ”μικροκόσμους” (microworlds), δηλαδή περιορισμένα και καλά καθορισμένα περιβάλλοντα (χώρους προβλήματος). Η θεωρία πίσω από τους μικροκόσμους ήταν ότι η ΤΝ όφειλε να παράγει ευφυία σε ένα περιορισμένο περιβάλλον. Μόλις αυτό επιλυνόταν, η πολυπλοκότητα του περιβάλλοντος θα αυξανόταν (χαλάρωση περιορισμών περιβάλλοντος) προκαλώντας επέκταση στην ΤΝ. Αφού αυτό επαναλαμβανόταν διαδοχικά, η ΤΝ θα έφτανε σε ένα σημείο που θα επέλυνε πραγματικά προβλήματα και περιβάλλοντα. Ηταν μία θεωρία αμοιβαία κλιμακούμενης πολυπλοκότητας περιβάλλοντος και ΤΝ.

Ενα από τα πιο κλασσικά παραδείγματα μικροκόσμου αποτελεί ο μικρόκοσμος των κύβων με διαχείρισή του από το SHRDLU (Winograd, 1972), ένα σύστημα γνώσης το οποίο στηρίζεται σε κανόνες της μορφής που περιγράφηκε στην παράγραφο 4.2.1. Το SHRDLU έλαβε το όνομά του από τα επτά πιο συνηθισμένα γράμματα της Αγγλικής γλώσσας. Αποτελεί συνδυασμό επικοινωνίας σε φυσική γλώσσα αφενός και επίλυσης προβλημάτων μετακίνησης και τοποθέτησης αντικειμένων του τεχνητού μικροκόσμου των κύβων (κύβων, σφαιρών, κώνων, πυραμίδων κλπ. διαφόρων μεγεθών και χρωμάτων, βλ. 15.3,15.6.1) αφετέρου. Περιγράφει κύβους και έναν ρομποτικό βραχίονα που χρησιμοποιείται για την μετακίνηση των κύβων. Είναι ενδιαφέρον ότι τέσσερις απλές ιδέες επιτρέπουν την ικανοποιητική προσομοίωση της κατανόησης και επικοινωνίας με το περιβάλλον/άνθρωπο και εισήγαγαν ένα πρώτο δείγμα ρομποτικής. Οι τέσσερις αυτές ιδέες είναι οι εξής:

Σχήμα 15.3: SHRLDU




- Περιορισμένο και καλά καθορισμένο περιβάλλον. 50 λέξεις είναι αρκετές για την περιγραφή και διαχείρισή του περιβάλλοντος.
- Ενσωμάτωση μία πρώτης έννοιας εκμετάλλευσης του προηγηθέντος διαλόγου για την κατανόηση αμφίβολων εντολών. Αρα πραγματοποιείται φυσική επεξεργασία λόγου και διάλογος με σκοπό την εκτέλεση της εντολής.
- Ελεγχος της δυνατότητας εκτέλεσης εντολής. Λόγω της μνήμης του σχετικά με τον προηγηθέντα διάλογο, μπορεί να διαπιστώσει αν κάτι είναι εφικτό ή όχι. Αρα, σε κάθε βήμα, πραγματοποιείται επίλυση του προβλήματος που εκφράζεται μέσω της εντολής.



- Εκμάθηση. είναι δυνατή η αποθήκευση νέων ονομάτων σε αντικείμενα ή σε συνδυασμούς τους.

### 15.6.2 Εμπειρη γνώση

Αν και το SHRLDU αποτέλεσε ένα ιδιαίτερα πετυχημένο πρόγραμμα ΤΝ, η επέκτασή του σε λιγότερο περιορισμένα περιβάλλοντα ήταν αποτυχής. Μάλιστα, ο Winograd πρώτος αποστασιοποιήθηκε από το κατασκεύασμά του, πιστεύοντας ότι η ΤΝ οδηγεί σε αδιέξοδο. Παρόλα αυτά, το SHRLDU απέδειξε ότι κατάλληλα δομημένα συστήματα ΤΝ μπορούν να δρουν ως ειδικοί σε καλά καθορισμένα περιβάλλοντα (βαθιά αλλά περιορισμένη γνώση ειδικού αντί για πλατιά και ρηχή καθημερινή γνώση). Ετσι, ένας κλάδος της ΤΝ αφιερώθηκε στην έμπειρη γνώση.

#### Συλλογιστική - συστήματα κανόνων (rule-based systems)

Λόγω της πολυπλοκότητας και έκτασής της, η έμπειρη γνώση δεν υλοποιείται συνήθως μέσα από σημασιολογικά δίκτυα ή σχήματα (τα οποία έχουν καθαρά δηλωτική μορφή) αλλά μέσα από κανόνες που επεξηγούν τι επακολουθεί (συμπέρασμα ή πράξη) εφόσον ένα σύνολο προϋποθέσεων ικανοποιείται. Οι κανόνες αυτοί είναι της μορφής:

<ΚΑΝΟΝΑΣ i> <ΠΡΟΫΠΟΘΕΣΗ i1> ΑΛΗΘΗΣ
<ΠΡΟΫΠΟΘΕΣΗ i2> ΑΛΗΘΗΣ
...

<ΠΡΟΫΠΟΘΕΣΗ in> ΑΛΗΘΗΣ
ΤΟΤΕ <ΕΠΑΚΟΛΟΥΘΟ i1> ΑΛΗΘΕΣ
< ΕΠΑΚΟΛΟΥΘΟ i2> ΑΛΗΘΕΣ
...
< ΕΠΑΚΟΛΟΥΘΟ im> ΑΛΗΘΕΣ

όπου οι προϋποθέσεις αποτελούν τα απαιτούμενα προκειμένου να ενεργοποιηθεί ο κανόνας, ενώ τα επακόλουθα είναι τα αποτελέσματα της ενεργοποίησης του κανόνα. Οι προϋποθέσεις συνδέονται μεταξύ τους με λογικούς τελεστές ( V και Λ του Πίνακα 1). Η ικανοποίησή τους ή η μή

ικανοποίησή τους εξαρτάται από την ικανοποίηση ή τη μη ικανοποίηση του λογικού συνδυασμού των προϋποθέσεων, οπότε προκαλείται η ενεργοποίηση ή η μη ενεργοποίηση των επακολούθων αντίστοιχα. Τα συστήματα κανόνων χρησιμοποιούνται για την κωδικοποίηση της γνώσης και την επαλήθευση/διάψευση κάποιων εικασιών. Τα συστήματα κανόνων χαρακτηρίζονται ως:

- Συστήματα σύνθεσης (synthesis systems), εφόσον δημιουργούν/συνθέτουν ένα σύνολο αποτελεσμάτων, όπως αυτό προκύπτει από κάποια δεδομένα/γεγονότα.
- Συστήματα ανάλυσης (analysis systems), εφόσον προσπαθούν να κατηγοριοποιήσουν/διαγνώσουν κάποια δεδομένα/γεγονότα βάσει των διαθέσιμων κανόνων.

Για την σύνθεση/ανάλυση δεδομένων μέσα από την επαλήθευση/διάψευσή τους, τα συστήματα κανόνων απαρτίζονται από:

- Σύνολο κανόνων.
- Σύνολο γεγονότων/δεδομένων/αληθειών σχετικά με κάποια αντικείμενα ή χαρακτηριστικά τους, το οποίο εκφράζεται ως βάση δεδομένων (database). Η βάση αυτή εμπλουτίζεται μέσα από ερωτήσεις του συστήματος στον χρήστη καθώς και μέσα από τα αποτελέσματα/συμπεράσματα των ενεργοποιημένων κανόνων.
- Σύνολο συμπερασμάτων (εικασιών), τα οποία πρέπει να επαληθευτούν ή να διαψευσθούν.
- Σύνολο διαδικασιών ελέγχου που καθορίζουν τη συλλογιστική (reasoning strategy) και καλούν τη συμπερασματική μηχανή (inference engine) για τη δημιουργία νέων γεγονότων που εμπλουτίζουν τη βάση δεδομένων και οδηγούν στην επαλήθευση ή τη διάψευση των συμπερασμάτων.

Η συλλογιστική καθορίζει τον τρόπο με τον οποίο το σύστημα κανόνων οδηγείται μέσα από τα δεδομένα και τους κανόνες στην επαλήθευση ή στη διάψευση των συμπερασμάτων. Βασικό πλεονέκτημά της αποτελεί η δυνατότητα επεξήγησης της πορείας προς την επαλήθευση ή τη διάψευση, για παράδειγμα το πώς διαπιστώθηκε κάποιο γεγονός (προϋπόθεση, επακόλουθο ή συμπέρασμα) καθώς και το γιατί ερωτήθηκε ο χρήστης σχετικά με κάποιο γεγονός. Για την επίτευξη της σύνθεσης/ανάλυσης υπάρχουν δύο είδη συλλογιστικής, η συλλογιστική προς τα εμπρός και η συλλογιστική προς τα πίσω.

*Η συλλογιστική προς τα εμπρός*

Ξεκινώντας από γεγονότα και δεδομένα της βάσης δεδομένων χρησιμοποιούνται οι κανόνες προκειμένου να δημιουργηθούν νέα γεγονότα και δεδομένα (επακόλουθα) τα οποία εμπλουτίζουν τη βάση δεδομένων. Η διαδικασία αυτή συνεχίζεται μέχρις ότου είτε να προκύψει το ζητούμενο συμπέρασμα είτε να καταστεί βέβαιο ότι αυτό δε μπορεί να προκύψει. Ετσι, η συλλογιστική προς τα εμπρός χρησιμοποιείται όταν ζητείται τι μπορεί να προκύψει από ένα σύνολο κανόνων και κάποια αρχική βάση δεδομένων (δηλαδή για συστήματα σύνθεσης). Εκφράζεται ως εξής:

**Βήμα 1.** Για κάθε κανόνα,

**Βήμα 1.1.** Απόπειρα ικανοποίησης του συνδυασμού των προϋποθέσεων βάσει των γεγονότων/δεδομένων της βάσης δεδομένων. Ελεγχος του εάν ο κανόνας ενεργοποιείται.
Εάν ναι,

**Βήμα 1.1.1.** Προσθήκη των νέων επακόλουθων στη βάση δεδομένων.

**Βήμα 2.** Διαπίστωση του εάν η βάση δεδομένων έχει αυξηθεί.

Εάν ναι,

**Βήμα 2.1.** Επιστροφή στο *Βήμα 1*.

Εάν όχι,

**Βήμα 2.2.** Τέλος μεθόδου. Αναφορά όλων των νέων επακόλουθων της βάσης δεδομένων.

Η συλλογιστική προς τα εμπρός είναι εξαντλητική. Αν και είναι δυνατό να τίθενται ερωτήσεις αληθείας στον χρήστη κάθε φορά που δεν ικανοποιείται μία προϋπόθεση, κάτι τέτοιο δεν είναι κατευθυνόμενο (άρα δεν είναι και ιδιαίτερα αποδοτικό). Γι' αυτόν τον λόγο, έχει αναπτυχθεί σειρά στρατηγικών (conflict resolution strategies) οι οποίες επιτρέπουν στη συμπερασματική μηχανή να αυξήσει την αποδοτικότητά της. Οι στρατηγικές αυτές περιλαμβάνουν:

- Την εξέταση των πιο γενικών κανόνων πρώτα (προκειμένου να εμπλουτιστεί η αρχική βάση δεδομένων).
- Τη μη εφαρμογή του ίδιου κανόνα στα ίδια δεδομένα.
- Την εφαρμογή των κανόνων στα πιο πρόσφατα στοιχεία της βάσης δεδομένων, έτσι ώστε να επαυξηθεί η βάση δεδομένων και να προχωρήσει η συλλογιστική συστηματικά/κατευθυνόμενα.
- Την προτίμηση για εφαρμογή των πιο ειδικών κανόνων έναντι των πιο γενικών (κατά την πορεία της συλλογιστικής διαδικασίας).

Σημειώνεται πάντως ότι τα πιο σημαντικό στοιχείο του συστήματος κανόνων αποτελεί ο τρόπος σύνταξης των προϋποθέσεων-επακολούθων των κανόνων καθώς και επιπλέον διαδικασίες ελέγχου της συμπερασματικής μηχανής. Οι τελευταίες εξαρτώνται από το εκάστοτε πρόβλημα προς επίλυση (problem-dependent) και δύνανται να καθορίζουν το πεδίο δράσης των στρατηγικών πχ. ότι αρχικά θα προτιμούνται οι πιο γενικοί κανόνες ώστε να εμπλουτιστεί ικανοποιητικά η βάση δεδομένων και κατόπιν θα εφαρμοστεί η στρατηγική της προτίμησης των πιο ειδικών κανόνων προκειμένου η αναζήτηση να επικεντρωθεί και καταστεί πιο κατευθυνόμενη.

Αρχικά αναζητείται κάποιος κανόνας ο οποίος περιέχει το ζητούμενο συμπέρασμα ως επακόλουθο. Αφού εντοπιστούν οι προϋποθέσεις οι οποίες πρέπει να ικανοποιηθούν προκειμένου να ενεργοποιηθεί ο κανόνας, πραγματοποιείται εκ νέου αναζήτηση των κανόνων οι οποίοι περιέχουν αυτές τις προϋποθέσεις και η διαδικασία συνεχίζεται μέχρις ότου είτε οι προϋποθέσεις ανάγονται σε αληθή γεγονότα/δεδομένα της βάσης δεδομένων (οπότε το ζητούμενο επαληθεύεται)

είτε καταστεί φανερό ότι οι προϋποθέσεις αυτές δεν είναι δυνατό να ικανοποιηθούν (οπότε το ζητούμενο διαψεύδεται). Λόγω του ότι είναι πιο κατευθυνόμενη, η συλλογιστική προς τα πίσω χρησιμοποιείται εάν το ζητούμενο συμπέρασμα είναι εξειδικευμένο (πχ. μία υπόθεση που αφορά σε κάποια συγκεκριμένα δεδομένα της βάσης δεδομένων) και ζητείται η επαλήθευσή/διάψευσή του. Εφαρμόζεται ως εξής:

**Βήμα 1.** Για κάθε πρωτογενές συμπέρασμα.

**Βήμα 1.1.** Εξέταση κάθε κανόνα που περιέχει το πρωτογενές συμπέρασμα ως συμπέρασμα:

**Βήμα 1.1.1.** Ελεγχος του εάν ο κανόνας ενεργοποιείται. (απόπειρα ικανοποίησης του συνδυασμού των προϋποθέσεων βάσει των γεγονότων/δεδομένων της βάσης δεδομένων).

Εάν ναι,

**Βήμα 1.1.1.1.** Αναφορά του πρωτογενούς αποτελέσματος.

Εάν όχι,

**Βήμα 1.1.1.2.** Διαγραφή του πρωτογενούς αποτελέσματος.

Εάν δεν είναι δυνατός ο έλεγχος (κάποιες προϋποθέσεις δεν είναι γνωστό αν ικανοποιούνται)

**Βήμα 1.1.1.2.** Αναδρομικός έλεγχος του εάν η κάθε προϋπόθεση ικανοποιείται.

Οπως και στη συλλογιστική προς τα εμπρός, είναι δυνατό να τίθενται ερωτήσεις αληθείας στον χρήστη κάθε φορά που δεν ικανοποιείται μία προϋπόθεση. Ομως, αντί της συνεχώς επαυξανόμενης βάσης δεδομένων της συλλογιστικής προς τα εμπρός, στη συλλογιστική προς τα πίσω αυξάνεται ο αριθμός των προϋποθέσεων που πρέπει να ικανοποιηθούν προκειμένου να επαληθευτεί το ζητούμενο συμπέρασμα.. Στη συλλογιστική προς τα πίσω, οι ερωτήσεις προς τον χρήστη μοιάζουν κατευθυνόμενες, μιας και γίνονται όλο και πιο λεπτομερείς. Γι' αυτό το λόγο, η συλλογιστική αυτή είναι πιο αποδοτική από τη συλλογιστική προς τα εμπρός.

Καθώς η συλλογιστική προς τα εμπρός παρουσιάζει προβλήματα συνδυαστικής έκρηξης σε μεγάλες βάσεις δεδομένων και για πολλούς κανόνες, αλλά είναι ιδιαίτερα αποτελεσματική και διεξοδική σε μικρές βάσεις δεδομένων και για λίγους κανόνες, είναι δυνατός ο συνδυασμός των δύο συλλογιστικών οπότε χρησιμοποιείται αρχικά η συλλογιστική προς τα πίσω για την απόδειξη πιο γενικών προϋποθέσεων ακολουθούμενη από τη συλλογιστική προς τα εμπρός για την απόδειξη του ζητούμενου.

Εμπειρα συστήματα (ΕΣ, expert systems)

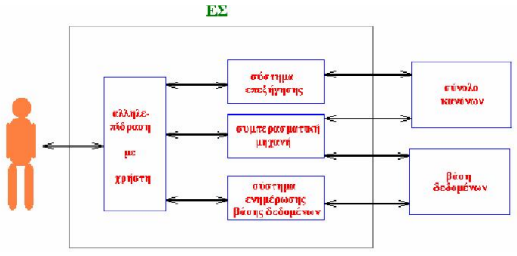
Η δομή των ΕΣ είναι ίδια με αυτή των συστημάτων κανόνων. Ετσι, ένα ΕΣ πρέπει να μπορεί να επεξηγεί τον τρόπο με τον οποίο φτάνει σε ένα αποτέλεσμα καθώς και να μπορεί να απαντά σε ερωτήσεις σχετικά με αυτό. Η κύρια διαφορά των ΕΣ και των συστημάτων κανόνων έγκειται:

(α) Στο θέμα το οποίο επιλύουν. Ενώ αυτό είναι γενικότερο για τα συστήματα κανόνων, τα ΕΣ προσεγγίζουν προβλήματα σε σημαντικούς και πολύ εξειδικευμένους/περιορισμένους τομείς (πχ. ιατρική, μαθηματικά, μηχανολογία, γεωλογία, επιστήμη υπολογιστών, επιχειρήσεις, νομική, άμυνα, εκπαίδευση), τα οποία απαιτούν έναν ή περισσότερους ειδικούς (πχ. γιατρό, ορυκτολόγο). Τα προβλήματα αυτά μπορεί να εκφράζονται ως προβλήματα διάγνωσης (πάθηση, μηχανική βλάβη, λάθος μαθητή κλπ.), κατασκευαστικά προβλήματα (ενός κτιρίου ή υπολογιστικού συστήματος), ή προβλήματα επεξήγησης (πχ. γεωλογικών δεδομένων). Τα συμπεράσματα στα οποία καταλήγουν τα ΕΣ αποτελούν αποφάσεις με τη μορφή παροχής εξειδικευμένης γνώσης, διάγνωσης και συμβουλών.

(β) Στην πολυπλοκότητά της δομής τους. Η κωδικοποίηση της πιο βαθιάς, εξειδικευμένης γνώσης είναι ευριστικής μορφής, βασισμένη περισσότερο σε εμπειρικούς κανόνες (οι οποίοι είναι διαισθητικοί) παρά σε συγκεκριμένα γεγονότα ή βεβαιότητες. Ετσι, απαιτείται όχι μόνο η έκφραση διαισθητικών κανόνων οι οποίοι απομακρύνονται αισθητά από τη δηλωτική γνώση, αλλά και στη σύνθεση κανόνων οι οποίοι εκφράζουν βαθιά, διαδικαστική γνώση του θέματος το οποίο επιλύουν. Συνεπώς, ο τρόπος αναπαράστασης της γνώσης καθώς και της συλλογιστικής σε ένα ΕΣ εξαρτάται κατά πολύ από το συγκεκριμένο πρόβλημα.

Η δημιουργία ενός ΕΣ απαιτεί την εξαγωγή γνώσης (knowledge acquisition) από τον ειδικό, γνώση η οποία είναι δύσκολο όχι μόνο να εξαχθεί αλλά και να αναπαρασταθεί με τέτοιο τρόπο ώστε να καταστεί διαχειρίσιμη από ένα σύστημα ΤΝ υλοποιημένο σε Η/Υ. Η δημιουργία ενός ΕΣ αποτελεί την ευθύνη του μηχανικού γνώσης (knowledge engineer), ενός εξειδικευμένου προγραμματιστή ο οποίος συνεργάζεται αφενός με τον ειδικό και αφετέρου με τους χρήστες προκειμένου να εξάγει τη γνώση και να υλοποιήσει κατάλληλα το σύστημα (από άποψη πληροφορίας, κατάλληλης γλώσσας προγραμματισμού, δομής ΕΣ και ικανοποιητικής αλληλεπίδρασης με τον χρήστη). Οι πρώτες προσπάθειες δημιουργίας του ΕΣ (έμπειρης γνώσης και αντίστοιχων κανόνων) είναι σχεδόν πάντα ανεπιτυχείς. Απαιτείται σειρά από συνεχώς βελτιούμενα ΕΣ τα οποία αφενός προσεγγίζουν τον τρόπο σκέψης και λήψης αποφάσεων του ειδικού και αφετέρου έχουν τη δυνατότητα (φαινομενικά) αβίαστης αλληλεπίδρασης με τον χρήστη όσον αφορά στις ερωτήσεις του ΕΣ προκειμένου να συλλεγεί πληροφορία καθώς και στις ερωτήσεις του χρήστη σχετικά με τις αποφάσεις του ΕΣ. Το 15.4 παρουσιάζει τα κυριότερα μέρη ενός ΕΣ.

Σχήμα 15.4: Βασική Δομή ΕΣ



Λόγω του ότι η δημιουργία ενός ΕΣ απαιτεί πολύ χρόνο καθώς και υπολογιστική προσπάθεια, πρέπει να πληρούνται τα εξής προκειμένου να αποφασιστεί η υλοποίησή του:

(α) Τα οφέλη πρέπει τουλάχιστον να αντισταθμίζουν το κόστος κατασκευής του ΕΣ.

(β) Η έμπειρη γνώση πρέπει να μη είναι εύκολα προσβάσιμη αλλά να είναι απαραίτητη ή ιδιαίτερα χρήσιμη σε πληθώρα περιπτώσεων. Εφόσον η εξειδικευμένη γνώση είναι εύκολο να βρεθεί τότε μάλλον δεν είναι απαραίτητο το ΕΣ, ενώ εάν η γνώση είναι πολύ εξειδικευμένη και η πρόσβαση σε αυτή δύσκολη ή/και ακριβή τότε η δημιουργία ενός ΕΣ είναι συμφέρουσα.

(γ) Το πρόβλημα πρέπει να μην αφορά σε κινητική επιδεξιότητα (καθαρά διαδικαστική γνώση), η οποία δε μπορεί να εκφραστεί μέσω δηλωτικής γνώσης (άρα κατηγορικής λογικής, σχημάτων και κανόνων).

(δ) Το πρόβλημα πρέπει να είναι καλά δομημένο και να μη εμπεριέχει πολλή καθημερινή γνώση, η οποία είναι δύσκολο να περιγραφεί και αναπαρασταθεί. Με άλλα λόγια, η δηλωτική γνώση καλά καθορισμένων και περιορισμένων πεδίων ενδείκνυται για ΕΣ, ενώ η πλατιά, γενικευμένη, εμπειρική γνώση όχι.

(ε) Πρέπει να μην είναι δυνατή η επίλυση του προβλήματος με άλλες υπολογιστικές τεχνικές. Αν, για παράδειγμα, υπάρχει κάποιος αλγόριθμος ο οποίος υλοποιεί τη λύση, τότε το ΕΣ δεν ενδείκνυται (λόγω κόστους, χρόνου – πολυπλοκότητας - και δυσκολίας κατασκευής).

(στ) Το πρόβλημα πρέπει να έχει κατάλληλο μέγεθος, δηλαδή να επαρκεί η γνώση ενός ή λίγων ειδικών προκειμένου να επιλυθεί το πρόβλημα σε μικρό χρονικό διάστημα (το πολύ σε μία ώρα).

(ζ) Οι ειδικοί οι οποίοι θα προμηθεύσουν τη γνώση πρέπει να είναι πρόθυμοι να συνεργαστούν, ενώ πρέπει να έχουν επιπλέον την ικανότητα μετάδοσης της γνώσης τους. Εφόσον ένας ειδικός αισθάνεται ότι απειλείται από το έτοιμο ΕΣ, προφανώς δεν θα μεταδώσει τη γνώση του σωστά και με σαφή τρόπο.

Από τα παραπάνω περιορίζεται σημαντικά ο αριθμός των προβλημάτων των οποίων η επίλυση μέσω ΕΣ κρίνεται σκόπιμη. Για τέτοια προβλήματα όμως, τα οφέλη είναι σημαντικότατα (οικονομία και αμεσότητα στην επίλυση).

**Κλασσικό ΕΣ (MYCIN)**

Το MYCIN αποτελεί ένα από τα πρώιμα - και πλέον κλασσικά - ΕΣ, το οποίο έχει επηρεάσει το σχεδιασμό των περισσοτέρων εμπορικών ΕΣ. Αποσκοπεί αφενός στη διάγνωση ορισμένων μολυσματικών ασθενειών του αίματος και αφετέρου στον καθορισμό της κατάλληλης αντιμικροβιακής θεραπείας. Επιπλέον διαθέται τη δυνατότητα λεπτομερούς επεξήγησης των αποφάσεων που λαμβάνει. Λόγω του ότι για τη ακριβή διάγνωση των μολυσματικών ασθενειών αίματος

τις οποίες χειρίζεται το MYCIN απαιτείται καλλιέργεια του αίματος του ασθενούς (τα αποτελέσματα της οποίας προκύπτουν μετά από 48 ώρες, χρόνος μέσα στον οποίο ο ασθενής ενδέχεται να έχει ήδη πεθάνει), είναι απαραίτητο η διάγνωση να πραγματοποιηθεί άμεσα έστω και με περιορισμένα δεδομένα (ελλιπείς πληροφορίες). Η προτεινόμενη θεραπεία αναγκαστικά λοιπόν καλύπτει μία ομπρέλλα από πιθανές αιτίες των συμπτωμάτων. Επιπλέον, η επιλογή της θεραπείας βασίζεται στην ευαισθησία του οργανισμού, στον εντοπισμό και το είδος της μόλυνσης καθώς και στην αποτελεσματικότητα του φαρμάκου.

