Αλγόριθμοι και Πολυπλοκότητα

Αρχοντία Γιαννοπούλου
Όλγα Φουρτουνέλλη

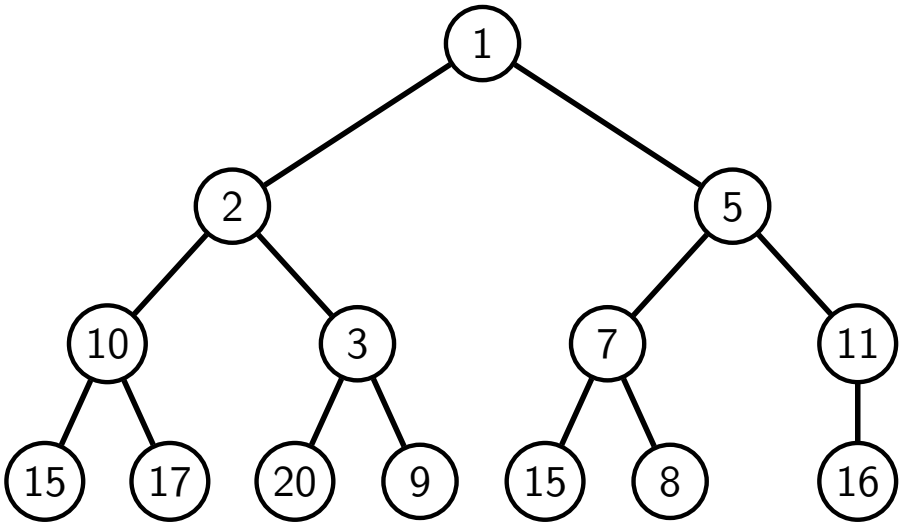
Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

Δομές Δεδομένων

Σωροί
Ξένα Σύνολα

Σωροί (Heaps)

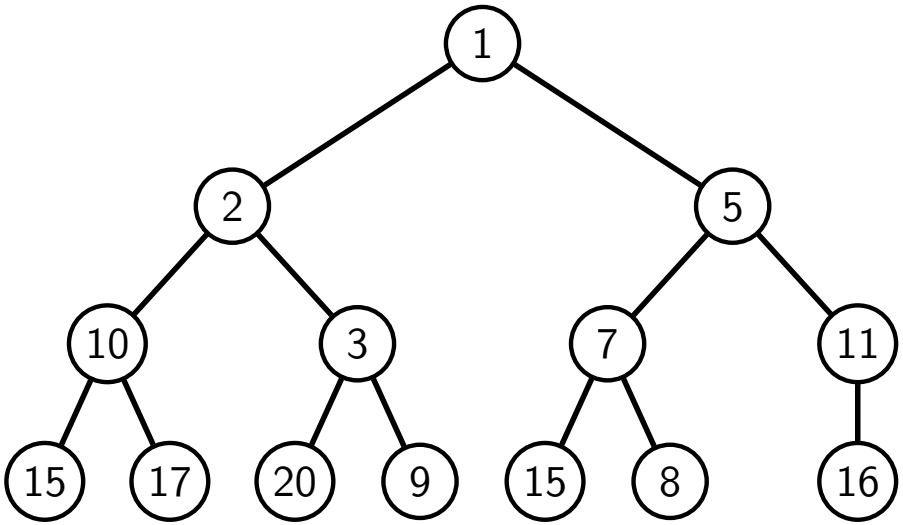
Σωρός: ισοσταθμισμένο δυαδικό δένδρο.



Σωροί (Heaps)

Σωρός: ισοσταθμισμένο δυαδικό δένδρο.

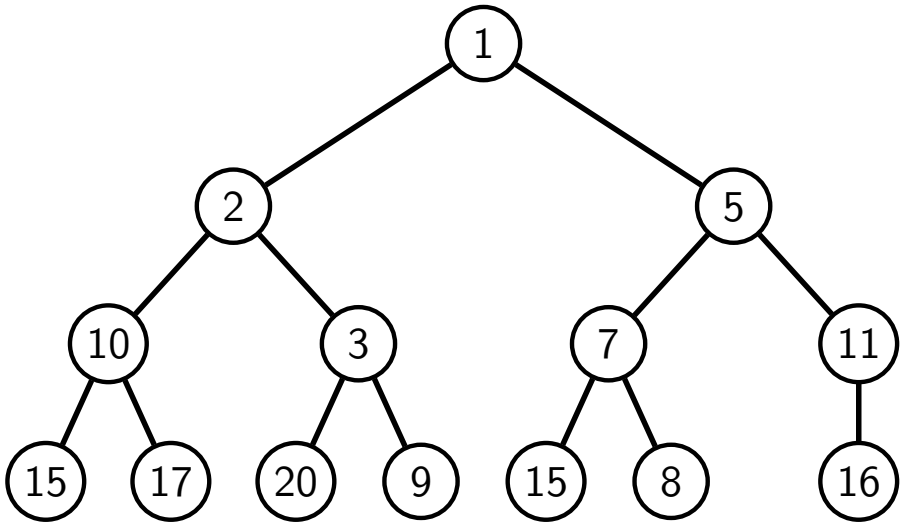
Συνδυάζει πλεονεκτήματα ταξινομημένου πίνακα και λίστας.



Σωροί (Heaps)

Σωρός: ισοσταθμισμένο δυαδικό δένδρο.

Συνδυάζει πλεονεκτήματα ταξινομημένου πίνακα και λίστας.

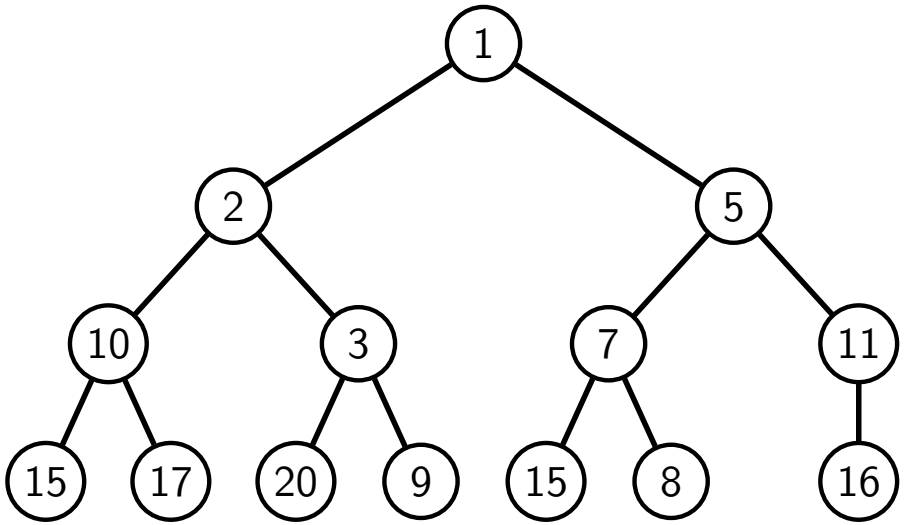


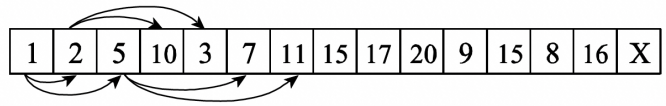
Διάταξη σωρού: για κάθε στοιχείο v, σε μία κορυφή i, το στοιχείο w στο γονέα του i ικανοποιεί τη σχέση: $\text{key}(w) \leq \text{key}(v)$

Σωροί (Heaps)

Σωρός: ισοσταθμισμένο δυαδικό δένδρο.

Συνδυάζει πλεονεκτήματα ταξινομημένου πίνακα και λίστας.





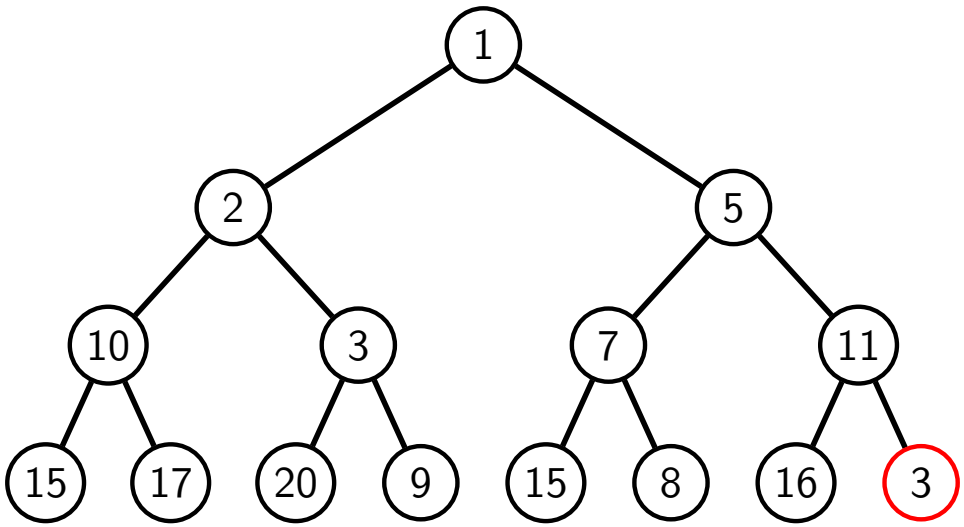
Διάταξη σωρού: για κάθε στοιχείο v, σε μία κορυφή i, το στοιχείο w στο γονέα του i ικανοποιεί τη σχέση: $\text{key}(w) \leq \text{key}(v)$

Για κάθε κορυφή στη θέση i:
$\text{parent}(i) = \lfloor i/2 \rfloor$, $\text{leftChild}(i) = 2i$, $\text{rightChild(i)} = 2i + 1$

Heapify-up

Στόχος: Διόρθωση ενός σχεδόν σωρού με μετακίνηση ενός στοιχείου προς τη ρίζα.

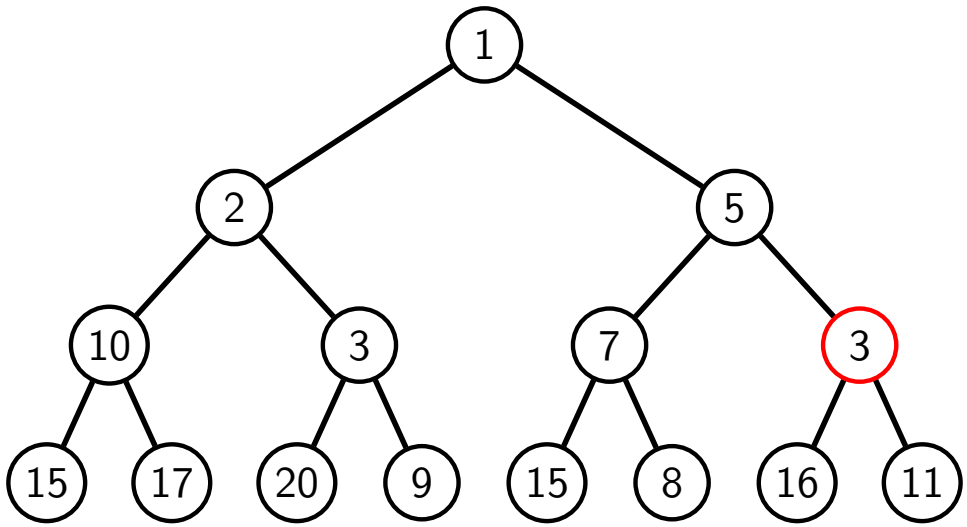
Σχεδόν σωρός με πολύ μικρό κλειδί στη θέση $H[i]$: το στοιχείο v στη θέση i, έχει πολύ μικρό κλειδί $H[i]$, που πιθανόν παραβιάζει την ιδιότητα διάταξης σωρού (και σε καμμία άλλη θέση δεν παραβιάζεται η ιδιότητα διάταξης σωρού).



Heapify-up

Στόχος: Διόρθωση ενός σχεδόν σωρού με μετακίνηση ενός στοιχείου προς τη ρίζα.

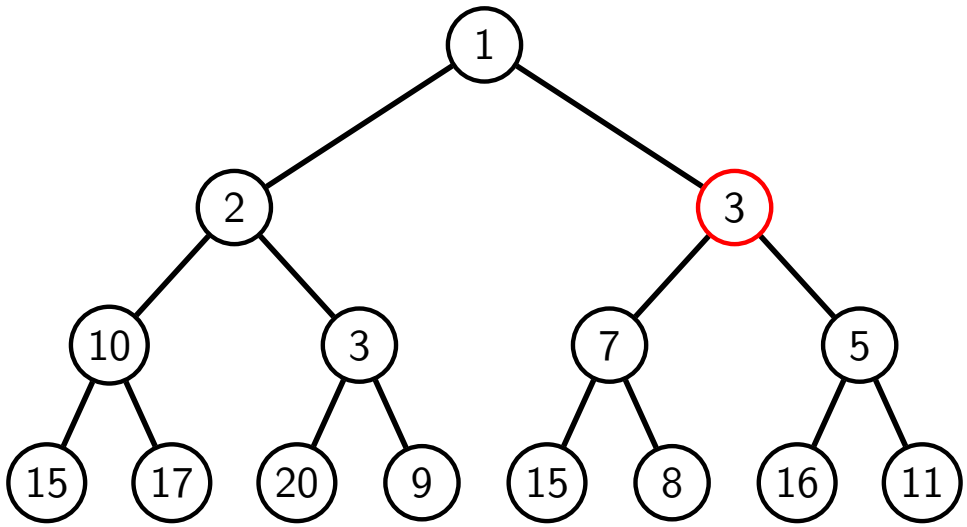
Σχεδόν σωρός με πολύ μικρό κλειδί στη θέση $H[i]$: το στοιχείο v στη θέση i, έχει πολύ μικρό κλειδί $H[i]$, που πιθανόν παραβιάζει την ιδιότητα διάταξης σωρού (και σε καμμία άλλη θέση δεν παραβιάζεται η ιδιότητα διάταξης σωρού).



Heapify-up

Στόχος: Διόρθωση ενός σχεδόν σωρού με μετακίνηση ενός στοιχείου προς τη ρίζα.

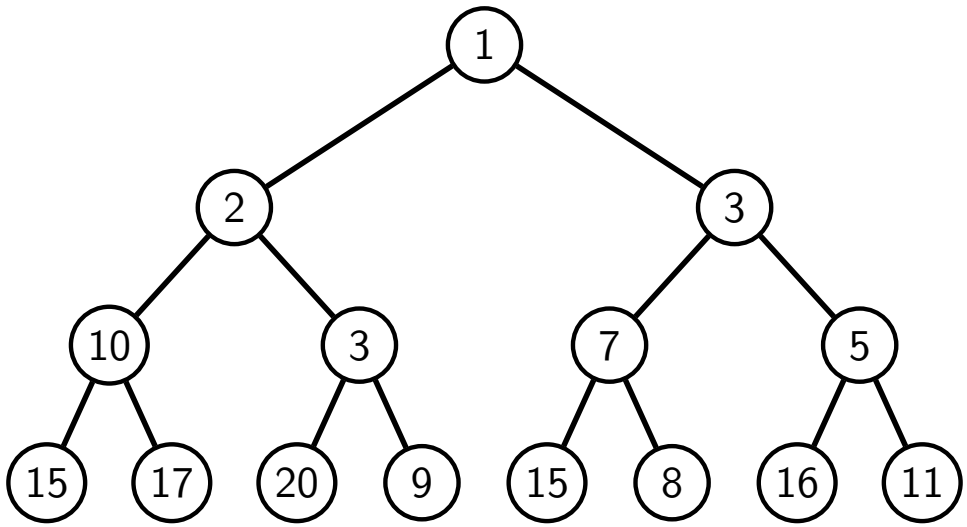
Σχεδόν σωρός με πολύ μικρό κλειδί στη θέση $H[i]$: το στοιχείο v στη θέση i, έχει πολύ μικρό κλειδί $H[i]$, που πιθανόν παραβιάζει την ιδιότητα διάταξης σωρού (και σε καμμία άλλη θέση δεν παραβιάζεται η ιδιότητα διάταξης σωρού).



Heapify-up

Στόχος: Διόρθωση ενός σχεδόν σωρού με μετακίνηση ενός στοιχείου προς τη ρίζα.

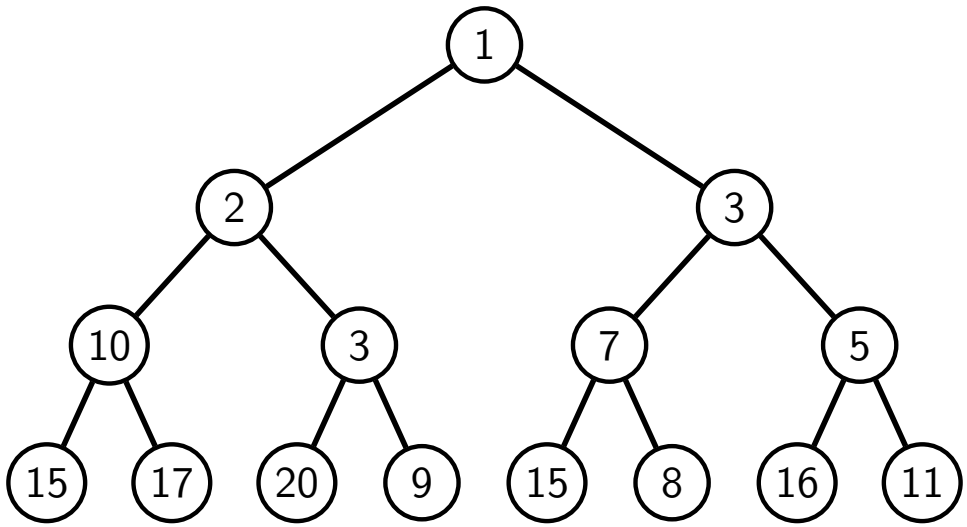
Σχεδόν σωρός με πολύ μικρό κλειδί στη θέση $H[i]$: το στοιχείο v στη θέση i, έχει πολύ μικρό κλειδί $H[i]$, που πιθανόν παραβιάζει την ιδιότητα διάταξης σωρού (και σε καμμία άλλη θέση δεν παραβιάζεται η ιδιότητα διάταξης σωρού).



Heapify-up

Στόχος: Διόρθωση ενός σχεδόν σωρού με μετακίνηση ενός στοιχείου προς τη ρίζα.

Σχεδόν σωρός με πολύ μικρό κλειδί στη θέση $H[i]$: το στοιχείο v στη θέση i, έχει πολύ μικρό κλειδί $H[i]$, που πιθανόν παραβιάζει την ιδιότητα διάταξης σωρού (και σε καμμία άλλη θέση δεν παραβιάζεται η ιδιότητα διάταξης σωρού).



Η διαδικασία heapify-up μετακινεί το στοιχείο προς τη ρίζα.

Η διαδικασία

Heapify-up(H,i):
Εάν $i > 1$ τότε:
  $j = \text{parent}(i) = \lfloor \frac{i}{2} \rfloor$
  Εάν $\text{key}(H[i]) < \text{key}(H[j])$ τότε
    Κάνε εναλλαγή των στοιχείων $H[i]$ και $H[j]$
    Heapify-up(H,j)

Η διαδικασία

Heapify-up(H,i):
Εάν $i > 1$ τότε:
  $j = \text{parent}(i) = \lfloor \frac{i}{2} \rfloor$
  Εάν $\text{key}(H[i]) < \text{key}(H[j])$ τότε
    Κάνε εναλλαγή των στοιχείων $H[i]$ και $H[j]$
    Heapify-up(H,j)

Χρόνος heapify-up: $\mathcal{O}(\log n)$

Η διαδικασία

Heapify-up(H,i):
Εάν $i > 1$ τότε:
  $j = \text{parent}(i) = \lfloor \frac{i}{2} \rfloor$
  Εάν $\text{key}(H[i]) < \text{key}(H[j])$ τότε
    Κάνε εναλλαγή των στοιχείων $H[i]$ και $H[j]$
    Heapify-up(H,j)

Χρόνος heapify-up: $\mathcal{O}(\log n)$

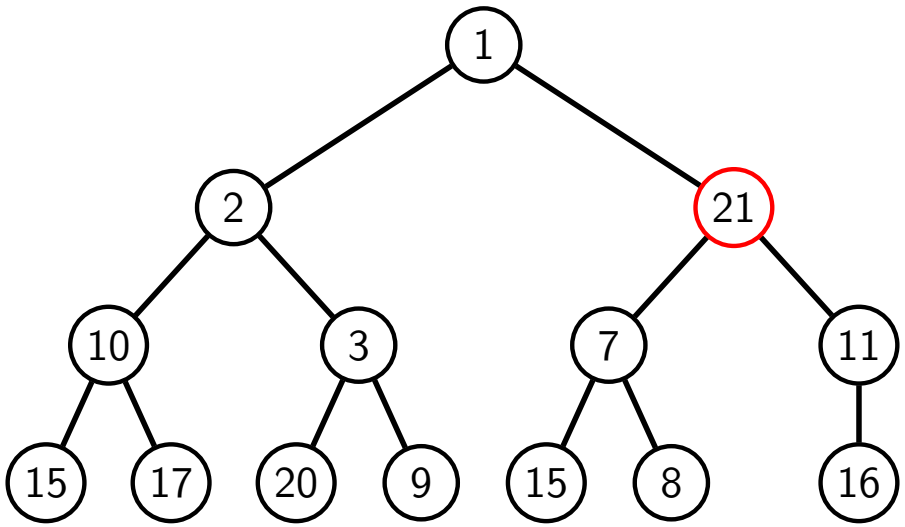
Χρησιμότητα: Αν θέλουμε να προσθέσουμε ένα στοιχείο στο σωρό, μας αρκεί $\mathcal{O}(\log n)$ χρόνος.

