ΑΛΓΟΡΙΘΜΟΙ & ΠΟΛΥΠΛΟΚΟΤΗΤΑ

2η Σειρά γραπτών ασκήσεων

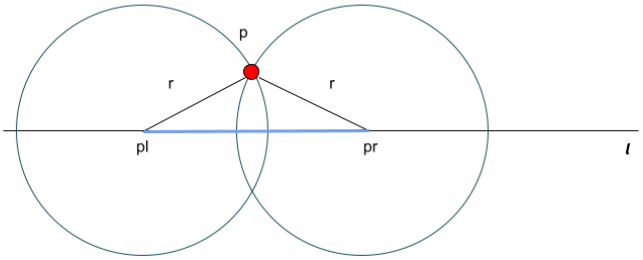
Ακαδημαϊκό έτος 2021-2022

$7^{ο}$ εξάμηνο

Νικόλας Μπέλλος | ΑΜ : el18183

Άσκηση 1

Αρχικά, παρατηρούμε ότι για κάθε σημείο p, για να μπορεί αυτό να καλυφθεί από ένα κύκλο ακτίνας r, θα πρέπει το κέντρο αυτού του κύκλου να βρίσκεται ανάμεσα στα αντίστοιχα σημεία pl και pr. Τα pl, pr είναι πάντα τέτοια ώστε να ισχύει d(pl,p) = d(p,pr) = r και pl ≤ pr.



Επομένως, για τα n αυτά σημεία αντιστοιχούμε κάθε σημείο p με το αντίστοιχο διάστημά του (pl, pr) το οποίο μπορούμε να το βρούμε κάθε φορά σε σταθερό χρόνο. Έτσι, έχουμε σπαταλήσει χρόνο O(n) για να βρούμε τα διαστήματα, και το πρόβλημά μας τώρα, ανάγεται σε αυτό του να βρούμε τον ελάχιστο αριθμό σημείων που χρειάζονται ώστε να ανήκουν όλα σε τουλάχιστον ένα διάστημα, δοσμένων n διαφορετικών μεταξύ τους διαστημάτων.

Αλγόριθμος

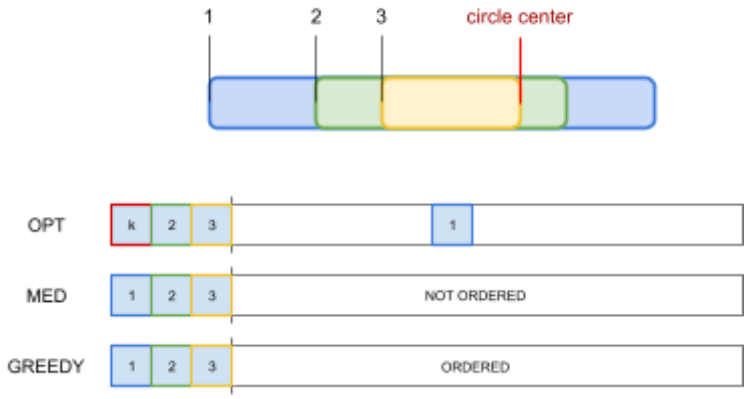
Αρχικά, θα ταξινομήσουμε όλα τα διαστήματα ως προς το pl (δηλαδή την αρχή του διαστήματος) σε αύξουσα σειρά (πολυπλοκότητα O(nlogn)). Έπειτα, ξεκινώντας από το διάστημα το οποίο ξεκινάει πρώτο (min pl), (έχοντας κρατήσει το τέλος του διαστήματος σε μία μεταβλητή, pr_1) παίρνουμε το διάστημα με την αμέσως μεταλύτερη αρχή διαστήματος και ανανεώνουμε το pr με τη μικρότερη τιμή που έχουμε βρεί. Σταματάμε να πηγαίνουμε στο επόμενο διάστημα μόλις ξεπεράσουμε τη μικρότερη τιμή pr που έχουμε βρεί, γιατί τότε θα είναι η "τελευταία ευκαιρία" που έχουμε να τοποθετήσουμε ένα σημείο στο διάστημα με το μικρότερο pr (min pr). Σε αυτό το σημείο τοποθετούμε ένα κύκλο με κέντρο του το σημείο αυτό (min pr) και αυτό που έχουμε καταφέρει είναι να περικλείσουμε στον ίδιο κύκλο όλα τα σημεία των οποίων τα αντίστοιχα διαστήματα είχαμε συναντήσει πριν σταματήσουμε το iteration. Έχοντας καλύψει ήδη τα πρώτα αυτά διαστήματα ο αλγόριθμος επαναλαμβάνεται εκ νέου για τα υπόλοιπα διαστήματα που έχουν μείνει (τα οποία εξακολουθούν να είναι ταξινομημένα). Ο αλγόριθμος αυτός κατατάσσεται στους greedy αλγορίθμους γιατί μπορούμε με ένα άπλητο κριτήριο κάθε φορά (αν ξεπεράσαμε το min pr) να επιλέξουμε το επόμενο βήμα του αλγορίθμου.

