Αλγόριθμοι και Πολυπλοκότητα

3η Σειρά Γραπτών Ασκήσεων

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Αμπατζή Ναυσικά (03117198)
7ο Εξάμηνο - Ροή Λ
Ακαδημαϊκό Έτος 2020-2021

1 Άσκηση 1: Ραντεβού μετά το lockdown

Στόχος αυτού του προβήματος είναι η εύρεση του συντομότερου μονοπατιού. Έτσι, θα βασιστούμε στον αλγόριθμο BFS, τον οποίο θα παραλλάξουμε κάπως. Αρχικά ο αλγόριθμος BFS προϋποθέτει ότι ο γράφος G(V,E) αναπαρίσταται μέσω λιστών γειτνίασης. Σε κάθε κόμβο του γραφήματος θα προσταρτίσουμε διάφορα επιπλέον πεδία.
Αρχικά θεωρούμε ότι υπάρχουν 3 είδη κορυφών:

- Ανεξερεύνητη (A) : Δεν την έχουμε επισκευθεί ακόμα.
- Υπό εξέταση (ΥΕ) : Την έχουμε επισκευθεί, αλλά δεν έχουν εξερευνηθεί οι γείτονές της.
- Εξερευνημένη (E) : Την έχουμε επισκευθεί και έχουν εξερευνηθεί και οι γείτονές της.

Η κατάσταση της κάθε κορυφής αποθηκεύεται στο πεδίο **u.status** και η απόσταση από τον αφαιτηριακό κόμβο s μέχρι τον u αποθηκεύεται στο πεδίο **u.when**. Τέλος ο προκάτοχος της u αποθηκεύεται στο πεδίο **u.π**. Εάν ο u δεν έχει προκάτοχο(πχ. εάν u=s ή ο u δεν έχει εντοπιστεί), τότε u.π = κενό.
Ο αλγόριθμος χρησιμοποιεί επίσης, μία ουρά FIFO για τη διαχείριση των υπό εξέταση κορυφών. Για το πρόβλημα αυτό κάθε φορά που μία κορυφή μπαίνει στην ουρά, θα αποθηκεύεται στο πεδίο **u.αναμονή** η τιμή 1,σε περίπτωση που θα πρέπει να παραμείνουμε εκεί για μία χρονική στιγμή. Διαφορετικά θα εποθηκέυεται η τιμή 0.

Αλγόριθμος

1. Για κάθε κόμβο $u \in G.V$ - s θέτουμε:
   - u.status = A, u.when = ∞ και u.π = κενό
2. Θέτουμε τις καταστάσεις για την κορυφή s ως εξής:
   - s.status = ΥΕ, s.when = 0, s.π = κενό

   Προσθέτουμε την κορυφή s στην ουρά. Το πεδίο u.αναμονή παίρνει την τιμή 0.
3. Μέχρι να γίνει η ανάθεση του πεδίου when στην κορυφή t:
   Υποθέτουμε ότι στην κεφαλή της ουράς βρίσκεται ο κόμβος u και τον αφαιρούμε. Θα εξεταστεί ο κόμβος u στην λίστα γειτνίασης του:
   - Κάθε γείτονα v της u (με κατεύθυνση από την u στον v), με v.status = A (δεν τον έχουμε επισκευθεί ακόμα) τον προσθέτουμε στην ουρά και θέτουμε v.status = ΥΕ, v.αναμονή = false και v.when = u.when + u.αναμονή + 1.
   - Εάν ο χρόνος αναμονής ήταν 0 (u.αναμονή = 0), αλλά παρατηρήσουμε ότι υπάρχει κάποιος κόμβος v που συνδέεται με την u αλλά η κατεύθυνση είναι από τον v στην u, τότε πρέπει να προσθέσουμε ξανά την u στην ουρά, ώστε να εξεταστεί και αυτός. Προσθέτοντας ξανά την u στην ουρά ορίζουμε το πεδίο u.αναμονή = 1.
4. Το πεδίο t.when μεταφέρεται σε ένα πεδίο $t.when_1$ και επαναλαμβάνεται η παραπάνω διαδικασία με τη διαφορά ότι στο βήμα 2, το πεδίο u.αναμονή παίρνει την τιμή 1. Το νέο αποτέλεσμα το μεταφέρουμε ξανά σε ένα νέο πεδίο $t.when_2$