Τοποθετούμε το νέο στοιχείο στο τέλος και καλούμε την heapify-up.

Heapify-down

Στόχος: Διόρθωση ενός σχεδόν σωρού με μετακίνηση ενός στοιχείου προς τα φύλλα.

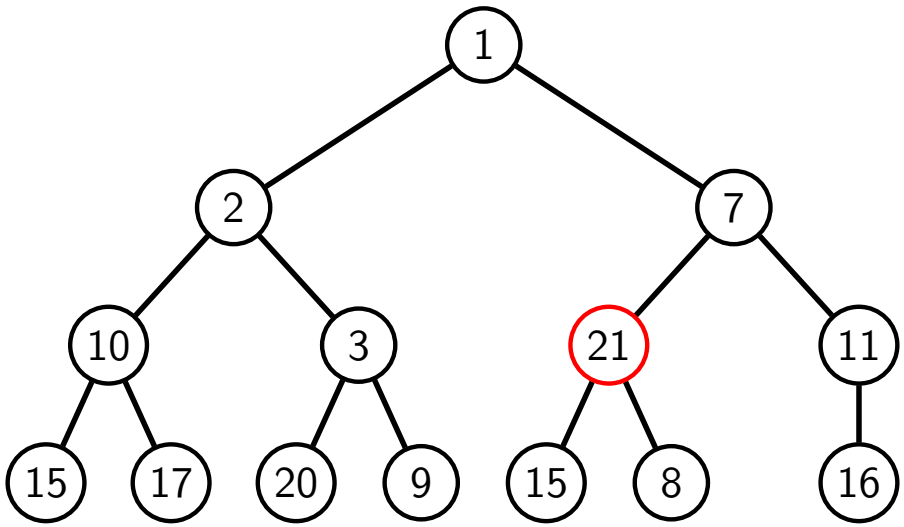
Σχεδόν σωρός με πολύ μεγάλο κλειδί στη θέση $H[i]$: το στοιχείο v στη θέση i, έχει πολύ μεγάλο κλειδί $H[i]$, που πιθανόν παραβιάζει την ιδιότητα διάταξης σωρού (και σε καμμία άλλη θέση δεν παραβιάζεται η ιδιότητα διάταξης σωρού).



Heapify-down

Στόχος: Διόρθωση ενός σχεδόν σωρού με μετακίνηση ενός στοιχείου προς τα φύλλα.

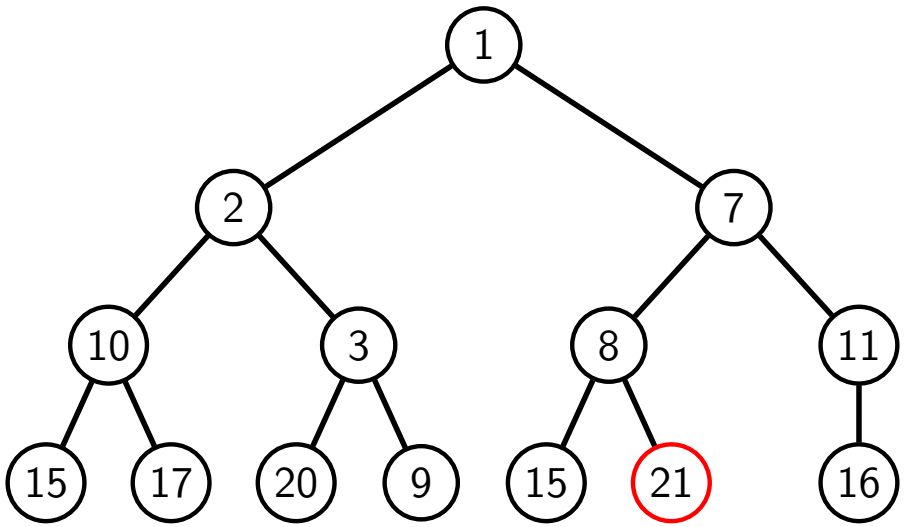
Σχεδόν σωρός με πολύ μεγάλο κλειδί στη θέση $H[i]$: το στοιχείο v στη θέση i, έχει πολύ μεγάλο κλειδί $H[i]$, που πιθανόν παραβιάζει την ιδιότητα διάταξης σωρού (και σε καμμία άλλη θέση δεν παραβιάζεται η ιδιότητα διάταξης σωρού).



Heapify-down

Στόχος: Διόρθωση ενός σχεδόν σωρού με μετακίνηση ενός στοιχείου προς τα φύλλα.

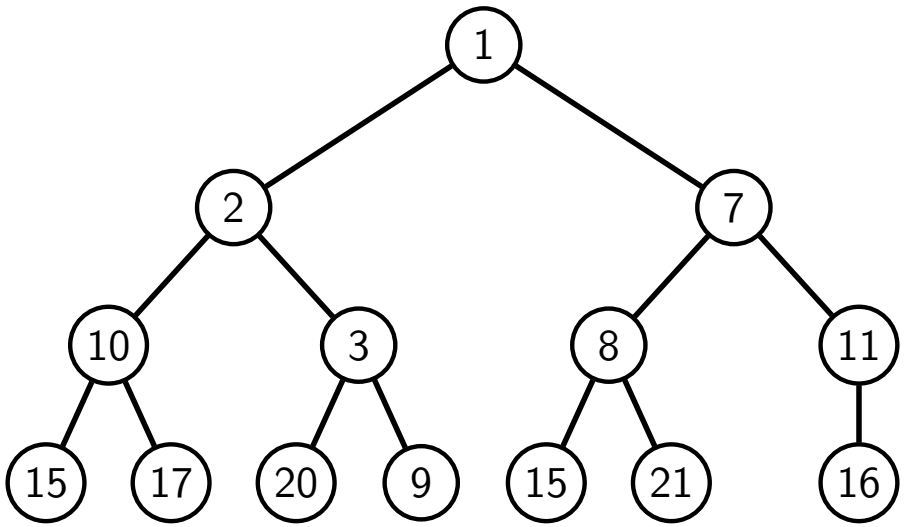
Σχεδόν σωρός με πολύ μεγάλο κλειδί στη θέση $H[i]$: το στοιχείο v στη θέση i, έχει πολύ μεγάλο κλειδί $H[i]$, που πιθανόν παραβιάζει την ιδιότητα διάταξης σωρού (και σε καμμία άλλη θέση δεν παραβιάζεται η ιδιότητα διάταξης σωρού).



Heapify-down

Στόχος: Διόρθωση ενός σχεδόν σωρού με μετακίνηση ενός στοιχείου προς τα φύλλα.

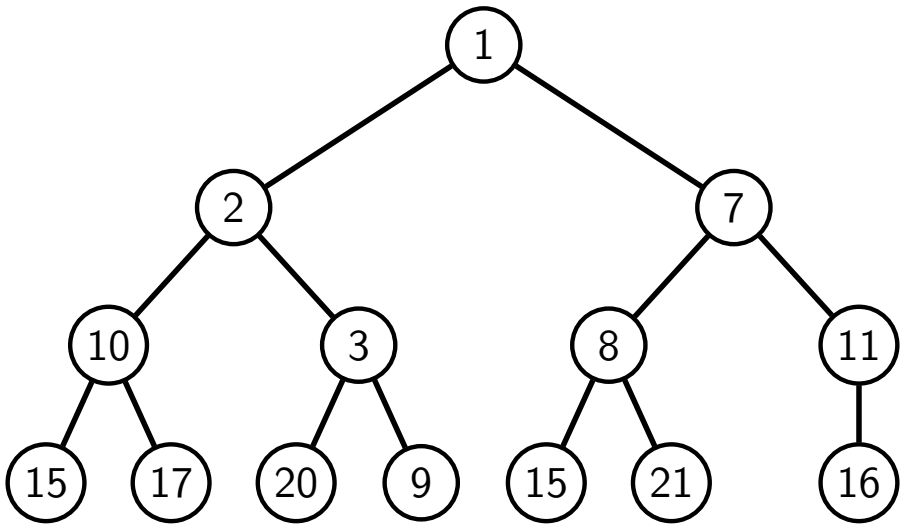
Σχεδόν σωρός με πολύ μεγάλο κλειδί στη θέση $H[i]$: το στοιχείο v στη θέση i, έχει πολύ μεγάλο κλειδί $H[i]$, που πιθανόν παραβιάζει την ιδιότητα διάταξης σωρού (και σε καμμία άλλη θέση δεν παραβιάζεται η ιδιότητα διάταξης σωρού).



Heapify-down

Στόχος: Διόρθωση ενός σχεδόν σωρού με μετακίνηση ενός στοιχείου προς τα φύλλα.

Σχεδόν σωρός με πολύ μεγάλο κλειδί στη θέση $H[i]$: το στοιχείο v στη θέση i, έχει πολύ μεγάλο κλειδί $H[i]$, που πιθανόν παραβιάζει την ιδιότητα διάταξης σωρού (και σε καμμία άλλη θέση δεν παραβιάζεται η ιδιότητα διάταξης σωρού).



Η διαδικασία heapify-down μετακινεί το στοιχείο προς τα φύλλα επιλέγοντας το παιδί με το μικρότερο κλειδί.

Ουρά Προτεραιότητας

Υλοποίηση με σωρό:

- CreateHeap(H): επιστρέφει κενό σωρό H έτοιμο να αποθηκεύσει $\leq n$ στοιχεία. Αρχικοποίηση: $\mathcal{O}(n)$

Ουρά Προτεραιότητας

Υλοποίηση με σωρό:

- CreateHeap(H): επιστρέφει κενό σωρό H έτοιμο να αποθηκεύσει $\leq n$ στοιχεία. Αρχικοποίηση: $\mathcal{O}(n)$
- Insert(H, v): εισάγει το στοιχείο v στον H: $\mathcal{O}(\log n)$ με Heapify-up

Ουρά Προτεραιότητας

Υλοποίηση με σωρό:

- CreateHeap(H): επιστρέφει κενό σωρό H έτοιμο να αποθηκεύσει $\leq n$ στοιχεία. Αρχικοποίηση: $\mathcal{O}(n)$
- Insert(H, v): εισάγει το στοιχείο v στον H: $\mathcal{O}(\log n)$ με Heapify-up
- Delete(H, i): διαγράφει το στοιχείο στη θέση i του H: $\mathcal{O}(\log n)$ με Heapify-down

Ουρά Προτεραιότητας

Υλοποίηση με σωρό:

- CreateHeap(H): επιστρέφει κενό σωρό H έτοιμο να αποθηκεύσει $\leq n$ στοιχεία. Αρχικοποίηση: $\mathcal{O}(n)$
- Insert(H, v): εισάγει το στοιχείο v στον H: $\mathcal{O}(\log n)$ με Heapify-up
- Delete(H, i): διαγράφει το στοιχείο στη θέση i του H: $\mathcal{O}(\log n)$ με Heapify-down
- FindMin(H): εύρεση μικρότερου στοιχείου στον H: $\mathcal{O}(1)$

Ουρά Προτεραιότητας

Υλοποίηση με σωρό:

- CreateHeap(H): επιστρέφει κενό σωρό H έτοιμο να αποθηκεύσει $\leq n$ στοιχεία. Αρχικοποίηση: $\mathcal{O}(n)$
- Insert(H, v): εισάγει το στοιχείο v στον H: $\mathcal{O}(\log n)$ με Heapify-up
- Delete(H, i): διαγράφει το στοιχείο στη θέση i του H: $\mathcal{O}(\log n)$ με Heapify-down
- FindMin(H): εύρεση μικρότερου στοιχείου στον H: $\mathcal{O}(1)$
- ExtractMin(H): εύρεση **και διαγραφή** μικρότερου στοιχείου από τον H: $\mathcal{O}(\log n)$

Εφαρμογή (άσκηση) - Heapsort

Heapsort: ταξινόμηση n στοιχείων με τη χρήση σωρού

Εφαρμογή (άσκηση) - Heapsort

Heapsort: ταξινόμηση n στοιχείων με τη χρήση σωρού

- Αρχικοποιούμε ένα σωρό με τα n στοιχεία.
- Καλούμε n φορές την ExtractMin.
- Συνολικά $\mathcal{O}(n \log n)$ βήματα.

Συντομότερη Διαδρομή

Δεδομένα:

- Κατευθυνόμενο γράφημα $G = (V, A)$.
- Αφετηρία s, προορισμός t.
- $\ell_e > 0$ πραγματικός αριθμός = μήκος της ακμής e.

Συντομότερη Διαδρομή

Δεδομένα:

- Κατευθυνόμενο γράφημα $G = (V, A)$.
- Αφετηρία s, προορισμός t.
- $\ell_e > 0$ πραγματικός αριθμός = μήκος της ακμής e.

Κόστος διαδρομής: άθροισμα μηκών των ακμών της διαδρομής.

Συντομότερη Διαδρομή

Δεδομένα:

- Κατευθυνόμενο γράφημα $G = (V, A)$.
- Αφετηρία s, προορισμός t.
- $\ell_e > 0$ πραγματικός αριθμός = μήκος της ακμής e.

Κόστος διαδρομής: άθροισμα μηκών των ακμών της διαδρομής.

Πρόβλημα συντομότερης διαδρομής: Να βρούμε τη συντομότερη σε μήκος κατευθυνόμενη διαδρομή από την κορυφή s στην κορυφή t.

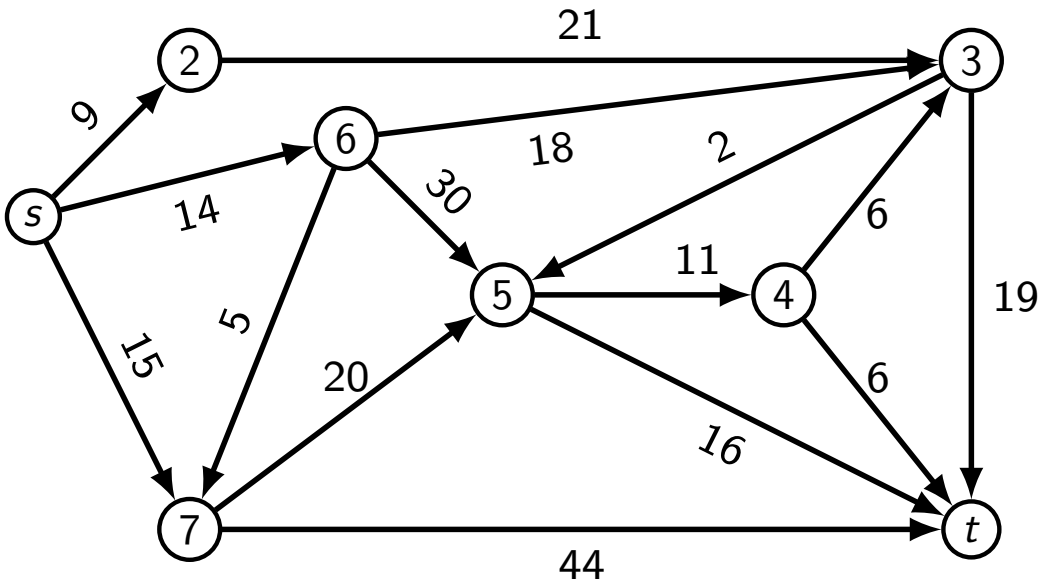
Συντομότερη Διαδρομή

Δεδομένα:

- Κατευθυνόμενο γράφημα $G = (V, A)$.
- Αφετηρία s, προορισμός t.
- $\ell_e > 0$ πραγματικός αριθμός = μήκος της ακμής e.

Κόστος διαδρομής: άθροισμα μηκών των ακμών της διαδρομής.

Πρόβλημα συντομότερης διαδρομής: Να βρούμε τη συντομότερη σε μήκος κατευθυνόμενη διαδρομή από την κορυφή s στην κορυφή t.



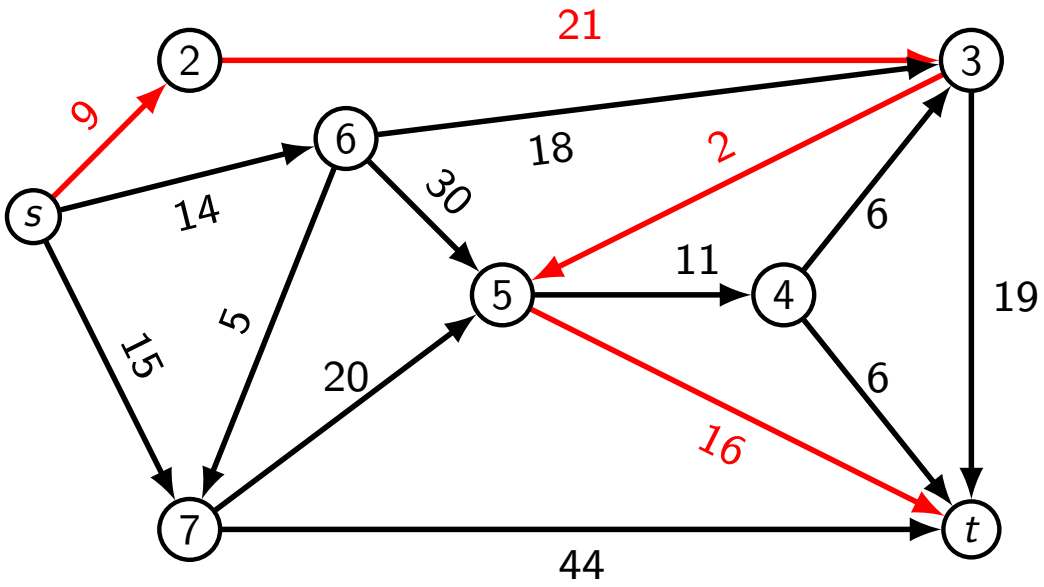
Συντομότερη Διαδρομή

Δεδομένα:

- Κατευθυνόμενο γράφημα $G = (V, A)$.
- Αφετηρία s, προορισμός t.
- $\ell_e > 0$ πραγματικός αριθμός = μήκος της ακμής e.

Κόστος διαδρομής: άθροισμα μηκών των ακμών της διαδρομής.

Πρόβλημα συντομότερης διαδρομής: Να βρούμε τη συντομότερη σε μήκος κατευθυνόμενη διαδρομή από την κορυφή s στην κορυφή t.



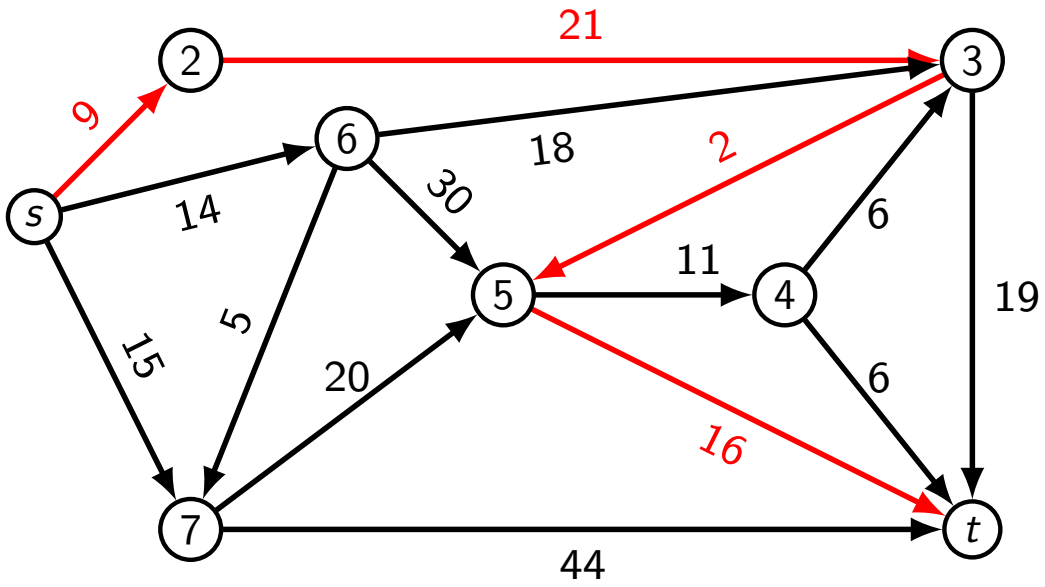
Συντομότερη Διαδρομή

Δεδομένα:

- Κατευθυνόμενο γράφημα $G = (V, A)$.
- Αφετηρία s, προορισμός t.
- $\ell_e > 0$ πραγματικός αριθμός = μήκος της ακμής e.

Κόστος διαδρομής: άθροισμα μηκών των ακμών της διαδρομής.

Πρόβλημα συντομότερης διαδρομής: Να βρούμε τη συντομότερη σε μήκος κατευθυνόμενη διαδρομή από την κορυφή s στην κορυφή t.



Μήκος διαδρομής
$(s \to 2 \to 3 \to 5 \to t) =$
$9 + 21 + 2 + 16 = 48$

Αλγόριθμος Dijkstra

- Διατήρησε ένα σύνολο S εξερευνημένων κορυφών $\{u\}$ για το οποίο έχουμε προσδιορίσει την απόσταση $d(u)$ της συντομότερης διαδρομής από την s στην u.

Αλγόριθμος Dijkstra

- Διατήρησε ένα σύνολο S εξερευνημένων κορυφών $\{u\}$ για το οποίο έχουμε προσδιορίσει την απόσταση $d(u)$ της συντομότερης διαδρομής από την s στην u.

- Αρχικοποίησε $S = \{s\}$, $d(s) = 0$.

Αλγόριθμος Dijkstra

- Διατήρησε ένα σύνολο S εξερευνημένων κορυφών $\{u\}$ για το οποίο έχουμε προσδιορίσει την απόσταση $d(u)$ της συντομότερης διαδρομής από την s στην u.

- Αρχικοποίησε $S = \{s\}$, $d(s) = 0$.

