Αλγόριθμοι και Πολυπλοκότητα

Άπληστοι Αλγόριθμοι

Δημήτρης Μιχαήλ

Τμήμα Πληροφορικής και Τηλεματικής
Χαροκόπειο Πανεπιστήμιο

Άπληστοι Αλγόριθμοι

Είναι δύσκολο να ορίσουμε ακριβώς την έννοια του άπληστου αλγόριθμου.

Γενικά ένας άπληστος αλγόριθμος (greedy algorithm) κατασκευάζει μια λύση τμηματικά επιλέγοντας πάντοτε το επόμενο τμήμα το οποίο προσφέρει το πιο προφανές και άμεσο όφελος.

Για κάποια προβλήματα αυτή η στρατηγική είναι καταστροφική αλλά για άλλα είναι βέλτιστη.

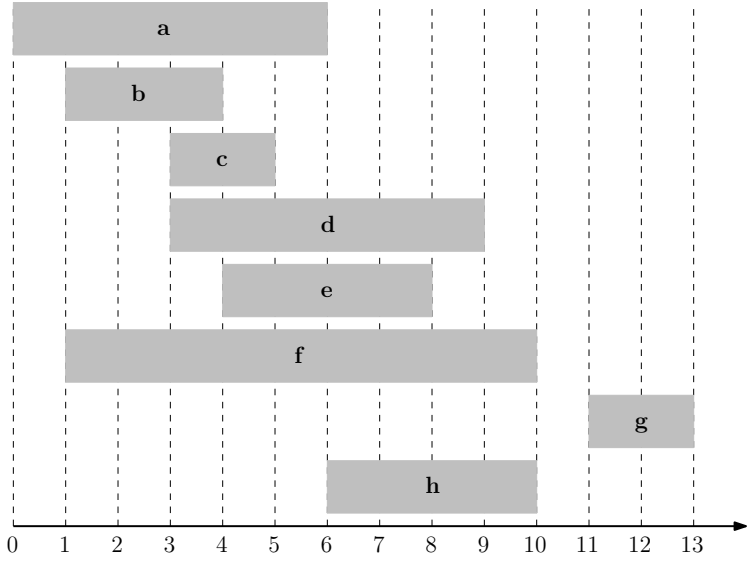
Χρονοπρογραμματισμός Διαστημάτων

Είσοδος.

- Σύνολο αιτημάτων $\{1, 2, \ldots, n\}$.
- Αίτημα i ξεκινά την χρονική στιγμή $s(i)$ και τερματίζει την $f(i)$.

Έξοδος.
Το μέγιστο συμβατό υποσύνολο αιτημάτων, όπου ένα υποσύνολο είναι συμβατό εαν ανά δύο τα αιτήματα δεν επικαλύπτονται χρονικά.

Χρονοπρογραμματισμός Διαστημάτων



Χρονοπρογραμματισμός Διαστημάτων

Ένας άπληστος αλγόριθμος για το πρόβλημα χρονοπρογραμματισμού διαστημάτων χρησιμοποιεί έναν απλό κανόνα για να κάνει επιλογή αιτημάτων με βάση την λογική που ακολουθεί.

1. επιλέγει ένα αίτημα i_1
2. διαγράφει όλα τα αιτήματα που δεν είναι συμβατά με το i_1
3. κατόπιν επιλέγει ένα άλλο αίτημα i_2
4. διαγράφει όλα τα αιτήματα που δεν είναι συμβατά με το i_2
5. συνεχίζει έτσι μέχρι να τελειώσουν τα δυνατά αιτήματα

Χρονοπρογραμματισμός Διαστημάτων

Ένας άπληστος αλγόριθμος για το πρόβλημα χρονοπρογραμματισμού διαστημάτων χρησιμοποιεί έναν απλό κανόνα για να κάνει επιλογή αιτημάτων με βάση την λογική που ακολουθεί.

1. επιλέγει ένα αίτημα i_1
2. διαγράφει όλα τα αιτήματα που δεν είναι συμβατά με το i_1
3. κατόπιν επιλέγει ένα άλλο αίτημα i_2
4. διαγράφει όλα τα αιτήματα που δεν είναι συμβατά με το i_2
5. συνεχίζει έτσι μέχρι να τελειώσουν τα δυνατά αιτήματα

Δεν μπορεί να πάρει πίσω μια επιλογή αιτήματος.

Χρονοπρογραμματισμός Διαστημάτων

Μπορούμε να παρουσιάσουμε τον ίδιο αλγόριθμο ως εξής:

1. ταξινόμησε τα αιτήματα με βάση κάποιο κριτήριο
2. διάλεξε ένα αίτημα εαν είναι συμβατό με τα αιτήματα που έχεις ήδη διαλέξει

Χρονοπρογραμματισμός Διαστημάτων

Μπορούμε να παρουσιάσουμε τον ίδιο αλγόριθμο ως εξής:

1. ταξινόμησε τα αιτήματα με βάση κάποιο κριτήριο
2. διάλεξε ένα αίτημα εαν είναι συμβατό με τα αιτήματα που έχεις ήδη διαλέξει

Λειτουργεί σωστά ένας τέτοιος αλγόριθμος;

Χρονοπρογραμματισμός Διαστημάτων

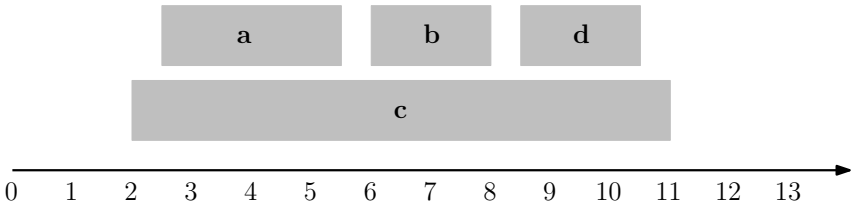
Μπορούμε να παρουσιάσουμε τον ίδιο αλγόριθμο ως εξής:

1. ταξινόμησε τα αιτήματα με βάση κάποιο κριτήριο
2. διάλεξε ένα αίτημα εαν είναι συμβατό με τα αιτήματα που έχεις ήδη διαλέξει

Λειτουργεί σωστά ένας τέτοιος αλγόριθμος;

Για να απαντήσουμε πρέπει καταρχήν να ορίσουμε τον κανόνα επιλογής των αιτημάτων.

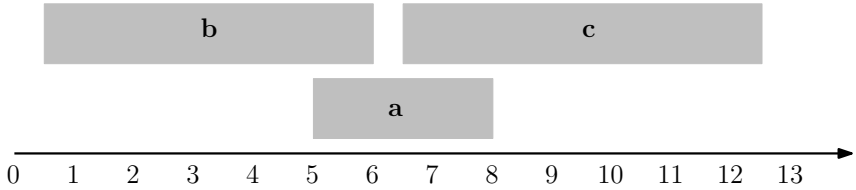
Φανταστείτε τον αλγόριθμο που επιλέγει πρώτα το αίτημα που ξεκινά πιο νωρίς.



Ο αλγόριθμος θα επιλέξει πρώτα το αίτημα **c** ενώ η καλύτερη λύση είναι **a**, **b**, και **d**.

Ο αλγόριθμος δεν είναι βέλτιστος!

Έστω ο αλγόριθμος που επιλέγει πρώτα το αίτημα που έχει την μικρότερη διάρκεια.



Ο αλγόριθμος θα επιλέξει πρώτα το αίτημα **a** ενώ η καλύτερη λύση είναι **b**, **c**.

Ο αλγόριθμος δεν είναι βέλτιστος!

Χρονοπρογραμματισμός Διαστημάτων

Αίτημα όπου $f(\cdot)$ είναι ελάχιστο

Έστω ο αλγόριθμος που επιλέγει πρώτα το αίτημα που τελειώνει πρώτο, δηλαδή το αίτημα i όπου ο χρόνος $f(i)$ είναι όσο το δυνατόν μικρότερος.

1. διασφαλίζει ότι οι πόροι μας θα απελευθερωθούν όσο το δυνατόν πιο σύντομα
2. μεγιστοποιεί τον χρόνο που απομένει για την ικανοποίηση των άλλων αιτημάτων
3. θα αποδείξουμε πως βρίσκει την βέλτιστη λύση

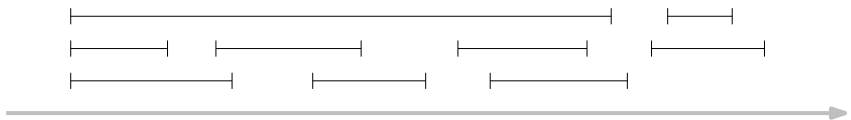
Χρονοπρογραμματισμός Διαστημάτων

Αίτημα όπου $f(\cdot)$ είναι ελάχιστο

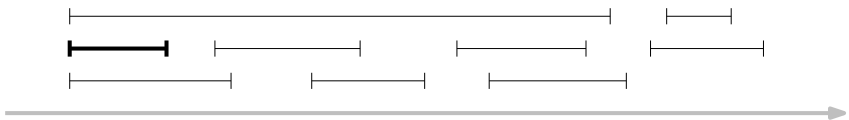
Συντομότερος χρόνος ολοκλήρωσης

Έστω R το σύνολο όλων των αιτημάτων και A κενό
while *R δεν είναι κενό* **do**
    Διάλεξε ένα αίτημα $i \in R$ που έχει το συντομότερο χρόνο ολοκλήρωσης
    Πρόσθεσε το i στο σύνολο A
    Διάγραψε από το R το αίτημα i και όλα τα αιτήματα που δεν είναι συμβατά με το i
end
return *το σύνολο A*

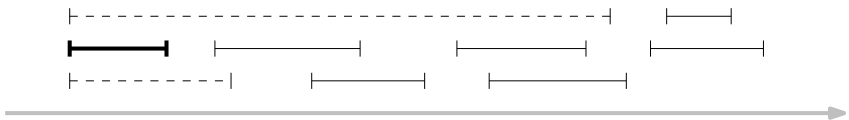
Παράδειγμα



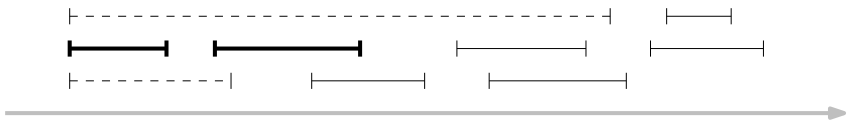
Παράδειγμα



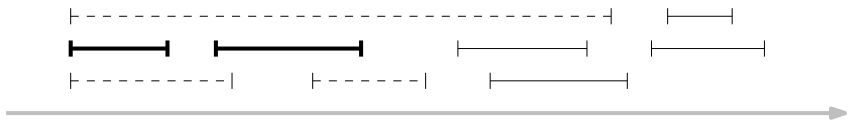
Παράδειγμα



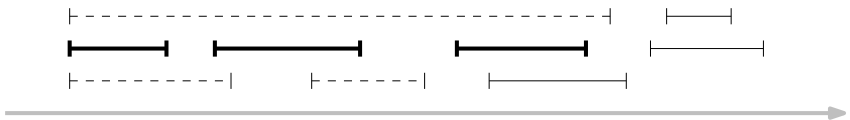
Παράδειγμα



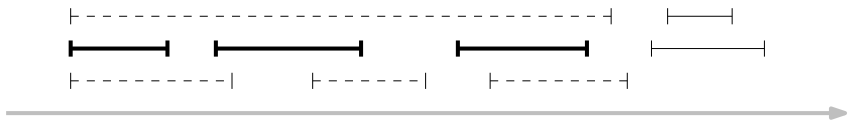
Παράδειγμα



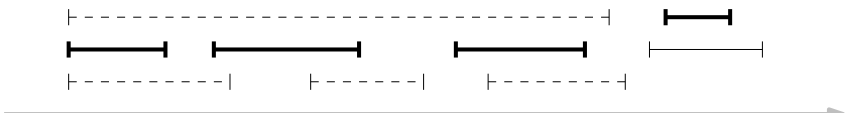
Παράδειγμα



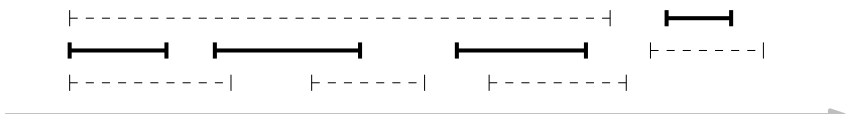
Παράδειγμα



Παράδειγμα



Παράδειγμα



Πρέπει να αποδείξουμε πως ο αλγόριθμος αυτός βρίσκει όντως την βέλτιστη λύση.

Ορθότητα. Καταρχήν είναι προφανές πως ο αλγόριθμος επιστρέφει ένα σύνολο αιτημάτων που είναι συμβατά.

Απομένει να δείξουμε πως η λύση είναι βέλτιστη. Έστω λοιπόν για λόγους σύγκρισης ότι O είναι ένα βέλτιστο σύνολο χρονικών διαστημάτων.

1. είναι υπερβολή να προσπαθήσουμε να αποδείξουμε πως $A = O$ μιας και μπορεί να υπάρχουν πολλές βέλτιστες λύσεις
2. αρκεί να δείξουμε πως $|A| = |O|$

Λύση αλγορίθμου. Έστω $i_1, \ldots, i_k$ το σύνολο αιτημάτων στο A με τη σειρά με την οποία προστέθηκαν στο A.

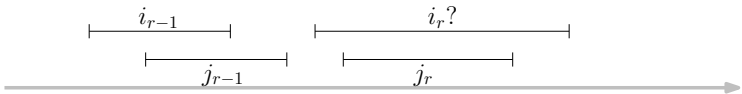
Βέλτιστη λύση. Έστω $j_1, \ldots, j_m$ το σύνολο αιτημάτων στο O. Υποθέστε πως είναι ταξινομημένα με τη σειρά των χρονικών σημείων έναρξης και τέλους (λόγω συμβατότητας τα σημεία έναρξης έχουν την ίδια σειρά με τα σημεία τέλους).

Θέλουμε να δείξουμε πως $k = m$.

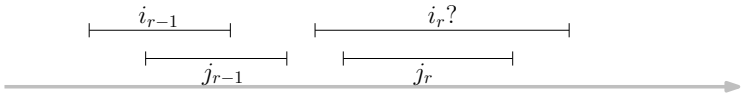
Λήμμα
Για όλους τους δείκτες $r \leq k$ ισχύει $f(i_r) \leq f(j_r)$.

Θα χρησιμοποιήσουμε επαγωγή. Για $r = 1$ ισχύει μιας και ο αλγόριθμος διαλέγει το αίτημα i_1 με τον ελάχιστο χρόνο τερματισμού.

Έστω τώρα ένα $r > 1$ και ας θεωρήσουμε πως ότι η πρόταση είναι αληθής για $r - 1$. Ξέρουμε δηλαδή πως $f(i_{r-1}) \leq f(j_{r-1})$.



Είναι δυνατόν να συμβεί αυτό που δείχνει το παραπάνω σχήμα;



Όχι γιατι ο αλγόριθμος μας θα είχε διαλέξει το j_r.

Πιο τυπικά μπορούμε να πούμε πως

- $f(j_{r-1}) \leq s(j_r)$ αφού το O έχει συμβατά διαστήματα
- $f(i_{r-1}) \leq f(j_{r-1})$ από την επαγωγική υπόθεση

και άρα $f(i_{r-1}) \leq s(j_r)$.

Με άλλα λόγια το διάστημα j_r ανήκει στο σύνολο R των διαθέσιμων χρονικών διαστημάτων την ώρα που ο άπληστος αλγόριθμος επιλέγει το διάστημα i_r. Επειδή ο αλγόριθμος διαλέγει πάντα το μικρότερο χρόνο τερματισμού έχουμε πως $f(i_r) \leq f(j_r)$.

Θεώρημα

Ο άπληστος αλγόριθμος επιστρέφει ένα βέλτιστο σύνολο A.

Θεώρημα

Ο άπληστος αλγόριθμος επιστρέφει ένα βέλτιστο σύνολο A.

Απόδειξη

Θα χρησιμοποιήσουμε εις άτοπον απαγωγή.

□

Θεώρημα

Ο άπληστος αλγόριθμος επιστρέφει ένα βέλτιστο σύνολο A.

Απόδειξη

Θα χρησιμοποιήσουμε εις άτοπον απαγωγή.

Αν το A δεν είναι βέλτιστο, τότε ένα βέλτιστο σύνολο O πρέπει να έχει περισσότερα αιτήματα, δηλαδή, πρέπει $m > k$.

□

Θεώρημα

Ο άπληστος αλγόριθμος επιστρέφει ένα βέλτιστο σύνολο A.

Απόδειξη

Θα χρησιμοποιήσουμε εις άτοπον απαγωγή.

Αν το A δεν είναι βέλτιστο, τότε ένα βέλτιστο σύνολο O πρέπει να έχει περισσότερα αιτήματα, δηλαδή, πρέπει $m > k$.