Ορίζουμε ως $time_{start}$ την χρονική στιγμή της άφιξης από την κορυφή s.
Εάν τουλάχιστον ένα από τα αθροίσματα $time_{start}$ + t.time1 και $time_{start}$ + t.time2 είναι μικρότρο ή ίσο από τη χρονική στιγμή T. Τότε κρατάμε το μεγαλύτερο $time_{start}$. Εάν καμία από τις δύο συνθήκες δεν ικανοποιείται, τότε δεν υπάρχει στιγμή άφιξης που να λύνει το πρόβλημα.

Απόδειξη Ορθότητας
Ο παραπάνω αλγόριθμος βρίσκει τα συντομότερα (χρονικά) μονοπάτια από μία κορυφή s. Εφόσον έχουμε στηριχθεί στον αλγόριθμο BFS, προσθέτοντας ορίσμενα πεδία ακόμα (.when και .αναμονή) και εξετάζοντας τα σε κάθε επανάληψη, ας θεωρήσουμε ότι η ορθότητα για τα υπόλοιπα πεδία και την εξέταση τους ισχύει.
Στον αλγόριθμο αυτό πραγματοποιούνται δύο εκτελέσεις μίας διαδικασίας που στηρίζεται στον BFS, μία ξεκινώντας από άρτια χρονική στιγμή και μία από περιττή. Αυτό είναι απαραίτητο διότι σε κάθε χρονική στιγμή οι κατευθύνσεις των ακμών αλλάζουν. Έτσι ο αρχικός γράφος θα είναι διαφορετικός εάν ξεκινήσουμε άρτια χρονική στιγμή ή περιττή.
Πολυπλοκότητα
Ο παραπάνω αλγόριθμος αποτελεί μία παραλλαγή του BFS, η οποία θα εκτελεστεί δύο φορες (ξεκινώντας από άρτια και περιττή χρονική στιγμή). Η διαφορά όσν αφορά τον χρόνο εκτέλεσης είναι ότι κάθε κόμβος θα προστεθί στην ουρά το πολύ δύο φορές (αντί για μία), καθώς οι κατευθύνσεις των κόμβων είναι μεταβλητές κάθε χρονική στιγμή. Επομένως θα αφαιρεθεί το πολύ δύο φορές από την ουρά. Έτσι, κατ'αντιστοιχία με τον BFS, ο συνολικός χρόνος που αναλώνεται στις πράξεις της ουράς είναι $O(2 * V)$. Η λίστα γειτνίασης κάθε κόμβου διατρέχεται όταν ο κόμβος αφαιρεθεί από την ουρά, με αποτέλεσμα κάθε τέτοια λίστα να διατρέχεται το πολύ δύο φορές. Δεδομένου ότι το άθροισμα των μηκών όλων των λιστών γειτνίασης είναι Θ(E), θα χρειαστεί χρόνος $O(2 * E)$. Τέλος, οι αρχικές τιμές αποδίδονται σε χρόνο O(V). Άρα η χρονική πολυπλοκότητα του παραπάνω αλγορίθμου προκύπτει ίση με O(V+E).

2 Άσκηση 2: Προγραμματίζοντας την αντίδραση

Για την επίλυση αυτού του προβλήματος θα χρησιμοποιήσουμε ξανά μία παραλλαγή του αλγορίθμου BFS.Αρχικά ο αλγόριθμος BFS προϋποθέτει ότι ο γράφος G(V,E) αναπαρίσταται μέσω λιστών γειτνίασης. Σε κάθε κόμβο του γραφήματος θα προσταρτίσουμε διάφορα επιπλέον πεδία.
Η κατάσταση της κάθε κορυφής αποθηκεύεται στο πεδίο **u.status** (από 1η άσκηση A,ΥE,E) και η απόσταση από τον αφαιτηριακό κόμβο s μέχρι τον u αποθηκεύεται στο πεδίο **u.dist**. Ο προκάτοχος της v αποθηκεύεται στο πεδίο **v.u.π**, εάν το μονοπάτι προς αυτή είχε ως αφετηρία την κορυφή u.
Ο αλγόριθμος χρησιμοποιεί επίσης, μία ουρά FIFO για τη διαχείριση των υπό εξέταση κορυφών. Για το πρόβλημα αυτό κάθε φορά που μία κορυφή μπαίνει σε μία ουρά FIFO ένα νέο πεδίο, το **u.counter**(το οποίο αρχικά έχει αποθηκευμένη την τιμή μηδέν) θα αυξάνεται κατά ένα. Τέλος ορίζουμε ένα νέο πεδίο, το**u.v.άφιξη**, όπου περιέχει την τιμή true εάν έχουμε ορίσει μονοπάτι με αφετηρία την u προς την v, αλλιώς θα περιέχεται η τιμή false.

