Δάσκος Ραφαήλ – Α.Μ.: 03116049

Αλγόριθμοι και Πολυπλοκότητα
Ροή Λ: Λογισμικό Η/Υ
Σ.Η.Μ.Μ.Υ. – Ε.Μ.Π. – 7ο εξάμηνο
3η σειρά γραπτών ασκήσεων

Άσκηση 1: Ο Άγιος Βασίλης και τα Χρυσά Κλειδιά

Για τη λύση του προβλήματος μας θα χρησιμοποιήσουμε dfs αλγόριθμο.
Αρχικά, όμως, σημειώνουμε σε λίστα γειτνίασης όλους τους γείτονες του κάθε κόμβου (γείτονα θεωρούμε κάθε κουτί που μπορεί να ανοίξει με τα κλειδιά που υπάρχουν μέσα στο κουτί που κοιτάμε).

Άσκηση 2: Μακρύτερο Μονοπάτι σε Δέντρο

Αν χρειαζόταν να υπολογίσουμε απλώς το μακρύτερο μονοπάτι θα μπορούσαμε να το κάνουμε γρήγορα με μια διάσχιση (post-order). Επίσης χρειάζεται να αναφέρουμε, γιατί θα μας βοηθήσει στον αλγόριθμο μας, στο μονοπάτι του δέντρου μπορεί να υπάρχει το πολύ μία κορυφή από την οποία το μονοπάτι περνάει από το δεξί και από το αριστερό υπόδεντρό της.
Το μακρύτερο, λοιπόν, μονοπάτι του δέντρου θα υπακούει σε αυτή την ιδιότητα που είπαμε. Έτσι θέλουμε να υπολογίσουμε για κάθε κόμβο μεγαλύτερο μονοπάτι (θα λέμε αυτή τη λύση ως opt για να μη γράφουμε πολλές φορές το καλύτερο μονοπάτι) που προκύπτει για τις δύο επιλογές που έχουμε (επιλογή των δύο υποδέντρα του ή όχι):
Το μονοπάτι περνά από τον κόμβο u κα επιλέγει και δύο από τα υπόδεντρα του. Τότε opt(u)=opt(καλύτερου υπόδεντρου)+cost(u)+opt($2^{ου}$ καλύτερου υπόδεντρου).
Το μονοπάτι ξεκινάει από τον κόμβο u και δεν περνά από δύο υπόδεντρα του. Τότε opt(u)=max{opt(καλύτερου υπόδεντρου)+cost(u),cost(u)}.
Με τη χρήση μιας post-order διάσχισης μπορούμε σε γραμμικό χρόνο να υπολογίσουμε εύκολα τα opt(φύλλων) και στη συνέχεια τα opt των προγόνων παίρνοντας τα opt των απογόνων τους που δεν περνάνε για κανέναν απόγονο από παραπάνω από ένα υπόδεντρο.
Σημαντικό για να μπορέσουμε να έχουμε σωστή λύση στον αλγόριθμό μας είναι να κρατήσουμε κατά τη διάρκεια της post-order τα εξής:

- Έναν πίνακα 2 θέσεων για κάθε κόμβο με τα opt αν το μονοπάτι περνάει και από τους δύο του απογόνους και αν περνάει μόνο από έναν.
- Έναν πίνακα 2 θέσεων με τους απογόνους σε κάθε μία από τις προαναφερθέντες περιπτώσεις.

- Την αξία του καλύτερου μονοπατιού που έχουμε μέχρι το σημείο που τρέχουμε.
- Τον κόμβο εκκίνησης του καλύτερου μονοπατιού μέχρι το σημείο που είμαστε.

Στο τέλος της διάσχισης μπορούμε να υπολογίσουμε μέσω του δείκτη στον κόμβο με το καλύτερο μονοπάτι και τους απογόνους του ποιο είναι αυτό και να υπολογίσουμε και την τιμή του.

Ορθότητα: Υποθέτουμε ότι για έναν τυχαίο κόμβο έχουμε υπολογίσει σωστά το opt των παιδιών του. Θα δείξουμε πως μπορούμε να υπολογίσουμε σωστά και το δικό του opt.
Επαγωγική υπόθεση: οι κόμβοι που εξετάζουμε είναι τα παιδιά του που έχουμε ήδη υπολογίσει το opt. Έστω ότι c_1, c_2 τα παιδία του κόμβου που κοιτάμε (s). τότε έχουμε πως η αξία του μακρύτερου μονοπατιού θα είναι ίση με:

$$opt(s) = (opt(c_1) + cost(s), opt(c_2) + cost(s), opt(c_1) + opt(c_2) + cost(s), cost(s)).$$

Ο αλγόριθμος που έχουμε πει πριν ακολουθεί ακριβώς αυτό το κριτήριο επομένως είναι λογικό να είναι και ορθός.

Πολυπλοκότητα: Η πολυπλοκότητα της λύσης ταυτίζεται με αυτή της διάσχισης σε ένα δέντρο που είναι O(n), αν n το πλήθος των κόμβων.

