Αλγόριθμοι και Πολυπλοκότητα

Ν. Μ. Μισυρλής

Τμήμα Πληροφορικής και Τηλεπικοινωνιών,

Πανεπιστήμιο Αθηνών

Δυναμικός Προγραμματισμός

Εισαγωγή

- Ο δυναμικός προγραμματισμός (dynamic programming), αποτελεί ένα ισχυρό εργαλείο, για την επίλυση συνδυαστικών προβλημάτων βελτιστοποίησης.
- Στον δυναμικό προγραμματισμό, η επίλυση ενός προβλήματος, προκύπτει από την λύση μικρότερων υποπροβλημάτων τα οποία **αλληλοεπικαλύπτονται** (overlapping subproblems).
- Κάθε υποπρόβλημα λύνεται μια μόνο φορά και η βέλτιστη λύση αποθηκεύεται σ' ένα πίνακα.

Δυναμικός Προγραμματισμός

Εισαγωγή

- Τέσσερα είναι τα βασικά βήματα για την ανάπτυξη ενός αλγορίθμου δυναμικού προγραμματισμού.
  1. Χαρακτηρισμός της δομής της βέλτιστης λύσης.
  2. Αναδρομικός ορισμός της τιμής της βέλτιστης λύσης.
  3. Υπολογισμός της τιμής της βέλτιστης λύσης από *κάτω προς τα πάνω*.
  4. Κατασκευή της βέλτιστης λύσης με χρήση πίνακα.

Δυναμικός Προγραμματισμός

Βασικά Στοιχεία Δυναμικού Προγραμματισμού

Βέλτιστα διασπώμενη δομή (optimal substructure)

- Κάθε υποστρατηγική μιας βέλτιστης στρατηγικής είναι και η ίδια βέλτιστη.
- Η βέλτιστη λύση ενός προβλήματος, εμπεριέχει βέλτιστες λύσεις υποπροβλημάτων του.

Δυναμικός Προγραμματισμός

Βέλτιστα διασπώμενη δομή (optimal substructure)

- Μια μεθοδολογία για να ανακαλύψουμε την βέλτιστα διασπώμενη δομή
  1. Αποδεικνύουμε ότι μια οποιαδήποτε λύση στο πρόβλημα, συνίσταται στο να κάνουμε μια επιλογή. Η επιλογή αυτή οδηγεί σε ένα ή περισσότερα υποπροβλήματα τα οποια πρέπει να επιλυθούν.
  2. Θεωρούμε τώρα ότι έχουμε δεδομένη την επιλογή η οποία οδηγεί στην βέλτιστη λύση, χωρίς να μας απασχολεί προς το παρόν, πως ορίζεται αυτή.
  3. Δεδομένης λοιπόν της επιλογής αυτής, ορίζουμε τα νέα υποπροβλήματα.
  4. Τέλος αποδεικνύουμε ότι όντως οι βέλτιστες λύσεις των υποπροβλημάτων αυτών οδηγούν στη βέλτιστη λύση του αρχικού προβλήματος. Η απόδειξη γίνεται συνήθως με *εις άτοπον απαγωγή*.

Δυναμικός Προγραμματισμός

Βέλτιστα διασπώμενη δομή (optimal substructure)

- Η τεχνική του δυναμικού προγραμματισμού χρησιμοποιεί τη βέλτιστα διασπώμενη δομή, από **κάτω προς τα πάνω** (bottom-up).
- Στην ουσία χρησιμοποιούμε μια αναδρομική σχέση ώστε να υπολογίσουμε τη βέλτιστη λύση του αρχικού προβλήματος.
- Κριτήριο **ανεξαρτησίας** (independency) των υποπροβλημάτων: διασφαλίζει τη βέλτιστα διασπώμενη δομή, δηλ. ότι η λύση ενός προβλήματος δεν επηρεάζει τα υπόλοιπα.

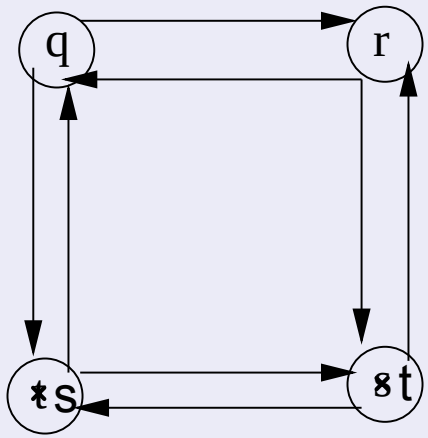
Δυναμικός Προγραμματισμός

Βέλτιστα διασπώμενη δομή (optimal substructure)

- Έστω το μέγιστο απλό μονοπάτι $q \rightsquigarrow t$, δυο μέγιστα απλά μονοπάτια $p_1 : q \rightsquigarrow r$ και $p_2 : r \rightsquigarrow t$.
- Όμως το p_1 είναι το $q \rightarrow s \rightarrow t \rightarrow r$.
- Ο συνδυασμός των λύσεων δίνει μέγιστο μονοπάτι που δεν είναι απλό.
- Δεν υπάρχει λύση για το δεύτερο υποπρόβλημα άρα το αρχικό μας πρόβλημα δεν έχει λύση.

Δυναμικός Προγραμματισμός

Το γράφημα του σχήματος δείχνει ότι δεν υπάρχει βέλτιστα διασπώμενη δομή για το πρόβλημα του μέγιστου απλού μονοπατιού



Δυναμικός Προγραμματισμός

Επικαλυπτόμενα υποπροβλήματα (Overlapping subproblems)

- Το δεύτερο κριτήριο είναι των **επικαλυπτόμενων υποπροβλημάτων**.
- Κάθε αλγόριθμος δυναμικού προγραμματισμού, παράγει πολυωνυμικό, πλήθος διαφορετικών υποπροβλημάτων.
- Κάθε ένα από τα υποπροβλήματα εμφανίζεται, πιθανόν, αρκετές φορές.
- Μόλις επιλυθεί ένα υποπρόβλημα, η τιμή της βέλτιστης λύσης του, αποθηκεύεται σ' ένα πίνακα.
- Όταν απαιτηθεί να επιλυθεί ξανά, τότε απλά ανακαλείται η τιμή της βέλτισης λύσης του υποπροβλήματος από τον πίνακα (σε σταθερό χρόνο).

