Προηγμένα Θέματα Αλγορίθμων

Διαλέξη 4: Αντισταθμιστική Ανάλυση II

Βασίλειος Νάκος

Τμήμα Πληροφορικής και Τηλεπικοινωνιών
Πανεπιστήμιο Αθηνών

Συντομότερα Μονοπάτια ως Κινητήριο Παράδειγμα.

Αλγόριθμος Dijkstra για Εύρεση Συντομότερων Μονοπατιών

1: Διατήρησε ένα σύνολο S, αρχικοποιημένο στο $\{s\}$
2: Κράτησε πίνακα d για κάθε $u \in V$
3: $d_s \leftarrow 0$ και $d_u \leftarrow \infty, \forall u \in V \setminus \{s\}$ (εισαγωγή κόμβου)
4: $d_u \leftarrow w_{(s,v)}, \forall (s,v) \in E$
5: Όσο $t \notin S$
6: $\quad u^\star := \text{argmin}_{u \in \bar{S}} d_u$ (εξαγωγή ελαχίστου)
7: $\quad S \leftarrow S \cup \{u^\star\}$
8: $\quad$ Για κάθε v γείτονα του $u^\star$
9: $\quad\quad d_v \leftarrow \min\{d_v, d_{u^\star} + w_{(u^\star,v)}\}$ (μείωση κλειδιού)
10: Επέστρεψε το d_t

Έστω $t_{\text{in}}, t_{\text{min}}, t_{\text{decKey}}$ τα κόστη να γίνουν οι αντίστοιχες πράξεις.

Συντομότερα Μονοπάτια ως Κινητήριο Παράδειγμα.

Αλγόριθμος Dijkstra για Εύρεση Συντομότερων Μονοπατιών

1: Διατήρησε ένα σύνολο S, αρχικοποιημένο στο $\{s\}$
2: Κράτησε πίνακα d για κάθε $u \in V$
3: $d_s \leftarrow 0$ και $d_u \leftarrow \infty, \forall u \in V \setminus \{s\}$ (εισαγωγή κόμβου)
4: $d_u \leftarrow w_{(s,v)}, \forall (s,v) \in E$
5: Όσο $t \notin S$
6: $\quad u^\star := \text{argmin}_{u \in \bar{S}} d_u$ (εξαγωγή ελαχίστου)
7: $\quad S \leftarrow S \cup \{u^\star\}$
8: $\quad$ Για κάθε v γείτονα του $u^\star$
9: $\quad\quad d_v \leftarrow \min\{d_v, d_{u^\star} + w_{(u^\star,v)}\}$ (μείωση κλειδιού)
10: Επέστρεψε το d_t

Έστω $t_{\text{in}}, t_{\text{min}}, t_{\text{decKey}}$ τα κόστη να γίνουν οι αντίστοιχες πράξεις.

Χρόνος εκτέλεσης: $O\left(n \cdot t_{\text{min}} + n \cdot t_{\text{in}} + m \cdot t_{\text{decKey}}\right)$

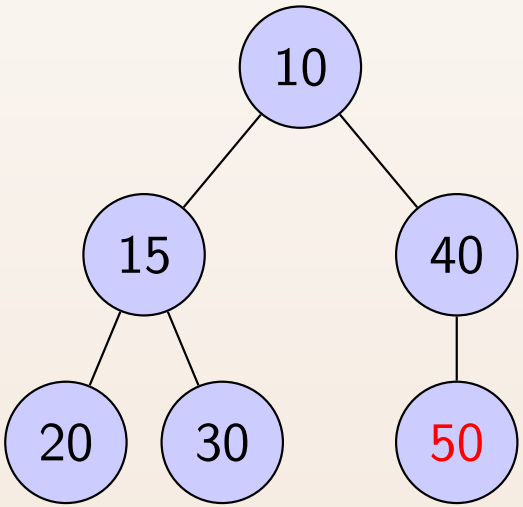
Δυαδικός Σωρός.

★ Σε κάθε υποδέντρο η τιμή της ρίζας είναι μικρότερη ή ίση από τις τιμές των παιδιών.

Δυαδικός Σωρός.

★ Σε κάθε υποδέντρο η τιμή της ρίζας είναι μικρότερη ή ίση από τις τιμές των παιδιών.

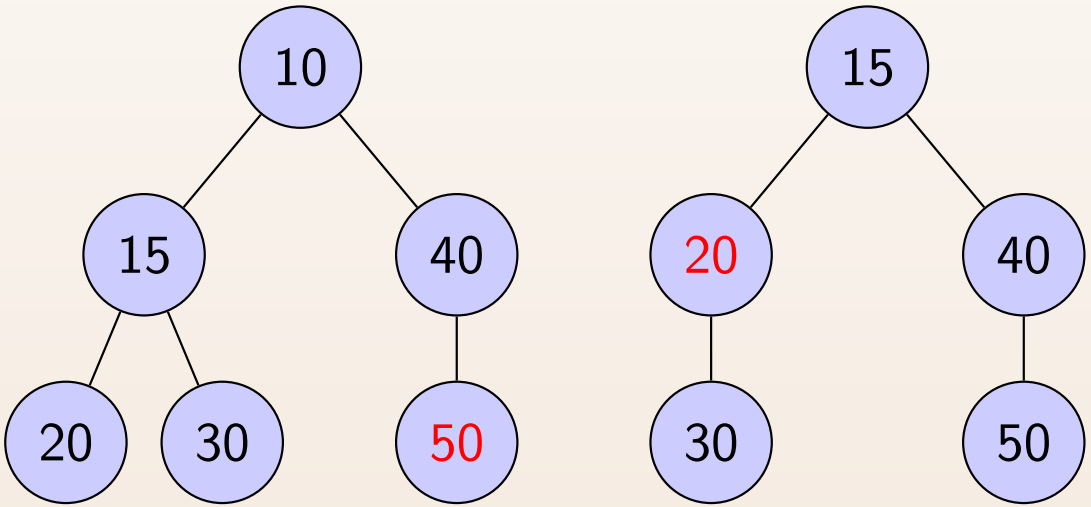
Στο παρακάτω παράδειγμα έχουμε εξαγωγή ελαχίστου και μετά μείωση κλειδιού του κόκκινου κόμβου.



Δυαδικός Σωρός.

★ Σε κάθε υποδέντρο η τιμή της ρίζας είναι μικρότερη ή ίση από τις τιμές των παιδιών.

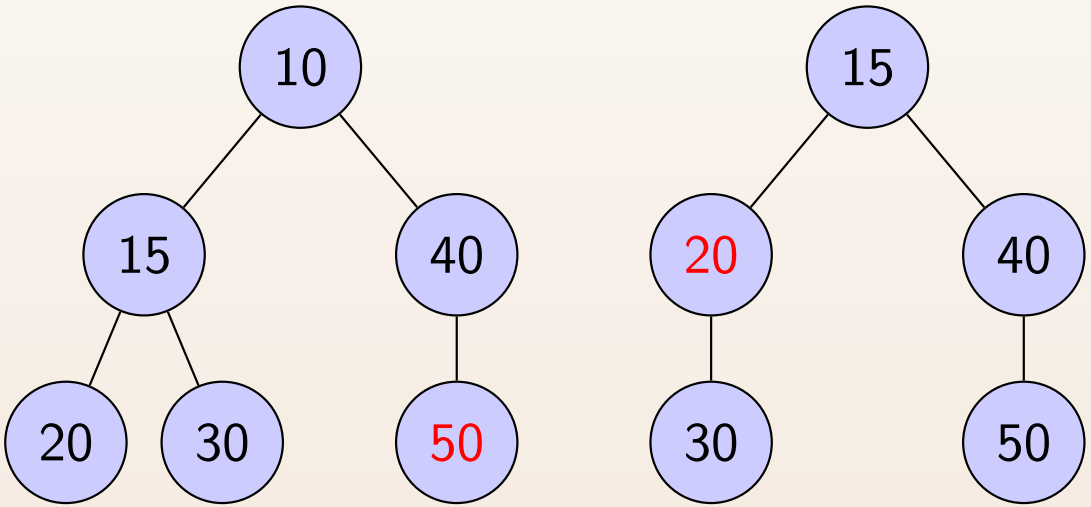
Στο παρακάτω παράδειγμα έχουμε εξαγωγή ελαχίστου και μετά μείωση κλειδιού του κόκκινου κόμβου.