Το MYCIN αποτελεί ΕΣ ανάλυσης μέσω συλλογιστικής προς τα πίσω με αναζήτηση σε βάθος και οπισθοδρόμηση. Απαρτίζεται από περίπου 500 κανόνες, όπου κάθε κανόνας συνοδεύεται από έναν βαθμό βεβαιότητας ο οποίος καθορίζει το πόσο ισχυρή είναι σχέση προϋποθέσεων-επακολούθων. Ο βαθμός βεβαιότητας λαμβάνει τιμές μέσα από το διάστημα [0,+1], όπου το 0 χαρακτηρίζει την πλήρη έλλειψη βεβαιότητας στο επακόλουθο και το 1 χαρακτηρίζει την πλήρη βεβαιότητα σε αυτό. Η γλώσσα προγραμματισμού η οποία χρησιμοποιήθηκε για την υλοποίηση του MYCIN είναι η LISP, άρα το ΕΣ εκφράζεται – στο κατώτερο επίπεδό του - καθαρά μέσω κατηγορικής λογικής, με κάθε κανόνα να αναπαριστάται ως μία έκφραση της κατηγορικής λογικής. Ετσι είναι δυνατή η χρήση δεδομένων καθώς και αποτελεσμάτων άλλων κανόνων ως όροι στο κατηγόρημα-έκφραση του αντίστοιχου κανόνα, κάτι που προσδίδει μεγάλη ευελιξία και οικονομία έκφρασης στο ΕΣ. Μειονέκτημα αυτής της αναπαράστασης αποτελεί η πυκνή επεξηγηματική ικανότητα του MYCIN η οποία μειώνει τη διαφάνειά του κατά τη συλλογιστική και επεξηγηματική λειτουργία του.

**Ασαφή Έμπειρα Συστήματα(fuzzy expert systems)**

Τα ασαφή ΕΣ αποτελούν επέκταση των ΕΣ της παραγράφου 4.2.2. Βασική πηγή έμνευσής τους είναι η ικανότητα της ΦΝ να χειρίζεται ανακριβή, αμφίσημα, ελλιπή, ποιοτικά ή γλωσσικά δεδομένα και πληροφορίες. Ετσι, τα ασαφή ΕΣ διαχειρίζονται τις μεταβλητές μέσω ασαφών (αντί δίτιμων) χαρακτηριστικών.

Ενα ασαφές ΕΣ έχει την ίδια δομή και τρόπο λειτουργίας με ένα κλασσικό ΕΣ. Η βάση δεδομένων του απαρτίζεται από κανόνες της μορφής ΕΑΝ-ΤΟΤΕ. Οι μεταβλητές-προϋποθέσεις λαμβάνουν κάποια ακριβή τιμή (ή κάποιο συγκεκριμένο διάστημα τιμών), η οποία εκφράζεται μέσω ποσοστών συμμετοχής από τις αντίστοιχες συναρτήσεις συμμετοχής που περιέχουν τη συγκεκριμένη τιμή. Αυτό επιτρέπει στους κανόνες να εκφράζονται ποιοτικά, με τρόπο που προσομοιώνει τη γλωσσική έκφραση της ΦΝ.

Τα παρακάτω στάδια ακολουθούνται για την δημιουργία και λειτουργία ενός ασαφούς ΕΣ:

- Στάδιο ασαφοποίησης (fuzzification stage). Ολες οι μεταβλητές του προβλήματος (σχετικές είτε με προϋποθέσεις είτε με επακόλουθα) εκφράζονται μέσω της ασαφούς λογικής (μέσω συναρτήσεων και ποσοστών συμμετοχής).

- Συμπερασματικό στάδιο (inference stage). Οι ΕΑΝ-ΤΟΤΕ κανόνες εκφράζουν ποιοτικά τη σχέση μεταξύ προϋποθέσεων και επακολούθων ως εξής:

  ΕΑΝ (Ι1 είναι mfΙ1i) ΑΛΗΘΗΣ Λ/V

(I 2 είναι mfI2j) ΑΛΗΘΗΣ Λ/V
... Λ/V
(I n είναι mfInk) ΑΛΗΘΗΣ
ΤΟΤΕ (Ο είναι mfOl)

όπου I1, I2,…, In είναι οι ασαφώς εκφρασμένες προϋποθέσεις, Ο είναι το ασαφώς εκφρασμένο επακόλουθο, και mfI1i, mfI2j,…, mfInk και mfOl είναι οι αντίστοιχες συναρτήσεις συμμετοχής των μεταβλητών που σχετίζονται με τον κανόνα. Οι βαθμοί βεβαιότητας των προϋποθέσεων υπολογίζονται από τα ποσοστά συμμετοχής τους στις αντίστοιχες συναρτήσεις συμμετοχής, και συνδυάζονται μέσω των λογικών τελεστών MIN ή MAX για Λ και V λογικούς τελεστές, αντίστοιχα (όπως στο MYCIN) ώστε να αποδώσουν το βαθμό βεβαιότητας του επακολούθου Ο. Αυτός μετασχηματίζει τη συνάρτηση συμμετοχής mfOl με κάποιον τρόπο

- Φάση σύνθεσης (composition stage). Οι κανόνες οι οποίοι αναφέρονται στην ίδια μεταβλητή-επακόλουθο και των οποίων οι βαθμοί βεβαιότητας (για τις διάφορες συναρτήσεις συμμετοχής είναι μη μηδενικοί συνδυάζονται (πχ. αθροίζονται), δημιουργώντας τρόπον τινά μία σύμπλοκη συνάρτηση συμμετοχής.

- Φάση απο-ασαφοποίησης (de-fuzzification stage). Κάθε επακόλουθο λαμβάνει μία τιμή (crisp value) η οποία προέρχεται από την μίας τιμής (πχ. το κέντρο βάρους της σύμπλοκης συνάρτησης συμμετοχής.

Τα βασικά πλεονεκτήματα των ασαφών ΕΣ έναντι των ΕΣ επικεντρώνουν στην:

(α) Ευρωστία λειτουργίας του ασαφούς ΕΣ. Η λειτουργία καθίσταται δυνατή ακόμη και με ελλιπή ή ανακριβή δεδομένα. Η ομαλή μετάβαση μεταξύ διαφορετικών επακολούθων ενισχύει τη διαφάνεια του ασαφούς ΕΣ καθώς και την εφαρμογή του σε προβλήματα ελέγχου.

(β) Ευελιξία. Η απόδοση του συστήματος είναι εύκολο να μεταβληθεί, πχ. μέσω αλλαγής του αριθμού, των ορίων ή των μορφών των συναρτήσεων συμμετοχής των μεταβλητών ή μέσω προσθήκης/διαγραφής/μετατροπής των κανόνων.

(γ) Οικονομία. Συνήθως, μικρός αριθμός κανόνων επαρκεί για ικανοποιητική λειτουργία.

## 15.7 Μηχανική Εκμάθηση

Ενα από τα σκληρότερα επιχειρήματα των πολέμιων της ΤΝ είναι ότι τα συστήματα ΤΝ δεν μπορούν να μάθουν έτσι ώστε να είναι ικανά να αλλάξουν και να προσαρμοστούν στις απαιτήσεις του περιβάλλοντός τους. Η εκμάθηση του περιβάλλοντος προκειμένου να επιτευχθεί η αλληλεπίδραση με αυτό αποτελεί σημαντικό βήμα στην ΤΝ.

### 15.7.1 Στρατηγικές Εκμάθησης τεχνικών ΤΝ

Για την εκπαίδευση ενός συστήματος ΤΝ είναι απαραίτητη η ύπαρξη διδάσκοντος, δηλαδή η παρέμβαση του χρήστη/προγραμματιστή στο σύστημα ΤΝ. Κάποιες στρατηγικές εκμάθησης συστημάτων ΤΝ αποτελούν:

- Αποστήθιση (rote learning). Με αυτή τη στρατηγική δεν εκτελείται καμία συμπερασματική διαδικασία ή άλλου είδους μετασχηματισμός της γνώσης του προβλήματος/περιβάλλοντος από το σύστημα ΤΝ.

- Εκμάθηση μέσω ανάθεσης γνώσης (learning by instruction). Η γνώση του διδάσκοντος μετασχηματίζεται (αναπαριστάται έτσι ώστε να βρίσκεται) σε μία μορφή η οποία (α) είναι επεξεργάσιμη από το σύστημα ΤΝ, και (β) μπορεί να ενσωματωθεί στην υπάρχουσα γνώση. Με άλλα λόγια, καθίσταται και πάλι αναγκαία η επιλογή κατάλληλης αναπαράστασης (μορφής) της γνώσης από τον διδάσκοντα. Η γνώση είναι δηλωτικής μορφής, άρα μπορεί να εκφράζεται μέσω παραδειγμάτων, σχημάτων κλπ

- Εκμάθηση μέσω αναλογιών (learning by analogy). Πραγματοποιείται εκμάθηση όχι παραδειγμάτων ή γνώσης αλλά της σχέσης που αυτά έχουν με αποθηκευμένη γνώση σε κάποιο άλλο πεδίο ή περιβάλλον. Απαιτείται σημαντικά περισσότερη συμπερασματική διαδικασία από ότι στις προηγούμενες δύο στρατηγικές εκμάθησης, αφού το σύστημα ΤΝ εκτελεί τους μετασχηματισμούς της γνώσης από το ένα πεδίο στο άλλο ανεξάρτητα από τον διδάσκοντα (ο οποίος απλά παρουσιάζει τα παραδείγματα ή την γνώση).

- Εκμάθηση μέσα από παραδείγματα (learning from examples). Βασισμένη στην επαγωγική γνώση, η στρατηγική αυτή αποτελεί τον πιο συνηθισμένο τρόπο εκμάθησης της ΤΝ. Ο διδάσκων παράγει ένα σύνολο παραδειγμάτων (θετικών μόνο ή θετικών καθώς και αρνητικών). Το σύστημα εκμάθησης προσπαθεί να δημιουργήσει μία υπόθεση η οποία πληροί όλα τα θετικά παραδείγματα αλλά κανένα από τα αρνητικά.

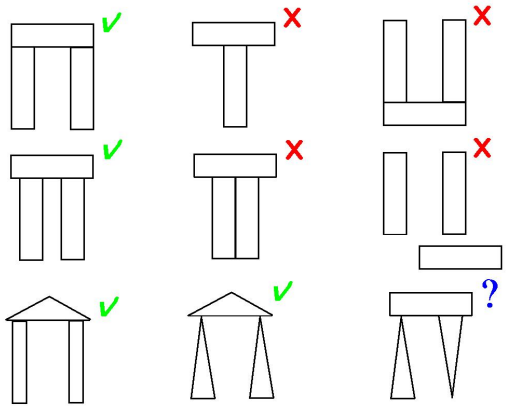
  Εστω ότι υπάρχουν μόνο θετικά παραδείγματα. Η παρουσίαση του πρώτου παραδείγματος δημιουργεί την αρχική υπόθεση. Για κάθε νέο παράδειγμα ελέγχεται αυτή η υπόθεση. Αν αυτή εξακολουθεί να ισχύει για το παράδειγμα η υπόθεση παραμένει ως έχει, αλλιώς η υπόθεση αλλάζει ώστε να ικανοποιεί το νέο παράδειγμα καθώς και όλα τα προηγούμενα παραδείγματα (πχ. γενικεύοντας ένα χαρακτηριστικό εάν η αντίστοιχη τιμή του για το νέο παραδείγματα είναι διαφορετική από τις προηγούμενες ή ακόμη αφαιρώντας το χαρακτηριστικό εάν εμφανίζονται όλες οι δυνατές τιμές του). Η εκμάθηση ολοκληρώνεται όταν και μόνο όταν η υπόθεση ικανοποιεί το σύνολο των παραδειγμάτων.

  Εφόσον τα παραδείγματα είναι θετικά και αρνητικά, το πρώτο παράδειγμα πρέπει αναγκαστικά να είναι θετικό ώστε να δημιουργηθεί η αρχική υπόθεση. Σε κάθε επόμενη παρουσίαση ενός παραδείγματος εξετάζεται εάν αυτό είναι θετικό ή αρνητικό. Εφόσον είναι θετικό ακολουθείται η διαδικασία που περιγράφηκε πριν. Εφόσον το παράδειγμα είναι αρνητικό, εντοπίζονται τα χαρακτηριστικά του τα οποία διαφέρουν από την υπάρχουσα υπόθεση καθώς και εκείνα τα οποία δεν διαφέρουν από την υπάρχουσα υπόθεση. Ενώ τα μεν περιορίζουν τις τιμές του αντίστοιχου χαρακτηριστικού, τα δε υποδηλώνουν ότι το

χαρακτηριστικό δεν είναι σημαντικό για τη δημιουργία της υπόθεσης. Είναι προτιμότερη η ύπαρξη και αρνητικών παραδειγμάτων στο σύνολο παραδειγμάτων, τα οποία τρόπον τινά οριοθετούν/περιορίζουν το μοντέλο επεξηγώντας τι δεν μπορεί να ισχύσει. Με τα αρνητικά παραδείγματα αποφεύγεται η υπεργενίκευση (overgeneralisation) του μοντέλου η οποία είναι δυνατό να συμβεί εάν υπάρχουν μόνο θετικά παραδείγματα. Πρόβλημα της στρατηγικής εκμάθησης μέσω παραδειγμάτων αποτελεί το πώς πραγματοποιείται η ανίχνευση των σημαντικών χαρακτηριστικών (επίπεδο περιγραφής της υπόθεσης). Το πρόβλημα αυτό αμβλύνεται εφόσον ώστε κάθε παράδειγμα να διαφέρει από τα προηγούμενα σε ένα μόνο χαρακτηριστικό ούτως ώστε η εκμάθηση να προχωρά σε μικρά βήματα.

Καθίσταται λοιπόν σαφής η σημασία που έχει η σωστή επιλογή του συνόλου παραδειγμάτων καθώς και η σειρά παρουσίασής τους στο σύστημα TN ώστε να αποφεύγονται οι αβάσιμες υποθέσεις (άρα εξαίρεται ο ρόλος του διδάσκοντος). Σημειώνεται ότι σε πολλές περιπτώσεις είναι προτιμότερος ο χαρακτηρισμός ενός παραδείγματος ως ειδική περίπτωση παρά η συλλήβδην αλλαγή της όλης υπόθεσης. Καθώς τα παραδείγματα εκφράζονται μέσω των χαρακτηριστικών τους, η υπόθεση περιγράφεται μέσω σημασιολογικών δικτύων, σχημάτων ή συστήματος κανόνων.
Ενα παράδειγμα δίνεται στο 15.5, όπου παρουσιάζεται το σύνολο παραδειγμάτων (συνοδευόμενων από ✔ και ✘ για θετικά και αρνητικά, αντίστοιχα, παραδείγματα) της αψίδας. Με την παρουσίαση των θετικών παραδειγμάτων φαίνεται ότι χαρακτηριστικά της αψίδας είναι:

Σχήμα 15.5: Παραδείγματα Αψίδας




(α) συμμετρική δομή ως προς τον μεσοκάθετο άξονά της,

(β) δύο όμοια υποστηρίγματα (παραλληλόγραμμα ή ισοσκελή τρίγωνα με την μα-

κρύτερη πλευρά ως ύψος και κάθετο άξονα συμμετρίας), το πάχος των οποίων δεν είναι καθορισμένο,

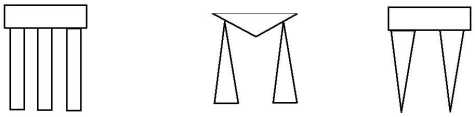
(γ) μία κορυφή (παραλληλόγραμμο ή ισοσκελές τρίγωνο με την μακρύτερη πλευρά ως βάση και κάθετο άξονα συμμετρίας), και

(δ) στήριξη η οποία δεν είναι αναγκαστικό να πραγματοποιείται στα άκρα της κορυφής. Από τα αρνητικά παραδείγματα τονίζεται:

(ε) η υποστηρικτική σχέση υποστηριγμάτων-κορυφής, καθώς και

(στ) ότι δεν επιτρέπεται να υπάρχει ένα μόνο υποστήριγμα ή τα υποστηρίγματα να εφάπτονται. Παρ' όλο που αυτό το σύνολο παραδειγμάτων μοιάζει επαρκές, το τελευταίο παράδειγμα του 15.6 δείχνει τη διαφορά ΦΝ και συστήματος ΤΝ. Η ΦΝ δέχεται αυτό το αντικείμενο ως αψίδα καθότι πληροί τη γενική μορφή αψίδας καθώς και το χρηστικό ρόλο της (να περνούν από κάτω άνθρωποι, οχήματα κλπ.), έστω και εάν δεν έχει συμμετρική δομή και τα υποστηρίγματα διαφέρουν μεταξύ τους και δεν το ένα από αυτά έχει σχήμα μη γνωστό από τα παραδείγματα.

Σχήμα 15.6: Αψίδες;



Μία άλλη έκφραση της στρατηγικής εκμάθησης μέσα από παραδείγματα χρησιμοποιεί τα διαθέσιμα παραδείγματα προκειμένου να δημιουργήσει έναν κανόνα-υπόθεση ο οποίος τα ικανοποιεί. Αρχικά τα χαρακτηριστικά καθώς και το αποτέλεσμα των παραδειγμάτων ταξινομούνται βάσει των αντίστοιχων τιμών τους. Κατόπιν, σχηματίζεται ένα δένδρο με αρχικό κόμβο (κορυφή) ένα από τα χαρακτηριστικά (πλην του αποτελέσματος) και παρακλάδια τις τιμές του χαρακτηριστικού από τα παραδείγματα. Για κάθε παρακλάδι με μικτό αποτέλεσμα επιλέγεται ένα νέο χαρακτηριστικό και η διαδικασία επαναλαμβάνεται. Η ανάπτυξη του δένδρου τερματίζεται όταν κανένα παρακλάδι δεν έχει μικτό αποτέλεσμα, οπότε ο κανόνας-υπόθεση εκφράζεται από τις διαδρομές μεταξύ αρχικού και τελικών κόμβων του δένδρου με το ζητούμενο αποτέλεσμα.

- Εκμάθηση μέσα από παρατήρηση και ανακάλυψη (learning from observation and discovery). Αποτελεί γενική μορφή της επαγωγικής διαδικασίας η οποία περιλαμβάνει τα συστήματα ανακάλυψης, την ανάπτυξη θεωριών, τη δημιουργία των κριτηρίων κατηγοριοποίησης για

τη παραγωγή ταξινομιών ή ιεραρχιών, χωρίς όμως την ύπαρξη διδάσκοντος. Απαιτείται περισσότερη συμπερασματική διαδικασία από ότι σε όλες τις άλλες μορφές εκμάθησης καθώς δεν υπάρχουν παραδείγματα ή δεν είναι γνωστή ποια είναι η απαραίτητη γνώση ή ο κατάλληλος μετασχηματισμός ή τα μέλη της αναλογίας, άρα το σύστημα ΤΝ πρέπει από μόνο να στρέψει την προσοχή του στο κατάλληλο μέρος του περιβάλλοντος και να εφαρμόσει τα κατάλληλα χαρακτηριστικά, μετασχηματισμό ή αναλογία ή, τέλος, να ανακαλύψει τα ενδιαφέροντα παραδείγματα-χαρακτηριστικά του προβλήματος.

### 15.7.2 Τεχνητά Νευρωνικά Δίκτυα (ΤΝΔ, artificial neural networks)

Τα τεχνητά νευρωνικά δίκτυα (ΤΝΔ) απορρέουν από την πεποίθηση ότι είναι δυνατό να προσομοιωθούν κάποιες ιδιότητες του εγκεφάλου, έτσι ώστε να δημιουργηθεί ένας τεχνητός εγκέφαλος (σε απλουστευμένη μορφή). Με αυτή την έννοια, τα ΤΝΔ βρίσκονται μεταξύ της ΤΝ και της προσομοίωσης του μέσου της ΦΝ. Υπερτερούν των προαναφερθεισών τεχνικών ΤΝ στη δυνατότητα άμεσης προσαρμογής σε μεταβαλλόμενο περιβάλλον χωρίς την ανάγκη διδάσκοντος καθώς και στην παραλληλία και κατανεμημένη επεξεργασία/αποθήκευση της γνώσης.

**Βασικά στοιχεία δομής εγκεφάλου**

Η δομή του εγκεφάλου είναι διαφορετική από αυτή των υπολογιστικών συστημάτων: ο εγκέφαλος δεν περιέχει σειρά από εντολές που εκτελούνται με αναφορά στις θέσεις μνήμης του, αλλά μοιάζει περισσότερο με σύστημα στοιχειωδών κυκλωμάτων συνδεδεμένων σε σειρά και παράλληλα τα οποία μετατρέπουν την είσοδο (αλληλεπίδραση μεγάλου αριθμού απλών στοιχείων/επεξεργαστών). Ο εγκέφαλος έχει βάρος 1.5 κιλά περίπου και αποτελείται από 1011 νευρώνες (εγκεφαλικά κύτταρα, neurons ή brain cells) με 1014 συνδέσεις (connections) μεταξύ τους. Υπάρχει εξειδίκευση νευρώνων – και τμημάτων του εγκεφάλου - που αφιερώνονται σε συγκεκριμένες λειτουργίες. Κατ' αναλογίαν, μπορούν να διακριθούν διάφορα είδη νευρώνων όσον αφορά στο σχήμα, στο μέγεθος, στη συνδεσμολογία καθώς και στη θέση τους στον εγκέφαλο (άρα και στη λειτουργία στην οποία συμμετέχουν). Ολοι όμως οι νευρώνες έχουν κάποια κοινά χαρακτηριστικά

- Τον πυρήνα (nucleus) όπου παράγονται συστατικά (πχ. μιτοχόνδρια, ένζυμα και ιόντα) τα οποία είναι απαραίτητα για τη διατήρηση και ομαλή λειτουργία του νευρώνα.
- Απολήξεις, οι οποίες έχουν μεγάλο μήκος και περιέχουν πληθώρα σημείων επαφής (τις συνάψεις, synapses) με τους συνδεδεμένους νευρώνες. Οι απολήξεις είναι δύο ειδών:

  (α) Δενδρίτες (dendrites). Αυτοί είναι της τάξης των χιλιάδων για κάθε νευρώνα και οι συνάψεις τους δέχονται τα ερεθίσματα τα οποία στέλνουν οι συνδεδεμένους νευρώνες.

  (β) Αξονας (axon). Είναι μοναδικός για κάθε νευρώνα και οι συνάψεις του μεταβιβάζουν τα ερεθίσματα του νευρώνα στους συνδεδεμένους νευρώνες.

Τα πιο σημαντικά στοιχεία της επικοινωνίας ενός νευρώνα με τους συνδεδεμένους νευρώνες είναι:

1. Οι συναπτικές δυνάμεις (connection strengths) των συνάψεων του νευρώνα. Η συναπτική δύναμη μίας σύναψης εκφράζει την ένταση και τον τρόπο (διεγερτικό, exhitatory ή κατασταλτικό, inhibitory) με τον οποίο ο νευρώνας επηρεάζει ή επηρεάζεται από τους άλλους νευρώνες. Παρ' όλο που ο διεγερτικός ή κατασταλτικός χαρακτήρας μίας σύνδεσης δεν είναι δυνατό να αλλάξει, η προσαρμογή στο μεταβαλλόμενο περιβάλλον μπορεί να προκαλέσει αλλαγή της έντασης της συναπτικής δύναμης (αύξηση ή μείωση).

2. Η κατάσταση του νευρώνα. Αυτή είναι είτε ενεργή (active) είτε ανενεργή (inactive) και καθορίζει το εάν ο νευρώνας μεταβιβάζει ερεθίσματα στους συνδεδεμένους στον άξονά του νευρώνες. Η κατάσταση εξαρτάται από το εάν το σύνολο των ερεθισμάτων στο νευρώνα (αλγεβρικό άθροισμα των διεγερτικών και κατασταλτικών συναπτικών δυνάμεων στο νευρώνα από τους συνδεδεμένους νευρώνες) ξεπερνά ένα κατώφλι. Το κατώφλι αυτό όχι μόνο διαφέρει από νευρώνα σε νευρώνα, αλλά και για κάθε νευρώνα είναι δυνατό να αλλάξει είτε λόγω κορεσμού (saturation, εφόσον ο νευρώνας καθίσταται πολύ συχνά ενεργός), είτε λόγω εκμάθησης, είτε ακόμα λόγω άλλων παραγόντων (πχ. γήρανση, ασθένεια κλπ.). Η ενεργοποίηση του νευρώνα εκφράζεται ως ακολουθία αποστολής δέλτα-ερεθισμάτων στους νευρώνες οι οποίοι έχουν συνάψεις στον άξονα του νευρώνα. Η ένταση καθώς και η συχνότητα της ακολουθίας εξαρτάται από το κατά πόσο το σύνολο των ερεθισμάτων στο νευρώνα ξεπερνά το κατώφλι.

