Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Αλγόριθμοι και Πολυπλοκότητα

Αλγόριθμοι 3η Σειρά Γραπτών Ασκήσεων

Φοιτητής: Μπεκρής Δημήτρης
ΑΜ: 03117116

Άσκηση 1η:

Το οδικό δίκτυο αναπαρίσταται σε ένα κατευθυνόμενο γράφο, όπου οι ακμές του αλλάζουν κατεύθυνση ανάλογα το αποτέλεσμα της πράξης "**t mod 2",** όπου **t:** η αντίστοιχη χρονική στιγμή.

Για την επίτευξη αυτής της εναλλαγής, θα χρησιμοποιήσουμε 2 διαφορετικούς γράφους με αντίθετες κατευθύνσεις. Έστω "n" ο αριθμός των κόμβων και "m" ο αριθμός των ακμών. Σύμφωνα με την παραπάνω υλοποίηση θα χρειαστούμε **O(n+m)**, χωρική πολυπλοκότητα για την αποθήκεύση των γράφων μας, η οποία είναι γραμμική ως προς την είσοδο.

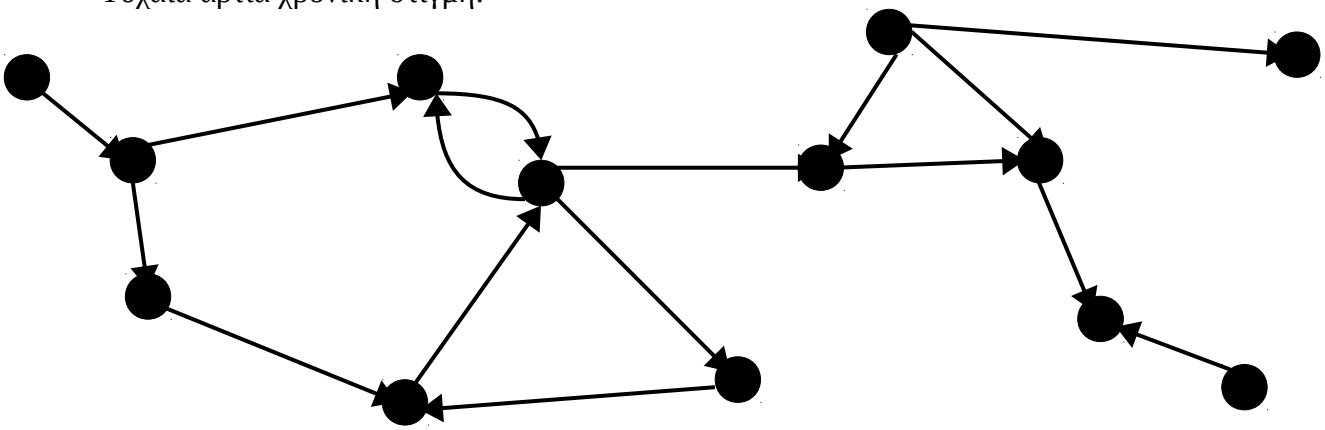
Η αποθήκευση θα γίνει με χρήση πίνακα απο λίστες γειτνίασης, όπου οι λίστες θα υλοποιηθούν με τη χρήση "vectors", με σκοπό την εκμετάλλευση της σταθερής χρονικά προσπέλασης των δεδομένων και τη σταθερή (κατά μέση τιμή)εισαγωγή στοιχείων.

Το πρόβλημα θα λυθεί με τις εξής παραδοχές:

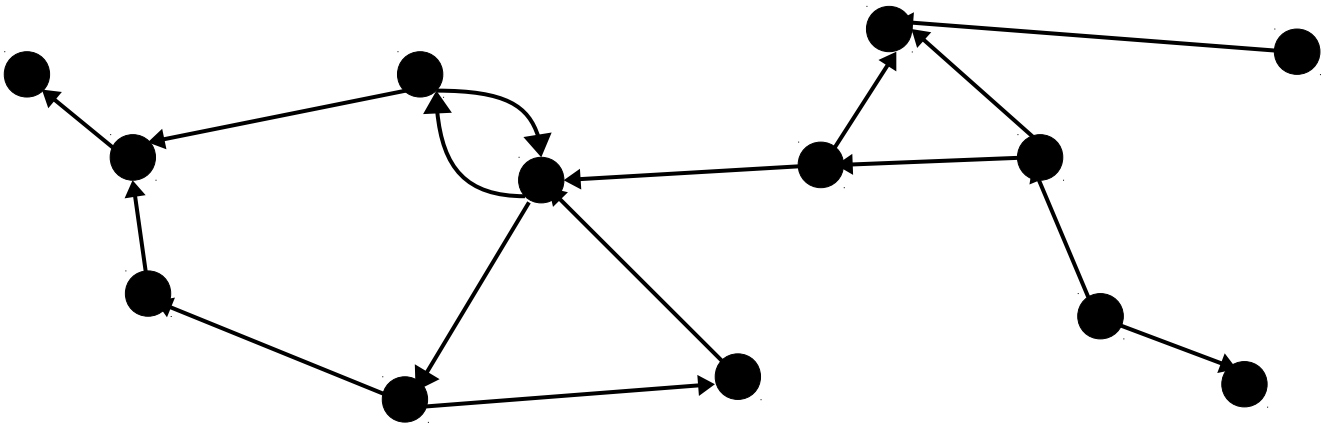
1. Κατά τη διάρκεια μιας χρονικής μονάδας, μπορούμε να εξερευνήσουμε όσους περισσότερους κόμβους μας επιτρέπει η εκάστοτε κατεύθυνση των ακμών.
2. Για την εξερεύνηση των γράφων υποθέτουμε (χωρίς βλάβη της γενικότητας) ότι, θα ξεκινήσει από τυχαία χρονική στιγμή, όπου το υπολοιπό της με το 2 είναι ίδιο με το $Tmod\,2$.

Για παράδειγμα οι 2 γράφοι θα μπορούσε να ήταν οι πρακάτω:

- Τυχαία άρτια χρονική στιγμή:



- Τυχαία περιττή χρονική στιγμή:



Ο αλγόριθμος στηρίζεται πάνω σε δύο παράλληλα BFS, με μία ουρά το καθένα για την εξερεύνηση των κόμβων. Ξεκινάει bfs στον πρώτο γράφο, με μια fifo ουρά. Κράταμε 2 πίνακες visited, ένα για το κάθε γράφημα σημειώνοντας εκεί την κατάσταση των κόμβων. Η αρχική τους κατάσταση είναι Α(ανεξερεύνητος), όταν τοποθετούνται στην ουρά γίνεται Ε(εξερευνημένος), καθώς προσθέτουμε τους γειτουνικούς τους κόμβους στην ουρά.Πιο αναλυτικά θέτουμε τους κόμβους ως εξερευνημένους κατά την εισοδό τους στην ουρά για την αποφυγή επικαλύψεων(πολλαπλές εισαγωγές του ίδιου κόμβου στην ουρά).