Χρησιμοποιώντας το προηγούμενο Λήμμα για $r = k$ έχουμε πως $f(i_k) \leq f(j_k)$.

□

Θεώρημα

Ο άπληστος αλγόριθμος επιστρέφει ένα βέλτιστο σύνολο A.

Απόδειξη

Θα χρησιμοποιήσουμε εις άτοπον απαγωγή.

Αν το A δεν είναι βέλτιστο, τότε ένα βέλτιστο σύνολο O πρέπει να έχει περισσότερα αιτήματα, δηλαδή, πρέπει $m > k$.

Χρησιμοποιώντας το προηγούμενο Λήμμα για $r = k$ έχουμε πως $f(i_k) \leq f(j_k)$.

Αφού $m > k$ υπάρχει ένα αίτημα j_{k+1} στο σύνολο O. Ξέρουμε όμως πως

- $f(j_k) \leq s(j_{k+1})$ και
- $f(i_k) \leq f(j_k)$

και άρα το αίτημα j_{k+1} είναι διαθέσιμο όταν ο άπληστος αλγόριθμος τερματίζει. Άτοπο. □

Αρχικοποίηση.

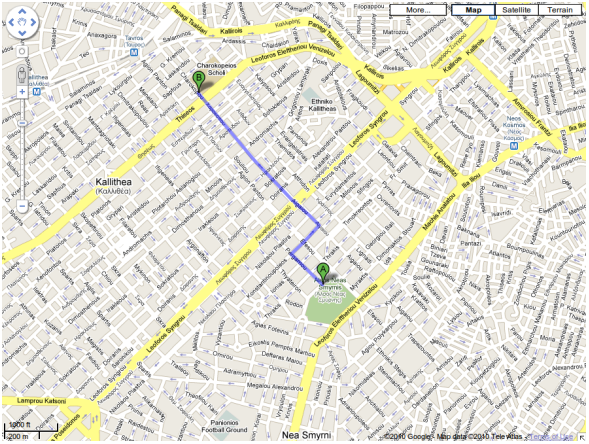
1. ταξινόμησε τα n αιτήματα κατά σειρά χρόνου τερματισμού ($f(i) < f(j)$ όταν $i < j$)
2. κατασκεύασε πίνακα $S[1 \ldots n]$ όπου το $S[i]$ περιέχει την τιμή $s(i)$

Αλγόριθμος. Επιλέγουμε αιτήματα με επεξεργασία των χρονικών διαστημάτων κατά αύξουσα σειρά ως προς το $f(i)$.

1. επέλεξε το πρώτο διάστημα
2. διατρέχουμε τα διαστήματα με τη σειρά μέχρι να φτάσουμε το πρώτο χρονικό διάστημα j για το οποίο $s(j) \geq f(1)$ και το επιλέγουμε
3. αν το πιο πρόσφατο διάστημα που έχουμε επιλέξει τελειώνει τη χρονική στιγμή f, διατρέχουμε τα διαστήματα μέχρι να φτάσουμε στο πρώτο j για το οποίο $s(j) \geq f$

Συντομότερες Διαδρομές σε Γραφήματα

Τα γραφήματα χρησιμοποιούνται για να μοντελοποιήσουν δίκτυα. Κατά συνέπεια, ένα βασικό αλγοριθμικό πρόβλημα είναι ο προσδιορισμός της συντομότερης διαδρομής μεταξύ των κόμβων ενός γραφήματος.



Συντομότερες Διαδρομές σε Γραφήματα

Το πρόβλημα.
Μας δίνεται ένα κατευθυνόμενο γράφημα $G(V, E)$ όπου

- έχουμε ένα καθορισμένο κόμβο εκκίνησης $s \in V$,
- κάθε ακμή $e \in E$ έχει μήκος $l(e) \geq 0$, το οποίο δηλώνει το χρόνο (ή την απόσταση ή το κόστος) που απαιτείται για τη διάσχιση της ακμής e,
- για μια διαδρομή P, το μήκος της P, είναι το άθροισμα των μηκών όλων των ακμών της και συμβολίζεται ως $l(P)$.

Σκοπός είναι να προσδιορίσουμε τη συντομότερη διαδρομή από τον s προς οποιονδήποτε άλλο κόμβο του γραφήματος.

Ο Αλγόριθμος του Dijkstra

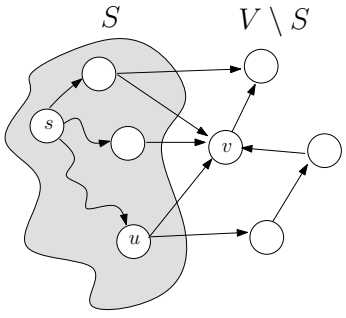
Το 1959 ο Edsger Dijkstra πρότεινε ένα πολύ απλό άπληστο αλγόριθμο για την επίλυση του προβλήματος των συντομότερων διαδρομών μιας αφετηρίας (single-source shortest paths).

Ο Αλγόριθμος του Dijkstra

Ο αλγόριθμος διατηρεί ένα σύνολο κορυφών S, όπου για κάθε $u \in S$ η απόσταση της συντομότερης διαδρομής $d(u)$ είναι γνωστή.

Για κάθε κόμβο $v \in V \setminus S$ προσδιορίζουμε τη συντομότερη διαδρομή που μπορεί να κατασκευαστεί αν διατρέξουμε μια διαδρομή μέσω του εξερευνημένου τμήματος S προς κάποιο κόμβο $u \in S$, και στην συνέχεια ακολουθήσουμε την ακμή (u, v).

Ο κόμβος που πετυχαίνει την συντομότερη διαδρομή προστίθεται στο σύνολο S.



Ο Αλγόριθμος του Dijkstra

Αλγόριθμος του Dijkstra

$S = \{s\}$
$d(s) = 0$
while *υπάρχουν ακμές από κόμβους του S σε κόμβους του $V \setminus S$* **do**
    Βρες τον κόμβο $v \in V \setminus S$ που ελαχιστοποιεί την ποσότητα

$$d'(v) = \min_{e=(u,v):u \in S} d(u) + l(e)$$

    Θέσε $d(v) = d'(v)$ και πρόσθεσε τον v στο σύνολο S.
end

Ο Αλγόριθμος του Dijkstra

Αλγόριθμος του Dijkstra

$S = \{s\}$
$d(s) = 0$
while *υπάρχουν ακμές από κόμβους του S σε κόμβους του $V \setminus S$* **do**
    Βρες τον κόμβο $v \in V \setminus S$ που ελαχιστοποιεί την ποσότητα

$$d'(v) = \min_{e=(u,v):u \in S} d(u) + l(e)$$

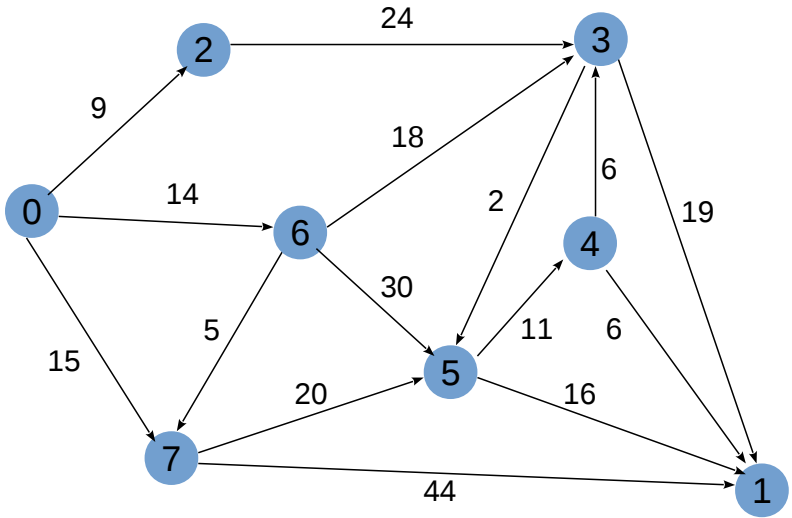
    Θέσε $d(v) = d'(v)$ και πρόσθεσε τον v στο σύνολο S.
end

Ο αλγόριθμος θεωρείται άπληστος αφού κάθε φορά που αυξάνει το μέγεθος του συνόλου S διαλέγει τον κόμβο που βρίσκεται πιο κοντά.

Ο Αλγόριθμος του Dijkstra

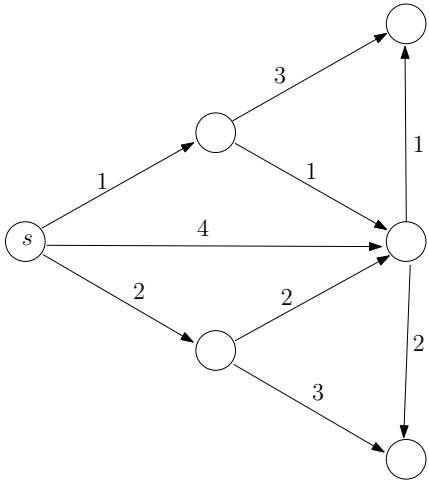
Άσκηση

Εκτελέστε τον αλγόριθμο στο παρακάτω γράφημα με αρχικό κόμβο $s = 0$.



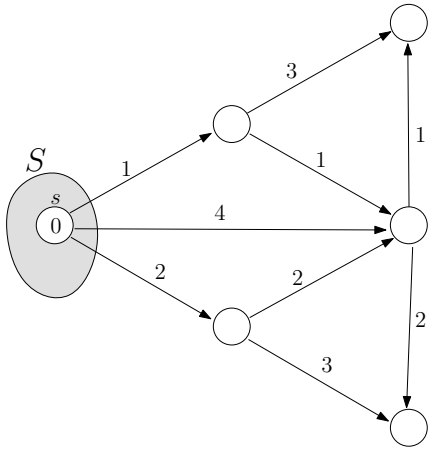
Ο Αλγόριθμος του Dijkstra

Παράδειγμα



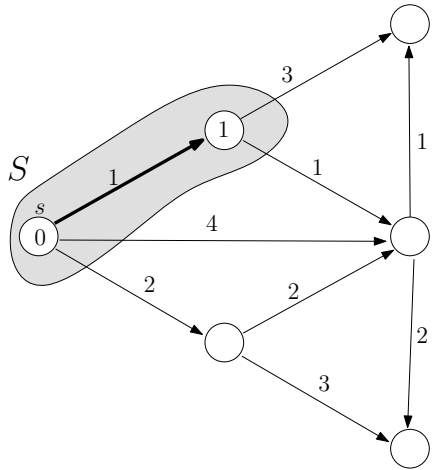
Ο Αλγόριθμος του Dijkstra

Παράδειγμα



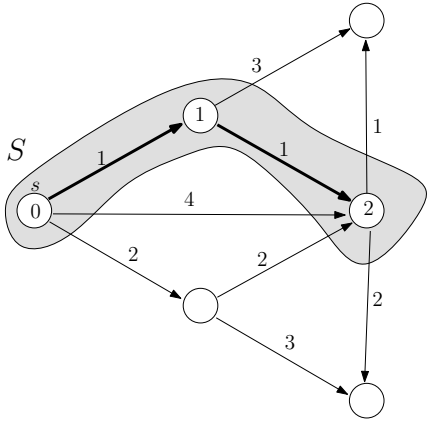
Ο Αλγόριθμος του Dijkstra

Παράδειγμα



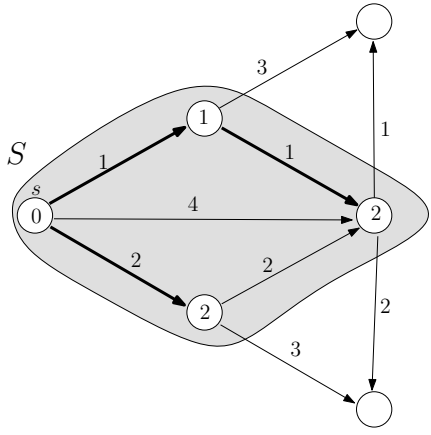
Ο Αλγόριθμος του Dijkstra

Παράδειγμα



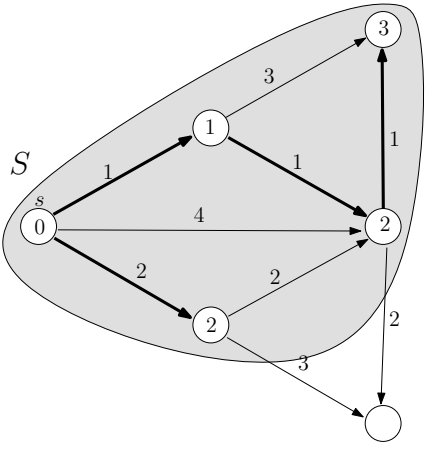
Ο Αλγόριθμος του Dijkstra

Παράδειγμα



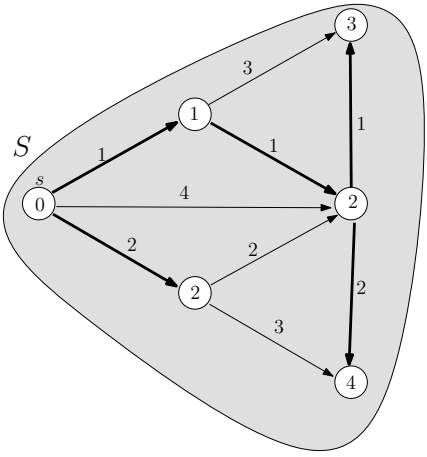
Ο Αλγόριθμος του Dijkstra

Παράδειγμα



Ο Αλγόριθμος του Dijkstra

Παράδειγμα



Αλγόριθμος του Dijkstra

$S = \{s\}$
$d(s) = 0$
while *υπάρχουν ακμές από κόμβους του* S *σε κόμβους του* $V \setminus S$ **do**
Βρες τον κόμβο $v \in V \setminus S$ που ελαχιστοποιεί την ποσότητα

$$d'(v) = \min_{e=(u,v):u\in S} d(u) + l(e)$$

Θέσε $d(v) = d'(v)$ και πρόσθεσε τον v στο σύνολο S.
end

Κάθε φορά που βάζουμε ένα κόμβο v στο σύνολο S κρατάμε την ακμή που χρησιμοποιήσαμε ώστε να σχηματίσουμε ένα δέντρο ελάχιστων διαδρομών.

Έστω για ένα κόμβο $v \in S$ η ακμή (u, v) που χρησιμοποιήσαμε. Το μονοπάτι P_v ορίζεται αναδρομικά ως η διαδρομή P_u ακολουθούμενη από την ακμή (u, v).

Ορθότητα

Λήμμα

Έστω το σύνολο S σε κάθε σημείο εκτέλεσης του αλγορίθμου. Για κάθε $u \in S$, η διαδρομή P_u είναι η συντομότερη διαδρομή $s \rightsquigarrow u$.

Ορθότητα

Λήμμα

Έστω το σύνολο S σε κάθε σημείο εκτέλεσης του αλγορίθμου. Για κάθε $u \in S$, η διαδρομή P_u είναι η συντομότερη διαδρομή $s - u$.

Απόδειξη

Θα χρησιμοποιήσουμε επαγωγή ως προς το μέγεθος του S. Η περίπτωση $|S| = 1$ είναι εύκολη αφού $S = \{s\}$ και $d(s) = 0$.

□

Ορθότητα

Λήμμα

Έστω το σύνολο S σε κάθε σημείο εκτέλεσης του αλγορίθμου. Για κάθε $u \in S$, η διαδρομή P_u είναι η συντομότερη διαδρομή $s - u$.

Απόδειξη

Θα χρησιμοποιήσουμε επαγωγή ως προς το μέγεθος του S. Η περίπτωση $|S| = 1$ είναι εύκολη αφού $S = \{s\}$ και $d(s) = 0$.

Έστω πως ισχύει όταν $|S| = k$ για κάποια τιμή $k \geq 1$. Αυξάνουμε τώρα το μέγεθος του S με την προσθήκη του κόμβου v. Έστω ότι (u, v) είναι η τελική ακμή της διαδρομής $s - v\, P_v$.

□

Ορθότητα

Λήμμα

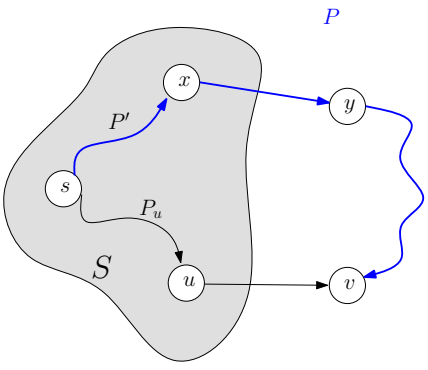
Έστω το σύνολο S σε κάθε σημείο εκτέλεσης του αλγορίθμου. Για κάθε $u \in S$, η διαδρομή P_u είναι η συντομότερη διαδρομή $s - u$.