Δυναμικός Προγραμματισμός

Επικαλυπτόμενα υποπροβλήματα (Overlapping subproblems)

Παρατηρήσεις

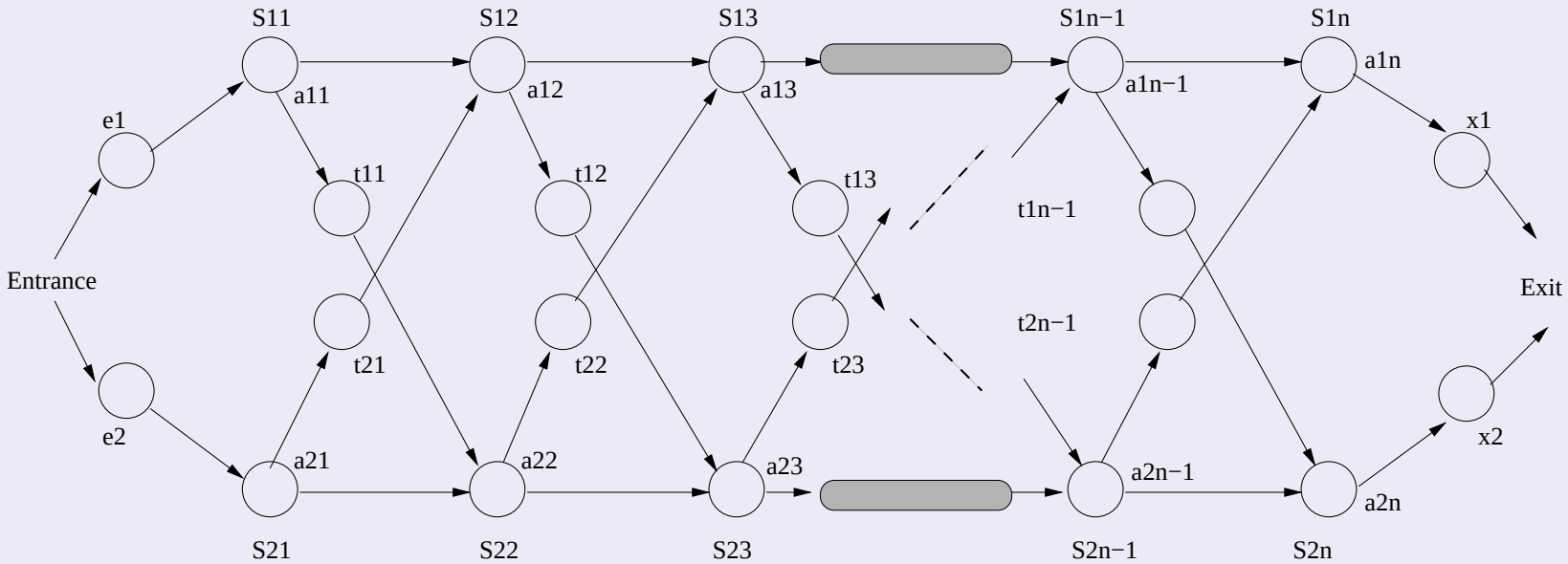
- Δεν θα πρέπει να δημιουργείται σύγχυση ανάμεσα στην ιδιότητα της επικάλυψης με αυτή της ανεξαρτησίας.
- περίπτωση 1: Το ίδιο υποπρόβλημα που εμφανίζεται περισσότερες φορές κατά την διάρκεια της εκτέλεσης.
- περίπτωση 2: Ανεξαρτησία των δεδομένων του υποπροβλήματος.

Δυναμικός Προγραμματισμός

Εφαρμογές

Χρονοδρομολόγηση γραμμής παραγωγής

- Ζητάμε να αποφασίσουμε από ποιούς σταθμούς θα περάσει το αυτοκίνητο έτσι ώστε να ελαχιστοποιήσουμε το χρόνο κατασκευής του.
- Εργοστάσιο κατασκευής αυτοκινήτων με δύο γραμμές παραγωγής.



Δυναμικός Προγραμματισμός

Εφαρμογές

Βήμα 1: Βέλτιστα διασπώμενη δομή:

- Η βέλτιστη λύση του προβλήματος της εύρεσης του συντομότερου μονοπατίου μέχρι τον S_{1j} εμπεριέχει τις βέλτιστες λύσεις στα υποπροβλήματα των συντομότερων μονοπατιών προς τους σταθμούς S_{1j-1} και S_{2j-1}.