Αλγόριθμος
Αρχικά αναζητούμε τις k κορυφές του γράφου όπου υπάρχουν σωματίδια και τις προσθέτουμε -αυξάνοντας κατά 1 το πεδίο u.counter- σε k ουρές FIFO. Για κάθε μία από τις k ουρές FIFO επαναλαμβάνουμεσ:

- Ελέγχουμε εάν το πεδίο u.counter περιέχει την τιμή k. Εάν ναι κάνουμε jump στο Τέλος. Διαφορετικά, έστω ότι η κεφαλή της κάθε ουράς, όπου έχουμε φτάσει μέσω ενός μονοπατιού με αφετηρία την κορυφή u(κάθε ουράς) είναι η v . Την αφαιρούμε και αναζητούμε τους γείτονές της, έστω n, με n.status = A. Κάθε γείτονα n τον προσθέτουμε στην ουρά, αφού αποθηκεύσουμε στο πεδίο n.counter την κατά 1 αυξημένη τιμή, στο πεδίο n.u.π την κεφαλή v, στο πεδίο n.status την τιμή ΥΕ και στο πεδίο n.άφιξη την τιμή true.

Τέλος:
Η ζητούμενη κορυφή είναι η κεφαλή v στην οποία φτάσαμε μέσω ενός μονοπατιού με αφετηρία την κορυφή u. Από τα πεδία v.u.π μπορούμε να βρούμε τα μονοπάτια που ακολουθήθηκαν.

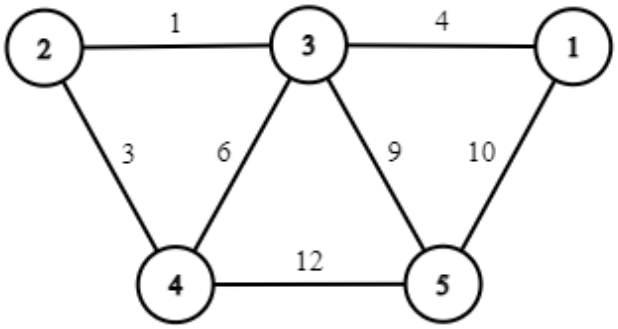
Απόδειξη Ορθότητας
Κατ'αρχάς ο παραπάνω αλγόριθμος βασίζεται στην ιδέα ότι, η ζήτουμενη κορυφή θα απέχει από κάθε αφετηρία είτε απόσταση d, είτε απόσταση d+1, ώστε να εξασφαλιστεί η ταχύτερη άφιξη σε αυτή. Η ιδέα είναι η ταυτόχρονη εκτέλεση k BFS, ώστε να σχηματίζουμε ίσες διαδρομές σε κάθε βήμα για όλες τις κορυφές u. Εφόσον κάθε φορά που μία κορυφή μπαίνει στην FIFO ουρά αποθηκεύουμε στο πεδίο n.status την τιμή ΥΕ, γνωρίζουμε ότι έτσι δε θα πρέπει να την επισκευτούμε ξανά. Με τον μετρητή εξασφαλίζουμε ότι όταν το πεδίο n.counter πάρει την τιμή k, θα έχουμε επισκευθεί όλες τις κορυφές με σωματίδια.