Τα βήματα του αλγορίθμου έχουν ως εξής:

1. Αρχικοποιούμε ένα μετρητή “timer”, όπου μετρά τις εναλλαγές των κατεθύνσεων(χρονικές μονάδες) και τοποθέτουμε τον αρχικό κόμβο και στις 2 ουρές και τον σημειώνουμε ως Ε στους αντίστοιχους πίνακες.
2. Εξάγουμε το πρώτο στοιχείο της 1ης ουράς(άρτιας ή περιττής με βάση την 2η παραδοχή).
3. Ελέγχουμε αν είναι ο τελικός ή οχι.
   a) Αν είναι, τότε σταματάει ο αλγόριθμος με αποθηκευμένο τον αριθμό των εναλλαγών στον timer και πηγαίνει στο βήμα 6.
   b) Αν δεν είναι τελικός:
      - Ελέγχουμε με O(1), αν η λίστα γειτνίασης είναι άδεια ή οχι.
        - Αν δεν είναι άδεια, τότε τοποθετούμε τους γείτονες στην ουρά, η οποία πληρεί τις προϋποθέσεις(κατάσταση Α στον αντίστοιχο πίνακα). Ενδεχομένως, να τοποθετηθούν και στις 2 ουρές(δυνατότητα αναμονής μέχρις ότου αλλάξει η κυκλοφορία). Τέλος, ενημερώνουμε τους πίνακες κάθε ουράς, στην οποία εισήχθησαν νεόι κόμβοι.
        - Αν είναι άδεια, συνεχίζουμε.
4. Αν η εν ενεργεία ουρά αδειάσει(έχουμε φτάσει σε αδιέξοδο και πρέπει να περιμένουμε για αλλαγή κατεύθυνσης), τότε αυξάνουμε τον μετρητή και μεταβαίνουμε στον άλλο γράφο.
5. Επανάλαβε τα βήματα 2-4.
6. Τερματισμός αλγορίθμου και εύρεση της τελευταίας στιγμής, που μπορούμε να ξεκινήσουμε και να φτάσουμε έγκαιρα στο κέντρο της πόλης.
   a) Αν ο timer είναι άρτιος τότε μπορούμε να ξεκινήσουμε χρονική στιγμή, της οποίας η διαίρεση με το 2 αφήνει διαφορετικό υπόλοιπο με αυτό της αφετηρίας και να χρειαστούμε **timer-1** χρονικές μονάδες.
   b) Αν είναι περιττός, πρέπει να ξεκινήσουμε χρονική στιγμή ,της οποίας η διαίρεση με το 2 αφήνει ίδιο υπόλοιπο με αυτό της αφετηρίας και να κινηθούμε για χρόνο ίσο με το **timer**.
      - Η εύρεση για την χρονική στιγμή μετά την επιλογή του υπολοίπου(άρτια ή περιττή), θα γίνει ως εξής:

- Ελέγχουμε αν το αποτέλεσμα της πράξης $(T-\Delta t_{min}) mod 2$, είναι συμβατό με αυτό στο οποίο καταληξαμε, αλλιώς διαλέγουμε την στιγμή $T-\Delta t_{min}-1$

c) Αν είναι μηδέν μπορούμε να κινηθουμε την Τ στιγμή.

Πολυπλοκότητα:

Ο αλγόριθμος πραγματοποιεί 2 παράλληλα BFS, τα οποία έχουν χρονική πολυπλοκότητα **O(n+m).** Στην χειρότερη περίπτωση, θα πρέπει να εξερευνήσουμε τους κόμβους δυό φορές, οπότε η πολυπλοκότητα είναι **O(n+m).**

Ορθότητα:

Την ορθότητα της άσκησης μας την εγγυάται η υπόθεση ότι, η διάσχιση όσων κόμβων μας επιτρέπεται διαρκεί 1 χρονική μονάδα σε συνδυασμό με την φύση του BFS. Σε μια τυχαία χρονική στιγμή, η διάσχιση κατά πλάτος μας εγγυάται την έυρεση του ελάχιστου χρόνου, καθώς τα βήματα στους επόμενους γείτονες μέχρις ότου να βρεθούμε σε αδιέξοδο πραγματοποιούνται στην ίδια χρονική στιγμή. Πράγμα το οποιο σημαίνει ότι, αν κάποια διαδρομή οδηγήσει στον τελικό κόμβο θα είναι μία από τις βέλτιστες(ελάχιστος χρόνος) χάρη στην παραπάνω αναλλοίωτη.

Η ενδεχόμενη προσθήκη κόμβων, οι οποίοι συναντήθηκαν κατά την διάσχιση του ενός BFS, στην ουρά του άλλου, μεταφράζεται ως την αναμονή σε τοποθεσίες, που έχουμε φθάσει μέχρις ότου να αλλάξει η κυκλοφορία. Με αυτό τον τρόπο, αποκτούμε την δυνατότα κίνησης από κάθε κόμβο και προς τις 2 κατευθύνσεις.

Εφόσον το BFS βρίσκει όλους τους γείτονες(γειτονικές τοποθεσίες) ενός κόμβου **u**(τοποθεσίας), ισχυριζόμαστε ότι, μια δεύτερη εξερεύνηση του ίδιου κόμβου **u**, θα μας οδηγούσε στο ίδιο αποτέλεσμα. Στηριζόμενοι στο τελευταίο, χρησιμοποιούμε 1 πίνακα κατάστασης για κάθε BFS, για την αποφυγή επικαλύψεων. Συνεπώς, λόγω των παραπάνω, καταλήγουμε σε βέλτιστη δυνατή λύση.

Η επιλογή της χρονικής στιγμής βασίζεται στο ότι, αν χρειαστούν άρτιος αριθμός εναλλαγών για να φτάσουμε στο κέντρο, τότε μπορεί να επιτυχθεί με τον αμέσως προηγούμενο περιττό αριθμό κι αφετηρία την επόμενη χρονική στιγμή. Αν ο προηγούμενος είναι αρνητικός μπορεί να φτάσει σε 0 χρονικές μονάδες εξαρχής.

Λύση με Διαφορετικές με παραδοχές:

Για την επίλυση του προβλήματος κάνουμε τις εξής υπόθεση:

1. Η μετακίνηση από έναν κόμβο σε έναν άλλον διαρκεί 1 χρονική μονάδα.

Θα κινηθούμε στο ίδιο σχεδιάγραμμα με την προηγούμενη λύση. Θα χρησιμοποιήσουμε 2 γράφους με αντίθετες κατευθύνσεις και 2 παράλληλα BFS, όπου για το καθένα θα έχουμε 1 πίνακα καταστάσεων και μια ουρά. Επίσης, θα ορίσουμε μια μεταβλητή **timer**, η οποία θα μετράει τα βήματα/χρονικές μονάδες για την άφιξη μας στον τελικό προορισμό. Δίνονται παρακάτω τα βήματα του αλγορίθμου, με αφετηρία ένα τυχαίο γράφημα από τα δύο:

1. Εισάγουμε και στις 2 ουρές τον κόμβο **s** .
2. Εξερευνούμε(εξάγουμε από την ουρά και προσθέτουμε τους γείτονες) τόσους κόμβους όσος είναι το αρχικό μέγεθος της σε κάθε επανάληψη. Με αυτό τον τρόπο, έχουμε την

δυνατότητα να κινηθούμε από όλους τους κόμβους, στους οποίους έχουμε φτάσει μέχρι εκείνη την στιγμή, σε απόσταση 1 ακμής.
3. Όταν εξαχθούν τόσοι κόμβοι όσος το αρχικό μέγεθος της ουράς, αυξάνουμε τον **timer** και αλλάζουμε ουρά και γράφο.
4. Ο αλγόριθμος τερματίζει όταν φτάσουμε στον τελικό κόμβο.