- Επαναληπτικά, διάλεξε την κορυφή v που δεν έχει εξερευνηθεί και ελαχιστοποιεί την τρέχουσα ελάχιστη απόσταση $\pi(v) = d(u) + \ell_{uv}$ από τις διαδρομές που περιέχουν την συντομότερη διαδρομή προς u στο S, ακολουθούμενη από ακμή $e = (u, v)$.

Αλγόριθμος Dijkstra

- Διατήρησε ένα σύνολο S εξερευνημένων κορυφών $\{u\}$ για το οποίο έχουμε προσδιορίσει την απόσταση $d(u)$ της συντομότερης διαδρομής από την s στην u.

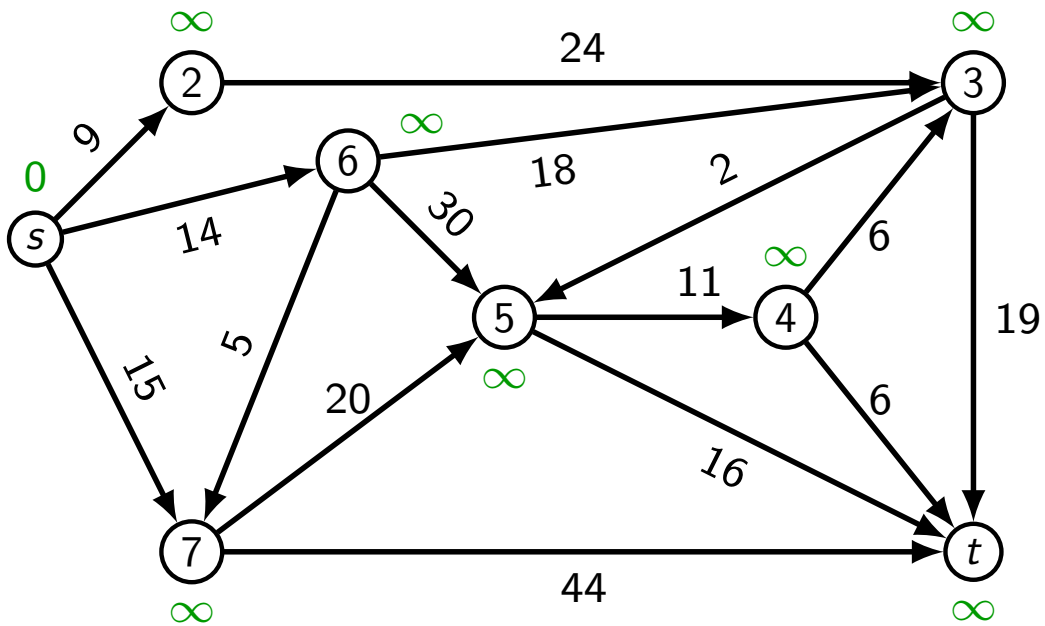
- Αρχικοποίησε $S = \{s\}$, $d(s) = 0$.

- Επαναληπτικά, διάλεξε την κορυφή v που δεν έχει εξερευνηθεί και ελαχιστοποιεί την τρέχουσα ελάχιστη απόσταση $\pi(v) = d(u) + \ell_{uv}$ από τις διαδρομές που περιέχουν την συντομότερη διαδρομή προς u στο S, ακολουθούμενη από ακμή $e = (u, v)$.

- Θέσε $d(v) = \pi(v)$ και πρόσθεσε την v στο S.