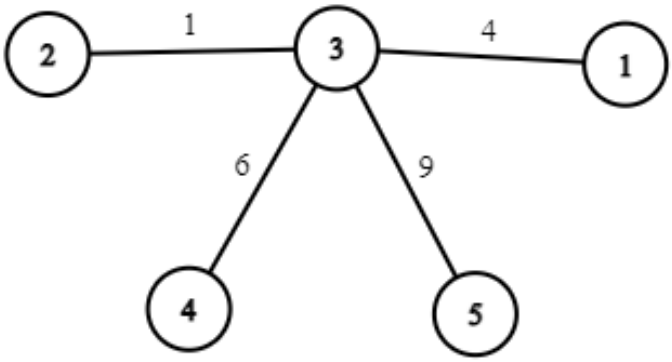
Πολυπλοκότητα
Η ιδέα του αλγορίθμου βασίζεται στον BFS, με τη διαφορά ότι στη χειρότερη περίπτωση θα πρέπει να εξεταστούν όλες οι κορυφές του G(V,E), για κάθε αφετηρία. Επομένος η πολυπλοκότητα είναι O(kV+kE). Επειδή, όμως σύμφωνα με την εκφώνηση το πλήθος όλων των κορυφών του γράφου είναι πολύ μεγαλύτερο από το πλήθος των κορυφών όπου θα υπάρχουν σωματίδια, τελικά η πολυπλοκότητα του παραπάνω αλγορίθμου προκύπτει ίση με O(V+E).

3 Άσκηση 4: Ελάχιστο Συνδετικό Δέντρο με Περιορισμούς

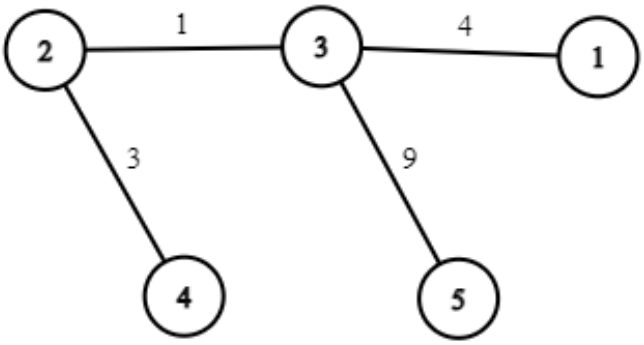
3.1 Διατύπωση του Greedy Αλγορίθμου

1. Με χρήση ενός αντιπαραδείγματος θα δειχθεί, ότι η άπληστη στρατηγική, η οποία συμπεριλαμβάνει στο συνδετικό δέντρο τις k ακμές μικρότερου βάρους που προσπίπτουν στην s, δεν οδηγεί πάντα στη βέλτιστη λύση. Θεωρούμε το ακόλουθο συνεκτικό μη κατευθυνόμενο γράφημα με 5 κορυφές, 6 ακμές και θετικά βάρη στις ακμές:
   Έστω ότι θέλουμε η κορυφή 3 να έχει βαθμό k = 4. Μία πιθανή επιλογή συνδετικού δέντρου είναι η εξής:





Ωστόσο το ελάχιστο συνδετικό δέντρο είναι το εξής:



Έτσι συμπαιρένουμε ότι δεν μπορούμε να χρησιμοποιήσουμε τον greedy αλγόριθμο για να υπολογίσουμε ένα Ελάχιστο Συνδετικό Δέντρο T*(s,k) του G στο οποίο μία συγκεκριμένη κορυφή s έχει βαθμό ίσο με k.

2. Τώρα θα διατυπώσουμε έναν αλγόριθμο, που επιλύει το παραπάνω πρόβλημα.

Θα θεωρήσουμε τις $u_0, u_1, ..., u_k$ ως ρίζες του δέντρου και ό,τι βρίσκεται κάτω από αυτές θα αποτελεί απόγονό τους.
Θεώρημα
Εάν (u_0, u_1) είναι η εκμή ελαχίστου βάρους ανάμεσα στις προσπίτουσες στην κορυφή u_0, τότε ένα ελάχιστο συνδετικό δέντρο για το G, τέτοιο ώστε $\deg(u_0) = k$, θα περ-