Άσκηση 3: Κλέφτες και Αστυνόμοι

Ο αλγόριθμος που θα θελήσουμε να φτιάξουμε είναι σαν το minimax αλγόριθμο. Αυτός βρίσκει τη βέλτιστη ακολουθία κινήσεων για παιχνίδι που παίζεται σε γύρους και με δύο παίχτες. Για να μπορέσουμε να βρούμε τη λύση μοντελοποιούμε κάθε κατάσταση του παιχνιδιού ως έναν κόμβο ενός γράφου που έχει ακμές προς όλες τις επόμενες δυνατές καταστάσεις που υπάρχουν. Επίσης σε κάθε κόμβο έχουμε μια συνάρτηση που μας δείχνει πόσο νικάει ο παίχτης που παίζει πρώτος. Ο παίκτης που παίζει πρώτος κάνει πάντα τις κινήσεις που μεγιστοποιούν την τιμή της συνάρτησης αυτής ενώ ο παίκτης που παίζει δεύτερος προσπαθεί να ελαχιστοποιήσει αυτή την τιμή.
Για την άσκηση μας φτιάχνουμε ένα γράφο G' ο οποίος περιέχει σαν καταστάσεις (V') όλες τις πιθανές τριάδες του παιχνιδιού (k, a, p), με k, a ∈ V και το p να είναι είτε 0 είτε 1. 0 σημαίνει πως η επόμενη κίνηση ανήκει στον Αντρέα και 1 ότι μετά παίζει ο Κώστα. Επίσης έχουμε πως k είναι η θέση στην οποία βρίσκεται ο Κώστας και a αυτή που είναι ο Αντρέας. Όταν, δηλαδή, έχουμε p=0 μετακινόμαστε από το k σε μια γειτονική κορυφή του γράφου G και όταν έχουμε p=1 μετακινόμαστε από το a σε μια γειτονική κορυφή (δεν μπορούμε να πάμε όταν κινούμε το a στη θέση d που είναι το καταφύγιο.
Στη συνέχεια, αφού έχουμε το γράφο, με τη χρήση BFS αλγορίθμου υπολογίζουμε την τριάδα (w, l, t) που είναι (πόσο απέχει ο κόμβος από την κοντινότερη νίκη για τον συγκεκριμένο παίχτη, πόσο απέχει ο κόμβος από την κοντινότερη ήττα, πόσο απέχει ο κόμβος από την κοντινότερη ισοπαλία). Νίκη σημαίνει να φτάσει ο Κώστας στο καταφύγιο αν παίζει εκείνος ή να βρεθεί ο Αντρέας στην ίδια θέση με τον Κώστα, ήττα

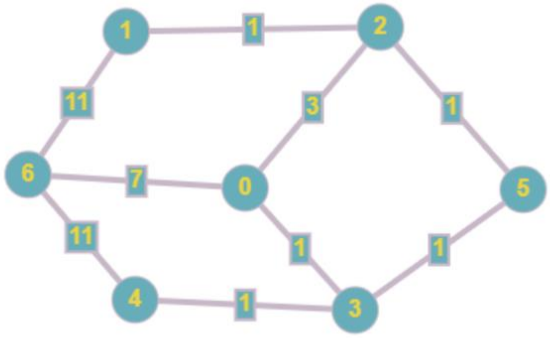
είναι ακριβώς το ανάποδο από αυτό. Ισοπαλία είναι η εύρεση κύκλου στο γράφο, να γυρίσουμε δηλαδή σε κάποια θέση που έχουμε ήδη επισκεφτεί.
Στη συνέχεια θα υπολογίσουμε τις τιμές q=b-a και m=b-c. Αφού τα έχουμε βρει όλα αυτά χρησιμοποιούμε τον αλγόριθμο minimax δύο φορές, μία για το γράφο με τη χρήση του q και μία με τη χρήση του m. Ο παίκτης max είναι ο Κώστας. Προσπαθεί να κάνει μέγιστη την q που σημαίνει να έρθει πιο κοντά στην νίκη του ενώ ο min είναι ο Αντρέας που επιχειρεί το αντίθετο.
Όταν τελειώσουμε αυτό κάνουμε trace του minimax με τα q και βλέπουμε την ακολουθία κινήσεων αλλά και ποιος θα νικήσει. Αν κερδίζει ο Αντρέας κάνουμε trace και με το m για να δούμε εάν ο Κώστας με διαφορετική ακολουθία κινήσεων μπορεί να προκαλέσει ισοπαλία και να μη χάσει το παιχνίδι.