Δυναμικός Προγραμματισμός

Εφαρμογές

Βήμα 2: Αναδρομικός ορισμός της τιμής της βέλτιστης λύσης

- Ορίζουμε τον πίνακα f διάστασης $2 \times n$, όπου στη θέση f_{ij} ($i = 1, 2$ και $j = 1, 2, \ldots, n$) αποθηκεύουμε τον ελάχιστο χρόνο μέχρι και τον σταθμό S_{ij}.
- Η βέλτιστη λύση:

$$f^* = \min\{f_{1n} + x_1, f_{2n} + x_2\} \tag{1}$$

- Δύο αναδρομικές σχέσεις που υπολογίζουν τις δύο γραμμές του πίνακα f.

$$f_{1j} = \begin{cases} e_1 + a_{11} & j = 1 \\ \min\{f_{1j-1} + a_{1j}, f_{2j-1} + t_{2j-1} + a_{1j}\} & j \geq 2 \end{cases} \tag{2}$$

και όμοια:

$$f_{2j} = \begin{cases} e_2 + a_{21} & j = 1 \\ \min\{f_{2j-1} + a_{2j}, f_{1j-1} + t_{1j-1} + a_{2j}\} & j \geq 2 \end{cases} \tag{3}$$

Δυναμικός Προγραμματισμός

Εφαρμογές

Βήμα 3: Υπολογισμός του ελάχιστου χρόνου

- από (1), (2) ή (3)

$$T_1(n) = T_2(n) = T_1(n-1) + T_2(n-1)$$

. Αρα

$$T_1(n) = T_2(n) = \Omega(2^n)$$

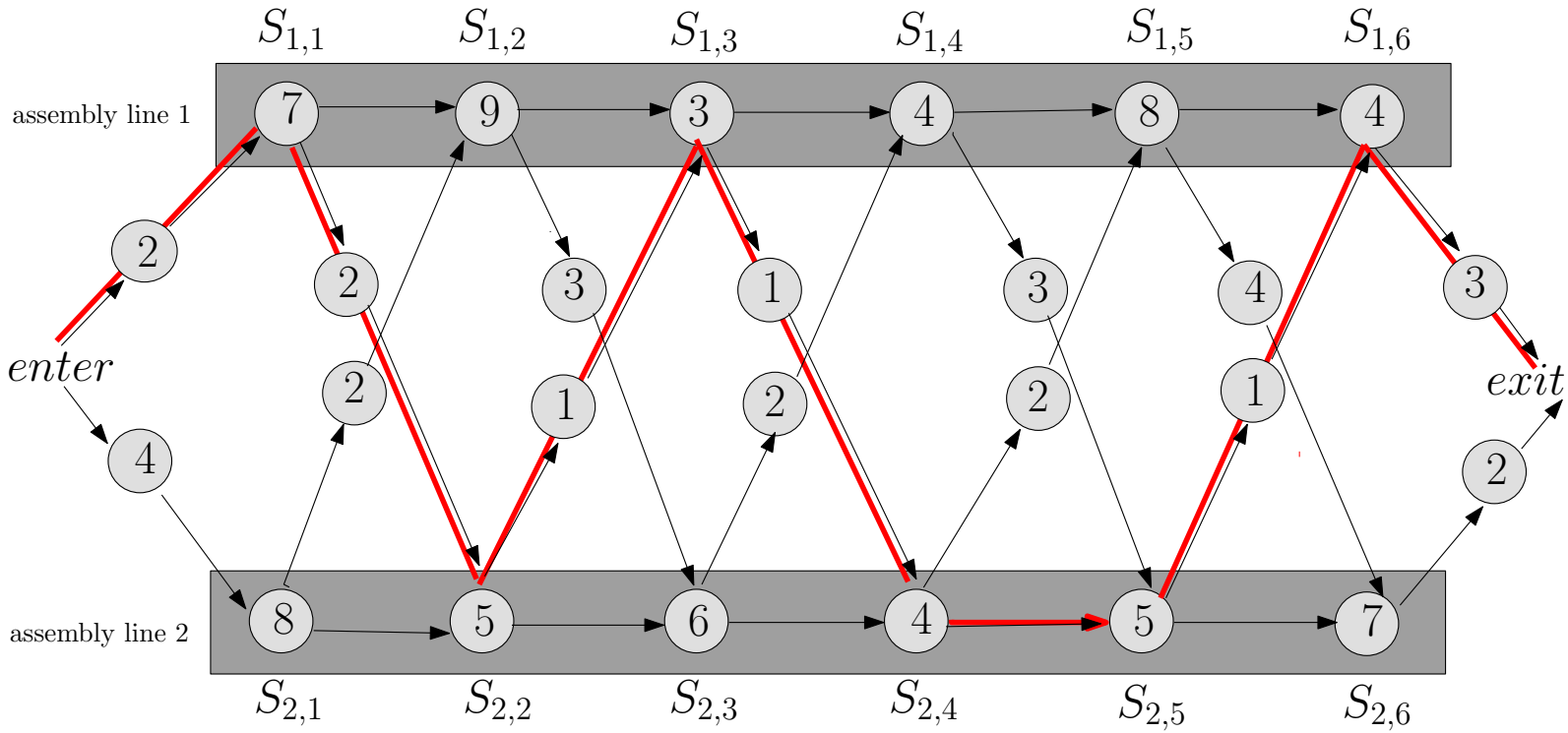
.

- Μπορούμε να κάνουμε καλύτερα από εκθετικό χρόνο · Ναι.
- Αποθηκεύουμε κάθε τιμή f_{ij} σε ένα πίνακα (διαστάσεων $2 \times n$)
- Ανακαλούμε κάθε φορά την τιμή από κάποιο προηγούμενο βήμα.
- Βέλτιστη διαδρομή σε χρόνο $\Theta(n)$.