ιέχει την ακμή (u_0, u_1).
Απόδειξη
Έστω T(k), ένα ελάχιστο συνδετικό δέντρο με $\deg(u_0) = k$. Εάν το T δεν περιέχει την παραπάνω ακμή, υποθέτουμε ότι στο δέντρο T η κορυφή u_1 είναι ένας απόγονος της ρίζας u_2.Εάν η ακμή (u_0, u_1) δεν είναι η ελαχίστου κόστους προσπίπτουσα στην κορυφή u_0, τότε εάν αφαιρέσουμε την ακμή (u_0, u_2) και προσθέσουμε την ακμή (u_0, u_1), θα προκύψει ένα δέντρο με $\deg(u_0) = k$ και κόστος μικρότερο από T(k). Αυτό όμως έρχεται σε αντίθεση με την υπόθεση ότι το T(k) είναι το ελάχιστο συνδετικό δέντρο. Έτσι συμπεραίνουμε ότι, η ακμή (u_0, u_2) είναι, όπως και η (u_0, u_1), προσπίπτουσα ακμή στο u_0 με ελάχιστο κόστος. Τελικά εάν αφαιρέσουμε την ακμή (u_0, u_2) και προσθέσουμε την ακμή (u_0, u_1), θα προκύψει ένα δέντρο με $\deg(u_0) = k$, το οποίο είναι και το ελάχιστο συνδετικό.
Αλγόριθμος

(a) Χρησιμοποιώντας τον αλγόριθμο του Prim, βρίσκουμε ένα ελάχιστο συνδετικό δέντρο T, στο υπογράφημα G που προκύπτει εάν αφαιρέσουμε από το αρχικό γράφημα την κορυφή u_0.
(b) Έστω ότι η ακμή (u_0, u_1) είναι η ελαχίστου κόστους ακμή που προσπίπτει στην κορυφή u_0. Προσθέτουμε αυτή την ακμή(και την κορυφή u_1 στο γράφημα.
(c) Όσο το πλήθος των ριζών είναι διαφορετικό του k:

- Για κάθε κορυφή u, τέτοια ώστε η ακμή (u_0, u) υπάρχει και η u δεν είναι ρίζα, θεωρούμε e την ακμή μέγιστου κόστους, όπου δεν περιέχει την κορυφή u_0 στο μονοπάτι u...u_0.
- Υπολογίζουμε το c'(u) = c(u_0,u) - c(e). Έστω u μία κορυφή με ελάχιστο c'(u):
- Τότε αφαιρούμε την e από το δέντρο και προσθέτουμε την κορυφή u.

Πρόταση:
Έστω T(k) ένα minimum spanning tree του G, με $\deg(v_0) = k$ και $u_1, ..., u_k$ οι κορυφές του. Τότε το δέντρο T(k+1) είναι ένα επίσης minimum spanning tree με κορυφές $(u_1, ..., u_k, u_{k+1})$.

Αλγόριθμος 1
Θα περιγράψουμε έναν αλγόριθμο για την εύρεση ενός minimum partition forest(F = (V,E')) με ρίζες $u_1, ..., u_p$ σε έναν συνδεδεμένο γράφο G(V,E,c).
Βρίσκουμε ένα ελάχιστο συνδετικό δέντρο (με έναν από τους γνωστούς αλγορίθμους) T = (V,S) για τον γράφο G.Ορίζουμε ως κορυφή την u_1 και θεωρούμε ότι E' = T.
Για i = 2 μέχρι p:

- Θεωρούμε ότι η u_s είναι η ρίζα, όπου η u_i είναι συνδεδεμένη στο δάσος F = (V,E').
- Έστω ότι e είναι η μεγαλύτερου κόστους ακμή στο μονοπάτι $u_i...u_s$.
- Την αφαιρούμε.
- Προσθέτουμε στις ρίζες την u_i.

Απόδειξη ορθότητας του Αλγορίθμου:
Για την απόδειξη ορθότητας θα χρησιμοποιήσουμε επαγωγή.