Πολυπλοκότητα

Η τελική **πολυπλοκότητα** θα είναι : $O(n) + O(nlogn) + O(n) =$ **O(nlogn).**

Απόδειξη ορθότητας

Περιγραφικά, αν ξεκινήσουμε να τοποθετήσουμε κάπου το πρώτο σημείο, ξεκινάμε από το σημείο που αρχίζει πρώτα το πρώτο διάστημα, έτσι είμαστε σίγουροι ότι δεν θα το αφήσουμε απέξω. Αν μετακινήσουμε το σημείο προς τα δεξιά, έτσι ώστε να μην βγούμε έξω από το πρώτο διάστημα, μας δίνει τη δυνατότητα να εντάξουμε ακόμα περισσότερα διαστήματα στο πρώτο σημείο. Αν συναντήσουμε το τέλος κάποιου διαστήματος θα πρέπει αναγκαστικά να βάλουμε το κέντρο ενός κύκλου εκεί γιατί αλλιώς το σημείο το οποίο αναπαριστά το διάστημα αυτό δεν θα περιέχεται σε κανένα κύκλο. Αυτή η λογική μας εξασφαλίζει ότι το πρώτο σημείο (δηλαδή το κέντρου κύκλου) θα εμπεριέχεται σε όσο το δυνατόν περισσότερα διαστήματα και όσα διαστήματα απομένουν θα είναι τα λιγότερα δυνατά (αυτό μας οδηγεί στην αρχή βελτιστότητας).



Πιο αναλυτικά, με βάση το παραπάνω σχήμα μπορούμε να δούμε ότι ο πίνακας MED με τον GREEDY ταυτίζονται για τα πρώτα στοιχεία του πίνακα, τα οποία αναπαριστούν σημεία/διαστήματα που θα ενταχθούν σε ένα κοινό κύκλο (άρα θα χρησιμοποιήσουμε 1 σημείο για 3 διαστήματα). Επομένως, από την αρχή βελτιστότητας βλέπουμε ότι ο GREEDY θα χρησιμοποιήσει το πολύ όσα σημεία και ο MED (GREEDY ≤ MED). Συγκρίνοντας, τώρα, τον OPT πίνακα με τον MED, αν εφαρμόσουμε τον παραπάνω αλγόριθμο στον OPT πίνακα τότε θα τοποθετήσουμε ένα κέντρο κύκλου το οποίο θα περιλαμβάνει τα διαστήματα 2 και 3, αλλά όχι το 1. Άρα, όταν φτάσουμε στο διάστημα 1 θα προσθέσουμε αναγκαστικά ένα ακόμα κέντρο κύκλου και επομένως ή λύση θα είναι λιγότερο βέλτιστη από αυτή του MED πίνακα. Άρα καταλήγουμε στο ότι OPT > MED > GREEDY και άρα ο παραπάνω αλγόριθμος με ταξινομημένα διαστήματα ως προς την αρχή τους βρίσκει το ελάχιστο πλήθος σημείων / κέντρα κύκλων που πρέπει να τοποθετηθούν.