Ορθότητα: Η ορθότητα μας προκύπτει από το γεγονός ότι έχουμε μια συνεπή ευριστική συνάρτηση που είναι η διαφορά μεταξύ της απόστασης νίκης του ενός από του άλλου η οποία όμως πάντα εξαρτάται από την κατάσταση του παιχνιδιού στην οποία βρισκόμαστε. Επίσης, επειδή με το BFS θα εξερευνήσουμε όλους τους κόμβους του γράφου έχουμε έναν εξαντλητικό χώρο αναζήτησης (περιέχει όλες τις πιθανές καταστάσεις). Τέλος έχουμε και τη δυνατότητα ισοπαλίας, που αν ένας παίκτης δεν μπορεί να κερδίσει μπορεί να θέσει ως στόχο την ισοπαλία και όχι τη νίκη. Όλα αυτά μας οδηγούν, μαζί με την ορθότητα του minimax, στο γεγονός ότι λαμβάνουμε βέλτιστη λύση από τον αλγόριθμό μας.

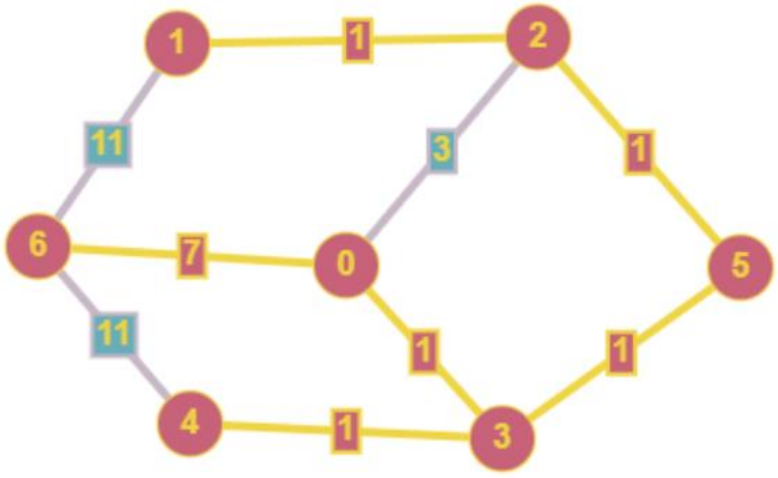
Πολυπλοκότητα: Ο γράφος G' που δημιουργούμε έχει $2V^2$ καταστάσεις και οι μεταβάσεις του στην επόμενη θέση του παιχνιδιού μπορούν να υπολογιστούν σε χρόνο O(1). Έτσι η συνολική πολυπλοκότητα σχηματισμού του είναι $O(V^2)$.
Επίσης, αυτός ο γράφος έχει περίπου E^2 ακμές επομένως η πολυπλοκότητα του BFS για να βρούμε τις τιμές (k, a, p) είναι ίση με $O(V^2+E^2)$.
Τέλος ο minimax κάνει μια διάσχιση του γράφου και επομένως έχει πολυπλοκότητα ίση με $O(V^2+E^2)$.
Οπότε καταλήγουμε πως η συνολική χρονική πολυπλοκότητα του αλγορίθμου μας είναι ίση με $O(V^2+E^2)$.

Άσκηση 4: Ελάχιστο Συνδετικό Δέντρο με Περιορισμούς

(α) Θα δείξουμε με τη χρήση ενός αντιπαραδείγματος πως η άπληστη λογική, η οποία συμπεριλαμβάνει στο συνδετικό δέντρο τις k ακμές μικρότερου βάρους που προσπίπτουν στην κορυφή s, δεν οδηγεί πάντα σε βέλτιστη λύση.
Έστω ότι έχουμε τον παρακάτω γράφο:



Και θέλουμε να έχουμε τουλάχιστον 2 ακμές στον κόμβο 0. Τότε το ελάχιστο συνδετικό δέντρο που θα προκύψει από το greedy κριτήριο θα λάβει κατευθείαν τις ακμές 0-2 και 0-3 καθώς είναι αυτές με το μικρότερο μήκος. Όμως έτσι το ελάχιστο συνδετικό δέντρο που θα προκύψει θα έχει άθροισμα βαρών ίσο με 18. Μπορούμε να δούμε πως από την παρακάτω εικόνα πως αν πάρουμε τις ακμές 0-6 και 0-3 θα έχουμε άθροισμα ίσο με 12.



Οι κίτρινες είναι οι ακμές που έχουμε επιλέξει και οι γκρι αυτές που δεν έχουμε διαλέξει. Το παραγόμενο δέντρο έχει 2 ακμές στο 0 και όπως έχουμε πει και προηγουμένως θα παραλείψουμε την ακμή με βάρος 3 παρόλο που είναι μικρότερη από την 7 γιατί έτσι το τελικό δέντρο δεν είναι το ελάχιστο που μπορούμε να πάρουμε.

(β) Θέλουμε τώρα να βρούμε έναν αλγόριθμο ο οποίος να υπολογίζει ένα ελάχιστο συνδετικό δέντρο Τα*(s, k) στο οποίο η κορυφή s να έχει βαθμό ίσο με k, καθώς το greedy κριτήριο δε μας κάνει για την εύρεση σωστής λύσης.