- Για k = 1, ο ισχυρισμός εμφανώς ισχύει.
- Υποθέτουμε ότι στο βήμα k, το δέντρο T(k) είναι ένα minimum spanning tree με $\deg(v_0)$ = k. Έστω ότι μετά από μία ακόμα επανάληψη του παραπάνω αλγορίθμου προκύπτει το δέντρο T(k+1). Υποθέτουμε ότι αυτό δεν είναι ελάχιστο συνδετικό δέντρο, αλλά ένα άλλο, έστω το T'(k+1) με $\deg(v_0)$ = k+1. Έστω ότι $u_1, ..., u_k$ είναι οι ρίζες του T(k), τότε από την παραπάνω πρόταση υποθέτουμε ότι οι κορυφές $u_1, ..., u_k, u_{k+1}$ ανήκουν στο δέντρο T'(k+1).
  Οι κορυφές του T(k) ορίζουν ένα ελάχιστο δάσος με ρίζες $u_1, ..., u_k$. Συνεπώς μπορούμε να θεωρήσουμε από τον Αλγόριθμο1, ότι αυτό το δάσος μπορεί να προκύψει από το αρχικό αφαιρώντας την μεγαλύτερου κόστους ακμή και συνδέοντας την κορυφή u στη ρίζα της. Το T(k+1) προκύπτει από το T(k), εάν προσθέσουμε την ακμή (u_0, u_{k+1}) και αφαιρέσουμε την κορυφή b, συνδέοντας στο T(k) την κορυφή u_{k+1} στη ρίζα. Γνωρίζοντας ότι αυτή η διαδικασία έχει το ελάχιστο κόστος, προκύπτει ότι:
  c(T(k+1)) = c(T(k)) + c(u_0, u_{k+1}) - c(b) ≤ c(T(k)) + c(u_0,u) - c(a) = c(T'(k+1)).
  Άρα εφόσον το T'(k+1) είναι ελαχίστου κόστους, τότε θα είναι και το T(k+1).

4 Άσκηση 5: Αλγόριθμος Boruvka

1. Απόδειξη ορθότητας

Θεωρούμε ένα συνεκτικό μη κατευθυνόμενο γράφημα G(V,E,w), με n κορυφές, m ακμές και θετικό βάρος w(e) σε κάθε ακμή που ανήκει στο E, όπου το βάρη των ακμών είναι όλα διαφορετικά μεταξύ τους. Υποστηρίζουμε, ότι αλγόριθμος Boruvka βρίσκει ένα minimum spanning tree.

Η ιδέα του αλγορίθμου:
Διαλέγει σε κάθε βήμα την λίστα γειτόνων κάθε κορυφής και ταυτόχρονα για όλες τις λίστες μαζί επιλέγει την ακμή που προσπίπτει σε κάθε κορυφή, η οποία είναι σίγουρα ακμή του ελάχιστου συνδετικού δέντρου. Τοποθετεί την ακμή στο δάσος και επαναλαμβάνει το ίδιο.

(a) Αρχικά θα αποδείξουμε ότι ο αλγόριθμος καταλήγει πάντα σε ένα συνδετικό δέντρο:
Υποθέτουμε ότι ο αλγόρθμος δεν καταλήγει πάντα σε συνδετικό δέντρο, αλλά δύναται να προκύψει κύκλος.
Έστω ότι ο κύκλος έχει σχηματιστεί από τις ακμές e1,...,ek και e1 η ελάχιστου κόστους προσπίπτουσα ακμή. Έτσι για να επιλεγεί η ακμή ek, σύμφωνα με τον αλγόριθμο θα πρέπει να ισχύει ότι: $w(e1) < ... < w(ek) < w(e1)$, κάτι το οποίο είναι εμφανώς άτοπο.
Επομένως ο αλγόριθμος οδηγεί πάντα σε ένα συνδετικό δέντρο(συνεκτικός και χωρίς κύκλους γράφος).

(b) Τώρα θα αποδείξουμε ότι το συνδετικό δέντρο είναι ελαχίστου κόστους:
Θεωρούμε ότι η υπόθεση αυτή δεν ισχύει και ότι υπάρχει τομή (S, $\overline{S}$), ώστε το minimum spanning tree T να μην περιέχει την ελαχίστου βάρους ακμή e'. Έτσι εάν προσθέσουμε το e' στο T θα δημιουργήσουμε κύκλο. Συμπεραίνουμε λοιπόν ότι υπάρχει κάποια άλλη ακμή στο T που διασχίζει την τομή. Επειδή κάθε βάρος έχει μοναδική τιμή, για όλες τις ακμές $e \in C \cap (S\overline{S})$, $e \neq e'$, ισχύει ότι $w(e) > w(e')$. Έτσι εάν στη θέση της e βάλουμε την e' θα έχουμε ένα μικρότερου βάρους συνδετικό δέντρο, το οποίο είναι άτοπο.
Συνεπώς ο αλγόριθμος Boruvka οδηγεί πάντα σε ένα minimum spanning tree.