Για την απόφαση της τελευταίας στιγμής, εκτελούμε τον παραπάνω αλγόριθμο 2 φορές μια για κάθε αφετηρία άρτια ή περιττή και κρατάμε την μικρότερη αλλά και την πληροφορία αν είναι άρτια ή περιττή. Βρίσκουμε σε σταθερό χρόνο την χρονική στιγμή t_k , που μπορούμε να ξεκινήσουμε και να φτάσουμε στην ώρα μας. Για την t_k ισχύει:

$$t_k \leq T(1)$$
$$t_k + timer \leq T(2)$$
$$t_k \, mod \, 2 = 1 \vee t_k \, mod \, 2 = 0 (3)$$

<u>Ορθότητα:</u>
Ο περιορισμος συγκεκριμένου αιρθμού εξερεύνησης κόμβων, πρακτικά μεταφράζονται στο μοναδιαίο χρονικά βήμα, που δύναται να κάνουμε κάθε φορά. Ελέγχοντας σε μια γειτονία (BFS), κινούμαστε σε όλους του γετονικούς κόμβους με κόστος 1. Το αρχικό μέγεθος της ουράς μεταφράζεται σαν τον αριθμο των τοποθεσιών, που έχουμε φτάσει μέχρι στιγμής και μόνο αυτοί πρέπει να εξερευνθούν για να γίνει το επόμενο βήμα.

Η χρήση 2 ουρών μας δίνει την δυνατότητα της αναμονής σε έναν κόμβο αλλά και την κίνηση προς την αντίθετη κατεύθυνση.Κινούμενοι προς όλες τις επιτρεπτές κατευθύνσεις, δίνοντας τη δυνατότητα αναμονής κι αποφεύγοντας κύκλους λόγω του πίνακα καταστάσεων, καταλήγουμε πάντα σε ελάχιστο χρόνο άφιξης.

<u>Πολυπλοκότητα:</u>

Απαιτείται γραμμικός χρονος εκτέλεσης $\boldsymbol{O(n+m)}$.

<u>Άσκηση 2:</u>

Για το παρακάτω πρόβλημα γίνονται οι εξής παραδοχές:

1. Όλα τα σωματίδια κινούνται ταυτόχρονα
2. Μπορούν σε μία νύχτα να μετακινηθούν κατά ένα κόμβο το πολύ
3. Γνωρίζουμε μόνο τις αρχικές θέσεις και όχι πότε αυτά θα κινηθούν
4. Εφόσον δεν ελέγχουμε τον χρόνο κινησής του, αρκεί να υπολογίσουμε το μονοπάτι, που πρέπει να ακολουθήσει το κάθε σωματίδιο για να φτάσει στο βέλτιστο σημείο συνάντησης. Υποθέτουμε ότι, θα κινηθούν τις κατάλληλες χρονικές στιγμές για να φτάσουν ταυτόχρονα.

Σύμφωνα με τις παραπάνω υποθέσεις, καταλήξαμε στον ακόλουθο αλγόριθμο.

Το πρόβλημα μπορεί να προσεγγιστεί με τον αλγόριθμο BFS. Μπορεί κανείς να σκεφτεί ότι, οι ακμές (οι μεταβάσεις των σωματιδίων) έχουν κόστος 1 μέρα(μεσάνυχτα) ή καθόλου, αν αυτά αποφασίσουν να μείνουν στην ίδια θέση. Οφείλουμε να σημειώσουμε ότι, αν έστω ένα σωματίδιο κινηθεί κάποια τυχαία νύχτα, τότε έχει νοήμα να συνυπολογισουμε την ημέρα ως παραπάνω καθυστέρηση διαφορετικά όχι.

Τα βήματα του αλγορίθμου είναι τα εξής:
1. Διαβάζουμε την είσοδο και αποθηκέυουμε τον γράφο σε μια λίστα γειτνίασης.

2. Χρησιμοποιούμε δυό πίνακες. Έναν πίνακα D[n] για την αποθήκευση των ελάχιστων μονοπατιών και έναν για την αποθήκευση των γονέων p[n], για το backtracking.
3. Για κάθε σωματίδιο στον γράφο ως αφετηρία, εκτελούμε Διάσχιση κατά Πλάτος και βρίσκουμε τις αποστάσεις του εκάστοτε σωματιδίου προς όλους τους κόμβους.
4. Εκτελούμε μία ακόμη φορά BFS στον γράφο, όπου για κάθε κόμβο που συναντάμε, βρίσκουμε από τους **k** διαφορετικούς πίνακες D[] την μεγαλύτερη απόσταση, η οποία δηλώνει το μεγαλύτερο χρονικό διάστημα ημερών, που μπορεί να κάνει άποιο σωματίδιο για να βρεθεί στον εκάστοτε κόμβο. Αποθηκεύουμε την τιμή αυτή καθώς και τον αντίστοιχο source vertice σε δυό μεταβλητές.
5. Σε κάθε βήμα του τελευταίου BFS, κρατάμε την μικρότερη-μεγαλύτερη τιμή των αποστάσεων μεταξύ του ***i-οστού*** και του ***i-1-οστου*** κόμβου και ανανεώνουμε τον κόμβο συνάντησης.

Στο τερματισμό του παραπάνω αλγορίθμου, θα έχουμε ως αποτέλεσμα k- πίνακες γονέων και τον κόμβο που θα γίνει η συνάντηση. Το backtracking γίνεται σε **O(n)** για κάθε σωματίδιο.

Πολυπλοκότητα:

Χρονική:
Η πολυπλοκότητα του αλγορίθμου περιλαμβάνει **(k+1) BFS** και στο τελευταίο BFS μια γραμμικότητα **O(k)** για κάθε κόμβο για την εύρεση του μεγαλυτέρου. Επίσης, **O(k * n)** για το backtraking. Οπότε συνολικά θα έχουμε:

$$\boldsymbol{O\left(k\left(n+m\right)+nk+m+kn\right)=O\left(n+m\right)}\text{, } \textit{καθώς δίνεται } n \gg k$$

Χωρική:
Χρειαζόμαστε μνήμη $\boldsymbol{O\left(n+m+kn\right)=O\left(n+m\right)}$, *όσο το μέγεθος του γράφου καθώς* $n \gg k$

Ορθότητα:

Την ορθότητα μας την εγγυάται το γεγονός ότι, λαμβάνουμε σαν ελάχιστο κόστος το μικρότερο δυνατό από τα μεγαλύτερα, που προκύπτουν για κάθε δέντρο συντομότερων μονοπατιών. Εφόσον δεν γνωρίζουμε τις χρονικές στιγμές που κινούνται, αρκεί να υπολογίσουμε το ελάχιστο πλήθος ημερών, που χρειάζονται για τη συνάντηση με βάση την αρχική τους θέση, όπου τις μέρες αντιστοιχούμε τις ακμές του γραφήματος.

Άσκηση 3:

Ο αλγόριθμος την εύρεση του ελάχιστου κόστους ενός κατευθυνόμενου γράφου G(V, E, c), χωρίζεται σε δύο μέρη. Αρχικά, εφόσον έχουμε κατευθυνόμενο γράφο υπάρχει και η περίπτωση ύπαρξης κύκλων. Μπορεί να παρατηρήσει κανείς ότι, ο GCD ενός κύκλου είναι ίδιος όσες φορές και να τον διανύσουμε. Επομένως, θα ισχυριζόταν κανείς ότι, η σύμπτηξη των στοιχείων, που τον απαρτίζουν, σε έναν κόμβο με τιμή τον αντίστοιχο GCD, δεν θα επηρέαζε τη λύση αρνητικώς.

Επίσης, εξ' ορισμού ο περίπατος είναι μια ακολουθία (ίδιων ή διαφορετικών)ακμών και (ίδιων ή διαφορετικών)κόμβων. Λόγω της μετρικής κόστους(GCD), που ακολουθούμε ο υπολογισμός του ελάχιστου κόστους ανάγεται στην έυρεση του ελάχιστου κόστους ενός μονοπατιού (μία φορα κάθε

ακμή και κόμβος),το οποίο θα μπορεί να συμπεριλαμβάνεται παραπάνω από μία φορά στον περίπατο.