Για τον αλγόριθμο που θα χρησιμοποιήσουμε για την εύρεση της λύσης είναι αρχικά να πούμε πως αν το πλήθος των ακμών του κόμβου s είναι ίσο με k τότε παίρνουμε όλες τις ακμές του και στη συνέχεια υπολογίζουμε, με έναν αλγόριθμο εύρεσης ελαχίστου

συνδετικού δέντρου, από το υπόλοιπο γράφημα ενώ θεωρούμε τους κόμβους που είναι ενωμένοι με τον s ως έναν και οι ακμές τους με τους υπόλοιπους να έχουν βάρος ίσο με την ελάχιστη ακμή που φεύγει από οποιονδήποτε από τους γείτονες του s προς κάθε έναν από τους μη γειτονικούς κόμβους του. Έτσι, εν τέλει, θα πάρουμε ένα ελάχιστο συνδετικό δέντρο με τον περιορισμό ότι ο κόμβος s έχει k ακμές σε αυτό.
Αν, όμως, ο κόμβος s έχει περισσότερες από k ακμές εργαζόμαστε ως εξής. Στην αρχή αφαιρούμε τον κόμβο s που μας ενδιαφέρει. Στη συνέχεια για κάθε έναν από τους υπογράφους που θα προκύψουν εξαιτίας της έλλειψης του κόμβου αυτού θα υπολογίσουμε τον ελάχιστο συνδετικό τους δέντρο με τη χρήση ενός από τους αλγορίθμους υπολογισμού, όπως για παράδειγμα τον Kruskal και Prim. Έπειτα βάζουμε τον κόμβο s και με κάθε ένα από τα υποδέντρα που έχουμε τοποθετούμε την ελάχιστη ακμή μεταξύ του κόμβου αυτού και του δέντρου.
Αν το σύνολο των ακμών που έχουμε προσθέσει, έστω a, είναι ίδιο με τον αριθμό k που θέλουμε τότε έχουμε τελειώσει τον υπολογισμό μας.
Αν όμως το k είναι μεγαλύτερο από το a χρησιμοποιούμε τα παρακάτω. Για κάθε μία από τις μη επιλεγμένες ακμές e_i που ενώνουν τον κόμβο s με κάποιον γείτονα του στο G που δεν υπάρχει όμως στο δέντρο που έχουμε υπολογίσει έως τώρα, ο γείτονας n_i θα υπολογίσουμε τη μέγιστη σε βάρος ακμή που υπάρχει στο μονοπάτι $s\text{-}n_i$, θα την ονομάσουμε αυτή m_i, επίσης δεν μπορούμε να επιλέξουμε ακμή του s γιατί αν μετά χρειαστεί να την αφαιρέσουμε θα μειώσουμε το a (βαθμός του s, όπως έχουμε πει). Έτσι μπορούμε να υπολογίσουμε το κόστος $c_i = w(m_i) - w(e_i)$, που μας δείχνει τι μπορούμε να σώσουμε αν αντικαταστήσουμε την ακμή m_i από την e_i στο συνδετικό δέντρο.
Από αυτά που έχουμε υπολογίσει ταξινομούμε τα (c_i, m_i, e_i, n_i) σε μια ουρά προτεραιότητας με φθίνον c_i.
Στη συνέχεια, όσο έχουμε πως το a είναι μικρότερο του k παίρνουμε την πρώτη τετράδα από την ουρά μας. Αν το m_i έχει αφαιρεθεί από το βρίσκουμε την ακμή που νέο m_i και c_i με βάση το δέντρο που έχουμε έως τώρα και ανανεώνουμε τη θέση της τετράδας στην ουρά μας με τα νέα στοιχεία που έχει. Αλλιώς αντικαθιστούμε την ακμή m_i με την e_i και αφαιρούμε την τετράδα από την ουρά.
Αυτός ο αλγόριθμος μας δίνει το επιθυμητό αποτέλεσμα καθώς κάθε φορά προσθέτουμε την ακμή που βελτιώνει με τον καλύτερο τρόπο το δέντρο μας, ή το χειροτερεύει κατά το μικρότερο ποσό) ανεξαρτήτως αν έχει μεγαλύτερο βάρος από κάποια από τις υπόλοιπες ακμές του s. Και ο αλγόριθμός μας θα σταματήσει όταν φτάσουμε βαθμό k. Επίσης δεν μπορεί να επιλεγεί κάποια από τις ήδη τοποθετημένες ακμές του κόμβου s καθώς το λύνουμε αυτό σαν πρόβλημα. Έτσι θα έχουμε πως θα καταφέρει κάποια στιγμή να φτάσει στο ελάχιστο συνδετικό με τον ζητούμενο περιορισμό.
Χρειαζόμαστε O(n) για τον υπολογισμό των m_i χρησιμοποιώντας ένα αλλαγμένο DFS (κρατάμε σε κάθε κόμβο το μέγιστο βάρος μέχρι εκείνον για να τον δουν και οι γείτονες του). Επίσης ο αλγόριθμος του Prim έχει πολυπλοκότητα $O(n^2)$. Η δεύτερη φάση του υπολογισμού με τις τετράδες και την ουρά έχει χρονική πολυπλοκότητα ίση με $O(n^2)$

καθώς θα κάνουμε το πολύ n φορές και μπορεί να χρειαστεί να υπολογίσουμε O(n) για το νέο m_i. Έτσι η συνολική πολυπλοκότητα του αλγορίθμου μας είναι ίση με $O(n^2)$.