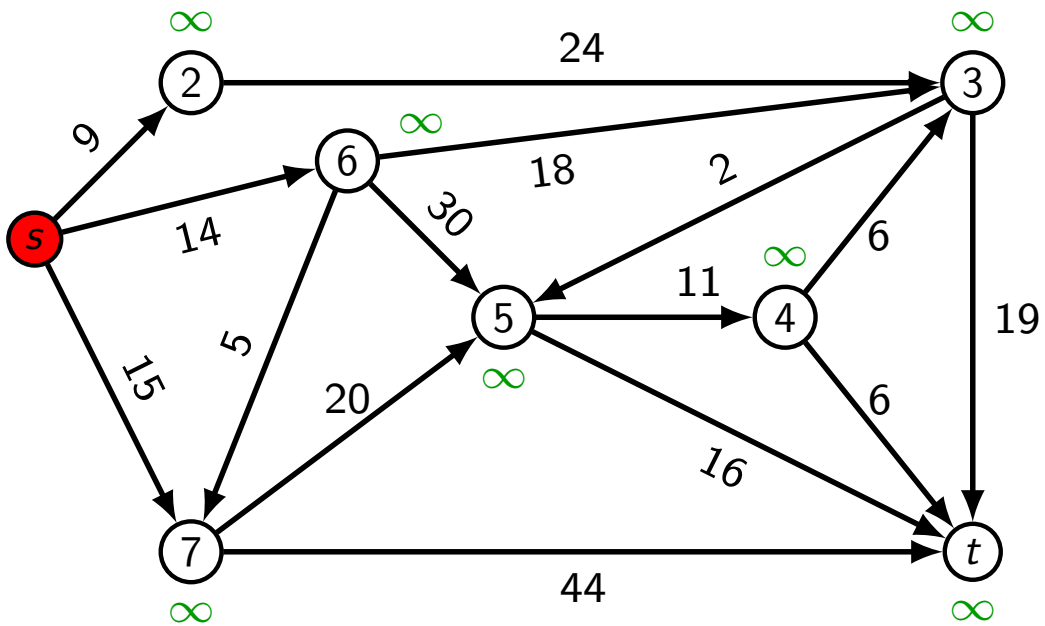
Παράδειγμα

$$S = \{\}$$
$$PQ = \{s, 2, 3, 4, 5, 6, 7, t\}$$



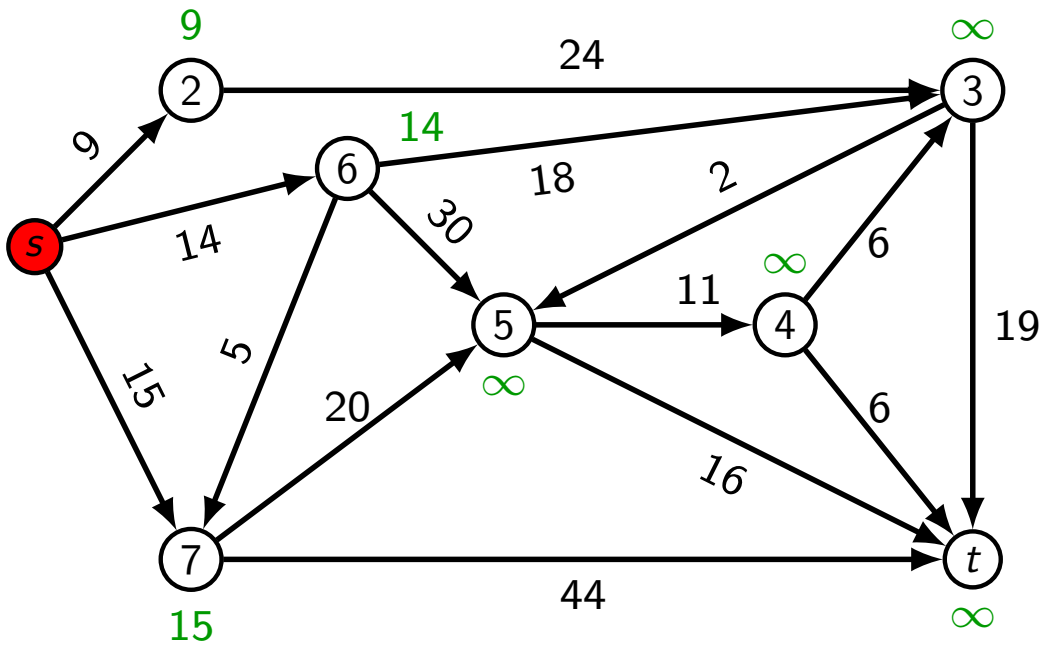
Παράδειγμα

$$\begin{aligned} S &= \{s\} \\ PQ &= \{2, 3, 4, 5, 6, 7, t\} \end{aligned}$$



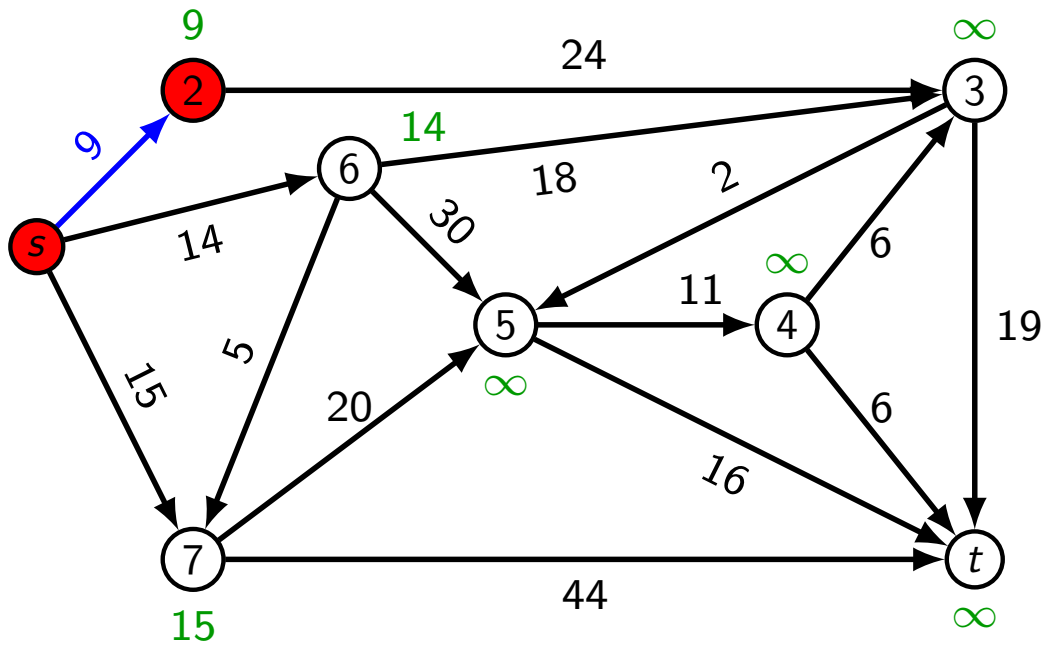
Παράδειγμα

$$\begin{aligned} S &= \{s\} \\ PQ &= \{2, 3, 4, 5, 6, 7, t\} \end{aligned}$$



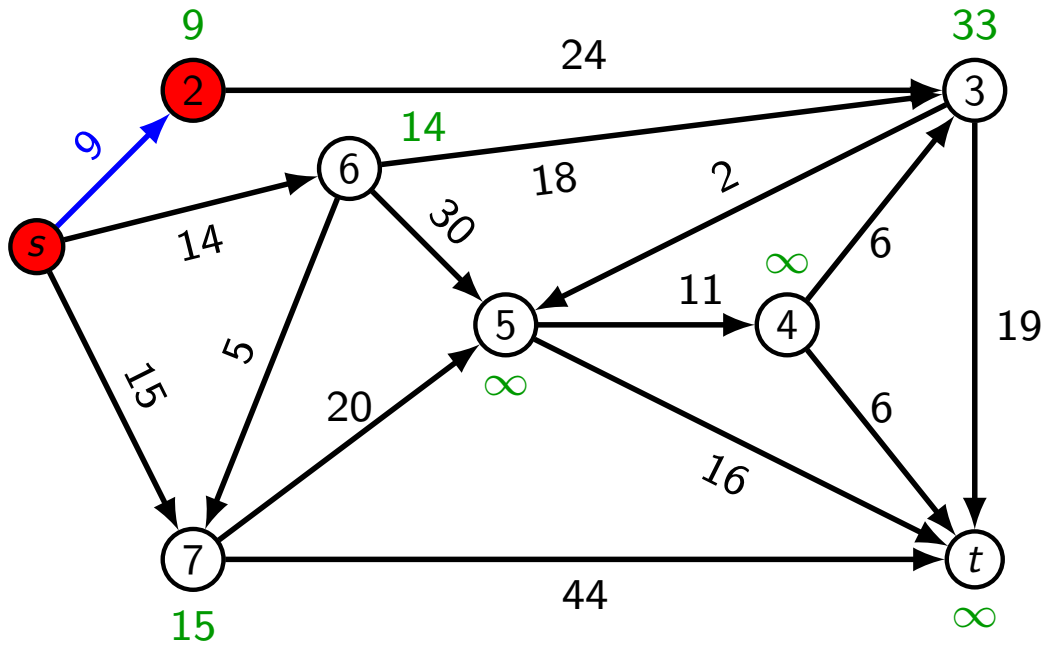
Παράδειγμα

$$\begin{aligned} S &= \{s, 2\} \\ PQ &= \{3, 4, 5, 6, 7, t\} \end{aligned}$$



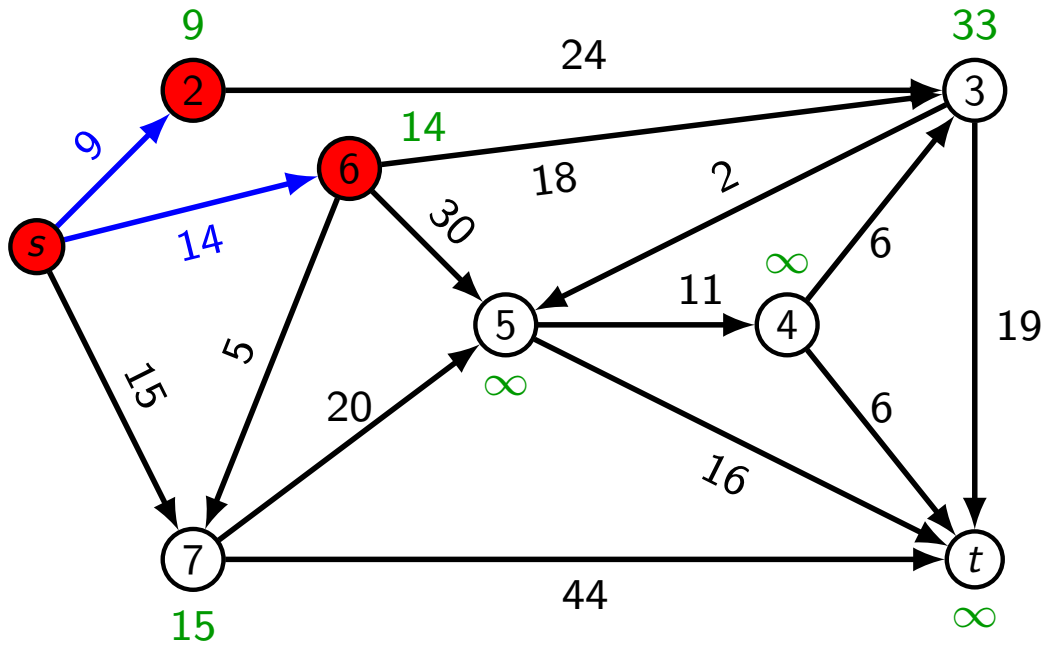
Παράδειγμα

$$\begin{aligned} S &= \{s, 2\} \\ PQ &= \{3, 4, 5, 6, 7, t\} \end{aligned}$$



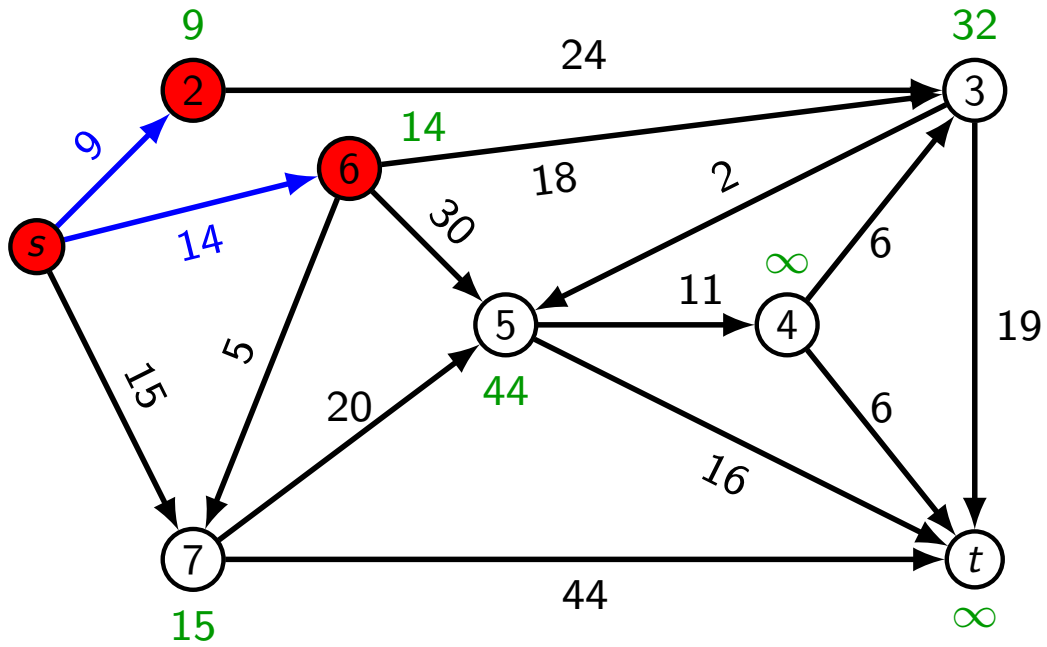
Παράδειγμα

$$\begin{aligned} S &= \{s, 2, 6\} \\ PQ &= \{3, 4, 5, 7, t\} \end{aligned}$$



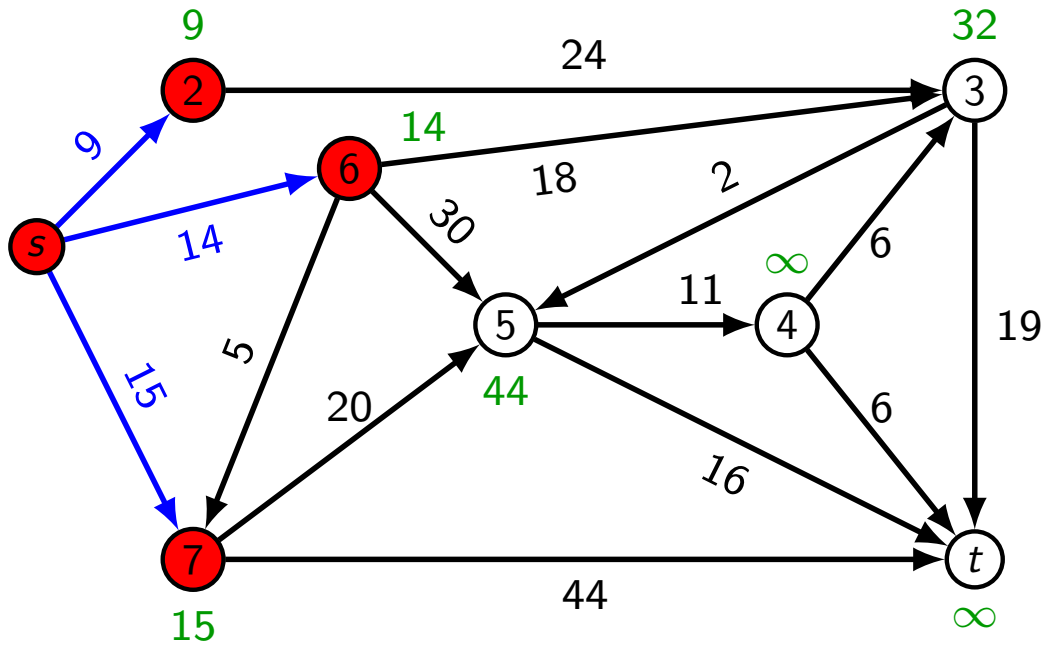
Παράδειγμα

$$\begin{aligned} S &= \{s, 2, 6\} \\ PQ &= \{3, 4, 5, 7, t\} \end{aligned}$$



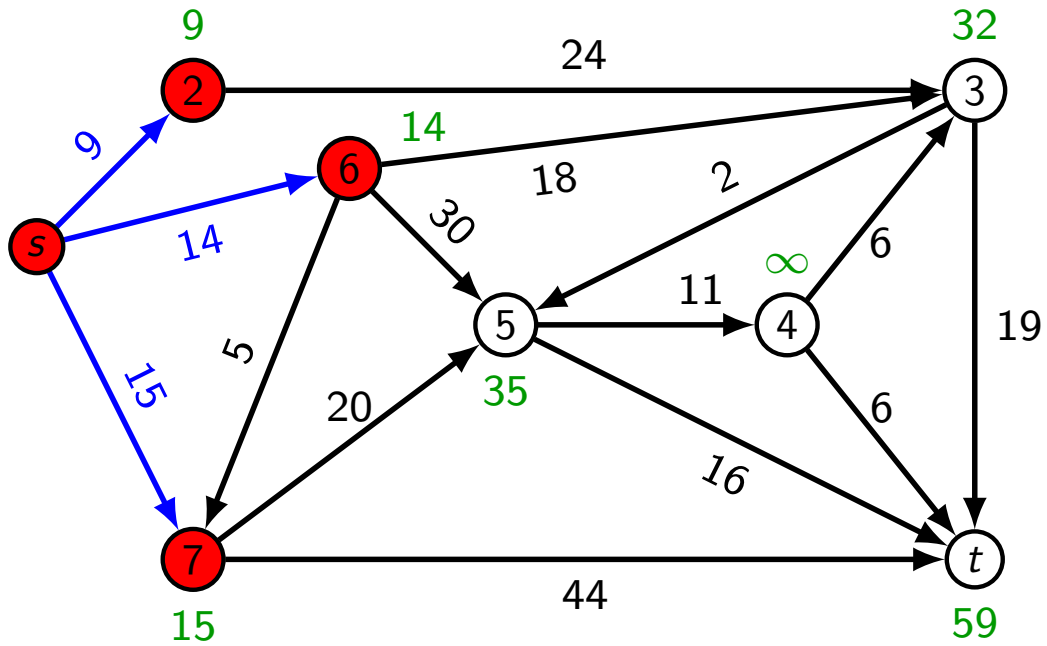
Παράδειγμα

$$S = \{s, 2, 6, 7\}$$
$$PQ = \{3, 4, 5, t\}$$



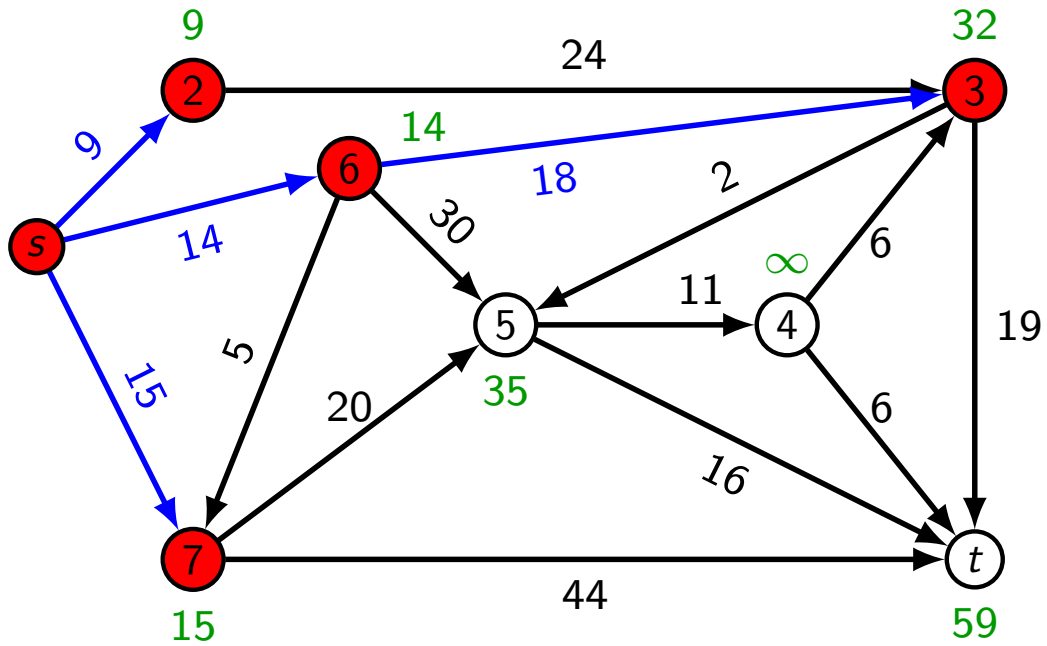
Παράδειγμα

$$S = \{s, 2, 6, 7\}$$
$$PQ = \{3, 4, 5, t\}$$



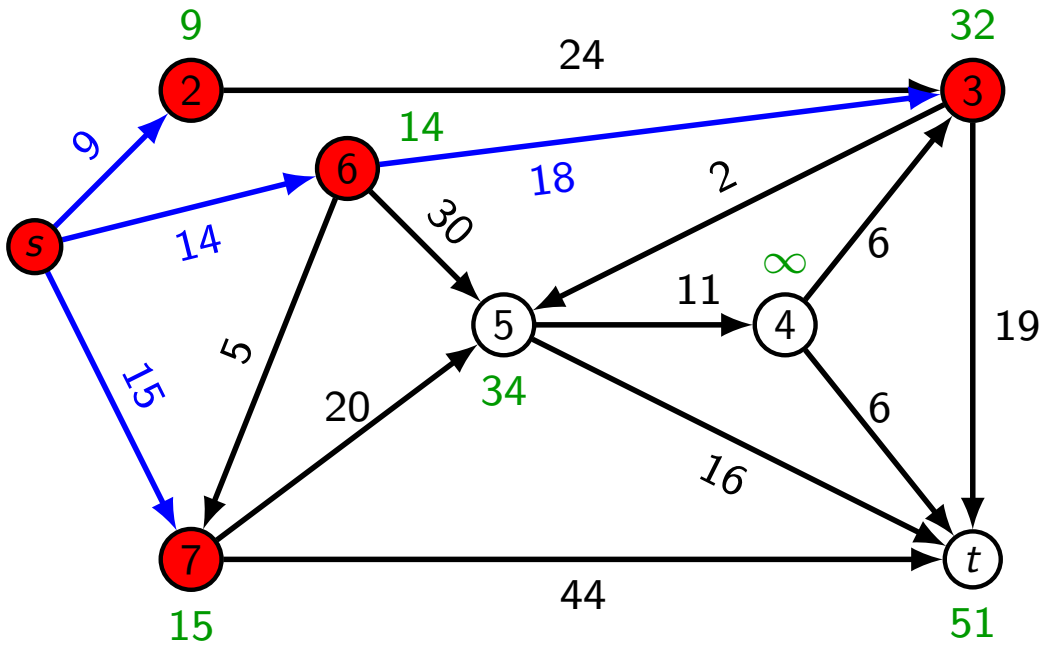
Παράδειγμα

$$S = \{s, 2, 6, 7, 3\}$$
$$PQ = \{4, 5, t\}$$



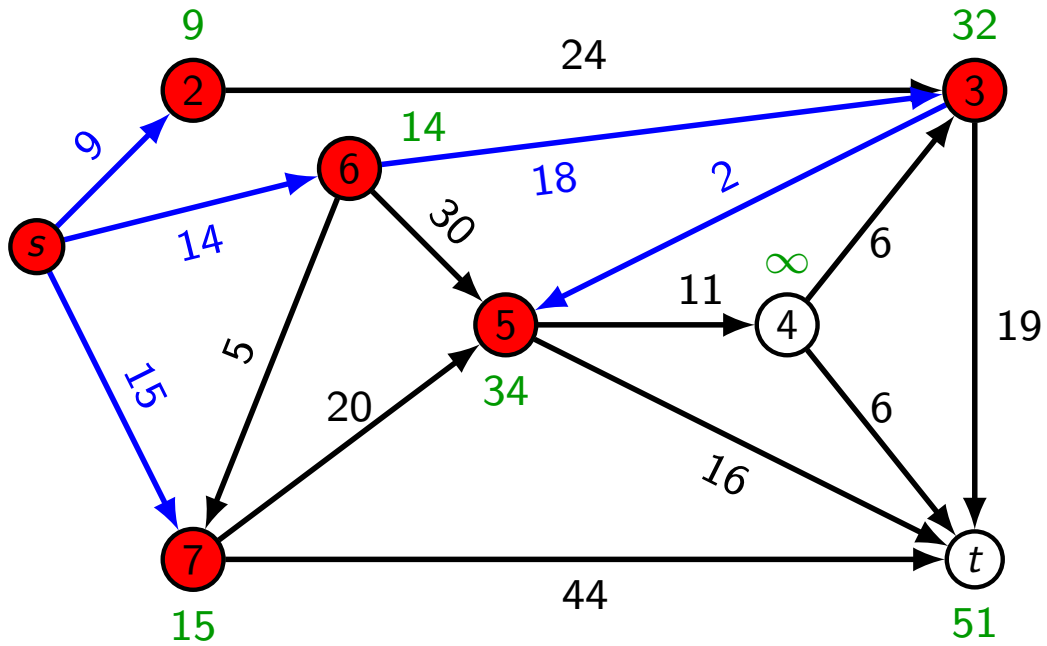
Παράδειγμα

$$S = \{s, 2, 6, 7, 3\}$$
$$PQ = \{4, 5, t\}$$



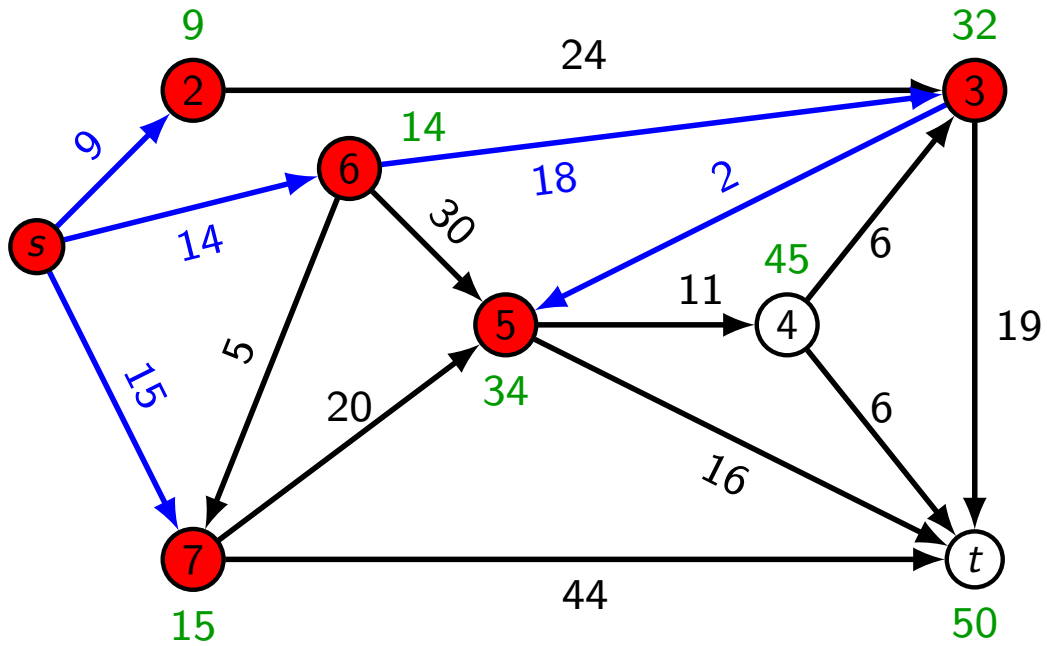
Παράδειγμα

$$S = \{s, 2, 6, 7, 3, 5\}$$
$$PQ = \{4, t\}$$



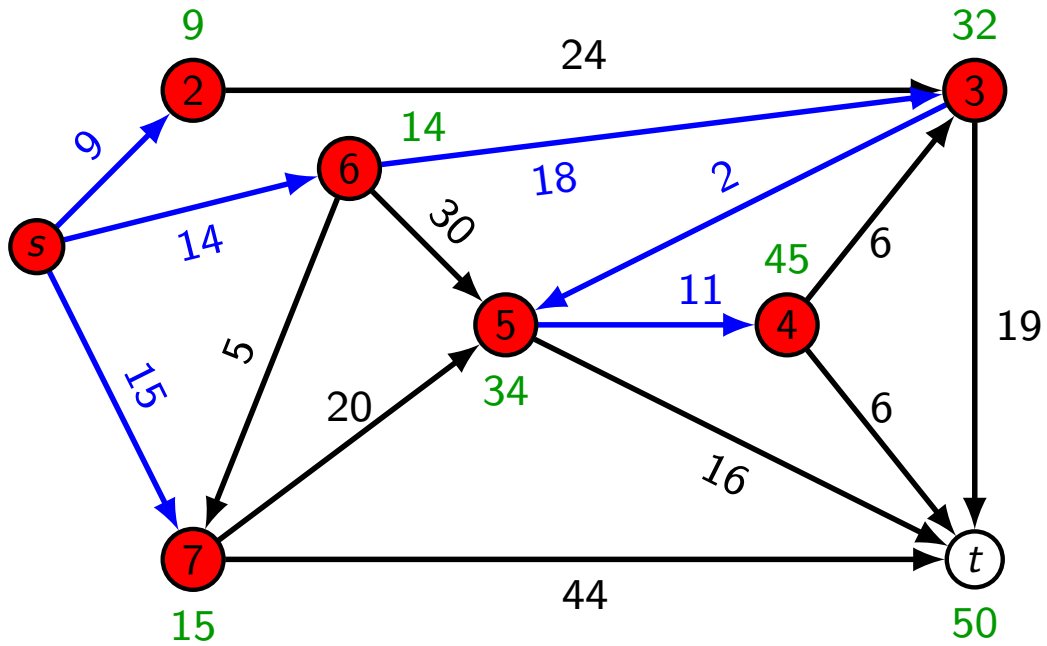
Παράδειγμα

$$\begin{aligned} S &= \{s, 2, 6, 7, 3, 5\} \\ PQ &= \{4, t\} \end{aligned}$$



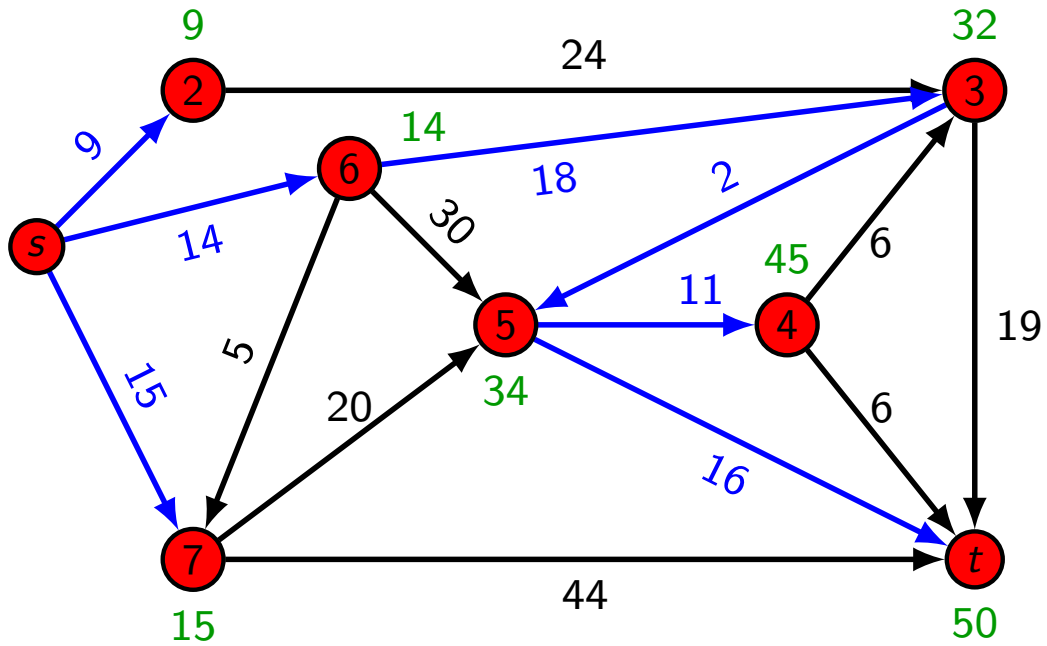
Παράδειγμα

$$S = \{s, 2, 6, 7, 3, 5, 4\}$$
$$PQ = \{t\}$$



Παράδειγμα

$$S = \{s, 2, 6, 7, 3, 5, 4, t\}$$
$$PQ = \{\}$$



Πολυπλοκότητα (revisited)

- Για κάθε ανεξερεύνητη κορυφή v, διατηρούμε $\pi(v) = \min_{u \in S}\{d(u) + \ell_{uv}\}$, όπου $\ell_{uv} = \infty$ αν δεν υπάρχει η ακμή (u, v).

Πολυπλοκότητα (revisited)

- Για κάθε ανεξερεύνητη κορυφή v, διατηρούμε $\pi(v) = \min_{u \in S}\{d(u) + \ell_{uv}\}$, όπου $\ell_{uv} = \infty$ αν δεν υπάρχει η ακμή (u, v).

- Επόμενη κορυφή προς εξερεύνηση = κορυφή με ελάχιστο $\pi(v)$. Όταν η v εντάσσεται στο S, για κάθε προσκείμενη ακμή $e = (v, w)$, ενημερώνουμε την τιμή $\pi(w) := min\{\pi(w), \pi(v) + \ell_e\}$. Συνολικά $\mathcal{O}(m)$ βήματα.

  Το κόστος του κάθε βήματος είναι $\mathcal{O}(\log n)$ το οποίο προέρχεται από την κλήση της ExtractMin και την διόρθωση του σωρού με κλήσεις Heapify-up.

Πολυπλοκότητα (revisited)

- Για κάθε ανεξερεύνητη κορυφή v, διατηρούμε $\pi(v) = \min_{u \in S}\{d(u) + \ell_{uv}\}$, όπου $\ell_{uv} = \infty$ αν δεν υπάρχει η ακμή (u, v).

- Επόμενη κορυφή προς εξερεύνηση = κορυφή με ελάχιστο $\pi(v)$. Όταν η v εντάσσεται στο S, για κάθε προσκείμενη ακμή $e = (v, w)$, ενημερώνουμε την τιμή $\pi(w) := min\{\pi(w), \pi(v) + \ell_e\}$. Συνολικά $\mathcal{O}(m)$ βήματα.

  Το κόστος του κάθε βήματος είναι $\mathcal{O}(\log n)$ το οποίο προέρχεται από την κλήση της ExtractMin και την διόρθωση του σωρού με κλήσεις Heapify-up.

- **Αποδοτική υλοποίηση**: Διατηρήστε ουρά προτεραιότητας PQ από ανεξερεύνητες κορυφές με προτεραιότητα $\pi(v)$.

Πολυπλοκότητα (revisited)

- Για κάθε ανεξερεύνητη κορυφή v, διατηρούμε $\pi(v) = \min_{u \in S}\{d(u) + \ell_{uv}\}$, όπου $\ell_{uv} = \infty$ αν δεν υπάρχει η ακμή (u, v).

- Επόμενη κορυφή προς εξερεύνηση = κορυφή με ελάχιστο $\pi(v)$. Όταν η v εντάσσεται στο S, για κάθε προσκείμενη ακμή $e = (v, w)$, ενημερώνουμε την τιμή $\pi(w) := min\{\pi(w), \pi(v) + \ell_e\}$. Συνολικά $\mathcal{O}(m)$ βήματα.

  Το κόστος του κάθε βήματος είναι $\mathcal{O}(\log n)$ το οποίο προέρχεται από την κλήση της ExtractMin και την διόρθωση του σωρού με κλήσεις Heapify-up.

- **Αποδοτική υλοποίηση**: Διατηρήστε ουρά προτεραιότητας PQ από ανεξερεύνητες κορυφές με προτεραιότητα $\pi(v)$.

- Σύνολο: $\mathcal{O}(m \log n)$ βήματα.

Ελάχιστον Επικαλύπτον Δέντρο (ΕΕΔ)

Δεδομένα

- μη-κατευθυνόμενο συνεκτικό γράφημα $G = (V, E)$
- πραγματικά κόστη ακμών c_e

Ελάχιστον Επικαλύπτον Δέντρο (ΕΕΔ)

Δεδομένα

- μη-κατευθυνόμενο συνεκτικό γράφημα $G = (V, E)$
- πραγματικά κόστη ακμών c_e

Ένα επικαλύπτον δέντρο είναι ένα υποσύνολο των ακμών $T \subseteq E$ τ.ώ. το T να είναι δέντρο, επικαλύπτον (επικαλύπτει / περιέχει όλες τις κορυφές του γραφήματος V).

Ελάχιστον Επικαλύπτον Δέντρο (ΕΕΔ)

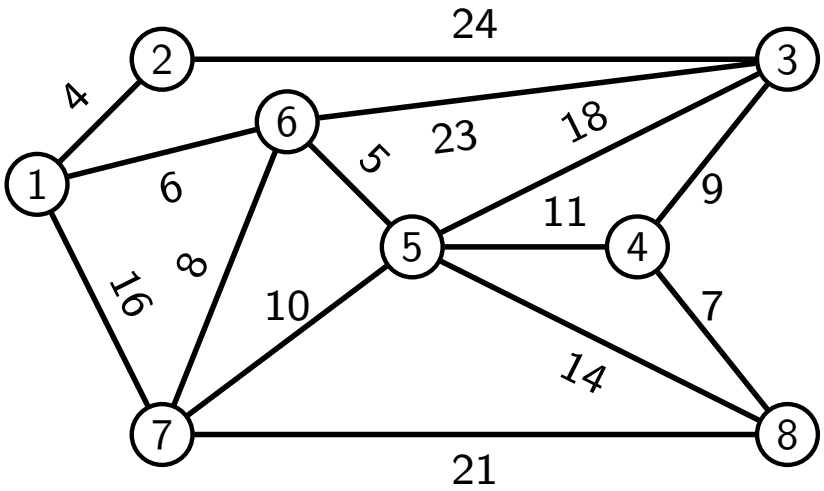
Δεδομένα

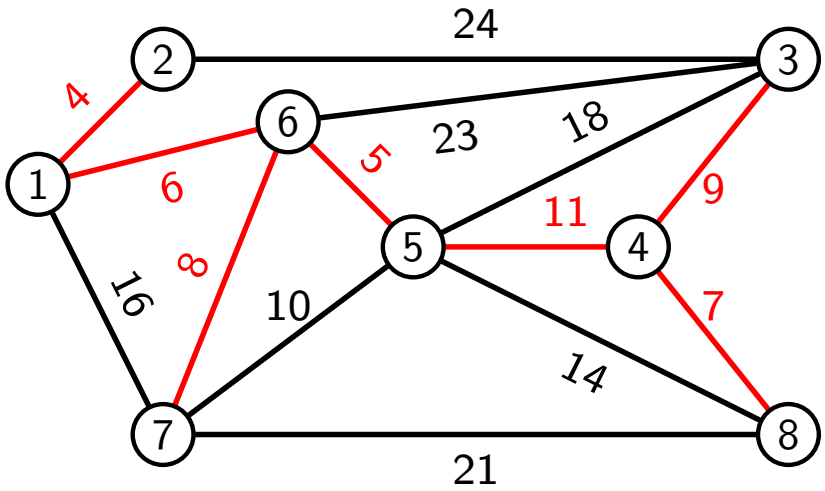
- μη-κατευθυνόμενο συνεκτικό γράφημα $G = (V, E)$
- πραγματικά κόστη ακμών c_e

Ένα επικαλύπτον δέντρο είναι ένα υποσύνολο των ακμών $T \subseteq E$ τ.ώ. το T να είναι δέντρο, επικαλύπτον (επικαλύπτει / περιέχει όλες τις κορυφές του γραφήματος V).