Επικαλούμενοι την ιδιότητα της μονοτονίας για την συνάρτηση κόστους το GCD, έχουμε:

Έστω 2 σύνολα κόμβων $A, A' \subseteq V$ *με την ιδιότητα* $A' \supseteq A$, τότε ισχύει:

$$GCD(A') \leq GCD(A) \quad (1)$$

Πορίσμα:

1. Η σχέση (1) μας αποδεικνυέι ότι, αν κάποια από τα στοιχεία του κύκλου ήταν μέρος της βέλτιστης λύσης έστω C^* , το γεγονός ότι, συμπτίσσουμε όλα τα στοιχεία του κύκλου σε ένα και προσθέτουμε στο ήδη υπάρχον σύνολο C^* παραπάνω κόμβους μας δίνει εξίσου βέλτιστη λύση.

1ο Μέρος - Εξάλειψη ΙΣΣ:
Ορμόμενοι το παραπάνω πόρισμα, για την εξάλειψη των κύκλων θα χρησιμοποιήσουμε τον αλγόριθμο Kusaraju, ο οποίος θα εντοπίσει τις ισχυρές συνεκικές συνιστώσες σε γραμμικό χρόνο **O(n+m)**. Έχοντας βρει τις ΙΣΣ, μπορούμε να τις συμπτίξουμε σε ένα κόμβο με κόστος το GCD των στοιχείων τους.

Δημιουργία Νέου Γράφου:

Μπορούμε με ένα πέρασμα σε όλες τις ακμές του αρχικού γράφου σε **O(m)** και με τη χρήση Union-Find(with path-compression, rank matrix), όπου ο χρόνος του find() εκυλίζεται σε σταθερό, να κατασκευάσουμε το νέο γράφο με συμπτιγμένες τις ΙΣΣ και να εξαλλείψουμε πλεονάζουσες ακμές, οι οποίες εισέρχονται ή εξέρχονται από μια ΙΣΣ.

2ο Μέρος Αλγορίθμου:

Το 2ο μέρος, έγκειται σε μια Διάσχιση κατά Βάθος, όπου η πληροφορία μεταδίδεται από τους μακρινούς γείτονες, στον εκάστοτε κόμβο, αφού τελειώσει η αναδρομή τους και επιστρέψουν στον σ' αυτόν. Παρατίθεται κώδικας για την βαθύτερη κατανόηση:

```
*initialize the global variable for the minimum cost*
global_cost = infinity
dfs(graph):
    for u in V:
        dfs_util(graph, u);

dfs_util(graph, u):
    visited[u] = 1;
*for every unvisited neighbor, visit it*
    for v in V(u):
        if (visited[v] == 0):
            dfs_util(v);
    min_cost = graph[u].cost, g = 0;
*when the exlporation of "u" has ended, find the minimum GCD of every children and update the cost of u in the graph*
*if u is leaf then it holds its cost*
    for v in V(u):
        g = gcd (graph[u].cost,graph [v].cost);
        if (min_cost > g):
            min_cost = g;
    graph[u].cost = min_cost;
*if the cost is less than the global, update it*
    if (global_cost > min_cost):
        global_cost = min_cost;
```

Ορθότητα:

Ο αλγόριθμος στηρίζεται (α) στην μονοτονία του GCD, (β) στην φύση του DFS.

(α) Μονοτονία του GCD:

Λόγω της μονοτονίας του GCD, είναι απαραίτητο να διερευνήσουμε όλο το γράφο, για πιθανούς περιπάτους, που θα μειώσουν το περισσότερο δυνατό το κόστος. Εν ολίγοις, επιθυμούμε να αυξήσουμε πληθικά το σύνολο C^*, το οποίο θα μας δώσει τη βέλτιστη λύση, το οποίο θα εξασφαλίσει την ορθότητα του αλγορίθμου.

(β) Φύση DFS:

Έστω d[u], f[u] οι χρόνοι πρώτης επίσκεψης και αναχώρησης αντίστοιχα για έναν κόμβο u. Για το DFS ισχύει ότι, αν $d[u]<d[v]\Rightarrow f[u]>f[v](1)$. Η σχέση (1) μας εξασφαλίζει ότι, η πληροφορία(minimum GCD), ταξιδεύει από το μέλλον(παιδιά) στο παρόν(γονέας), καθώς διασχίζουμε σε βάθος το γράφο υπολογίζοντας το GCD των παιδιών και επιστρέφοντας αυτή την τιμή στο γονέα, όταν τελείωσει η αναδρομή τους. Ο πατέρας, λοιπόν, μπορεί πλεόν να αποφασίσει ποια πληροφορία θα κρατήσει(GCD διαφορετικών υποδέντρων, αφού πλέον δεν υπάρχουν κύκλοι) για να ανανεώσει την τιμή του.

Η ορθότητα του αλγορίθμου εξάγεται ως εξής:

Έστω ότι, βρισκόμαστε στον κόμβο u, ο οποίος πρέπει να επιλέξει από k(όσοι είναι οι γείτονες κόμβοι-ακμές) διαφορετικές τιμές GCD $(u,v\in V(u))$.

Από αυτές θα επιλεξει την μικρότερη, έστω g_{min} για κάποιο κόμβο v_k. Άρα θα ισχύει:

$$g(u,v_k)\leq g(u,v_i)\,v_i\in V(u)(2).$$

Ο κόμβος u θα μεταφέρει αυτή την πληροφορία στον πατέρα του, όπου θα προστεθουν κι άλλα στοιχεία στην λύση και η επιλογή θα γίνει μεταξύ των GCD του πατέρα κι όλων των u-παιδιών. Το γεγονός ότι, δεν έχουμε κύκλους μας δίνει τη δυνατότητα να ισχυριστούμε δεν παραλείπουμε πιθανή λύση και ότι, για κάθε υποπρόβλημα έχουμε βρει τη βέλτιστη λύση.

Πολυπλοκότητα:

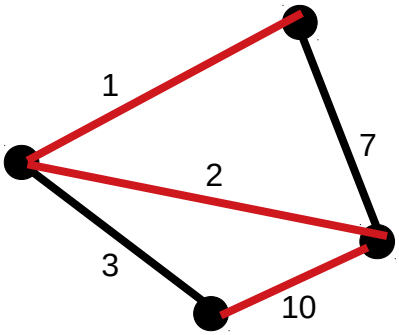
Η χρονικη πολυπλοκότητα είναι γραμμική για το πρώτο μέρος, όπως περιγράφηκε. Στο δεύτερο μέρος το DFS διαρκεί **O(n+m)**, όπου για κάθε κόμβο για την εύρεση του minimum, θέλουμε $\boldsymbol{O(k_u)}$, όπου k_u ο αριθμός των ακμών, που εξέρχονται από τον κόμβο u.

Ισχύει $\sum_{u\in V} k_u = m$, m αριθμός ακμών του γράφου

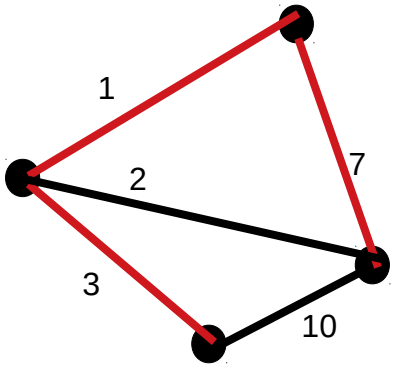
Άρα συνολικά χρειαζόμαστε **O(2(n+m) + m + n+2m) = O(n+m)**.