Απόδειξη

Θα χρησιμοποιήσουμε επαγωγή ως προς το μέγεθος του S. Η περίπτωση $|S| = 1$ είναι εύκολη αφού $S = \{s\}$ και $d(s) = 0$.

Έστω πως ισχύει όταν $|S| = k$ για κάποια τιμή $k \geq 1$. Αυξάνουμε τώρα το μέγεθος του S με την προσθήκη του κόμβου v. Έστω ότι (u, v) είναι η τελική ακμή της διαδρομής $s - v$ P_v.

Θα δείξουμε πως οποιαδήποτε άλλη διαδρομή P από το s στο v έχει τουλάχιστον το ίδιο μήκος με την P_v. Η διαδρομή P για να φτάσει στο v πρέπει να αφήσει κάπου το σύνολο S. Έστω y ο πρώτος κόμβος εκτός S και x ο αμέσως προηγούμενος.



□

Ορθότητα

Λήμμα

Έστω το σύνολο S σε κάθε σημείο εκτέλεσης του αλγορίθμου. Για κάθε $u \in S$, η διαδρομή P_u είναι η συντομότερη διαδρομή $s - u$.

Απόδειξη

Θα χρησιμοποιήσουμε επαγωγή ως προς το μέγεθος του S. Η περίπτωση $|S| = 1$ είναι εύκολη αφού $S = \{s\}$ και $d(s) = 0$.

Έστω πως ισχύει όταν $|S| = k$ για κάποια τιμή $k \geq 1$. Αυξάνουμε τώρα το μέγεθος του S με την προσθήκη του κόμβου v. Έστω ότι (u, v) είναι η τελική ακμή της διαδρομής $s - v$ P_v.

Θα δείξουμε πως οποιαδήποτε άλλη διαδρομή P από το s στο v έχει τουλάχιστον το ίδιο μήκος με την P_v. Η διαδρομή P για να φτάσει στο v πρέπει να αφήσει κάπου το σύνολο S. Έστω y ο πρώτος κόμβος εκτός S και x ο αμέσως προηγούμενος.

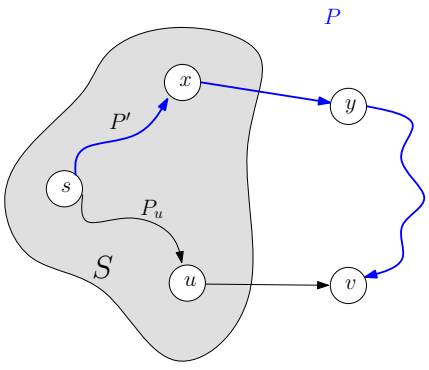
$$l(P) \geq l(P') + l(x, y) \geq d(x) + l(x, y) \geq d'(y) \geq d'(v) = l(P_v)$$

μη-αρνητικά βάρη

επαγωγή

ορισμός $d'(y)$

επιλογή αλγορίθμου



□

Αλγόριθμος του Dijkstra

$S = \{s\}$
$d(s) = 0$
while *υπάρχουν ακμές από κόμβους του* S *σε κόμβους του* $V \setminus S$ **do**
  Βρες τον κόμβο $v \in V \setminus S$ που ελαχιστοποιεί την ποσότητα

$$d'(v) = \min_{e=(u,v):u \in S} d(u) + l(e)$$

  Θέσε $d(v) = d'(v)$ και πρόσθεσε τον v στο σύνολο S.
end

Μια άμεση υλοποίηση των παραπάνω χρειάζεται χρόνο $\mathcal{O}(nm)$ αφού για κάθε κόμβο πρέπει να βρεί την ακμή που δίνει την ελάχιστη τιμή.

1. Οι τιμές των ελάχιστων ποσοτήτων

$$d'(v) = \min_{e=(u,v):u\in S} d(u) + l(e)$$

για κάθε κόμβο $v \in V \setminus S$ δεν είναι ανάγκη να υπολογίζονται από την αρχή.

Μπορούμε να τις έχουμε αποθηκευμένες σε ένα πίνακα και κάθε φορά που προσθέτουμε έναν κόμβο στο S να ανανεώνουμε αυτές που πρέπει.

2. Για να βρούμε τον κόμβο με την ελάχιστη τέτοια τιμή θα χρησιμοποιήσουμε μια ουρά προτεραιότητας.

Μια ουρά προτεραιότητας είναι μια δομή δεδομένων που διατηρεί ένα σύνολο στοιχείων S, όπου κάθε στοιχείο $v \in S$ έχει μια συσχετισμένη τιμή $key(v)$ που υποδηλώνει την προτεραιότητα του στοιχείου v.

Μικρότερα κλειδιά αντιπροσωπεύουν υψηλότερες προτεραιότητες.

Λειτουργίες. Μια ουρά προτεραιότητας Q υποστηρίζει τα εξής:

- *Insert*(Q, x, k) – εισαγωγή του στοιχείου x με κλειδί k
- *ExtractMin*(Q) – επιστροφή και διαγραφή του στοιχείου με το μικρότερο κλειδί (μεγαλύτερη προτεραιότητα)
- *FindMin*(Q) – επιστροφή του στοιχείου με το μικρότερο κλειδί
- *DecreaseKey*(Q, x, k') – μειώνει το κλειδί του στοιχείου x σε k' (αύξηση προτεραιότητας)
- *IsEmpty*(Q) – έλεγχος κενής ουράς

Υλοποίηση Ουράς Προτεραιότητας με Σωρό

Επανάληψη

- *Insert*(Q, x, k) – εισαγωγή του στοιχείου x με κλειδί k. Εαν ο σωρός έχει n στοιχεία πέρνει χρόνο $\mathcal{O}(\log n)$.
- *ExtractMin*(Q) – επιστροφή και διαγραφή του στοιχείου με το μικρότερο κλειδί (μεγαλύτερη προτεραιότητα). Εαν ο σωρός έχει n στοιχεία πέρνει χρόνο $\mathcal{O}(\log n)$.
- *FindMin*(Q) – επιστροφή του στοιχείου με το μικρότερο κλειδί. Πέρνει χρόνο $\mathcal{O}(1)$.
- *DecreaseKey*(Q, x, k') – μειώνει το κλειδί του στοιχείου x σε k' (αύξηση προτεραιότητας). Εαν ο σωρός έχει n στοιχεία πέρνει χρόνο $\mathcal{O}(\log n)$.
- *IsEmpty*(Q) – έλεγχος κενής ουράς. Πέρνει χρόνο $\mathcal{O}(1)$.

Ο Αλγόριθμος του Dijkstra

Αλγόριθμος του Dijkstra

$S \leftarrow \emptyset$
θέσε $d[v] = \infty$ για κάθε $v \in V$
έστω Q η ουρά προτεραιότητας
Insert$(Q, s, 0)$
while *υπάρχουν κόμβοι στην* Q **do**
    $u \leftarrow$ *ExtractMin*(Q)
    $S \leftarrow S \cup \{u\}$
    for *κάθε κόμβο* v *όπου* $(u, v) \in E$ **do**
        if $d[v] = \infty$ **then**
            Insert$(Q, v, d[u] + l(u, v))$
            $d[v] \leftarrow d[u] + l(u, v)$
        else
            if $d[u] + l(u, v) < d[v]$ **then**
                DecreaseKey$(Q, v, d[u] + l(u, v))$
                $d[v] \leftarrow d[u] + l(u, v)$
            end
        end
    end
end

Ο αλγόριθμος χρησιμοποιεί μια ουρά προτεραιότητας και ένα πίνακα με τιμές για κάθε κόμβο.

Εκτελεί το πολύ

1. n φορές την λειτουργία *ExtractMin*() αφού κάθε φορά που την εκτελούμε έχουμε υπολογίσει την ελάχιστη απόσταση προς ακόμα ένα κόμβο
2. m φορές την λειτουργία *DecreaseKey*() αφού αυτή η λειτουργία συμβαίνει μια φορά για κάθε ακμή $e = (u, v)$ όταν ο κόμβος u προστίθεται στο σύνολο S

Dijkstra

Χρόνος Εκτέλεσης

λειτουργία PQ	Dijkstra	Binary Heap	Fibonacci Heap[1]
Insert	n	$\log n$	1
ExtractMin	n	$\log n$	$\log n$
DecreaseKey	m	$\log n$	1
IsEmpty	n	1	1
Σύνολο		$m \log n$	$m + n \log n$

[1] amortized bounds

Ελάχιστο Γεννητικό Δέντρο

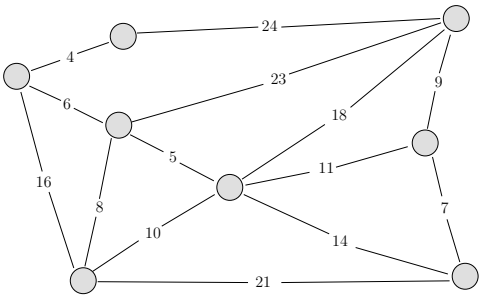
Minimum Spanning Tree

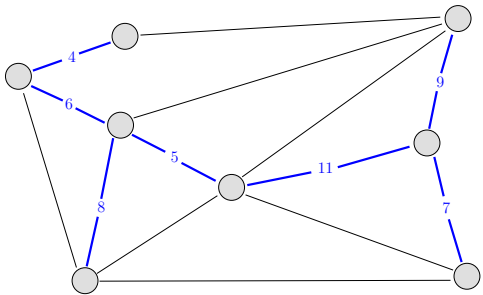
Έστω ένα συνεκτικό γράφημα $G = (V, E)$ όπου κάθε ακμή $e \in E$ έχει ένα πραγματικό βάρος $c_e \geq 0$. Ένα ελάχιστο γεννητικό δέντρο (MST) είναι ένα υποσύνολο ακμών $T \subseteq E$ όπου T είναι ένα συνεκτικό δέντρο και το άθροισμα των βαρών των ακμών του δέντρου είναι ελάχιστο.

Ελάχιστο Γεννητικό Δέντρο

Minimum Spanning Tree

Έστω ένα συνεκτικό γράφημα $G = (V, E)$ όπου κάθε ακμή $e \in E$ έχει ένα πραγματικό βάρος $c_e \geq 0$. Ένα ελάχιστο γεννητικό δέντρο (MST) είναι ένα υποσύνολο ακμών $T \subseteq E$ όπου T είναι ένα συνεκτικό δέντρο και το άθροισμα των βαρών των ακμών του δέντρου είναι ελάχιστο.

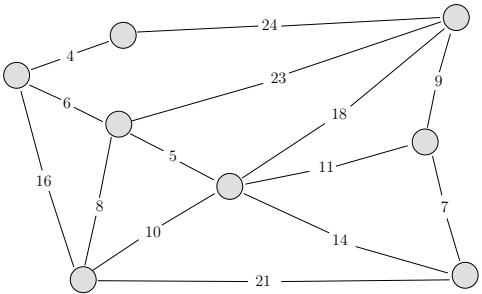


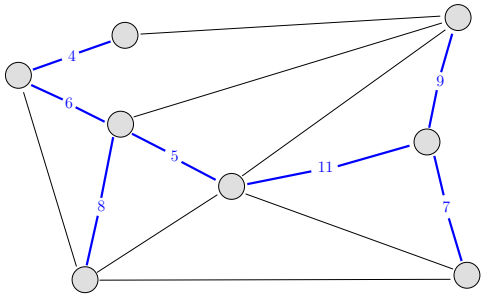


Ελάχιστο Γεννητικό Δέντρο

Minimum Spanning Tree

Έστω ένα συνεκτικό γράφημα $G = (V, E)$ όπου κάθε ακμή $e \in E$ έχει ένα πραγματικό βάρος $c_e \geq 0$. Ένα ελάχιστο γεννητικό δέντρο (MST) είναι ένα υποσύνολο ακμών $T \subseteq E$ όπου T είναι ένα συνεκτικό δέντρο και το άθροισμα των βαρών των ακμών του δέντρου είναι ελάχιστο.





Το γράφημα με n κόμβους και όλες τις $\binom{n}{2}$ ακμές έχει n^{n-2} συνεκτικά δέντρα (Cayley's Theorem, 1889). Άρα η λύση ωμής βίας αποκλείεται.

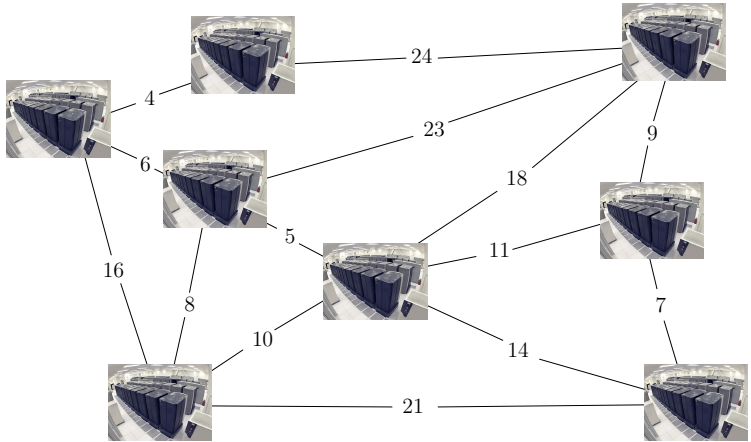
Εφαρμογές MST

Το MST πρόβλημα είναι πάρα πολύ σημαντικό με εφαρμογές σε πολλά πεδία.

1. σχεδίαση δικτύων
   - τηλεφωνικό, ηλεκτρικό, υδραυλικό, υπολογιστών, δρόμων
2. προσεγγιστικούς αλγορίθμους για ”δύσκολα” προβλήματα
   - traveling salesman problem
   - steiner tree problem
3. εμφανίζεται ως υποπρόβλημα στην λύση διαφόρων άλλων προβλημάτων

Παράδειγμα Εφαρμογής MST

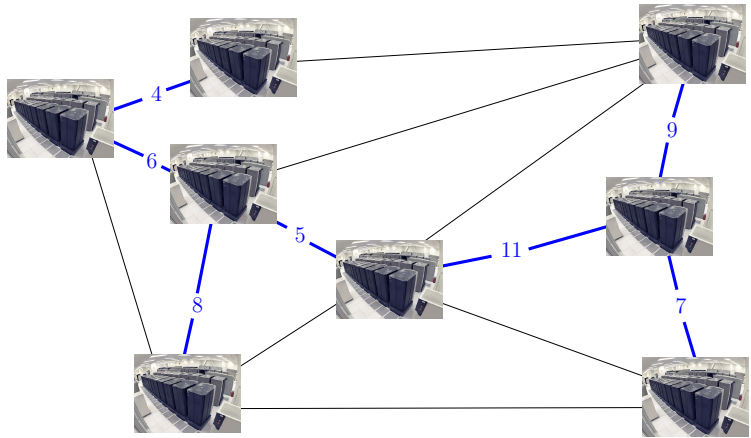
Ένας πάροχος έχει κάποια data centers και θέλει να τα συνδέσει μεταξύ τους με οπτικές ίνες, έτσι ώστε όλα τα data centers να μπορούν να επικοινωνούν μεταξύ τους.



Οι ακμές είναι οι δυνατές συνδέσεις που μπορούμε να κάνουμε. Το βάρος κάθε ακμής είναι το κόστος της ακμής (είτε κόστος κατασκευής, είτε κόστος επικοινωνίας, κ.τ.λ.).

Παράδειγμα Εφαρμογής MST

Ένας πάροχος έχει κάποια data centers και θέλει να τα συνδέσει μεταξύ τους με οπτικές ίνες, έτσι ώστε όλα τα data centers να μπορούν να επικοινωνούν μεταξύ τους.



Οι ακμές είναι οι δυνατές συνδέσεις που μπορούμε να κάνουμε. Το βάρος κάθε ακμής είναι το κόστος της ακμής (είτε κόστος κατασκευής, είτε κόστος επικοινωνίας, κ.τ.λ.).

Άπληστοι Αλγόριθμοι για MST

Μπορούμε εύκολα να σκεφτούμε διάφορους άπληστους αλγορίθμους για το πρόβλημα.

1. Ένας απλός αλγόριθμος ξεκινά χωρίς καθόλου ακμές και δομεί ένα γεννητικό δέντρο με διαδοχική εισαγωγή ακμών από το E κατά αύξουσα σειρά κόστους. Εαν μια ακμή δημιουργεί κύκλο απλά απορρίπτεται. Αυτός είναι ο αλγόριθμος του Kruskal.
2. Ένας άλλος απλός αλγόριθμος είναι παρόμοιος με τον αλγόριθμο του Dijkstra. Διατηρούμε ένα σύνολο $S \subseteq V$ για το οποίο έχουμε ήδη κατασκευάσει ένα γεννητικό δέντρο και προσθέτουμε την πιο φτηνή ακμή μεταξύ των S και $V \setminus S$. Ο αλγόριθμος ξεκινά με $S = \{s\}$ όπου $s \in V$ είναι οποιοσδήποτε κόμβος. Αυτός είναι ο αλγόριθμος του Prim.