Πρόβλημα: Εύρεση ενός ελάχιστου επικαλύπτοντος δέντρου (του οποίου το άθροισμα των κοστών των ακμών $\sum_{e \in T} c_e$ είναι ελάχιστο).

Παράδειγμα


1
2
3
4
5
6
7
8
24
4
6
5
23
18
16
8
10
11
9
7
14
21


1
2
3
4
5
6
7
8
24
4
6
5
23
18
16
8
10
11
9
7
14
21

Αλγόριθμοι (άπληστοι)

- **Αλγόριθμος του Kruskal:** Ξεκινάμε με κενό $T = \emptyset$. Θεωρούμε τις ακμές σε αύξουσα σειρά με βάση το κόστος. Εισάγουμε μια ακμή e στο T εκτός εάν δημιουργείται κύκλος.
  Τερματισμός: όταν όλες οι κορυφές "καλύπτονται" (ανήκουν στο T).

- **Αλγόριθμος Αντίστροφης Διαγραφής:** Ξεκινάμε με $T = E$. Θεωρούμε τις ακμές σε φθίνουσα σειρά με βάση το κόστος. Διαγράφουμε την ακμή e από το T εκτός εάν το T γίνεται μη συνεκτικό.
  Τερματίζει όταν απαλειφθούν όλοι οι κύκλοι.

- **Αλγόριθμος του Prim:** Ξεκινάμε με μια αρχική κορυφή s: με απληστία μεγαλώνουμε δέντρο T από την s προς τα έξω. Σε κάθε βήμα, προσθέτουμε στο T την ακμή e με το μικρότερο κόστος που έχει ακριβώς ένα της άκρο στο T (αποφυγή κύκλου).
  Τερματίζει όταν καλύψει όλες τις κορυφές ("επικαλύπτον" δένδρο).

Αλγόριθμος του Prim

- Αρχικοποιούμε το $S = \{\text{οποιαδήποτε κορυφή}\}$.

Αλγόριθμος του Prim

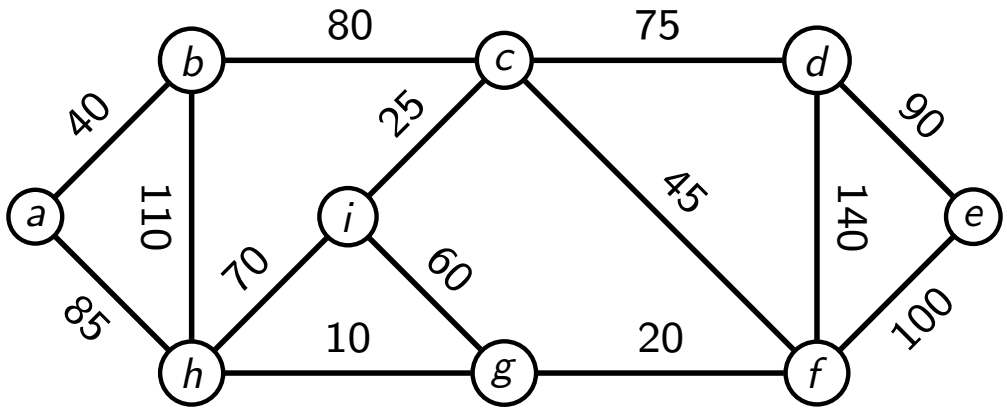
- Αρχικοποιούμε το $S = \{\text{οποιαδήποτε κορυφή}\}$.

- Εφαρμόζουμε την ιδιότητα αποκοπής στο S.

Αλγόριθμος του Prim

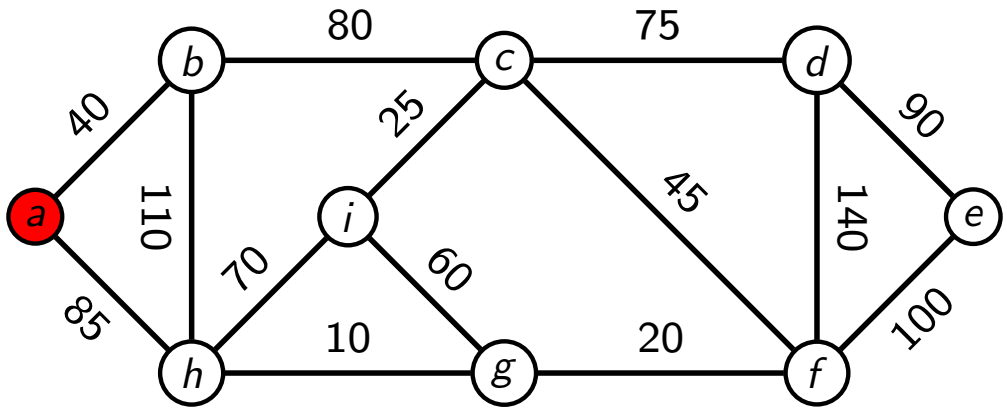
- Αρχικοποιούμε το $S = \{$οποιαδήποτε κορυφή$\}$.
- Εφαρμόζουμε την ιδιότητα αποκοπής στο S.
- Προσθέτουμε στο T την ακμή με το ελάχιστο κόστος στο σύνολο ακμών αποκοπής που αντιστοιχεί στο S, και προσθέτουμε μια νέα κορυφή (2ο άκρο ακμής) στο S.

Παράδειγμα



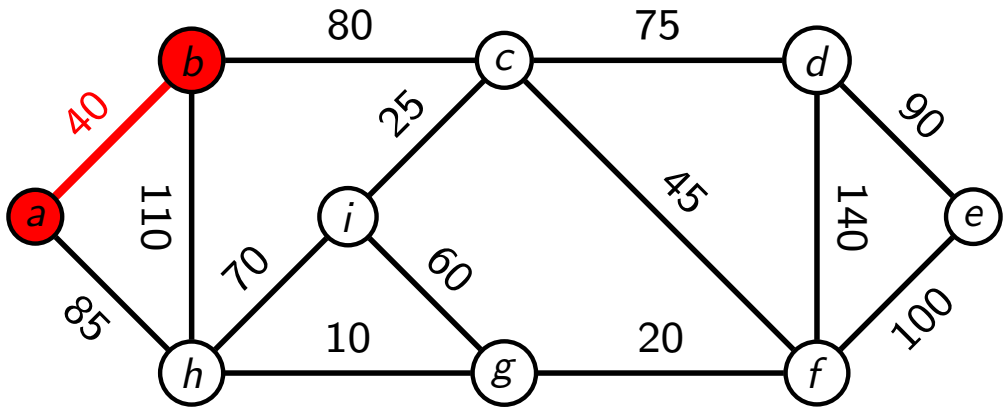
Ξεκινάμε από την κορυφή a.

Παράδειγμα



Ξεκινάμε από την κορυφή a.

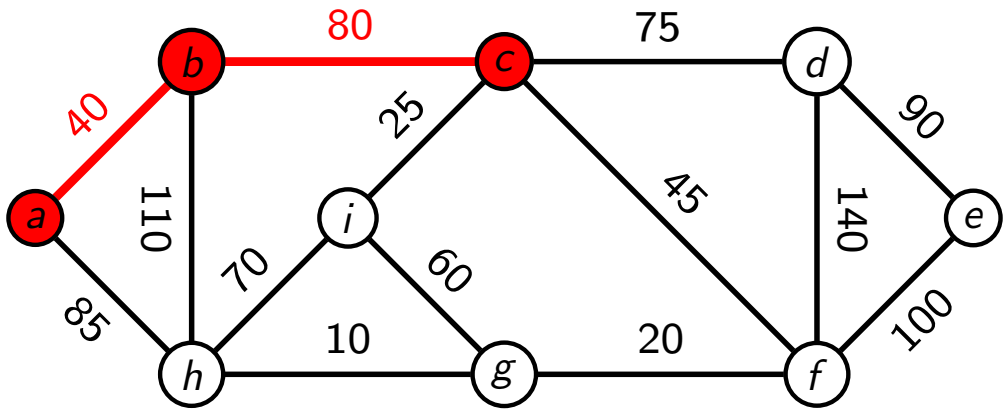
Παράδειγμα



Ξεκινάμε από την κορυφή a.

$T = \{\{a, b\}$

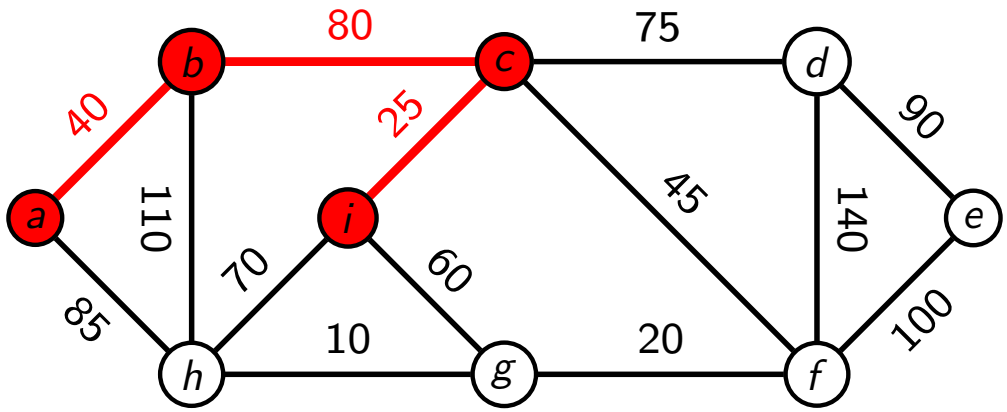
Παράδειγμα



Ξεκινάμε από την κορυφή a.

$T = \{\{a, b\}, \{b, c\}$

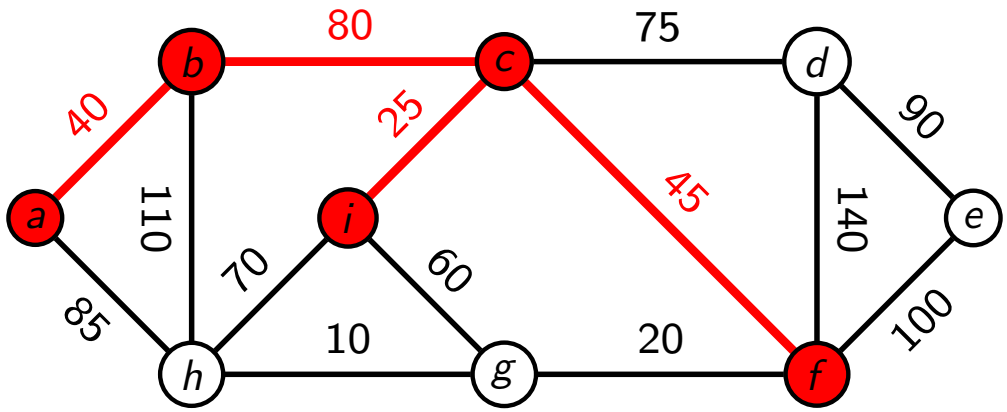
Παράδειγμα



Ξεκινάμε από την κορυφή a.

$$T = \{\{a, b\}, \{b, c\}, \{c, i\}$$

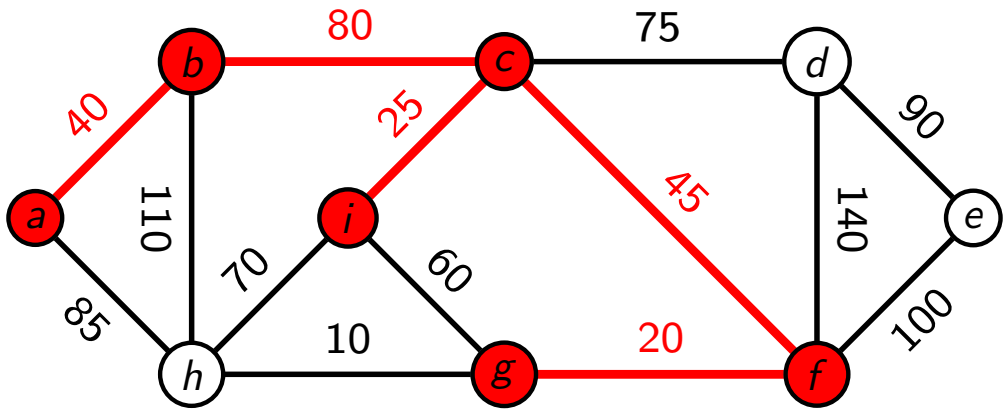
Παράδειγμα



Ξεκινάμε από την κορυφή a.

$$T = \{\{a, b\}, \{b, c\}, \{c, i\}, \{c, f\}$$

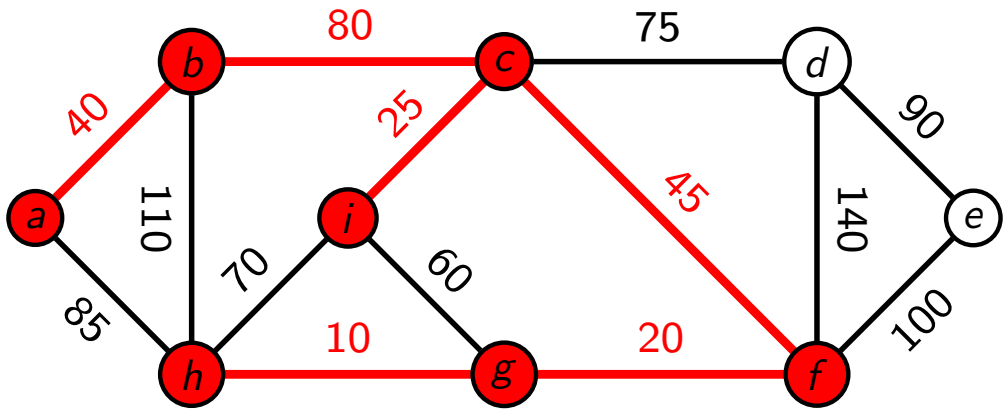
Παράδειγμα



Ξεκινάμε από την κορυφή a.

$$T = \{\{a, b\}, \{b, c\}, \{c, i\}, \{c, f\}, \{f, g\}$$

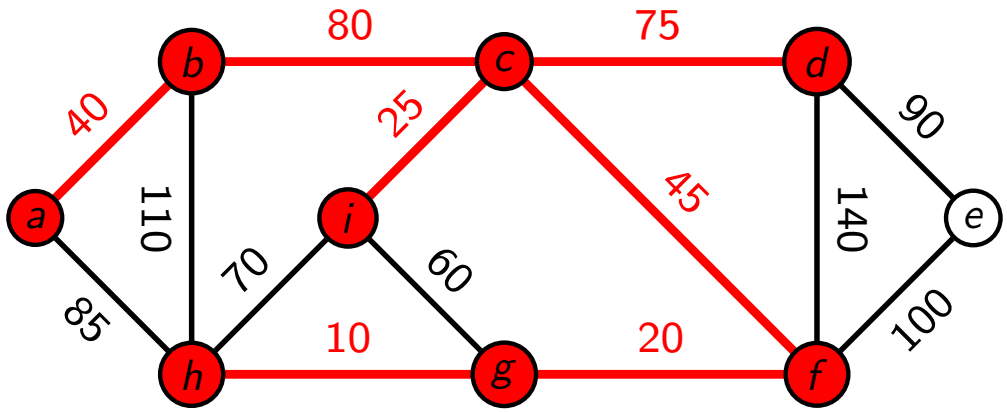
Παράδειγμα



Ξεκινάμε από την κορυφή a.

$$T = \{\{a, b\}, \{b, c\}, \{c, i\}, \{c, f\}, \{f, g\}, \{g, h\}$$

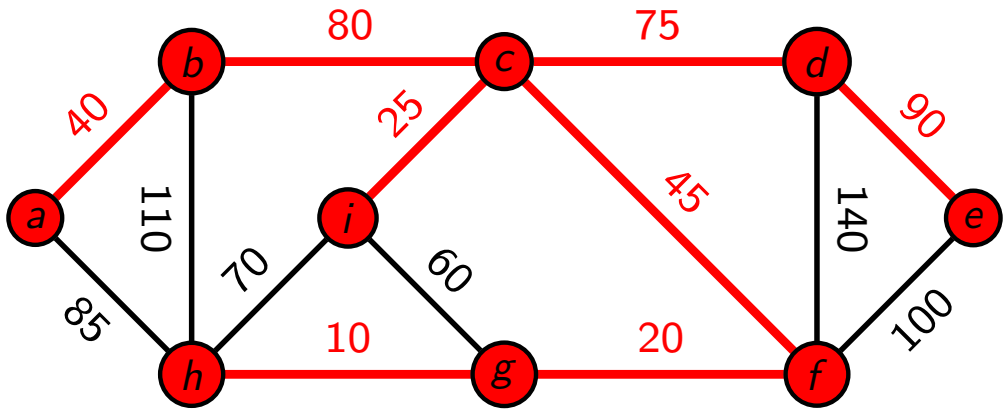
Παράδειγμα



Ξεκινάμε από την κορυφή a.

$$T = \{\{a, b\}, \{b, c\}, \{c, i\}, \{c, f\}, \{f, g\}, \{g, h\}, \{c, d\}$$

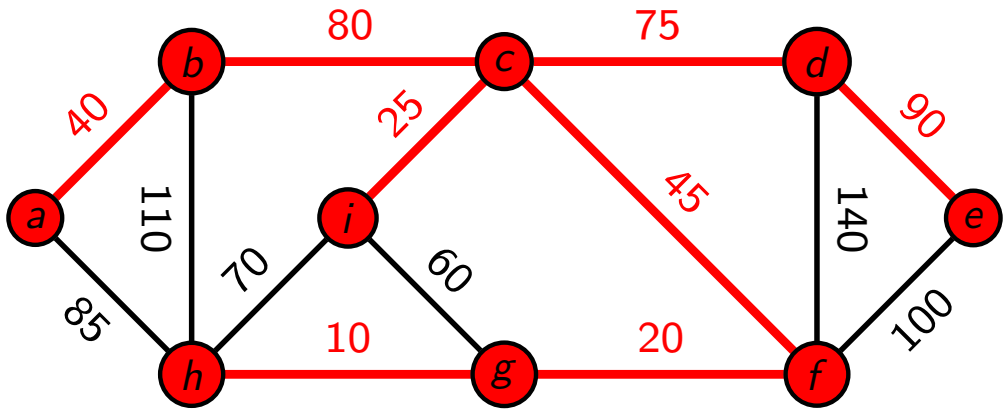
Παράδειγμα



Ξεκινάμε από την κορυφή a.

$$T = \{\{a, b\}, \{b, c\}, \{c, i\}, \{c, f\}, \{f, g\}, \{g, h\}, \{c, d\}, \{d, e\}$$

Παράδειγμα



Ξεκινάμε από την κορυφή a.

$$T = \{\{a, b\}, \{b, c\}, \{c, i\}, \{c, f\}, \{f, g\}, \{g, h\}, \{c, d\}, \{d, e\}\}$$

Ψευδοκώδικας

Prim(G, c, s)
Για κάθε $(v \in V)$ $\alpha[v] \leftarrow \infty$ και $\pi(v) \leftarrow 0$
Αρχικοποίησε μια άδεια ουρά προτεραιότητας Q
$\alpha[s] \leftarrow 0$
Για κάθε $(v \in V)$ εισήγαγε την v στην Q
Αρχικοποίησε σύνολο εξερευνημένων κορυφών $S \leftarrow \emptyset$
Ενόσω (Q δεν είναι άδεια)
  $u \leftarrow$ διέγραψε το ελάχιστο στοιχείο της Q
  $S \leftarrow S \cup \{u\}$
  $T \leftarrow T \cup \{u, \pi\{u\}\}$
  Για κάθε (ακμή $e = \{u, v\}$ περιέχει την u)
    Εάν $(v \notin S)$ και $c_e < \alpha[v]$ τότε $\alpha[v] \leftarrow c_e$ και $\pi(v) \leftarrow u$.

Υλοποίηση (revisited)

Χρησιμοποιούμε ουρά προτεραιότητας όπως ο Dijkstra

- Διατηρούμε σύνολο κορυφών S που έχουν εξερευνηθεί (και T).

Υλοποίηση (revisited)

Χρησιμοποιούμε ουρά προτεραιότητας όπως ο Dijkstra

- Διατηρούμε σύνολο κορυφών S που έχουν εξερευνηθεί (και T).
- Κάθε κορυφή v που δεν έχει εξερευνηθεί βρίσκεται στο Q, με κόστος επισύναψης $\alpha[v] =$ κόστος φθηνότερης ακμής από v σε κορυφή στο S. Το $\alpha[]$ είναι η προτεραιότητα στην ουρά Q.

Υλοποίηση (revisited)

Χρησιμοποιούμε ουρά προτεραιότητας όπως ο Dijkstra

- Διατηρούμε σύνολο κορυφών S που έχουν εξερευνηθεί (και T).
- Κάθε κορυφή v που δεν έχει εξερευνηθεί βρίσκεται στο Q, με κόστος επισύναψης $\alpha[v] =$ κόστος φθηνότερης ακμής από v σε κορυφή στο S. Το $\alpha[]$ είναι η προτεραιότητα στην ουρά Q.
- $\mathcal{O}(n^2)$ με πίνακα κορυφών, $\mathcal{O}(m \log n)$ updates με δυαδικό σωρό (λογαριθμικό κόστος ελέγχου $v \in S$ και ανανέωσης σωρού).