Δυαδικός Σωρός.

★ Σε κάθε υποδέντρο η τιμή της ρίζας είναι μικρότερη ή ίση από τις τιμές των παιδιών.

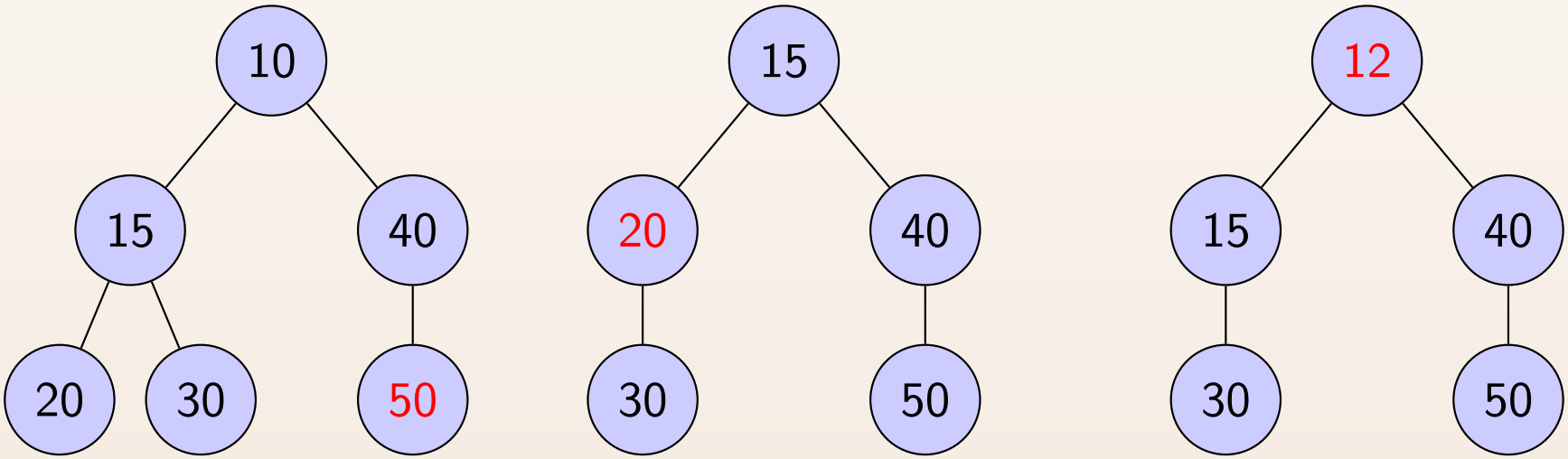
Στο παρακάτω παράδειγμα έχουμε εξαγωγή ελαχίστου και μετά μείωση κλειδιού του κόκκινου κόμβου.



Δυαδικός Σωρός.

★ Σε κάθε υποδέντρο η τιμή της ρίζας είναι μικρότερη ή ίση από τις τιμές των παιδιών.

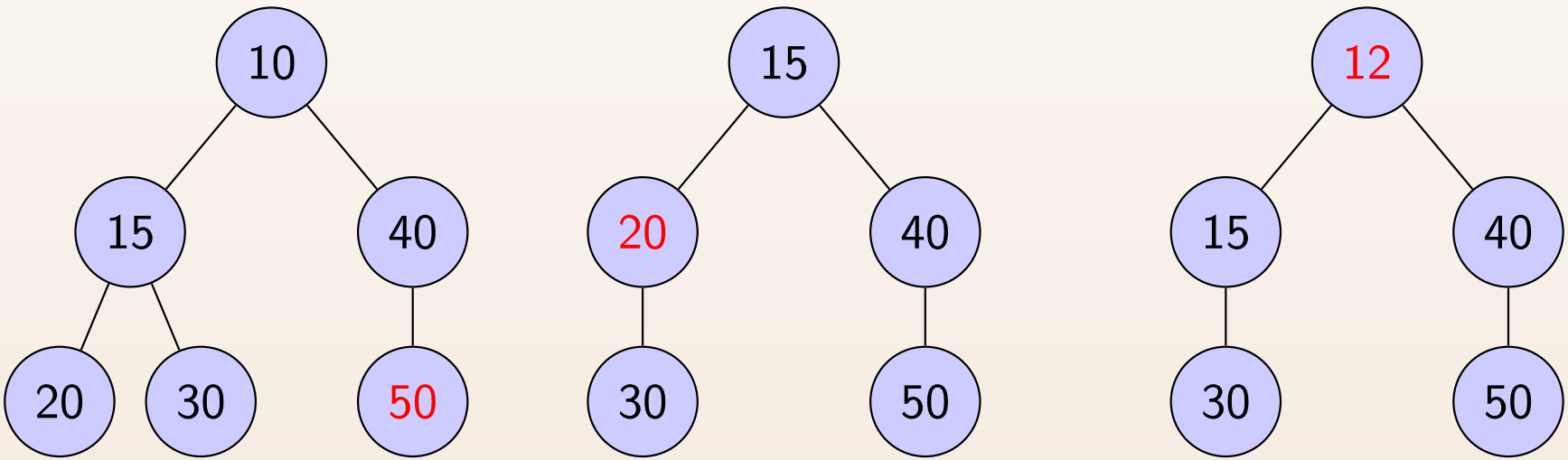
Στο παρακάτω παράδειγμα έχουμε εξαγωγή ελαχίστου και μετά μείωση κλειδιού του κόκκινου κόμβου.



Δυαδικός Σωρός.

★ Σε κάθε υποδέντρο η τιμή της ρίζας είναι μικρότερη ή ίση από τις τιμές των παιδιών.

Στο παρακάτω παράδειγμα έχουμε εξαγωγή ελαχίστου και μετά μείωση κλειδιού του κόκκινου κόμβου.

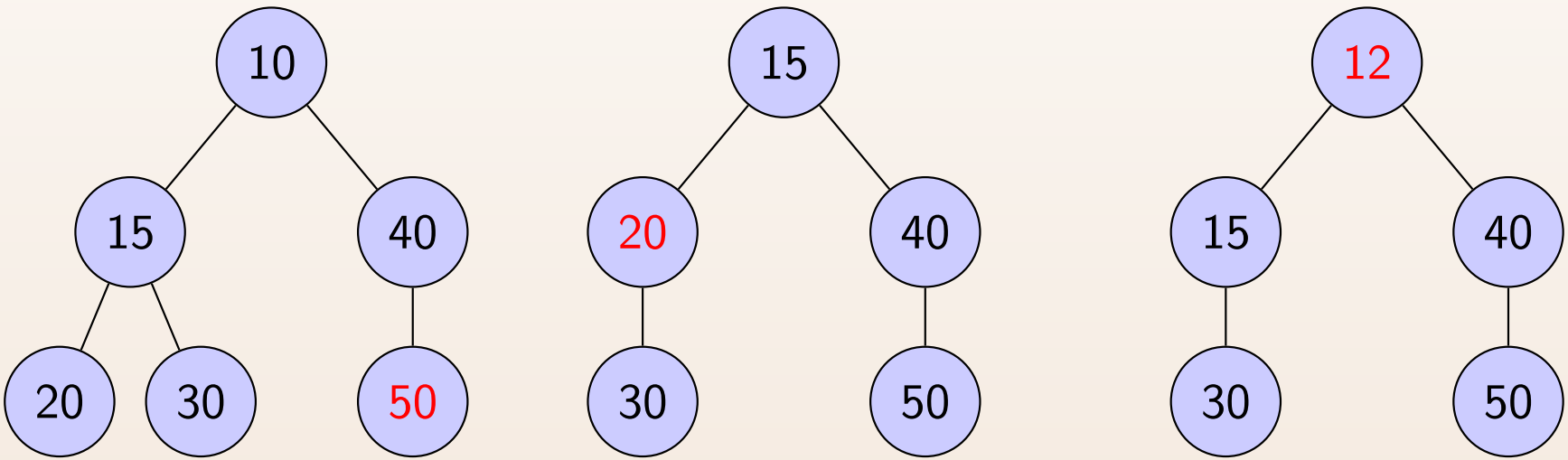


$$t_{\text{in}}, t_{\text{min}}, t_{\text{decKey}} = O(\log n)$$

Δυαδικός Σωρός.