Άσκηση 4:

α)
Με λίγη σκέψη εύκολα καταλήγει κανείς ότι, το άπληστο κριτήριο δεν μας οδηγεί στο σωστο αποτέλεσμα. Για παράδειγμα, στην Εικόνα 1 το άπληστο κριτήριο(προσαρμοσμένο ώστε να χουμε τον επιθυμητο βαθμό του κόμβου **s** στο MST) με k = 2 δίνει συνολικό κόστος 13, ενώ υπάρχει συνδετικό δέντρο με κόστος 11 Εικόνα 2.



Εικόνα 1



Εικόνα 2

β)

Για την εύρεση ενός MST with constraints, θα εργαστούμε ως εξής:

Έστω ο κόμβος **s ,** με deg(s) στο αρχικό γράφημα. Αρχικά, στηριζόμαστε στο γεγόνος ότι, αν αφαιρέσουμε τον κόμβο **s** τις ακμές e_i , οι οποίες συνδέονται στον κόμβο **s**, θα δημιουργηθεί τουλάχιστον μία συνεκτική συνιστώσα. Έστω **β**, ο αριθμός των συνεκτικών συνιστωσών, που δημιουργούνται μετά την διαγραφή των ακμών e_i και του κόμβου **s,** για τον οποίο ισχύει $\beta \geq deg(s)$.

<u>1ο Μέρος Αλγορίθμου:</u>
Με ένα DFS/BFS σε χρόνο **O(n+m)**, μπορούμε να αποφανθούμε τον αριθμό **β.** Οφείλουμε να σημειώσουμε στο σημείο αυτό ότι, θα πρέπει να χρησιμοποιηθεί η δομή Union-Find (with path compression, rank matirx), η οποία θα αξιοποιηθεί στο 2ο μέρους το αλγορίθμου. Για κάθε συνεκτική συνιστώσα, που προκύπτει μπορούμε με έναν αλγόριθμο εύρεσης των MST π.χ. Prim, Kruskal, να βρούμε τα MSTs.

Έστω ότι, χρησιμοποιούμε τον αλγόριθμο του Prim με μια απλή υλοποίηση ουράς FIFO. Τότε, η συνολική πολυπλοκότητα για τις **β** εκτελέσεις του αλγορίθμου θα κοστίσει:

$$\boldsymbol{O\left(n_1^2+n_2^2+n_3^2+\ldots n_\beta^2\right) \leq O\left(n^2\right)(1), \text{όπου } n=n_1+n_2+n_3+\ldots n_\beta}$$

Η (1) ισχύει καθώς έχουμε,

$$n>0, n_1, n_2 \ldots n_\beta \geq 0$$

$$n=n_1+n_2+n_3+\ldots n_\beta \rightarrow n^2=\left(n_1+n_2+n_3+\ldots n_\beta\right)^2=\sum_{i=1}^{\beta} n_i^2+2 n_i\left(n_{(i+1)} \ldots n_\beta\right)+\left(n_{(i+1)}+\ldots n_k\right)^2 (2)$$

$$n^2 \geq n_1^2+n_2^2 \ldots+n_\beta^2 (2) \Leftrightarrow \sum_{i=1}^{\beta} 2 n_i\left(n_{(i+1)} \ldots n_\beta\right)+\left(n_{(i+1)}+\ldots n_k\right)^2 \geq 0, \text{που ισχύει}$$

2ο Μέρος Αλγορίθμου:

Έστω k, ο επιθυμητός βαθμός του κόμβου **s** στο τελικό MST, για τον οποίο ισχύει $k \geq deg(s)$.
Στη συνέχεια έχουμε τις εξής περιπτώσεις:

1.Αν β > k:
Τότε είναι φανερό ότι, δεν δύναται να υπάρξει *δέντρο*, καθώς θα έχουμε ξεχωριστές συνεκτικές συνιστώσες(*δάσος).*

2. Αν β = k:

α.Αν k = deg(s):
Τότε πρέπεινα προστεθούν όλες οι ακμές.
β. Αν k<deg(s):
Τότε καλούμαστε να επιλέξουμε τις ελαχίστου βάρους ακμές e_i in E(s) για τη σύνδεση της κάθε συνεκτικής συνιστώσας. Το τελευταίο μπορεί να πραγματοποιηθεί με ένα πέρασμα των ακμών που προσπίπτουν στο **s,** σε χρόνο O(deg(s)) = O(n), αφού deg(s) <= n-1 σε συνδυασμό με τη δομή Union-Find(που έχει δημιουργηθεί στο 1ο μέρος), κρατώντας την ελαχίστου βάρους για κάθε συνεκτική συνιστώσα.

3. Αν β < k:

Σε αυτή την περίπτωση έχουμε πλεονασμό επιλογών μεταξύ των ακμών e_i in E(s). Αρχικά, προσθέτουμε την ελάφρύτερη ακμή του **s**, για να συνδεθεί στο γράφημα. Τώρα έχουμε το γνήσιο MST, έστω **T*,** το οποίο δεν πληρεί τον περιορισμό για τον βαθμό του **s**.

Σε δεύτερη φάση, για κάθε ακμή e_i in E(s), εκτελούμε την εξής διαδικασία:

α. Επειδή το **T*,** είναι συνεκτικό δέντρο, γνωρίζουμε ότι, η προσθήκη μιας ακμής e_i , θα δημιουργήσει κύκλο C, ο οποίος θα περιέχει σίγουρα τον κόμβο **s** και τον κόμβο $\boldsymbol{v_i}$ για τον οποίο ισχύει $e_i=\{s,v_i\}$. Με μια διάσχιση DFS, βρίσκουμε την μεγίστου βάρους ακμή για το μονοπάτι $\{s,v_i\}$, έστω e_j του σε χρόνο **O(n+m)**, η οποία θα είναι η μεγίστου βάρους ακμή στον κύκλο C, που δημιουργείται αν προσθέσουμε την ακμή e_i **.**

β. Υπολογίζουμε τη διαφορά, $diff_j=w(e_j)-w(e_i)$, η οποία ορίζει το κέρδος της ανταλλαγής των ακμών e_i,e_j στο δέντρο.

γ. Δημιουργώ τούπλες $(diff_j,e_j,v_i)$, και τις τοποθετώ σε μια priority qeue ,σε φθίνουσα σειρά του diff.

Το παραπάνω βήμα, διαρκεί $\boldsymbol{O(n(n+m))=O(n^2)}$, αφού έχουμε δέντρο **m = n-1**.

Έστω, $deg_T(s)$, ο βαθμός του **s** , στο επιθυμητό MST with constraints **T**. Όσο $deg_T(s)<k$ και $diff_j>0$ εκτελούμε τα παρακάτω:

1. Για το πρώτο στοιχείο της PQ, αν η ακμή e_j έχει αφαιρεθεί από το δέντρο, βρίσκω την μεγίστου βάρους ακμή στο μέρχι τότε γράφημα(όπως στο βήμα α. σε **O(n)**) και ενημερώνω την προτεραιοτητάς της στην ουρά.
2. Αν δεν έχει αφαιρεθεί, αντικαθιστώ, την ακμή e_j με την $e_i=\{s,v_i\}$ και διαγράφω την τούπλα.

Η ανταλλαγή των ακμών χρειάζεται, λοιπόν, $O(kn^2)$ καθώς η διαγραφή μιας τούπλας-ακμής μπορεί να γεννήσει **n** γραμμικότητες, αν όλες οι ακμές e_i είχαν ως ανταλλαξιμή τους την ίδια ακμή. Συνολικά ο αλγόριθμος $O(kn^2)$.