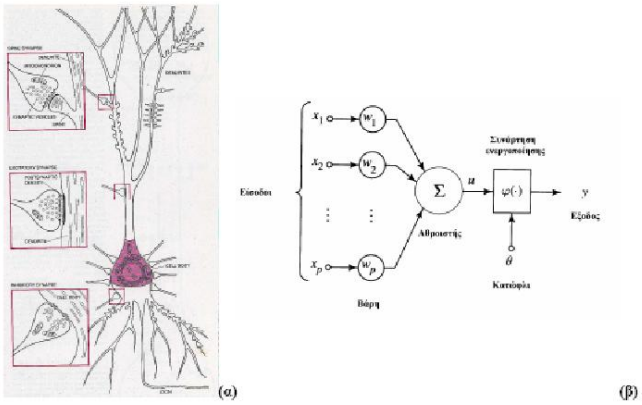
3. Η πυκνή δομή και συνδεσμολογία του εγκεφάλου. Η αποθήκευση και διαχείριση (κωδικοποίηση) της γνώσης πραγματοποιείται από τη συλλογική αλληλεπίδραση των συνάψεων καθώς και των καταστάσεων των συνδεδεμένων νευρώνων. Σε όλον τον εγκέφαλο υπάρχει περίσσεια νευρώνων, οπότε η κωδικοποίηση είναι αραιή, κατανεμημένη και εκτελείται παράλληλα από πολλούς νευρώνες. Αυτό αποδεικνύεται και από την καταστροφή μικρών περιοχών ή μεμονωμένων νευρώνων του εγκεφάλου σε ατυχήματα ή εγκεφαλικά: οι γειτονικοί νευρώνες οι οποίοι εκτελούν παρόμοια λειτουργία με τους απωλεσθέντες νευρώνες προσαρμόζονται ώστε να αναλάβουν τη λειτουργία τους και να τους αντικαταστήσουν σε κάποιο βαθμό (πλαστικότητα εγκεφάλου). Η αραιή και κατανεμημένη κωδικοποίηση είναι υπεύθυνη για την ευρωστία του εγκεφάλου σε ελλιπή ή μερικώς παραμορφωμένα ερεθίσματα ή εισόδους.

**Βασικά στοιχεία δομής ΤΝΔ**

Εμπνευσμένα από τα βασικά τμήματα καθώς και από τον τρόπο επικοινωνίας των νευρώνων, οι κόμβοι των ΤΝΔ 15.7β απαρτίζονται από:

Τις εισόδους (inputs) στον κόμβο x1, x2,…, xp, οι οποίες προσομοιώνουν τις συνάψεις των δενδριτών του νευρώνα και εκφράζουν τα ερεθίσματα των συνδεδεμένων κόμβων ή του περιβάλλοντος.

Σχήμα 15.7: Νευρώνας του ανθρώπινου εγκεφάλου (α), κόβος ΤΝΔ (β)




Τα βάρη (weights ή connection strengths) w1, w2, …, wp, τα οποία αντιστοιχούν στις συναπτικές δυνάμεις των νευρώνων και αναπαριστούν την επίδραση που έχουν οι αντίστοιχες είσοδοι x1, x2,…, xp στην κατάσταση του κόμβου. Τα βάρη χαρακτηρίζονται από ένα πρόσημο (+ ή –, αντίστοιχο της διεγερτικής ή κατασταλτικής σύναψης) και από έναν αριθμό (αντίστοιχο της έντασης της σύναψης).

Ενα αθροιστή Σ ο οποίος αθροίζει (αλγεβρικά) τα γινόμενα των εισόδων με τα αντίστοιχα βάρη ώστε να υπολογιστεί η συνολική είσοδος u στον κόμβο ().

Μία συνάρτηση ενεργοποίησης (activation function) φ και ένα κατώφλι (threshold) θ. Στην απλούστερη περίπτωση γίνεται σύγκριση της συνολικής εισόδου u με το κατώφλι θ προκειμένου να κριθεί εάν ο κόμβος θα γίνει ο ίδιος ενεργός και θα μεταφέρει ερεθίσματα σε άλλους συνδεδεμένους κόμβους. Στη γενικότερη περίπτωση, το κατώφλι θ προστίθεται στη συνολική είσοδο u οπότε – μέσω της φ - υπολογίζεται η έξοδος y του κόμβου. Η έξοδος αποτελεί αριθμό ο οποίος αντιστοιχεί στην ένταση και συχνότητα της ακολουθίας δέλτα-ερεθισμάτων του ενεργού νευρώνα. Στο 15.7 παρουσιάζεται ένα παράδειγμα συνάρτησης ενεργοποίησης, η οποία δίνεται από το τύπο για □= u+ θ (θ=-0.3).

Τα ΤΝΔ (15.8(β)) έχουν ως αφετηρία τους τον εγκέφαλο, αλλά περιορίζονται σε όχι περισσότερους από 1000 κόμβους. Η δομή τους είναι η εξής:

- Οι κόμβοι έχουν καθορισμένη διάταξη και βρίσκονται συνήθως σε επάλληλα επίπεδα (layers). Ανάλογα με τη θέση του στο ΤΝΔ, ένας κόμβος χαρακτηρίζεται ως:

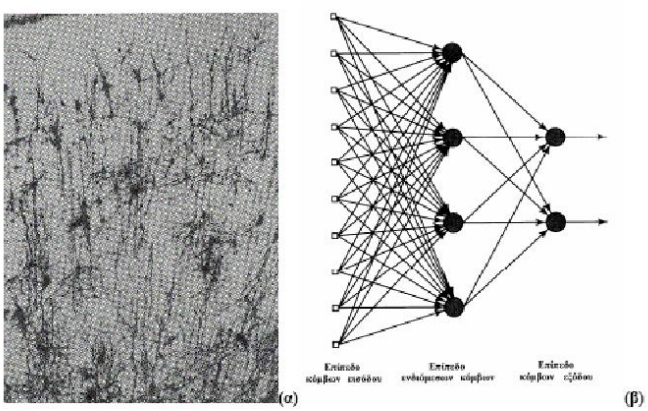
  Κόμβος εισόδου (input node), οπότε βρίσκεται στο κατώτερο επίπεδο του ΤΝΔ (επίπεδο εισόδου (input layer), το αριστερότερο κάθετο επίπεδο κόμβων στο 15.8(β)) και λαμβάνει ως είσοδο πληροφορία από το περιβάλλον, πχ. την τιμή ενός συγκεκριμένου pixel από κάποια ψηφιακή εικόνα, έναν αριθμό κλπ. Πριν την εισαγωγή

της, η πληροφορία κανονικοποιείται σε συγκεκριμένο διάστημα το οποίο μπορεί να δεχτεί η συνάρτηση ενεργοποίησης του κόμβου.

Κόμβος εξόδου (output node), οπότε βρίσκεται στο ανώτερο επίπεδο του ΤΝΔ (επίπεδο εξόδου (output layer), το δεξιότερο κάθετο επίπεδο κόμβων στο 15.8(β)) και αποδίδει την έξοδό του στο περιβάλλον.

Ενδιάμεσος κόμβος (hidden node), οπότε βρίσκεται σε ενδιάμεσο επίπεδο κόμβων (hidden layer), δέχεται εισόδους από άλλους κόμβους και - μέσω της συνάρτησης ενεργοποίησης του - αποδίδει την έξοδό του σε άλλους κόμβους του ΤΝΔ.

Σχήμα 15.8: Τμήμα ανθρώπινου εγκεφάλου (α), κόβος ΤΝΔ (β)



- Οι συνδέσεις του ΤΝΔ είναι δυνατό να εφαρμόζονται μεταξύ κόμβων διαφορετικών επιπέδων - και να κατευθύνονται είτε προς κόμβο επόμενου επίπεδου (forward connection) είτε προς κόμβο προηγούμενου επιπέδου (backward connection) -, αλλά είναι επίσης δυνατές και μεταξύ κόμβων του ιδίου επιπέδου ή ακόμα και μεταξύ του ίδιου κόμβου (self-connection). Η συνδεσμολογία (connectivity) του ΤΝΔ εκφράζει τον συνολικό τρόπο με τον οποίο οι κόμβοι του ΤΝΔ αλληλεπιδρούν. Λόγω του ότι η συνδεσμολογία είναι κατά πολύ υπεύθυνη για τον τρόπο εκμάθησης και τα χαρακτηριστικά λειτουργίας του ΤΝΔ, γίνεται αναφορά σε συγκεκριμένες δομές ή αρχιτεκτονικές ΤΝΔ οι οποίες έχουν ως ειδοποιό διαφορά τους τη δομή των κόμβων (συναρτήσεις ενεργοποίησης και κατώφλια), τον αριθμό των επιπέδων κόμβων και τη συνδεσμολογία τους.

- Τα βάρη των συνδέσεων λαμβάνουν αριθμητικές τιμές μέσα από συγκεκριμένο διάστημα (πχ. [-1,+1]), τα όρια του οποίου δεν υπερβαίνονται προκειμένου να μην κορεστούν οι συνδέσεις και, κατά συνέπεια, η έξοδος του κόμβου.

- Είθισται τα ρολόγια των κόμβων του ιδίου επιπέδου να είναι συγχρονισμένα (ενιαίο ρολόϊ ανά επίπεδο) και τα ρολόγια διαδοχικών επιπέδων να λειτουργούν έτσι ώστε να πραγματοποιείται πρώτα η ενεργοποίηση όλων των κόμβων του επιπέδου εισόδου, αμέσως μετά του πρώτου ενδιάμεσου επιπέδου και ου τω καθ΄ εξής μέχρι το επίπεδο εξόδου.

**Εκμάθηση ΤΝΔ**

Στη γενική περίπτωση, η λειτουργία του ΤΝΔ ξεκινά με την ταυτόχρονη ενεργοποίηση των κόμβων εισόδου από τις πληροφορίες του περιβάλλοντος, συνεχίζεται με την ταυτόχρονη ενεργοποίηση των κόμβων ενδιάμεσων επιπέδων και ολοκληρώνεται με την ταυτόχρονη ενεργοποίηση των κόμβων του ανώτερου επιπέδου και την απόδοση των αντίστοιχων εξόδων. Για συγκεκριμένες εισόδους και καθορισμένη αρχιτεκτονική, οι έξοδοι του ΤΝΔ εξαρτώνται από τον αριθμό των κόμβων στα ενδιάμεσα επίπεδα καθώς και από τα βάρη των συνδέσεων: μεταβάλλοντας οποιοδήποτε από τα παραπάνω αλλάζουν και οι έξοδοι. Η μεταβολή των βαρών των συνδέσεων συνεπάγεται άμεση μεταβολή της συμπεριφοράς του ΤΝΔ, άρα είναι δυνατό με την κατάλληλη ρύθμισή τους (fine-tuning) να αποδίδονται συγκεκριμένες έξοδοι για συγκεκριμένες εισόδους. Η διαδικασία σταδιακής μεταβολής των βαρών ενός ΤΝΔ προκειμένου να επιτευχθεί η αντιστοιχία συγκεκριμένων ζευγών εισόδων-εξόδων αποτελεί την εκμάθηση του ΤΝΔ (ANN learning). Αυτή πραγματοποιείται ως εξής:

**Βήμα 1.** Συλλογή συνόλου γνωστών ζευγών εισόδων-εξόδων (x,y) προς εκμάθηση (σύνολο εκμάθησης, training set).

**Βήμα 2.** Επιλογή τυχαίων τιμών για τα βάρη του ΤΝΔ μέσα από το συγκεκριμένο διάστημα το οποίο δεν προκαλεί κορεσμό των συνδέσεων. Ανάθεση της τιμής 0 στη μεταβλητή η οποία εκφράζει τον αριθμό επαναλήψεων. Ανάθεση της τιμής 0 στη μεταβλητή η οποία εκφράζει αλλαγή βαρών.

**Βήμα 3.** Για κάθε ζεύγος εισόδων-εξόδων (x,y):

Βήμα 3.1. Είσοδος του x στο (επίπεδο εισόδου του) ΤΝΔ, ενεργοποίηση των κόμβων του ΤΝΔ και απόδοση της εξόδου yΤΝΔ.

Βήμα 3.2. Σύγκριση της εξόδου yΤΝΔ με την επιθυμητή έξοδο y και υπολογισμός του σφάλματος e. Ελεγχος του εάν το σφάλμα e υπερβαίνει μία επιτρεπόμενη τιμή.

Βήμα 3.2.1. Ανάθεση της τιμής 1 στη μεταβλητή η οποία εκφράζει αλλαγή βαρών. Πραγματοποίηση αλλαγής των βαρών του ΤΝΔ ανά επίπεδο. Πρώτα μεταβάλλονται τα βάρη των κόμβων του επιπέδου εξόδου, μετά τα βάρη των κόμβων του αμέσως προηγούμενου επιπέδου και τελευταία τα βάρη του επιπέδου εισόδου. Η μεταβολή του κάθε βάρους είναι μικρή, ανάλογη της τρέχουσας τιμής του καθώς και (α) ανάλογη του σφάλματος e εάν ο αντίστοιχος κόμβος ανήκει στο επίπεδο εξόδου, (β) ανάλογη της αλλαγής των βαρών των συνδεδεμένων κόμβων ανωτέρων επιπέδων εάν ο αντίστοιχος κόμβος δεν ανήκει στο επίπεδο εξόδου.

Βήμα 3.2.2. Επιστροφή στο Βήμα 3.

**Βήμα 4.** Ελεγχος του εάν η μεταβλητή η οποία εκφράζει αλλαγή βαρών λαμβάνει την τιμή 0.

Βήμα 4.1. Απόφαση ότι έχει επιτευχθεί σύγκλιση (convergence) του ΤΝΔ για το σύνολο εκμάθησης (δηλαδή το σφάλμα e είναι ικανοποιητικά μικρό για όλα τα ζεύγη εισόδων-εξόδων του συνόλου εκμάθησης). Μετάβαση στο Βήμα 5.

Βήμα 4.2. Αύξηση της τιμής της μεταβλητής η οποία εκφράζει τον αριθμό επαναλήψεων κατά 1. Ελεγχος του εάν η μεταβλητή αυτή υπερβαίνει μία προκαθορισμένη τιμή (ανώτατο επιτρεπτό αριθμό επαναλήψεων εκμάθησης).
Εάν ναι,

Βήμα 4.2.1. Απόφαση ότι δεν είναι δυνατό να επιτευχθεί σύγκλιση του ΤΝΔ. Μετάβαση στο Βήμα 5.

Βήμα 4.2.2. Ανάθεση της τιμής 0 στη μεταβλητή η οποία εκφράζει αλλαγή βαρών. Μετάβαση στο Βήμα 3.

**Βήμα 5.** Τέλος εκμάθησης.

Το **Βήμα 3** επαναλαμβάνεται αρκετές φορές (της τάξης των εκατοντάδων ή, ακόμη και, χιλιάδων φορών) και τερματίζεται είτε όταν επιτευχθεί η σύγκλιση, είτε όταν ο αριθμός επαναλήψεων υπερβεί το προκαθορισμένο όριο οπότε και συμπεραίνεται ότι δεν είναι δυνατή η σύγκλιση. Στην προηγούμενη μέθοδο εκμάθησης το σφάλμα e και η αλλαγή των βαρών υπολογίζονται ανεξάρτητα για κάθε ζεύγος εισόδων-εξόδων του συνόλου εκμάθησης (on-line ή pattern training). Εναλλακτικά, είναι δυνατόν το σφάλμα να υπολογίζεται συνολικά για όλα τα ζεύγη εισόδων-εξόδων, οπότε η αλλαγή των βαρών πραγματοποιείται μία φορά για ολόκληρο το σύνολο εκμάθησης (batch training). Από τα προηγούμενα καθίσταται προφανές ότι τα βάρη των συνδέσεων αποτελούν τις κατεξοχήν αποθήκες γνώσης των ΤΝΔ. Μέσω της εκμάθησης είναι δυνατό να προσεγγιστούν: Η ρύθμιση των βαρών πρέπει να είναι τέτοια ώστε το ΤΝΔ να είναι ικανό να αποκρίνεται σωστά και σε νέες εισόδους (test patterns), άρα να επιδεικνύει γενίκευση (generalisation). Η γενίκευση χαρακτηρίζει την απαιτούμενη μορφή και πολυπλοκότητα της προσεγγιζόμενης συνάρτησης ή διαχωριστικής καμπύλης έτσι ώστε η ορθή απόκριση να επεκτείνεται σε μη τετριμμένα σύνολα ελέγχου (text sets). Για την επιτυχημένη γενίκευση, συνήθης στρατηγική κατά την εκμάθηση είναι συγχρόνως με το σύνολο εκμάθησης να χρησιμοποιείται και ένα σύνολο τεκμηρίωσης (validation set): η ρύθμιση των βαρών εκτελείται από το σύνολο εκμάθησης (έλεγχος ικανοποιητικής προσέγγισης της συνάρτησης ή διαχωριστικής καμπύλης) σε συνδυασμό με το σύνολο τεκμηρίωσης (επιλογή καταλληλότερης συνάρτησης ή διαχωριστικής καμπύλης).

- Η επιλογή αρχικών βαρών του ΤΝΔ. Είναι δυνατόν κάποια αρχικά βάρη να μην επιτρέπουν ή να επιβραδύνουν σημαντικά τη σύγκλιση, ενώ κάποια άλλα να την επισπεύδουν.

- Η σειρά παρουσίασης των ζευγών του συνόλου εκμάθησης (ισχύει για τη πρώτη μέθοδο εκμάθησης). Για γρηγορότερη σύγκλιση του ΤΝΔ, είναι σκόπιμο να αλλάζει η σειρά παρουσίασης των ζευγών σε κάθε συνολική παρουσίασή τους. Με αυτό τον τρόπο, η κάθε αλλαγή βαρών δεν ακολουθείται πάντα από την ίδια αλλαγή βαρών, οπότε οι διαδοχικές αλλαγές βαρών δεν εξουδετερώνονται πάντα κατά τον ίδιο τρόπο.

- Ο αριθμός των κόμβων στα ενδιάμεσα επίπεδα. Η επιλογή του σωστού αριθμού κόμβων ανά ενδιάμεσο επίπεδο προωθεί τη δημιουργία συνάρτησης ή διαχωριστικής καμπύλης

κατάλληλης πολυπλοκότητας. Αντίθετα, η ύπαρξη μη επαρκούς αριθμού κόμβων δεν επιτρέπει την ικανοποιητική προσέγγιση, ενώ η ύπαρξη περισσοτέρων κόμβων από ό,τι είναι απαραίτητο δημιουργεί προβλήματα γενίκευσης

- Ο αριθμός ενδιάμεσων επιπέδων. Η μη χρήση ενδιάμεσων επιπέδων κόμβων επιτρέπει μόνο την επίλυση γραμμικά διαχωρίσιμων (linearly separable) προβλημάτων. Εχει αποδειχτεί ότι η χρήση ενός ενδιάμεσου επιπέδου κόμβων επιτρέπει την προσέγγιση οποιασδήποτε συνάρτησης ή διαχωριστικής καμπύλης (Cybenko, 1989), με την πολυπλοκότητα της προσέγγισης να εξαρτάται από τον αριθμό των κόμβων στο ενδιάμεσο επίπεδο. Η χρήση περισσοτέρων του ενός ενδιάμεσων επιπέδων επιτρέπει την πιο εύκολη δημιουργία πολύπλοκων συναρτήσεων ή διαχωριστικών καμπυλών.

Καθότι, για λόγους γενίκευσης, προτιμάται η πιο απλή συνάρτηση ή διαχωριστική καμπύλη η οποία ικανοποιεί συγχρόνως τα σύνολα εκμάθησης και τεκμηρίωσης, είναι προτιμότερη η επιλογή ενός μικρού ΤΝΔ (λίγοι κόμβοι, λίγα ενδιάμεσα επίπεδα) και η μετέπειτα επέκτασή του εάν και μόνο εάν η πολυπλοκότητα της προσεγγιζόμενης συνάρτησης δεν επιτυγχάνεται για διαφορετικές αρχικοποιήσεις βαρών και διαφορετικές παρουσιάσεις των ζευγών του συνόλου εκμάθησης.

**Αποτίμηση ΤΝΔ**

Τα ΤΝΔ υπερτερούν των προαναφερθεισών τεχνικών ΤΝ όσον αφορά στην:

- (α) Αμεση εκμάθηση (προσαρμογή στο περιβάλλον) μέσα από παραδείγματα χωρίς την ανάγκη ύπαρξης διδάσκοντος ή την ανάγκη προσδιορισμού της αναλυτικής λύσης.
- (β) Κατανεμημένη αναπαράσταση της γνώσης στα βάρη των ΤΝΔ έναντι της δυαδικής αναπαράστασης των τεχνικών ΤΝ. Αυτή η αναπαράσταση επιτρέπει την αποδοτική διαχείριση πολλαπλών δεδομένων και περιορισμών χωρίς τη δημιουργία προβλημάτων συνδυαστικής έκρηξης.
- (γ) Σε συνδυασμό με την παράλληλη και αραιή κωδικοποίηση της γνώσης στα βάρη, η κατανεμημένη αναπαράσταση προωθεί την ευρωστία των ΤΝΔ σε ελλιπή ή παραμορφωμένα δεδομένα. Η λειτουργία δε σταματά σε περίπτωση εισαγωγής λανθασμένων δεδομένων ή λόγω της έλλειψης κάποιων χαρακτηριστικών των δεδομένων. Απλά, παρατηρείται μία σταδιακή επιδείνωση (graceful degradation) της ακρίβειας των ΤΝΔ η οποία είναι ανάλογη της απόκλισης της εισαγόμενης τιμής από την αναμενόμενη καθώς και ανάλογη του ποσοστού των χαρακτηριστικών τα οποία λείπουν. Το ίδιο ισχύει και για τη βλάβη μέρους των ΤΝΔ (πχ. κάποιων κόμβων, συνδέσεων ή βαρών), οπότε η επιδείνωση της λειτουργίας τους είναι ανάλογη του αριθμού των μη λειτουργικών κόμβων ή του αριθμού και μεγέθους του βάρους των μη λειτουργικών συνδέσεων.
- (δ) Δυνατότητα γενίκευσης και ορθής απάντησης σε νέες εισόδους.

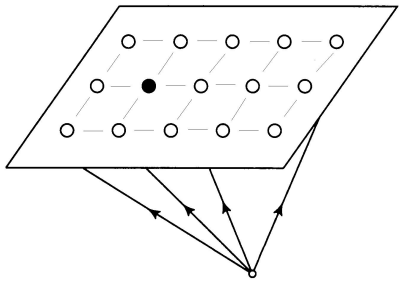
Τα επόμενα δύο παραδείγματα τονίζουν αυτά τα σημεία.

**Ταξινόμηση (κατηγοριοποίηση) μερών λόγου** Τα ΤΝΔ αυτοοργάνωσης (self-organising maps, Kohonen, 1988) είναι εμπνευσμένα από τους τοπολογικούς χάρτες του εγκεφάλου. Οι κόμβοι δομούνται σε ένα επίπεδο το οποίο αναπτύσσεται ως πλέγμα σε μία ή δύο διαστάσεις (15.9) με κατασταλτικές συνδέσεις μεταξύ γειτονικών κόμβων. Κατά την εκμάθηση του ΤΝΔ αυτοοργάνωσης:

(α) οι κόμβοι ανταγωνίζονται (competition) μεταξύ τους όσον αφορά στην ομοιότητα των βαρών τους με την εκάστοτε είσοδο,

(β) ο κόμβος με την μεγαλύτερη ομοιότητα των βαρών του με την είσοδο, καθώς και - σε μικρότερο βαθμό - οι άμεσοι γείτονές του, μεταβάλλουν τα βάρη τους ώστε αυτά να μοιάζουν ακόμη περισσότερο στην εκάστοτε είσοδο.

Σχήμα 15.9: ΤΝΔ αυτοοργάνωσης



Αυτός ο τρόπος εκμάθησης έχει ως αποτέλεσμα το ΤΝΔ αυτοοργάνωσης να διαμορφώνει τα βάρη των κόμβων του έτσι ώστε τα βάρη κοντινών κόμβων μοιάζουν ενώ αυτά μακρινών κόμβων διαφέρουν σημαντικά. Αρα, το πλέγμα αποτελεί έναν τοπολογικό χάρτη όπου δημιουργούνται περιοχές του πλέγματος με κοινά χαρακτηριστικά. Κατά τον έλεγχο του ΤΝΔ, οι είσοδοι εφαρμόζονται σε όλους τους κόμβους του ΤΝΔ και επικρατεί ο κόμβος ο οποίος μοιάζει περισσότερο στη είσοδο (winner-take-all). καθότι είσοδοι οι οποίες μοιάζουν τοποθετούνται κοντά στον τοπολογικό χάρτη, ενώ είσοδοι οι οποίες διαφέρουν σημαντικά τοποθετούνται μακριά, δημιουργείται μία κατηγοριοποίηση των εισόδων ανάλογα με τις περιοχές του πλέγματος οι οποίες

έχουν καθοριστεί κατά την εκμάθηση. ΤΝΔ αυτοοργάνωσης έχουν χρησιμοποιηθεί επιτυχώς για την ταξινόμηση και κατηγοριοποίηση δεδομένων.