Άσκηση 2

1. Το συγκεκριμένο πρόβλημα μπορεί να λυθεί με μία greedy προσέγγιση, αρκεί να βρούμε το άπληστο κριτήριο. Παρατηρούμε, ότι για κάθε πελάτη δημιουργείται ο λόγος wi/pi ο οποίος αναπαριστά το πόσο πολύτιμη είναι κάθε μονάδα του χρόνου για κάποιο πελάτη i και όσο πιο μεγάλος αυτός ο λόγος τόσο και μεγαλύτερη αξία έχει να τον εξυπηρετήσουμε πρώτο. Τη συγκεκριμένη παρατήρηση μπορούμε εύκολα να την επαληθεύσουμε αν έχουμε μόνο δύο πελάτες, όπου αν συγκρίνουμε τις

περιπτώσεις (1) Cost1 = w1p1 + w2(p1+p2) και (2) Cost2 = w2p2 + w1(p1+p2) καταλήγουμε στο ότι Cost1 < Cost2 μόνο όταν w1/p1 > w2/p2. Καταλήγουμε λοιπόν, στο ότι κάθε φορά θα διαλέγουμε από όσους πελάτες έχουν μείνει αυτόν με τον μεγαλύτερο λόγο wi/pi.

Πολυπλοκότητα

Η τελική **πολυπλοκότητα** θα είναι **O(nlogn)** γιατί θα χρειαστεί να ταξινομήσουμε τους λόγους αυτούς των πελατών.

Απόδειξη ορθότητας

Από την παραπάνω παρατήρηση, αποδείξαμε ότι ισχύει : w1/p1 > w2/p2 ⇔ Cost1 < Cost2 ⇔ w1p1 + w2(p1+p2) < w2p2 + w1(p1+p2). Αν λοιπόν, έχουμε δύο οποιουσδήποτε πελάτες (έστω i και j) και τους βάλουμε σε μία τυχαία σειρά θα προκύψει το άθροισμα :

$$\text{Cost} = \ldots + \text{wi}(\ldots + \text{pi}) + \ldots + \text{wj}(\ldots + \text{pi} + \ldots + \text{pj} + \ldots)$$
$$= \ldots + \text{wi}(\ldots) + \text{wipi} + \ldots + \text{wj}(\ldots) + \text{wj}(\text{pi} + \text{pj})$$

Στο άθροισμα αυτό βλέπουμε ότι προκύπτουν οι όροι wipi και wj(pi + pj) και για αυτούς γνωρίζουμε ότι ο πρώτος θα είναι μικρότερος πάντα όταν wi/pi > wj/pj. Άρα, με επιχείρημα ανταλλαγής οδηγούμαστε στο άπληστο κριτήριο το οποίο μας λέει ότι για κάθε ζευγάρι πελατών τοποθετούμε πρώτο αυτόν με το μεγαλύτερο λόγο wi/pi και το οποίο είναι ταυτόσημο με το να τους κοιτάξουμε σε φθίνουσα σειρά κατά wi/pi.

2. Για τη περίπτωση που έχουμε δύο πωλητές η greedy προσέγγιση δεν θα μας δώσει σωστό αποτέλεσμα και επομένως θα πρέπει να βασιστούμε σε κάποιο αλγόριθμο dp. Η γενική σκέψη είναι ότι για κάθε πελάτη μέσω μίας αναδρομικής σχέσης θα παίρνουμε το ελάχιστο κάθε φορά από το να τον τοποθετήσουμε στο πωλητή 1 ή στο πωλητή 2. Η σειρά, επίσης, με την οποία θα διαλέγουμε τους πελάτες θα είναι και πάλι φθίνουσα ως προς το λόγο wi/pi. Η αναδρομική σχέση αυτή, που προκύπτει, είναι η παρακάτω :

$$T(i,P) = min \begin{cases} T(i+1, P+p_i) + w_i(P+p_i) \\ T(i+1,P) + w_i(\sum_{j \leqslant i, p_j \setminus P} p_j) \end{cases}$$

όπου i είναι οι i-οστοί πρώτοι πελάτες ταξινομημένοι και P το άθροισμα των χρόνων pi που θα περιμένουν οι πελάτες στο πρώτο πωλητή. Χωρίς memoization η συγκεκριμένη υλοποίηση θα είχε εκθετικό χρόνο υλοποίησης, γι αυτό έχουμε στη μνήμη και έναν πίνακα μεγέθους nxP, όπου P το άθροισμα όλων των pi, στον οποίο αποθηκεύουμε τα αποτελέσματα της αναδρομικής σχέσης.

Πολυπλοκότητα

Η τελική πολυπλοκότητα θα είναι $O(n * P)$, όπου P το άθροισμα των pi.