2. **Υλοποίηση του αλγορίθμου σε χρόνο O(mlongn)**

Βήμα 1: Αρχικά ο αλγόριθμος δημιουργεί μία λίστα από n δέντρα, που αποτελούνται από έναν κόμβο.
Βήμα 2: Όσο η λίστα περιέχει περισσότερα από ένα κόμβο:

- Βήμα 2.1: Για κάθε δέντρο στη λίστα βρίσκει τον κόμβο που δεν συνδέεται με το δέντρο, με την μικρότερου κόστους προσπίπτουσα ακμή στο δέντρο.
- Βήμα 2.2: Προσθέτει όλες τις ακμές που βρέθηκαν στον νέο γράφο, δημιουργώντας ένα νέο σύνολο από δέντρα.

Σε κάθε βήμα ενώνονται δέντρα, μέχρι να προκύψει ένα τελικό δέντρο(minimum spanning tree).Εδώ το πλήθος των κορυφών κάθε φορά υποδιπλασιάζεται.Επομένως χρειαζόμαστε το πολύ logn επαναλήψεις για να έχουμε το ελάχιστο συνδετικό δέντρο.

Για το Βήμα 2 θα χρησιμοποιηθεί η δομή union find, η οποία αρχικά περιέχει όλες τις κορυφές. Τα σύνολά της είναι τα ήδη υπάρχοντα δέντρα που έχουν προκύψει από την εκτέλεση του αλγορίθμου. Αρχικά εκτελούμε το find για όλες τις ακμές, ώστε να βρούμε την μικρότερου κόστους ακμή που συνδέει κάθε ζεύγος δέντρων, τα οποία ενώνουμε με union. Ο χρόνος που απαιτείται είναι O(m).
Τελικά η χρονική πολυπλοκότητα του αλγορίθμου προκύπτει O(mlogn).

3. **Συνδυασμός Boruvka-Prim**
Αρχικά εκτελούμε το Βήμα 2 του αλγορίθμου Boruvka, έστω k φορές. Το κόστος θα είναι O(k*m) και θα έχουμε το πολύ $n/2^r$ κορυφές. Στη συνέχεια εκτελούμε τον αλγόριθμο του Prim, ο οποίος θα απαιτεί χρόνο O(n/2,logn/2). Άρα η χρονική πολυπλοκότητα προκύπτει ίση με O(k*m + n/2,logn/2) και εάν θέσουμε r=log(logn) προκύπτει το ζητούμενο O(m*log(logn))

4. **Βολική Κλάση Γραφημάτων**
Για την απόδειξη θα εκτελέσουμε τον αλγόριθμο Boruvka, αλλά σε κάθε επανάληψή του συμπτίσουμε τις ακμές και κρατάμε πάντα αυτή με το μικρότερο κόστος.
Εφόσον τα δέντρα πλέον είναι κόμβοι η ορθότητα του αλγορίθμου δεν "χαλάει". Η σύμπτιξη πραγματοποιείται σε χρόνο O(m). Στο k-οστό βήμα θα έχουμε το πολύ $E/2^{2k}$ ακμές. Από την ιδιότητα της κλειστότητας έχουμε ότι $|E_k| = O(V_k)$ Η ζητούμενη πολυπλοκότητα προκύπτει ως εξής: T(V) = T(V/2) + O(V) = O(V)

5 Άσκηση 6: Το σύνολο των Συνδετικών Δέντρων

1. Εάν από το T1 αφαιρέσουμε την ακμή e, τότε θα προκύψουν δύο συνεκτικές συνιστώσες V1 και V2. Εφόσον και το T2 είναι συνεκτικό, θα περιέχει κάποια ακμή που να ενώνει τις V1 και V2. Επιπλέον η ακμή e' δεν γίνεται να ανήκει στο T1, αφού θα σχηματιζόταν κύκλος ανάμεσα στις V1 και V2. Έτσι εάν προσθέσουμε την e' η συνεκτικότητα διατηρείται και προκύπτει ένα νέο συνδετικό δέντρο με n-1 ακμές.

