Αλγόριθμοι και Πολυπλοκότητα

2η Σειρά Γραπτών Ασκήσεων

7ο Εξάμηνο, Ακαδημαϊκό Έτος 2021 – 2022

Ονοματεπώνυμο	Αριθμός Μητρώου
Στεφανάκης Γεώργιος	el18436

Άσκηση 1η – Δίσκοι και Σημεία

Λόγω της γεωμετρίας είναι προφανές πως ένας κύκλος ακτίνας r με κέντρο επάνω στην ευθεία l μπορεί να καλύψει σημεία που βρίσκονται σε απόσταση το πολύ σε απόσταση r μακριά από την ευθεία. Εάν τώρα για κάποιο σημείο εκτός ευθείας A εμείς φέρουμε τα ευθύγραμμα τμήματα AB και AC, με B, C να τέμνουν την l, τέτοια ώστε $|AB| = |AC| = r$, καταλαβαίνουμε πως οποιοσδήποτε κύκλος εκτός του τμήματος BC, έστω (D, r) δεν καλύπτει το σημείο A αφού $|DA| > |AB| = |AC| = r$. Συνεπώς είναι αναγκαία και ικανή συνθήκη ότι για να καλύπτεται ένα σημείο πρέπει να υπάρχει δίσκος ανάμεσα στα σημεία B και C.

Το πρόβλημά μας λοιπόν ανάγεται στην εύρεση του ελάχιστου πλήθους σημείων (κέντρα κύκλων ακτίνας r) τέτοια ώστε να υπάρχει τουλάχιστον ένα σημείο μέσα σε κάθε διάστημα που ορίζουν τα ευθύγραμμα τμήματα που περιγράφθηκαν παραπάνω. Έχουμε n σημεία που το καθένα ορίζει ένα διάστημα στην ευθεία l, συνεπώς κατασκευάζουμε n διαστήματα τα οποία ταξινομούμε αρχικά ως προς το δεξί τους άκρο κατά αύξουσα σειρά. Αρχικοποιούμε μία λίστα με τις θέσεις των ζητούμενων σημείων σε κενή και διατρέχοντας τα διαστήματα εφαρμόζουμε το εξής άπληστο κριτήριο:

- Αν το τελευταίο σημείο που προσθέσαμε είναι δεξιά του αριστερού άκρου του διαστήματος τότε προχωράμε καθώς το τελευταίο σημείο που προσθέσαμε βρίσκεται εντός του διαστήματος και συνεπώς ο δίσκος του καλύπτει το αρχικό σημείο.
- Αλλιώς, προσθέτουμε έναν δίσκο με κέντρο το δεξί άκρο του τρέχοντος διαστήματος.

Στο τέλος της επανάληψης θα έχουμε στη λίστα των ζητούμενων σημείων τις θέσεις στις οποίες πρέπει να τοποθετήσουμε τα κέντρα των κύκλων ακτίνας r. Η πολυπλοκότητα είναι $O(n \log n)$ λόγω της ταξινόμησης των διαστημάτων.

Απόδειξη ορθότητας:

Έστω $(s_1, f_1), (s_2, f_2), \dots, (s_n, f_n)$ τα διαστήματα με $f_1 \leq f_2 \leq \cdots \leq f_n$, k το πλήθος των σημείων που επιστρέφει ο άπληστος αλγόριθμος που περιγράφθηκε και m το πλήθος των σημείων που βρίσκει ο βέλτιστος αλγόριθμος.