Θα δούμε πως και οι δύο αυτοί αλγόριθμοι παράγουν βέλτιστες λύσεις.

Ιδιότητα Τομής

Cut Property

Λήμμα

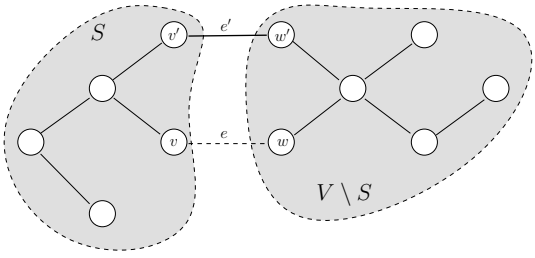
Υποθέστε πως όλα τα βάρη των ακμών είναι διαφορετικά. Έστω $S \subset V$ ένα υποσύνολο κόμβων διάφορο του κενού και έστω $e = (v, w)$ η ακμή ελάχιστου κόστους με $v \in S$ και $w \in V \setminus S$. Κάθε ελάχιστο γεννητικό δέντρο (MST) θα περιέχει την ακμή e.

Ιδιότητα Τομής

Cut Property

Λήμμα

Υποθέστε πως όλα τα βάρη των ακμών είναι διαφορετικά. Έστω $S \subset V$ ένα υποσύνολο κόμβων διάφορο του κενού και έστω $e = (v, w)$ η ακμή ελάχιστου κόστους με $v \in S$ και $w \in V \setminus S$. Κάθε ελάχιστο γεννητικό δέντρο (MST) θα περιέχει την ακμή e.

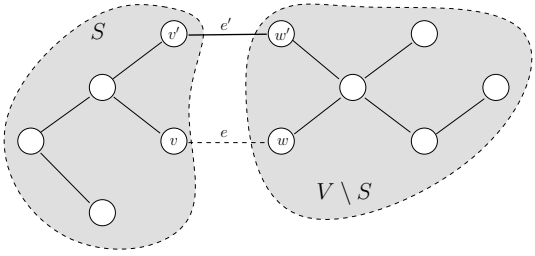


Ιδιότητα Τομής

Cut Property

Λήμμα

Υποθέστε πως όλα τα βάρη των ακμών είναι διαφορετικά. Έστω $S \subset V$ ένα υποσύνολο κόμβων διάφορο του κενού και έστω $e = (v, w)$ η ακμή ελάχιστου κόστους με $v \in S$ και $w \in V \setminus S$. Κάθε ελάχιστο γεννητικό δέντρο (MST) θα περιέχει την ακμή e.



Απόδειξη

Έστω T ένα MST που δεν περιέχει την e. Υπάρχει διαδρομή μεταξύ v και w, η οποία έστω πως περνάει την τομή με την ακμή $e' = (v', w')$.

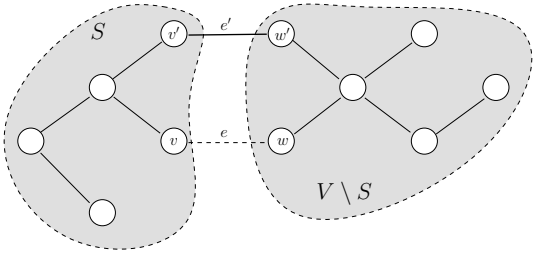
□

Ιδιότητα Τομής

Cut Property

Λήμμα

Υποθέστε πως όλα τα βάρη των ακμών είναι διαφορετικά. Έστω $S \subset V$ ένα υποσύνολο κόμβων διάφορο του κενού και έστω $e = (v, w)$ η ακμή ελάχιστου κόστους με $v \in S$ και $w \in V \setminus S$. Κάθε ελάχιστο γεννητικό δέντρο (MST) θα περιέχει την ακμή e.



Απόδειξη

Έστω T ένα MST που δεν περιέχει την e. Υπάρχει διαδρομή μεταξύ v και w, η οποία έστω πως περνάει την τομή με την ακμή $e' = (v', w')$.

Το σύνολο ακμών $T' = T \setminus \{e'\} \cup \{e\}$ έχει μικρότερο κόστος. Επίσης είναι συνεκτικό και δεν περιέχει κύκλους.

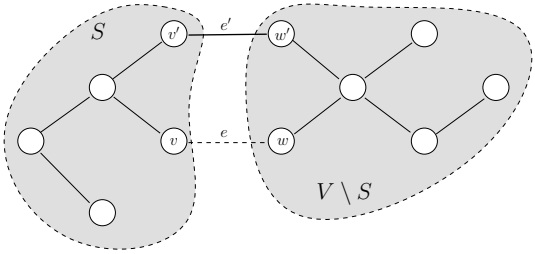
□

Ιδιότητα Τομής

Cut Property

Λήμμα

Υποθέστε πως όλα τα βάρη των ακμών είναι διαφορετικά. Έστω $S \subset V$ ένα υποσύνολο κόμβων διάφορο του κενού και έστω $e = (v, w)$ η ακμή ελάχιστου κόστους με $v \in S$ και $w \in V \setminus S$. Κάθε ελάχιστο γεννητικό δέντρο (MST) θα περιέχει την ακμή e.



Απόδειξη

Έστω T ένα MST που δεν περιέχει την e. Υπάρχει διαδρομή μεταξύ v και w, η οποία έστω πως περνάει την τομή με την ακμή $e' = (v', w')$.

Το σύνολο ακμών $T' = T \setminus \{e'\} \cup \{e\}$ έχει μικρότερο κόστος. Επίσης είναι συνεκτικό και δεν περιέχει κύκλους.

Άτοπο. □

Ιδιότητα Κύκλου

Cycle Property

Λήμμα

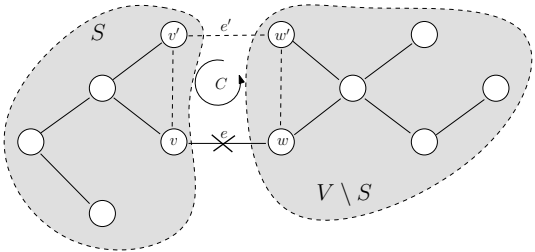
Υποθέστε πως όλα τα βάρη των ακμών είναι διαφορετικά. Έστω C ένας κύκλος του γραφήματος $G(V, E)$, και έστω ότι η ακμή $e = (v, w) \in E$ είναι η πιο ακριβή ακμή που ανήκει στον C. Τότε η ακμή e δεν ανήκει σε κανένα ελάχιστο γεννητικό δέντρο (MST) του G.

Ιδιότητα Κύκλου

Cycle Property

Λήμμα

Υποθέστε πως όλα τα βάρη των ακμών είναι διαφορετικά. Έστω C ένας κύκλος του γραφήματος $G(V, E)$, και έστω ότι η ακμή $e = (v, w) \in E$ είναι η πιο ακριβή ακμή που ανήκει στον C. Τότε η ακμή e δεν ανήκει σε κανένα ελάχιστο γεννητικό δέντρο (MST) του G.

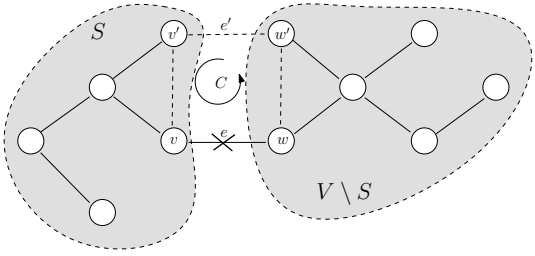


Ιδιότητα Κύκλου

Cycle Property

Λήμμα

Υποθέστε πως όλα τα βάρη των ακμών είναι διαφορετικά. Έστω C ένας κύκλος του γραφήματος $G(V, E)$, και έστω ότι η ακμή $e = (v, w) \in E$ είναι η πιο ακριβή ακμή που ανήκει στον C. Τότε η ακμή e δεν ανήκει σε κανένα ελάχιστο γεννητικό δέντρο (MST) του G.



Απόδειξη

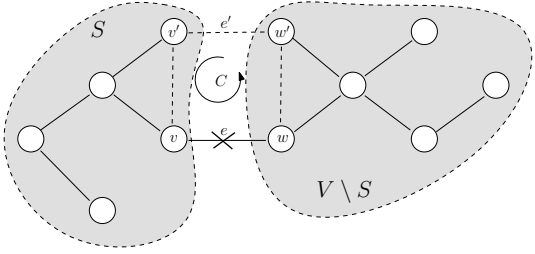
Έστω T ένα MST που περιέχει την e. Διαγράφοντας την e από το T, χωρίζουμε τους κόμβους σε δύο υποσύνολα S και $V \setminus S$.

Ιδιότητα Κύκλου

Cycle Property

Λήμμα

Υποθέστε πως όλα τα βάρη των ακμών είναι διαφορετικά. Έστω C ένας κύκλος του γραφήματος $G(V, E)$, και έστω ότι η ακμή $e = (v, w) \in E$ είναι η πιο ακριβή ακμή που ανήκει στον C. Τότε η ακμή e δεν ανήκει σε κανένα ελάχιστο γεννητικό δέντρο (MST) του G.



Απόδειξη

Έστω T ένα MST που περιέχει την e. Διαγράφοντας την e από το T, χωρίζουμε τους κόμβους σε δύο υποσύνολα S και $V \setminus S$.

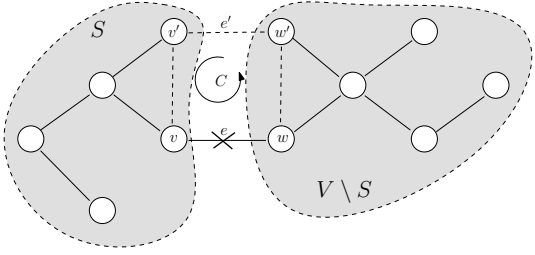
Ακολουθώντας τον κύκλο C από v προς w μπορούμε να βρούμε μια ακμή e' που περνάει την τομή $(S, V \setminus S)$. Αυτή η ακμή έχει βάρος μικρότερο της e από την υπόθεση.

Ιδιότητα Κύκλου

Cycle Property

Λήμμα

Υποθέστε πως όλα τα βάρη των ακμών είναι διαφορετικά. Έστω C ένας κύκλος του γραφήματος $G(V, E)$, και έστω ότι η ακμή $e = (v, w) \in E$ είναι η πιο ακριβή ακμή που ανήκει στον C. Τότε η ακμή e δεν ανήκει σε κανένα ελάχιστο γεννητικό δέντρο (MST) του G.



Απόδειξη

Έστω T ένα MST που περιέχει την e. Διαγράφοντας την e από το T, χωρίζουμε τους κόμβους σε δύο υποσύνολα S και $V \setminus S$.

Ακολουθώντας τον κύκλο C από v προς w μπορούμε να βρούμε μια ακμή e' που περνάει την τομή $(S, V \setminus S)$. Αυτή η ακμή έχει βάρος μικρότερο της e από την υπόθεση.

Το σύνολο ακμών $T' = T \setminus \{e\} \cup \{e'\}$ έχει μικρότερο κόστος. Επίσης είναι συνεκτικό και δεν περιέχει κύκλους.

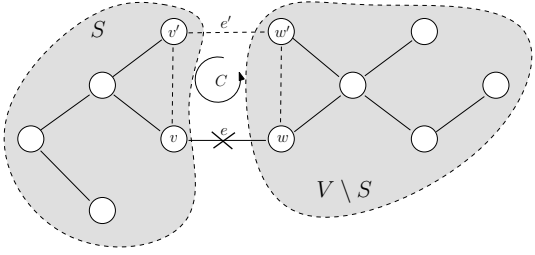
□

Ιδιότητα Κύκλου

Cycle Property

Λήμμα

Υποθέστε πως όλα τα βάρη των ακμών είναι διαφορετικά. Έστω C ένας κύκλος του γραφήματος $G(V, E)$, και έστω ότι η ακμή $e = (v, w) \in E$ είναι η πιο ακριβή ακμή που ανήκει στον C. Τότε η ακμή e δεν ανήκει σε κανένα ελάχιστο γεννητικό δέντρο (MST) του G.



Απόδειξη

Έστω T ένα MST που περιέχει την e. Διαγράφοντας την e από το T, χωρίζουμε τους κόμβους σε δύο υποσύνολα S και $V \setminus S$.

Ακολουθώντας τον κύκλο C από v προς w μπορούμε να βρούμε μια ακμή e' που περνάει την τομή $(S, V \setminus S)$. Αυτή η ακμή έχει βάρος μικρότερο της e από την υπόθεση.

Το σύνολο ακμών $T' = T \setminus \{e\} \cup \{e'\}$ έχει μικρότερο κόστος. Επίσης είναι συνεκτικό και δεν περιέχει κύκλους.

Άτοπο. □

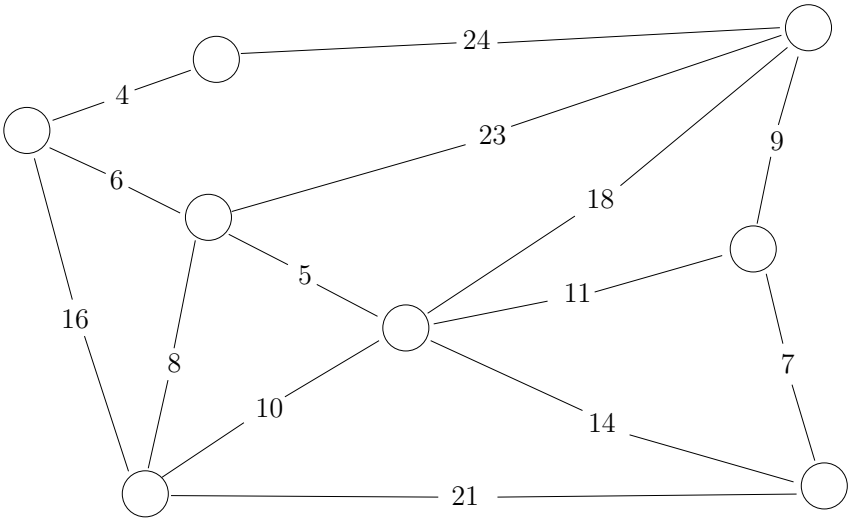
Αλγόριθμος του Prim

Αλγόριθμος του Prim για MST

Διάλεξε έναν οποιοδήποτε κόμβο $s \in V$
$S \leftarrow \{s\}$.
$T \leftarrow \emptyset$
while $S \neq V$ **do**
    διάλεξε την ακμή $e = (u, v)$ με ελάχιστο κόστος όπου $u \in S$ και $v \in V \setminus S$
    $T \leftarrow T \cup \{e\}$
end
return *το σύνολο* T