Αλγόριθμος του Kruskal

Θεωρούμε ακμές με αυξανόμενη σειρά κόστους, διατηρούμε T.

Αλγόριθμος του Kruskal

Θεωρούμε ακμές με αυξανόμενη σειρά κόστους, διατηρούμε T.

Περίπτωση 1: Αν προσθέτοντας την e στο T δημιουργεί κύκλο, απορρίπτουμε την e σύμφωνα με την ιδιότητα κύκλου.

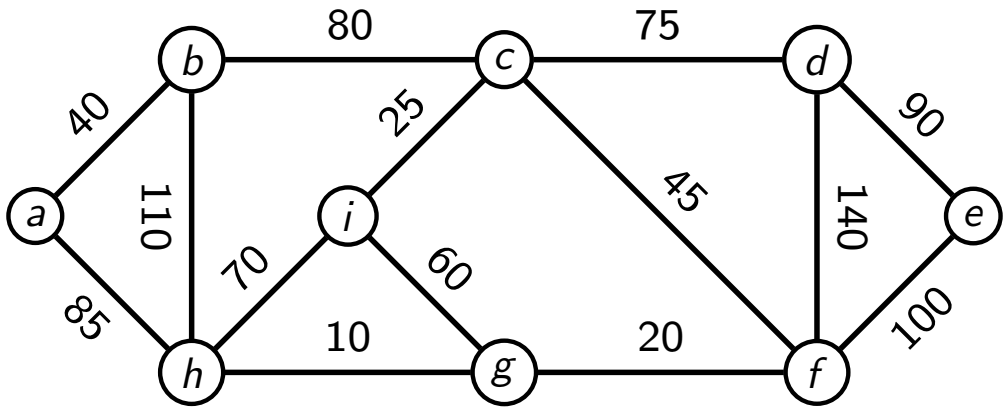
Αλγόριθμος του Kruskal

Θεωρούμε ακμές με αυξανόμενη σειρά κόστους, διατηρούμε T.

Περίπτωση 1: Αν προσθέτοντας την e στο T δημιουργεί κύκλο, απορρίπτουμε την e σύμφωνα με την ιδιότητα κύκλου.

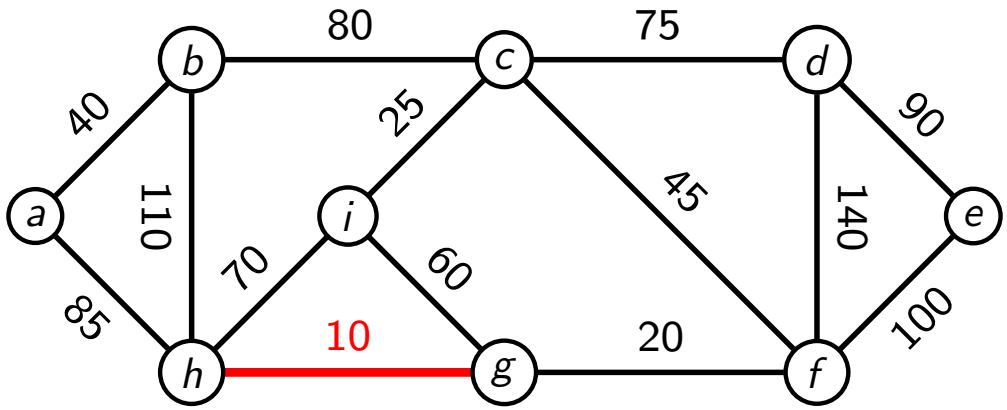
Περίπτωση 2: Αλλιώς, εισάγουμε $e = \{u, v\}$ στο T από την ιδιότητα αποκοπής, όπου $S = \{$κορυφές στη συνεκτική συνιστώσα της $u\}$, e η φθηνότερη στις ακμές αποκοπής.

Παράδειγμα



Ο αλγόριθμος ταξινομεί τις ακμές κατά αύξον κόστος. Ακμή που δεν δημιουργεί κύκλο μπαίνει στο ΕΕΔ αλλιώς απορρίπτεται.

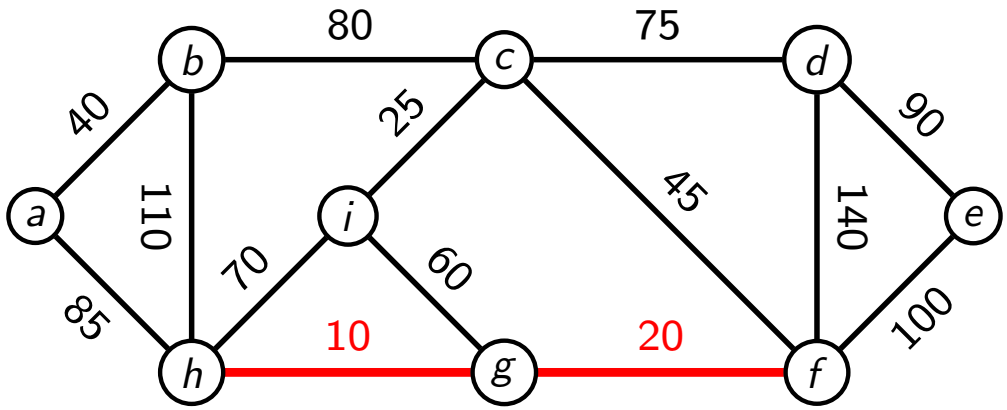
Παράδειγμα



Ο αλγόριθμος ταξινομεί τις ακμές κατά αύξον κόστος. Ακμή που δεν δημιουργεί κύκλο μπαίνει στο ΕΕΔ αλλιώς απορρίπτεται.

$T = \{\{g, h\}$

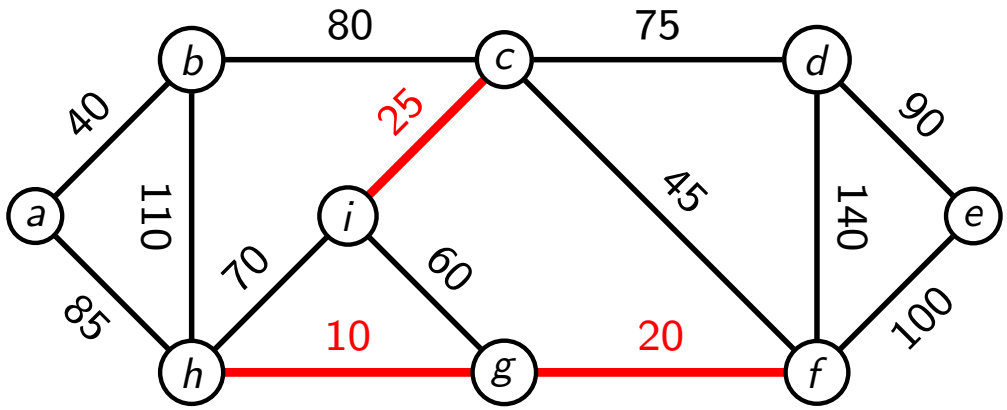
Παράδειγμα



Ο αλγόριθμος ταξινομεί τις ακμές κατά αύξον κόστος. Ακμή που δεν δημιουργεί κύκλο μπαίνει στο ΕΕΔ αλλιώς απορρίπτεται.

$$T = \{\{g, h\}, \{f, g\}$$

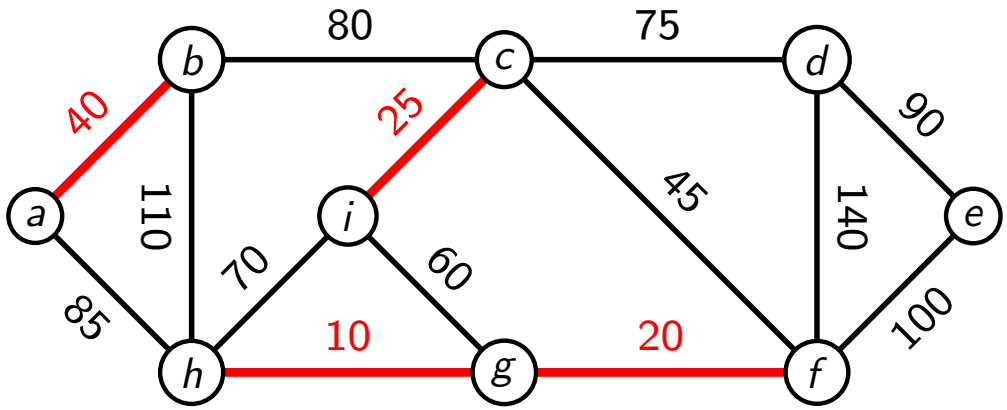
Παράδειγμα



Ο αλγόριθμος ταξινομεί τις ακμές κατά αύξον κόστος. Ακμή που δεν δημιουργεί κύκλο μπαίνει στο ΕΕΔ αλλιώς απορρίπτεται.

$$T = \{\{g, h\}, \{f, g\}, \{c, i\}$$

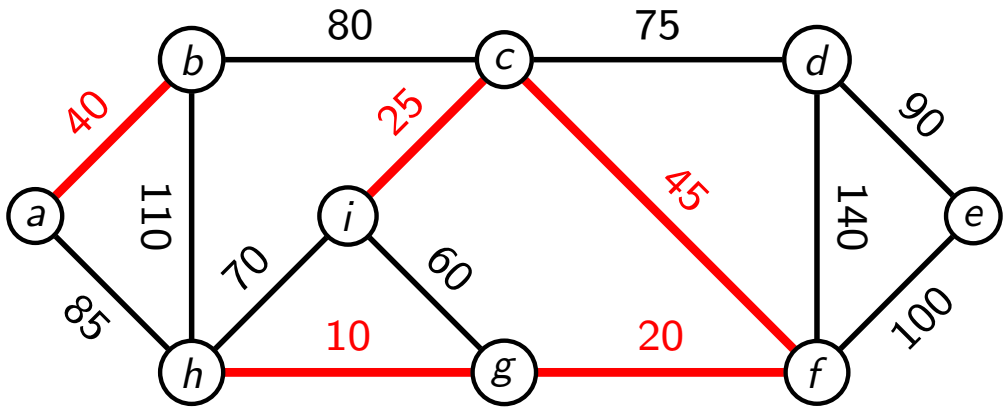
Παράδειγμα



Ο αλγόριθμος ταξινομεί τις ακμές κατά αύξον κόστος. Ακμή που δεν δημιουργεί κύκλο μπαίνει στο ΕΕΔ αλλιώς απορρίπτεται.

$$T = \{\{g, h\}, \{f, g\}, \{c, i\}, \{a, b\}$$

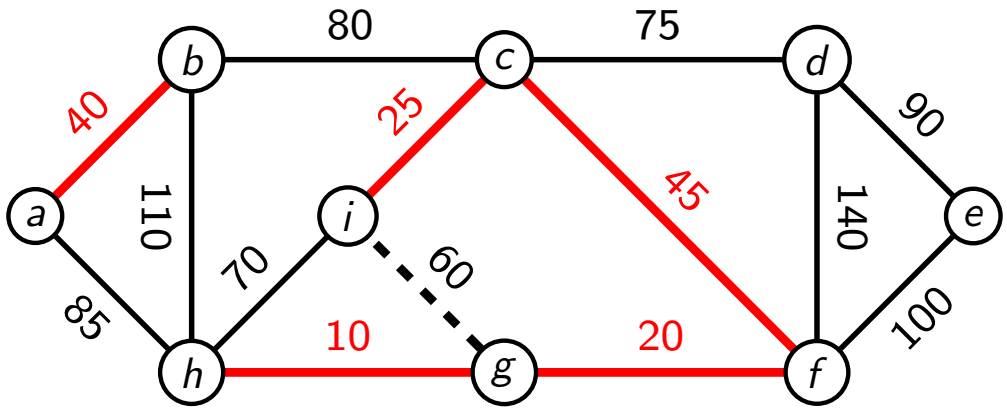
Παράδειγμα



Ο αλγόριθμος ταξινομεί τις ακμές κατά αύξον κόστος. Ακμή που δεν δημιουργεί κύκλο μπαίνει στο ΕΕΔ αλλιώς απορρίπτεται.

$$T = \{\{g, h\}, \{f, g\}, \{c, i\}, \{a, b\}, \{c, f\}$$

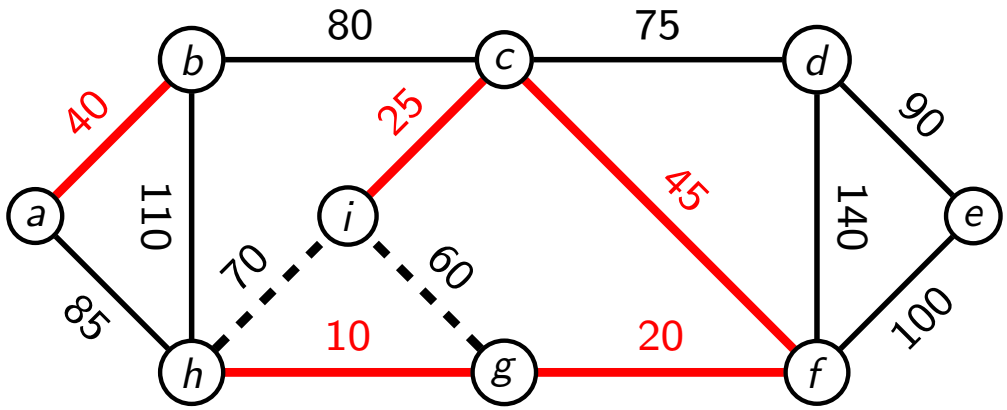
Παράδειγμα



Ο αλγόριθμος ταξινομεί τις ακμές κατά αύξον κόστος. Ακμή που δεν δημιουργεί κύκλο μπαίνει στο ΕΕΔ αλλιώς απορρίπτεται.

$$T = \{\{g, h\}, \{f, g\}, \{c, i\}, \{a, b\}, \{c, f\}$$

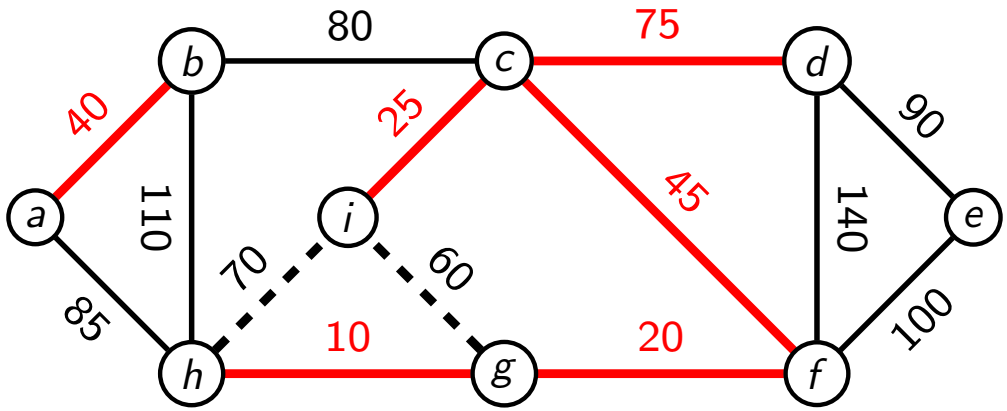
Παράδειγμα



Ο αλγόριθμος ταξινομεί τις ακμές κατά αύξον κόστος. Ακμή που δεν δημιουργεί κύκλο μπαίνει στο ΕΕΔ αλλιώς απορρίπτεται.

$$T = \{\{g, h\}, \{f, g\}, \{c, i\}, \{a, b\}, \{c, f\}$$

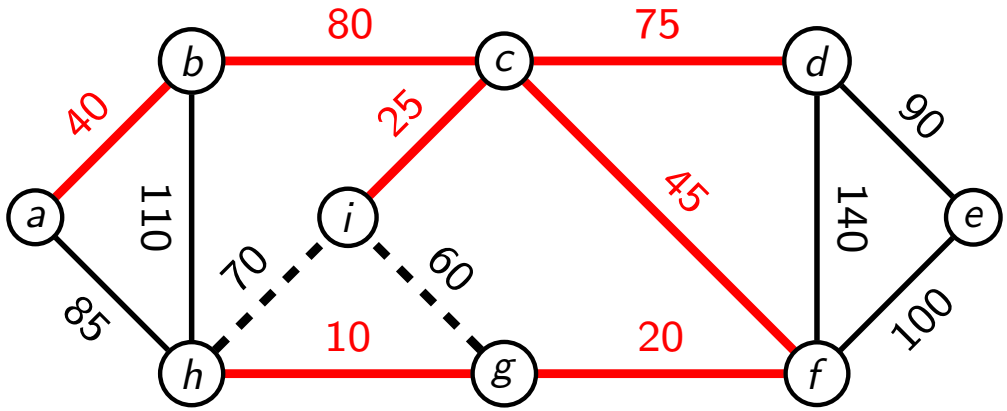
Παράδειγμα



Ο αλγόριθμος ταξινομεί τις ακμές κατά αύξον κόστος. Ακμή που δεν δημιουργεί κύκλο μπαίνει στο ΕΕΔ αλλιώς απορρίπτεται.

$$T = \{\{g, h\}, \{f, g\}, \{c, i\}, \{a, b\}, \{c, f\}, \{c, d\}$$

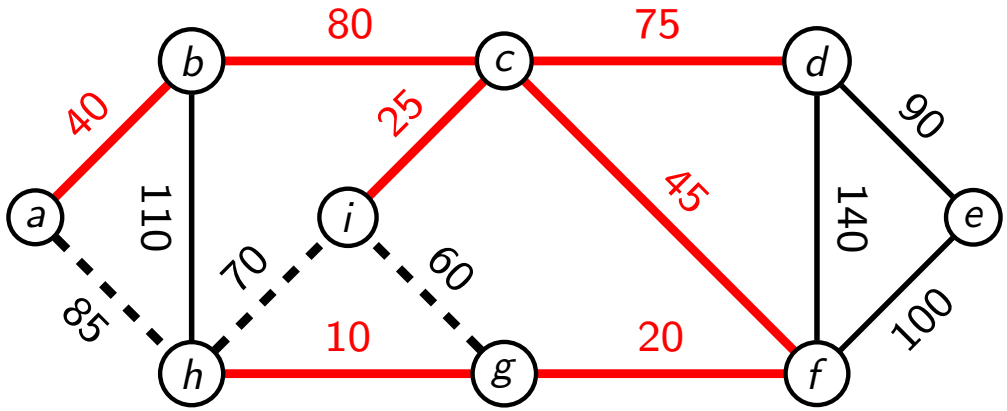
Παράδειγμα



Ο αλγόριθμος ταξινομεί τις ακμές κατά αύξον κόστος. Ακμή που δεν δημιουργεί κύκλο μπαίνει στο ΕΕΔ αλλιώς απορρίπτεται.

$$T = \{\{g, h\}, \{f, g\}, \{c, i\}, \{a, b\}, \{c, f\}, \{c, d\}, \{b, c\}$$

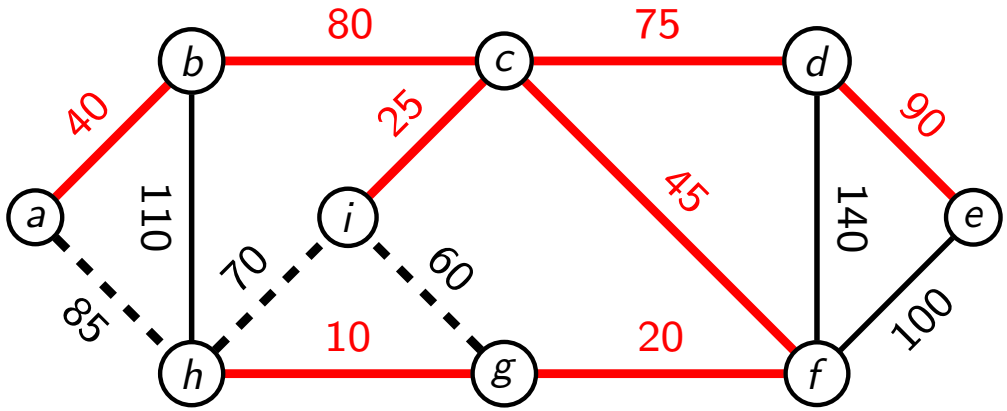
Παράδειγμα



Ο αλγόριθμος ταξινομεί τις ακμές κατά αύξον κόστος. Ακμή που δεν δημιουργεί κύκλο μπαίνει στο ΕΕΔ αλλιώς απορρίπτεται.

$$T = \{\{g, h\}, \{f, g\}, \{c, i\}, \{a, b\}, \{c, f\}, \{c, d\}, \{b, c\}$$

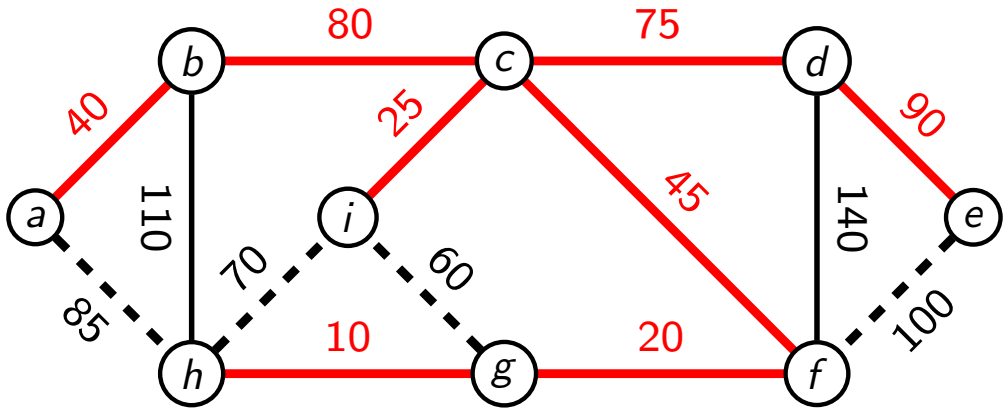
Παράδειγμα



Ο αλγόριθμος ταξινομεί τις ακμές κατά αύξον κόστος. Ακμή που δεν δημιουργεί κύκλο μπαίνει στο ΕΕΔ αλλιώς απορρίπτεται.