## 15.8 Βελτιστοποίηση – Γενετικοί Αλγόρθμοι

Οπως φαίνεται από τις προαναφερθείσες μεθόδους αναζήτησης και ικανοποίησης περιορισμών, η ΤΝ δεν δρα προσεγγιστικά, αλλά αναζητά μία ακριβή (και σε πολλές περιπτώσεις, τη βέλτιστη) λύση. Γι' αυτό στις απαιτήσεις μνήμης των δύο αυτών τεχνικών περιλαμβάνεται και η πορεία προς την τρέχουσα κατάσταση (ώστε να μην την επαναλαμβάνεται κάποια κατάσταση) καθώς και η ποιότητα της – μέχρι στιγμής - καλύτερης λύσης. Η ακολουθία των μεταβάσεων πραγματοποιείται σειριακά και είναι ομαλή από άποψη "γειτονιάς" (οι διαδοχικές καταστάσεις συνδέονται, επεκτείνουν η μία την άλλη, άρα "μοιάζουν" μεταξύ τους) αλλά όχι από άποψη ποιότητας. Εμπνευσμένοι από τη κληρονομικότητα που απαντάται στα έμβια όντα - και συγκεκριμένα από την εξέλιξη των όντων, την επικράτηση των καλύτερων όσον αφορά τη δυνατότητα προσαρμογής τους στο περιβάλλον -, οι γενετικοί αλγόριθμοι (ΓΑ) αποτελούν μία στοχαστική (και όχι εξαντλητική) μέθοδο βέλτιστης αναζήτησης/ικανοποίησης περιορισμών, δηλ. αποσκοπούν στην ταχεία και οικονομική εύρεση μιας επαρκώς καλής (και όχι απαραίτητα της καλύτερης) λύσης. Έτσι, 30 χρόνια πριν βιολόγοι, χρησιμοποίησαν Η/Υ για προσομοίωση βιολογικών συστημάτων, πληθυσμών. Δημιουργήθηκαν έτσι οι ΓΑ, οι οποίοι στηρίζονται στην αναλογία με τα φυσικά φαινόμενα/μηχανισμούς και εστιάζουν στην εξέλιξη (κατά Darwin), την κληρονομικότητα, τον πολλαπλασιασμό και την επικράτηση των πλέον συμβατών (με το περιβάλλον) ατόμων του πληθυσμού. Η υλοποίηση πραγματοποιείται στο υπερ-συμβολικό επίπεδο. Έτσι,

- Ο πληθυσμός χρωμοσωμάτων αποτελεί σύνολο δυνατών καταστάσεων του προβλήματος.
- Τα χρωμοσώματα, αποτελούμενα από γονίδια (χαρακτηριστικά), εξελίσσονται σε ακολουθία γενιών μέσω διασταυρώσεων, μεταλλάξεων σε επίπεδο γονιδίου και επιλογής.
- Η συνάρτηση καταλληλότητας εκφράζει τον σκοπό του πληθυσμού και συνεπώς την καταλληλότητα/ποιότητα λύσης.

Τα πιο κατάλληλα χρωμοσώματα επιβιώνουν σε κάθε γενιά, οδηγώντας σε βελτιστοποίηση του πληθυσμού όσον αφορά στους συνδυασμούς χρωμοσωμάτων σε σχέση με τις απαιτήσεις του προβλήματος.
Σημειώνεται ότι, καθώς το μέγεθος του προβλήματος μεγαλώνει, η οικονομία μνήμης (και συνεπώς η υπεροχή των ΓΑ έναντι των κλασσικών τεχνικών) καθίσταται προφανής. Γενικά, οι ΓΑ προσφέρουν ικανοποιητικές λύσεις σε περιορισμένο χρόνο και βέλτιστη λύση για επαρκή χρόνο λειτουργίας (αριθμό γενεών). Γενικά, η κλασσική αναζήτηση προσαρμόζει το πρόβλημα στην δομή του Η/Υ και (σχεδόν) εξαντλητικά εξερευνά τον χώρο του προβλήματος για την βέλτιστη λύση από υποψήφια λύση σε γειτονοκή υποψήφια λύση. Αντίθετα, οι ΓΑ, μέσω της διασταύρωσης και της μετάλλαξης, πραγματοποιούν μη γειτονική αναζήτηση για μία (σχεδόν) βέλτιστη λύση σε ικανό χρόνο, με χρήση περιορισμένης μήνης. Αυτό σημαίνει ότι μία υποψήφια λύση μπορεί να εμφανίζεται στον πληθυσμό σε περισσότερες από μία γενεές. Μάλιστα, όσο καλύτερη είναι η υποψήφια λύση τόσο πιο συχνά θα εμφανίζεται στον πληθυσμό.

## 15.9 Διαδραστικό Υλικό – Σύνδεσμοι

- http://www.cs.waikato.ac.nz/ml/weka/
- https://en.wikipedia.org/wiki/Artificial_intelligence http://www.mathworks.com/products/matlab/online/
- https://archive.ics.uci.edu/ml/datasets.html

## 15.10 Ασκήσεις

Μπορείτε να χρησιμοποιήσετε τον πρώτο από τους παραπάνω free sql editors για τα προβλήματα που ακολουθούν. Οι πίνακες βρίσκονται στα δεξιά.

1. Να σχεδιαστεί ένα νευρωνικό δίκτυο για την πύλη OR
2. Να σχεδιαστεί ένα νευρωνικό δίκτυο για την πύλη AND
3. Να σχεδιαστεί το δέντρο ευρετικής αναζήτησης για το παζλ που ακολουθεί:

| | | |
|---|---|---|
| 4 | 1 | 3 |
| 7 | 2 | 6 |
| | 5 | 8 |

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | |

4. Χρησιμοποίστε data sets από το Irvine Machine Learning Depository (4ος σύνδεσμος) και χρησομοποιήστε το weka για να κάνετε ταξινόμηση (classification). Παρατηρήστε ποιοί αλγόριθμοι είναι πιο αποδοτικοί και πως βελτιώνονται τα αποτελέσματα όταν το σύστημα εκπαιδεύεται με μεγάλο όγκο δεδομένων.

# Βιβλιογραφία


[1] Τατιάνα Ταμπουρατζή 2011. Τεχνητή νοημοσύνη και Συστήματα Παραγωγής. http://www.tex.unipi.gr/undergraduate/notes/ai/ai_notes_tt_2011.pdf

[2] Behrouz Forouzan "Εισαγωγή στην Επιστήμη των Υπολογιστών" τρίτη έκδοση 2014

[3] Newell A., Shaw J.C., Simon H.A., 1956. Empirical explorations of the logic theory machine: a case study in heuristics, in Proceedings of the Western Joint Computer Conference, pp. 218-239. Also in Computers and Thought (Feigenbaum E. and Feldman J., eds.), McGraw-Hill: New York, U.S.A., 1963, pp. 134-152.

[4] Newell A., Simon H., 1972. Human Problem Solving. Prentice-Hall: Englewood Cliffs, NJ, U.S.A.

[5] Searle J.R., 1980. Minds, brains, and programs, Behavioral and Brain Sciences, Vol.3, pp. 417-424.

[6] Turing A.M., 1950. Computing machinery and intelligence. Mind, Vol. LIX, pp. 433-460.

[7] Hodges W., 1977. Logic. Penguin Books: Harmondsworth, Middlesex, UK.

[8] Yager R.R., Zadeh L.A. (eds.), 1992. An Introduction to Fuzzy Logic Applications in Intelligent Systems. Kluwer Academic Publishers: Boston, U.S.A.

[9] Zadeh, L.A., 1996. Fuzzy Logic = Computing with Words, IEEE Transactions on Fuzzy Systems Vol. 2, pp. 103-111.

[10] Waltz D., 1975. Understanding line drawings of scenes with shadows, in The Psychology of Computer Vision (P.H.Winston ed.), McGraw-Hill: New York, U.S.A.

[11] Minsky, M., 1975. A framework for representing knowledge (MIT-AI Laboratory Memo 306, June, 1974) reprinted in The Psychology of Computer Vision (P.H.Winston), McGraw-Hill, New York, U.S.A.

[12] Shortcliffe, E.H., 1976. Computer-based medical consultations: MYCIN. Elsevier Publishing Corporation Inc., New York, U.S.A.

[13] Weizenbaum, J. 1966. ELIZA - A Computer Program For the Study of Natural Language Communication Between Man and Machine, Communications of the ACM, Vol. 9, pp. 36-45.

[14] Winograd, T., 1972. Procedures as a representation for data in a computer program for understanding natural language, Cognitive Psychology, Vol. 3., pp. 1-191. Also in Understanding Natural Language (1972), Academic Press, New York, U.S.A.

[15] Cybenko, G., 1989. Approximation by superpositions of a sigmoidal function, Mathematics of Control, Signals, and Systems, Vol. 2, pp. 303-314.

[16] Kohonen, T., 1988. Self-Organisation and Associative Memory (3rd edition). Springer-Verlag: New York, U.S.A.

[17] Quinlan, J.R., 1986. Induction of decision trees, Machine Learning, Vol. 1, pp. 81-106.

[18] D. Rumelhart, J.McClelland, 1986. On learning the past tenses of English verbs, pp. 216-271, in Parallel Distributed Processing – Vol. 2 (Rumelhart D., McClelland J., eds.), MIT Press: Cambridge, MA, U.S.A Goldberg, D.E., 1989. Genetic Algorithms in Search Optimization and Machine Learning. Addison Wesley.

# Κεφάλαιο 16

# Κρυπτογραφία και Ασφάλεια

## 16.1 Ιστορική αναδρομή

Η τέχνη της κρυπτογραφίας ξεκίνησε εδώ και 2500 χρόνια, το λιγότερο και έπαιξε σημαντικό ρόλο στην ιστορία απο τότε. Στην αρχαία Ελλάδα, οι έφοροι της Σπάρτης επικοινωνούσαν με τους στρατηγούς χρησιμοποιώντας μακριές και στενές κορδέλες τις οποίες τύλιγαν γύρω από μια σκυτάλη (κύλινδρο) και μετά γράφαν το μήνυμα κατά μήκος της σκυτάλης. Για να διαβάσει κάποιος το μήνυμα, έπρεπε να έχει μια παρόμοια σκυτάλη με αυτή που είχε χρησιμοποιηθεί για την κωδικοποίηση και να τυλίξει την κορδέλα γύρω από τη σκυτάλη με τον ίδιο τρόπο. Αυτό το σύστημα (διπλής κατεύθυνσης) κρυπτογραφίας είναι ένα κλασικό σύστημα με ένα κλειδί (τη σκυτάλη).

Ο Ιούλιος Καίσαρας επικοινωνούσε με τους φίλους του αντικαθιστώντας κάθε γράμμα με ένα άλλο, το οποίο προέκυπτε με ολίσθηση κατά $k$ βήματα στο αλφάβητο. Αυτό είναι ένα από τα πιο απλά, εύκολα αλλά και ανασφαλή κρυπτοσυστήματα που έχουν προταθεί.

Οι Βενετοί ήταν οι πρώτοι που χρησιμοποίησαν κρυπτογραφία συστηματικά, από το 13ο αιώνα, για διπλωματική αλληλογραφία. Οι πρώτες δημοσιεύσεις (στα λατινικά) περί κρυπτογραφίας φάνηκαν το 1500 ("Στεγανογραφία") και το 1518 από τον αββά Ιωάννη Τριθέμιο και αργότερα δημοσιεύτηκε το "Περι κρυπτικών συμβόλων και γραμμάτων" από τον J. B. Porta (1538 - 1615, Ιταλό φυσικό και μαθηματικό). Από τότε η κρυπτογραφία έγινε αντικείμενο ιδιαίτερου ενδιαφέροντος και απέκτησε εφαρμογές. Οι αλχημιστές χρησιμοποιούσαν σύμβολα για να κρυπτογραφήσουν τους τύπους τους, αλλά και πολλοί φιλόσοφοι ενδιαφέρθηκαν για την κρυπτογραφία: Ο Sir Francis Bacon (1561 - 1626) επινόησε ένα σύστημα κρυπτογράφησης όπου κάθε γράμμα αντικαθίσταται με μια λέξη πέντε γραμμάτων και ο Leonardo Da Vinci (1452 - 1519) χρησιμοποιούσε μια μέθοδο κρυπτογράφησης με καθρέπτη.

Ο Edgar Allan Poe (1809-1849) στο κλασικό διήγημα "Το χρυσό έντομο" ("The Gold Bug") που δημοσίευσε το 1843, εξηγεί τις βασικές αρχές σπασίματος των κωδίκων και υποστηρίζει την άποψη ότι ο ανθρώπινος νους μπορεί να σπάσει οποιοδήποτε κρυπτογραφημένο κείμενο που η ανθρώπινη ευρηματικότητα μπορεί να επινοήσει. Ακόμη περιγράφει ένα σύστημα με το οποίο κάθε κρυπτογραφημένο κείμενο που προέρχεται από μια ευρωπαϊκή γλώσσα μπορεί να

αποκρυπτογραφηθεί, αν έχει κρυπτογραφηθεί με αλφαβητική αντικατάσταση, μετρώντας τη συχνότητα των γραμμάτων της γλώσσας.

Ίσως από τα διασημότερα κρυπτογραφήματα, το *σημείωμα του Zimmerman* (*the Zimmerman Note*) ώθησε τις ΗΠΑ στον πρώτο παγκόσμιο πόλεμο. Όταν το κρυπτογράφημα αποκρυπτογραφήθηκε το 1917, οι Αμερικανοί έμαθαν ότι η Γερμανία είχε προσπαθήσει να πείσει το Μεξικό να μπει στον πόλεμο με το μέρος της, υποσχόμενη παραχωρήσεις εδαφών των ΗΠΑ στο Μεξικό.

Τον ίδιο περίπου καιρό, ο Gilbert S. Vernam της AT&T ανέπτυξε τον πρώτο πραγματικά αθραυστο κώδικα που ονομάστηκε βέβαια *κρυπτογράφηση Vernam* ( *The Vernam Cipher* ). Μια ξεχωριστή ιδιότητα αυτού του κώδικα είναι η απαίτηση για ένα κλειδί με μήκος όσο και το μήνυμα που πρέπει να μεταδοθεί και το οποίο δεν ξαναχρησιμοποιείται για την αποστολή άλλου μηνύματος (η κρυπτογράφηση Vernam είναι γνωστή επίσης και ως *κρυπτογράφηση με μπλοκάκι μιας χρήσης* (*one-time-pad*) από την πρακτική της προμήθευσης κατασκόπων με το κείμενο-κλειδί γραμμένο σ'ένα μπλοκάκι του οποίου κάθε κομμάτι χρησιμοποιείται μια φορά και μετά καταστρέφεται). Η ανακάλυψη του συστήματος αυτού δεν εκτιμήθηκε ιδιαίτερα εκείνη την εποχή, πιο πολύ επειδή δεν είχε αποδειχτεί ακόμη ότι είναι άθραυστος κώδικας και επειδή η απαίτηση για πολλά και μεγάλα κλειδιά την έκαναν μη πρακτική για γενική χρήση. Εξαιτίας των μη πρακτικών απαιτήσεων της κρυπρογράφησης Vernam, άλλες (πιο αδύνατες) μέθοδοι συνεχισαν να χρησιμοποιούνται ευρέως. [1] Έτσι, κατά το δεύτερο παγκόσμιο πόλεμο, οι Σύμμαχοι ήταν σε θέση να αποκρυπτογραφούν τα περισσότερα από τα μυστικά μηνύματα που στέλνονταν από τους Γερμανούς. Η εγγενής δυσκολία του σπασίματος των ολοένα και πιο περίπλοκων κρυπτογραφικών μεθόδων ήταν μάλιστα ένας από τους παράγοντες που προώθησε την ανάπτυξη των ηλεκτρονικών υπολογιστών.

Η περίφημη *Μηχανή-Αίνιγμα* (*Enigma*) που χρησιμοποιήθηκε απο τους Γερμανούς κατα το δεύτερο παγκόσμιο πόλεμο για κρυπτογράφηση ραδιοτηλεπικοινωνιών πυροδότησε μια από τις πιο έντονες προσπάθειες αποκρυπτογράφησης στην ιστορία. Ο κώδικας Αίνιγμα θυμίζει έναν παλιότερο κώδικα τύπου Vigenére αλλά είναι πολύ πιο πολύπλοκος. Μια βασική ιδιότητα της μηχανής αυτής ήταν η αυτο-αντιστροφή: εάν το κωδικοποιημένο κείμενο δινόταν ώς είσοδος στη μηχανή, τότε η έξοδος θα ήταν το αρχικό μήνυμα (αν φυσικά η μηχανή είχε την ίδια αρχική κατάσταση με τη μηχανή που είχε κάνει την κωδικοποίηση). Παρόλο που αυτό αποτελούσε τρομερή ευκολία για τους χειριστές της μηχανής, αποδείχτηκε ότι ήταν και μεγάλη αδυναμία του κώδικα Αίνιγμα. Πριν τον πόλεμο, η γαλλική αντικατασκοπεία είχε αποκτήσει αντίγραφα των εντολών της μηχανής-Αίνιγμα και πέρασε την πληροφορία αυτή στους Πολωνούς που υπέκλεπταν και ανέλυαν τις γερμανικές ραδιο-επικοινωνίες. Με τη βοήθεια των εντολών αυτών, οι Πολωνοί κρυπταναλυτές μπόρεσαν να συμπεράνουν τη συνδεσμολογία-καλωδίωση της μηχανής, οπότε έγινε δυνατό να διαβάζονται τα κρυπτογραφημένα κείμενα, αρκεί να είναι γνωστή η αρχική κατάσταση της μηχανής. Παρόλο που οι Βρετανοί τα έμαθαν όλα αυτά απο τους Πολωνούς, είχαν μικρή αξία γι'αυτούς επειδή οι Γερμανοί έκαναν κάποιες τροποποιήσεις στη μηχανή πριν τον πόλεμο. Οι Βρετανοί συγκέντρωσαν μια ομάδα κρυπταναλυτών και μαθηματικών, συμπεριλαμβανομένου και του Alan Turing, σε μια βικτωριανή έπαυλη στο Buckinghamshire που

[1] Αξίζει πάντως να αναφερθεί ότι όταν το 1967 ο στρατός της Βολιβίας συνέλαβε και εκτέλεσε τον επαναστάτη Che Guevara, βρήκαν στην κατοχή του ένα χαρτί που έδειχνε πως προετοίμαζε ένα μήνυμα για αποστολή στον Κουβανό πρόεδρο Fidel Castro. Ο Che Guevara χρησιμοποιούσε τον άσπαστο κώδικα Vernam που είχε εφεύρει ο Vernam 1918.

ονομαζόταν Bletcley Park. Χρησιμοποιώντας τις πληροφορίες των Πολωνών, η ομάδα βάσισε τις προσπάθειές της στη λεγόμενη μέθοδο πιθανής λέξης. Η μέθοδος αυτή βασίζεται στο γεγονός ότι σε κάποιες περιπτώσεις μια συγκεκριμένη ακολουθία συμβόλων σχεδόν σίγουρα αντιπροσωπεύει μια γνωστή λέξη. Μαντεύοντας σωστά μερικές από τις κρυπτογραφημένες λέξεις του κρυπτοκειμένου, μπορούσαν να καθορίζουν τη συνδεσμολογία της μηχανής, δοκιμάζοντας όλες τις πιθανές συνδεσμολογίες και προσδιορίζοντας ποια είχε ως αποτέλεσμα τα υποτιθέμενα ζευγάρια κρυπτογραφημένων-αποκρυπτογραφημένων λέξεων. Ο Turing αντιλήφθηκε ότι μόνο μια αυτόματη και σχετικά γρήγορη μηχανή θα μπορούσε να τα βγάλει πέρα με τις δοκιμές, όποτε και οδηγήθηκε στην κατασκευή ενός εξομοιωτή της μηχανής-Αίνιγμα με το όνομα Bombe.

Η σημερινή μορφή των κρυπτογραφικών συστημάτων έχει καθοριστεί σε πολύ μεγάλο βαθμό από δύο, κεφαλαιώδους σημασίας για την κρυπτογραφία και τις επικοινωνίες γενικότερα, επιστημονικές εργασίες Kerchoff, Shannon, που δημοσιεύτηκαν στα 1883 και 1949 αντίστοιχα. Στην πρώτη, ο Kerchoffs έθεσε τη βασική σχεδιαστική αρχή που έκτοτε διέπει κάθε κρυπτογραφικό σύστημα, σύμφωνα με την οποία η ασφάλεια ενός συστήματος πρέπει να έγκειται *μόνο* στη μυστικότητα του κλειδιού και να μην εξαρτάται από τη μυστικότητα του αλγορίθμου κρυπτογράφησης. Η δεύτερη εργασία ανήκει στο θεμελιωτή της Θεωρίας Πληροφορίας Claude Shannon. Στην εργασία αυτή η κρυπτογραφία μετατρέπεται σε αυστηρό επιστημονικό πεδίο, όπου ορίζεται η έννοια του κρυπτοσυστήματος και η απόλυτη ασφάλεια. Η εργασία αυτή του Shannon αποτέλεσε ισχυρή κινητήρια δύναμη για την ταχεία εξέλιξη της έρευνας στο χώρο της κρυπτογραφίας, η οποία έλαβε χώρα στο δεύτερο μισό του εικοστού αιώνα και συνεχίζεται μέχρι σήμερα. Όλοι οι σύγχρονοι κρυπτογραφικοί αλγόριθμοι σχεδιάζονται υπό το πρίσμα των εννοιών που εισήγαγε ο Shannon. Σημαντική τομή επίσης στο χώρο της κρυπτογραφίας αποτέλεσε η εργασία των Diffie-Hellman το 1976 Diffie, όπου προτάθηκε μία διαφορετική τεχνική κρυπτογράφησης που καλείται *κρυπτογραφία δημοσίου κλειδιού*, η οποία επιλύει προβλήματα που σχετίζονται τόσο με την ασφαλή ανταλλαγή του κλειδιού όσο και με την πιστοποίηση της ταυτότητας των χρηστών

Το ακαδημαϊκό ενδιαφέρον στην κρυπτολογία αναπτύχθηκε πιο έντονα το 1976, όταν οι Whitfield Diffie, Martin E. Hellman και Ralph C. Merkle (τότε στο Stanford University) ανακάλυψαν τις αρχές της κρυπτογραφίας δημόσιου κλειδιού (public key cryptography) [2]. Το 1977 οι Ronald L. Rivest, Adi Shamir και Leonard M. Adleman (τότε στο MIT) σχεδίασαν μια πρακτική υλοποίηση της κρυπτογραφίας δημοσίου κλειδιού, τον αλγόριθμο RSA (βλέπε [10]). Το 1982 ο Shamir έσπασε ένα από τα πρώτα συστήματα κρυπτογράφησης δημοσίου κλειδιού, το κρυπτοσύστημα σακιδίου (knapsack cipher).

Στις αρχές της δεκαετίας του 1980, οι φυσικοί ανέπτυξαν κρυπτογραφικά σχήματα βασισμένα στην κβαντομηχανική.

Θεωρούμε ότι αξίζει εδώ μια ειδική αναφορά μιας άλλης επιστημονικής περιοχής, η οποία συγγενεύει αρκετά με την κρυπτογραφία: της προσπάθειας αποκρυπτογράφησης των αρχαίων γραφών. Είναι χαρακτηριστικό ότι οι πλέον ρηξικέλευθες ανακαλύψεις σε αυτόν τον τομέα δεν έχουν γίνει από θεωρητικούς επιστήμονες αρχαιολόγους, φιλόλογους, αιγυπτιολόγους, κλασικιστές, κλπ, αλλά από μηχανικούς και θετικούς επιστήμονες οι οποίοι έδειξαν ερασιτεχνικό ενδιαφέρον για τα προβλήματα της αποκρυπτογράφησης και τα αντιμετώπισαν με μαθηματικό τρόπο σκέψης.

Η πλέον εντυπωσιακή σχετική ανακάλυψη είναι η αποκρυπτογράφηση της Γραμμικής γραφής Β′ των πινακίδων του μυκηναϊκού πολιτισμού από τον Γάλλο αρχιτέκτονα Michel Ventris κατά

τη δεκαετία του 1950. Μέχρι τότε οι αρχαιολόγοι, σε μια αξιοπερίεργη επίδειξη επιστημονικής πλάνης, θεωρούσαν ότι η γραφή αυτή ανήκε σε σημιτική γλώσσα, όχι ελληνική, παρόλο που ήταν προφανές ότι τα μυκηναϊκά αρχαιολογικά ευρήματα ανήκαν στον πρώιμο πολιτισμό των Ελλήνων και σχετιζόταν με το ιστορικό γεγονός του Τρωικού Πολέμου και πλήθος ελληνικών μυθολογικών παραδόσεων. Ο Ventris όμως βασίστηκε στην εκ των προτέρων παραδοχή ότι η γλώσσα των μυκηναϊκών πινακίδων είναι ελληνική και κατέληξε έτσι στην πλήρη και αναμφισβήτητη αποκρυπτογράφησή της. Αυτή η ανακάλυψη επιβεβαίωσε την ελληνικότητα του μυκηναϊκού πολιτισμού και διέψευσε τις απόψεις περί ελεύσεως των Ελλήνων στην Ελλάδα μετά το 12ο αιώνα π.Χ., οι οποίες επικρατούσαν τότε.

Η Γραμμική γραφή Α′ του μινωικού πολιτισμού, που δεν έχει πλήρως αποκρυπτογραφηθεί, είναι μάλλον επίσης ελληνική και συγγενεύει με τη Γραμμική Β′, καθώς και με συστήματα γραφής των Χετταίων της Ανατολικής Μικράς Ασίας, των Ετρούσκων της Βόρειας Ιταλίας, των Λυδών, των Λυκών και των Κάρων της Δυτικής Μικράς Ασίας.