Μπορούμε με σιγουριά να ισχυριστούμε ότι για όλα τα διαστήματα, τα σημεία που κατασκευάζει ο βέλτιστος αλγόριθμος, θα είναι πάντα μικρότερα ή ίσα από εκείνα που κατασκευάζει ο άπληστος αλγόριθμος. Αυτό ισχύει διότι ο άπληστος τοποθετεί όταν χρειάζεται ένα σημείο στο τέλος του τρέχοντος διαστήματος για να καταφέρει να το καλύψει. Επιπλέον αφού γνωρίζουμε ότι εκτός των ορίων του διαστήματος κανένας κύκλος δεν καλύπτει το διάστημα αυτό, αναγκαστικά και ο βέλτιστος αν χρειαστεί θα τοποθετήσει ένα κέντρο που θα βρίσκεται μέσα στο τρέχον διάστημα και άρα αυτό θα είναι μικρότερο ή ίσο από το κέντρο του δίσκου που πρόσθεσε ο άπληστος. Δηλαδή, για κάθε $i \leq m$, το i-οστό σημείο του άπληστου είναι μεγαλύτερο ή ίσο από το i-οστό σημείο του βέλτιστου.

Έστω τώρα ότι $k > m$, δηλαδή ότι ο άπληστος βρίσκει περισσότερα σημεία από τον βέλτιστο. Αν ο άπληστος χρειάζεται να προσθέσει κάποιο $m + 1$ σημείο σημαίνει ότι υπάρχει κάποιο διάστημα $j \in [k + 1, n]$ τέτοιο ώστε το s_j να είναι μεγαλύτερο του τελευταίου σημείου. Όμως ισχύει ότι το τελευταίο σημείου του βέλτιστου είναι αριστερότερα του m-

οστού σημείου του άπληστου, και άρα και του s_j. Άρα για το διάστημα j ο βέλτιστος χρειάζεται κι άλλο σημείο πράγμα που σημαίνει ότι ο βέλτιστος παράγει $m+1$ σημεία, δηλαδή άτοπο.

Άρα τελικά $k \leq m$ και αφού ο βέλτιστος φτιάχνει m σημεία, ο άπληστος αλγόριθμος είναι βέλτιστος και κατασκευάζει και εκείνος m σημεία.

Άσκηση 2η – Παραλαβή Πακέτων

α) Ένα άπληστο κριτήριο το οποίο μας εξασφαλίζει ότι θα έχουμε το ελάχιστο δυνατό βεβαρυμένο χρόνο εξυπηρέτησης είναι το να επιλέγουμε σε κάθε βήμα τον πελάτη με τη μεγαλύτερη αναλογία w/p. Πρέπει λοιπόν να κατασκευάσουμε μία λίστα που περιέχει στοιχεία της μορφής $\left(\frac{w_i}{p_i}, i\right)$, $\forall i = 1, \dots, n$. Ύστερα να ταξινομήσουμε τη λίστα σε φθίνουσα σειρά ως προς τις αναλογίες. Το αποτέλεσμα είναι η σειρά που εμφανίζονται οι δείκτες i στην ταξινομημένη λίστα.

Όσον αφορά την απόδειξη ορθότητας του κριτηρίου θα χρησιμοποιήσουμε επιχείρημα ανταλλαγής.

Μπορούμε να θεωρήσουμε ορθά ότι για ένα ζεύγος διαδοχικών στοιχείων i, j στη βέλτιστη δρομολόγηση, η εναλλαγή των θέσεών τους δεν θα επηρεάσει αρνητικά τον χρόνο εξυπηρέτησης, ήτοι ο νέος χρόνος εξυπηρέτησης που θα προκύψει θα είναι πάντα μικρότερος ή ίσος του προηγούμενου. Έτσι θα «πλησιάσουμε» τον optimal στον greedy εφαρμόζοντας όσα swaps χρειαστούν. Για i, j με $\frac{w_i}{p_i} \geq \frac{w_j}{p_j}$ έχουμε:

- Χρόνος πριν την εναλλαγή των i, j: $t = \Sigma_1 + w_j(T + p_j) + w_i(T + p_j + p_i) + \Sigma_2$ (αφού ο βέλτιστος δεν ακολουθεί τη δική μας δρομολόγηση τα i και j θα είναι αντεστραμμένα)
- Χρόνος μετά την εναλλαγή των i, j: $t' = \Sigma_1 + w_i(T + p_i) + w_j(T + p_i + p_j) + \Sigma_2$