Για περισσότερους από 2 υπαλλήλους, στην αναδρομική σχέση θα πρέπει να κρατάμε το άθροισμα των pi για κάθε πωλητή εκτός από τον τελευταίο (του οποίου το άθροισμα εννοείται). Επομένως για m πωλητές ο πίνακας T θα είναι της μορφής $T(i, P_1, \ldots, P_{m-1})$ και επομένως η υπολογιστική πολυπλοκότητα του αλγορίθμου θα αυξάνεται εκθετικά ανάλογα με το πλήθος των πωλητών.

Πολυπλοκότητα

Η τελική πολυπλοκότητα θα είναι $O(n * P^{m-1})$, όπου P το άθροισμα των pi και m το πλήθος των πωλητών.

Άσκηση 3

(α) Το συγκεκριμένο πρόβλημα θα το αντιμετωπίσουμε με μέθοδο επίλυσης δυναμικού προγραμματισμού (dp). Παρατηρούμε, αρχικά, ότι δεν υπάρχει νόημα να τοποθετήσουμε στέγαστρο το οποίο να αρχίζει ή να τελειώνει σε κάποιο σημείο εκτός των xi, καθώς αυτό θα αύξανε το συνολικό κόστος χωρίς λόγο. Σχηματίζουμε λοιπόν μία αναδρομική σχέση, της οποίας η γενική ιδέα είναι να ελέγξουμε για κάθε θέση i αν συμφέρει για κάθε προηγούμενη θέση j να φτιάξουμε ένα ενιαίο στέγαστρο που να εκτείνεται από τη θέση j μέχρι την i και για να είναι αποδοτικός ο αλγόριθμος θα διατηρούμε ένα πίνακα C στον οποίο θα είναι αποθηκευμένες οι βέλτιστες τιμές για κάθε σημείο i. Τέλος, πό αρχή βελτιστότητας, γνωρίζουμε ότι ο αλγόριθμος θα επιστρέφει πάντα τη βέλτιστη λύση.

Αναδρομική σχέση

$$C(i) = \begin{cases} 0, & i = -1 \\ \min\limits_{0 \le j \le i} \{C(j-1) + (x_i - x_j)^2 + c\}, & i \in [0, N-1] \end{cases}$$

Πολυπλοκότητα

Η τελική πολυπλοκότητα θα είναι $\Theta(n^2)$, γιατί για κάθε σημείο i θα κάνουμε n ελέγχους προς τα πίσω.

(β) Το συγκεκριμένο πρόβλημα ονομάζεται και convex hull problem \ πρόβλημα κυρτού περιβλήματος. Για την εύρεση γραμμικού αλγορίθμου για την επίλυση του προβλήματος αυτού θα εκμεταλλευτούμε ότι οι κλήσεις (ai) των ευθειών είναι ήδη ταξινομημένες και έτσι θα μπορούμε σε O(n+k) να αντιστοιχίσουμε μία ευθεία για κάθε ένα xi από τα k σημεία, η οποία θα είναι αυτή που θα επιστρέφει και το ελάχιστο yi. Για αρχή, θα χρειαστεί να κρατάμε κάποιου είδους λίστα στην οποία θα αποθηκεύουμε ποιές από τις ευθείες θα περιέχονται στο κυρτό περίβλημα.

Αλγόριθμος

Ξεκινώντας από την ευθεία με τη μεγαλύτερη κλίση (ai) την οποία δεν έχουμε διαλέξει ακόμα, βρίσκουμε τα σημεία τομής αυτής στον άξονα y με τη τελευταία (y1) και τη προτελευταία (y2) ευθεία που έχουμε αποθηκευμένη στη λίστα με τις ευθείες (αν στη λίστα δεν έχουμε πάνω από 1 ευθεία τότε προσθέτουμε στο τέλος της λίστας την νέα ευθεία). Συγκρίνουμε τα y1 και y2 και αν y1 < y2 τότε τοποθετούμε τη νέα ευθεία στο τέλος της λίστας μας. Διαφορετικά, αφαιρούμε τη τελευταία ευθεία από τη λίστα και εφαρμόζουμε τον ίδιο αλγόριθμο για τις επόμενες δύο ευθείες. Η υπολογιστική πολυπλοκότητα της κατασκευής αυτής της λίστας (του περιβλήματος) θα είναι O(n) και θα είναι amortized, δηλαδή κάθε ευθεία δεν θα εντάσεται στη λίστα σε O(1), αλλά επειδή κάθε μία θα συμμετέχει σε 2 το πολύ συγκρίσεις ο συνολικός χρόνος θα είναι γραμμικός.