Άσκηση 5: Αλγόριθμος Boruvka

(α) Διατύπωση του αλγορίθμου του Boruvka: Αρχικά έχουμε όλες τις ακμές του G αχρωμάτιστες και θεωρώ πως κάθε κόμβος αποτελεί τετριμμένα ένα δέντρο. Για κάθε ένα από τα δέντρα που έχουμε (στην αρχή απλοί κόμβοι) επιλέγουμε όλες τις ελάχιστες ακμές που προσπίπτουν στους κόμβους που έχουμε στην αρχή και τις τοποθετούμε στο δάσος που έχουμε. Στη συνέχεια για κάθε ένα από τα διαφορετικά δέντρα που έχουμε μέχρι εκείνο το σημείο επιλέγουμε την ελάχιστη ακμή που μας οδηγεί σε διαφορετικό δέντρο και την προσθέτουμε στο δάσος μας. Συνεχίζουμε με τον ίδιο τρόπο έως ότου να έχουμε μόνο ένα δέντρο στο τέλος.

(i) θα θεωρήσω για να κάνω πιο απλό το πρόβλημά μου πως κάθε ακμή έχει μοναδικό βάρος (δεν υπάρχουν διαφορετικές ακμές με ίδια τιμή βάρους).

Έστω τώρα ότι μετά την εκτέλεση του Boruvka δε θα πάρω ένα συνδετικό δέντρο. Θα πρέπει τότε να έχουμε καταλήξει σε δέντρα T_u και T_v που να μην μπορώ να τα ενώσω. Αυτό θα ήταν άτοπο γιατί αφού το γράφημα είναι συνεκτικό, υπάρχει για κάθε κόμβο στο πρώτο δέντρο μονοπάτι προς κόμβο στο δεύτερο επομένως πάντα θα μπορεί να βρεθεί εν τέλει ακμή που να τα ενώνει. Το δεύτερο πρόβλημα που μπορεί να έχουμε είναι όταν πάμε να επιλέξουμε μια ακμή e = u-v μονοπάτι η οποία είναι η ελάχιστη που προσπίπτει πάνω στο T_u και στο T_v και θα δημιουργηθεί κύκλος (τα δύο δέντρα μου θα είναι υπογράφοι ενός άλλου δέντρου που έχει ήδη υπολογιστεί). Άρα θα έχουν επιλεγεί ακμές που θα οδηγούν από το u στο v χωρίς το e. Επίσης θεωρώ ότι μέχρι τότε έχω ακολουθήσει τα βήματα τοποθέτησης ακμών. Τότε, αν έχω το μονοπάτι T_u-e_1-T_2-e_2-…-…- e_{n-1}-T_v. Έχουμε ότι η ακμή e είναι διαφορετική της e_1 και εφόσον είναι η μικρότερη που προσπίπτει στο T_u ισχύει ότι $cost(e)<cost(e_1)$, όμως, αφού ισχύει αυτό για να έχει επιλεγεί η e_1 πρέπει να είναι η μικρότερη ακμή που προσπίπτει στο T_2 και συνεχίζει αντίστοιχα. Οπότε καταλήγουμε πως: $cost(e)<cost(e_1)< cost(e_2)<…< cost(e_{n-1})$. Αυτό σημαίνει ότι η e_{n-1} πρέπει να είναι η ακμή που έχει το μικρότερο βάρος από αυτές που προσπίπτουν στο T_v που όμως από την υπόθεσή μας είναι άτοπο, επομένως ο αλγόριθμος του Boruvka πάντα μας οδηγεί σε συνδετικό δέντρο.

(ii) έστω Τα το δέντρο που προκύπτει από τον αλγόριθμο του Boruvka, αλλά Τ’ το συνδετικό δέντρο ελαχίστου βάρους. Έστω ότι τα δύο αυτά δέντρα είναι διαφορετικά και e’ η κορυφή που ανήκει στο Τ αλλά όχι στο Τ’. Ορίζουμε ως p το μονοπάτι στο Τ’ που ενώνει τους κόμβους στους οποίους προσπίπτει η e’. Από τη στιγμή που έχουμε επιλέξει την ακμή με τον αλγόριθμό μας σημαίνει ότι τουλάχιστον μια ακμή του μονοπατιού p δεν έχει επιλεγεί. Ωστόσο, σύμφωνα με τον αλγόριθμο θα έχουμε ότι w(e)>w(e’). Όμως έχουμε πει ότι το Τ’ είναι το συνδετικό δέντρο ελαχίστου βάρους, άτοπο καθώς δεν έχει

την ακμή e’ που όπως είδαμε ελαχιστοποιεί τα βάρη. Άρα, ο αλγόριθμος του Boruvka μας οδηγεί σε συνδετικό δέντρο ελαχίστου βάρους.