$$t - t' = w_j p_i - w_i p_j \geq 0$$

Λόγω της παραπάνω ανίσωσης. Συνεπώς, μπορούμε να κάνουμε όσες εναλλαγές θέσεων θέλουμε χωρίς να επηρεάσουμε αρνητικά τον τελικό αποτέλεσμα. Έτσι ξεκινώντας από τη δρομολόγηση του βέλτιστου μπορούμε χωρίς βλάβη να καταλήξουμε στη δρομολόγηση που φτιάχνει ο άπληστος αλγόριθμός μας και προφανώς, λόγω της παραπάνω απόδειξης και του ότι ο greedy δεν μπορεί να βρει καλύτερη λύση από τον βέλτιστο, ο άπληστος αλγόριθμος είναι και εκείνος βέλτιστος.

β) Όπως και στο προηγούμενο ερώτημα, αρχικά πρέπει να ταξινομήσουμε τις διεργασίες κατά φθίνουσα σειρά ως προς το πηλίκο $\frac{w_i}{p_i}$ καθώς η σειρά επιλογής της επόμενης διεργασίας για κάθε υπάλληλο πρέπει να ακολουθεί αυτή την ταξινόμηση. Σε κάθε βήμα πρέπει να επιλέξουμε εάν την επόμενη διεργασία θα την δρομολογήσουμε στο πρώτο ή στο δεύτερο υπάλληλο. Οι διεργασίες που θα έχουν αναλάβει μέχρι πριν την δρομολόγηση της i-οστής διεργασίας διαμοιράζονται στους δύο υπαλλήλους ως εξής: Ο πρώτος υπάλληλος έχει αναλάβει διεργασίες που κοστίζουν P ενώ ο δεύτερος έχει αναλάβει διεργασίες που κοστίζουν όσο το συνολικό άθροισμα των κοστών διεργασιών πλην το άθροισμα των διεργασιών του πρώτου, P. Άρα το state space αποτελείται από τις δύο μεταβλητές $(i,\ P)$ με i τις πρώτες διεργασίες που δρομολογούνται (σύμφωνα με την αναφερθείσα ταξινόμηση) και P το άθροισμα των κοστών που έχει αναλάβει ο πρώτος υπάλληλος. Αφού Η λύση δυναμικού προγραμματισμού, λοιπόν, έχει την εξής μορφή:

$$\mathrm{dp}(i, P) = \min \begin{cases} \mathrm{dp}(i-1, P - p_i) + w_i P \\ \mathrm{dp}(i-1, P) + w_i \left(\sum_{j=1}^{i} p_j - P \right) \end{cases}$$

Επιπλέον, αρχικοποιούμε τα $\mathrm{dp}(i, j) = +\infty$ ώστε να γεμίσουμε σωστά τον πίνακα N γραμμών και $\sum_{j=1}^{N} p_j$ στηλών του δυναμικού προγραμματισμού. Επίσης, πρέπει για j αρνητικό να ισχύει επιπλέον $\mathrm{dp}(i, j) = +\infty$.

Ο τρόπος με τον οποίο θα γεμίσουμε το πινακάκι φαίνεται στις δύο περιπτώσεις που προκύπτουν:

- Φτάσαμε να έχουμε δρομολογήσει τις i πρώτες διεργασίες με άθροισμα P στον 1ο υπάλληλο αφού οι $i-1$ πρώτες διεργασίες είχαν ανατεθεί επίσης στον πρώτο και στη συνέχεια ανατέθηκε σε αυτόν και η i-οστή προσθέτοντας στο συνολικό κόστος $w_i P$
- Φτάσαμε να έχουμε δρομολογήσει κάποιες από τις $i-1$ πρώτες διεργασίες στον πρώτο υπάλληλο, ο οποίος έχει συνολικό κόστος μέχρι στιγμής ίσο με P, και όταν φτάσαμε στην i-οστή διεργασία την αναθέσαμε στον δεύτερο υπάλληλο, ο οποίο πλέον έχει κόστος ίσο με $\sum_{j=1}^{i}(p_j - P)$ και επομένως στο συνολικό βεβαρυμένο χρόνο του προστίθεται $w_i \sum_{j=1}^{i}(p_j - P)$