★ Σε κάθε υποδέντρο η τιμή της ρίζας είναι μικρότερη ή ίση από τις τιμές των παιδιών.

Στο παρακάτω παράδειγμα έχουμε εξαγωγή ελαχίστου και μετά μείωση κλειδιού του κόκκινου κόμβου.



$$t_{\text{in}}, t_{\text{min}}, t_{\text{decKey}} = O(\log n)$$

Διωνυμικός Σωρός.

Ένας διωνυμικός σωρός θα είναι μια συλλογή από **διωνυμικά δέντρα**.

Διωνυμικός Σωρός.

Ένας διωνυμικός σωρός θα είναι μια συλλογή από **διωνυμικά δέντρα**.

★ Ένα διωνυμικό k-δέντρο (εναλλακτικά, βαθμού k) θα αποτελείται από μία ρίζα r η οποία θα έχει σαν παιδιά διωνυμικά δέντρου βαθμού $k-1, k-2, \ldots, 0$.

Διωνυμικός Σωρός.

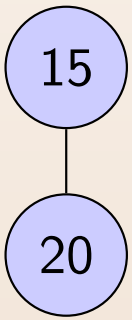
Ένας διωνυμικός σωρός θα είναι μια συλλογή από **διωνυμικά δέντρα**.

★ Ένα διωνυμικό k-δέντρο (εναλλακτικά, βαθμού k) θα αποτελείται από μία ρίζα r η οποία θα έχει σαν παιδιά διωνυμικά δέντρου βαθμού $k-1, k-2, \ldots, 0$. Ένα διωνυμικό δέντρο μεγέθους 0 θα είναι ένας κόμβος μόνος του.

Διωνυμικός Σωρός.

Ένας διωνυμικός σωρός θα είναι μια συλλογή από **διωνυμικά δέντρα**.

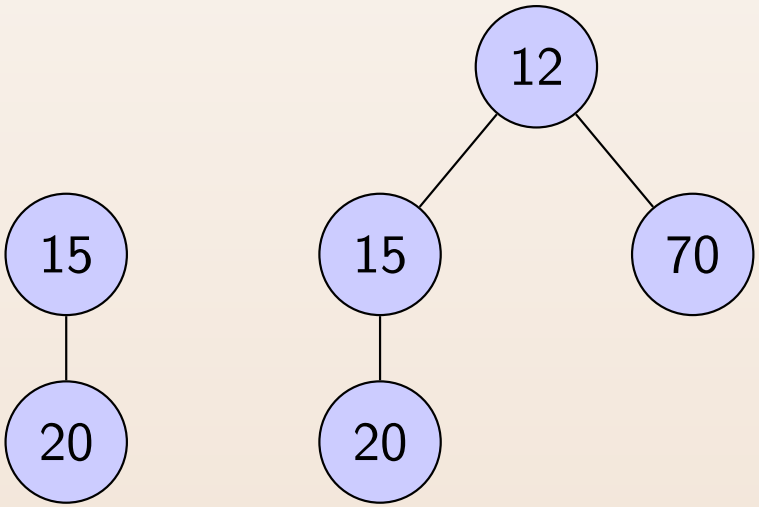
★ Ένα διωνυμικό k-δέντρο (εναλλακτικά, βαθμού k) θα αποτελείται από μία ρίζα r η οποία θα έχει σαν παιδιά διωνυμικά δέντρου βαθμού $k-1, k-2, \ldots, 0$. Ένα διωνυμικό δέντρο μεγέθους 0 θα είναι ένας κόμβος μόνος του. Κάθε κόμβος έχει πάντα τιμή μικρότερη ή ίση από τα παιδιά του.



Διωνυμικός Σωρός.

Ένας διωνυμικός σωρός θα είναι μια συλλογή από **διωνυμικά δέντρα**.

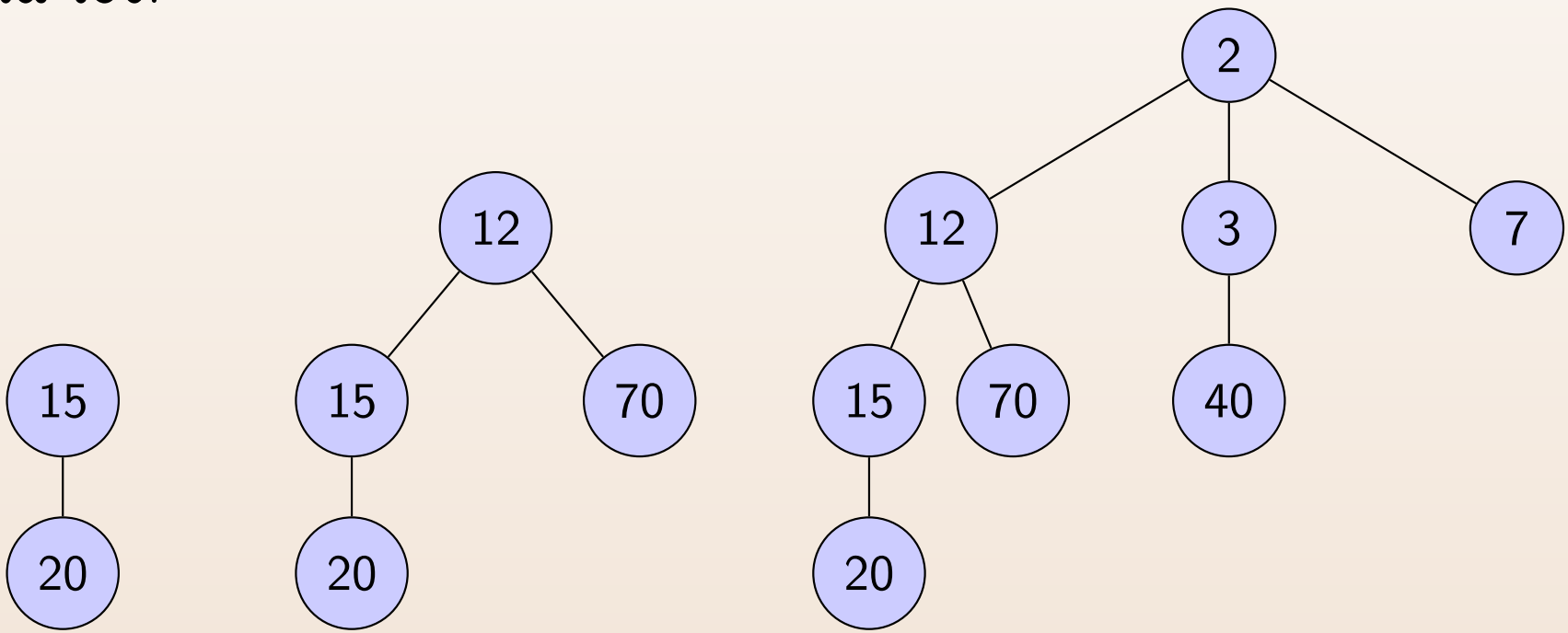
★ Ένα διωνυμικό k-δέντρο (εναλλακτικά, βαθμού k) θα αποτελείται από μία ρίζα r η οποία θα έχει σαν παιδιά διωνυμικά δέντρου βαθμού $k-1, k-2, \ldots, 0$. Ένα διωνυμικό δέντρο μεγέθους 0 θα είναι ένας κόμβος μόνος του. Κάθε κόμβος έχει πάντα τιμή μικρότερη ή ίση από τα παιδιά του.



Διωνυμικός Σωρός.

Ένας διωνυμικός σωρός θα είναι μια συλλογή από **διωνυμικά δέντρα**.

★ Ένα διωνυμικό k-δέντρο (εναλλακτικά, βαθμού k) θα αποτελείται από μία ρίζα r η οποία θα έχει σαν παιδιά διωνυμικά δέντρου βαθμού $k-1, k-2, \ldots, 0$. Ένα διωνυμικό δέντρο μεγέθους 0 θα είναι ένας κόμβος μόνος του. Κάθε κόμβος έχει πάντα τιμή μικρότερη ή ίση από τα παιδιά του.



Διωνυμικός σωρός.

Ισοδύναμος ορισμός: Ένα k-δέντρο μπορεί να προκύψει παίρνοντας δύο $(k-1)$-δέντρα και ενώνοντας τις ρίζες τους (αυτό με τη μικρότερη ρίζα θα γίνει ο πατέρας).