Για τα k σημεία, τώρα, ξεκινώντας από το πρώτο σημείο θα ελέγχουμε σε ποιά ευθεία που αποθηκεύσαμε στη τελική λίστα αντιστοιχεί (δηλαδή επιστρέφει ελάχιστη τιμή) συγκρίνοντας το yi της πρώτης ευθείας με της δεύτερης. Ωστόσο, πειδή και τα σημεία αυτά είναι ταξινομημένα κατά αύξουσα σειρά, δεν θα χρειάζεται για κάθε σημείο να ξεκινάμε από την αρχή της λίστας για να ελέγχουμε σε ποιά ευθεία αντιστοιχεί, αλλά θα ξεκινάμε από εκείνη στην οποία αντιστοιχήθηκε το προηγούμενο σημείο. Έτσι, σε O(k) μπορούμε να βρούμε τους yi δείκτες, με τη τελική πολυπλοκότητα του αλγορίθμου να είναι O(n+k).

Για να χρησιμοποιήσουμε τη λύση του παραπάνω προβλήματος για τη βελτίωση της πολυπλοκότητας του (α), αρκεί να παρατηρήσουμε ότι η αναδρομική σχέση που χρησιμοποιήσαμε νωρίτερα μπορεί να μετασχηματιστεί ώς εξής :

$$C(i) = x_i^2 + c + \min_{0 \le j \le i}\{C(j-1) + x_j^2 - 2x_i x_j\}$$
$$a[j] = -2x_j$$
$$b[j] = C(j-1) + x_j^2$$

Χάρη σε αυτό το μετασχηματισμό, σε κάθε βήμα παρατηρούμε ότι το $x_i^2 + c$ υπολογίζεται σε σταθερό χρόνο, και για τον υπολογισμό της παρένθεσης του min ανάγουμε το συγκεκριμένο πρόβλημα σε αυτό του convex hull όπου έχουμε n ευθείες και n σημεία $\Theta(2n) \equiv \Theta(n)$. Επομένως η τελική υπολογιστική πολυπλοκότητα θα είναι γραμμική $\Theta(n)$.

Άσκηση 4

(α) 1ο σκέλος : Το συγκεκριμένο πρόβλημα είναι αντίστοιχο με το βέλτιστο διαχωρισμό n διατεταγμένων στοιχείων με k-1 φράγματα. Θα προσεγγίσουμε τη λύση του προβλήματος αυτού με dp, φτιάχνοντας μία αναδρομική σχέση η οποία ξεκινώντας από το λεωφορείο k θα προσδιορίζει το q_k, δηλαδή τη θέση του φράγματος k-1, και αναδρομικά θα υπολογίζει τη θέση των υπόλοιπων k-2 φραγμάτων, χάρη στην αρχή της βελτιστότητας. Η αναδρομική σχέση αυτή είναι η παρακάτω:

Αναδρομική σχέση

$$S(n,k) = min_{i=1}^{n}\{S(i,k-1) + \sum_{j>i,k>i,j<k} A_{jk}\}$$
$$S(1,k) = A_1$$
$$S(n,1) = \sum_{i-1}^{n} A_i$$

Για να μην επαναλαμβάνουμε καταστάσεις που συναντάμε στο recursion tree, με τη τεχνική του memoization διατηρούμε στη μνήμη ένα πίνακα S μεγέθους nxk ο οποίος αποθηκεύει σε κάθε θέση το βέλτιστο δείκτη ευαισθησίας για n μαθητές και k λεωφορεία. Ακόμη, για να μην υπολογίζουμε πολλαπλές φορές τα αθροίσματα μεταξύ θέσεων i και j του δείκτη ευαισθησίας A_{ij} μπορούμε να τα υπολογίσουμε μία φορά στην αρχή και να τα αποθηκεύσουμε σε ένα διδιάστατο πίνακα μεγέθους nxn

Πολυπλοκότητα

Οι βαθμοί ελευθερίας της αναδρομικής συνάρτησης είναι n και k, άρα για να γεμίσουμε το πίνακα S θα χρειαστούμε σίγουρα O(n*k). Όμως, για κάθε κλήση της αναδρομής εξετάζουμε όλες τις πιθανές θέσεις για το τελευταίο φράγμα, οι οποίες στη γενική περίπτωση είναι n, άρα η τελική πολυπλοκότητα θα είναι $O(n^2k)$.