## 16.2 Κρυπτογραφία - Μια εισαγωγή

Με τον όρο κρυπτολογία αναφερόμαστε τόσο στην κρυπτογραφία όσο και στην κρυπτανάλυση: Η κρυπτογραφία ασχολείται με το σχεδιασμό κρυπτοσυστημάτων, ενώ η κρυπτανάλυση μελετά το σπάσιμο των κρυπτοσυστημάτων.

Ένα κρυπτοσύστημα, γενικά, αποτελείται από δύο αλγορίθμους: έναν αλγόριθμο κρυπτογράφησης ή κωδικοποίησης (encryption or enciphering algorithm) $E$ και έναν αλγόριθμο αποκρυπτογράφησης ή αποκωδικοποίησης (decryption or deciphering algorithm) $D$. (Τυπικός ορισμός του όρου κρυπτοσύστημα δίνεται στο κεφάλαιο 4) Το αρχικό κείμενο (plaintext - απλό κείμενο) είναι το κείμενο (μήνυμα…) προς κρυπτογράφηση. Χρησιμοποιώντας το αρχικό κείμενο για είσοδο του αλγορίθμου κρυπτογράφησης, παίρνουμε στην έξοδο το κρυπτοκείμενο (cryptotext ή ciphertext). Ο αλγόριθμος αποκρυπτογράφησης (η αντίστροφη διαδικασία δηλαδή) χρησιμοποιεί για είσοδο το κρυπτοκείμενο και εξάγει το αντίστοιχο αρχικό κείμενο.

Τα κρυπτοσυστήματα μπορούν να ταξινομηθούν ανάλογα με τον αριθμό των κλειδιών που χρησιμοποιούν:

- Κανένα κλειδί: Αν δεν χρησιμοποιούνται καθόλου κλειδιά, τότε το όλο κρυπτοσύστημα βασίζεται στον αλγόριθμο κρυπτογράφησης και αποκρυπτογράφησης. Αυτός ο αλγόριθμος θα πρέπει να κρατείται μυστικός, δηλαδή να είναι γνωστός μόνο σε εκείνους που είναι υπεύθυνοι για τα κρυπτογραφημένα κείμενα.

- Ένα κλειδί: Κλασική (διπλής κατεύθυνσης) κρυπτογραφία: Οι αλγόριθμοι $E$ και $D$ χρειάζονται μια παράμετρο $k$, η οποία ονομάζεται κλειδί (κρυπτογράφησης-αποκρυπτογράφησης). Θα συμβολίζουμε τις διαδικασίες κρυπτογράφησης και αποκρυπτογράφησης με $E_k$ και $D_k$ αντίστοιχα. Ο αλγόριθμος κρυπτογράφησης-αποκρυπτογράφησης μπορεί να κοινοποιηθεί (να γίνει γνωστός σε όλους) αλλά το κλειδί κρυπτογράφησης θα πρέπει να είναι μυστικό.

- Δύο κλειδιά: Η κρυπτογραφία με δημόσιο κλειδί (ή μονής κατεύθυνσης κρυπτογραφία):

Οι αλγόριθμοι για κρυπτογράφηση και αποκρυπτογράφηση χρησιμοποιούν διαφορετικές παραμέτρους (κλειδιά): Ο αλγόριθμος κρυπτογράφησης χρησιμοποιεί το κλειδί κρυπτογράφησης $k$ (το οποίο μπορεί να κοινοποιηθεί - δημόσιο κλειδί), και ο αλγόριθμος αποκρυπτογράφησης χρησιμοποιεί το κλειδί αποκρυπτογράφησης $k'$, που θα πρέπει να είναι μυστικό.

Ο Sir Francis Bacon (1561-1626) πρότεινε τρεις γενικές απαιτήσεις τις οποίες ένα κρυπτοσύστημα θα πρέπει να πληροί:

1. Δεδομένων των αλγορίθμων κρυπτογράφησης και του αρχικού κείμενου θα πρέπει να είναι εύκολος ο υπολογισμός του κρυπτοκειμένου. (Δεδομένων των $E_k$ και $x$ να είναι εύκολο να υπολογιστεί το $E_k(x)$ ).

2. Αν είναι γνωστό μόνο το κρυπτοκείμενο, χωρίς να είναι γνωστό το $D_k$ (δηλαδή ο αλγόριθμος ή το κλειδί αποκρυπτογράφησης), θα πρέπει να είναι αδύνατο να βρεθεί το αρχικό κείμενο.

3. Το κρυπτοκείμενο δεν θα πρέπει να είναι ύποπτο: να δείχνει αθώο.

Η απαίτηση 3 (μπορεί να επιτευχθεί χρησιμοποιώντας τη μέθοδο παρεμβολής σκουπιδιών - βλέπε παράδειγμα 1.2) δεν θεωρείται σημαντική πλέον, καθώς τόσο τα αρχικά κείμενα όσο και τα κρυπτοκείμενα αναπαρίστανται ως δυαδικές ακολουθίες οι οποίες από μόνες τους δείχνουν αρκετά αθώες. Οι απαιτήσεις (1) και (2) μπορούν να διατυπωθούν χρησιμοποιώντας ορολογία από την θεωρία πολυπλοκότητας: Η κρυπτογράφηση να είναι εύκολη (με χαμηλό πολυωνυμικό χρόνο, γραμμικό χρόνο), ενώ η κρυπτανάλυση να είναι υπολογιστικά απρόσιτη.

Η σύγχρονη κρυπτογραφία θεμελιώνεται πάνω στη θεωρία της υπολογιστικής πολυπλοκότητας. Ένα σύστημα θεωρείται *ασφαλές* όταν έχει αποδειχτεί κάτω όριο για την κρυπτανάλυσή του. Σημειώνουμε ότι κατά το σχεδιασμό ενός κρυπτοσυστήματος, μας ενδιαφέρουν τα κάτω όρια ($\Omega$) ή η μέση πολυπλοκότητα του αντιστοίχου προβλήματος κρυπτανάλυσης, και όχι η πολυπλοκότητα της χειρότερης περίπτωσης ( άνω όρια - $O$ ).

Η κρυπτανάλυση είναι δυνατόν να είναι είτε παθητική (υποκλοπή: ο κρυπταναλυτής προσπαθεί να αποκρυπτογραφήσει μηνύματα, να βρει κλειδιά, …) ή ενεργητική (παρεμβολή: ο κρυπταναλυτής παραποιεί δεδομένα που διαβιβάζονται, γράφει και διαβιβάζει δικά του κείμενα, ξαναδιαβιβάζει παλιό κρυπτοκείμενο, …)

Η κρυπτανάλυση ενός συστήματος (με κλειδί) μπορεί να ξεκινήσει από διαφορετικά αρχικά σενάρια:

- Το κρυπτοκείμενο μόνο: Μόνο το κρυπτοκείμενο είναι γνωστό στον κρυπταναλυτή.
- Γνωστό αρχικό κείμενο: Ο κρυπταναλυτής γνωρίζει κάποια ζευγάρια $(x, E_k(x))$ (αρχικών κειμένων και των αντιστοίχων κρυπτοκειμένων) και προσπαθεί να βρει το κλειδί $k$.
- Επιλεγμένο αρχικό κείμενο: Μερικά ζευγάρια $(x, E_k(x))$ είναι γνωστά στον κρυπταναλυτή όπου, επιπλέον, το αρχικό κείμενο $x$ έχει επιλεγεί από τον ίδιο τον κρυπταναλυτή.

- Επιλεγμένο κρυπτοκείμενο: Μερικά ζευγάρια $(x, y = E_k(x))$ είναι γνωστά στον κρυπταναλυτή όπου, επιπλέον, το κρυπτοκείμενο $y$ έχει επιλεγεί από τον ίδιο τον κρυπταναλυτή.
- Κλειδί κρυπτογράφησης: Ο κρυπταναλυτής γνωρίζει το κλειδί $k$, και συνεπώς και τη μέθοδο κρυπτογράφησης $E_k$, και προσπαθεί να βρει το κλειδί $k'$ και άρα τη μέθοδο αποκρυπτογράφησης $D_{k'}$. (Αυτό το σενάριο είναι τυπικό για τα κρυπτοσυστήματα δημοσίου κλειδιού).

Οι αλγόριθμοι κρυπτογράφησης και αποκρυπτογράφησης είναι γνωστοί στον κρυπταναλυτή και στα τέσσερα παραπάνω σενάρια — όχι όμως και το κλειδί αποκρυπτογράφησης.

Οι κρυπταναλυτικές επιθέσεις εναντίον κλασικών συστημάτων βασίζονται γενικά στη θεωρία πιθανοτήτων, στη στατιστική, στη γραμμική άλγεβρα, στην αφηρημένη άλγεβρα (θεωρία ομάδων) και στη θεωρία πολυπλοκότητας, ενώ η κρυπτανάλυση των συστημάτων δημοσίου κλειδιού βασίζεται κυρίως στη θεωρία αριθμών και στη θεωρία υπολογιστικής πολυπλοκότητας.

Ένας σημαντικός παράγοντας για την κρυπτανάλυση είναι η επιπλέον πληροφορία που διαθέτει ο κρυπταναλυτής (side information): Ο κρυπταναλυτής μπορεί να είναι οικείος με συγκεκριμένα στιγμιότυπα του συστήματος που σκοπεύει να σπάσει, ή ίσως να μπορεί να κάνει μερικές παραδοχές για το κρυπτοκείμενο, που θα του δώσουν ένα μεγάλο πλεονέκτημα στην επίθεσή του εναντίον του συστήματος. Για παράδειγμα η επίθεση εναντίον του γερμανικού συστήματος "Αίνιγμα" (Enigma), βασίστηκε στη θεώρηση ότι κάποια λέξη στο κρυπτοκείμενο θα έπρεπε να αντιστοιχεί στη λέξη "Unterseeboot" (υποβρύχιο). Ένα άλλο παράδειγμα επιπλέον πληροφόρησης είναι το ακόλουθο:

Θεωρήστε τον ακόλουθο γρίφο. Ξεδιαλύνετε κάθε έναν από τους εξής αναγραμματισμούς:

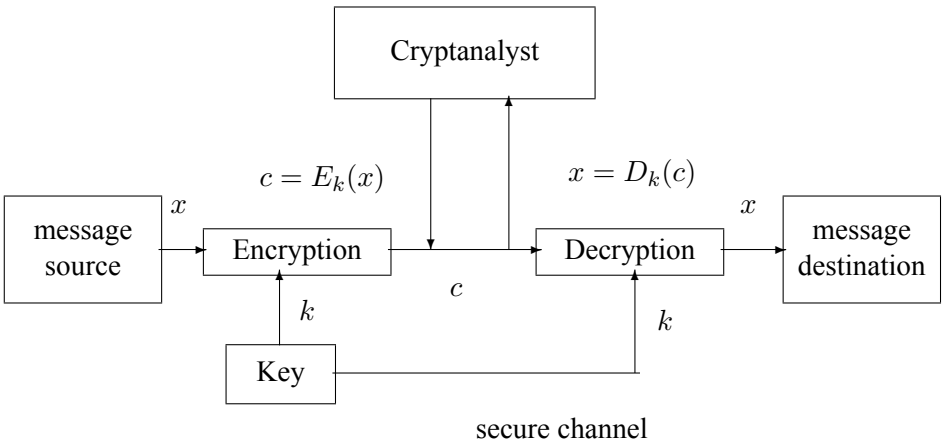
```
  DFOR    STOUL  IIBAGONMRLH  ECSEEMDR
IAERFRR  GEDOD    LADCLIAC    RITSAAEM
```

Δεν είναι δύσκολο να αναγνωρίσετε τις λέξεις, αλλά ο γρίφος γίνεται αρκετά εύκολος αν χρησιμοποιήσουμε την επιπλέον πληροφορία-υπόδειξη: "οι λέξεις αντιστοιχούν σε ονόματα αυτοκινήτων"

## 16.3 Κλασικά Συστήματα

Στην κλασική διπλής κατεύθυνσης κρυπτογραφία, οι αλγόριθμοι κρυπτογράφησης και αποκρυπτογράφησης είναι γνωστοί σε όλους, και το ίδιο κλειδί χρησιμοποιείται και για τις δύο κατευθύνσεις (κρυπτογράφηση-αποκρυπτογράφηση). Με άλλα λόγια, στα κλασικά συστήματα, η αποκρυπτογράφηση είναι εύκολη αν το κλειδί κρυπτογράφησης είναι γνωστό. Αντίθετα, στην κρυπτογραφία δημοσίου κλειδιού το κλειδί $k$ μπορεί με ασφάλεια να δημοσιοποιηθεί χωρίς να αποκαλυφθεί το κλειδί $k'$. Για αυτό το λόγο τα κλασικά συστήματα αναφέρονται επίσης και ως *συμμετρικά* ή *διπλής κατεύθυνσης συστήματα*, και τα συστήματα δημοσίου κλειδιού ως *μη-συμμετρικά* ή *μονής κατεύθυνσης* συστήματα (αυτό σημαίνει ότι η διαδικασία κρυπτογράφησης είναι μονής κατεύθυνσης - δεν μπορεί εύκολα να αντιστραφεί).

Μια (πολύ παλιά) κατηγοριοποίηση των κλασικών κρυπτοσυστημάτων είναι σε συστήματα *αντικατάστασης (substitution)* και *μετάθεσης (permutation)* (ή *αναδιάταξης (transposition)*).




Σχήμα 16.1: Συμβατικά (κλασικά) κρυπτοσυστήματα διπλής κατεύθυνσης

Στα συστήματα αντικατάστασης (substitution ciphers), τα γράμματα του αρχικού κειμένου αντικαθίστανται από άλλα τα οποία διατηρούνται στην ίδια διάταξη όπως και τα πρωτότυπά τους στο αρχικό κείμενο. Αν οι αντικαταστάτες παραμένουν οι ίδιοι σε όλο το κείμενο (κάθε γράμμα του αρχικού κειμένου αναπαρίσταται πάντοτε από το ίδιο σύμβολο-αντικαταστάτη) τότε το σύστημα ονομάζεται *μονοαλφαβητικό*. Αν το αρχικό κείμενο είναι σε κάποια φυσική γλώσσα, η κρυπτανάλυση είναι πάντοτε εφικτή βασιζόμενη στη στατιστική κατανομή των γραμμάτων. Στα *πολυαλφαβητικά* συστήματα αντικατάστασης κάθε γράμμα του αρχικού κειμένου μπορεί να έχει πολλούς αντικαταστάτες και κάθε φορά χρησιμοποιείται διαφορετικός αντικαταστάτης.

Στα συστήματα μετάθεσης (ή αναδιάταξης) τα γράμματα του αρχικού κειμένου αναδιατάσσονται. Αυτή η μέθοδος είναι υπερβολικά απλή, οπότε θα πρέπει να συνδυαστεί με κάποια άλλη ιδέα (μέθοδο παρεμβολής σκουπιδιών, …). Ένα παράδειγμα συστήματος που χρησιμοποιεί τη μέθοδο παρεμβολής σκουπιδιών δίνεται στο παράδειγμα 16.1 παρακάτω.

Μια άλλη κατηγοριοποίηση των κρυπτοσυστημάτων θα μπορούσε να είναι σε συστήματα αντικατάστασης *χωρίς συμφραζόμενα (context-free)* και σε συστήματα αντικατάστασης *με συμφραζόμενα (context-sensitive)* : Στα συστήματα χωρίς συμφραζόμενα κάθε γράμμα κωδικοποιείται ξεχωριστά ενώ σε εκείνα με συμφραζόμενα η κωδικοποίηση γίνεται ανά ομάδες (blocks).

*Παράδειγμα* 16.1. Μέθοδος Παρεμβολής Σκουπιδιών Σύστημα RICHELIEU

Αυτό το σύστημα χρησιμοποιεί τη μέθοδο “παρεμβολής σκουπιδιών”, και μπορεί να κρυπτογραφήσει ένα αρχικό κείμενο παράγοντας κρυπτοκείμενο που δείχνει πολύ αθώο …. Ο Richelieu χρησιμοποιούσε κομάτια χαρτονιού με τρύπες. Μόνο τα γράμματα που φαίνονται από τις τρύπες είναι σημαντικά. Για παράδειγμα, θεωρείστε ένα block 70 χαρακτήρων διατεταγμένων σε 7 γραμμές και 10 στήλες. Επίσης θεωρείστε ότι οι τρύπες είναι στις θέσεις:

$$(1,8),(2,9),(3,6),(4,5),(4,6),(5,1),(5,6),(5,7),(5,9),(6,2),(6,10),(7,9),(7,10)$$

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | | | | | | □ | | |
| 2 | | | | | | | | | □ | |
| 3 | | | | | | □ | | | | |
| 4 | | | | | □ | □ | | | | |
| 5 | □ | | | | | □ | □ | | □ | |
| 6 | | □ | | | | | | | | □ |
| 7 | | | | | | | | | □ | □ |

Κρυπτοκείμενο:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| I | | L | O | V | E | | Y | O | U |
| I | | H | A | V | E | | Y | O | U |
| D | E | E | P | | U | N | D | E | R |
| M | Y | | S | K | I | N | | M | Y |
| L | O | V | E | | L | A | S | T | S |
| F | O | R | | E | V | E | R | I | N |
| H | Y | P | E | R | S | P | A | C | E |

Το αρχικό κείμενο εδώ είναι YOU KILL AT ONCE, ενώ το κρυπτοκείμενο είναι αρκετά αθώο (άσχετο).

### 16.3.1 Μονοαλφαβητικά Συστήματα Αντικατάστασης

Ένα κρυπτοσύστημα ονομάζεται μονοαλφαβητικό αν κάθε γράμμα του αρχικού κειμένου αναπαρίσταται πάντοτε από το ίδιο σύμβολο-αντικαταστάτη (κάθε εμφάνιση ενός συμβόλου του αρχικού κειμένου κρυπτογραφείται με τον ίδιο πάντα αντικαταστάτη).

*Παράδειγμα* 16.2. Το Κρυπτοσύστημα του ΚΑΙΣΑΡΑ

Το κρυπτοσύστημα του Καίσαρα είναι από τα πρώτα κρυπτογραφικά σχήματα που χρησιμοποιήθηκαν. Είναι επίσης πολύ απλό, και μπορεί κανείς να το σπάσει πολύ εύκολα. Το σύστημα του Καίσαρα βασίζεται σε αντικαταστάσεις (μονοαλφαβητικό σύστημα αντικατάστασης): ολίσθηση κατά $k$ θέσεις, δηλαδή κάθε γράμμα αντικαθίσταται με άλλο, προχωρώντας $k$ θέσεις στο αλφάβητο modulo το μέγεθος του αλφαβήτου. ($k = 1, \ldots, 25$). Π.χ. για $k = 3$:

Τα γράμματα του αρχικού κειμένου:
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Οι αντικαταστάτες του κρυπτοκειμένου:
D E F G H I J K L M N O P Q R S T U V W X Y Z A B C

Για παράδειγμα το αρχικό κείμενο I LOVE MATH κρυπτογραφείται ως LORYH PDWK. Η μέθοδος κρυπτογράφησης $E_k$ είναι ολίσθηση μπροστά κατά $k$ βήματα στο αλφάβητο, και η μέθοδος αποκρυπτογράφησης είναι ολίσθηση πίσω κατά $k$ βήματα στο αλφάβητο. Παρακάτω δίνονται κάποιες προφανείς ιδιότητες των $E_k$ και $D_k$ :

$$E_i \circ D_j = D_j \circ E_i \text{ (αντιμεταθετική ιδιότητα)}$$

$$D_k = E_{26-k}\ ,\ D_k \circ E_k = D_0 = E_0 = D_{26} = E_{26}.$$

Αν το αρχικό κείμενο ανήκει σε μια γνωστή φυσική γλώσσα, η κρυπτανάλυση αυτού του συστήματος είναι πολύ εύκολη: από τη στιγμή που ο συνολικός αριθμός των δυνατών κλειδιών είναι αρκετά μικρός (25) μπορεί κανείς απλά να τα δοκιμάσει όλα σε ένα μικρό μέρος του κρυπτοκειμένου και να δει ποιο από όλα οδηγεί σε αρχικό κείμενο που έχει νόημα. Επίσης, όπως και όλα τα μονοαλφαβητικά συστήματα αντικατάστασης, το σύστημα του Καίσαρα μπορεί να το σπάσει κανείς υπολογίζοντας τις συχνότητες εμφάνισης των γραμμάτων.

□

*Παράδειγμα* 16.3. Μια παραλλαγή: Το σύστημα του Καίσαρα με λέξη-κλειδί (KEYWORD-CAESAR system)

Για ένα Καισαρικό κρυπτοσύστημα μπορούμε να χρησιμοποιήσουμε οποιαδήποτε αντιμετάθεση του χρησιμοποιούμενου αλφαβήτου, προκειμένου να ορίσουμε τους αντικαταστάτες για το κάθε γράμμα του αλφαβήτου. Το κλειδί για το συγκεκριμένο σύστημα είναι η αντιμετάθεση που χρησιμοποιείται. Το σύστημα του Καίσαρα με λέξη κλειδί παράγει μια συγκεκριμένη αντιμετάθεση χρησιμοποιώντας έναν αριθμό (κλειδί) και μια λέξη ή μια μικρή φράση (λέξη-κλειδί): Επιλέγουμε έναν αριθμό $k$ τέτοιον ώστε $0 \leq k \leq 26$ (=πληθάριθμος του αλφαβήτου του απλού κειμένου) και μια λέξη-κλειδί (ή φράση-κλειδί) στην οποία όλα τα γράμματα είναι διαφορετικά. Για παράδειγμα, θεωρείστε $k = 7$, και τη λέξη-κλειδί "TEN FOUR". Γράψτε τη λέξη-κλεδί κάτω από το αλφάβητο, ξεκινώντας από το $k$-οστό γράμμα:

```
0             7                                   25
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
              T E N F O U R
```