(β) pseudocode for Boruvka ‘s algorithm:

```
input: A graph G whose edges have distinct weights.
output: F is the minimum spanning forest of G.
Initialize a forest F to be a set of one-vertex trees, one for each vertex of the graph.

while F has more than one component do
    Find the connected components of F and label each vertex of G by its component
    Initialize the cheapest edge for each component to "None"
    for each edge uv of G do
        if u and v have different component labels:
            if uv is cheaper than the cheapest edge for the component of u then
                Set uv as the cheapest edge for the component of u
            if uv is cheaper than the cheapest edge for the component of v then
                Set uv as the cheapest edge for the component of v
    for each component whose cheapest edge is not "None" do
        Add its cheapest edge to F
```

Αυτός είναι ο ψευδοκώδικας του Boruvka όπως μπορούμε να το βρούμε και στο Wikipedia. Ο αλγόριθμος αυτός χρειάζεται για το εξωτερικό loop O(logV) πράξεις μέχρι να τερματίσει καθώς κάθε φορά μειώνει στο μισό το πλήθος των δέντρων που υπάρχουν και άρα ο χρόνος τρεξίματός είναι O(E*logV), αν πάρουμε, λοιπόν E = m και V = n τότε θα έχουμε χρόνο O(m*logn).

(γ) Χρήση του αλγόριθμου του Prim με σωρό Fibonacci για να μειώσω την πολυπλοκότητα του αλγορίθμου του Boruvka σε O(m*loglogn).
Θα τρέξω τον αλγόριθμό μου loglogn φορές, έτσι αντί να πάρω μόνο ένα δέντρο στο τέλος, θα έχω ένα δάσος με n/logn δέντρα. Στη συνέχεια φτιάχνω ένα γράφημα G’’ που να έχει έναν κόμβο για κάθε ένα από τα δέντρα του δάσους και ακμές τις ελαφρύτερες ακμές που ενώνουν τα δέντρα μεταξύ τους. Στη συνέχεια χρησιμοποιώ τον αλγόριθμο του Prim με Fibonacci σωρούς και σε χρόνο O(V+E), με V=n/logn και E=m παίρνω ένα δέντρο στο οποίο αν απλοποιήσω τα συστατικά του με βάση τα δέντρα που έχει βρει ο αλγόριθμος του Boruvka στην αρχή θα έχω πολυπλοκότητα ίση με O(mloglogn)

(δ) Ξαναγράφω τον αλγόριθμο του Boruvka:

Ξεκινάω από ένα γράφημα G που ανήκει σε μια βολική κλάση γραφημάτων.

Για κάθε έναν από τους κόμβους επιλέγω την ελαφρύτερη ακμή που προσπίπτει σε αυτόν.

Δημιουργώ υπερκόμβους ενώνοντας τα στοιχεία που είχαν τις ελαφρύτερες ακμές.

Πραγματοποιώ τα παραπάνω μέχρι να έχουμε σύνολο δέντρων ίσο με ένα.

Επειδή στον αλγόριθμο του Boruvka σε κάθε στάδιο υποδιπλασιασμό των κόμβων (δέντρων που δημιουργούνται) θα έχουμε πολυπλοκότητα ίση με:

$$O(|V|+|V|/2+|V|/4+\ldots+1) = O(|V|)$$

Άσκηση 6: Μονοπάτια Ελάχιστου Bottleneck Κόστους για όλα τα Ζεύγη Κορυφών

(α) Θέλουμε να δείξουμε ότι αν έχουμε ένα T* ελάχιστο συνδετικό δέντρο του ενός απλού συνεκτικού μη κατευθυνόμενου γραφήματος G (V, E, l) , τότε για κάθε ζευγάρι κορυφών: u, v ∈ V θα ισχύει πως το μονοπάτι u-v στο T* (το οποίο αναγκαστικά αφού έχουμε δέντρο είναι και μοναδικό) αποτελεί ένα μονοπάτι ελαχίστου κόστους. Έχουμε, δηλαδή, πως το bottleneck κόστος του κάθε μονοπατιού είναι το ελάχιστο. Άρα θέλουμε να δείξουμε πως τα κόστη $\max_{e\in p}\{l(e)\}$ είναι τα ελάχιστα δυνατά για τα μονοπάτια μας στο γράφημα T*, εάν p τα μοναδικά μονοπάτια μεταξύ δύο κόμβων που υπάρχουν σε αυτό και l(e) το κόστος κάθε μιας από τις ακμές αυτών των μονοπατιών.

Για να το αποδείξουμε αυτό χρησιμοποιήσουμε τη βοήθεια του Kruskal αλγορίθμου. Θέτουμε P_i τις πρώτες i ακμές που έχει εξετάσει ο αλγόριθμος αυτός, S_i τις ήδη επιλεγμένες ακμές στο ελάχιστο συνδετικό δέντρο και G[X] το γράφημα που έχει όλους τους κόμβους του αρχικού ενωμένες με τις X ακμές.