$$T = \{\{g, h\}, \{f, g\}, \{c, i\}, \{a, b\}, \{c, f\}, \{c, d\}, \{b, c\}, \{d, e\}$$

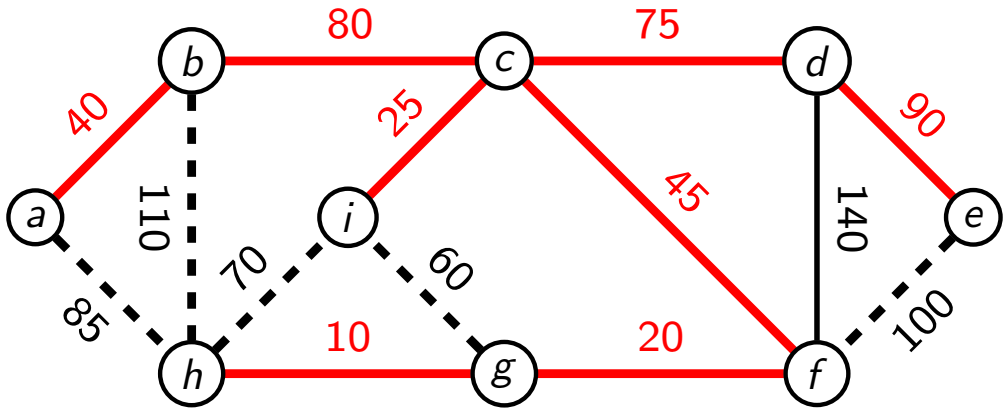
Παράδειγμα



Ο αλγόριθμος ταξινομεί τις ακμές κατά αύξον κόστος. Ακμή που δεν δημιουργεί κύκλο μπαίνει στο ΕΕΔ αλλιώς απορρίπτεται.

$$T = \{\{g,h\},\{f,g\},\{c,i\},\{a,b\},\{c,f\},\{c,d\},\{b,c\},\{d,e\}$$

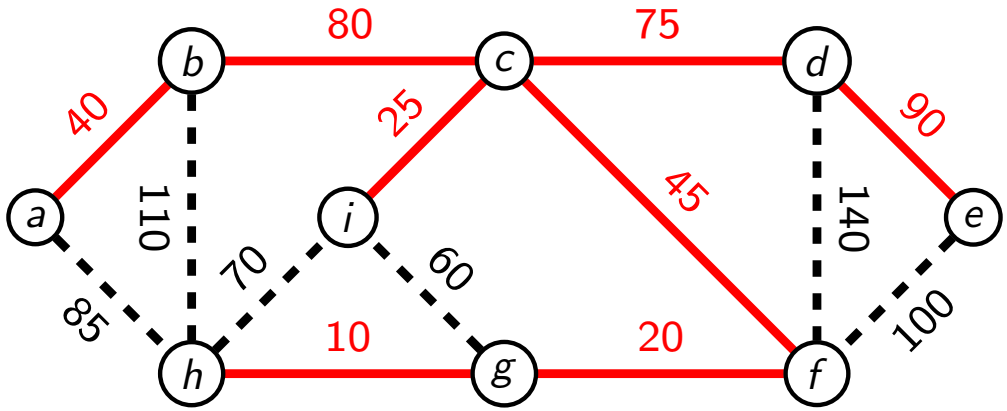
Παράδειγμα



Ο αλγόριθμος ταξινομεί τις ακμές κατά αύξον κόστος. Ακμή που δεν δημιουργεί κύκλο μπαίνει στο ΕΕΔ αλλιώς απορρίπτεται.

$$T = \{\{g, h\}, \{f, g\}, \{c, i\}, \{a, b\}, \{c, f\}, \{c, d\}, \{b, c\}, \{d, e\}$$

Παράδειγμα



Ο αλγόριθμος ταξινομεί τις ακμές κατά αύξον κόστος. Ακμή που δεν δημιουργεί κύκλο μπαίνει στο ΕΕΔ αλλιώς απορρίπτεται.

$$T = \{\{g, h\}, \{f, g\}, \{c, i\}, \{a, b\}, \{c, f\}, \{c, d\}, \{b, c\}, \{d, e\}\}$$

Ψευδοκώδικας

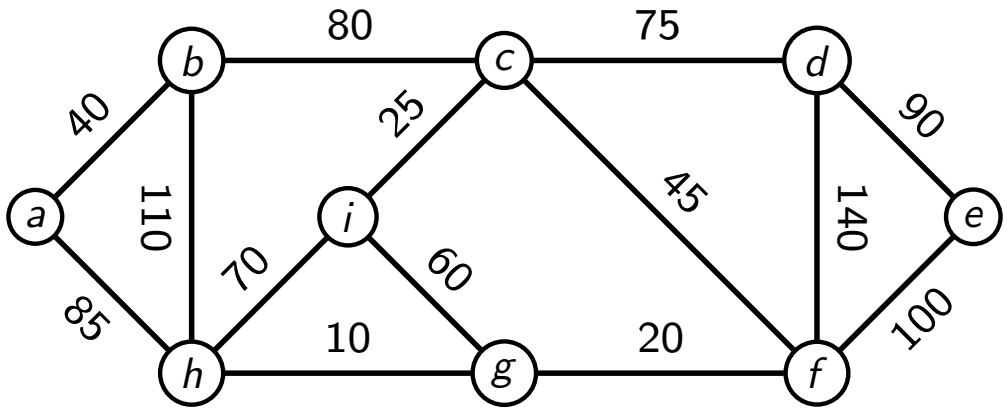
Kruskal(G, c)
Ταξινόμησε κόστη ακμών ώστε $c_1 \leq c_2 \leq \cdots \leq c_m$.
$T \leftarrow \emptyset$
Για κάθε $(u \in V)$ φτιάξε ένα σύνολο που περιέχει την u
Για $i = 1$ έως m
  $\{u, v\} = e_i$
  Εάν οι u, v περιέχονται σε διαφορετικά σύνολα
    $T \leftarrow T \cup \{e_i\}$
    ένωσε τα σύνολα/συνιστώσες που περιέχουν τις u, v
επίστρεψε T

Ψευδοκώδικας

Kruskal(G, c)
Ταξινόμησε κόστη ακμών ώστε $c_1 \leq c_2 \leq \cdots \leq c_m$.
$T \leftarrow \emptyset$
Για κάθε $(u \in V)$ φτιάξε ένα σύνολο που περιέχει την u
Για $i = 1$ έως m
  $\{u, v\} = e_i$
  Εάν οι u, v περιέχονται σε διαφορετικά σύνολα
    $T \leftarrow T \cup \{e_i\}$
    ένωσε τα σύνολα/συνιστώσες που περιέχουν τις u, v
επίστρεψε T

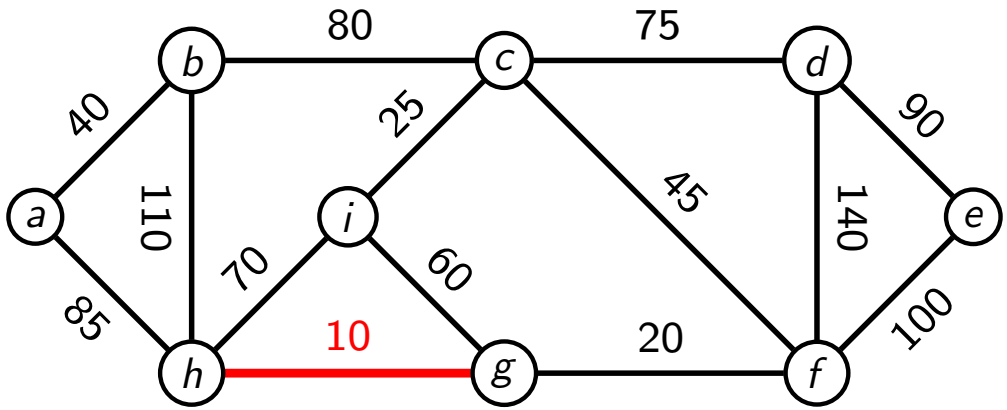
Πώς;