Συμπληρώστε τη δεύτερη γραμμή με τα υπόλοιπα γράμματα σε αλφαβητική σειρά, μετά τη λέξη-κλειδί:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
P S V W X Y Z T E N F O U R A B C D G H I J K L M Q
```

Δεν είναι απαραίτητο τα γράμματα στη λέξη-κλειδί να είναι διαφορετικά: απλά θα μπορούσε να γραφεί η λέξη-κλειδί χωρίς επαναλήψεις. Ο αριθμός των κλειδιών σε ένα τέτοιο σύστημα είναι μεγάλος ( υπάρχουν 26! αντιμεταθέσεις για το Αγγλικό αλφάβητο ), οπότε η κρυπτανάλυση θα πρέπει να βασιστεί στη μέτρηση συχνοτήτων. □

Ένας (απλός) τρόπος άμυνας απέναντι στην κρυπτανάλυση τη βασισμένη σε μετρήσεις συχνότητας, είναι το κρυπτοσύστημα με ομόφωνα (HOMOPHONES): Τα γράμματα του αρχικού κειμένου έχουν πολλούς αντικαταστάτες (λέγονται *ομόφωνα-homophones*) - ο αριθμός των αντικαταστατών είναι ανάλογος της συχνότητας με την οποία εμφανίζεται το γράμμα στη γλώσσα του απλού κειμένου. Σε αυτό το σύστημα προκειμένου να κρυπτογραφήσουμε την εμφάνιση κάποιου γράμματος επιλέγουμε τυχαία έναν από τους αντικαταστάτες. Το σύστημα αυτό δεν είναι μονοαλφαβητικό, και επιπλέον, η μέθοδος κρυπτογράφησης δεν είναι (ντετερμινιστική) συνάρτηση. Σημειώνουμε ότι ένα απλό κείμενο είναι δυνατόν να αντιστοιχεί σε αρκετά διαφορετικά

κρυπτοκείμενα, αντιθέτως ένα κρυπτοκείμενο αντιστοιχεί μόνο σε ένα απλό κείμενο. Χρησιμοποιώντας την παραπάνω μέθοδο, μπορούμε να δούμε ότι κάθε σύμβολο στο κρυπτοκείμενο εμφανίζεται με την ίδια συχνότητα. Η κρυπτανάλυση του συγκεκριμένου συστήματος μπορεί να βασιστεί σε στατιστικές κατανομές ζευγαριών ή τριάδων γραμμάτων (στη γλώσσα του απλού κειμένου).

### 16.3.2 Πολυαλφαβητικά Συστήματα Αντικατάστασης

Αν σε ένα κρυπτοσύστημα, κάθε (διατεταγμένο) ζευγάρι γραμμάτων (ή block n γραμμάτων — n-γράμματο) του αλφαβήτου του απλού κειμένου κωδικοποιείται πάντοτε κατά τον ίδιο τρόπο (με τον ίδιο αντικαταστάτη) τότε το σύστημα εξακολουθεί να είναι μονοαλφαβητικό, και συνήθως λέγεται *μονοαλφαβητικό με την ευρεία έννοια (monoalphabetic in a wider sense)*. Στα πολυαλφαβητικά κρυπτοσυστήματα αντικατάστασης, ένα γράμμα δεν αντικαθίσταται από το ίδιο σύμβολο παντού στο κείμενο: η χρήση των αντικαταστατών ποικίλει στα διάφορα μέρη του απλού κειμένου, και δεν είναι καν μονοαλφαβητική με την ευρεία έννοια (η αντικατάσταση του κάθε γράμματος του κειμένου γίνεται *με συμφραζόμενα*. Παρατηρήστε ότι τα μονολφαβητικά συστήματα με την ευρεία έννοια είναι επίσης με συμφραζόμενα).

Η γερμανική μηχανή Αίνιγμα θεωρείται σύστημα πολυαλφαβητικής αντικατάστασης (μετά από κάθε γράμμα του κειμένου, τα γρανάζια γυρίζουν, δίνοντας ένα νέο πρότυπο κρυπτογράφησης).

**Ο κώδικας αντικατάστασης 2-γραμμάτων PLAYFAIR** (Βαρώνος Playfair του Αγ. Ανδρέα)

Το κρυπτοσύστημα Playfair βασίζεται στο Αγγλικό αλφάβητο, και είναι μονοαλφαβητικό σύστημα με την ευρεία έννοια.

Διατάσσουμε τα γράμματα του Αγγλικού αλφαβήτου, παραλείποντας το J (το J και το I θεωρούνται σαν το ίδιο γράμμα) σε έναν πίνακα $5 \times 5$ (που λέγεται τετράγωνο Playfair), με κάποιο τυχαίο τρόπο. Συνήθως το τετράγωνο Playfair δημιουργείται βασισμένο σε μια λέξη-κλειδί (ή φράση): τα γράμματα της λέξης κλειδί (οι επαναλήψεις διαγράφονται) τοποθετούνται κατά σειρές, ακολουθούμενα από τα υπόλοιπα γράμματα σε αλφαβητική σειρά. (Οι λέξεις-κλειδιά χρησιμοποιούνται προκειμένου να γίνει η διαχείρηση του κλειδιού ευκολότερη) Για παράδειγμα, η λέξη-κλειδί PLAYFAIR δίνει τον παρακάτω πίνακα:

| | | | | |
|---|---|---|---|---|
| P | L | A | Y | F |
| I | R | B | C | D |
| E | G | H | K | M |
| N | O | Q | S | T |
| U | V | W | X | Z |

Χωρίζουμε το αρχικό κείμενο σε blocks που αποτελούνται από δύο γράμματα το καθένα (2-γράμματα). Δεν πρέπει κανένα block να περιέχει δύο φορές το ίδιο γράμμα: οποτεδήποτε συμβαίνει αυτό, παρεμβάλουμε (μεταξύ των δύο γραμμάτων στο αρχικό κείμενο) ένα *μηδενικό χαρακτήρα* (συνήθως το Q). Επίσης το απλό κείμενο πρέπει να περιέχει άρτιο αριθμό γραμμάτων (διαφορετικά βάζουμε ένα μηδενικό χαρακτήρα στο τέλος του). Οι διαδικασίες κρυπτογράφησης και αποκρυπτογράφησης λειτουργούν στα 2-γράμματα όπως φαίνεται παρακάτω:

*Κρυπτογράφηση:*

Αν τα δύο γράμματα του block δεν είναι στην ίδια γραμμή ή στήλη του τετραγώνου Playfair, τότε κωδικοποιούμε χρησιμοποιώντας τις άλλες δύο γωνίες του ορθογωνίου που ορίζεται από τα δύο γράμματα του block. Για παράδειγμα, χρησιμοποιώντας το παραπάνω τετράγωνο Playfair, το 2-γραμμα AD ορίζει το ορθογώνιο AFBD και κρυπτογραφείται ως FB.

Αν τα δύο γράμματα του block είναι στην ίδια γραμμή (στήλη), επιλέγουμε (κυκλικά) τα γειτονικά γράμματα δεξιά (κάτω) για κάθε γράμμα του block. Έτσι, το RC κωδικοποιείται με το BD, το AW με το BA, το UV με το VW.

Με αυτή τη μέθοδο το αρχικό κείμενο MATHEMATICS χωρίζεται σε 2-γράμματα ως MA TH EM AT IC SQ και κρυπτογραφείται: HF QM GE FQ RD TS.

Η μέθοδος αποκρυπτογράφησης είναι προφανής: αφού το PLAYFAIR είναι ένα κλασικό διπλής-κατεύθυνσης κρυπτοσύστημα η αποκρυπτογράφηση προκύπτει (εύκολα) από τη μέθοδο κρυπτογράφησης.

Η κρυπτανάλυση αυτού του συστήματος μπορεί επίσης να βασιστεί στη μέθοδο της μέτρησης συχνοτήτων (όπως και για όλα τα μονοαλφαβητικά συστήματα αντικατάστασης), αλλά η ανάλυση είναι πιο πολύπλοκή καθώς η κρυπτογράφηση βασίζεται σε 2-γράμματα, και όχι σε απλά γράμματα. □

Τα πολυαλφαβητικά συστήματα παρέχουν μια πολύ καλή άμυνα εναντίον της μέτρησης συχνοτήτων διότι κάθε γράμμα δεν αναπαρίσταται με το ίδιο σύμβολο παντού στο απλό κείμενο.

**Κρυπτοσύστημα VIGENÉRE** ( Blaise de Vigenére 1523-1596)

Το σύστημα του Vigenére είναι από τα πιο παλιά και τα πιο γνωστά πολυαλφαβητικά κρυπτοσυστήματα (στην πραγματικότητα ο κώδικας Αίνιγμα και ο κώδικας Vernam είναι Vigenére συστήματα). Αρχικά αντιστοιχίζουμε σε κάθε γράμμα του αλφαβήτου έναν αριθμό $(A-0, B-1, \ldots, Z-25)$. Το VIGENÉRE μπορεί να θεωρηθεί ένα σύστημα του Καίσαρα στο οποίο το κλειδί αλλάζει από βήμα σε βήμα. Το κλειδί στο σύστημα VIGENÉRE είναι ένα διάνυσμα $k = (k_0, k_1, \ldots, k_{r-1})$. Αυτό το διάνυσμα είναι η λέξη-κλειδί, και μπορεί να είναι οποιαδήποτε λέξη ή φράση (επαναλήψεις γραμμάτων επιτρέπονται). Ο αριθμός $r$ λέγεται περίοδος του συστήματος. Το αρχικό κείμενο διαιρείται σε blocks μεγέθους $r$ και κάθε block κρυπτογραφείται χρησιμοποιώντας τη λέξη-κλειδί ως εξής: γράφουμε τη λέξη-κλειδί κάτω από το block του αρχικού κειμένου και κρυπτογραφούμε κάθε γράμμα του κειμένου χρησιμοποιώντας ένα σύστημα του Καίσαρα με το $k$ να ισούται με τον αριθμό που αντιστοιχεί στο γράμμα της λέξης-κλειδί που είναι γραμμένη από κάτω. Έτσι η αντικατάσταση VIGENÉRE για κάθε block του αρχικού κειμένου ορίζεται ως:

$$f : (x_0, \ldots, x_{r-1}) \to (x_0 + k_0, \ldots, x_{r-1} + k_{r-1})$$

Η κρυπτανάλυσή του γίνεται με τον ίδιο ακριβώς τρόπο με τον οποίο γίνεται και η κρυπτανάλυση του συστήματος του Καίσαρα. Τα γράμματα ανά $r$ έχουν κρυπτογραφηθεί με τον ίδιο αριθμό ολίσθησης. Αν όμως δεν γνωρίζουμε το μήκος $r$ της λέξης-κλειδί θα πρέπει να το βρούμε. Αυτό γίνεται δύσκολα, με παρατήρηση των επαναλήψεων ίδιων ακολουθιών γραμμάτων σε διαφορετικά σημεία του κειμένου. Αν π.χ. επαλαμβάνεται μια λέξη σε δύο σημεία με απόσταση $m$

μεταξύ τους, τότε ίσως πρόκειται για την ίδια λέξη που κρυπτογραφήθηκε με το ίδιο μέρος του κλειδιού. Τότε το $m$ είναι πολλαπλάσιο του $r$.

Οι διαδικασίες κρυπτογράφησης και αποκρυπτογράφησης μπορούν να εκτελεστούν και χωρίς υπολογιστή χρησιμοποιώντας έναν πίνακα παρόμοιο με τον παρακάτω, στον οποίο κάθε στήλη μπορεί να θεωρηθεί ως ένα σύστημα του Καίσαρα. Για να χρησιμοποιήσουμε αυτόν τον πίνακα διαβάζουμε το κείμενο από την πρώτη στήλη, το κλειδί από την πρώτη γραμμή και το κρυπτοκείμενο είναι το γράμμα που βρίσκεται στην τομή τους. Για την αποκρυπτογράφηση, βρίσκουμε το γράμμα του κρυπτοκειμένου στη στήλη που υποδεικνύεται από το κλειδί και διαβάζουμε το αντίστοιχο γράμμα του αρχικού κειμένου από την πρώτη στήλη της γραμμής στην οποία βρίσκεται το γράμμα του κρυπτοκειμένου. (Σημειώνεται ότι ο ρόλος των γραμμών και των στηλών μπορεί να εναλλαχθεί)

| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A |
| C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B |
| D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C |
| E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D |
| F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E |
| G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F |
| H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G |
| I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H |
| J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I |
| K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J |
| L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K |
| M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L |
| N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M |
| O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N |
| P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P |
| R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q |
| S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R |
| T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S |
| U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T |
| V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U |
| W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V |
| X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W |
| Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X |
| Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y |

Η κρυπτογράφηση του αρχικού κειμένου χρειάζεται μια λέξη-κλειδί μήκους $r$ η οποία επαναλαμβάνεται για να καλύψει όλο το αρχικό κείμενο. Τέτοια πολυαλφαβητικά συστήματα, όπου τα αλφάβητα των αντικαταστατών επαναλαμβάνονται περιοδικά συνήθως ονομάζονται *περιοδικά*.

Αν γνωρίζουμε την περίοδο κάποιου περιοδικού πολυαλφαβητικού συστήματος, τότε η κρυπτανάλυσή του μπορεί να αναχθεί στην κρυπτανάλυση ενός μονοαλφαβητικού συστήματος: Θεωρούμε ότι η περιοδός είναι $r$. Διατάσσουμε τα γράμματα του κρυπτοκειμένου σε γραμμές με $r$ στήλες σε κάθε γραμμή (γράφουμε $r$ γράμματα σε κάθε γραμμή). Δύο εμφανίσεις του ίδιου γράμματος στην ίδια στήλη αντιπροσωπεύουν το ίδιο αρχικό γράμμα. Οπότε μπορούμε να αποκρυπτογραφήσουμε κάθε στήλη με μέτρηση συχνοτήτων.

Αν η περίοδος που χρησιμοποιείται σε ένα περιοδικό σύστημα τύπου VIGENÉRE είναι άγνωστη, μπορεί (ίσως) να βρεθεί με την *μέθοδο του Kasiski* (F.W. Kasiski, 1860): Αυτή η μέθοδος υπολογίζει την περίοδο ψάχνοντας τις εμφανίσεις της ίδιας λέξης στο κρυπτοκείμενο. Υποθέτουμε ότι μια συγκεκριμένη λέξη εμφανίζεται δύο φορές στο κρυπτοκείμενο, και ότι μεσολαβούν $m$ γράμματα μεταξύ των εμφανίσεων αυτών (δηλαδή από την αρχή της πρώτης μέχρι την αρχή της δεύτερης). Αυτό μπορεί να οφείλεται στο ότι οι δύο εμφανίσεις αντιστοιχούν στο ίδιο κομμάτι του αρχικού κειμένου, που έχει κρυπτογραφηθεί ξεκινώντας από την ίδια θέση του κλειδιού. Σε

αυτήν την περίπτωση, η απόσταση $m$ μεταξύ των δύο εμφανίσεων στο κρυπτοκείμενο θα πρέπει να είναι πολλαπλάσιο του μήκους του κλειδιού. Αν βρεθούν αρκετές τέτοιες διπλές εμφανίσεις στο κρυπτοκείμενο, μπορούμε να κάνουμε μια καλή εκτίμηση για το μήκος του κλειδιού.

**Το κρυπτοσύστημα AUTOCLAVE** (G. Cardano[2])

Το σύστημα AUTOCLAVE είναι μια παραλλαγή του γενικού συστήματος VIGENÉRE όπου το αρχικό κείμενο χρησιμοποιείται επίσης και ως κλειδί κρυπτογράφησης, μετά από κάποια ολίσθηση. Συνήθως, μια λέξη-κλειδί χρησιμοποιείται στην αρχή του κειμένου (η οποία δίνει την απαιτούμενη ολίσθηση). Για παράδειγμα, το αρχικό κείμενο EVERYTHING IS MATHEMATICS, χρησιμοποιώντας τη λέξη-κλειδί CRYPTO θα κρυπτογραφηθεί ως εξής:

| | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Αρχικό κείμενο | E | V | E | R | Y | T | H | I | N | G | I | S | M | A | T | H | E | M | A | T | I | C | S |
| Κλειδί | C | R | Y | P | T | O | E | V | E | R | Y | T | H | I | N | G | I | S | M | A | T | H | E |
| Κρυπτοκείμενο | G | M | C | G | R | H | L | D | R | X | G | L | T | I | G | N | M | E | M | T | B | J | W |

Σε μια παραλλαγή του AUTOCLAVE, το κρυπτοκείμενο που δημιουργείται με τον παραπάνω τρόπο, χρησιμοποιείται ως κλειδί κρυπτογράφησης μετά τη λέξη κλειδί κ.ο.κ. Για παράδειγμα:

| | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Αρχικό κείμενο | E | V | E | R | Y | T | H | I | N | G | I | S | M | A | T | H | E | M | A | T | I | C | S |
| Κλειδί | C | R | Y | P | T | O | G | M | C | G | R | H | N | U | P | M | Z | Z | Z | U | I | T | D |
| Κρυπτοκείμενο | G | M | C | G | R | H | N | U | P | M | Z | Z | Z | U | I | T | D | L | Z | N | Q | V | V |

Η κρυπτανάλυση του συστήματος AUTOCLAVE είναι σχετικά απλή: Χρειάζεται μόνο να μαντέψουμε το μήκος της λέξης-κλειδί, έστω $n$. Στο παραπάνω κρυπτοκείμενο, αν γνωρίζουμε το μήκος της λέξης κλειδί (= 6), τότε έχουμε το εξής σενάριο:

| | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Κρυπτοκείμενο | G | M | C | G | R | H | N | U | P | M | Z | Z | Z | U | I | T | D | L | Z | N | Q | V | V |
| Κλειδί | | | | | | G | M | C | G | R | H | N | U | P | M | Z | Z | Z | U | I | T | D | |
| Αρχικό κείμενο | | | | | | H | I | N | G | I | S | M | A | T | H | E | M | A | T | I | C | S | |

Η κρυπτανάλυση του προηγούμενου συστήματος AUTOCLAVE έχει ως εξής: Αρχικά χρησιμοποιούμε τη μέθοδο Kasiski για να καθορίσουμε το μήκος του κλειδιού (την *περίοδο*). Η μέθοδος του Kasiski δεν είναι τόσο καλή για τα συστήματα AUTOCLAVE όσο είναι για το σύστημα VIGENÉRE, αλλά συνήθως είναι αρκετά καλή. Για παράδειγμα, ας υποθέσουμε ότι το αρχικό κείμενο περιέχει δύο εμφανίσεις της λέξη THE και ότι η απόσταση μεταξύ αυτών των δύο εμφανίσεων είναι δύο φορές η περίοδος. Τότε κάποια ακολουθία τριών γραμμάτων, έστω AID θα βρίσκεται στο ενδιάμεσο των δύο εμφανίσεων. Κατά τη διαδικασία κρυπτογράφησης θα έχουμε:

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Αρχικό κείμενο | … | T | H | E | … | A | I | D | ... | T | H | E ... |
| Κλειδί | ... | | | | ... | T | H | E | ... | A | I | D ... |
| Κρυπτοκείμενο | ... | | | | ... | T | P | H | ... | T | P | H ... |

[2]G. Cardano(1501-1576), Ιταλός Μαθηματικός, γιατρός και φιλόσοφος - Τύποι επίλυσης εξισώσεων τρίτου και τετάρτου βαθμού

οπότε, συναντάμε το ΤΡΗ δύο φορές στο κρυπτοκείμενο και η απόσταση μεταξύ των δύο εμφανίσεων δίνει την περίοδο του συστήματος. Από τη στιγμή που είναι γνωστή η περίοδος, βρίσκουμε τη λέξη-κλειδί με εξαντλητικό ψάξιμο βασισμένο στη μέτρηση συχνοτήτων του κάθε γράμματος.

Υπάρχει επίσης μια παραλλαγή του κώδικα PLAYFAIR, το ΠΕΡΙΟΔΙΚΟ PLAYFAIR, το οποίο είναι πολυαλφαβητικό (και εξαρτάται από τα συμφραζόμενα, όπως και το μονοαλφαβητικό PLAYFAIR): Αντί για ένα τετράγωνο PLAYFAIR, χρησιμοποιούμε περισσότερα τετράγωνα, έστω $k$. Το πρώτο γράμμα του αρχικού κειμένου, κρυπτογραφείται σύμφωνα με το πρώτο τετράγωνο, το δεύτερο ώς $k$-οστό γράμμα με το δεύτερο ως $k$-οστό τετράγωνο αντίστοιχα και το $k+1$-οστό γράμμα με το πρώτο τετράγωνο κ.ο.κ.

**Ο κώδικας Vernam ή Μπλοκάκι μιας Χρήσης** (Vernam, 1917)

Το μπλοκάκι (για το κλειδί) μιας χρήσης είναι ένα τέλειο κρυπτογραφικό σύστημα που παρέχει απόλυτη μυστικότητα με την έννοια ότι το κρυπτοκείμενο δεν περικλείει καμία απολύτως πληροφορία σχετικά με το αρχικό κείμενο ή το σύστημα. Ο κρυπταναλυτής έχει την ίδια πληροφορία για το αρχικό κείμενο ή το σύστημα είτε γνωρίζοντας το κρυπτοκείμενο είτε όχι. Μάλιστα, αυτό ισχύει ακόμη και αν γνωρίζει ένα μέρος του αρχικού μηνύματος.

Το μπλοκάκι μιας χρήσης είναι ένα κρυπτοσύστημα μυστικού κλειδιού (κλασικό) όπου το κλειδί έχει το ίδιο μήκος με το κείμενο προς κρυπτογράφηση. Επιπλέον, το κλειδί το χρησιμοποιούμε μόνο μια φορά και μετά το "πετούμε", δεν το ξαναχρησιμοποιούμε.

Το αρχικό κείμενο $M$ αναπαριστάται ως δυαδική ακολουθία, όπως επίσης και το κλειδί $K$. Το κρυπτοκείμενο $C$ προκύπτει από τη αποκλειστική διάζευξη ανά bit (XOR) (ή πρόσθεση modulo 2) του αρχικού κειμένου με το κλειδί:

$C = M \oplus K$. Η αποκρυπτογράφηση δίνεται από το $M = C \oplus K$.

Αποδεικνύεται ότι είναι αδύνατο για τον κρυπταναλυτή να σπάσει τον κώδικα που χρησιμοποιεί μπλοκάκι μιας χρήσης: Οποιοδήποτε κρυπτοκείμενο $C$ δεν αποκαλύπτει καμία πληροφορία για το αρχικό κείμενο $M$ αφού κάθε μήνυμα $M$ θα μπορούσε να παραγάγει το $C$, αν το κλειδί $K$ ήταν ίσο με $K = C \oplus M$.

Η κρυπτογράφηση με μπλοκάκι μιας χρήσης είναι αποδεδειγμένα ασφαλής με πληροφοριοθεωρητική έννοια, αφού ο υποκλοπέας δεν έχει ποτέ αρκετή πληροφορία για να αποκρυπτογραφήσει το κρυπτοκείμενο και καμμία ποσότητα υπολογιστικής δύναμης δεν μπορεί να τον βοηθήσει.

Από την άλλη πλευρά, το μπλοκάκι μιας χρήσης δεν είναι πρακτικό αφού ένα μεγάλο κλειδί πρέπει να δημιουργηθεί, να διανεμηθεί και να αποθηκευτεί.

Τα σύγχρονα κρυπτοσυστήματα χρησιμοποιούν σχετικά μικρά κλειδιά (56 - 1000 bits) και (επιδιώκουν να) είναι ασφαλή με την υπολογιστική έννοια, εννοώντας ότι η κρυπτανάλυσή τους είναι υπολογιστικά απρόσιτη (ανέφικτη).

## 16.4 Κρυπτοσυστήματα Πακέτου (Block Ciphers)

### 16.4.1 Τα κρυπτοσυστήματα DES (Data Encryption Standard) και AES (Adavanced Encrytpion Standard)

Το 1972 το αμερικάνικο NBS (National Bureau of Standards) , νυν NIST (National Institute of Standards and Technology), ξεκίνησε ένα πρόγραμμα για την προστασία δεδομένων, στου οποίου τα πλαίσια αναζήτησε έναν αλγόριθμο κρυπτογράφησης που θα λειτουργούσε ως πρότυπο. Το 1974, η IBM πρότεινε ένα κρυπτοσύστημα βασισμένο στον αλγόριθμο «Lusifer», το οποίο σταδιακά εξελίχθηκε στο DES (Data Encryption Standard).

Το DES τέθηκε σε ισχύ το 1977 από το αμερικάνικο Υπουργείο Εμπορίου ως πρότυπο για την προστασία *ευαίσθητων αλλά όχι απόρρητων* δεδομένων. Μετά το NBS, πολλοί άλλοι οργανισμοί υιοθέτησαν το DES. Ανάμεσά τους το ANSI (American National Standards Institute) και η American Bankers Association.