Για αρχή θα πάρουμε την διεργασία υπ' αριθμόν 1, εκείνη δηλαδή με το μεγαλύτερο πηλίκο $\frac{w_i}{p_i}$, και θα γεμίσουμε τα κουτάκια $\mathrm{dp}(1, p_1) = w_1 p_1$ και $\mathrm{dp}(1,0) = w_1 p_1$. Στη συνέχεια, για i από 2 μέχρι N, γεμίζουμε το υπόλοιπο πινακάκι και όταν ολοκληρώσουμε και τη γραμμή N, θα έχουμε φροντίσει να έχουμε αποθηκευμένο και το κελί με συντεταγμένες $\mathrm{dp}(N, j)$ με την ελάχιστη τιμή. Έτσι, κοιτώντας από που ήρθαμε, μπορούμε να βρούμε τις αναθέσεις κάθε υπαλλήλου.

Για όλες τις υπόλοιπες γραμμές, καθώς επαναλαμβάνουμε τη διαδικασία, θα ελέγχουμε εάν μπορούμε να φτάσουμε σε αυτό το κελί από κάποιο κελί της προηγούμενης γραμμής το οποίο δεν έχει την τιμή $+\infty$. Αν δεν υπάρχει τρόπος να φτάσουμε στο κελί αυτό τότε θα πάρει τιμή $+\infty$, πράγμα που σημαίνει ότι δεν είναι εφικτό να φτάσουμε εκεί.

Όσον αφορά την χρονική πολυπλοκότητα, κάθε φορά το άθροισμα $\sum_{j=1}^{i} p_j$ μπορούμε να το υπολογίζουμε εύκολα σε $\mathrm{O}(1)$ εάν αρχικά υπολογίσουμε τον prefix sum πίνακα των p. Οπότε, λόγω του state space προκύπτει η συνολική πολυπλοκότητα $\mathrm{O}(N\,P)$.

Για $m \geq 3$ υπαλλήλους μπορούμε να ακολουθήσουμε παρόμοια λογική με τη διαφορά ότι θα έχουμε $m-1$ επιπλέον διαστάσεις που θα αναπαριστούν το άθροισμα των κοστών των διεργασιών που θα έχουν αναλάβει και οι υπόλοιποι $m-1$ υπάλληλοι. Έτσι η dp σχέση μας θα έχει m διαστάσεις, ενώ τον πίνακα του dp θα τον κατασκευάσουμε με αντίστοιχο τρόπο. Η συνολική πολυπλοκότητα στη γενική περίπτωση θα είναι $\mathrm{O}(N\,P^{m-1})$.

Άσκηση 3η – Τοποθέτηση Στεγάστρων (και Κυρτό Κάλυμμα)

α) Μια λύση δυναμικού προγραμματισμού για το πρόβλημα μπορεί να είναι η εξής: Αν κάποιο στέγαστρο ξεκινάει από ένα σημείο x_i και τελειώνει σε κάποιο σημείο x_j, τότε μπορούμε ξεκινώντας από το τελικό σημείο x_f να βρούμε κάποιο σημείο x_i που αν χρησιμοποιηθεί ως αρχή της τελευταίας στέγης θα δώσει το ελάχιστο κόστος.

$$\mathrm{cost}(i) = \begin{cases} 0, & i = 1 \\ \min\limits_{0 \leq j \leq i} \left(\mathrm{cost}(j-1) + (x_i - x_j)^2 + C\right), & i \neq 1 \end{cases}$$

Η συνάρτηση cost υπολογίζει το ελάχιστο κόστος για να καλυφθούν όλα τα σημεία ξεκινώντας από το i, και βρίσκει το σημείο j όπου τελειώνει αυτό το στέγαστρο. Θα πάρουμε την απάντηση που θέλουμε καλώντας το $\mathrm{cost}(x_f)$. Η χρονική πολυπλοκότητα είναι $\mathrm{O}(N^2)$ καθώς για κάθε ένα από τα N σημεία πρέπει να υπολογίσουμε την ελάχιστη τιμή μέχρι κάποιο σημείο j, κάτι που στη γενική περίπτωση απαιτεί γραμμικό χρόνο.