2ο σκέλος : Στη περίπτωση που το ζητούμενο δεν είναι η ελαχιστοποίηση του συνολικού δείκτη ευαισθησίας, αλλά του μέγιστου δείκτη ευαισθησίας των λεωφορείων, θα χρειαστεί να τροποποιήσουμε την αναδρομική μας σχέση ώστε να μην επιστρέφει κάποιο άθροισμα επιμέρους αθροισμάτων, αλλά το

μέγιστο των αθροισμάτων A_{ij} κάθε λεωφορείου. Και πάλι θα χρειαστεί να διατηρούμε ένα πίνακα S μεγέθους nxk. Επομένως, η προκύπτει η παρακάτω σχέση.

Αναδρομική σχέση

$$S(n,k) = min\{max_{i=1}^{n}\{S(i,k-1), \sum_{j>i,k>i,j<k} A_{jk}\}\}$$
$$S(1,k) = A_1$$
$$S(n,1) = \sum_{i-1}^{n} A_i$$

Πολυπλοκότητα

Η πολυπλοκότητα δεδομένου ότι έχουμε το πίνακα A_{ij} ήδη έτοιμο και λόγω memoization είναι και πάλι $O(n^2k)$.

2ος τρόπος : Εκτός από την επίλυση του προβλήματος αυτού με dp, δηλαδή μέσω μίας αναδρομικής σχέσης, θα μπορούσαμε να το λύσουμε και μέσω μίας υλοποίησης binary search στο άθροισμα όλων των επιμέρους γινομένων A_{ij}. Με άκρα δηλαδή τα 1 και $\sum_{i=0,j=0}^{n} A_{ij}$, για κάθε τιμή του binary search θα γεμίζουμε κάθε λεωφορείο μέχρι το επιμέρους άθροισμα γινομένων μόλις να μην ξεπερνά τη τιμή αυτή.

Πολυπλοκότητα

Αν S είναι το συνολικό άθροισμα όλων των επιμέρους γινομένων A_{ij} τότε η τελική πολυπλοκότητα είναι $O(nlogS + n^2)$. Το n^2 προκύπτει γιατί χρειαζόμαστε τουλάχιστον τετραγωνικό χρόνο για να υπολογίσουμε το S.

Άσκηση 5

α) Σκοπός μας είναι να αφαιρέσουμε την ακμή e από το T1 και να την αντικαταστήσουμε με μία e' διατηρώντας τη συνεκτικότητα του δέντρου. Αρχικά, έχοντας βγάλει την ακμή e από το δέντρο T1, θα εκτελέσουμε τον αλγόριθμο του union find, ώστε να διαχωρίσουμε σε γραμμικό χρόνο τις δύο συνεκτικές συνιστώσες που έμειναν. Έπειτα, για κάθε ακμή e = (u,v) του T2 , θα συγκρίνουμε τις τιμές find(u) και find(v) και αν αυτές είναι διαφορετικές τότε η ακμή αυτή είναι μία από αυτές που ψάχνουμε (e') γιατί αυτή η ακμή θα συνδέει τις δύο συνιστώσες που προέκυψαν από την αφαίρεση της e.

β) 1ο σκέλος : Αν έχουμε δύο διαφορετικά μεταξύ τους συνδετικά δέντρα T1 και T2 με απόσταση N μεταξύ τους τότε από το ερώτημα α) έχουμε αποδείξει ότι για μία ακμή e του T1 μπορούμε να βρούμε μία e' του T2. Έτσι, αν αντικαταστήσουμε πχ την ακμή e με την e' μειώνουμε την απόσταση των δύο δέντρων κατά ένα και πλέον θα έχουν απόσταση N-1. Επαγωγικά αποδεικνύεται, λοιπόν, ότι μπορούμε να μειώσουμε την απόσταση από N σε 0 κάνοντας ένα βήμα κάθε φορά, και επομένως στο γράφο H αυτό αντιστοιχεί σε ένα μονοπάτι μεταξύ δύο κόμβων με μήκος μονοπατιού ίσο με N. Επίσης, όταν δεν

θα υπάρχει άλλο ζεύγος ακμών e και e' τότε όλες οι ακμές των δύο δέντρων θα ταυτίζονται και άρα τα βήματα N του αλγορίθμου θα ισούνται με την απόσταση των T1 και T2 (N = |T1\T2|).