Δυναμικός Προγραμματισμός

1. $f_{11} \leftarrow e_1 + a_{1,1}$
2. $f_{21} \leftarrow e_2 + a_{2,1}$
3. **for** $j = 2$ **to** n **do**
4. **if** $f_{1,j-1} + a_{1,j} \leq f_{2,j-1} + t_{2,j-1} + a_{1,j}$ **then**
5. $f_{1j} = f_{1,j-1} + a_{1,j}$
6. $l_{1j} = 1$
7. **else**
8. $f_{1j} = f_{2,j-1} + t_{2,j-1} + a_{1,j}$
9. $l_{1j} = 2$
10. **end if**
11. **if** $f_{2,j-1} + a_{2,j} \leq f_{1,j-1} + t_{1,j-1} + a_{2,j}$ **then**
12. $f_{2j} = f_{2,j-1} + a_{2,j}$
13. $l_{2j} = 2$
14. **else**
15. $f_{2j} = f_{1,j-1} + t_{1,j-1} + a_{2,j}$
16. $l_{2j} = 1$
17. **end if**
18. **if** $f_{1n} + x_1 \leq f_{2n} + x_2$ **then**
19. $f^* = f_{1n} + x_1$
20. $l^* = 1$
21. **else**
22. $f^* = f_{2n} + x_2$
23. $l^* = 2$
24. **end if**

Δυναμικός Προγραμματισμός



	1	2	3	4	5	6
$f_1[j]$	9	18	20	24	32	35
$f_2[j]$	12	16	22	25	30	37

$f^* = 38$

	2	3	4	5	6
$\ell_1[j]$	1	2	1	1	2
$\ell_2[j]$	1	2	1	1	2

$\ell^* = 1$

Δυναμικός Προγραμματισμός

Αλυσιδωτός πολλαπλασιασμός πινάκων

- Ζητάμε να υπολογίσουμε το γινόμενο

$$A_1 \cdot A_2 \cdot \ldots A_n \tag{4}$$

  όσο το δυνατόν πιο αποδοτικά (με το λιγότερο δυνατό κόστος).
- Το κόστος πολλαπλασιασμού δύο πινάκων A, B με διαστάσεις $p \times q$ και $q \times r$ αντίστοιχα, είναι $p \times q \times r$.
- Το κόστος του πολλαπλασιασμού μεταβαλλεται ανάλογα με τη σειρά που θα εκτελέσουμε τους πολλαπλασιασμούς.
- Παράδειγμα: Έστω το γινόμενο $A_1 \cdot A_2 \cdot A_3 \cdot A_4$. $(A_1 \cdot ((A_2 \cdot A_3) \cdot A_4))$ ή

$$((A_1 \cdot A_2) \cdot (A_3 \cdot A_4))$$

Δυναμικός Προγραμματισμός

Αλυσιδωτός πολλαπλασιασμός πινάκων

- Άσκηση: Να δειχθεί ότι ο υπολογισμός του αριθμού των παρενθετοποιήσεων δίδεται από την αναδρομή:

$$T(n) = \begin{cases} 1 & \text{αν } , n = 1 \\ \sum_{k=1}^{n-1} T(k)T(n-k) & , \text{αν } n \geq 2 \end{cases} \tag{5}$$

- Επίσης να δειχθέι ότι η λύση είναι $\Omega(2^n)$.

Δυναμικός Προγραμματισμός

Αλυσιδωτός πολλαπλασιασμός πινάκων

Βήμα 1: Βέλτιστα διασπώμενη δομή Βήμα 1: Βέλτιστα διασπώμενη δομή

- θα χρησιμοποιήσουμε τον συμβολισμό A_{ij} εννοώντας το γινόμενο $A_i \cdot A_{i+1} \ldots A_j$ όπου $i \leq j$.
- Συμβολικά έχουμε:

$$\text{cost}(A_{ij}) = \text{cost}(A_{ik}) + \text{cost}(A_{k+1,j}) + \text{cost}(A_{ik} \cdot A_{k+1,j}) \tag{6}$$

- Η βέλτιστη παρενθετοποίηση του γινομένου A_{ij} είναι αυτή που προκύπτει από την βέλτιστη παρενθετοποίηση των γινομένων A_{ik} και $A_{k+1,j}$.
- Η βέλτιστη λύση του προβλήματος μας, έγκειται στον υπολογισμό του ελαχίστου κόστους της παρενθετοποίησης του γινομένου A_{1n}.

Δυναμικός Προγραμματισμός

Αλυσιδωτός πολλαπλασιασμός πινάκων

Βήμα 2: Αναδρομικός ορισμός της τιμής της βέλτιστης λύσης

- Ο πίνακας A_{ik} έχει διαστάσεις $p_{i-1} \times p_k$ και αντίστοιχα ο $A_{k+1,j}$, $p_k \times p_j$

$$\text{cost}(A_{ik} \cdot A_{k+1,j}) = p_{i-1} \times p_k \times p_j. \qquad 1 \leq i \leq k < j \leq n$$

- Ορίζουμε τον πίνακα m διάστασης $n \times n$, όπου στη θέση $m[i,j]$ αποθηκεύουμε το ελάχιστο κόστος που απαιτείται για τον υπολογισμό του γινομένου A_{ij}.
- Στη θέση $m[1,n]$ αποθηκεύεται το ελάχιστο κόστος υπολογισμού του γινομένου A_{1n}.

$$m[i,j] = \min_{i \leq k < j} \left\{ m[i,k] + m[k+1,j] + p_{i-1} \cdot p_k \cdot p_j \right\}.$$

Δυναμικός Προγραμματισμός

Αλυσιδωτός πολλαπλασιασμός πινάκων

Βήμα 2: Αναδρομικός ορισμός της τιμής της βέλτιστης λύσης

- Συνοπτικά λοιπόν έχουμε:

$$m[i,j] = \begin{cases} 0 & , i = j \\ \min_{i \le k < j} \left\{ m[i,k] + m[k+1,j] + p_{i-1} \cdot p_k \cdot p_j \right\}, & i < j \end{cases} \tag{7}$$

Δυναμικός Προγραμματισμός

Αλυσιδωτός πολλαπλασιασμός πινάκων

Η 7 υλοποιείται ως εξής:

ΑΝΑΔΡΟΜΙΚΗ ΑΚΟΛΟΥΘΙΑ ΠΙΝΑΚΩΝ (p, i, j)