Μία αφελής προσέγγιση



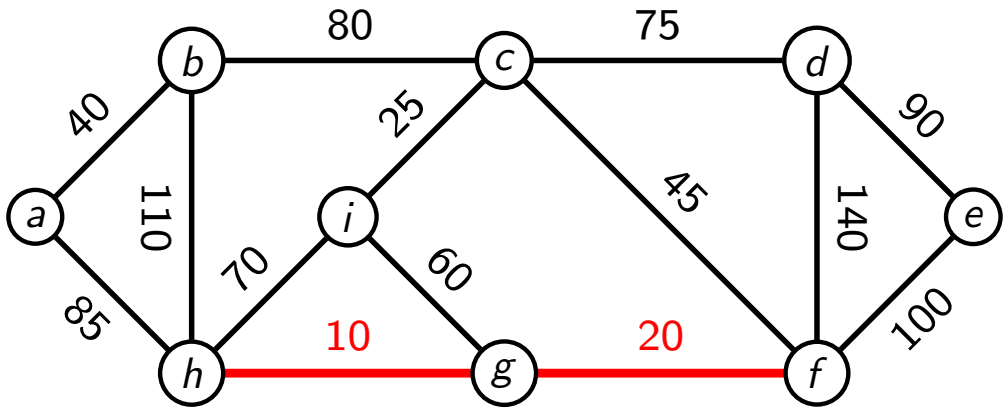
a	b	c	d	e	f	g	h	i
0	1	2	3	4	5	6	7	8

Μία αφελής προσέγγιση



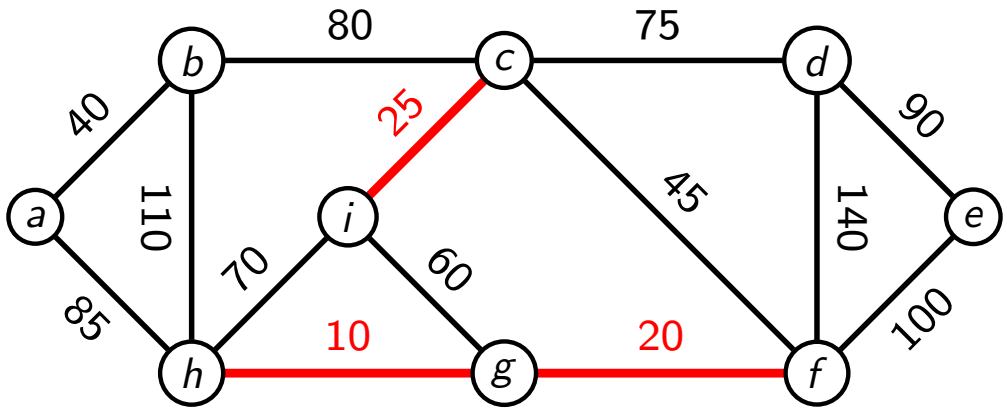
a	b	c	d	e	f	g	h	i
0	1	2	3	4	5	6	6	8

Μία αφελής προσέγγιση



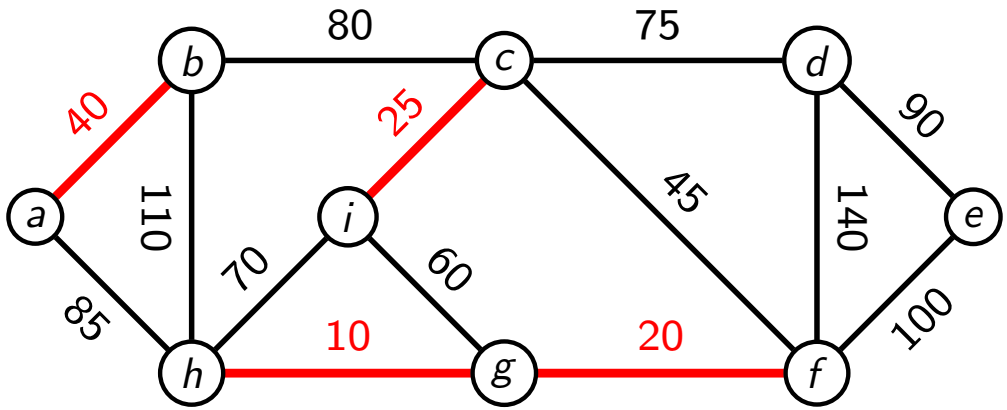
a	b	c	d	e	f	g	h	i
0	1	2	3	4	5	5	5	8

Μία αφελής προσέγγιση



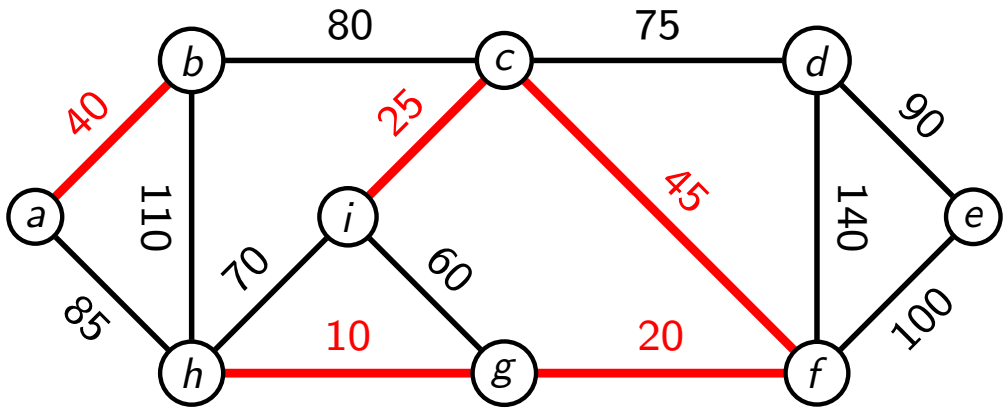
a	b	c	d	e	f	g	h	i
0	1	2	3	4	5	5	5	2

Μία αφελής προσέγγιση



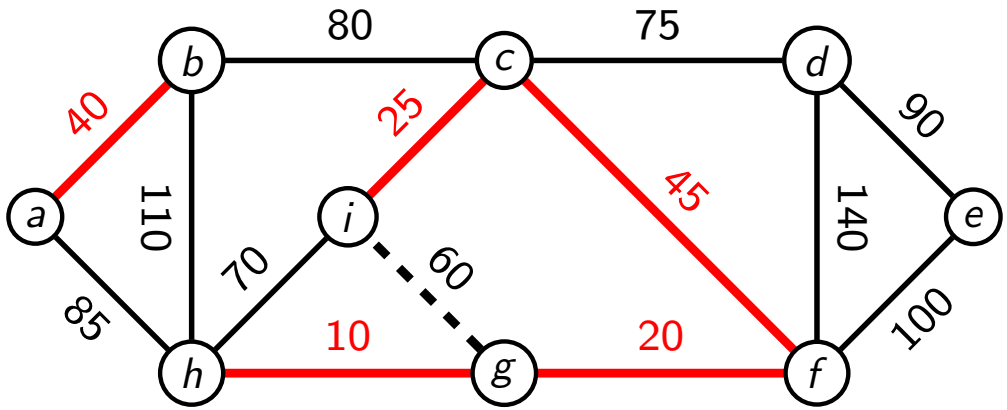
a	b	c	d	e	f	g	h	i
0	0	2	3	4	5	5	5	2

Μία αφελής προσέγγιση



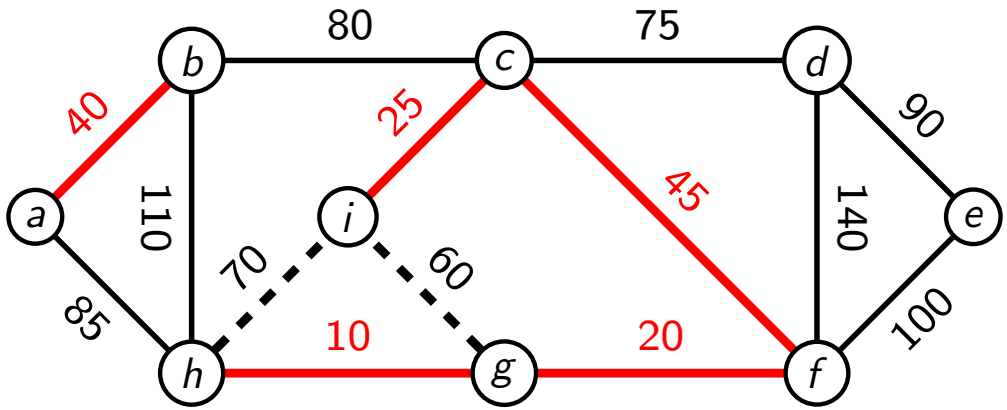
a	b	c	d	e	f	g	h	i
0	0	2	3	4	2	2	2	2

Μία αφελής προσέγγιση



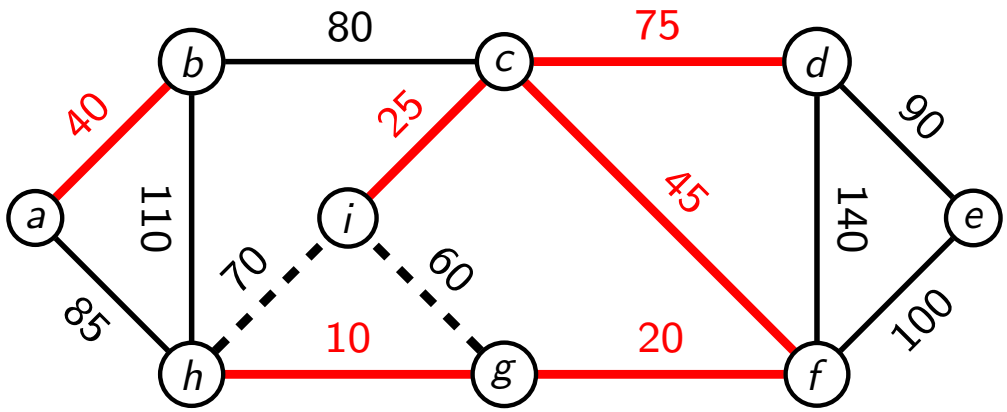
a	b	c	d	e	f	g	h	i
0	0	2	3	4	2	2	2	2

Μία αφελής προσέγγιση



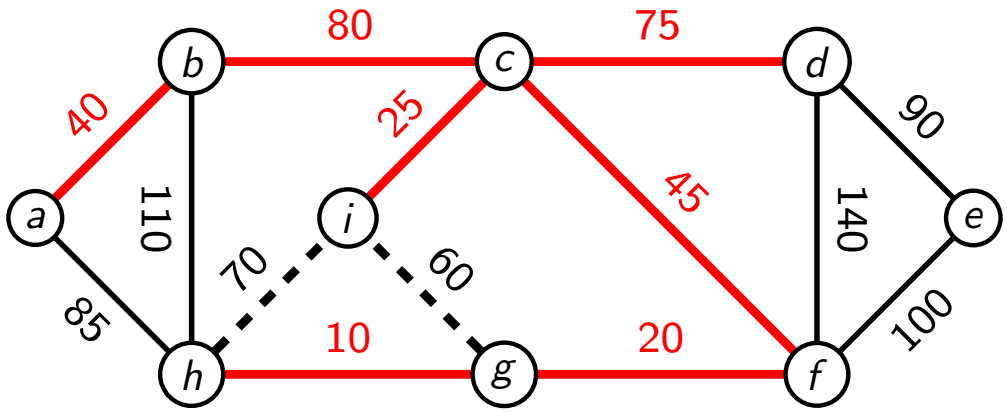
a	b	c	d	e	f	g	h	i
0	0	2	3	4	2	2	2	2

Μία αφελής προσέγγιση



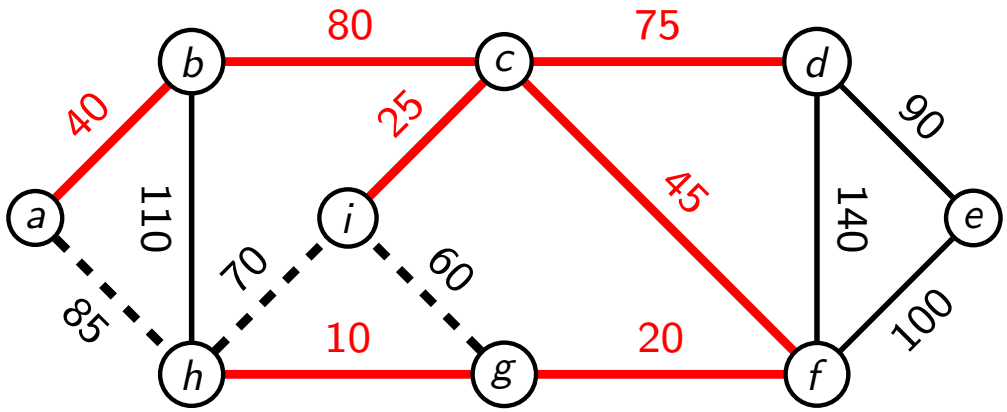
a	b	c	d	e	f	g	h	i
0	0	2	2	4	2	2	2	2

Μία αφελής προσέγγιση



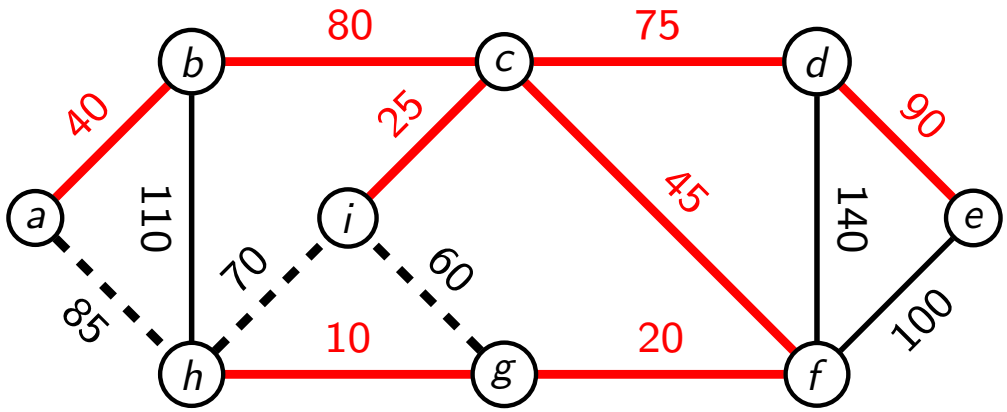
a	b	c	d	e	f	g	h	i
0	0	0	0	4	0	0	0	0

Μία αφελής προσέγγιση



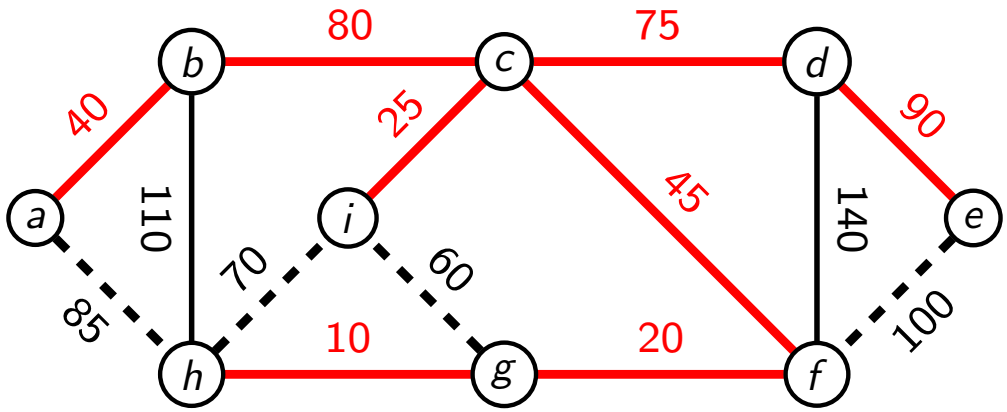
a	b	c	d	e	f	g	h	i
0	0	0	0	4	0	0	0	0

Μία αφελής προσέγγιση



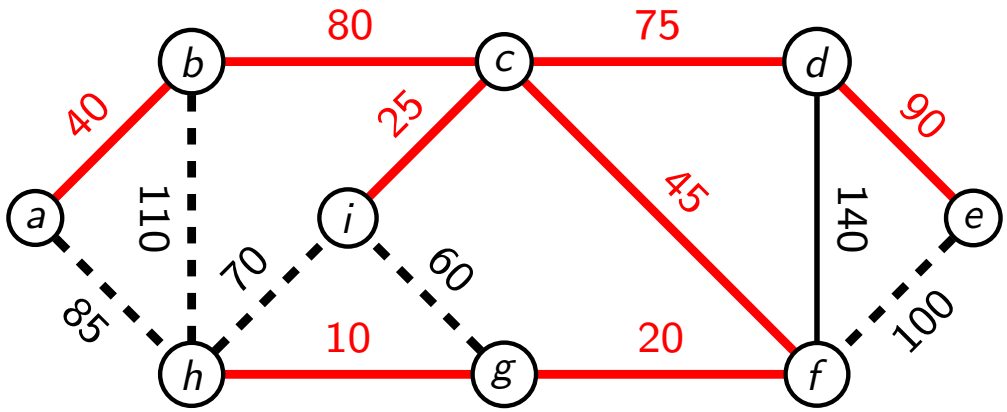
a	b	c	d	e	f	g	h	i
0	0	0	0	0	0	0	0	0

Μία αφελής προσέγγιση



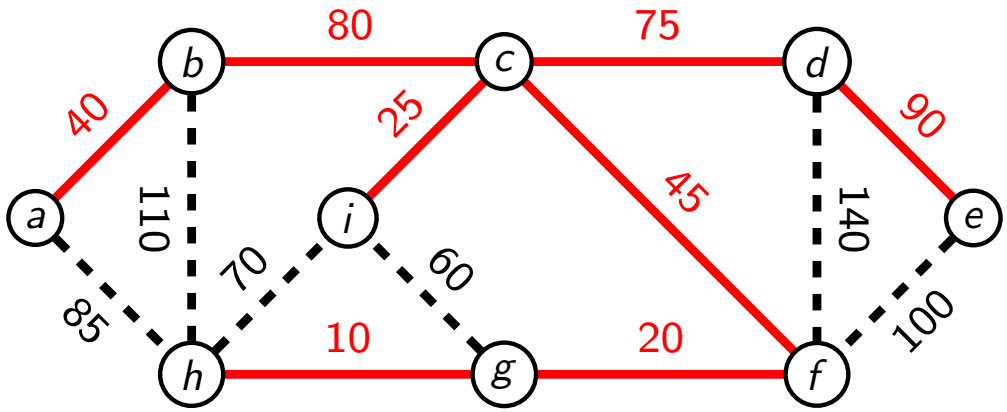
a	b	c	d	e	f	g	h	i
0	0	0	0	0	0	0	0	0

Μία αφελής προσέγγιση



a	b	c	d	e	f	g	h	i
0	0	0	0	0	0	0	0	0

Μία αφελής προσέγγιση



a	b	c	d	e	f	g	h	i
0	0	0	0	0	0	0	0	0

Δομή δεδομένων ξένων συνόλων (union-find)

- Διατηρεί μία συλλογή $\mathcal{S} = \{S_1, S_2 \ldots, S_k\}$ από (δυναμικά) σύνολα ανά δύο ξένα.

- Κάθε σύνολο έχει έναν αντιπρόσωπο ο οποίος είναι μέλος του συνόλου.

Δομή δεδομένων ξένων συνόλων (union-find)

- Διατηρεί μία συλλογή $\mathcal{S} = \{S_1, S_2 \ldots, S_k\}$ από (δυναμικά) σύνολα ανά δύο ξένα.

- Κάθε σύνολο έχει έναν αντιπρόσωπο ο οποίος είναι μέλος του συνόλου.

Μπορούμε να εκτελούμε τις παρακάτω πράξεις:

- $\mathrm{MakeSet}(x)$: κατασκευή ενός νέου συνόλου με μοναδικό μέλος (και αντιπρόσωπο) το στοιχείο x.

- $\mathrm{Union}(x, y)$: Συγχωνεύει τα δύο σύνολα που περιέχουν τα x και y σε ένα νέο σύνολο. Ο αντιπρόσωπος του νέου συνόλου είναι κάποιος από τους προηγούμενους δύο αντιπροσώπους. Μπορούμε να υποθέσουμε ότι το σύνολο του άλλου αντιπροσώπου διαγράφεται. (Υποθέτουμε ότι τα δύο σύνολα πριν ήταν ξένα).

- $\mathrm{Find}(x)$: Επιστρέφει τον αντιπρόσωπο του **μοναδικού** συνόλου που περιέχει το x.

Εφαρμογή (συνεκτικές συνιστώσες)

Συνεκτικές Συνιστώσες$(G = (V, E))$
Για κάθε κορυφή $v \in V$
  MakeSet(v)
Για κάθε ακμή $\{u, v\} \in E$
  Εάν Find$(u) \neq$ Find(v)
    Union(u, v)

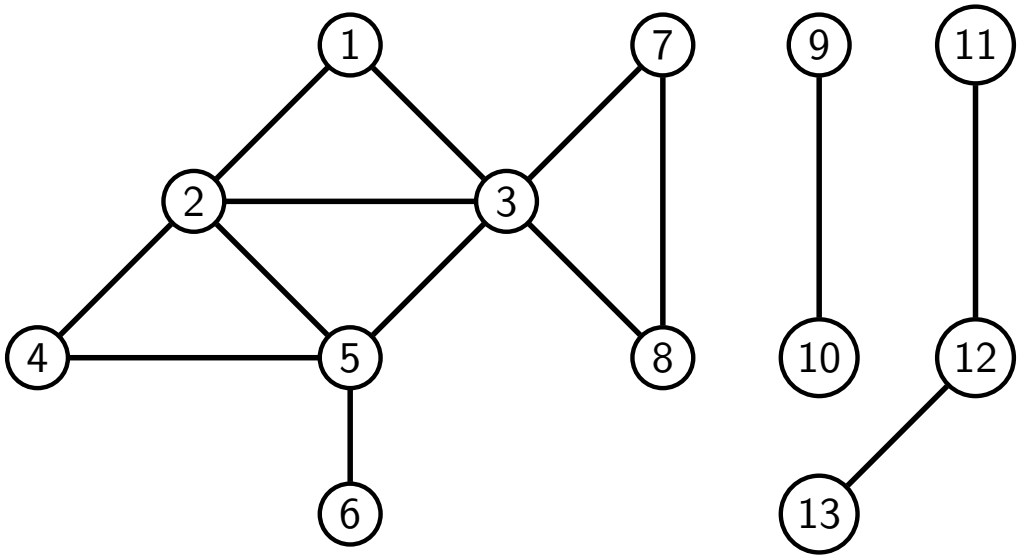
Εφαρμογή (συνεκτικές συνιστώσες)

Συνεκτικές Συνιστώσες$(G = (V, E))$
Για κάθε κορυφή $v \in V$
  MakeSet(v)
Για κάθε ακμή $\{u, v\} \in E$
  Εάν Find$(u) \neq$ Find(v)
    Union(u, v)

Τότε για δύο κορυφές x, y μπορούμε εύκολα να αποφασίσουμε αν ανήκουν στην ίδια συνεκτική συνιστώσα.

Ίδια Συνιστώσα(u, v)
Εάν Find$(u) =$ Find(v) τότε
  επίστρεψε ΑΛΗΘΕΣ
αλλιώς επίστρεψε ΨΕΥΔΕΣ

Παράδειγμα



Κατευθυνόμενα Δέντρα με Ρίζα

- Κάθε σύνολο αναπαριστάται από ένα κατευθυνόμενο δέντρο με ρίζα.

Κατευθυνόμενα Δέντρα με Ρίζα

- Κάθε σύνολο αναπαριστάται από ένα κατευθυνόμενο δέντρο με ρίζα.
- Η κορυφή κάθε δέντρου αναπαριστά ένα στοιχείο του συνόλου.

Κατευθυνόμενα Δέντρα με Ρίζα

- Κάθε σύνολο αναπαριστάται από ένα κατευθυνόμενο δέντρο με ρίζα.
- Η κορυφή κάθε δέντρου αναπαριστά ένα στοιχείο του συνόλου.
- Σε κάθε δέντρο κάθε κορυφή έχει μία μόνο εξερχόμενη ακμή προς τον γονιό της.

Κατευθυνόμενα Δέντρα με Ρίζα

- Κάθε σύνολο αναπαριστάται από ένα κατευθυνόμενο δέντρο με ρίζα.
- Η κορυφή κάθε δέντρου αναπαριστά ένα στοιχείο του συνόλου.
- Σε κάθε δέντρο κάθε κορυφή έχει μία μόνο εξερχόμενη ακμή προς τον γονιό της.
- Η ρίζα έχει μία ακμή προς τον εαυτό της (θηλιά) η κορυφή που αντιστοιχεί στη ρίζα είναι ο αντιπρόσωπος του συνόλου.

Κατευθυνόμενα Δέντρα με Ρίζα

- Κάθε σύνολο αναπαριστάται από ένα κατευθυνόμενο δέντρο με ρίζα.
- Η κορυφή κάθε δέντρου αναπαριστά ένα στοιχείο του συνόλου.
- Σε κάθε δέντρο κάθε κορυφή έχει μία μόνο εξερχόμενη ακμή προς τον γονιό της.
- Η ρίζα έχει μία ακμή προς τον εαυτό της (θηλιά) η κορυφή που αντιστοιχεί στη ρίζα είναι ο αντιπρόσωπος του συνόλου.
- Όταν έχουμε πάνω από ένα ξένα σύνολα τότε έχουμε κατευθυνόμενο δάσος με ρίζες.

Παράδειγμα

Υλοποίηση των πράξεων

- Η MakeSet(x) φτιάχνει ένα δέντρο με μία κορυφή x

Υλοποίηση των πράξεων

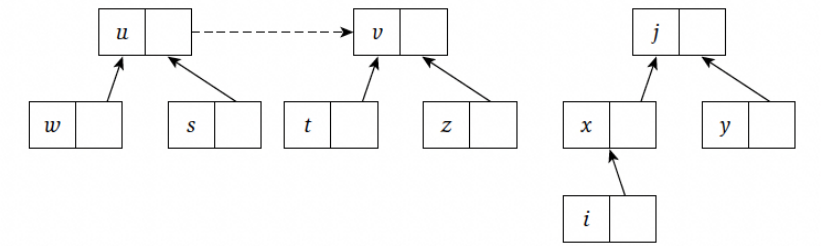
- Η $\mathrm{MakeSet}(x)$ φτιάχνει ένα δέντρο με μία κορυφή x
- Η $\mathrm{Find}(x)$ ακολουθεί το μοναπάτι ώσπου να βρει τη ρίζα του δέντρου που περιέχει την x (την κορυφή που επιστρέφει με θηλιά στον εαυτό της).

Υλοποίηση των πράξεων

- Η $\mathrm{MakeSet}(x)$ φτιάχνει ένα δέντρο με μία κορυφή x

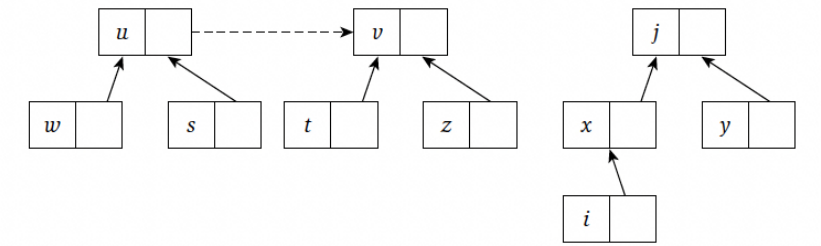
- Η $\mathrm{Find}(x)$ ακολουθεί το μοναπάτι ώσπου να βρει τη ρίζα του δέντρου που περιέχει την x (την κορυφή που επιστρέφει με θηλιά στον εαυτό της).

- Η $\mathrm{Union}(x, y)$ προσθέτει μία ακμή από τη ρίζα του ενός δέντρου στη ρίζα του άλλου αφαιρώντας παράλληλα τη θηλιά.



Υλοποίηση των πράξεων

- Η $\mathrm{MakeSet}(x)$ φτιάχνει ένα δέντρο με μία κορυφή x
- Η $\mathrm{Find}(x)$ ακολουθεί το μοναπάτι ώσπου να βρει τη ρίζα του δέντρου που περιέχει την x (την κορυφή που επιστρέφει με θηλιά στον εαυτό της).
- Η $\mathrm{Union}(x, y)$ προσθέτει μία ακμή από τη ρίζα του ενός δέντρου στη ρίζα του άλλου αφαιρώντας παράλληλα τη θηλιά.



Αν το δέντρο ενός συνόλου αντιστοιχεί σε μονοπάτι μπορεί να χρειαστεί να διατρέξουμε όλες τις κορυφές του μέχρι να βρούμε τη ρίζα.

Βάρη στα δέντρα

Για κάθε δέντρο αποθηκεύουμε το μέγεθός του και κατά το $\mathrm{Union}(x, y)$ κατευθύνουμε τη ρίζα του μικρότερου δέντρου στη ρίζα του μεγαλύτερου.

Βάρη στα δέντρα

Για κάθε δέντρο αποθηκεύουμε το μέγεθός του και κατά το $\mathrm{Union}(x, y)$ κατευθύνουμε τη ρίζα του μικρότερου δέντρου στη ρίζα του μεγαλύτερου.

Μας αρκούν $2 \log n$ βήματα για να αποφασίσουμε αν δύο από τα n στοιχεία βρίσκονται στο ίδιο σύνολο.

Βάρη στα δέντρα

Για κάθε δέντρο αποθηκεύουμε το μέγεθός του και κατά το $\mathrm{Union}(x, y)$ κατευθύνουμε τη ρίζα του μικρότερου δέντρου στη ρίζα του μεγαλύτερου.

Μας αρκούν $2 \log n$ βήματα για να αποφασίσουμε αν δύο από τα n στοιχεία βρίσκονται στο ίδιο σύνολο.

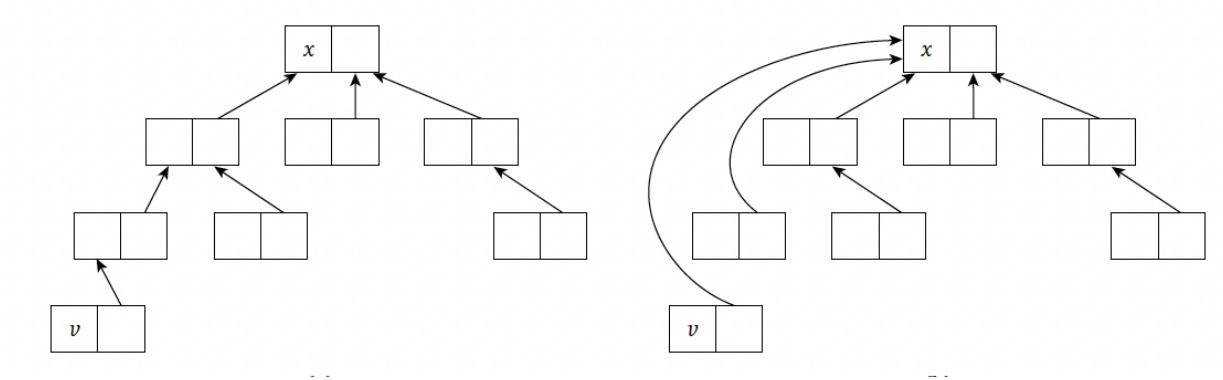
Το αποδεικνύουμε με επαγωγή δείχνοντας ότι σε ένα σύνολο με k στοιχεία μας αρκούν $\log k$ βήματα για να βρούμε τον αντιπρόσωπό του ξεκινώντας από οποιοδήποτε στοιχείο του.

Συμπίεση Μονοπατιών

Σε μία εφαρμογή της $\mathrm{Find}(v)$ για κάθε κορυφή u του μονοπατιού που συνδέει την v με τη ρίζα x συνδέουμε κατευθείαν την u με την x.

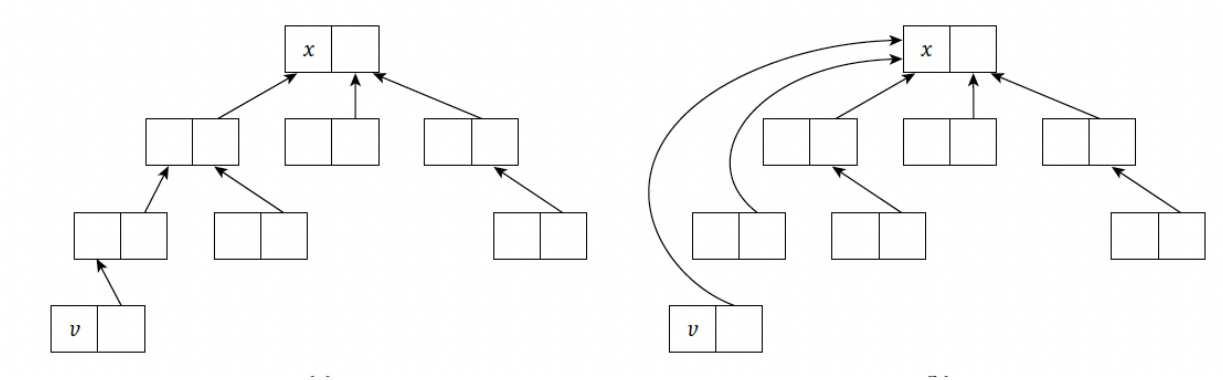
Συμπίεση Μονοπατιών

Σε μία εφαρμογή της $\mathrm{Find}(v)$ για κάθε κορυφή u του μονοπατιού που συνδέει την v με τη ρίζα x συνδέουμε κατευθείαν την u με την x.



Συμπίεση Μονοπατιών

Σε μία εφαρμογή της $\mathrm{Find}(v)$ για κάθε κορυφή u του μονοπατιού που συνδέει την v με τη ρίζα x συνδέουμε κατευθείαν την u με την x.



Πετυχαίνουμε να μικρύνουμε το ύψος των δέντρων χωρίς να αλλάξουμε τα βάρη.

Υλοποίηση

Χρησιμοποιούμε την δομή δεδομένων union-find.

Υλοποίηση

Χρησιμοποιούμε την δομή δεδομένων union-find.

- Στόχος: σύνολο με τις ακμές στο ΕΕΔ T.

Υλοποίηση

Χρησιμοποιούμε την δομή δεδομένων union-find.

- Στόχος: σύνολο με τις ακμές στο ΕΕΔ T.

- Διατηρούμε ένα σύνολο για κάθε συνεκτική συνιστώσα του T.

Υλοποίηση

Χρησιμοποιούμε την δομή δεδομένων union-find.

- Στόχος: σύνολο με τις ακμές στο ΕΕΔ T.
- Διατηρούμε ένα σύνολο για κάθε συνεκτική συνιστώσα του T.
- $\mathcal{O}(m \log n)$ για ταξινόμηση, $\mathcal{O}(m\alpha(n))$ για εύρεση της ένωσης.