Διωνυμικός σωρός.

Ισοδύναμος ορισμός: Ένα k-δέντρο μπορεί να προκύψει παίρνοντας δύο $(k-1)$-δέντρα και ενώνοντας τις ρίζες τους (αυτό με τη μικρότερη ρίζα θα γίνει ο πατέρας).

★ Διωνυμικός σωρός: θα κρατάμε πάντα μια συλλογή από διωνυμικά δέντρα σε μία λίστα, με την αναλλοίωτη ότι δεν υπάρχουν δύο δέντρα με τον ίδιο βαθμό.

Διωνυμικός σωρός.

Ισοδύναμος ορισμός: Ένα k-δέντρο μπορεί να προκύψει παίρνοντας δύο $(k-1)$-δέντρα και ενώνοντας τις ρίζες τους (αυτό με τη μικρότερη ρίζα θα γίνει ο πατέρας).

★ Διωνυμικός σωρός: θα κρατάμε πάντα μια συλλογή από διωνυμικά δέντρα σε μία λίστα, με την αναλλοίωτη ότι δεν υπάρχουν δύο δέντρα με τον ίδιο βαθμό.

👁 Ένα δέντρο βαθμού k έχει πάντα 2^k κόμβους.

Διωνυμικός σωρός.

Ισοδύναμος ορισμός: Ένα k-δέντρο μπορεί να προκύψει παίρνοντας δύο $(k-1)$-δέντρα και ενώνοντας τις ρίζες τους (αυτό με τη μικρότερη ρίζα θα γίνει ο πατέρας).

★ Διωνυμικός σωρός: θα κρατάμε πάντα μια συλλογή από διωνυμικά δέντρα σε μία λίστα, με την αναλλοίωτη ότι δεν υπάρχουν δύο δέντρα με τον ίδιο βαθμό.

👁 Ένα δέντρο βαθμού k έχει πάντα 2^k κόμβους.
Λόγω επαγωγής: $2^{k-1} + 2^{k-1} = 2^k$.

Διωνυμικός σωρός.

Ισοδύναμος ορισμός: Ένα k-δέντρο μπορεί να προκύψει παίρνοντας δύο $(k-1)$-δέντρα και ενώνοντας τις ρίζες τους (αυτό με τη μικρότερη ρίζα θα γίνει ο πατέρας).

★ Διωνυμικός σωρός: θα κρατάμε πάντα μια συλλογή από διωνυμικά δέντρα σε μία λίστα, με την αναλλοίωτη ότι δεν υπάρχουν δύο δέντρα με τον ίδιο βαθμό.

👁 Ένα δέντρο βαθμού k έχει πάντα 2^k κόμβους.
Λόγω επαγωγής: $2^{k-1} + 2^{k-1} = 2^k$.
👁 Υπάρχουν το πολύ $\log n$ δέντρα στη λίστα, το καθένα βάθους $\lceil \log n \rceil$.

$$L := [T_0, T_1, \ldots, T_{\lceil \log n \rceil}]$$

Υλοποίηση πράξεων.

$\star$ Θα ορίσουμε την πράξη συγχώνευσης η οποία συγχωνεύει δύο k-δέντρα σε ένα $(k+1)$-δέντρο σε χρόνο $O(1)$

Υλοποίηση πράξεων.

$\star$ Θα ορίσουμε την πράξη συγχώνευσης η οποία συγχωνεύει δύο k-δέντρα σε ένα $(k+1)$-δέντρο σε χρόνο $O(1)$

$\star$ Εισαγωγή στοιχείου x: Φτιάχνουμε ένα νέο δέντρο βαθμού μηδέν με ρίζα το x και το βάζουμε στη λίστα.

Υλοποίηση πράξεων.

$\star$ Θα ορίσουμε την πράξη συγχώνευσης η οποία συγχωνεύει δύο k-δέντρα σε ένα $(k+1)$-δέντρο σε χρόνο $O(1)$

$\star$ Εισαγωγή στοιχείου x: Φτιάχνουμε ένα νέο δέντρο βαθμού μηδέν με ρίζα το x και το βάζουμε στη λίστα. Αν υπάρχει ήδη 0-δέντρο, τα συγχωνεύουμε και παίρνουμε ένα 0-δέντρο.

Υλοποίηση πράξεων.

$\star$ Θα ορίσουμε την πράξη συγχώνευσης η οποία συγχωνεύει δύο k-δέντρα σε ένα $(k+1)$-δέντρο σε χρόνο $O(1)$

$\star$ Εισαγωγή στοιχείου x: Φτιάχνουμε ένα νέο δέντρο βαθμού μηδέν με ρίζα το x και το βάζουμε στη λίστα. Αν υπάρχει ήδη 0-δέντρο, τα συγχωνεύουμε και παίρνουμε ένα 0-δέντρο. Αν υπάρχει και άλλο 1-δέντρο βαθμού 1 πρέπει συνεχίζουμε τη συγχώνευση, μετά πάμε στα 2-δέντρο και ούτω καθεξής.

Υλοποίηση πράξεων.

$\star$ Θα ορίσουμε την πράξη συγχώνευσης η οποία συγχωνεύει δύο k-δέντρα σε ένα $(k+1)$-δέντρο σε χρόνο $O(1)$

$\star$ Εισαγωγή στοιχείου x: Φτιάχνουμε ένα νέο δέντρο βαθμού μηδέν με ρίζα το x και το βάζουμε στη λίστα. Αν υπάρχει ήδη 0-δέντρο, τα συγχωνεύουμε και παίρνουμε ένα 0-δέντρο. Αν υπάρχει και άλλο 1-δέντρο βαθμού 1 πρέπει συνεχίζουμε τη συγχώνευση, μετά πάμε στα 2-δέντρο και ούτω καθεξής.

$\star$ Εξαγωγή ελαχίστου: Διατρέχουμε τη λίστα, βρίσκουμε το δέντρο με την ελάχιστη ρίζα και αφαιρούμε τη ρίζα.

Υλοποίηση πράξεων.

$\star$ Θα ορίσουμε την πράξη συγχώνευσης η οποία συγχωνεύει δύο k-δέντρα σε ένα $(k+1)$-δέντρο σε χρόνο $O(1)$

$\star$ Εισαγωγή στοιχείου x: Φτιάχνουμε ένα νέο δέντρο βαθμού μηδέν με ρίζα το x και το βάζουμε στη λίστα. Αν υπάρχει ήδη 0-δέντρο, τα συγχωνεύουμε και παίρνουμε ένα 0-δέντρο. Αν υπάρχει και άλλο 1-δέντρο βαθμού 1 πρέπει συνεχίζουμε τη συγχώνευση, μετά πάμε στα 2-δέντρο και ούτω καθεξής.

$\star$ Εξαγωγή ελαχίστου: Διατρέχουμε τη λίστα, βρίσκουμε το δέντρο με την ελάχιστη ρίζα και αφαιρούμε τη ρίζα. Αν έχει βαθμό k θα μείνουμε με k άλλα δέντρα.

Υλοποίηση πράξεων.

$\star$ Θα ορίσουμε την πράξη συγχώνευσης η οποία συγχωνεύει δύο k-δέντρα σε ένα $(k+1)$-δέντρο σε χρόνο $O(1)$

$\star$ Εισαγωγή στοιχείου x: Φτιάχνουμε ένα νέο δέντρο βαθμού μηδέν με ρίζα το x και το βάζουμε στη λίστα. Αν υπάρχει ήδη 0-δέντρο, τα συγχωνεύουμε και παίρνουμε ένα 0-δέντρο. Αν υπάρχει και άλλο 1-δέντρο βαθμού 1 πρέπει συνεχίζουμε τη συγχώνευση, μετά πάμε στα 2-δέντρο και ούτω καθεξής.

$\star$ Εξαγωγή ελαχίστου: Διατρέχουμε τη λίστα, βρίσκουμε το δέντρο με την ελάχιστη ρίζα και αφαιρούμε τη ρίζα. Αν έχει βαθμό k θα μείνουμε με k άλλα δέντρα. Για να διατηρήσουμε την αναλλοίωτη, κάνουμε όπως και πριν διαδοχική συγχώνευση, από τα μικρότερα προς τα μεγαλύτερα.

Υλοποίηση πράξεων.