1. Αν $i = j$
2. τότε επιστροφή 0
3. $m[i, j] = \infty$
4. Για $k = i$ έως $j - 1$
5. $q =$ΑΝΑΔΡΟΜΙΚΗ ΑΚΟΛΟΥΘΙΑ ΠΙΝΑΚΩΝ (p, i, k)
6. $+$ ΑΝΑΔΡΟΜΙΚΗ ΑΚΟΛΟΥΘΙΑ ΠΙΝΑΚΩΝ $(p, k + 1, n)$
7. $+p_{i-1}p_k p_j$
8. αν $q < m[i, j]$
9. τότε $m[i, j] = q$
10. επιστροφή $m[i, j]$

Δυναμικός Προγραμματισμός

Αλυσιδωτός πολλαπλασιασμός πινάκων

$$T(1) \geq 1$$

$$T(n) \geq 1 + \sum_{k=1}^{n-1} (T(k) + T(n-k) + 1), n > 1$$

$$T(n) \geq 2 \sum_{k=1}^{n-1} T(i) + n$$

Μέθοδος αντικατάστασης

Θα δείξουμε ότι

$$T(n) \geq 2^{n-1}$$

Βάση: $n = 1$ έχουμε

$$T(1) \geq 2^0 = 1$$

το οποίο ισχύει.

Βήμα: Για $n \geq 2$ έχουμε

Δυναμικός Προγραμματισμός

Αλυσιδωτός πολλαπλασιασμός πινάκων- Μέθοδος αντικατάστασης

$$
\begin{aligned}
T(n) &\geq 2\sum_{i=1}^{n-1} 2^{i-1} + n \\
&= 2\sum_{i=0}^{n-2} 2^{i} + n \\
&= 2(2^0 + 2^1 + \cdots + 2^{n-2}) + n \\
&= \left(\frac{2^{n-2}\cdot 2 - 2^0}{2-1}\right) + n \\
&= 2(2^{n-1} - 1) + n \\
&= 2^n + n - 2 \geq 2^n
\end{aligned}
$$

Επομένως

$$T(n) = \Omega(2^n).$$

Δυναμικός Προγραμματισμός

Υπομνηματισμός

- Η βασική ιδέα είναι να **υπομνηματίσουμε** τον φυσικό, αλλά βραδύ, αναδρομικό αλγόριθμο.
- Τηρούμε έναν πίνακα με λύσεις υποπροβλημάτων
- Ένας υπομνηματικός αναδρομικός αλγόριθμος τηρεί ένα στοιχείο πίνακα για τη λύση του κάθε υποπροβλήματος. Το κάθε στοιχείο πίνακα αρχικά περιέχει μια ειδική τιμή η οποία υποδεικνύει ότι το στοιχείο δεν έχει ακόμα συμπληρωθεί. Όταν το αντίστοιχο υποπρόβλημα απαντάται για πρώτη φορά κατά την εκτέλεση του αναδρομικού αλγορίθμου, υπολογίζεται η λύση του και κατόπιν αποθηκεύεται στον πίνακα. Στη συνέχεια, κάθε φορά που απαντάται ξανά το συγκεκριμένο υποπρόβλημα, απλώς ανακαλείται και επιστρέφεται η τιμή που είναι αποθηκευμένη στον πίνακα.

Δυναμικός Προγραμματισμός

Υπομνηματισμός

ΥΠΟΜΝΗΜΑΤΙΚΗ ΑΚΟΛΟΥΘΙΑ ΠΙΝΑΚΩΝ (p)

1. $n \leftarrow$ μήκος$[p] - 1$
2. Για $i \leftarrow 1$ έως n
3. για $j \leftarrow 1$ έως n
4. $m[i, j] \leftarrow \infty$
5. επιστροφή ΑΝΑΚΛΗΣΗ ΑΛΛΗΛΟΥΧΙΑΣ $(p, 1, n)$