Για κάθε i το σύνολο των συνδεδεμένων κόμβων στο $G[P_i]$ ταυτίζεται αυτό του $G[S_i]$ καθώς αυτά τα δύο σύνολα διαφέρουν μόνο στις ακμές που πραγματοποιούν κύκλο και άρα θα έχουν το ίδιο πλήθος ενωμένων κόμβων.

Έστω, τώρα πως έχουμε y τέτοιο ώστε το E_y να έχει όλες τις ακμές με μήκος μικρότερο ή ίσο του m, αν m είναι το bottleneck κόστος μεταξύ δύο κόμβων u, v (δηλαδή ο μικρότερος δυνατός αριθμός που μπορούμε να μεταβούμε από τον ένα κόμβο στον άλλο χρησιμοποιώντας μόνο ακμές μεγέθους το πολύ m). Τότε θα έχουμε ότι τα u, v ανήκουν στους ενωμένους κόμβους του $G[P_y]$ σύμφωνα με τους ορισμούς των m, y. Και με βάση την προηγούμενη διατύπωση θα ανήκουν και στο $G[S_y]$.

Όταν τελειώσει ο αλγόριθμός μας θα προκύψει ότι για κάθε δύο κόμβους u, v το μονοπάτι που θα υπάρχει στο παραγόμενο ΕΣΔ θα περιέχει ακμές που θα έχουν μήκος το πολύ ίσο με το bottleneck κόστος του μονοπατιού. Επομένως, καθώς ο αλγόριθμος Kruskal μπορεί να μας δώσει όλα τα ΕΣΔ του γραφήματος μας, μπορούμε να πούμε ότι για κάθε T* (ΕΣΔ του G) το μοναδικό μονοπάτι u-v είναι ελαχίστου κόστους.

(β) Για το ερώτημα αυτό έχουμε ως δεδομένο το ελάχιστου συνδετικό δέντρο και θέλουμε να υπολογίσουμε το συνολικό κόστος των μονοπατιών ελαχίστου κόστους για όλα τα ζευγάρια των διαφορετικών κορυφών του γραφήματος G.

Για να μπορέσουμε να το πετύχουμε αυτό χρειάζεται αρχικά να ταξινομήσουμε κατά φθίνουσα σειρά τις ακμές που υπάρχουν στο ΕΣΔ που έχουμε. Θα πρέπει να τοποθετήσουμε και μια μεταβλητή αριθμός που θα μπορεί να αλλάζει για κάθε κόμβο μετά από κάθε διαγραφή και θα μας δείχνει με ποια στοιχεία είναι συνδεδεμένοι οι κόμβοι μετά από μια διαγραφή (στο δέντρο αυτό θα έχουμε τα μέγιστα στην κορυφή και όσο πάμε προς τα κάτω θα μειώνονται οι τιμές των ακμών και στα φύλλα θα έχουμε τους κόμβους του γραφήματος). Αρχικά θα έχουμε όλες οι μεταβλητές αριθμός είναι αρχικοποιημένες με 1. Θα χρησιμοποιήσουμε και έναν πίνακα n μεγέθους που θα τοποθετούμε σε αυτόν αναλόγως με τον αριθμό που είμαστε ποιος είναι ο πατέρας των στοιχείων με τον συγκεκριμένο αριθμό.

Έπειτα χρησιμοποιούμε την παρακάτω αναδρομή. Θέτουμε w = μέγιστο u-v βάρος ακμής και δημιουργούμε έναν κόμβο για να τον τοποθετήσουμε στο καρτεσιανό δέντρο που θα φτιάξουμε. Μετά κάνουμε DFS αλγόριθμο από τους κόμβους u και v ταυτόχρονα και ο πρώτος αλγόριθμος που θα τερματίσει έχει το μικρότερο υπόδεντρο και στη συνέχεια βάζουμε μια νέα τιμή στη μεταβλητή αριθμός σε όλους τους κόμβους του υπόδεντρου αυτού (αυξάνουμε την τελευταία τιμή αριθμός στο υπόδεντρο του u αν αυτό τελείωσε πρώτο ή αντίστοιχα και για το v). Θα χρειαστεί να βάλουμε και στο κελί του νέου αριθμού πως ο πατέρας του είναι η ακμή που μόλις βγάλαμε για να ξέρουμε πως όταν φύγει ακμή από αυτό το υπόδεντρο θα τοποθετηθεί κάτω από τον πατέρα που έχουμε σημειώσει.

Συνεχίζουμε με τον τρόπο που αναφέραμε μέχρι να μην υπάρχουν άλλες ακμές. Έτσι θα έχουμε καταλήξει σε απλούς κόμβους οι οποίοι θα έχουν ο καθένας έναν ξεχωριστό αριθμό και με τη βοήθεια αυτού θα μπορέσουμε να τους βάλουμε σωστά κάτω από την τελευταία ακμή στην οποία αντιστοιχούν ως φύλλα.