$\star$ Θα ορίσουμε την πράξη συγχώνευσης η οποία συγχωνεύει δύο k-δέντρα σε ένα $(k+1)$-δέντρο σε χρόνο $O(1)$

$\star$ Εισαγωγή στοιχείου x: Φτιάχνουμε ένα νέο δέντρο βαθμού μηδέν με ρίζα το x και το βάζουμε στη λίστα. Αν υπάρχει ήδη 0-δέντρο, τα συγχωνεύουμε και παίρνουμε ένα 0-δέντρο. Αν υπάρχει και άλλο 1-δέντρο βαθμού 1 πρέπει συνεχίζουμε τη συγχώνευση, μετά πάμε στα 2-δέντρο και ούτω καθεξής.

$\star$ Εξαγωγή ελαχίστου: Διατρέχουμε τη λίστα, βρίσκουμε το δέντρο με την ελάχιστη ρίζα και αφαιρούμε τη ρίζα. Αν έχει βαθμό k θα μείνουμε με k άλλα δέντρα. Για να διατηρήσουμε την αναλλοίωτη, κάνουμε όπως και πριν διαδοχική συγχώνευση, από τα μικρότερα προς τα μεγαλύτερα.

👁 Ο χρόνος είναι $O(\log n)$ επειδή είναι ακριβώς σαν να προσθέτουμε δύο δυαδικούς αριθμούς.

Αντισταθμιστική Ανάλυση σε Διωνυμικούς Σωρούς.

Η παραπάνω ανάλυση δίνει $t_{\text{in}}, t_{\text{min}}, t_{\text{decKey}} = O(\log n)$.

Αντισταθμιστική Ανάλυση σε Διωνυμικούς Σωρούς.

Η παραπαπάνω ανάλυση δίνει $t_{\text{in}}, t_{\text{min}}, t_{\text{decKey}} = O(\log n)$.

Ας ορίσουμε τις πράξεις λίγο διαφορετικά, χωρίς να αλλάξει κάτι επί της ουσίας:

1. Συγχώνευση: ενώνουμε δύο k-δέντρα σε ένα $(k+1)$-δέντρο.
2. Εισαγωγή: Βάζουμε έναν κόμβο στο σωρό **χωρίς συγχώνευση**.
3. Εξαγωγή Ελαχίστου: Αφαιρούμε το ελάχιστο **χωρίς συγχώνευση**.

Αντισταθμιστική Ανάλυση σε Διωνυμικούς Σωρούς.

Η παραπαπάνω ανάλυση δίνει $t_{\text{in}}, t_{\text{min}}, t_{\text{decKey}} = O(\log n)$.

Ας ορίσουμε τις πράξεις λίγο διαφορετικά, χωρίς να αλλάξει κάτι επί της ουσίας:

1. Συγχώνευση: ενώνουμε δύο k-δέντρα σε ένα $(k+1)$-δέντρο.
2. Εισαγωγή: Βάζουμε έναν κόμβο στο σωρό **χωρίς συγχώνευση**.
3. Εξαγωγή Ελαχίστου: Αφαιρούμε το ελάχιστο **χωρίς συγχώνευση**.

★ Για να διατηρείται η σημασιολογία του σωρού πρέπει μετά από κάθε Εισαγωγή και κάθε Εξαγωγή Ελαχίστου να κάνουμε συγχωνεύσεις μέχρι να ισχύει ξανά η αναλλοίωτη.

Αντισταθμιστική Ανάλυση σε Διωνυμικούς Σωρούς.

Η παραπάνω ανάλυση δίνει $t_{\text{in}}, t_{\text{min}}, t_{\text{decKey}} = O(\log n)$.

Ας ορίσουμε τις πράξεις λίγο διαφορετικά, χωρίς να αλλάξει κάτι επί της ουσίας:

1. Συγχώνευση: ενώνουμε δύο k-δέντρα σε ένα $(k+1)$-δέντρο.
2. Εισαγωγή: Βάζουμε έναν κόμβο στο σωρό **χωρίς συγχώνευση**.
3. Εξαγωγή Ελαχίστου: Αφαιρούμε το ελάχιστο **χωρίς συγχώνευση**.

★ Για να διατηρείται η σημασιολογία του σωρού πρέπει μετά από κάθε Εισαγωγή και κάθε Εξαγωγή Ελαχίστου να κάνουμε συγχωνεύσεις μέχρι να ισχύει ξανά η αναλλοίωτη.

☛ Θα δείξουμε ότι πλέον ο συνολικός χρόνος για i εισαγωγές και e εξαγωγές ελαχίστου (με διατήρηση της αναλλοίωτης) είναι $O(i + e \log n)$.

Χρεωστικό σχήμα.

👍 Κάθε εισαγωγή και κάθε εξαγωγή ελαχίστου (χωρίς τις συγχωνεύσεις) παίρνει χρόνο $O(1)$ και $O(\log n)$ αντίστοιχα.

Χρεωστικό σχήμα.

👍 Κάθε εισαγωγή και κάθε εξαγωγή ελαχίστου (χωρίς τις συγχωνεύσεις) παίρνει χρόνο $O(1)$ και $O(\log n)$ αντίστοιχα.

Αλλά πως θα χρεώσω τις συγχωνεύσεις;

1. Για κάθε εισαγωγή στοιχείου πληρώνουμε $O(1)$ και δίνουμε μία δραχμή στο δέντρο που δημιουργείται.

Χρεωστικό σχήμα.

👍 Κάθε εισαγωγή και κάθε εξαγωγή ελαχίστου (χωρίς τις συγχωνεύσεις) παίρνει χρόνο $O(1)$ και $O(\log n)$ αντίστοιχα.

Αλλά πως θα χρεώσω τις συγχωνεύσεις;

1. Για κάθε εισαγωγή στοιχείου πληρώνουμε $O(1)$ και δίνουμε μία δραχμή στο δέντρο που δημιουργείται.
2. Για κάθε εξαγωγή ελαχίστου πληρώνουμε $O(\log n)$ και δίνουμε μία δραχμή σε καθένα από τα δέντρα που δημιουργούνται.

Χρεωστικό σχήμα.

👍 Κάθε εισαγωγή και κάθε εξαγωγή ελαχίστου (χωρίς τις συγχωνεύσεις) παίρνει χρόνο $O(1)$ και $O(\log n)$ αντίστοιχα.

Αλλά πως θα χρεώσω τις συγχωνεύσεις;

1. Για κάθε εισαγωγή στοιχείου πληρώνουμε $O(1)$ και δίνουμε μία δραχμή στο δέντρο που δημιουργείται.
2. Για κάθε εξαγωγή ελαχίστου πληρώνουμε $O(\log n)$ και δίνουμε μία δραχμή σε καθένα από τα δέντρα που δημιουργούνται.
3. Όταν δύο k-δέντρα συγχωνεύονται, έχουμε δύο δραχμές:

Χρεωστικό σχήμα.

👍 Κάθε εισαγωγή και κάθε εξαγωγή ελαχίστου (χωρίς τις συγχωνεύσεις) παίρνει χρόνο $O(1)$ και $O(\log n)$ αντίστοιχα.

Αλλά πως θα χρεώσω τις συγχωνεύσεις;

1. Για κάθε εισαγωγή στοιχείου πληρώνουμε $O(1)$ και δίνουμε μία δραχμή στο δέντρο που δημιουργείται.
2. Για κάθε εξαγωγή ελαχίστου πληρώνουμε $O(\log n)$ και δίνουμε μία δραχμή σε καθένα από τα δέντρα που δημιουργούνται.
3. Όταν δύο k-δέντρα συγχωνεύονται, έχουμε δύο δραχμές: χρησιμοποιούμε τη μία να πληρώσουμε τον χρόνο συγχώνευσης

Χρεωστικό σχήμα.

👍 Κάθε εισαγωγή και κάθε εξαγωγή ελαχίστου (χωρίς τις συγχωνεύσεις) παίρνει χρόνο $O(1)$ και $O(\log n)$ αντίστοιχα.