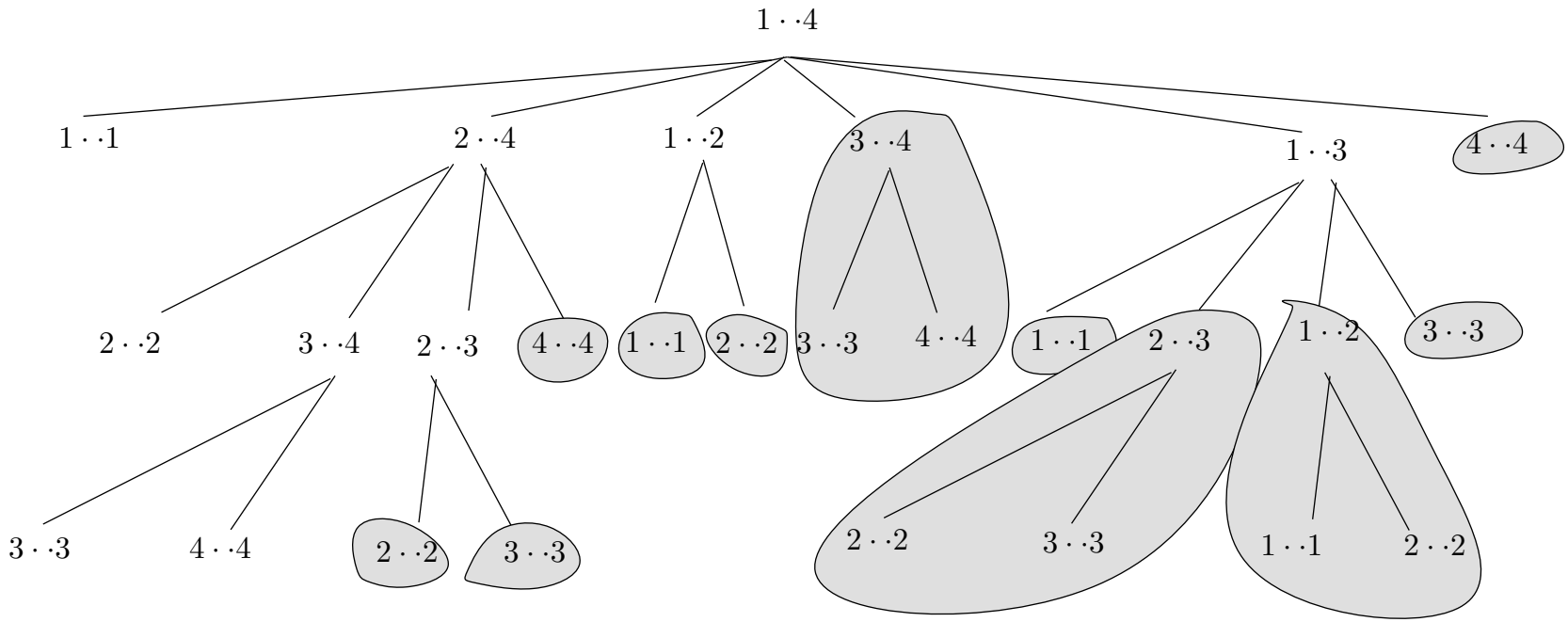
Δυναμικός Προγραμματισμός

Υπομνηματισμός

ΑΝΑΚΛΗΣΗ ΑΛΛΗΛΟΥΧΙΑΣ (p, i, j)

1. αν $m[i, j] < \infty$
2. τότε επιστροφή $m[i, j]$
3. αν $i = j$
4. $m[i, j] \leftarrow 0$
5. άλλως για $k \leftarrow i$ έως $j - 1$
6. $q \leftarrow$ ΑΝΑΚΛΗΣΗ ΑΛΛΗΛΟΥΧΙΑΣ (p, i, k)
7. $+$ΑΝΑΚΛΗΣΗ ΑΛΛΗΛΟΥΧΙΑΣ $(p, k + 1, j) + p_{i-1} p_k p_j$
8. αν $q < m[i, j]$
9. τότε $m[i, j] \leftarrow q$
10. επιστροφή $m[i, j]$

Δυναμικός Προγραμματισμός



Δυναμικός Προγραμματισμός

Αλυσιδωτός πολλαπλασιασμός πινάκων

Βήμα 3: Υπολογισμός του ελαχίστου κόστους

$Matrix - Chain - Order(p)$

1. $n = length[p] - 1$
2. **for** $i = 1$ **to** n **do**
3. $\quad m[i, i] = 0$
4. **end for**
5. **for** $l = 2$ **to** n **do**
6. $\quad$ **for** $i = 1$ **to** $n - l + 1$ **do**
7. $\quad\quad j = i + l - 1$
8. $\quad\quad m[i, j] = \infty$
9. $\quad\quad$ **for** $k = i$ **to** $j - 1$ **do**
10. $\quad\quad\quad q = m[i, k] + m[k + 1, j] + p_{i-1} \cdot p_k \cdot p_j$

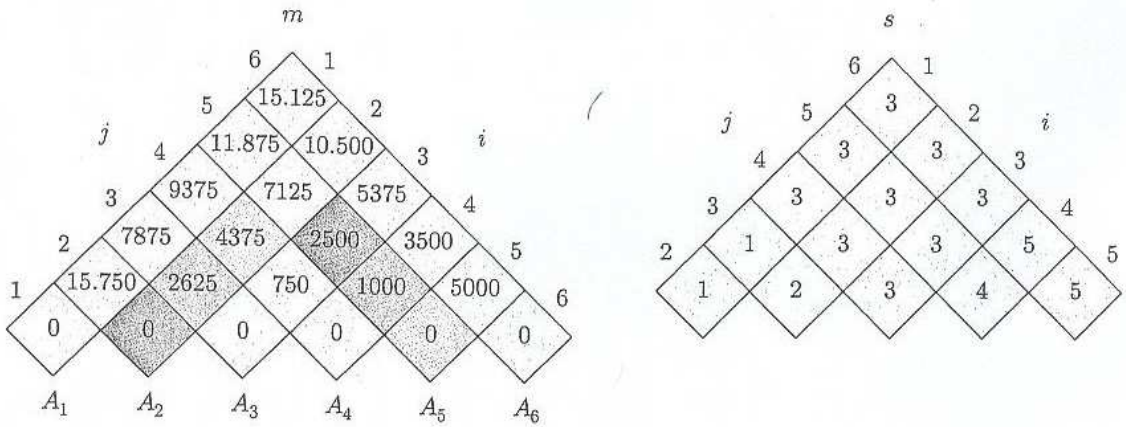
Δυναμικός Προγραμματισμός

Αλυσιδωτός πολλαπλασιασμός πινάκων

```
11.                         if q < m[i, j] then
12.                             m[i, j] = q
13.                             s[i, j] = k
14.                         end if
15.                     end for
16.                 end for
17.             end for
18.             return m, s
```

Βήμα 4: Κατασκευή της βέλτιστης λύσης

Δυναμικός Προγραμματισμός



Σχήμα 15.3 Οι πίνακες m και s που υπολογίζονται από τη διαδικασία ΔΙΑΤΑΞΗ ΑΛΛΗΛΟΥΧΙΑΣ ΠΙΝΑΚΩΝ για $n = 6$ και τις παρακάτω διαστάσεις πινάκων:

πίνακας	διαστάσεις
A_1	30×35
A_2	35×15
A_3	15×5
A_4	5×10
A_5	10×20
A_6	20×25

Οι πίνακες είναι στραμμένοι έτσι ώστε η κύρια διαγώνιος να εκτείνεται κατά την οριζόντια διεύθυνση. Στον πίνακα m χρησιμοποιούνται μόνο η κύρια διαγώνιος και το άνω τριγωνικό τμήμα, ενώ στον πίνακα s χρησιμοποιείται μόνο το άνω τριγωνικό τμήμα. Το ελάχιστο πλήθος βαθμωτών πολλαπλασιασμών που απαιτούνται για τον πολλαπλασιασμό των 6 πινάκων είναι $m[1, 6] = 15.125$. Από τα στοιχεία με την εντονότερη σκίαση, τα στοιχεία που έχουν την ίδια σκίαση συνδυάζονται μεταξύ τους ανά δύο στη γραμμή 9 κατά τον υπολογισμό του στοιχείου

$$m[2,5] = \min \begin{cases} m[2,2] + m[3,5] + p_1 p_2 p_5 = 0 + 2500 + 35 \cdot 15 \cdot 20 = 13.000\,, \\ m[2,3] + m[4,5] + p_1 p_3 p_5 = 2625 + 1000 + 35 \cdot 5 \cdot 20 = 7125\,, \\ m[2,4] + m[5,5] + p_1 p_4 p_5 = 4375 + 0 + 35 \cdot 10 \cdot 20 = 11.375, \end{cases}$$
$$= 7125\,.$$

Δυναμικός Προγραμματισμός

$Print - Optimal - Parents(s, i, j)$

1. **if** $i = j$ **then**
2. τύπωσε ''A''$_i$
3. **else**
4. τύπωσε '('
5. $Print - Optimal - Parents(s, i, s[i, j])$
6. $Print - Optimal - Parents(s, s[i, j] + 1, j)$
7. τύπωσε ')'
8. **end if**

Δυναμικός Προγραμματισμός

Το πρόβλημα του σακιδίου

- Ένα σύνολο $X = \{x_1, x_2, \ldots, x_n\}$ από n αντικείμενα
- το καθένα έχει βάρος (θετικός ακέραιος) a_i και ένα αξία (θετικό) c_i, $i = 1, 2, \ldots, n$.
- Δίνεται ένας ακέραιος θετικός W.
- Ζητάμε να επιλέξουμε ένα υποσύνολο $Y \subseteq X$ τέτοιο ώστε

$$\max \sum_{x_i \in Y} c_i$$

και επιπλέον

$$\sum_{x_i \in Y} a_i \leq W$$

- Το πρόβλημα αυτό είναι $\mathcal{NP} - \mathrm{hard}$.
- Θα δώσουμε αλγόριθμο δυναμικού προγραμματισμού, που επιλύει το πρόβλημα αυτό σε **ψευδοπολυωνυμικό** χρόνο.

Δυναμικός Προγραμματισμός

Το πρόβλημα του σακιδίου

Βήμα 1: Βέλτιστα διασπώμενη δομή

- $$P_k(y) = \left\{ \max \sum_{j=1}^{k} c_j x_j, \quad \sum_{j=1}^{k} a_j x_j \leq y, \ x_j = 0 \text{ ή } 1 \right\} \tag{8}$$
- $$0 \leq y \leq W, 1 \leq k \leq n$$

Δυναμικός Προγραμματισμός

Το πρόβλημα του σακιδίου

Βήμα 2: Αναδρομικός ορισμός της τιμής της βέλτιστης λύσης

- Έστω $f_k(y)$ η τιμή της βέλτιστης λύσης του υποπροβλήματος $P_k(y)$. Τότε διακρίνουμε δύο περιπτώσεις
- 1. Αν $x_n \notin P_n(W)$ τότε $f_n(W) = f_{n-1}(W)$
- 2. Αν $x_n \in P_n(W)$ τότε $f_n(W) = c_n + f_{n-1}(W - a_n)$
- Αν $W < a_n$ τότε $f_n(W) = f_{n-1}(W)$
- διαφορετικά βέλτιστη λύση από 1. και 2.

Δυναμικός Προγραμματισμός

Το πρόβλημα του σακιδίου

Βήμα 2: Αναδρομικός ορισμός της τιμής της βέλτιστης λύσης

- Έστω $f_k(y)$ η τιμή της βέλτιστης λύσης του υποπροβλήματος $P_k(y)$. Τότε

$$f_{k+1}(y) = \begin{cases} f_k(y) & \text{αν } a_{k+1} > y \\ \max\left\{f_k(y),\ f_k(y - a_{k+1}) + c_{k+1}\right\}, & \text{διαφορετικά} \end{cases} \tag{9}$$

- για $1 \leq k \leq n$ και $0 \leq y \leq W$.
- Η βέλτιστη λύση του προβλήματος $P_n(W)$ έχει την τιμή $f_n(W)$.

Δυναμικός Προγραμματισμός

Αναδρομικός Αλγόριθμος

```
ΚΟΣΤΟΣ(k, y)
1  αν k = 1 τότε
2      αν a_1 > y τότε
3          f(1, y) ← 0
4      άλλως
5          f(1, y) ← c_1
6  άλλως
7      αν a_{k+1} > y τότε
8          f(k, y) ← ΚΟΣΤΟΣ(k, y)
9      άλλως
10         f(k, y) ← max{ΚΟΣΤΟΣ(k, y), ΚΟΣΤΟΣ(k, y − a_{k+1}) + c_{k+1}}
11 Επιστροφή f(k, y)
```

Απόδειξη Ορθότητας

Επαγωγή

Βάση. Για $k = 1$ έχουμε, λόγω των 1-5

$$\text{ΚΟΣΤΟΣ}(1, y) = \begin{cases} 0, & \text{αν } a_1 > y \\ c_1, & \text{διαφορετικά} \end{cases}$$