Ορθότητα:

Έστω T_{opt} , το βέλτιστο δέντρο, στο οποίο δεν ανταλλάσουμε τις βαρύτερες ακμές ενός κύκλου και $e \in T_{opt}$ και $e' \in T$, ακμές στις οποίες διαφέρουν τα δυο δέντρα. Εφόσον ισχύει $w(e') \le w(e)$, ανταλλάσσοντας αυτές τις 2 ακμές προκύπτει $T_{opt}', w(T_{opt}') \le w(T_{opt})$. Συνεχίζοντας την ανταλλαγή μεταξύ όλων των ακμών που διαφέρουν θα έχουμε $T_{opt}'' = T, w(T) \le w(T_{opt})$. Επομένως, ο αλγόριθμος καταλήγει πάντα σε βέλτιστη λύση.

Η μη ύπαρξη κύκλου είναι προφανής, καθώς αφαιρούμε πάντα μια ακμή του κύκλου, που δημιουργείται.

Άσκηση 5:

α)
Ορθότητα του Αλγορίθμου Boruvka:

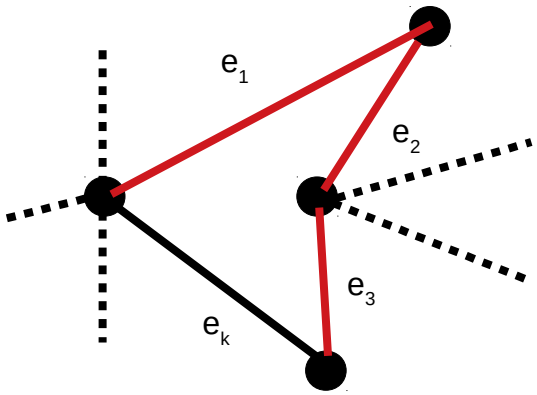
Αρκεί να δείξουμε ότι, καταλήγουμε πάντα σε συνεκτίκό ακυκλικό γράφημα με ελάχιστο συνολικό κόστος ακμών.

Μη Ύπαρξη Κύκλων:

Έστω ότι, βρισκόμαστε στην k-οστη επανάληψη του αλγορίθμου και έχουμε επιλέξει μια ακολουθία ακμών $e_1, e_2, \dots e_k$, οι οποίες κλείνουν κύκλο. Έστω ότι η μικρότερη είναι η e_1 , τότε θα είχαμε μια διάταξη όπως την ακόλουθη:

$$w(e_1) < w(e_2) < w(e_3) \dots < w(e_k) \quad (1)$$

Για να επιλεχθεί κι η e_k θα πρεέπει να ισχύει $w(e_k) < w(e_1) \quad (2)$, **Άτοπο** από (1).
Για την οπτικοποίηση της απόδειξης παρατίθεται το παρακάτω σχήμα:



Εικόνα 1

Αν θεωρήσουμε την e_1 ως μικρότερη, τότε για να επιλεχθεί η e_k θα πρέπει να ισχύει η (2), το οποίο είναι Άτοπο, διότι θα έιχε επιλεχθεί εξαρχής.

Επομένως, σε κάθε επανάληψη ο αλγόριθμος συνδέει συνεκικές συνιστώσες, αποφεύγοντας τη δημιουργία κύκλων. Τελικά, προκύπτει **συνδετικό δέντρο**.

Απόδειξη Ελαχίστου Κόστους:

Έστω ότι δεν καταλήγουμε πάντα σε MST. Θα υπάρχει λοιπόν ένα MST **T***, για το οποίο θα έχουμε το ελάχιστο κόστος. Έστω **T** το δέντρο που καταλήγει ο αλγόριθμος.

$\forall e \in T^*, \exists e' \in T$, τέτοια ώστε να γεφυρώσει την τομή του γραφήματος $T^* \setminus e$ (απόδειξη στην Άσκηση 6) $(S, S \setminus V)$. Επομένως, αν ανταλλάξουμε τις e, e' εφόσον ισχύει $w(e') < w(e)$, θα καταλήξουμε σε συνδετικό δέντρο αυστηρά μικρότερου βάρους. Άτοπο καθώς έχουμε υποθέσει ότι, το **T*** είναι MST.

Επομένως, ο αλγόριθμος Boruvka βρίσκει πάντα ένα MST .
β)
Βήματα Αλγορίθμου:

1. Ξεκινάμε από δάσος με κόμβους.
2. Όσο υπάρχει δάσος, βρες την ελάχιστου βάρους ακμή για κάθε συνιστώσα και προσθεσέ τη στο MST.
3. Για κάθε ακμή που ενώνει 2 δέντρα, ενταξέ τη στο MST και σύμπτιξε τις συνιστώσες.
4. Τερματισμός του αλγορίθμου, όταν έχουμε μόνο ένα δέντρο το Minimum Spanning Tree.

Το βήμα 2 γίνεται σε **O(m),** αν χρησιμοποιήσουμε δομή Union Find(with path-compression, rank matrix). Στην αρχή η δομή περιέχει disjoint κόμβους, οι οποίοι συμπτίσονται σε συνεκτικές συνίστωσες(ελάχιστα συνδετικά δέντρα). Σε κάθε βήμα για κάθε ακμή του γράφου γίνονται 2 find για την έυρεση των γονέων καθε κόμβου. Για τις ακμές, με διαφορετικούς πατεράδες κρατάμε τις ελαχίστου βάρους. Εν τέλει κάνουμε “k” Unions, όπου $k = \frac{n}{2^i}$, i-οστη επανάληψη αλγορίθμου.
Εν τέλει, ο αλγόριθμος τερματίζει μετά από **O(log (n))** βήματα καθλως σε κάθε βήμα οι σνεκτικές συνιστώσες μειώνονται στο μισό.

γ)
Η χρήση του Hybrid αλγόριθμου DJP-Boruvka γίνεται ως εξής. Εκτελούμε κάποια βήματα του αλγορίθμου Boruvka, ώστε να γίνει contracion κάποιου πλήθους ακμών. Εν τέλει, θα προκύψει ένα πυκνό γράφημα, έστω **T** , στο οποίο θα εκτελέσουμε τον αλγόριθμο Prim/Jarnik’s. Το πλήθος των βημάτων, που θα εκτελέσουμε είναι loglog(n), ώστε να είναι δυνατή η βελτίωση του αλγορίθμου.

Μετά από **O(loglogn)** βήματα του Boruvka, θα έχουμε $n' \leq \frac{n}{logn}, m' \leq m(1)$ και χρόνο εκτέλεσης **O(mloglogn)**. Η υλοποίηση του αλγορίθμου Prim με Fibonacci Heap έχει πολυπλοκότητα $\boldsymbol{O(m'+n'logn') \leq O(m+n(1-\frac{loglogn}{logn})) = O(m+n) = O(m)}$, αν υποθέσουμε ότι έχουμε πυκνό γράφημα, καθώς το Fibonacci Heap, έχει καλή απόδοση στα πυκνά γραφήματα.

Η πολυπλοκότητα του Hybrid, καθορίζεται από την πολυπλοκότητα του Boruvka. Σε sparse graphs, όπου είναι **m/n = O(1)** ο αλγόριθμος έχει πολυπλοκότητα σε **O(m)** γραμμικό χρόνο καθώς ο Boruvka εκτελείται σε γραμμικό χρόνο, ενώ σε dense graphs, όπου $\boldsymbol{m/n = \Omega(\frac{n}{loglogn})}$ έχει $\boldsymbol{O(n^2)}$, εφόσον ο Boruvka εκτελείται σε $\boldsymbol{O(n^2)}$.