Αλλά πως θα χρεώσω τις συγχωνεύσεις;

1. Για κάθε εισαγωγή στοιχείου πληρώνουμε $O(1)$ και δίνουμε μία δραχμή στο δέντρο που δημιουργείται.
2. Για κάθε εξαγωγή ελαχίστου πληρώνουμε $O(\log n)$ και δίνουμε μία δραχμή σε καθένα από τα δέντρα που δημιουργούνται.
3. Όταν δύο k-δέντρα συγχωνεύονται, έχουμε δύο δραχμές: χρησιμοποιούμε τη μία να πληρώσουμε τον χρόνο συγχώνευσης και την άλλη τη δίνουμε στο $(k+1)$-δέντρο που δημιουργείται!

Χρεωστικό σχήμα.

👍 Κάθε εισαγωγή και κάθε εξαγωγή ελαχίστου (χωρίς τις συγχωνεύσεις) παίρνει χρόνο $O(1)$ και $O(\log n)$ αντίστοιχα.

Αλλά πως θα χρεώσω τις συγχωνεύσεις;

1. Για κάθε εισαγωγή στοιχείου πληρώνουμε $O(1)$ και δίνουμε μία δραχμή στο δέντρο που δημιουργείται.
2. Για κάθε εξαγωγή ελαχίστου πληρώνουμε $O(\log n)$ και δίνουμε μία δραχμή σε καθένα από τα δέντρα που δημιουργούνται.
3. Όταν δύο k-δέντρα συγχωνεύονται, έχουμε δύο δραχμές: χρησιμοποιούμε τη μία να πληρώσουμε τον χρόνο συγχώνευσης και την άλλη τη δίνουμε στο $(k+1)$-δέντρο που δημιουργείται!

Συνολικός χρόνος σε συγχωνεύσεις $= O(\#\text{ δραχμών}) = O(i + e \log n)$ ✓

Σωροί Fibonacci.

⋆ Δυαδικός σωρός: $t_{\text{in}}, t_{\text{min}}, t_{\text{decKey}} = O(\log n)$ (χειρότερη περίπτωση)

⋆ Διωνυμικός σωρός: $t_{\text{in}} = O(1)$, $t_{\text{min}}, t_{\text{decKey}} = O(\log n)$ (αντισταθμιστικά)

Σωροί Fibonacci.

⋆ Δυαδικός σωρός: $t_{\text{in}}, t_{\min}, t_{\text{decKey}} = O(\log n)$ (χειρότερη περίπτωση)

⋆ Διωνυμικός σωρός: $t_{\text{in}} = O(1)$, $t_{\min}, t_{\text{decKey}} = O(\log n)$ (αντισταθμιστικά)

👁 Για τον αλγόριθμο του Dijkstra θέλουμε όμως ιδανικά $t_{\text{decKey}} = O(1)$ αντισταθμιστικό χρόνο $\Rightarrow$ σωροί Fibonacci.

Σωροί Fibonacci.

⋆ Δυαδικός σωρός: $t_{\text{in}}, t_{\min}, t_{\text{decKey}} = O(\log n)$ (χειρότερη περίπτωση)

⋆ Διωνυμικός σωρός: $t_{\text{in}} = O(1)$, $t_{\min}, t_{\text{decKey}} = O(\log n)$ (αντισταθμιστικά)

👁 Για τον αλγόριθμο του Dijkstra θέλουμε όμως ιδανικά $t_{\text{decKey}} = O(1)$ αντισταθμιστικό χρόνο $\Rightarrow$ σωροί Fibonacci.

Βελτιώνουν τους Διωνυμικούς σωρούς πετυχαίνοντας $t_{\text{decKey}} = O(1)$.

Σωροί Fibonacci.

⋆ Δυαδικός σωρός: $t_{\text{in}}, t_{\min}, t_{\text{decKey}} = O(\log n)$ (χειρότερη περίπτωση)

⋆ Διωνυμικός σωρός: $t_{\text{in}} = O(1)$, $t_{\min}, t_{\text{decKey}} = O(\log n)$ (αντισταθμιστικά)

👁 Για τον αλγόριθμο του Dijkstra θέλουμε όμως ιδανικά $t_{\text{decKey}} = O(1)$ αντισταθμιστικό χρόνο $\Rightarrow$ σωροί Fibonacci.

Βελτιώνουν τους Διωνυμικούς σωρούς πετυχαίνοντας $t_{\text{decKey}} = O(1)$.

Συνήθως πιο αργοί στην πράξη από τους δυαδικούς σωρούς,

Σωροί Fibonacci.

⋆ Δυαδικός σωρός: $t_{\text{in}}, t_{\min}, t_{\text{decKey}} = O(\log n)$ (χειρότερη περίπτωση)

⋆ Διωνυμικός σωρός: $t_{\text{in}} = O(1)$, $t_{\min}, t_{\text{decKey}} = O(\log n)$ (αντισταθμιστικά)

👁 Για τον αλγόριθμο του Dijkstra θέλουμε όμως ιδανικά $t_{\text{decKey}} = O(1)$ αντισταθμιστικό χρόνο $\Rightarrow$ σωροί Fibonacci.

Βελτιώνουν τους Διωνυμικούς σωρούς πετυχαίνοντας $t_{\text{decKey}} = O(1)$.

Συνήθως πιο αργοί στην πράξη από τους δυαδικούς σωρούς, αλλά εξαιρετικό παράδειγμα αντισταθμιστικής ανάλυσης.

Σωροί Fibonacci.

- Στους διωνυμικούς σωρούς, η διατήρηση της αναλλοίωτης μετά την εισαγωγή μοιάζει μη αποδοτική

Σωροί Fibonacci.

- Στους διωνυμικούς σωρούς, η διατήρηση της αναλλοίωτης μετά την εισαγωγή μοιάζει μη αποδοτική καθώς χρειαζόμαστε την αναλλοίωτη να ισχύει μόνο για την εξαγωγή ελαχίστου.

Σωροί Fibonacci.

👁 Στους διωνυμικούς σωρούς, η διατήρηση της αναλλοίωτης μετά την εισαγωγή μοιάζει μη αποδοτική καθώς χρειαζόμαστε την αναλλοίωτη να ισχύει μόνο για την εξαγωγή ελαχίστου.

★ Θα είμαστε λίγο *οκνηροί* στην *εισαγωγή*

Σωροί Fibonacci.

👁 Στους διωνυμικούς σωρούς, η διατήρηση της αναλλοίωτης μετά την εισαγωγή μοιάζει μη αποδοτική καθώς χρειαζόμαστε την αναλλοίωτη να ισχύει μόνο για την εξαγωγή ελαχίστου.

★ Θα είμαστε λίγο *οκνηροί* στην *εισαγωγή*: θέλουμε πάλι ένα δέντρο *βαθμού* k να έχει εκθετικό πλήθος κόμβων, αλλά του επιτρέπουμε να χάσει μερικούς κόμβους.

Σωροί Fibonacci.

👁 Στους διωνυμικούς σωρούς, η διατήρηση της αναλλοίωτης μετά την εισαγωγή μοιάζει μη αποδοτική καθώς χρειαζόμαστε την αναλλοίωτη να ισχύει μόνο για την εξαγωγή ελαχίστου.

★ Θα είμαστε λίγο *οκνηροί* στην *εισαγωγή*: θέλουμε πάλι ένα δέντρο *βαθμού* k να έχει εκθετικό πλήθος κόμβων, αλλά του επιτρέπουμε να χάσει μερικούς κόμβους.

👉Ας διατηρήσουμε την αναλλοίωτη **μόνο** μετά την εξαγωγή ελαχίστου.

Σωροί Fibonacci.

👁 Στους διωνυμικούς σωρούς, η διατήρηση της αναλλοίωτης μετά την εισαγωγή μοιάζει μη αποδοτική καθώς χρειαζόμαστε την αναλλοίωτη να ισχύει μόνο για την εξαγωγή ελαχίστου.

★ Θα είμαστε λίγο *οκνηροί* στην *εισαγωγή*: θέλουμε πάλι ένα δέντρο *βαθμού* k να έχει εκθετικό πλήθος κόμβων, αλλά του επιτρέπουμε να χάσει μερικούς κόμβους.

👉Ας διατηρήσουμε την αναλλοίωτη **μόνο** μετά την εξαγωγή ελαχίστου.

Και η μείωση κλειδιού;

Σωροί Fibonacci.