Η λύση του προβλήματός μας είναι να υπολογίσουμε πλέον όλους του ελάχιστους κοινούς προγόνους μεταξύ δύο φύλλων του δέντρου, τον πρώτο δηλαδή κοινό πρόγονο των δύο φύλλων που συναντάμε καθώς ανεβαίνουμε προς την κορυφή του δέντρου. Με ένα πέρασμα ξεκινώντας από τα φύλλα υπολογίζουμε πόσα φύλλα έχει στη δεξιά του μεριά και πόσα στην αριστερή ο κάθε κόμβος. Έχει τόσα δεξιά φύλλα όσα το άθροισμα όλων των φύλλων του δεξιού του παιδιού και αριστερά φύλλα όσο το άθροισμα του αριστερού του παιδιού, οπότε υπολογίζεται σε χρόνο O(n) καθώς είναι ένα πέρασμα από κάθε κόμβο (με post-order και έχουμε n+(n-1) κόμβους, όσοι δηλαδή ο αρχικοί κόμβοι και το πλήθος των ακμών).

Αν ξέρουμε πόσα φύλλα έχει δεξιά (έστω r στο πλήθος) και πόσα αριστερά (έστω l στο πλήθος) ένας κόμβος, τότε μπορούμε να αποφανθούμε ότι το πλήθος των συνδυασμών u, v που έχουν bottleneck κόστος ίσο με τον αριθμό του κόμβου (έστω w) είναι ίσος με r*l. Οπότε εξαιτίας αυτών των συνδυασμών προστίθεται στο συνολικό άθροισμα ένα κόστος

ίσο με w*r*l. Επομένως μπορούμε ταυτόχρονα με το προηγούμενο πέρασμα που κάνουμε να υπολογίζουμε, πριν φύγουμε από κάθε κόμβο το bottleneck κόστος που προκύπτει εξαιτίας αυτού του αριθμού, όπως είπαμε και πριν λίγο, και να το προσθέτουμε στο συνολικό άθροισμα που θα έχουμε.
Έτσι στο τέλος αυτής της αναδρομής θα έχουμε άθροισμα ίσο με το ζητούμενο συνολικό κόστος των μονοπατιών ελαχίστου κόστους για όλα τα ζευγάρια των διαφορετικών κορυφών του γραφήματος G.

Ορθότητα: ο αλγόριθμος αυτός τοποθετεί αρχικά όλες τις μεγαλύτερες ακμές κρατώντας τη συνεκτικότητα όλων των κόμβων, καθώς τοποθετούνται οι κόμβοι έπειτα από αυτόν που πρέπει είτε δεξιά είτε αριστερά). Με τον τρόπο αυτό το τελικό μας δέντρο θα έχει όλα τα bottleneck κόστη αν βρούμε ποια είναι η μεγαλύτερη ακμή που υπάρχει που ενώνει τους δύο κόμβους (και γι' αυτό μας ενδιαφέρει το LCA των κόμβων αυτών). Στη συνέχεια μπορούμε να δούμε πως για να πάρουμε το επιθυμητό μας αποτέλεσμα χρειάζεται να βρούμε πόσα ζευγάρια έχουν το bottleneck που προκύπτει από την τιμή των ενδιάμεσων κόμβων. Για το λόγο αυτό είναι το πλήθος των ζευγαριών που δημιουργούνται από τα δεξιά φύλλα και από τα αριστερά φύλλα του κάθε κόμβου, καθώς αυτός είναι ο πρώτος κοινός πρόγονος που έχουν και οι δύο. Αν τα προσθέσουμε όλα αυτά στο τέλος παίρνουμε και το επιθυμητό αποτέλεσμα.

Πολυπλοκότητα**:** όπως αναφέραμε αρχικά θα χρησιμοποιήσουμε έναν αλγόριθμο ταξινόμησης για να έχουμε τις ακμές κατά φθίνουσα σειρά. Η πολυπλοκότητα ενός τέτοιου αλγορίθμου είναι O(n*logn), καθώς έχουμε n-1 στο πλήθος ακμές. Στη συνέχεια τις n-1 ακμές τις αφαιρούμε 1 προς 1 και σε κάθε μία από τις διαγραφές έχουμε χρόνο O(logn), άρα συνολικά O(n*logn) για όλες τις διαγραφές και τη δημιουργία του καρτεσιανού μας δέντρου. Τέλος, η post-order διάσχιση του δέντρου κάνει χρόνο O(n) καθώς όπως είπαμε και προηγουμένως έχουμε 2n-1 συνολικές κορυφές.
Έτσι θα καταλήξουμε σε χρόνο O(n*logn+n*log+n). Οπότε η πολυπλοκότητα του αλγορίθμου που χρησιμοποιήσαμε για την εύρεση του ζητούμενού μας είναι O(n*logn).