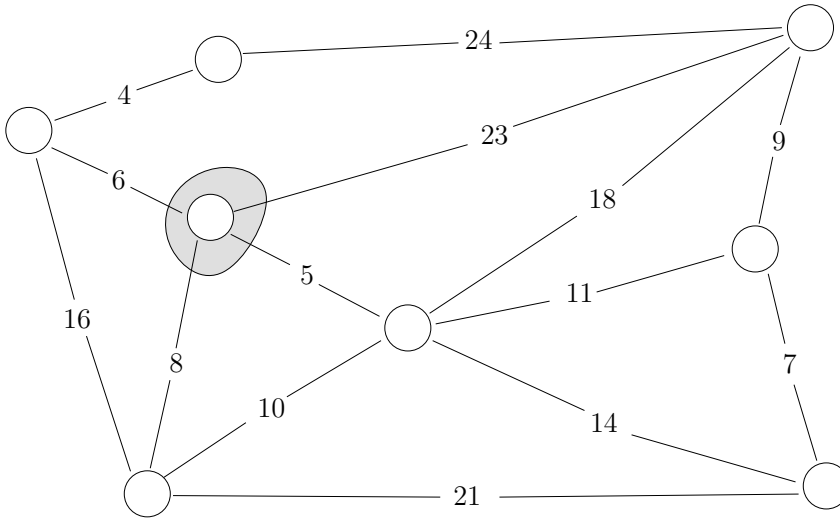
Αλγόριθμος του Prim

Παράδειγμα Εκτέλεσης



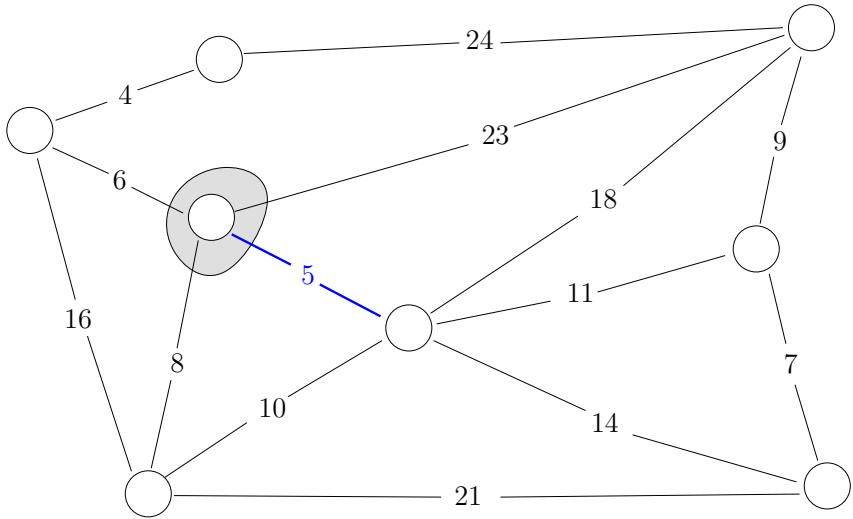
Αλγόριθμος του Prim

Παράδειγμα Εκτέλεσης



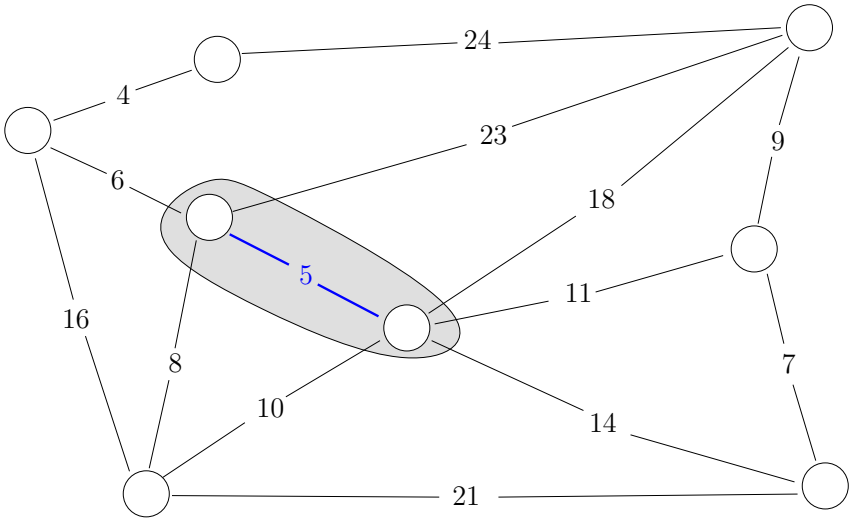
Αλγόριθμος του Prim

Παράδειγμα Εκτέλεσης



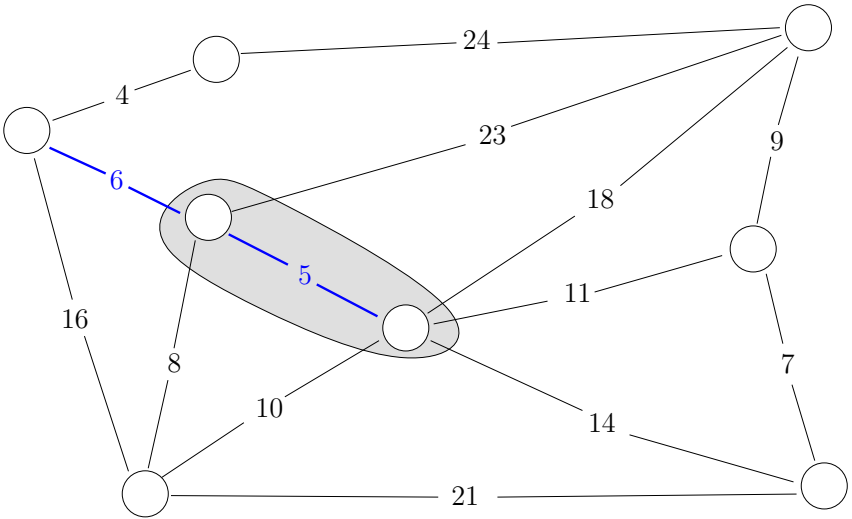
Αλγόριθμος του Prim

Παράδειγμα Εκτέλεσης



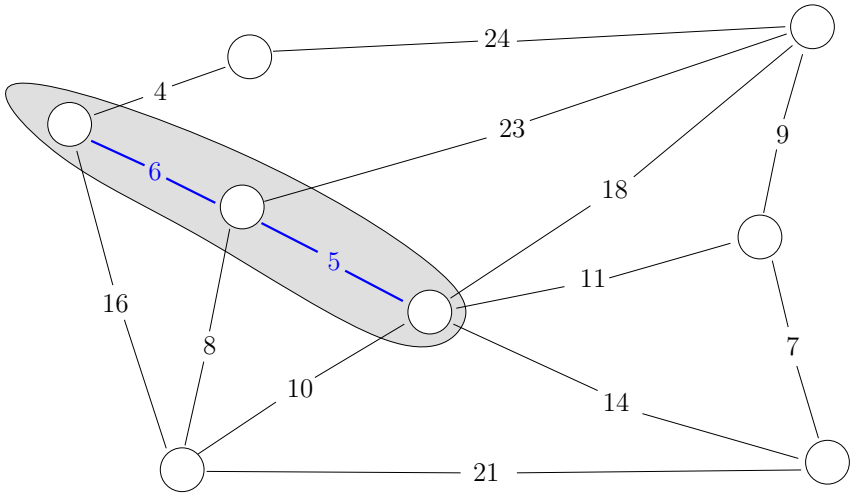
Αλγόριθμος του Prim

Παράδειγμα Εκτέλεσης



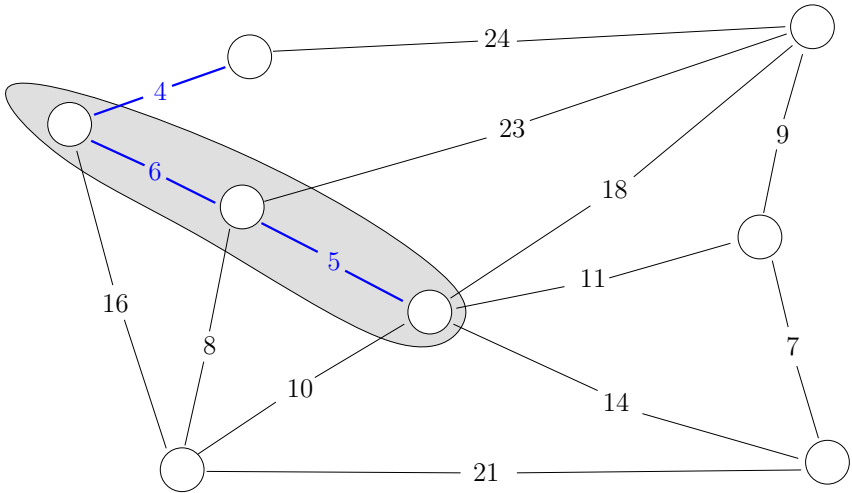
Αλγόριθμος του Prim

Παράδειγμα Εκτέλεσης



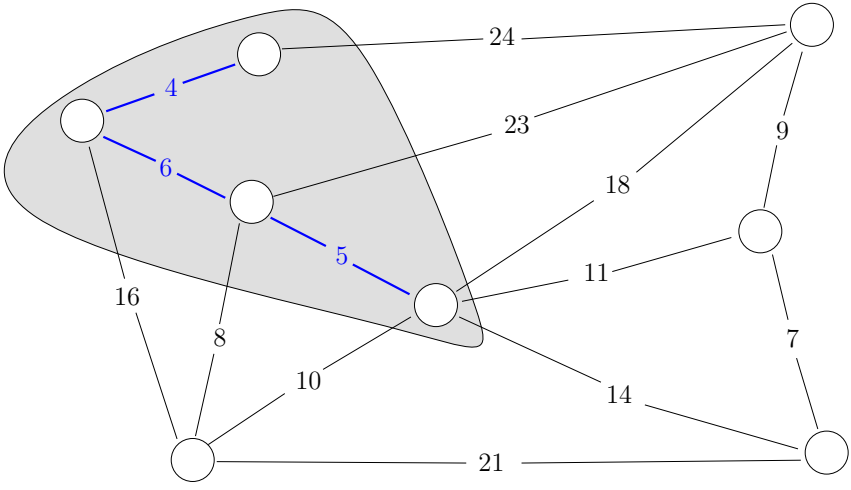
Αλγόριθμος του Prim

Παράδειγμα Εκτέλεσης



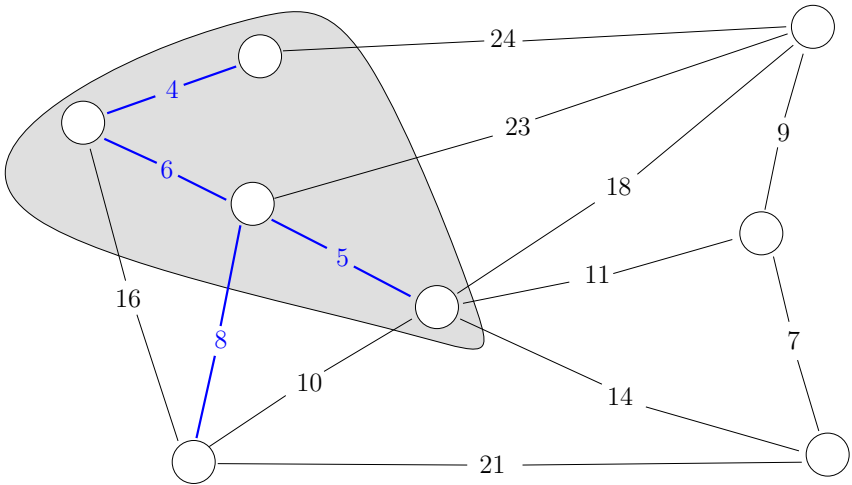
Αλγόριθμος του Prim

Παράδειγμα Εκτέλεσης



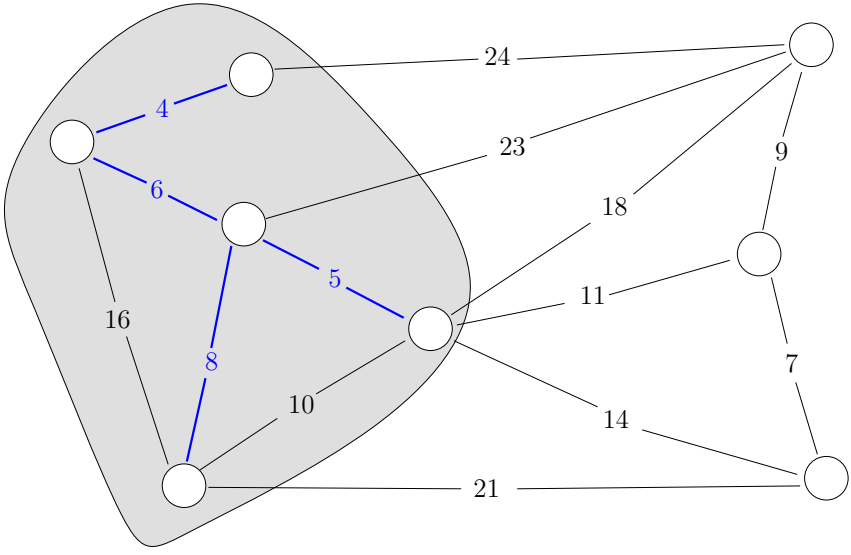
Αλγόριθμος του Prim

Παράδειγμα Εκτέλεσης



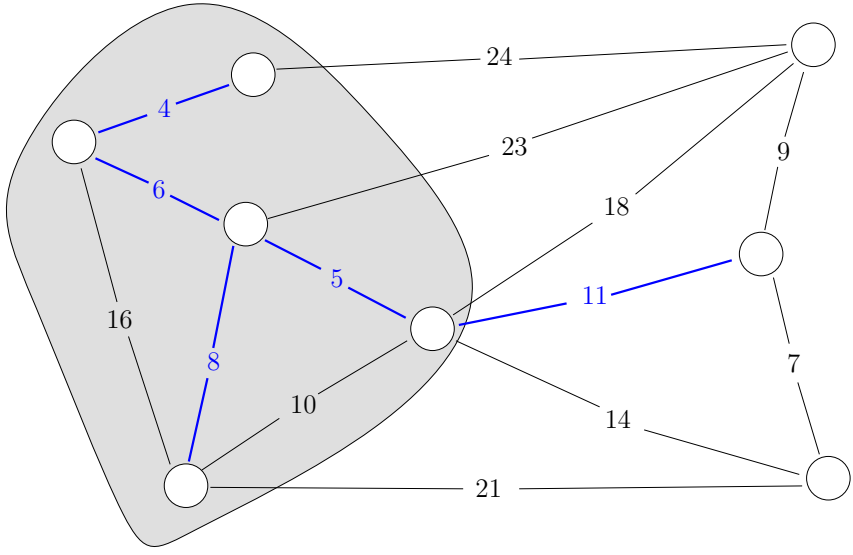
Αλγόριθμος του Prim

Παράδειγμα Εκτέλεσης



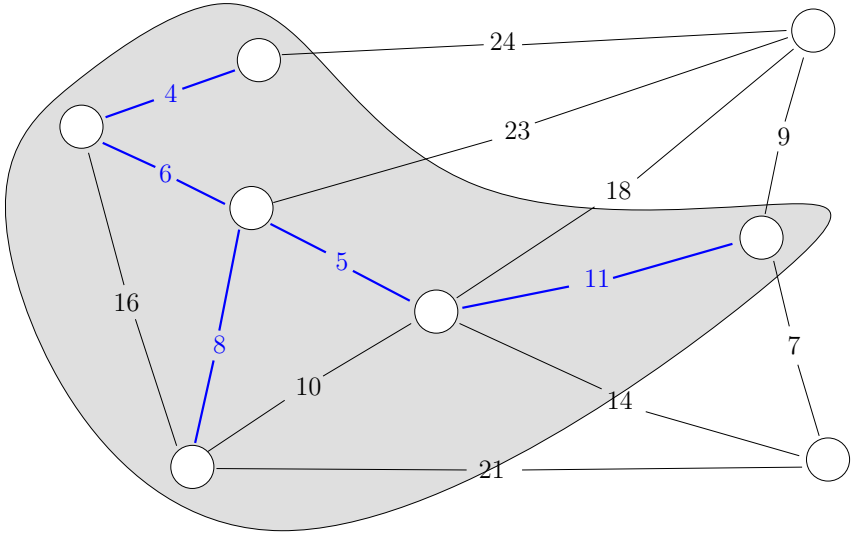
Αλγόριθμος του Prim

Παράδειγμα Εκτέλεσης



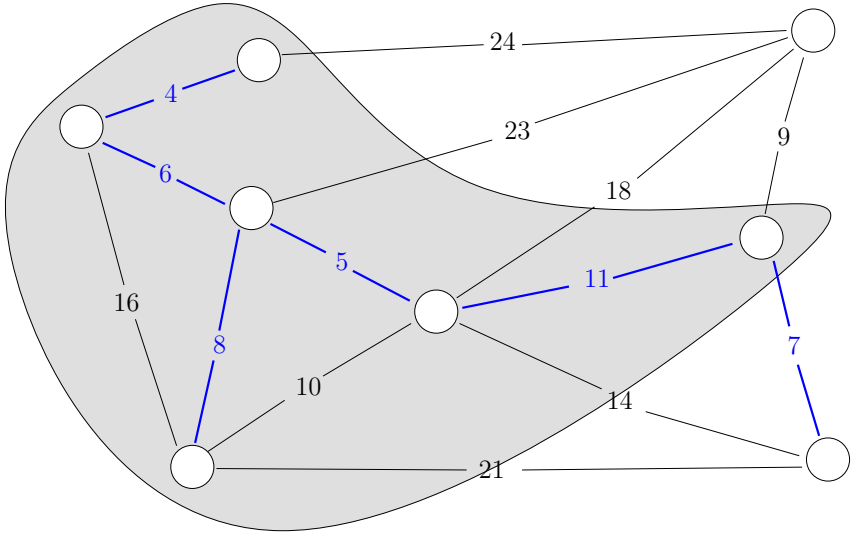
Αλγόριθμος του Prim

Παράδειγμα Εκτέλεσης



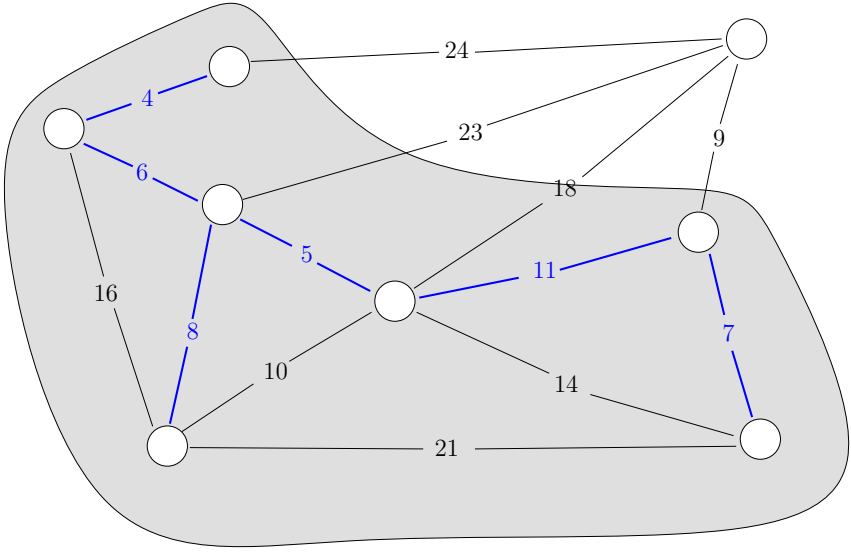
Αλγόριθμος του Prim

Παράδειγμα Εκτέλεσης



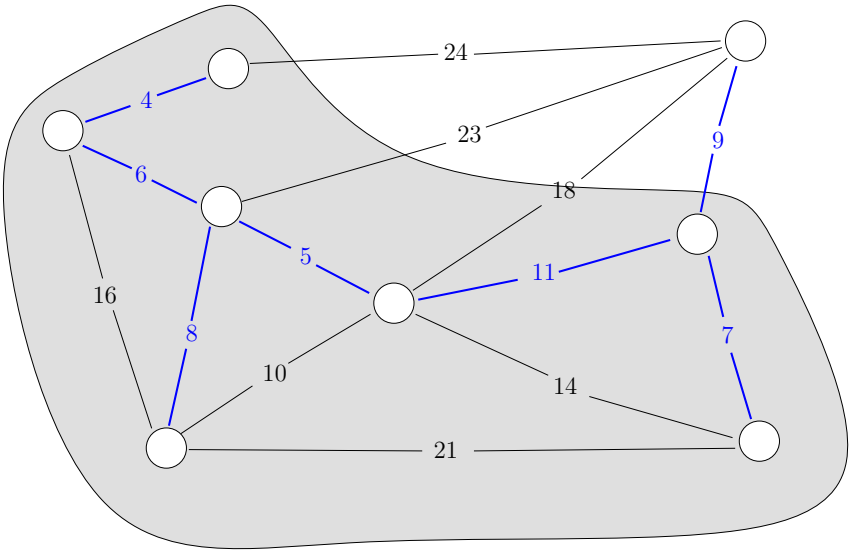
Αλγόριθμος του Prim

Παράδειγμα Εκτέλεσης



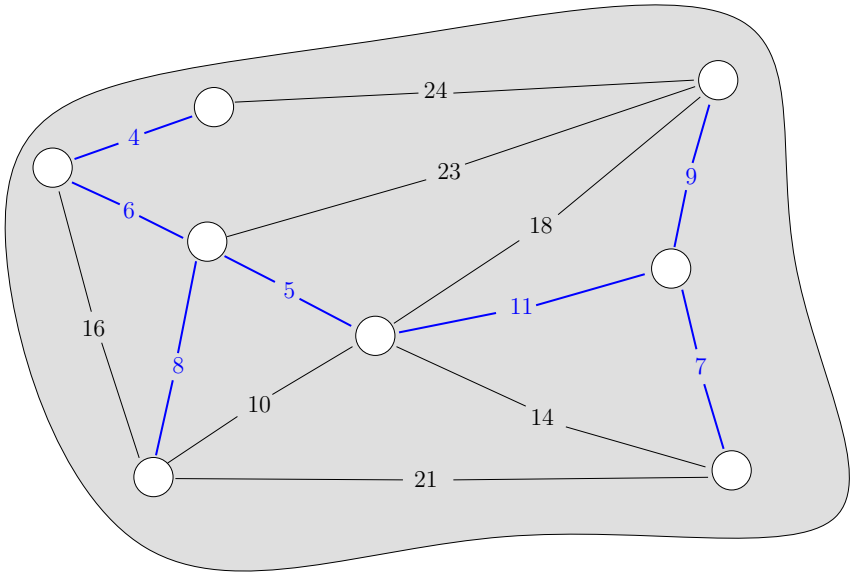
Αλγόριθμος του Prim

Παράδειγμα Εκτέλεσης



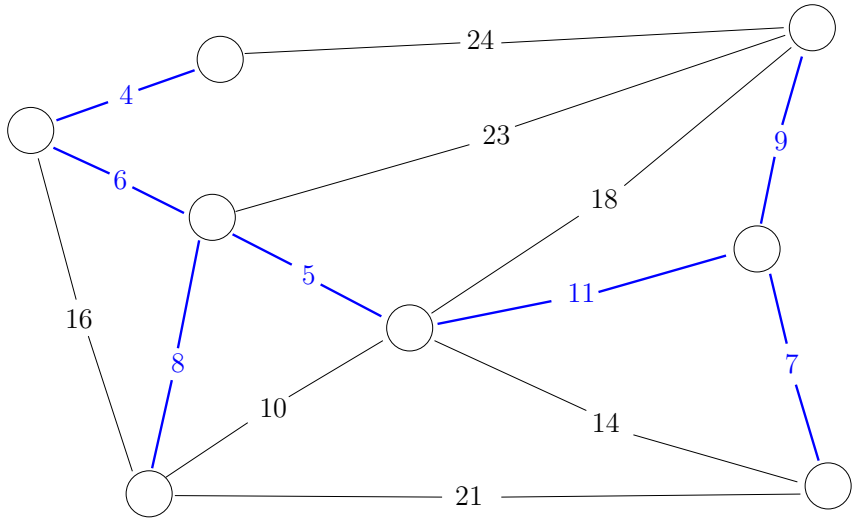
Αλγόριθμος του Prim

Παράδειγμα Εκτέλεσης



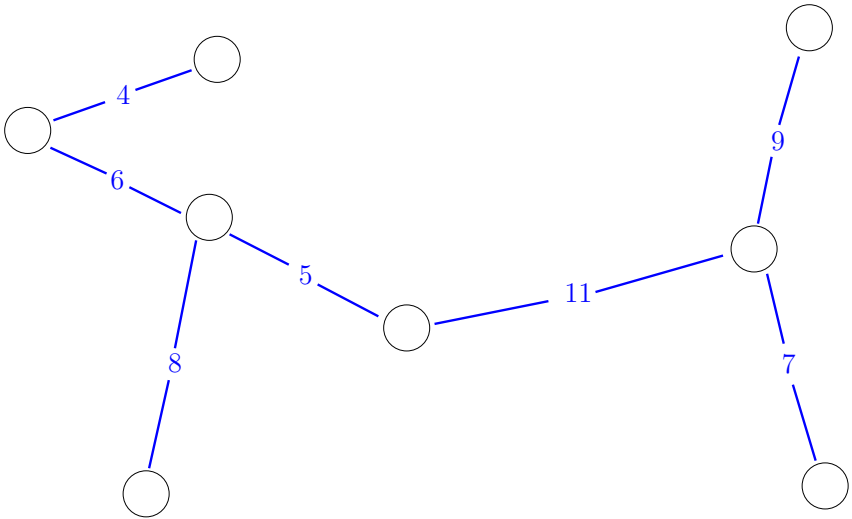
Αλγόριθμος του Prim

Παράδειγμα Εκτέλεσης



Αλγόριθμος του Prim

Παράδειγμα Εκτέλεσης



Θεώρημα

Ο αλγόριθμος του Prim παράγει ένα ελάχιστο γεννητικό δέντρο του G.

Απόδειξη

Προκύπτει απευθείας από την ιδιότητα τομής (cut property) αφού ο αλγόριθμος προσθέτει πάντα την ακμή με ελάχιστο κόστος που περνάει μια τομή $(S, V \setminus S)$. □