(Εναλλακτική σκέψη) Με άτοπο θα υποθέσουμε ότι το H δεν είναι συνδετικό. Τότε θα πρέπει να ισχύει ότι για δύο διαφορετικά συνδετικά δέντρα δεν μπορούμε να μεταβούμε από το ένα στο άλλο με ανταλλαγή ακμών. Αν έχουμε δύο δάση τα οποία μπορούν να ενωθούν με δύο διαφορετικές ακμές e και e' αντίστοιχα τότε θεωρούμε ότι θα φτιαχτούν δύο συνδετικά δέντρα όπου το ένα θα περιέχει την ακμή e και το άλλο την ακμή e'. Για να μην μπορούμε να μεταβούμε από το ένα δέντρο στο άλλο, τότε θα έπρεπε να μην μπορούμε να κάνουμε ανταλλαγή των ακμών αυτών. Για να γίνει όμως αυτό θα έπρεπε οι ακμές να αποτελούν γέφυρες του αρχικού γράφου G το οποίο είναι άτοπο γιατί τότε τα δύο δάση θα ενώνονταν με μοναδική ακμή (τη γέφυρα του γράφου) και άρα οι ακμές e και e' θα ταυτίζονταν.

2ο σκελος : Εφαρμόζουμε τον αλγόριθμο του α) κάθε φορά ξεκινώντας από το T1 και αντικαθιστούμε την ακμή e του T1 με την e' ώστε να οδηγηθούμε πιο "κοντά" στο T2. Ώστε να αποφύγουμε να κάνουμε κύκλους αποκλείουμε στα επόμενα βήματα όλα τα δέντρα που περιέχουν την ακμή e (ουσιαστικά αφαιρούμε την e από τον γράφο G). Επίσης, λόγω του ότι κάθε δύο κόμβοι απέχουν απόσταση 1 μεταξύ τους (δηλαδή βάρος 1 στο γράφο H) τότε κάθε μονοπάτι μεταξύ των T1, T2 θα είναι συντομότερο στο συγκεκριμένο αλγόριθμο, καθώς αποφεύγουμε τους κύκλους.

γ) Στόχος είναι να υπολογίσουμε το άθροισμα των βαρών των ακμών του ελάχιστου συνδετικού δέντρου που περιέχει την ακμή e, όπου e κάθε ακμή του αρχικού γράφου G. Αρχικά, μπορούμε μέσω του αλγόριθμου του prim η του kruskal (O(mlogm)) να υπολογίσουμε το ελάχιστο συνδετικό δέντρο όλου του γραφήματος (MST). Για όσες ακμές περιέχονται σε αυτό το δέντρο, ο αλγόριθμος θα επιστρέφει το βάρος του δέντρου αυτού. Τώρα, για τις υπόλοιπες ακμές (έστω e η κάθε τέτοια ακμή), επειδή το δέντρο που ψάχνουμε απέχει απόσταση μόνο 1 από το MST αρκεί να βρούμε μόνο μία ακμή e' την οποία θα αντικαταστήσουμε με την e. Αυτή η ακμή e = (u,v) θα πρέπει να αντικαθιστά κάποια άλλη η οποία βρίσκεται στο μονοπάτι που συνδέει τις κορυφές u και v, ώστε αυτές να παραμείνουν συνδεδεμένες. Άρα, θα τρέξουμε ένα dfs για να βρούμε το μονοπάτι από το u στο v και θα αφαιρέσουμε από το δέντρο την ακμή με το μεγαλύτερο βάρος, ώστε να προκύψει δέντρο με ελάχιστο βάρος. Για να μην τρέξουμε m φορές το dfs, μπορούμε από πριν να έχουμε υπολογίσει το βάρος της μέγιστης ακμής στο MST από κάθε ακμή πρός κάθε άλλη (O(n^2)). Μία τέτοια υλοποίηση θα είχε συνολική υπολογιστική πολυπλοκότητα $O(mlogm + n^2)$.