δ)
Η υλοποίηση του αλγορίθμου δεν αλλάζει πολύ, εκτός από το γεγονός ότι, μείωνουμε το πλήθος των ακμών που ψάχνουμε κάθε φορά. Το τελευταίο, το επιτυγχάνουμε με contraction των ακμών, τις οποίες έχουμε επιλέξει ως ελάχιστου βάρους ακμές μεταξύ των δέντρων. Μετά το contraction φροντίζουμε να εξαλλείψουμε *παράλληλες ακμές*, κρατώντας τις ελαχίστου βάρους.

Η ορθότητα διατηρείται για τον ίδιο λόγο, μόνο που τώρα δεν εχουμε περιττές ακμές.

Πολυπλοκότητα:

Το contraction υλοποιείται εύκολα με ένα πέρασμα των ακμών σε **O(m)**(με την υπόθεση οτι διατηρούμε την υλοποίηση του Union-Find) εξαλείφοντας τις εσωτερικές ακμές του κάθε δέντρου και κρατώντας τις ελαχίστου βάρους εξωτερικές ακμές για κάθε ζευγάρι δέντρων-κόμβων με πλεονάζον αριθμό ακμών(παράλληλες). Εφόσον ο $G \in C : \text{κλάση βολικών γράφων}$, τότε σε κάθε βήμα i γνωρίζουμε ότι ,λόγω της κλειστότητας, θα ισχύει $|E_i| = |V_i|$.
Επομένως, έχουμε την αναδρομική σχέση:

$$T(|V|) = T(\frac{|V|}{2}) + O(|V|) \xrightarrow{MT} T(|V|) = O(|V|)$$

Σημείωση:

- Στη γενική περίπτωση έχουμε $O(|V^2|)$, καθώς $|E| \leq |V|^2$.

Άσκηση 6:

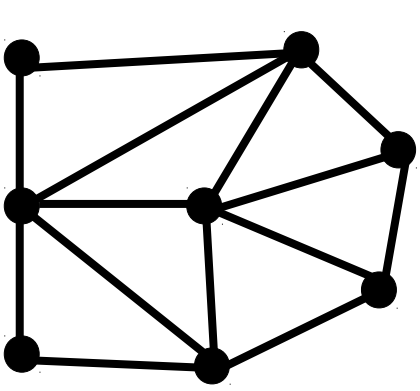
α)
Απόδειξη Ύπαρξης ακμής e':

Για κάθε $e \in T_1 \setminus T_2$, θα δείξω ότι, $\exists e' \in T_2 \setminus T_1$, τέτοια ώστε να ισχύει:

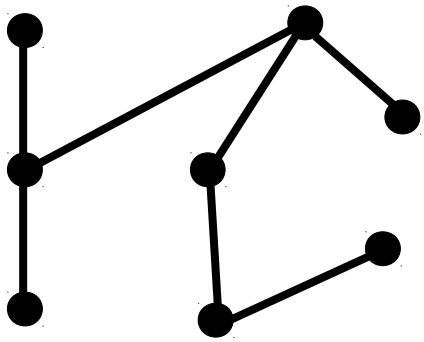
$T_1 \setminus \{e\} \cup e'$ είναι συνδετικό δέντρο.

Έστω ότι δεν ισχύει. Επομένως, $\exists e \in T_1 \setminus T_2, \forall e' \in T_2 \setminus T_1$ ισχύει ότι, το $T_1 \cup \{e'\} \setminus e$ δεν είναι συνδετικό δέντρο.

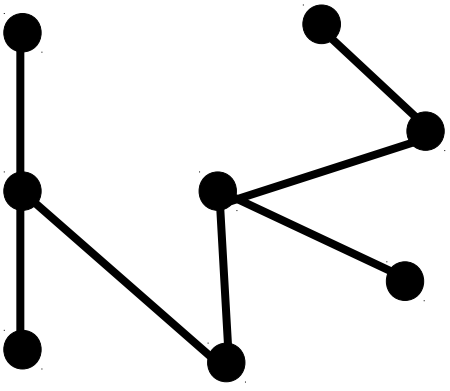
Έστω τα παρακάτω συνδετικά δέντρα T_1, T_2 για τον γράφο G.



Γράφος G

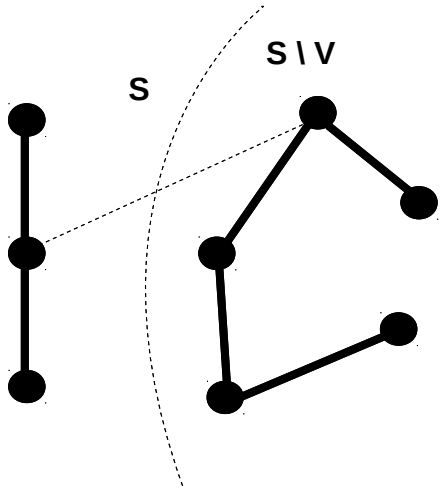


Συνδετικό Δέντρο T_1



Συνδετικό Δέντρο T_2

Το γράφημα $T_1 \setminus e$, θα δημιουργούσε μια τομή σαν αυτή στην Εικόνα 1.



Εικόνα 1

Συνεπώς, δημιουργουνται 2 συνεκτικές συνιστώσες. Με την υπόθεση ότι, $\nexists e' \in T_2 \setminus T_1$, ώστε να γεφυρώνει αυτή την τομή, καταλήγουμε σε άτοπο, καθώς γνωρίζουμε ότι, το T_2 είναι συνδετικό δέντρο, γεγονός το οποίο συνεπάγεται, την ένωση μιας οποιασδήποτε τομής στο γράφημα. Επομένως, υπάρχει ακμή e' ώστε να προκύπτει συνδετικό δέντρο.

Αφαιρώντας την ακμή e από το T_1 , προκύπτει ακριβώς μία τομή(αφού πρόκειται για συνδετικό δέντρο) και επειδή η σύνδεση των συνόλων $S, S \setminus V$ είναι ισοδύναμη με την μη δημιουργία κύκλου, είμαστε σίγουροι ότι, η προσθήκη της **e'**(της οποίας αποδείχθηκε η ύπαρξη) θα αποδόσει ένα νέο συνδετικό δέντρο $T_1 \setminus \{e\} \cup e'$.

Για την ακμή **e'**, μπορεί να ισχύει μόνο ότι, $e' \in T_2 \setminus T_1$, καθώς αν υποθέσουμε ότι, ανήκει κι στα δύο δέντρα τότε το T_1 θα είχε δύο διαφορετικές ακμές για την συνένωση της ίδιας τομής το οποίο είναι άτοπο.

Παρατήρηση:

- Μπορεί κανείς έυκολα λόγω της διατύπωσης της υπόθεσης του λήμματος, να καταλήξει στην μοναδικότητα της ακμής **e'**. Διότι, αν υποθέσουμε ότι, υπάρχουν παραπάνω από μία ακμές **e'**, ώστε αν προσθέσουμε την **e'** στο $T_1 \setminus \{e\}$ να μετασχηματιστεί σε συνδετικό δέντρο, τότε καταλήγουμε σε άτοπο, καθώς θα είχαμε παραπάνω ακμές του T_2 για την γεφύρωση της τομής $(S, S \setminus V)$.