Η υλοποίηση του αλγορίθμου του Prim είναι παρόμοια με την υλοποίηση του αλγορίθμου του Dijkstra.

- δεδομένου του συνόλου S κάθε κόμβος έχει ένα κόστος ”προσάρτησης”
- το κόστος προσάρτησης του κόμβου $v \in V \setminus S$ είναι

$$\pi(v) = \min_{e=(u,v):u \in S} c_e$$

- πρέπει να μπορούμε να διαλέξουμε τον κόμβο $v \in V \setminus S$ με το ελάχιστο κόστος προσάρτησης
- επίσης πρέπει να μπορούμε να ανανεώσουμε τα κόστη προσάρτησης κάθε φορά που αυξάνουμε το μέγεθος του S κατά 1.

Θα χρησιμοποιήσουμε πάλι ουρά προτεραιότητας και προτεραιότητα το κόστος προσάρτησης.

Υλοποίηση του αλγορίθμου του Prim

Αλγόριθμος του Prim

```
for κάθε κόμβο v ∈ V do
    d[v] = ∞
end
έστω Q η ουρά προτεραιότητας
διάλεξε οποιονδήποτε κόμβο s ∈ V
for κάθε ακμή e = (s, u) ∈ E do
    Insert(Q, u, c_e)
    d[u] ← c_e
end
S = {s}
while η ουρά Q δεν είναι άδεια do
    u ← ExtractMin(Q)
    S ← S ∪ {u}
    for κάθε κόμβο v όπου e = (u, v) ∈ E do
        if d[v] = ∞ then
            Insert(Q, v, c_e)
            d[v] ← c_e
        else
            if c_e < d[v] then
                DecreaseKey(Q, v, c_e)
                d[v] ← c_e
            end
        end
    end
```

Ο αλγόριθμος εκτελεί

- το πολύ n λειτουργίες *Insert* στην ουρά,
- το πολύ n λειτουργίες *ExtractMin* και
- το πολύ m λειτουργίες *DecreaseKey*.

Με την χρήστη ως δυαδικού σωρού για ουρά προτεραιότητας ο αλγόριθμος τρέχει σε χρόνο $\mathcal{O}(m \log n)$.

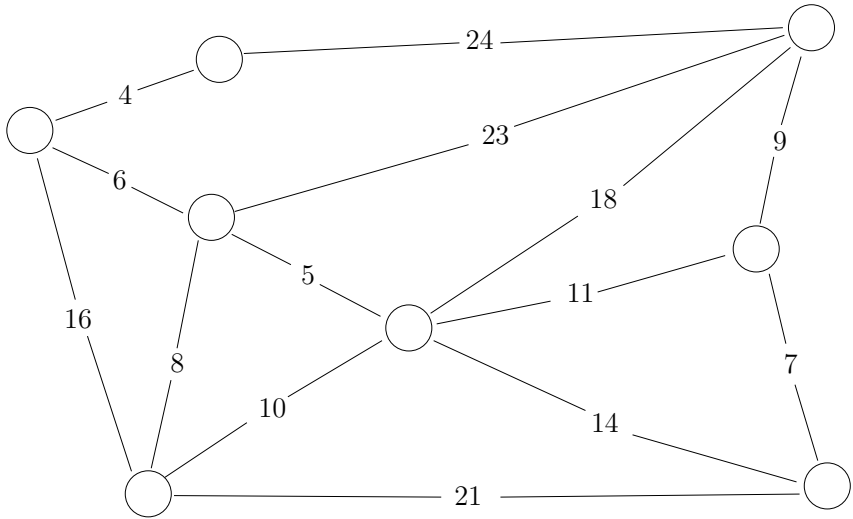
Αλγόριθμος του Kruskal

Αλγόριθμος του Kruskal για MST

Ταξινόμησε τις ακμές σε αύξουσα σειρά με βάση τα βάρη c_e
$T \leftarrow \emptyset$
while *δεν έχουμε δει όλες τις ακμές* **do**
    έστω e η επόμενη ακμή στην ταξινομημένη σειρά
    if $T \cup \{e\}$ *δεν περιέχει κύκλο* **then**
        $T \leftarrow T \cup \{e\}$
    end
end
return *το σύνολο* T