ή, λόγω της (9)

$$\text{ΚΟΣΤΟΣ}(1, y) = f_1(y), \ \ 0 \leq y \leq W.$$

Επαγωγικό Βήμα
Υποθέτουμε ότι ισχύει

$$\text{ΚΟΣΤΟΣ}(k, y) = f_k(y), \ \ 0 \leq y \leq W. \tag{10}$$

και θα δείξουμε ότι

$$\text{ΚΟΣΤΟΣ}(k+1, y) = f_{k+1}(y), \ \ 0 \leq y \leq W. \tag{11}$$

Δυναμικός Προγραμματισμός

Ορθότητα

Έχουμε, λόγω των 7-10, ότι

$$\text{ΚΟΣΤΟΣ}(k+1,y) = \begin{cases} \text{ΚΟΣΤΟΣ}(k,y), & \text{αν } a_{k+1} > y \\ \max\{\text{ΚΟΣΤΟΣ}(k,y), \text{ΚΟΣΤΟΣ}(k,y-a_{k+1}) + c_{k+1}\} \end{cases} \tag{12}$$

Λόγω της επαγωγικής υπόθεσης (10) έχουμε

$$\begin{aligned} &\text{ΚΟΣΤΟΣ}(k,y) = f_k(y) \\ &\text{και} \\ &\text{ΚΟΣΤΟΣ}(k,y-a_{k+1}) = f_k(y-a_{k+1}) \end{aligned} \tag{13}$$

Η (12), λόγω της (13) δίνει την (11).

Δυναμικός Προγραμματισμός

Το πρόβλημα του σακιδίου

Βήμα 3: Υπολογισμός της βέλτιστης λύσης

$$DYNAMIC - KNAPSACK(A, C, W)$$

1. $n = length(C)$
2. **for** $y = 0$ **to** W **do**
3. **if** $a_1 > y$ **then**
4. $f_1(y) = 0$
5. **else**
6. $f_1(y) = c_1$
7. **end if**
8. **end for**
9. **for** $k = 1$ **to** $n - 1$ **do**

Δυναμικός Προγραμματισμός

Το πρόβλημα του σακιδίου

10. **for** $y = 0$ **to** W **do**
11. **if** $a_{k+1} > y$ **then**
12. $f_{k+1}(y) = f_k(y)$
13. $x_{k+1}^y = 0$
14. **else if** $f_k(y) > f_k(y - a_{k+1}) + c_{k+1}$
15. $f_{k+1}(y) = f_k(y)$
16. $x_{k+1}^y = 0$
17. **else**
18. $f_{k+1}(y) = f_k(y - a_{k+1}) + c_{k+1}$
19. $x_{k+1}^y = 1$
20. **end if**
21. **end for**
22. **end for**
23. **return** f, x

Δυναμικός Προγραμματισμός

Το πρόβλημα του σακιδίου

Πολυπλοκότητα

$$T(n) = O(nW)$$

Δυναμικός Προγραμματισμός

Παράδειγμα

Βήμα 4: Κατασκευή της βέλτιστης λύσης

Αντικείμενα με βάρη και αξίες

$$(9, 20), (8, 16), (6, 11), (5, 9), (4, 7), (1, 1)$$

	1	2	3	4	5	6	7	8	9	10	11	12
1	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	20(1)	20(1)	20(1)	20(1)
2	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	16(1)	20(0)	20(0)	20(0)	20(0)
3	0(0)	0(0)	0(0)	0(0)	0(0)	11(1)	11(1)	16(0)	20(0)	20(0)	20(0)	20(0)
4	0(0)	0(0)	0(0)	0(0)	9(1)	11(0)	11(0)	16(0)	20(0)	20(0)	20(0)	20(0)
5	0(0)	0(0)	0(0)	7(1)	9(0)	11(0)	11(0)	16(0)	20(0)	20(0)	20(0)	23(1)
6	1(1)	1(1)	1(1)	7(0)	9(0)	11(0)	12(1)	16(0)	20(0)	21(1)	21(1)	23(0)

Δυναμικός Προγραμματισμός

Το πρόβλημα του σακιδίου

Βήμα 4: Κατασκευή της βέλτιστης λύσης

Αλγόριθμος κατασκευής βέλτιστης λύσης από τον πίνακα

Δυναμικός Προγραμματισμός

Το πρόβλημα του σακιδίου

$$Construct - Knapsack(x)$$

1. τύπωσε: αξία βέλτιστης λύσης: $f_n(W)$
2. τύπωσε 'Αντικείμενα στην λύση: '
3. **while** $W > 0$ **do**
4. **if** $x_k^W = 1$ **then**
5. τύπωσε: k
6. $W = W - a_k$
7. **end if**
8. $k = k - 1$
9. **end while**

Δυναμικός Προγραμματισμός

Το πρόβλημα του σακιδίου

$$(9, 20), (8, 16), (6, 11), (5, 9), (4, 7), (1, 1)$$

Βήμα 4: Κατασκευή της βέλτιστης λύσης

	1	2	3	4	5	6	7	8	9	10	11	12
1	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	20(1)	20(1)	20(1)	20(1)
2	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	**16(1)**	20(0)	20(0)	20(0)	20(0)
3	0(0)	0(0)	0(0)	0(0)	0(0)	11(1)	11(1)	**16(0)**	20(0)	20(0)	20(0)	20(0)
4	0(0)	0(0)	0(0)	0(0)	9(1)	11(0)	11(0)	**16(0)**	20(0)	20(0)	20(0)	20(0)
5	0(0)	0(0)	0(0)	7(1)	9(0)	11(0)	11(0)	16(0)	20(0)	20(0)	20(0)	**23(1)**
6	1(1)	1(1)	1(1)	7(0)	9(0)	11(0)	12(1)	16(0)	20(0)	21(1)	21(1)	**23(0)**

Πίνακας: Παράδειγμα υπολογισμού της βέλτιστης λύσης. Με έντονα γράμματα έχουν σημειωθεί οι θέσεις του πίνακα που περνάει ο αλγόριθμος ώστε να τυπώσει την βέλτιστη λύση, που είναι η επιλογή των αντικειμένων 2, 5 με συνολικό κέρδος 23.