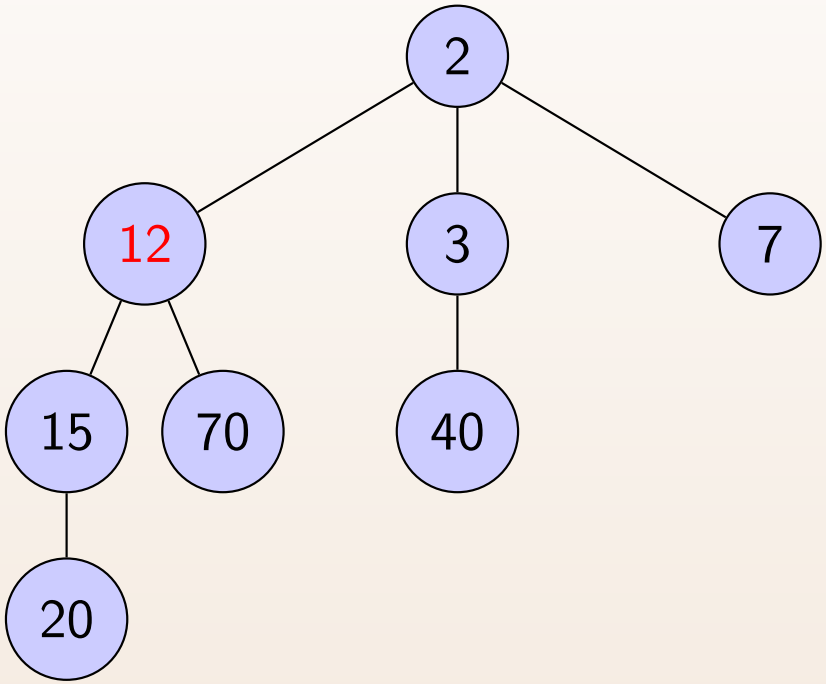
👁 Στους διωνυμικούς σωρούς, η διατήρηση της αναλλοίωτης μετά την εισαγωγή μοιάζει μη αποδοτική καθώς χρειαζόμαστε την αναλλοίωτη να ισχύει μόνο για την εξαγωγή ελαχίστου.

★ Θα είμαστε λίγο *οκνηροί* στην *εισαγωγή*: θέλουμε πάλι ένα δέντρο *βαθμού* k να έχει εκθετικό πλήθος κόμβων, αλλά του επιτρέπουμε να χάσει μερικούς κόμβους.

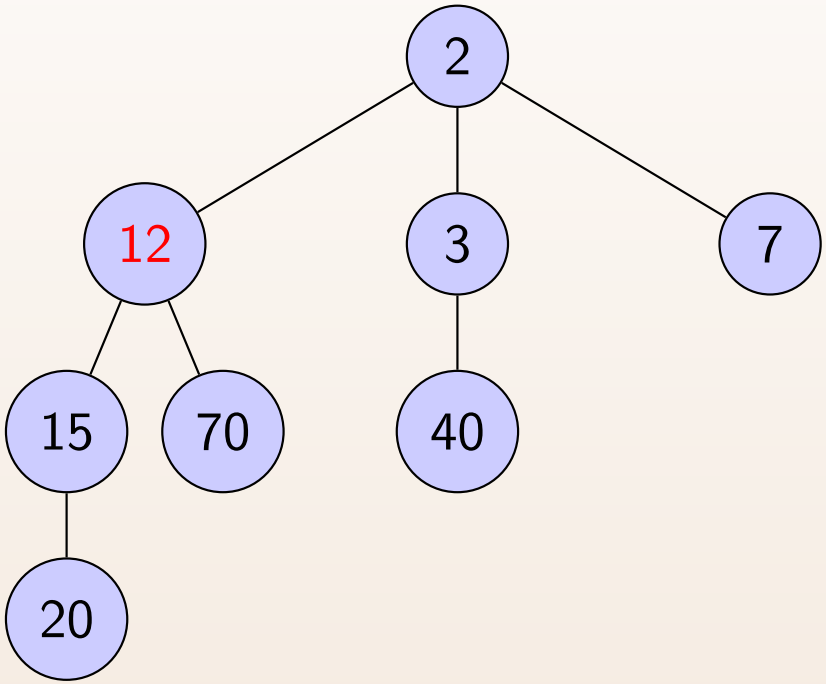
👉Ας διατηρήσουμε την αναλλοίωτη **μόνο** μετά την εξαγωγή ελαχίστου.

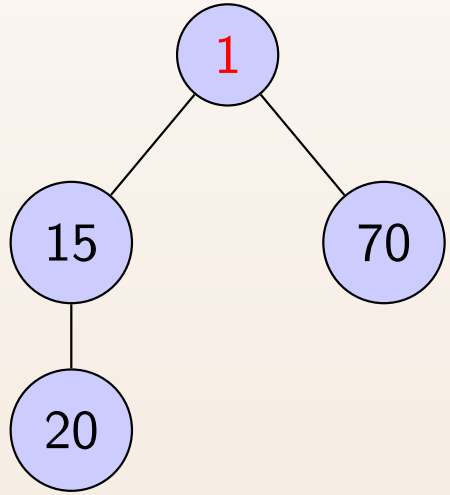
Και η μείωση κλειδιού; Κάθε φορά που μειώνουμε το κλειδί του x βγάζουμε το x και το υπόδεντρό του σε ένα ξεχωριστό δέντρο.

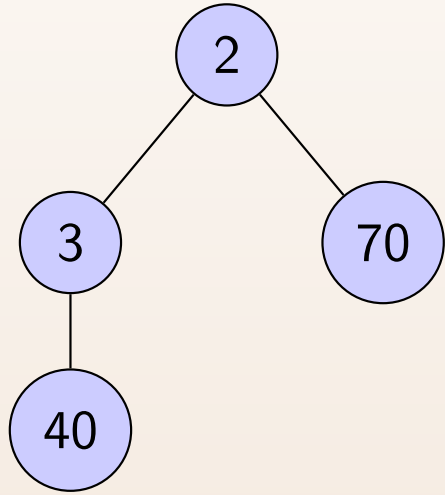
Παράδειγμα μείωσης κλειδιού.



Παράδειγμα μείωσης κλειδιού.







Βελτίωση μείωσης κλειδιού.

Θα κρατήσουμε για κάθε στοιχείο x μία μεταβλητή $\mathrm{mark}(x)$

Βελτίωση μείωσης κλειδιού.

Θα κρατήσουμε για κάθε στοιχείο x μία μεταβλητή $\mathrm{mark}(x)$:

1. x δεν έχει χάσει κανένα παιδί $\Rightarrow \mathrm{mark}(x) = 0$.

Βελτίωση μείωσης κλειδιού.

Θα κρατήσουμε για κάθε στοιχείο x μία μεταβλητή $\mathrm{mark}(x)$:

1. x δεν έχει χάσει κανένα παιδί $\Rightarrow \mathrm{mark}(x) = 0$.
2. x έχει χάσει ένα παιδί $\Rightarrow \mathrm{mark}(x) = 1$.

Βελτίωση μείωσης κλειδιού.

Θα κρατήσουμε για κάθε στοιχείο x μία μεταβλητή $\mathrm{mark}(x)$:

1. x δεν έχει χάσει κανένα παιδί $\Rightarrow \mathrm{mark}(x) = 0$.
2. x έχει χάσει ένα παιδί $\Rightarrow \mathrm{mark}(x) = 1$.
3. Αν το κλειδί του x μειωθεί, τότε κοιτάμε αν ο πατέρας του, έστω y, ικανοποιεί $\mathrm{mark}(y) = 1$,

Βελτίωση μείωσης κλειδιού.

Θα κρατήσουμε για κάθε στοιχείο x μία μεταβλητή $\mathrm{mark}(x)$:

1. x δεν έχει χάσει κανένα παιδί $\Rightarrow \mathrm{mark}(x) = 0$.
2. x έχει χάσει ένα παιδί $\Rightarrow \mathrm{mark}(x) = 1$.
3. Αν το κλειδί του x μειωθεί, τότε κοιτάμε αν ο πατέρας του, έστω y, ικανοποιεί $\mathrm{mark}(y) = 1$, και αν ναι θα αποκόψουμε και το y σε ένα δέντρο μόνο του (και κάνουμε $\mathrm{mark}(y) = 1$).

Βελτίωση μείωσης κλειδιού.