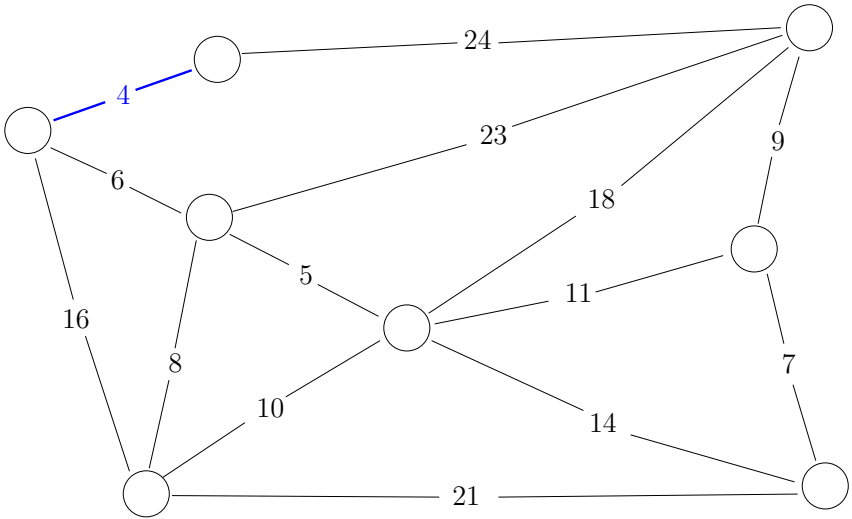
Αλγόριθμος του Kruskal

Παράδειγμα Εκτέλεσης



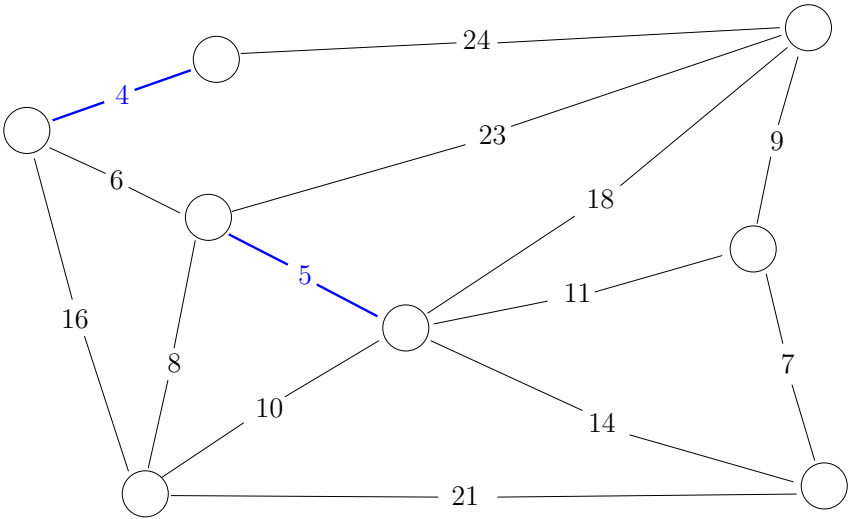
Αλγόριθμος του Kruskal

Παράδειγμα Εκτέλεσης



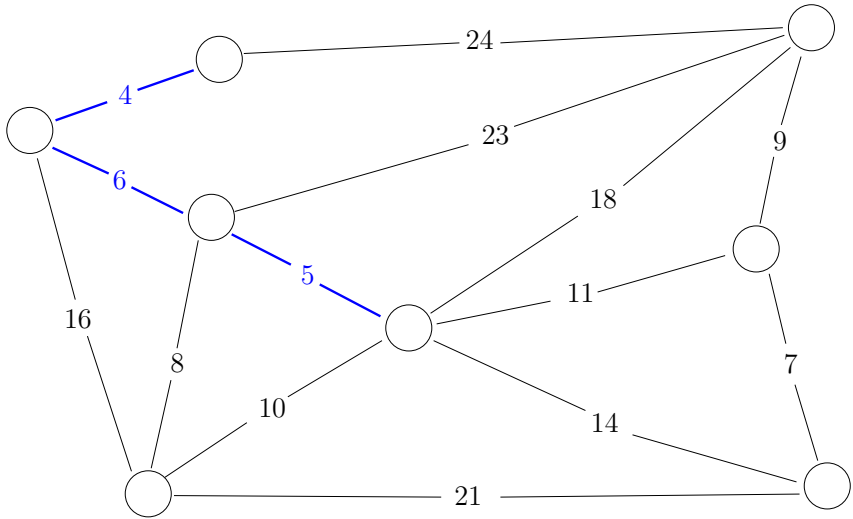
Αλγόριθμος του Kruskal

Παράδειγμα Εκτέλεσης



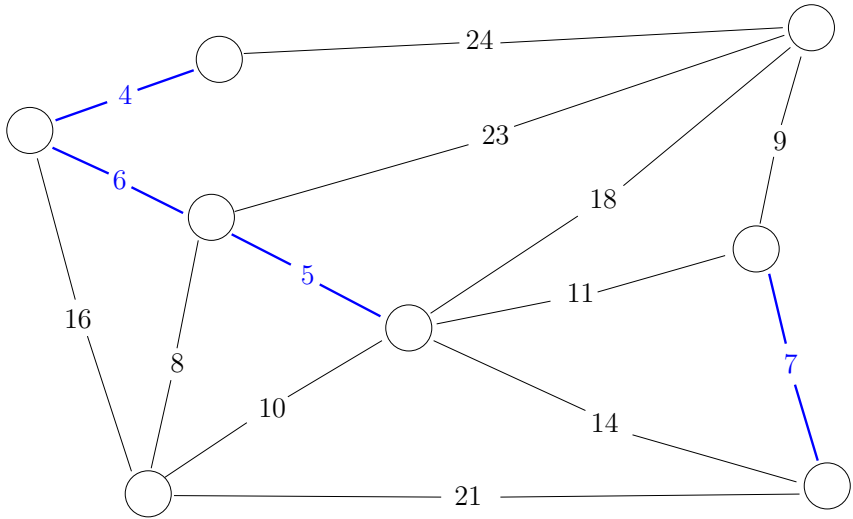
Αλγόριθμος του Kruskal

Παράδειγμα Εκτέλεσης



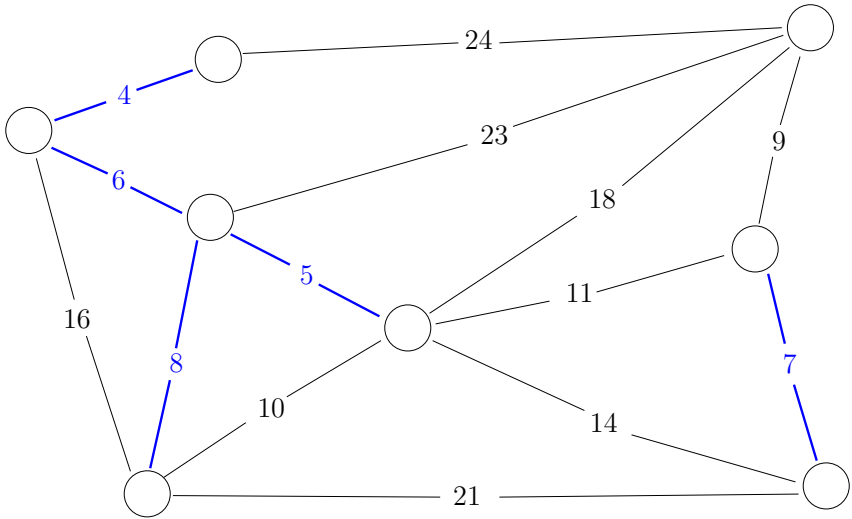
Αλγόριθμος του Kruskal

Παράδειγμα Εκτέλεσης



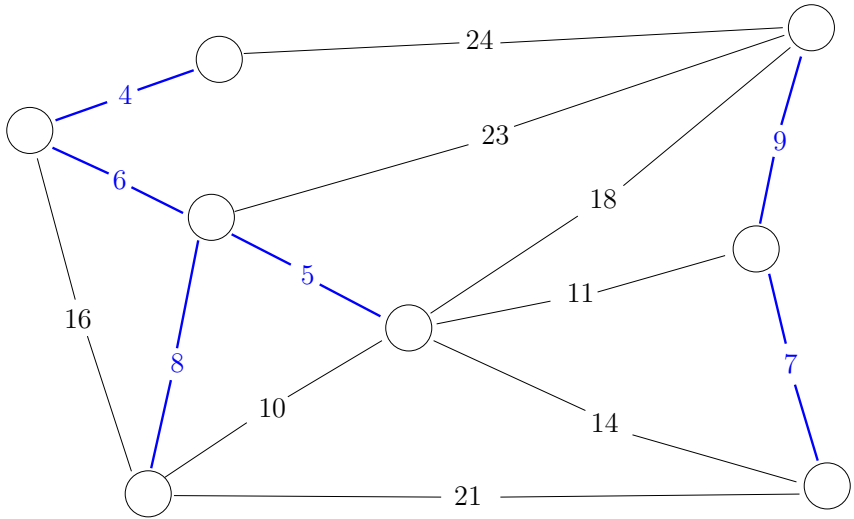
Αλγόριθμος του Kruskal

Παράδειγμα Εκτέλεσης



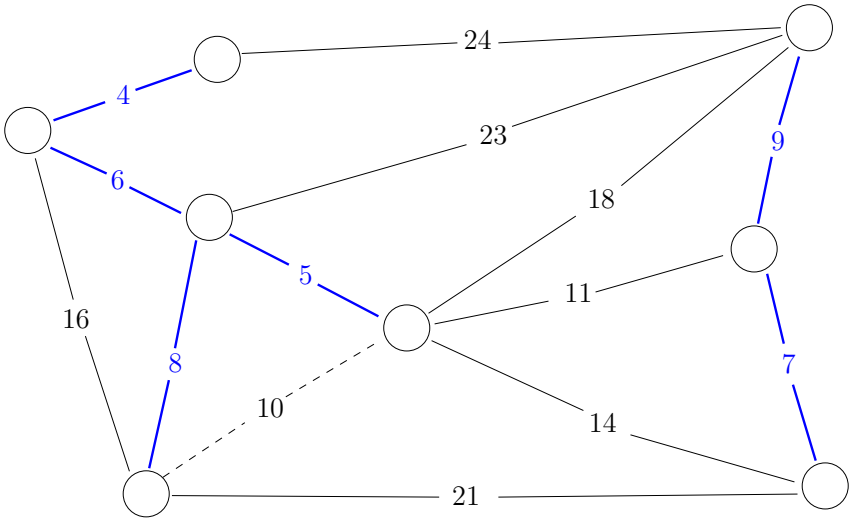
Αλγόριθμος του Kruskal

Παράδειγμα Εκτέλεσης



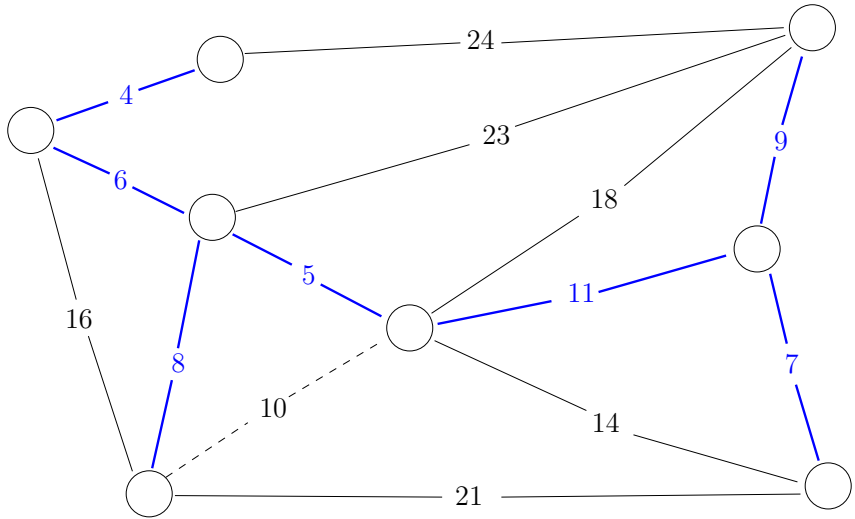
Αλγόριθμος του Kruskal

Παράδειγμα Εκτέλεσης



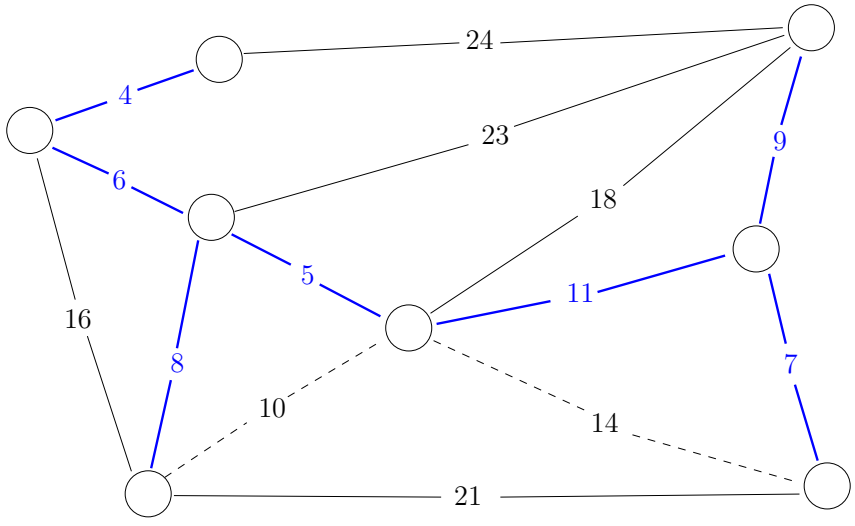
Αλγόριθμος του Kruskal

Παράδειγμα Εκτέλεσης



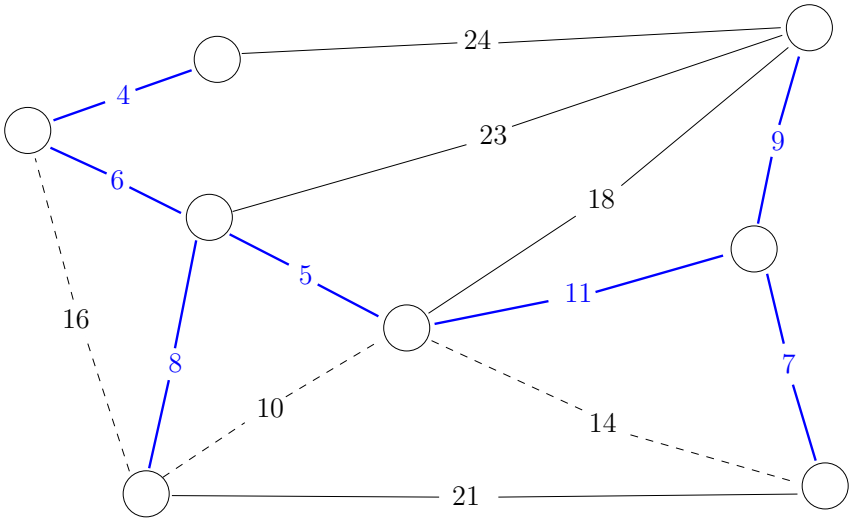
Αλγόριθμος του Kruskal

Παράδειγμα Εκτέλεσης



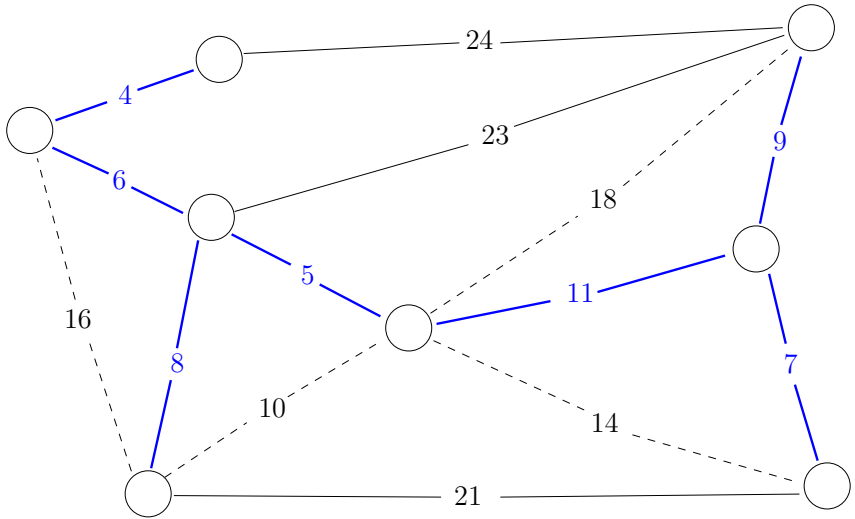
Αλγόριθμος του Kruskal

Παράδειγμα Εκτέλεσης



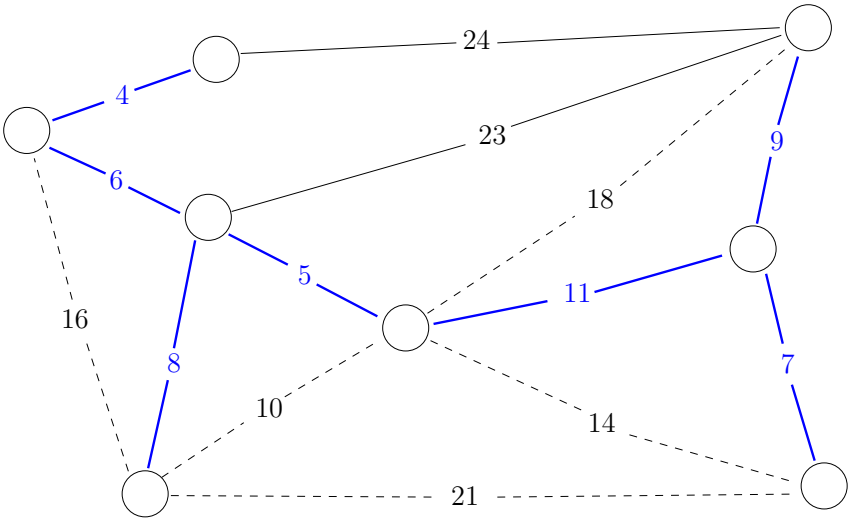
Αλγόριθμος του Kruskal

Παράδειγμα Εκτέλεσης



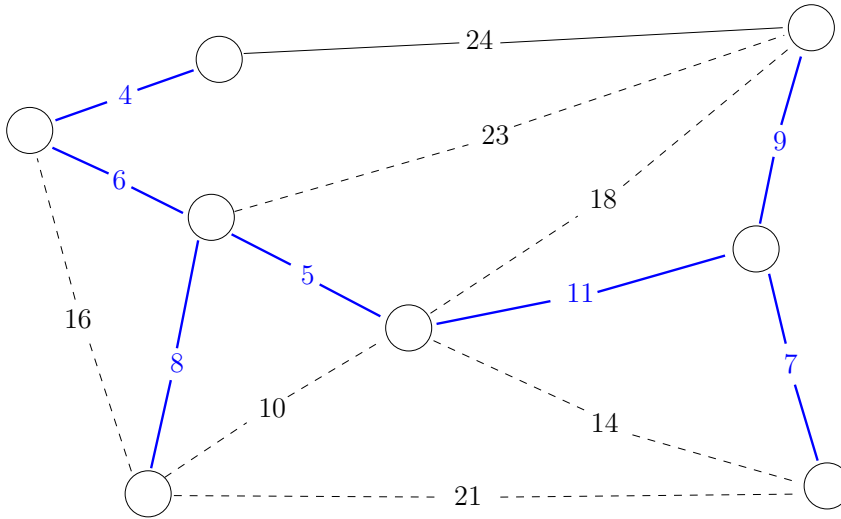
Αλγόριθμος του Kruskal

Παράδειγμα Εκτέλεσης



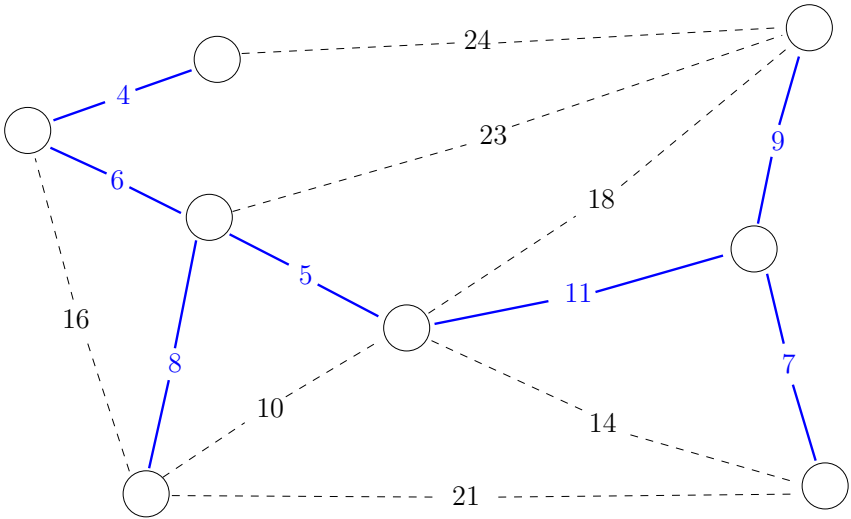
Αλγόριθμος του Kruskal

Παράδειγμα Εκτέλεσης



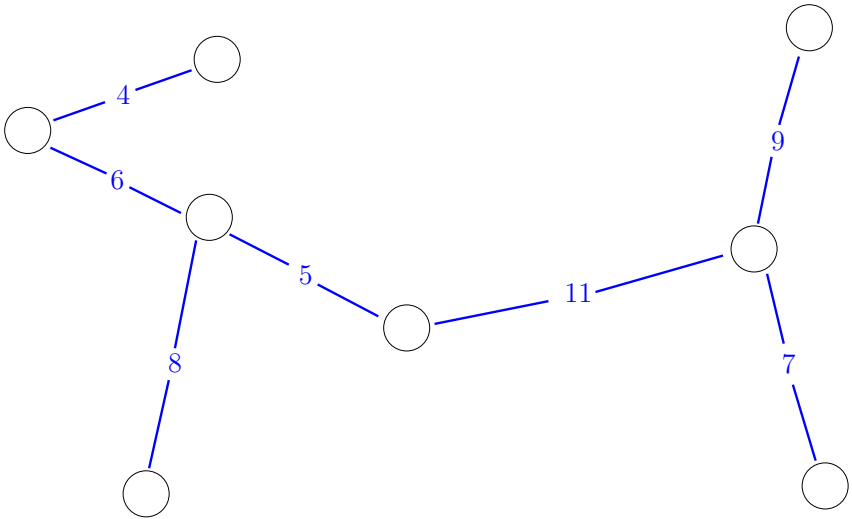
Αλγόριθμος του Kruskal

Παράδειγμα Εκτέλεσης



Αλγόριθμος του Kruskal

Παράδειγμα Εκτέλεσης



Αλγόριθμος του Kruskal

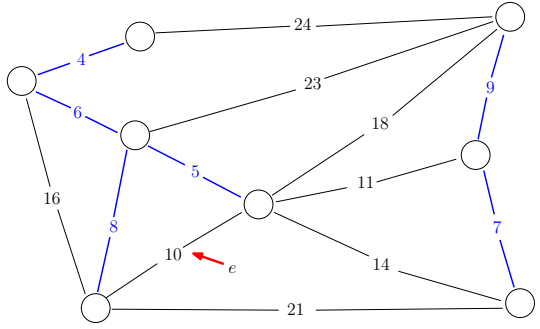
Ορθότητα

Θεώρημα

Ο αλγόριθμος του Kruskal παράγει ένα ελάχιστο γεννητικό δέντρο του G.