Το DES είναι ένα κρυπτοσύστημα πακέτου (block chipher) διαστάσεων $64 \times 56$, του οποίου κύριο δομικό στοιχείο είναι ένα Feistel δίκτυο διαστάσεων $64 \times 48 \times 16$. Παρά το μεγάλο χρονικό διάστημα στο οποίο παρέμεινε το ευρύτερα διαδεδομένο κρυπτοσύστημα παγκοσμίως, το DES αποδείχθηκε ιδιαίτερα ανθεκτικό στις κρυπταναλυτικές επιθέσεις. Οι πρώτες επιθέσεις που είχαν ουσιαστικά καλύτερο αποτέλεσμα από την *εξαντλητική αναζήτηση κλειδιού* αναπτύχθηκαν μέσα στη δεκαετία του '90.

Το DES δεν είναι πληροφοριοθεωρητικά ασφαλές αλλά η κρυπτανάλυσή του είναι υπολογιστικά απρόσιτη.

Κρυπτογράφηση: Χωρίζουμε το αρχικό κείμενο σε κομμάτια των 64 bits, και σε κάθε κομμάτι εφαρμόζουμε μια συγκεκριμένη μετάθεση των bits του αρχικού κειμένου (αυτή η μετάθεση δεν έχει προφανή κρυπτογραφική σημασία). Το κείμενο που προκύπτει χωρίζεται σε αριστερά και δεξιά κομμάτια των 32 bits $L$ και $R$ αντίστοιχα, τα οποία κρυπτογραφούνται με τις εξής πράξεις ($i = 1, \ldots, 16$)

$$
\begin{aligned}
L_i &= R_{i-1} \\
R_i &= L_{i-1} \oplus f(R_{i-1}, K_{i-1})
\end{aligned}
$$

Η συνάρτηση $f$ ορίζεται με τη χρήση συναρτήσεων αντικατάστασης (ή *κουτιών* $S$), οι οποίες απεικονίζουν εισόδους των 6 bits σε εξόδους των 4 bits. Τα 16 κλειδιά $K_i$ προκύπτουν από το αρχικό κλειδί $K$ με κατάλληλες μεταθέσεις και επιλογή των ψηφίων του.

Το DES είναι αντιστρέψιμο ανεξάρτητα από τον ορισμό της $f$.

Ο αλγόριθμος του DES είναι πολύ γρήγορος, ιδίως με εξειδικευμένο hardware. Από την άλλη πλευρά, η κρυπτανάλυση μας οδηγεί σε διάφορα μη-γραμμικά συστήματα εξισώσεων και τα προβλήματα που προκύπτουν είναι το λιγότερο NP-πλήρη.

Το 1993 οι Biham και Shamir ανέπτυξαν μια επίθεση με επιλεγμένα απλά κείμενα γνωστή ως *διαφορική κρυπτανάλυση* ενώ το 1994 ο Matsui εισήγαγε την *γραμμική κρυπτανάλυση*, μια επίθεση με γνωστά απλά κείμενα. Τόσο η διαφορική όσο και η γραμμική κρυπτανάλυση αποτελούν

ιδιαίτερα σημαντικές κρυπταναλυτικές τεχνικές, η πρακτική τους ωστόσο επίδραση στην ασφάλεια του DES είναι μικρή αφού απαιτούν έναν τεράστιο αριθμό επιλεγμένων ή γνωστών απλών κειμένων αντίστοιχα. Το καθοριστικό ρήγμα στην ασφάλεια του DES προέρχεται από το μικρό μήκος του κλειδιού που χρησιμοποιείται. Για τα σημερινά τεχνολογικά δεδομένα το μήκος κλειδιού 56 bit παρέχει πράγματι μικρή ασφάλεια απέναντι στην τετριμμένη επίθεση της εξαντλητικής αναζήτησης. Ενδεικτικά αναφέρουμε ότι το 1998 η ειδικά κατασκευασμένη μηχανή Deep Crack της Electronic Frontier Foundation έσπασε το DES σε 56 ώρες έχοντας ψάξει το 25% του χώρου των κλειδιών.

Ως προσωρινή λύση για ασφαλέστερη κρυπτογράφηση είχε ήδη προταθεί από το 1993 το Triple-DES, το οποίο αν και είναι πράγματι ασφαλέστερο του DES, είναι και τρεις φορές πιο αργό.

Τον Ιανουάριο του 1997 το NIST ανακοίνωσε την πρόθεση για την ανάπτυξη ενός νέου κρυπτογραφικού προτύπου, του AES (Advanced Encryption Standard), που θα αντικαθιστούσε το DES. Τον Σεπτέμβριο του 1997 ανακοινώθηκε η έναρξη ανοιχτού διαγωνισμού. Το NIST δεν θα προχωρούσε σε αξιολόγηση της ασφάλειας και της αποδοτικότητας των υποψηφίων κρυπτοσυστημάτων, αλλά κάλεσε την κρυπτολογική κοινότητα να πραγματοποιήσει επιθέσεις. Τα σχετικά αποτελέσματα θα ανακοινώνονταν σε μια σειρά συνεδρίων.

Η επιλογή έγινε τόσο με βάση την ασφάλεια — αντοχή στις επιθέσεις που πραγματοποιήθηκαν, αλλά και θεωρητική άμυνα απέναντι σε μεθόδους γραμμικής και διαφορικής κρυπτανάλυσης — όσο και με βάση την ταχύτητα, την απλότητα και την ευελιξία του Rijndael. Το AES τέθηκε σε ισχύ στις 26 Μαΐου 2002 από το αμερικάνικο Υπουργείο Εμπορίου.

## 16.5 Συστήματα δημοσίου κλειδιού

Η ιδέα της κρυπτογραφίας δημοσίου κλειδιού πρωτοπαρουσιάστηκε από τους Diffie και Hellman σε μια εργασία που δημοσιεύτηκε το 1976 ([2]). Η κρυπτογραφία δημόσιου κλειδιού παρουσιάστηκε σχετικά αργά (αν σκεφτούμε ότι είναι μια σχετικά απλή ιδέα και η ιστορία της κρυπτογραφίας πολύ μεγάλη), βασικά διότι οι περισσότερες θεμελιώδεις έννοιες συνδέονται ιδιαίτερα με τη θεωρία πολυπλοκότητας, η οποία αναπτύχθηκε σχετικά πρόσφατα.

*Παράδειγμα* 16.4. Αυτό το παράδειγμα δίνει μια πολύ γενική ιδέα της κρυπτογραφίας δημοσίου κλειδιού. Σκεφθείτε τον τηλεφωνικό κατάλογο. Είναι εύκολο να βρείτε το νούμερο του τηλεφώνου ενός ατόμου, αλλά μάλλον δύσκολο να βρείτε το πρόσωπο που έχει ένα συγκεκριμένο τηλεφωνικό νούμερο. Αυτό δίνει την ιδέα της συνάρτησης μονής κατεύθυνσης: Είναι εύκολο να υπολογίσει κάποιος τη συνάρτηση αλλά ο αντίστροφος υπολογισμός είναι πολύ δύσκολος (απρόσιτος). Σε αυτό το παράδειγμα ο τηλεφωνικός κατάλογος είναι το κλειδί κρυπτογράφησης που μπορεί να δημοσιευτεί με ασφάλεια (δημόσιο κλειδί). Το αρχικό κείμενο μπορεί να κρυπτογραφηθεί ως εξής: Αντικαθιστούμε κάθε γράμμα του αρχικού κειμένου με το νούμερο τηλεφώνου ενός ατόμου που το όνομά του αρχίζει με αυτό το γράμμα (επιλεγμένου τυχαία από τον κατάλογο). Το σύστημα είναι πολυαλφαβητικό αφού είναι εξαιρετικά απίθανο δύο εμφανίσεις του ίδιου γράμματος να κρυπτογραφηθούν με τον ίδιο τρόπο. Σημειώστε επίσης ότι η κρυπτογράφηση είναι μη ντετερμινιστική. Προκειμένου να αποκρυπτογραφήσουμε το κρυπτοκείμενο χρειάζεται να επεξεργαστούμε έναν κατάλογο ονομάτων και τους αντίστοιχους αριθμούς τηλεφώνων, ταξινομημένα με βάση τους αριθμούς τηλεφώνων (inverse catalog-index). Αυτός ο

αντίστροφος κατάλογος είναι το κλειδί αποκρυπτογράφησης (ιδιωτικό κλειδί). □

### 16.5.1 Γενικά

Η κρυπτογραφία δημοσίου κλειδιού βασίζεται στην ιδέα των *συναρτήσεων μονής-κατεύθυνσης*. Μία συνάρτηση $f(x)$ ονομάζεται μονής-κατεύθυνσης αν είναι εύκολο να υπολογίσουμε το $f(x)$ από το $x$ (εύκολο = σε χαμηλό πολυωνυμικό χρόνο, προτιμότερα γραμμικό), ενώ ο αντίστροφος υπολογισμός δηλαδή του $x$ από το $f(x)$ είναι απρόσιτος. Ένα πρόβλημα λέγεται συνήθως *απρόσιτο* αν δεν υπάρχει γνωστός πολυωνυμικός αλγόριθμος που να το λύνει, και *προσιτό* εάν υπάρχει τέτοιος αλγόριθμος (τότε λέμε ότι το πρόβλημα ανήκει στην κλάση $P$). Σημειώστε ότι απρόσιτα συνήθως θεωρούνται τα προβλήματα που είναι τουλάχιστον τόσο δύσκολα όσο και τα δυσκολότερα προβλήματα της κλάσης $NP$. Η χρήση των συναρτήσεων μονής-κατεύθυνσης προσφέρει στα κρυπτοσυστήματα δημοσίου κλειδιού την ιδιότητα: όταν γνωρίζουμε μόνο το κλειδί κρυπτογράφησης δεν μπορούμε να το χρησιμοποιήσουμε για να βρούμε το κλειδί αποκρυπτογράφησης χωρίς να χρειαστεί ένας πολύ μεγάλος υπολογισμός.

Ο ορισμός των συναρτήσεων μονής-κατεύθυνσης που δόθηκε παραπάνω, δεν είναι ακριβής από μαθηματική σκοπιά. Η έννοια του "απρόσιτου" (πρακτικά μη υπολογίσιμου) είναι στενά συνδεδεμένη με την τρέχουσα διαθέσιμη υπολογιστική δύναμη: είναι ουσιαστικά μια εμπειρική έννοια που εξαρτάται από την εξέλιξη της τεχνολογίας των υπολογιστών (π.χ. τεχνικές μαζικά παράλληλης επεξεργασίας). Έτσι, μια συνάρτηση που σήμερα θεωρείται ως ασφαλής συνάρτηση μονής-κατεύθυνσης μπορεί να χάσει αυτή της την ιδιότητα σε μερικές δεκαετίες.

Προκειμένου να *αποδείξουμε* ότι ένα κρυπτοσύστημα είναι σύστημα δημοσίου κλειδιού (δηλαδή ότι μια συνάρτηση είναι μονής-κατεύθυνσης) θα πρέπει να αποδείξουμε μαθηματικά ότι υπάρχει ένα μη τετριμένο κάτω όριο στο υπολογιστικό έργο που απαιτείται για την υλοποίηση ενός αλγορίθμου αποκρυπτογράφησης με άγνωστο το κλειδί αποκρυπτογράφησης. Ακόμη και για τα καλύτερα γνωστά συστήματα δημοσίου κλειδιού και τις συναρτήσεις που χρησιμοποιούν, δεν έχει αποδειχτεί ότι υπάρχει κάποιο τέτοιο κάτω όριο. Έτσι, τα περισσότερα κρυπτοσυστήματα δημοσίου κλειδιού που χρησιμοποιούνται σήμερα δεν είναι *αποδεδειγμένα* δημοσίου κλειδιού (απρόσιτα): είναι πάντα πιθανό ένας ικανός κρυπταναλυτής να τα σπάσει.

Στην κρυπτογραφία δημοσίου κλειδιού υπάρχουν δύο κλειδιά: το κλειδί κρυπτογράφησης, το οποίο μπορεί με ασφάλεια να δημοσιοποιηθεί (χωρίς να βάζει σε κίνδυνο τη μυστικότητα του κλειδιού αποκρυπτογράφησης), και το κλειδί αποκρυπτογράφησης που πρέπει να παραμείνει μυστικό, δηλαδή να είναι γνωστό μόνο σε έναν χρήστη του κρυπτοσυστήματος. Τα κλειδιά κρυπτογράφησης μπορούν να δημοσιοποιηθούν σε έναν "τηλεφωνικό κατάλογο" που περιέχει όλους τους χρήστες και τα αντίστοιχα κλειδιά κωδικοποίησής τους (στην θέση των τηλεφωνικών αριθμών). Αν ο χρήστης Α θέλει να επικοινωνήσει ιδιατέρως με τον χρήστη Β, θα πρέπει να βρει το κλειδί κρυπτογράφησης (δημόσιο κλειδί) του Β, και να κρυπτογραφήσει το μήνυμά του με αυτό το κλειδί. Τότε το κρυπτογραφημένο μήνυμα δεν μπορεί να διαβαστεί από οποιονδήποτε άλλον (ούτε από τον ίδιο τον Α), εκτός από τον χρήστη Β που κατέχει το αντίστοιχο κλειδί αποκρυπτογράφησης.

*Παράδειγμα* 16.5. Ένα κρυπτοσύστημα δημοσίου κλειδιού βασισμένο στο πρόβλημα ΣΑΚΙ-

ΔΙΟΥ (KNAPSACK problem).[3] Αυτό το πρόβλημα είναι γνωστό και ως κρυπτοσύστημα Merkle-Hellman, και το έχει σπάσει ο Shamir.

Το πρόβλημα ΣΑΚΙΔΙΟΥ είναι **NP**-πλήρες, οπότε θεωρείται αρκετά απρόσιτο υπολογιστικά.

Στην πραγματικότητα, γι'αυτό το κρυπτοσύστημα δεν χρησιμοποιούμε το πρόβλημα του σακιδίου, αλλά το πρόβλημα του *Αθροίσματος Υποσυνόλων (Subset Sum)*.

Το πρόβλημα του σακιδίου (για τους σκοπούς αυτού του παραδείγματος) αποτελείται από μια $n$-άδα $A = (a_1, \ldots, a_n)$ από διαφορετικούς θετικούς ακεραίους, που ονομάζεται διάνυσμα του σακιδίου και ένα θετικό ακέραιο $k$, την χωρητικότητα του σακιδίου. Το πρόβλημα είναι να βρούμε τέτοιους ακεραίους $a_i$ των οποίων το άθροισμα να είναι ίσο με $k$. Προφανώς πάντα μπορεί να βρεθεί μια λύση ελέγχοντας εξαντλητικά όλα τα $2^n$ υποσύνολα του $A$, ψάχνοντας μήπως κάποιο από αυτά έχει άθροισμα το $k$. Για μεγάλο $n$ αυτός ο υπολογισμός είναι απρόσιτος. Ορίζουμε μια συνάρτηση $f(x)$ ως εξής: Έστω $\vec{x}$ η δυαδική αναπαράσταση (με $n$ bits) του ακεραίου $x$, $0 \leq x \leq 2^{n-1} - 1$ (προσθέτοντας μηδενικά στην αρχή, αν είναι απαραίτητο). Το $f(x)$ είναι ίσο με το άθροισμα όλων των $a_i$ τέτοιων ώστε το $i$-οστό bit του $x$ να είναι άσσος. Για παράδειγμα:

$$\begin{aligned} f(1) &= f(0\ldots001) = a_n \\ f(3) &= f(0\ldots011) = a_{n-1} + a_n \end{aligned}$$

Χρησιμοποιώντας πολλαπλασιασμό διανυσμάτων μπορούμε να συμβολίσουμε: $f(x) = A \cdot B_x$ όπου $A$ είναι το διάνυσμα σακιδίου και $B_x$ είναι η δυαδική αναπαράσταση του $x$ γραμμένη ως διάνυσμα-στήλη.

Με τον παραπάνω ορισμό είναι εύκολο να υπολογίσουμε το $f(x)$ από το $x$, ενώ το να υπολογίσουμε το $x$ από το $f(x)$ είναι ισοδύναμο με το να λύσουμε το πρόβλημα του σακιδίου, αφού το $x$ αναπαριστά (στη δυαδική του μορφή) τα αντικείμενα του A που έχουν άθροισμα $f(x)$.

Η συνάρτηση $f(x)$ μπορεί να χρησιμοποιηθεί για κρυπτογράφηση με σχετικά απλό τρόπο: Το αρχικό κείμενο γράφεται με τη μορφή δυαδικής ακολουθίας αντικαθιστώντας κάθε γράμμα του αλφαβήτου με τον αντίστοιχο αριθμό (π.χ. ASCII) (γραμμένο σε δυαδική μορφή). Κάθε ακουλουθία των n bits κρυπτογραφείται υπολογίζοντας τη συνάρτηση $f$ για το συγκεκριμένο κομμάτι (block). Η αποκρυπτογράφηση, από την άλλη πλευρά, είναι εξίσου δύσκολη με ένα NP-πλήρες πρόβλημα, όχι μόνο για τον κρυπταναλυτή, αλλά και για τον ίδιο το χρήστη. Αυτό μπορεί να αποφευχθεί σχεδιάζοντας το σύστημα έτσι ώστε ο νόμιμος χρήστης (που γνωρίζει μια μυστική καταπακτή) να χρειάζεται να λύσει ένα εύκολο στιγμιότυπο του προβλήματος του σακιδίου, ενώ ο κρυπταναλυτής να βρίσκεται απέναντι στο γενικό (δύσκολο) πρόβλημα.

Υπάρχουν κλάσεις από εύκολα προβλήματα σακιδίου. Μια τέτοια κλάση αποτελείται από τα προβλήματα σακιδίου που χρησιμοποιούν υπεραυξητικά διανύσματα σακιδίου $A$. Ένα διάνυ-

[3] Το πρόβλημα KNAPSACK ορίζεται ως εξής: Θεωρήστε έναν ακέραιο (την *χωρητικότητα*) $M > 0$ και ένα σύνολο $S$ αντικειμένων $s_i$, που στο καθένα αντιστοιχούν δύο αριθμοί: η *τιμή* $v_i$, και το *βάρος* $w_i$. Έτσι, $S = \{s_1, \ldots, s_n\}$ όπου $s_i = (v_i, w_i)$. Το πρόβλημα είναι να βρεθεί ένα υποσύνολο $S' \subset \{1, 2 \ldots, n\}$ τέτοιο ώστε τα μέλη του να ικανοποιούν τη συνθήκη $\sum_{i \in S'} w_i \leq M$ και $\sum_{i \in S'} v_i$ να είναι όσο το δυνατόν μεγαλύτερο. Αυτό σημαίνει ότι, πρέπει να επιλέξουμε ορισμένα αντικείμενα από το $S$ (χωρίς επαναλήψεις), τέτοια ώστε να μεγιστοποιήσουμε τη συνολική τους τιμή, ενώ τα βάρη τους να μην ξεπερνούν τη χωρητικότητα $M$.

σμα σακιδίου (ή μια $n$-άδα γενικά) $A = (a_1, \ldots, a_n)$ λέγεται *υπεραυξητική* αν κάθε αριθμός $a_i$ υπερβαίνει το άθροισμα όλων των προηγούμενων αριθμών:

$$a_j > \sum_{i=1}^{j-1} a_i, \ for\ j = 2, \ldots, n$$

Σε αυτή την περίπτωση το πρόβλημα του σακιδίου μπορεί να λυθεί σε *γραμμικό χρόνο* αφού ένα πέρασμα του διανύσματος του σακιδίου είναι αρκετό. Είναι επίσης προφανές (από τον αλγόριθμο που λύνει το πρόβλημα) ότι το υπεραυξητικό πρόβλημα σακιδίου έχει πάντα *το πολύ μία λύση*.

Εάν χρησιμοποιούμε υπεραυξητική ακολουθία $A$ για ένα κρυπτοσύστημα, δεν πρέπει να τη δημοσιεύσουμε, γιατί θα έκανε την κρυπτανάλυση γραμμική (άρα εύκολη). Αυτό μπορούμε να το αποφύγουμε ανακατεύοντας το $A$ σε ένα διάνυσμα $A'$ που θα μοιάζει με ένα τυχαίο διάνυσμα σακιδίου. Το ανακάτεμα αυτό μπορεί να γίνει με πολλαπλασιασμό με modulo:

Διαλέγουμε έναν ακέραιο (modulus) $m > \sum a_i$ και έναν πολλαπλασιαστή $t$ έτσι ώστε ο $t$ και ο $m$ να μην έχουν κοινούς παράγοντες. Η εκλογή του $t$ εξασφαλίζει την ύπαρξη του $t^{-1}$ (αντίστροφος του $t$) έτσι ώστε $tt^{-1} = 1 \pmod m$. Υπολογίζουμε τα γινόμενα $a'_i = ta_i$ και ορίζουμε το διάνυσμα $A = (a'_1, \ldots, a'_n)$. Το διάνυσμα $A' = tA$ μπορεί να δημοσιευτεί ως κλειδί κρυπτογράφησης, αλλά οι όροι $t, t^{-1}, m$ πρέπει να κρατηθούν μυστικοί (secret trapdoor).

Ο νόμιμος αποδέκτης, για να αποκρυπτογραφήσει ένα block κρυπτοκειμένου $c'$ (block με $n$ bits) πρέπει πρώτα να υπολογίσει το $c = t^{-1}c' \bmod m$, και μετά να λύσει το εύκολο πρόβλημα σακιδίου βασισμένο στο $A = t^{-1}A' \bmod m$. Η μοναδική λύση είναι το σωστό block αρχικού κειμένου $p$ αφού:

$$c = t^{-1}c' = t^{-1}A'p = t^{-1}tAp = Ap \pmod m$$

που σημαίνει ότι παρόλο που η κρυπτογράφηση έγινε χρησιμοποιώντας το $A'$, η αποκρυπτογράφηση μπορεί να βασιστεί στο $A$ και τους μυστικούς όρους. Αυτό το κρυπτοσύστημα δημοσίου κλειδιού που βασίζεται στο πρόβλημα του σακιδίου έσπασε από τον Shamir [12] με αλγόριθμο πολυωνυμικού χρόνου. Η κρυπταναλυτική επίθεση βασίζεται στο γεγονός ότι δεν είναι απαραίτητο για τον κρυπταναλυτή να βρει τον ίδιο πολλαπλασιαστή $t$ και modulus $m$ με αυτα που χρησιμοποιεί ο σχεδιαστής του συστήματος. Αρκεί να βρει $t'$ και $m'$ τέτοια ώστε ο πολλαπλασιασμός του δημοσιευμένου διανύσματος με το $(t')^{-1} \pmod{m'}$ να είναι ένα υπεραυξητικό διάνυσμα. Έτσι, ο κρυπταναλυτής μπορεί να σπάσει το σύστημα με προεπεξεργασία, αφού το κλειδί δημοσιευτεί. □

### 16.5.2 Σχεδιάζοντας ένα κρυπτοσύστημα δημοσίου κλειδιού

Ένα κρυπτοσύστημα δημοσίου κλειδιού μπορούμε να το κατασκευάσουμε με τα ακόλουθα γενικά βήματα:

- Διαλέγουμε ένα δύσκολο πρόβλημα Π. Το Π θα πρέπει να είναι υπολογιστικά απρόσιτο. Το Π λέγεται υποκείμενο πρόβλημα.
- Βρίσκουμε ένα εύκολο υποπρόβλημα $\Pi_{\text{easy}}$ του Π. Το $\Pi_{\text{easy}}$ θα πρέπει να χρειάζεται μικρό πολυωνυμικό χρόνο.

- "Ανακατεύουμε" το $\Pi_{\text{easy}}$ με τέτοιο τρόπο ώστε το προκύπτον πρόβλημα $\Pi_{\text{shuffle}}$ να μοιάζει με το γενικό πρόβλημα $\Pi$.
- Δημοσιεύουμε το $\Pi_{\text{shuffle}}$ και τη μέθοδο κρυπτογράφησης. Η μέθοδος ανάκτησης του $\Pi_{\text{easy}}$ από το $\Pi_{\text{shuffle}}$ είναι η μυστική καταπακτή. Έτσι ο νόμιμος αποδέκτης θα πρέπει να λύσει το $\Pi_{\text{easy}}$ ενώ ο κρυπταναλυτής βρίσκεται ενάντια στο γενικό πρόβλημα $\Pi$.

Το κρυπτοσύστημα στο παράδειγμα 16.5 είναι μια τυπική επίδειξη των παραπάνω στοιχείων για το σχεδιασμό κρυπτοσυστημάτων δημοσίου κλειδιού: $\Pi$ είναι το πρόβλημα σακιδίου (NP-πλήρες), $\Pi_{\text{easy}}$ είναι το πρόβλημα σακιδίου με υπεραυξητικό διάνυσμα σακιδίου και το $\Pi_{\text{shuffle}}$ προκύπτει από πολλαπλασισμό modulo όπως περιγράψαμε παραπάνω.

Η κρυπτανάλυση του κρυπτοσυστήματος δημοσίου κλειδιού (υπολογισμός του $f^{-1}(x)$ από το $x$ χωρίς να ξέρουμε τη μυστική καταπακτή) περιλαμβάνει τη λύση του γενικού απρόσιτου προβλήματος (του υποκείμενου προβλήματος) που έχει επιλεχτεί προσεκτικά από το σχεδιαστή του συστήματος (πρόβλημα $\Pi$, σύμφωνα με την παραπάνω συζήτηση). Στο παράδειγμα 16.5, παρόλο που χρησιμοποιήθηκε ένα πρόβλημα αποδεδειγμένα απρόσιτο (πρόβλημα σακιδίου, **NP**-πλήρες), το κρυπτοσύστημα που προέκυψε, αποδείχτηκε ότι είναι μάλλον ασθενές. Από την άλλη πλευρά, το πιο μελετημένο κρυπτοσύστημα δημοσίου κλειδιού που υπάρχει (ως τώρα), το RSA, έχει αντέξει σε όλες τις κρυπταναλυτικές επιθέσεις, παρόλο που η πολυπλοκότητα του υποκείμενου προβλήματος (παραγοντοποίηση) δεν έχει κατηγοριοποιηθεί (η παραγοντοποίηση *δεν έχει αποδειχθεί ακόμη* αν είναι όντως απρόσιτο πρόβλημα, αν και πιστεύεται ευρύτατα ότι είναι).

Τα περισσότερα συστήματα δημοσίου κλειδιού βασίζονται σε αριθμοθεωρητικά υποκείμενα προβλήματα. Ο παρακάτω κατάλογος απαριθμεί μερικά θεμελιώδη αριθμοθεωρητικά προβλήματα που έχουν αντισταθεί ως τώρα σε όλες τις προσπάθειες να ταξινομηθεί η πολυπλοκότητά τους και έχουν αποδειχτεί πολύ χρήσιμα στην κρυπτογραφία δημοσίου κλειδιού. Κανένα από αυτά τα προβλήματα δεν είναι γνωστό να είναι πλήρες για κάποια κλάση πολυπλοκότητας ή να έχει κάποιο ντετερμινιστικό ή πιθανοτικό πολυωνυμικό αλγόριθμο.

| | |
|---|---|
| FACTOR($n$) | Βρές τους πρώτους παράγοντες του $n$ |
| SQUAREFREENESS($n$) | Αποφάσισε αν το τετράγωνο ενός πρώτου διαιρεί το $n$ (Αποφάσισε αν το $n$ είναι γινόμενο διακριτών πρώτων) |
| QUAD-RESIDUE($a, n$) | Αποφάσισε αν $x^2 = a \pmod n$ ισχύει για κάποιο $x$ |
| SQUAREROOT($a, n$) | Βρές ένα $x$ τέτοιο ώστε $x^2 = a \pmod n$ |
| DISCRETE-LOG($a, b, n$) | Βρές ένα $x$ τέτοιο ώστε $a^x = b \pmod n$ |

### 16.5.3 Κρυπτανάλυση και ασφάλεια

Τυπικά, θα ήταν επιθυμητό να βρούμε ένα κάτω όριο στην ποσότητα των υπολογισμών που είναι απαραίτητοι στον κρυπταναλυτή για να σπάσει το κρυπτοσύστημα δημοσίου κλειδιού (να λύσει το υποκείμενο πρόβλημα). Δυστυχώς, τέτοια θεωρητικά κάτω όρια δεν έχουν αποδειχτεί για κανένα από τα πιο γνωστά συστήματα δημοσίου κλειδιού. Για παράδειγμα, αν το FACTOR($n$)