Εύρεση της ακμής e':

Για τον αλγόριθμο εύρεσης της ακμής **e'**, θα στηριχτούμε στον παραπάνω ισχυρισμό για την τομή $(S, S \setminus V)$. Αν αφαιρέσουμε την ακμή **e** από το T_1 , τότε είμαστε βέβαιοι ότι, θα δημιουργηθεί μια τομή. Στηριζόμενοι σε αυτό, μπορούμε με μια διάσχιση DFS ή BFS στο T_1 , αφού το αφαιρέσουμε την **e**, να χωρίσουμε τους κόμβους σε εξερευνημένους και μη.

Εν συνεχεία, θα ξεκινήσουμε ένα DFS ή BFS στο T_2 από το σύνολο των εξερευνημένων του T_1 . Μόλις καταλήξουμε σε μη εξερευνημένο κόμβο του T_1, γνωρίζουμε ότι, η ακμή, η οποία ερευνούμε είναι αυτή, που έκανε αυτή την μετάβαση από το σύνολο S στο $S \setminus V$.

Η χρονική πολυπλοκότητα του αλγορίθμου έγκειται σε γραμμικό χρόνο **O(n),** αφού έχουμε δέντρο**.**

β)
Απόδειξη Συνεκτικότητας:

Αν υποθέσουμε ότι, το γράφημα Η δεν είναι συνεκτικό, τότε θα υπάρχει τουλάχιστον 1 τομή $(S, S\setminus V)$, το οποίο σημαίνει ότι, δεν υπάρχει ακολουθία διαδοχικών μετασχηματισμών, όπου έχοντας ως αφετηρία ένα οποιόδηποτε $T\in S$ να καταληξουμε σε κάποιο $T'\in S\setminus V$. Άτοπο, καθώς δείξαμε ότι, για οποιαδήποτε T_1, T_2 $\forall e\in T_1\setminus T_2$ υπάρχει ανταλλάξιμη ακμή $e'\in T_2\setminus T_1$, η οποία καθιστά το $T_1\setminus e\cup\{e'\}$ συνδετικό δέντρο, του οποίου η μορφή προσεγγίζει αυτή του T_2 . Επομένως, με διαδοχικές ανταλλαγές μπορούμε να φτάσουμε στο T_2 και τελικά υπάρχει μονοπάτι στο Η.

Απόδειξη Ισοδυναμίας με Επαγωγή:

Επαγωγική Βάση(εξ ορισμού): $|T_1\setminus T_2|=1\Leftrightarrow d_H(T_1, T_2)=1$
Επαγωγική Υπόθεση: $|T_1\setminus T_2|=k\Leftrightarrow d_H(T_1, T_2)=k$
Θα αποδείξω ότι, $|T_1\setminus T_2|=k+1\Leftrightarrow d_H(T_1, T_2)=k+1$

i) *Ευθύ* $(\Rightarrow)$:
Έστω $|T_1\setminus T_2|=k+1$, τότε από Ερώτημα (α) για κάποια $e\in T_1\setminus T_2 \exists e'\in T_2\setminus T_1$, για την οποία ισχύει ότι το $T_1'=T_1\setminus e\cup\{e'\}$ είναι συνδετικό και ότι $|T_1'\setminus T_2|=k\Rightarrow d_H(T_1', T_2)=k\ (1)$
Επίσης, $|T_1'\setminus T_1|=1\Rightarrow d_H(T_1', T_1)=1\ (2)$

Αθροίζοντας τις (1), (2) και από τριγωνική ανισότητα έχουμε $d_H(T_1, T_2)\le k+1$. Αν υποθέσουμε ότι, ισχύει το **"<"**, τότε $d_H(T_1, T_2)=k\Rightarrow|T_1\setminus T_2|=k$ **Άτοπο** από υπόθεση.

Συνεπώς, $d_H(T_1, T_2)=k+1$

ii) *Αντίστροφο* $(\Leftarrow)$:

Έστω $d_H(T_1, T_2)=k+1$, τότε από Ερώτημα (α) για κάποια $e\in T_1\setminus T_2 \exists e'\in T_2\setminus T_1$, για την οποία ισχύει ότι το $T_1'=T_1\setminus e\cup\{e'\}$ είναι συνδετικό και ότι $|T_1'\setminus T_2|=k\Rightarrow d_H(T_1', T_2)=k\ (1)$
και ισχύουν:
$d_H(T_1', T_1)=1\Rightarrow|T_1'\setminus T_1|=1\ (1)$
$d_H(T_1', T_2)=k\Rightarrow|T_1'\setminus T_2|=k\ (2)$

Από (1), (2) $|T_1\setminus T_2|=k-1\ (3)\vee|T_1\setminus T_2|=k+1\ (4)$

Αν ισχύει η (4), τότε θα έχουμε (λόγω του (i)) $|T_1\setminus T_2|=k-1$ **Άτοπο.**

Εύρεση Συντομότερου Μονοπατιού:

Για την εύρεση του ΣΜ θα στηριχτούμε στην πρόταση $|T_1\setminus T_2|=k\Leftrightarrow d_H(T_1, T_2)=k\ (1)$ και στον αλοριθμο του (α).

1. Με ένα BFS/ DFS, διασχίζουμε το δέντρο T_1 και συγκρίνουμε τις πλευρες του με αυτές του T_2 .
2. Σημειώνουμε τις πλευρές, που διαφέρουν, έστω το σύνολο Ε, και τον αριθμό τους.

3. Λόγω της (1) θα χρειαστεί να εκτελέσουμε τον αλοριθμο του (α), τόσες φορές όσες υποδεικνύει η απόσταση $d_H(T_1, T_2)$.
4. Για κάθε $e \in E$, *όπου* $e \in T_1 \setminus T_2$, εκτελούμε τον αγόριθμο του (α) και βρίσκουμε μια ανταλλάξιμη ακμή, που θα δημιουργήσει ένα νεό συνδετικό δέντρο, το οποίο θα μοίαζει συνεχώς όλο κι περισσότερο στο T_2 .
5. Κάθε δέντρο **T'**, το οποίο παράγεται, το αποθηκεύουμε σε ένα πίνακα γονέων για τη δημιουργία του μονοπατιού στο H.

Η πολυπλοκότητα είναι $\boldsymbol{O(kn)}$, όπου $k = d_H(T_1, T_2)$ άρα $\boldsymbol{O(n^2)}$, αφού στην χειρότερη περίπτωση θα διαφέρουν κατά **n-1** ακμές(εφόσον έχουμε συνδετικό δέντρο).

γ)
Μπορούμε να εφρμόσουμε τον αλοριθμο Kruskal και σε χρόνο $\boldsymbol{O(mlogm)}$ να βρούμε το MST(G). Ύστερα, για κάθε ακμή e_i :

1. Την προσθέτουμε στο MST(G) και γνωρίζουμε ότι, έχει δημιουργηθεί κύκλος.
2. Με ένα DFS, σε χρόνο $\boldsymbol{O(n)}$, βρισκουμε τον κύκλο και την αμεσως μικροτέρη ή ίση σε βάρος ακμή από την e_i και την αφαιρούμε .

<u>Πολυπλοκότητα:</u>

Η συνολική πολυπλοκότητα είναι $\boldsymbol{O(nm)}$.

<u>Ορθότητα:</u>

Την ορθότητα μας την εγγυάται αρχικά ο αλγόριθμος Kruskal για την εύρεση του MST. Έπειτα, η αφαίρεση της μεγίστου ακμής κάθε φορά σε συνδυασμό με την ικανοποίηση των περιορισμών(εκάστοτε ακμή στο κύκλο), μας δίνει το ελάχιστο δυνατό συνδετικό δέντο, παρουσία της επιθυμητής ακμής.