Απόδειξη

Θα εξετάσουμε τις ακμές με την σειρά που τις εξετάζει και ο αλγόριθμος. Έστω λοιπόν η ακμή $e = (v, w) \in E$ την ώρα που ο αλγόριθμος εξετάζει την ένταξη της στο δέντρο.



Περίπτωση 1. Εαν η ακμή δημιουργεί κύκλο, τότε είναι η πιο ακριβή ακμή του κύκλου (γιατί;) και άρα από την ιδιότητα κύκλου δεν ανήκει σε κανένα MST. □

Αλγόριθμος του Kruskal

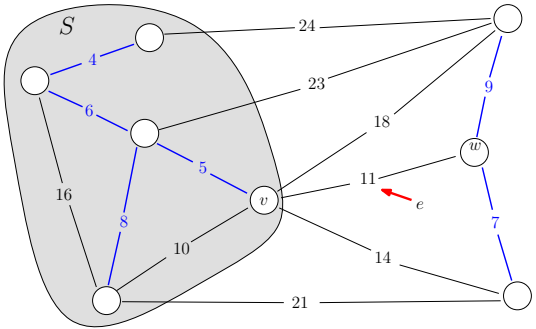
Ορθότητα

Θεώρημα

Ο αλγόριθμος του Kruskal παράγει ένα ελάχιστο γεννητικό δέντρο του G.

Απόδειξη

Θα εξετάσουμε τις ακμές με την σειρά που τις εξετάζει και ο αλγόριθμος. Έστω λοιπόν η ακμή $e = (v, w) \in E$ την ώρα που ο αλγόριθμος εξετάζει την ένταξη της στο δέντρο.



Περίπτωση 2. Εαν η ακμή δεν δημιουργεί κύκλο τότε ας ορίσουμε το σύνολο S ως το σύνολο των κόμβων για τους οποίους ο κόμβος v έχει διαδρομή. Η ακμή e είναι η ελάχιστου κόστους ακμή με ακριβώς ένα κόμβο στο σύνολο S και άρα ανήκει στο MST λόγω ιδιότητας τομής.

Μια αποδοτική υλοποίηση του αλγορίθμου χρειάζεται δύο μέρη:

1. ταξινόμηση ακμών
   - στην γενική περίπτωση χρειάζεται χρόνο $\mathcal{O}(m \log m) = \mathcal{O}(m \log n)$
2. αναγνώριση κύκλων
   - μπορούμε να ελέγχουμε για ύπαρξη κύκλων εκτελώντας διάσχιση DFS
   - είναι όμως πολύ αργό μιας και οδηγεί σε $\mathcal{O}(n)$ χρόνο για έναν έλεγχο και άρα σε $\mathcal{O}(nm)$ αλγόριθμο
   - θα δούμε μια καινούρια δομή δεδομένων που μας βοηθάει

Η Δομή Union-Find

Η δομή Union-Find μας επιτρέπει να διατηρήσουμε **ξένα σύνολα** (όπως για παράδειγμα οι συνεκτικές συνιστώσες ενός γραφήματος).

Η δομή μας παρέχει 3 βασικές λειτουργίες

1. *MakeSet*(x) – δημιουργεί το σύνολο $\{x\}$
2. *Find*(x) – επιστρέφει το όνομα (έναν αντιπρόσωπο) του συνόλου που περιέχει το στοιχείο x
3. *Union*(x, y) – συγχωνεύει τα ξένα σύνολα στα οποία ανήκουν τα στοιχεία x και y

Όταν μιλάμε για το όνομα ενός συνόλου, έχουμε αρκετή ελευθερία. Αυτό που μας ενδιαφέρει είναι να είναι συνεπή, με την έννοια πως *Find*(v) και *Find*(w) πρέπει να επιστρέφουν το ίδιο όνομα αν τα v και w ανήκουν στο ίδιο σύνολο.

Αλγόριθμος του Kruskal

Με δομή Union-Find

Αλγόριθμος του Kruskal για MST

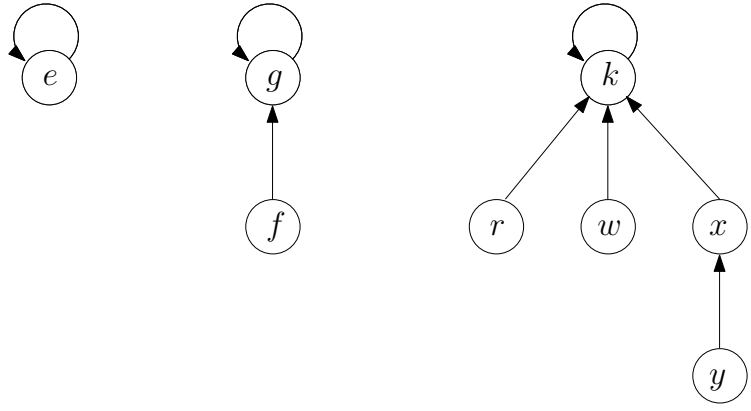
Ταξινόμησε τις ακμές σε αύξουσα σειρά με βάση τα βάρη c_e
$T \leftarrow \emptyset$
for $v \in V$ **do**
  MakeSet(v)
end
while *δεν έχουμε δει όλες τις ακμές* **do**
  έστω $e = (v, w)$ η επόμενη ακμή στην ταξινομημένη σειρά
  if *Find*$(v) \neq$ *Find*(w) **then**
    $T \leftarrow T \cup \{e\}$
    Union(v, w)
  end
end
return *το σύνολο* T

Υλοποίηση της Δομής Union-Find

Θα δείξουμε μια υλοποίηση με δείκτες.

- κάθε σύνολο θα είναι ένα προσανατολισμένο δέντρο
- κόμβοι του δέντρου είναι τα στοιχεία του συνόλου (χωρίς κάποια ιδιαίτερη σειρά)
- κάθε κόμβος έχει ένα γονικό δείκτη που τελικά οδηγεί στη ρίζα του δέντρου
- το στοιχείο στη ρίζα του δέντρου είναι ο *αντιπρόσωπος* ή το *όνομα* του συνόλου
- ο γονικός δείκτης στην ρίζα δείχνει στον εαυτό του
- για λόγους που θα προκύψουν σύντομα κάθε κόμβος γνωρίζει το ύψος (ή το μέγεθος) του υποδέντρου που κρέμεται από αυτόν

Υλοποίηση της Δομής Union-Find



Αναπαράσταση προσανατολισμένου δέντρου τριών συνόλων

1. $\{e\}$
2. $\{g, f\}$
3. $\{k, r, w, x, y\}$

Υλοποίηση της Δομής Union-Find

MakeSet(x)

$\pi(x) = x$
$rank(x) = 0$

Η *MakeSet*(x) απλά δημιουργεί ένα κόμβο ο οποίος δείχνει στον εαυτό του.

Find(x)

while $x \neq \pi(x)$ **do**
  $x = \pi(x)$
end
return x

Η *Find*(x) ξεκινά από τον κόμβο του στοιχείου x και διασχίζει το δέντρο προς τα πάνω μέχρι να βρει και να επιστρέψει την ρίζα. Η ρίζα είναι το όνομα ή ο αντιπρόσωπος του συνόλου στο οποίο ανήκει το στοιχείο x.

Υλοποίηση της Δομής Union-Find

Union by Rank

Η υλοποίηση της *Union*(x, y) γίνεται εύκολα

1. βρίσκουμε τους δύο αντιπροσώπους με *Find*(x) και *Find*(y)
2. κρεμάμε το ένα δέντρο από το άλλο
3. για να μην καταλήξουμε σε δέντρα με μεγάλο ύψος (το οποίο καθυστερεί τον χρόνο της αναζήτησης) κρεμάμε το μικρότερο δέντρο κάτω από τη ρίζα του μεγαλύτερου

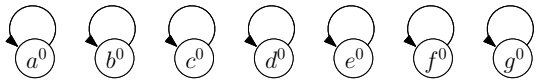
Union(x,y)

$r_x = Find(x)$
$r_y = Find(y)$
if $r_x = r_y$ **then**
  return
end
if $rank(r_x) > rank(r_y)$ **then**
  $\pi(r_y) = r_x$
else
  $\pi(r_x) = r_y$
  if $rank(r_x) = rank(r_y)$ **then**
    $rank(r_y) = rank(r_y) + 1$
  end
end

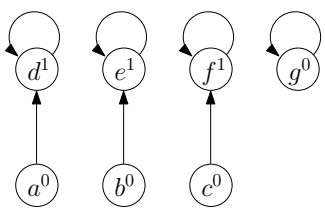
Union-Find

Παράδειγμα

MakeSet(a), MakeSet(b), ..., MakeSet(g)



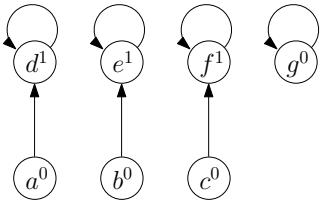
Union(a,d), Union(b,e), Union(c,f)



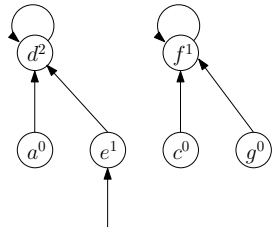
Union-Find

Παράδειγμα

Union(a,d), Union(b,e), Union(c,f)



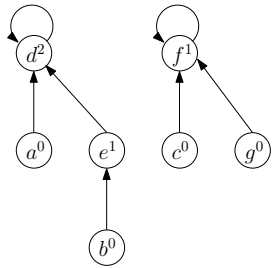
Union(c,g), Union(e,a)



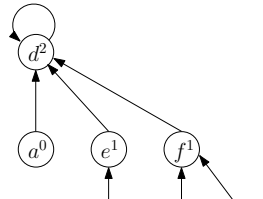
Union-Find

Παράδειγμα

Union(c,g), Union(e,a)



Union(b,g)



Union-Find by Rank

Χρόνοι Εκτέλεσης

1. η *MakeSet*(x) έχει προφανώς σταθερό χρόνο εκτέλεσης $\mathcal{O}(1)$
2. η συνάρτηση *Find*(x) εξαρτάται από το ύψος του δέντρου όπου βρίσκεται το στοιχείο x
3. η συνάρτηση *Union*(a, b) το ίδιο

Θα δούμε τώρα γιατί κρατάμε σε κάθε κόμβο την τάξη (rank) που είναι το ύψος του υποδέντρου το οποίο έχει ρίζα τον κόμβο αυτό.

Union-Find by Rank

Ιδιότητες Τάξεων (rank)

Ιδιότητα

Για έναν κόμβο x έχουμε $\mathit{rank}(x) < \mathit{rank}(\pi(x))$.

Ιδιότητα

Ένας ριζικός κόμβος τάξης k έχει τουλάχιστον 2^k κόμβους στο δέντρο του.

Απόδειξη

Θα χρησιμοποιήσουμε επαγωγή στην τάξη του κόμβου. Για $k = 0$ και $k = 1$ είναι προφανές αφού έχει αντίστοιχα 1 και 2 κόμβους στο δέντρο.

Έστω πως ισχύει για $k - 1$. Για να δημιουργηθεί ένας κόμβος τάξης k πρέπει να ενωθούν κόμβοι τάξης $k - 1$. Το δέντρο που προκύπτει έχει (λόγω υπόθεσης επαγωγής) τουλάχιστον $2 \cdot 2^{k-1} = 2^k$ κόμβους. □

Η ιδιότητα αυτή ισχύει και για μη ριζικούς κόμβους, αφού κάθε κόμβος ήταν ρίζα κάποτε. Μόλις ένας κόμβος πάψει να είναι ρίζα, η τάξη του αλλά και ο αριθμός των απογόνων του δεν αλλάζει.

Union-Find by Rank

Ιδιότητες Τάξεων (rank)

- Λόγω της πρώτης ιδιότητας ένα στοιχείο έχει το πολύ έναν πρόγονο τάξης k.
- Αυτό σημαίνει πως διαφορετικοί κόμβοι τάξης k δεν μπορούν να έχουν κοινούς απογόνους.
- Καταλήγουμε λοιπόν στο εξής συμπέρασμα.

Ιδιότητα

Αν συνολικά υπάρχουν n στοιχεία, τότε μπορεί να υπάρχουν το πολύ $\frac{n}{2^k}$ κόμβοι τάξης k.

- Η τελευταία ιδιότητα σημαίνει πως η μέγιστη δυνατή τάξη είναι $\log n$, και άρα όλα τα δέντρα έχουν ύψος $\leq \log n$.

Union-Find by Rank

Χρόνοι Εκτέλεσης

1. η *MakeSet*(x) έχει προφανώς σταθερό χρόνο εκτέλεσης $\mathcal{O}(1)$
2. η συνάρτηση *Find*(x) εξαρτάται από το ύψος του δέντρου όπου βρίσκεται το στοιχείο x
3. η συνάρτηση *Union*(a, b) το ίδιο

Το ύψος ενός δέντρου είναι $\leq \log n$ και άρα οι *Find*(x) και *Union*(a, b) πέρνουν χρόνο $\mathcal{O}(\log n)$.

Αλγόριθμος του Kruskal

Με δομή Union-Find

Αλγόριθμος του Kruskal για MST

Ταξινόμησε τις ακμές σε αύξουσα σειρά με βάση τα βάρη c_e
$T \leftarrow \emptyset$
for $v \in V$ **do**
  MakeSet(v)
end
while *δεν έχουμε δει όλες τις ακμές* **do**
  έστω $e = (v, w)$ η επόμενη ακμή στην ταξινομημένη σειρά
  if *Find*(v) $\neq$ *Find*(w) **then**
    $T \leftarrow T \cup \{e\}$
    Union(v, w)
  end
end
return *το σύνολο* T

- $\mathcal{O}(m \log m) = \mathcal{O}(m \log n)$ για ταξινόμηση των ακμών
- $\mathcal{O}(n)$ χρόνο για να κατασκευάσει όλα τα αρχικά σύνολα
- $\mathcal{O}(m \log n)$ για όλα τα *Find*(x)
- $\mathcal{O}(m \log n)$ για όλα τα *Union*(a, b)

σύνολο $\mathcal{O}(m \log n)$

Απαλοιφή Υπόθεσης για Διαφορετικά Βάρη

Εαν έχουμε ένα πρόβλημα όπου υπάρχουν ακμές με ίδια βάρη

- ”διαταρράσουμε” τα βάρη των ακμών προσθέτοντας ή αφαιρώντας διαφορετικούς, εξαιρετικά μικρούς αριθμούς
- με αυτό τον τρόπο όλα τα βάρη γίνονται διαφορετικά
- φροντίζουμε οι διαταραχή αυτή να μην αλλάζει την σχετική σειρά των δέντρων που δεν είχαν ίδιο αρχικό κόστος

Αυτό είναι ίδιο με το να εκφράσουμε συμβολικά αυτή την διαταραχή, επιλύοντας τις διαφορές λεξικογραφικά:

```
bool less(i, j)
{
    if ( cost(ei) == cost(ej) )
        return i < j;
    else
        return cost(ei) < cost(ej);
}
```