λύνεται σε χαμηλό πολυωνυμικό χρόνο (πράγμα μάλλον απίθανο), τότε το RSA και συναφή συστήματα θα κατέρρεαν.

Ακόμη, το πρόβλημα της κρυπτανάλυσης (με δεδομένο το κρυπτοκείμενο και το κλειδί κρυπτογράφησης) για ένα σύστημα δημοσίου κλειδιού, είναι στο $NP \cap co - NP$ και άρα είναι μάλλον απίθανο ότι κάποιο πρόβλημα κρυπτανάλυσης θα είναι NP-πλήρες (γιατί θα σήμαινε ότι NP=co-NP). Από την άλλη πλευρά, για το σενάριο "κλειδί κρυπτογράφησης μόνο", το πρόβλημα κρυπτανάλυσης είναι συνήθως NP-πλήρες.

Συνήθως για να παρουσιάσουμε μια απόδειξη της ασφάλειας ενός συγκεκριμένου συστήματος δημοσίου κλειδιού, πρέπει να κατασκευάσουμε μια αναγωγή, όπου δείχνουμε πως να λύσουμε ένα δύσκολο πρόβλημα (π.χ. παραγοντοποίηση) εάν έχουμε τη δυνατότητα να σπάσουμε το κρυπτογραφικό σύστημα. Στο σχεδιασμό συστήματος δημοσίου κλειδιού, πολλές παράμετροι (όπως το μέγεθος του κλειδιού) πρέπει να καθοριστούν με βάση την υπολογιστική δύναμη που υποτίθεται ότι έχει ο επίδοξος εχθρός που προσπαθεί να σπάσει το σύστημα. Είναι πάντα συνετό να επιτυγχάνουμε η επίθεση με πολυωνυμικό χρόνο να είναι όχι μόνο πιθανοτική, αλλά και μη-ομοιόμορφη. Περισσότερο τυπική μελέτη αυτών των θεμάτων γίνεται στο κεφάλαιο **??**.

## 16.6 Στόχοι της Κρυπτολογίας

Καθώς οι υπολογιστές και τα δίκτυα υπολογιστών συνεχίζουν να αναπτύσσονται και να επεκτείνονται, οι εφαρμογές της κρυπτολογίας μεγαλώνουν όπως και η ανάγκη για κρυπτογραφικά συστήματα.

Η κρυπτολογία παρέχει μεθόδους ώστε να ικανοποιούνται οι βασικές απαιτήσεις ασφαλούς επικοινωνίας. Μερικές από αυτές, και κάποιες πιο εξειδικευμένες, αναφέρονται παρακάτω (βλ. και [**?**]):

- *Μυστικότητα (Privacy)*. Ο πιθανός "κατάσκοπος" να μην μαθαίνει τίποτα χρήσιμο για το μήνυμα που στάλθηκε.
- *Πιστοποίηση (Authentication)*. Ο αποδέκτης του μηνύματος να μπορεί να πειστεί ότι το μήνυμα εστάλη πράγματι από τον υποτιθέμενο αποστολέα.
- Εμπιστευτικότητα (confidentiality): κανείς μη εξουσιοδοτημένος χρήστης δεν πρέπει να έχει πρόσβαση στο μεταδιδόμενο μήνυμα.
- Έλεγχος ακεραιότητας των δεδομένων (data integrity): αλλοίωση των δεδομένων κατά τη μετάδοση πρέπει να γίνεται αντιληπτή στον παραλήπτη.
- *Υπογραφές (Signatures)*. Ο αποδέκτης ενός μηνύματος να μπορεί να πείσει κάποιον τρίτο ότι το μήνυμα πράγματι εστάλη από τον υποτιθέμενο αποστολέα.
- *Μη Αποκήρυξη (Non-Repudiation)*: κανείς δεν μπορεί να αποποιηθεί την υπογραφή του.
- *Ελαχιστότητα (Minimality)*. Τίποτα δεν κοινοποιείται στους άλλους, εκτός από αυτό που ρητά επιθυμούμε να κοινοποιήσουμε.

- *Ταυτόχρονη Αμοιβαία Ανταλλαγή (Simultaneous Exchange)*. Κάτι πολύτιμο (π.χ. υπογραφή σε συμβόλαιο), δεν πρέπει να αποστέλλεται εωσότου κάτι άλλο πολύτιμο (π.χ. η υπογραφή του άλλου) να παραληφθεί.

- *Συντονισμός (Coordination)*. Σε επικοινωνία μεταξύ πολλών πλευρών, οι πλευρές μπορούν να συντονίσουν τις ενέργειές τους για την επίτευξη κοινών στόχων ακόμη και με την παρουσία αντιπάλων (κατασκόπων).

- *Κατώφλι Συνεργασίας (Collaboration Threshold)*. Σε ένα σενάριο με πολλές πλευρές, οι επιθυμητές ιδιότητες ισχύουν μόνο για όσο ο αριθμός των αντιπάλων δεν υπερβαίνει το δεδομένο κατώφλι (όριο).

- *Μερική Αποκάλυψη Απορρήτων (Partial Disclosure of Secrets)*. Δύο η περισσότερες πλευρές που κατέχουν τα απόρρητα $x1, x2, \ldots$, επιθυμούν να διανείμουν την τιμή της $f(x1, x2, \ldots)$ χωρίς να αποκαλύψουν τα $x1, x2, \ldots$ σε κανέναν άλλο.

- *Μη Συνειδητή Μεταφορά (Oblivious Transfer)*. Η πλευρά Α που κατέχει ένα απόρρητο θέλει να το μεταφέρει σε μια άλλη πλευρά Β, έτσι ώστε η Α να μη γνωρίζει αν η Β παρέλαβε το μυστικό (αλλά η Β το γνωρίζει). Ή, η Α κατέχει πολλά μυστικά και θέλει να μεταφέρει μόνο ένα στη Β έτσι ώστε μόνο η Β να γνωρίζει ποιο μυστικό μεταφέρθηκε.

- *Αποδείξεις και Πρωτόκολλα Μηδενικής Γνώσης (Zero-Knowledge Proofs and Protocols)*. Κατά την εκτέλεση ενός πρωτοκόλλου, κανείς (ούτε οι νόμιμοι συμμετέχοντες) δεν μπορεί να μάθει τίποτε άλλο, εκτός από αυτό που προβλέπεται από το πρωτόκολλο.

- *Ανώνυμες Δοσοληψίες (Anonymous Transactions)*, ψηφιακά ψευδώνυμα (anonymous credentials). Έγκυρα ψηφιακά πιστοποιητικά που δεν μπορούν να συνδεθούν με τον πραγματικό κάτοχο.

- Τράπεζες, ψηφοφορίες, κρυπτονομίσματα (cryptocurrencies, π.χ. bitcoin) και άλλες μη θεωρητικές εφαρμογές….

Από την άλλη πλευρά, η *κρυπτανάλυση (cryptanalysis)* Κρυπτανάλυση ορίζεται ως η μελέτη των μαθηματικών τεχνικών που αποσκοπούν στην παραβίαση, σε ένα κρυπτογραφικό σύστημα, των παραπάνω χαρακτηριστικών. Το κρυπτόγραμμα, εφόσον μεταδίδεται μέσα από δημόσιο τηλεπικοινωνιακό κανάλι, είναι διαθέσιμο στον οποιονδήποτε, ακόμα και στον επίδοξο υποκλοπέα. Η εμπιστευτικότητα σε ένα κρυπτογραφικό σύστημα παραβιάζεται αν κάποιος υποκλοπέας καταφέρει τελικά, χωρίς τη γνώση του κλειδιού, να διαβάσει το αρχικό μήνυμα (ή, ισοδύναμα, να ανακαλύψει το κλειδί αποκρυπτογράφησης). Έχουν αναπτυχθεί πολλές κρυπταναλυτικές τεχνικές για όλα τα είδη κρυπτογραφικών αλγορίθμων. Η εμφάνιση κάθε κρυπταναλυτικής τεχνικής έχει εν τέλει ως αποτέλεσμα το να καθορίζονται νέες σχεδιαστικές αρχές που πρέπει να λαμβάνονται υπ' όψιν κατά την κατασκευή κρυπτογραφικών αλγορίθμων. Κατά συνέπεια, η κρυπτογραφία και η κρυπτανάλυση συμβαδίζουν ως προς την εξέλιξή τους. Ο όρος *κρυπτολογία (cryptology)* Κρυπτολογία έχει παγιωθεί, προκειμένου να συμπεριλαμβάνει ταυτόχρονα τόσο την κρυπτογραφία όσο και την κρυπτανάλυση.

## 16.7 Μοντέλα ασφάλειας

Για να συζητήσουμε για την ασφάλεια, χρειάζεται να ορίσουμε κάποιο *μοντέλο ασφάλειας*, που περιγράφει συνήθως τις δυνατότητες του αντιπάλου. Κατ'αρχήν, δεχόμαστε την παρακάτω βασική υπόθεση.

**Θεμελιώδης αρχή (Kerckhoffs)**: όλοι οι αλγόριθμοι είναι γνωστοί, *μόνο το κλειδί είναι άγνωστο* (μην υποτιμάς τον αντίπαλο!).

Με βάση το τι διαθέτει (ή μπορεί να βρει) ο αντίπαλος ορίζονται 4 βασικοί τύποι κρυπταναλυτικών επιθέσεων:

1. Κρυπτοκείμενο μόνο (**ciphertext only – CO**). Ο κρυπταναλυτής διαθέτει μόνο το κρυπτοκείμενο.
2. Γνωστό αρχικό κείμενο (**known plaintext attack – KPA**). Ο κρυπταναλυτής διαθέτει κάποια ζεύγη αρχικού κειμένου–κρυπτοκειμένου.
3. Επιλεγμένο αρχικό κείμενο (**chosen plaintext attack – CPA**). Ο κρυπταναλυτής διαθέτει κάποια ζεύγη αρχικού κειμένου–κρυπτοκειμένου, με αρχικά κείμενα της επιλογής του.
4. Επιλεγμένο κρυπτοκείμενο (**chosen ciphertext attack – CCA**). Ο κρυπταναλυτής διαθέτει κάποια ζεύγη αρχικού κειμένου–κρυπτοκειμένου για ορισμένα κρυπτοκείμενα της επιλογής του (ισοδύναμα, έχει προσωρινή δυνατότητα αποκρυπτογράφησης).

Η σύγχρονη τάση είναι να ασχολούμαστε με συστήματα που μπορούν να αντέξουν στις πιο ισχυρές επιθέσεις CPA και CCA°η τελευταία έχει ιδιαίτερο νόημα στην κρυπτογραφία δημοσίου κλειδιού, όπου ο αντίπαλος εξ ορισμού διαθέτει να παράγει ζεύγη αρχικού κειμένου – κρυπτοκειμένου κατά βούληση.

Μια πιο πρόσφατη θεώρηση, προτείνει την περαιτέρω ισχυροποίηση των δύο ισχυρότερων μοντέλων, CPA και CCA, μέσω ενός κατάλληλα διατυπωμένου παιγνίου:

**The indistiguishability game (Το παίγνιο της μη-διακρισιμότητας)**

- Δίνεται ένα κρυπτοκείμενο $c$. Ακόμη και αν ο αντίπαλος γνωρίζει ότι το αρχικό κείμενο είναι είτε το $m_0$ είτε το $m_1$, δεν θα πρέπει να είναι σε θέση να ξεχωρίσει ποιο από τα δύο είναι το σωστό με πιθανότητα σημαντικά μεγαλύτερη από $\frac{1}{2}$.
- Μπορεί να διατυπωθεί σαν παίγνιο μεταξύ ενός αντιπάλου και ενός κρυπτοσυστήματος.
- Μπορεί να εφαρμοστεί σε όλες τις επιθέσεις. Περισσότερο γνωστό για IND-CPA και IND-CCA (και IND-CCA2) στα πλαίσια της κρυπτογραφίας δημοσίου κλειδιού (όπου εξ ορισμού ο αντίπαλος έχει δυνατότητα CPA τουλάχιστον).

## 16.8 Θεωρία αριθμών και Κρυπτογραφία

Τα κρυπτοσυστήματα δημοσίου κλειδιού (καθώς και τα κλασικά κρυπτοσυστήματα) χρησιμοποιούν εκτενώς πολλά θέματα της Θεωρίας Αριθμών. Η τυποποίηση των περισσοτέρων κρυπτογραφικών συστημάτων χρησιμοποιεί πολλές αριθμοθεωρητικές έννοιες καθώς επίσης και στοιχεία Αφηρημένης Άλγεβρας (Θεωρία Ομάδων - Πεπερασμένα Σώματα).

Εκτός από τις στοιχειώδεις έννοιες και συμβολισμούς, η κρυπτογραφία χρησιμοποιεί εκτενώς τη θεωρία αριθμών που αφορά ισοτιμίες, πράξεις σε πεπερασμένα σώματα και τετραγωνικά υπόλοιπα.

Ειδικότερα, η Θεωρία Αριθμών που χρειάζεται για να προσεγγίσουμε τα πιο βασικά κρυπτογραφικά συστήματα και ιδέες, περιλαμβάνει τουλάχιστον τα εξής:

- Θεμελιώδεις ορισμούς, συμβολισμούς, διαιρετότητα, τον ευκλείδιο αλγόριθμο, το θεμελιώδες θεώρημα της αριθμητικής, το θεώρημα των πρώτων αριθμών.
- Θεωρία ομάδων, δακτυλίων, σωμάτων και στοιχειώδεις αλγεβρικές έννοιες και ισοτιμίες, το Κινέζικο Θεώρημα Υπολοίπων, το Μικρό Θεώρημα του Fermat, η συνάρτηση $\phi$ του Euler και οι ιδιότητές της.
- Γραμμικές ισοτιμίες, πρώτοι moduli, δυναμοϋπόλοιπα, παραγοντοποίηση.
- Τετραγωνικά υπόλοιπα, σύμβολο Legendre, τετραγωνική αμοιβαιότητα, σύμβολο Jacobi.

Πολλοί αλγόριθμοι για αριθμοθεωρητικά προβλήματα, έλεγχοι για το αν ένας αριθμός είναι πρώτος, γεννήτριες τυχαίων αριθμών, έλεγχοι ασφάλειας χρησιμοποιούνται επίσης στην κρυπτογραφία.

## 16.9 Primality – Factoring

Ένα πολύ σημαντικό στοιχείο για την κρυπτογράφηση, αποκρυπτογράφηση, αλλά κυρίως για την κρυπτανάλυση ενός κρυπτογραφικού συστήματος είναι ο έλεγχος ενός αριθμού για το αν είναι πρώτος (primality test) αλλά και η ανάλυση σε πρώτους παράγοντες (factoring).

Η προσπάθεια να απαντηθεί το αν ένας αριθμός είναι πρώτος είναι ένα ιδιαίτερα παλιό πρόβλημα και έχει απασχολήσει πολύ τη μαθηματική κοινότητα. Ένα από τα παλιότερα primality tests είναι το *Κόσκινο του Ερατοσθένη*. Η πολυπλοκότητα του είναι πολυωνυμική ώς προς το μέγεθος της εισόδου $k$, αλλά εκθετική ως προς το μήκος της αναπαράστασης του $k$.

Μετά από τα πρώτα, παλιά αποτελέσματα που απαντούσαν στο ερώτημα με εξαντλητικές μεθόδους, είχε μείνει για πολλά χρόνια ανοιχτό το ερώτημα αν το πρόβλημα μπορεί να λυθεί σε χρόνο πολυωνυμικό ως προς το μέγεθος της εισόδου.

Σημαντικές εξελίξεις πάνω στο πρόβλημα έγιναν την δεκαετία του ’70. Το 1976 ο Miller επινόησε ένα ντετερμινιστικό αλγόριθμο που μπορούσε να δώσει απάντηση στο πρόβλημα σε πολυωνυμικό χρόνο, αλλά ο αλγόριθμος βασιζόταν στην εκτεταμένη υπόθεση του Riemann. Δηλαδή, ο Miller έδειξε οτι το πρόβλημα βρίσκεται στην κλάση **P** αν η υπόθεση του Riemann

είναι σωστή. Το τελευταίο είναι ένα από τα πλέον γνωστά ανοιχτά προβλήματα που μένουν άλυτα εδώ και 100 περίπου χρόνια αν και οι περισσότεροι θεωρούν ότι ισχύει. Ένα χρόνο μετά τον Miller, οι Solovay και Strassen δημοσίευσαν ένα νέο πιθανοτικό αλγόριθμο που έδινε απάντηση στο πρόβλημα. Λίγο αργότερα, ο Rabin τροποποίησε τον αλγόριθμο του Miller σε έναν επίσης πιθανοτικό πολυωνυμικό αλγόριθμο. Το 1983 οι Adleman, Pomerance και Rumely παρουσίασαν για πρώτη φορά μια νέα μέθοδο που αποδεχόταν το πρόβλημα των πρώτων σε χρόνο $(\log n)^{O(\log\log\log n)}$. Αν και πρακτικά σχεδόν πολυωνυμικός χρόνος, η προσπάθειά τους δεν επέτρεπε στους θεωρητικούς να θέσουν το πρόβλημα στο **P**. Το 1986 οι Goldwasser και Killian προτείνουν ένα νέο αλγόριθμο με αναμενόμενο πολυωνυμικό χρόνο απόφασης που βασιζόταν σε ελλειπτικές καμπύλες. Οι Adleman και Huang το 1992, τροποποίησαν τον αλγόριθμο των Goldwasser και Killian, δείχνοντας ότι το πρόβλημα είναι στο **ZPP** (Zero error Probabilistic Polynomial Time).

Οριστικό τέλος δόθηκε με την εργασία των Manindra Agrawal, Neeraj Kayal και Nitin Saxena που εκδόθηκε τις πρώτες μέρες του Αυγούστου του 2002 θέτοντας το πρόβλημα στο **P** (γνωστός πλέον και ως αλγόριθμος AKS). Παρά την αρχική πολύπλοκη μορφή της απόδειξης της ορθότητας του αλγορίθμου τους, δεν πέρασε πολύ καιρός για να πάρει μια πιο απλή και αποτελεσματική μορφή μετά από παρατηρήσεις του Lenstra.

Αντίθετα με το πρόβλημα Primality, για το Factoring πιστεύεται ότι δεν υπάρχει πολυωνυμικός αλγόριθμος (ντετερμινιστικός ή πιθανοτικός). Ο καλύτερος γνωστός αλγόριθμος έχει πολυπλοκότητα τάξης $2^{\tilde{O}((\log n)^{\frac{1}{3}})}$ που είναι υπερ-πολυωνυμική, αφού το πλήθος των ψηφίων του αριθμού βρίσκεται στον εκθέτη, αν και υψωμένο σε αριθμό μικρότερο του 1 (αυτό είναι ένδειξη ότι το πρόβλημα δεν είναι τόσο δύσκολο όσο τα **NP**-πλήρη). Η δυσκολία του Factoring είναι βασικό συστατικό πολλών κρυπτοσυστημάτων δημοσίου κλειδιού, με κυριότερο το RSA. Επομένως, τυχόν εύρεση αποδοτικού αλγορίθμου θα οδηγούσε σε κατάρρευση τα συστήματα αυτά. Αξίζει τέλος να σημειωθεί ότι, με βάση και όσα είπαμε παραπάνω, δεν μπορεί να αποκλειστεί το ενδεχόμενο το Factoring να επιλύεται πολυωνυμικά, ακόμη και αν $\mathbf{P} \neq \mathbf{NP}$.

## 16.10 Η κρυπτολογία στον σύγχρονο κόσμο

Είναι κοινός πλέον τόπος ότι οι κρυπτογραφικές εφαρμογές, πρωτόκολλα και τεχνικές έχουν κομβικό ρόλο στη σύγχρονη τεχνολογία, ειδικά στους τομείς της ασφαλούς επικοινωνίας, της ασφαλούς πρόσβασης, των ηλεκτρονικών ψηφοφοριών, της ανάκτησης και διαχείρισης ευαίσθητων δεδομένων, και των ηλεκτρονικών συναλλαγών, με πρόσφατη σημαντικότατη εξέλιξη την ανάπτυξη του κρυπτονομίσματος Bitcoin, και αρκετών ήδη διαδόχων του, με κυρίαρχο χαρακτηριστικό την απουσία κεντρικού ελέγχου.

Οι παραπάνω εφαρμογές, και πολλές άλλες που δεν αναφέρθηκαν, μπόρεσαν να πραγματοποιηθούν χάρη στην αλματώδη ανάπτυξη επαναστατικών ιδεών και αλγορίθμων, όπως η τέλεια μυστικότητα, η κρυπτογραφία δημοσίου κλειδιού, η ασφαλής ανταλλαγή κλειδιού από απόσταση, οι ψηφιακές υπογραφές, τα διαλογικά συστήματα αποδείξεων και οι αποδείξεις μηδενικής γνώσης, οι γεννήτριες ψευδοτυχαιότητας, η σύνθεση πρωτοκόλλων, οι συναρτήσεις κατακερματισμού χωρίς συγκρούσεις, η υπολογιστική πολυπλοκότητα, και πολλά άλλα.

Κοινό χαρακτηριστικό των παραπάνω μεθόδων είναι ότι η ασφάλειά τους εδράζεται, όλο και περισσότερο, σε αυστηρές μαθηματικές αποδείξεις. Για παράδειγμα, είμαστε πλέον σε θέση να διενεργούμε ηλεκτρονικές ψηφοφορίες που παρέχουν αποδείξεις ορθότητας για διάφορες φάσεις της λειτουργίας τους. Ή, να διενεργούμε συναλλαγές χωρίς κεντρική αρχή, μέσω του Bitcoin ή άλλων κρυπτονομισμάτων, με απόδειξη εγκυρότητας για κάθε συναλλαγή, που επικυρώνεται συλλογικά! Τα κρυπτονομίσματα είναι μια επανάσταση σε εξέλιξη, ανοίγοντας δρόμους για αποκεντρωμένη και αποδεδειγμένα ασφαλή ψηφιακή υλοποίηση λειτουργιών που μέχρι σήμερα απαιτούσαν την ύπαρξη κάποιας αρχής, όπως για παράδειγμα τη σύναψη συμβολαίων.

Τα μαθηματικά γίνονται για άλλη μια φορά επίκαιρα, βοηθώντας στην εμπέδωση εμπιστοσύνης σε κρίσιμες λειτουργίες, και μέσω αυτής στο άνοιγμα της κρυπτογραφίας στο πλατύ κοινό. Οι περισσότεροι αλγόριθμοι είναι πλέον τελείως ανοιχτοί, και η ασφάλειά τους βασίζεται αποκλειστικά σε αλγορίθμους που μας επιτρέπουν να εκτελούμε αποδοτικά πράξεις με αριθμούς χιλιάδων ψηφίων, ώστε η υπολογιστική δυσκολία (πολυπλοκότητα) των αντίστροφων πράξεων να είναι τεράστια. Η κρυπτογραφία έχει φύγει οριστικά από τα στεγανά των μυστικών υπηρεσιών και τη στρατιωτική χρήση και είναι έτοιμη να προσφέρει ακόμη περισσότερο τις υπηρεσίες της στο σύνολο της ανθρωπότητας πλέον, προάγοντας τη δημοκρατία, τον σεβασμό της ιδιωτικής ζωής, και τελικά την ενεργό και ισότιμη συμμετοχή όλων στο οικονομικό, πολιτικό, και κοινωνικό γίγνεσθαι.

## 16.11 Διαδραστικό Υλικό – Σύνδεσμοι

- Διαδραστικές Παρουσιάσεις - Video
  - Το πανεπιστήμιο του Rhode Island έχει συγκεντρώσει παρουσιάσεις κλασικών κρυπτοσυστημάτων στους παρακάτω συνδέσμους :
    * Κρυπτοσυστήματα Ολίσθησης
    * Κρυπτοσυστήματα Affine
    * Κρυπτοσυστήματα Αντικατάστασης
    * Vigenére
  - The BLACK Chamber, Διαδραστικές παρουσιάσεις κλασικών συστημάτων από τον Simon Singh, συγγραφέα ενός από τα πιο διάσημα βιβλία κρυπτογραφίας για το ευρύ κοινό
  - Προσομοιωτής Μηχανής Enigma
  - Αποσυναρμολόγηση Μηχανής Enigma
  - Ανακατασκευή μηχανής Bombe του Turing
  - Επίδειξη τέλειας μυστικότητας Shannon
  - Διαλέξεις για αποδείξεις μέσω κρυπτογραφικών αναγωγών
- Διαδραστικές Υλοποιήσεις
  - Sharky's Vigenére Cipher

  - Vigenére Cipher Codebreaker

- Κώδικας
  - Βιβλιοθήκη κλασικών κρυπτοσυστημάτων στο Sage
  - Μηχανή Enigma σε Javascript
  - Κρυπτογράφηση και κρυπτανάλυση κλασικών κρυπτοσυστημάτων σε Python

# Βιβλιογραφία


[1] M. Bellare and S. Goldwasser. *Lecture notes in Cryptography*. 1995.

[2] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.

[3] J. Daemen and V. Rijmen. *The Design of Rijndael:AES-The Advanced Encryption Standard*. Springer, 2002.

[4] S. Haber and W.S. Stornetta. How to timestamp a digital document. *Journal of Cryptology*, 3:pp 99–111, 1991.

[5] D. Kahn. *The Codebreakers. The Story of Secret Writing.* Macmillan, 1967.

[6] Neal Koblitz. *A Course in Number Theory and Cryptography*. Springer Verlag, 1994.

[7] Evangelos Kranakis. *Primality and Cryptography*. John Wiley & Sons, 1987.

[8] Ralph C. Merkle. Secure communications over insecure channels. *Communications of the ACM*, 21(4):294–299, April 1978.

[9] A. J. Menezes, P.C van Oorschot, and S. A. Vanstone. *Handbook of applied cryptography*. CRC Press, 1996.

[10] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[11] C. E. Shannon. A mathematical theory of communication. *Bell Systems Technical Journal*, 27:379–423, 623–656, 1948.

[12] A Shamir. A polynomial-time algorithm for breaking the basic merkle-hellman cryptosystem. *Transactions on Information Theory*, 30:pp 699–704, 1984.

[13] Douglas R. Stinson. *Cryptography. Theory and Practice.* Discrete Mathematics and its Applications. CRC Press, 1995.

[14] C. E. Shannon, “Communication theory of secrecy systems,” *Bell System Technical Journal*, vol. 28, no. 4, pp. 656–715, 1949.

[15] A. J. Menezes, P. C. Van Oorschot and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996.

[16] Bluetooth SIG, Specification of the Bluetooth system, version 1.1, February 22, 2001, http://www.bluetooth.com.

[17] M. Bellare and S. Goldwasser. *Lecture notes in Cryptography*. 1995.

[18] Satoshi Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System", The Cryptography Mailing list at metzdowd.com, 2008.

# Ευρετήριο