   Αλγόριθμος:
   Βήμα 1: Αρχίζουμε από μία κορυφή u του T1 που προσπίπτει στην e = u,v και διασχίζουμε χρησιμοποιόντας τον αλγόριθμο DFS στην V. Βήμα 2: Εξετάζουμε για όλες τις ακμές e εάν υπάρχει κάποια που να ανήκει στο T2. Βήμα 3: Αφαιρούμε τις ακμές e που ανήκουν στο παραπάνω μονοπάτι, αλλά δεν ανήκουν στο δέντρο T2. Χρονική πολυπλοκότητα: Για το βήμα 1 θέλουμε χρόνο $O(|V|)$ και για τα βήματα 2 και 3 O(1). Άρα η πολυπλοκότητα είναι $O(|V|)$.

2. Θα αποδείξουμε ότι για δύο συνδετικά δέντρα του G, T1 και T2, για τα οποια ισχύει ότι $|T1 \setminus T2| = \text{dist}$, το μήκος του συντομότερου μονοπατιού που τα συνδέει στο H είναι ίσο με dist. Για την απόδειξη θα χρησιμοποιήσουμε επαγωγή:

   - Βάση: Από την εκφώνηση, η υπόθεση αληθεύει για dist = 1.
   - Υπόθεση: Υποτέτουμε ότι ο παραπάνω ισχυρισμός αληθεύει για $|T1 \setminus T2| =$ 1,2,...,dist.
   - Βήμα: Θα δείξουμε ότι ισχύει και για $|T1 \setminus T2| = \text{dist} + 1$. Αρχικά έστω ότι αφαιρούμε από το δέντρο T1 μία ακμή e. Από το προηγούμενο ερώτημα γνωρίζουμε ότι θα υπάρχει κάποια αμή e', που ανήκει στο T2 αλλά όχι στο T1, η οποία θα ενώσει ξανά την τομή που δημιουργήθηκε. Από την επαγωγική υπόθεση ισχύει ότι το νέο δέντρο που προέκυψε θα διαφέρει από το T2 κατά dist ακμές, κι έτσι το μήκος του συντομότερου μονοπατιού μεταξύ των T1 και T2 θα είναι μικρότερο ή ίσο του dist+1. Επίσης από την επαγωγική υπόυεση εάν το μήκος του συντομότερου μονοπατιού μεταξύ των T1 και T2 είναι dist, τότε τα T1 και T2 θα διαφέρουν μεταξύ τους κατά dist+1 ακμές. Αυτό όμως είναι άτοπο. Άρα συμπεραίνουμε ότι το μήκος του συντομότερου μονοπατιού μεταξύ των T1 και T2 θα είναι μεγαλύτερο ή ίσο του dist+1.
     Τελικά προκύπτει ότι το μήκος του συντομότερου μονοπατιού μεταξύ των T1 και T2 θα είναι ίσο με dist+1.

   Αλγόριθμος για το συντομότερο μονοπάτι
   Για να υπολογίσουμε ένα συντομότερο μονοπάτι στο H μεταξύ των T1 και T2, βρίσκουμε όλες τις ακμές(e) που ανήκουν στο T2, αλλά όχι στο T1. Τις ακμές αυτές τις αποθηκεύουμε σε μία λίστα και για κάθε ακμή αυτής της λίστας βρίσκουμε ακμή (e') που υπάρχει στο T1 αλλά όχι στο T2, χρησιμοπιώντας τον αλγόριθμο από το προηγούμενο ερώτημα. Στη συνέχεια αφαιρούμε την e' και προσθέτουμε την e στο T1.
   Απόδειξη ορθότητας
   Έστω ότι οι ακμές αυτές είναι n σε πλήθος. Τότε μετά από n βήματα θα έχουμε το

δέντρο T2. Έτσι έχουμε βρει και το συντομότερο μονοπάτι μεταξύ των T1 και T2. Τέλος για $|T1 \setminus T2|$ φορές επαναλαμβάνουμε τη διαδικασία.
Πολυπλοκότητα
$\overline{\mathrm{O(V + V^*|T1 \setminus T2|)/} = \mathrm{O(V^*|T1 \setminus T2|)}}$