Θα κρατήσουμε για κάθε στοιχείο x μία μεταβλητή $\mathrm{mark}(x)$:

1. x δεν έχει χάσει κανένα παιδί $\Rightarrow \mathrm{mark}(x) = 0$.
2. x έχει χάσει ένα παιδί $\Rightarrow \mathrm{mark}(x) = 1$.
3. Αν το κλειδί του x μειωθεί, τότε κοιτάμε αν ο πατέρας του, έστω y, ικανοποιεί $\mathrm{mark}(y) = 1$, και αν ναι θα αποκόψουμε και το y σε ένα δέντρο μόνο του (και κάνουμε $\mathrm{mark}(y) = 1$).

👉Μία μείωση κλειδιού ενδέχεται να οδηγήσει σε αλυσιδωτή αποκοπή.

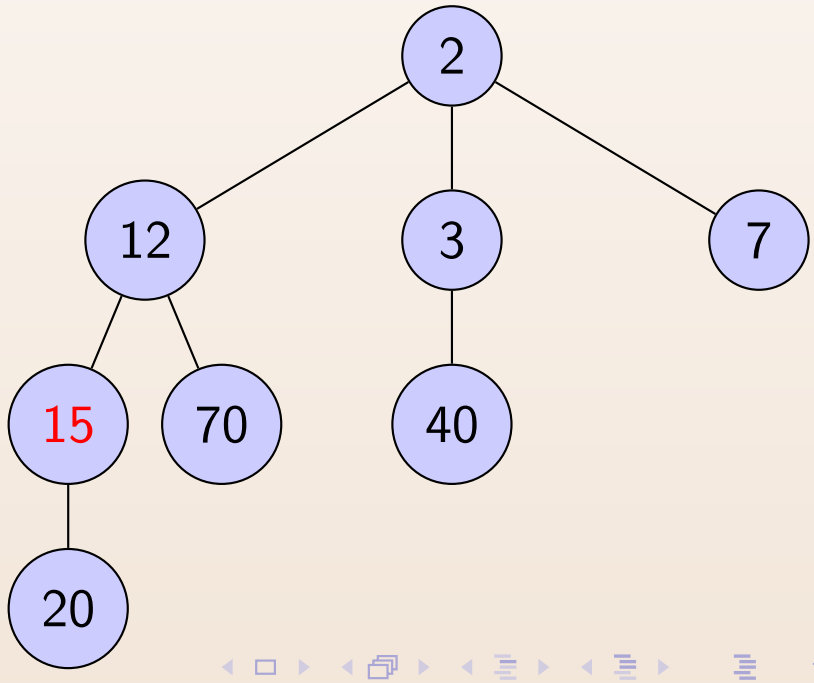
Βελτίωση μείωσης κλειδιού.

Θα κρατήσουμε για κάθε στοιχείο x μία μεταβλητή $\mathrm{mark}(x)$:

1. x δεν έχει χάσει κανένα παιδί $\Rightarrow \mathrm{mark}(x) = 0$.
2. x έχει χάσει ένα παιδί $\Rightarrow \mathrm{mark}(x) = 1$.
3. Αν το κλειδί του x μειωθεί, τότε κοιτάμε αν ο πατέρας του, έστω y, ικανοποιεί $\mathrm{mark}(y) = 1$, και αν ναι θα αποκόψουμε και το y σε ένα δέντρο μόνο του (και κάνουμε $\mathrm{mark}(y) = 1$).

☛Μία μείωση κλειδιού ενδέχεται να οδηγήσει σε αλυσιδωτή αποκοπή.

Η μείωση κλειδιού του κόμβου με τιμή 15 μπορεί να εξαναγκάσει τον 12 και τον 2 να μείνουν μόνοι τους σε ένα δέντρο, αφήνοντας έτσι 5 δέντρα.



Ανάλυση δομής.

Λήμμα

Ένα δέντρο βαθμού k έχει τουλάχιστον $F_k \geq \sqrt{2}^k$ κόμβους, όπου F_k είναι ο k-οστός αριθμός Fibonacci.

Ανάλυση δομής.

Λήμμα

Ένα δέντρο βαθμού k έχει τουλάχιστον $F_k \geq \sqrt{2}^k$ κόμβους, όπου F_k είναι ο k-οστός αριθμός Fibonacci.

Απόδειξη με επαγωγή: Σε ένα k-δέντρο το πολύ να έχει φύγει ένα παιδί της ρίζα, στη χειρότερη το $k-1$ δέντρο

$\Rightarrow$

Ανάλυση δομής.

Λήμμα

Ένα δέντρο βαθμού k έχει τουλάχιστον $F_k \geq \sqrt{2}^k$ κόμβους, όπου F_k είναι ο k-οστός αριθμός Fibonacci.

Απόδειξη με επαγωγή: Σε ένα k-δέντρο το πολύ να έχει φύγει ένα παιδί της ρίζα, στη χειρότερη το $k-1$ δέντρο
$\Rightarrow \geq F_{k-2} + F_{k-1} + \ldots + F_0 = F_k$✓

★ Συνάρτηση δυναμικού $\Phi(t) := T(t) + 2 \cdot M(t)$,

Ανάλυση δομής.

Λήμμα

Ένα δέντρο βαθμού k έχει τουλάχιστον $F_k \geq \sqrt{2}^k$ κόμβους, όπου F_k είναι ο k-οστός αριθμός Fibonacci.

Απόδειξη με επαγωγή: Σε ένα k-δέντρο το πολύ να έχει φύγει ένα παιδί της ρίζα, στη χειρότερη το $k-1$ δέντρο
$\Rightarrow \geq F_{k-2} + F_{k-1} + \ldots + F_0 = F_k$✓

★ Συνάρτηση δυναμικού $\Phi(t) := T(t) + 2 \cdot M(t)$, όπου

1. $T(t)$ είνα το πλήθος των τρέχοντων δέντρων στον σωρό,
2. $M(t)$ το πλήθος των x με $\mathrm{mark}(x) = 1$.

Ανάλυση δομής.

⋆ Εισαγωγή: $O(1)$

Ανάλυση δομής.

⋆ Εισαγωγή: $O(1)$

⋆ Ελάττωση κλειδιού: Έστω c το πλήθος αλυσιδωτών αποκοπών.

Ανάλυση δομής.

⋆ Εισαγωγή: $O(1)$

⋆ Ελάττωση κλειδιού: Έστω c το πλήθος αλυσιδωτών αποκοπών.

$$t_{\text{decKey}} = c + \Delta\Phi =$$

Ανάλυση δομής.

⋆ Εισαγωγή: $O(1)$

⋆ Ελάττωση κλειδιού: Έστω c το πλήθος αλυσιδωτών αποκοπών.

$$t_{\text{decKey}} = c + \Delta\Phi = \ = c + (c + 2(1 - c)) = O(1)\checkmark$$

Ανάλυση δομής.

⋆ Εισαγωγή: $O(1)$

⋆ Ελάττωση κλειδιού: Έστω c το πλήθος αλυσιδωτών αποκοπών.

$$t_{\text{decKey}} = c + \Delta\Phi = \ = c + (c + 2(1 - c)) = O(1)\checkmark.$$

⋆ Εξαγωγή ελαχίστου:

$$t_{\min} = O(\log n) + T(t-1) + (T(t) - T(t-1)) = O(\log n)\checkmark,$$

Ανάλυση δομής.

⋆ Εισαγωγή: $O(1)$

⋆ Ελάττωση κλειδιού: Έστω c το πλήθος αλυσιδωτών αποκοπών.

$$t_{\text{decKey}} = c + \Delta\Phi = \ = c + (c + 2(1 - c)) = O(1)\checkmark.$$

⋆ Εξαγωγή ελαχίστου:

$$t_{\min} = O(\log n) + T(t-1) + (T(t) - T(t-1)) = O(\log n)\checkmark,$$

διότι στο τέλος της t-οστής στιγμής γίνεται διατήρηση της αναλλοίωτης και άρα $T(t) = O(\log n)$.

Συνοψίζοντας.

Θεώρημα

Ο σωρός Fibonacci έχει αντισταθμιστικούς χρόνους $t_{\text{decKey}} = O(1), t_{\text{in}} = O(1), t_{\text{min}} = O(\log n)$.

Θεώρημα

Ο αλγόριθμος Dijkstra μπορεί να υλοποιηθεί σε χρόνο $O(m + n \log n)$.