Άσκηση 4η – Δρομολόγια Λεωφορείων στην Εποχή του Κορωνοϊού

Για τη λύση του προβλήματος με δυναμικό προγραμματισμό, ας θεωρήσουμε $\mathrm{dp}(i, j)$ τον ελάχιστο συνολικό βαθμό ευαισθησίας των λεωφορείων στην κατάσταση όπου έχουμε μοιράσει τους j πρώτους φοιτητές σε i λεωφορεία. Ξεκινώντας από την κλήση του $\mathrm{dp}(K, N)$, θα βρίσκουμε σε κάθε βήμα την ελάχιστη τιμή των λύσεων διαμοιρασμού x

φοιτητών στα $i-1$ πρώτα λεωφορεία, συν το άθροισμα των βαθμών ευαισθησίας των υπολοίπων $j-x$ φοιτητών που θα μπουν στο i-οστό λεωφορείο. Η σχέση δυναμικού προγραμματισμού είναι η εξής:

$$\mathrm{dp}(i,j) = \min_{1 \le x < j} \left(\mathrm{dp}(i-1,x) + \sum_{a=x+1}^{j} \sum_{b=a+1}^{j} A_{ab} \right)$$

Το state space έχει μέγεθος $N \times K$ και συνεπώς και το πινακάκι του dp. Όσον αφορά την πολυπλοκότητα, για κάθε υποπρόβλημα πρέπει να βρούμε την ελάχιστη τιμή μεταξύ N δυνατών επιλογών και επιπλέον, για κάθε μία από αυτές τις επιλογές πρέπει να υπολογίσουμε το διπλό άθροισμα της παραπάνω σχέση. Αυτό στη γενική περίπτωση κοστίζει $\mathrm{O}(N^2)$. Δηλαδή στην απλούστερη λύση, η συνολική πολυπλοκότητα είναι $O(K\,N^4)$. Με μία προεργασία στον πίνακα A μπορούμε να υπολογίσουμε για κάθε ζεύγος συντεταγμένων (x,y) το άθροισμα των στοιχείων του πίνακα που βρίσκονται το πολύ μέχρι την γραμμή x και το πολύ μέχρι την στήλη y και να τα ανακτούμε όποτε θα χρειαζόμαστε σε χρόνο $\mathrm{O}(1)$. Δηλαδή μπορούμε να υπολογίζουμε το διπλό άθροισμα σύμφωνα με τον εξής τύπο:

$$\mathrm{sums}_{xy} = \mathrm{sums}_{x-1\,y} + \mathrm{sums}_{x\,y-1} - \mathrm{sums}_{x-1\,y-1} + A_{xy}$$

Έτσι λοιπόν το διπλό άθροισμα υπολογίζεται ως εξής:

$$\sum_{a=x+1}^{j} \sum_{b=a+1}^{j} A_{ab} = \frac{1}{2}\left(\mathrm{sums}_{jj} - \mathrm{sums}_{jx} - \mathrm{sums}_{xj} + \mathrm{sums}_{xx}\right)$$

Και τελικά πετυχαίνουμε χρονική πολυπλοκότητα $\mathrm{O}(N^2\,K)$.

Για να βρούμε μία διάταξη φοιτητών σε λεωφορεία που ελαχιστοποιεί το μέγιστο βαθμό ευαισθησίας που εμφανίζεται σε κάποιο λεωφορείο, μπορούμε να κάνουμε δυαδική αναζήτηση στις τιμές του δείκτη ευαισθησίας. Το διάστημα της αναζήτησης έχει αριστερό άκρο το 0 και δεξί άκρο το συνολικό βαθμό ευαισθησίας εάν τοποθετήσουμε όλους τους φοιτητές σε ένα λεωφορείο, έστω ότι αυτός ο βαθμός είναι M. Θα κάνουμε αναζήτηση στο διάστημα $[0, M]$. Για κάθε μέσο του διαστήματος, θα τοποθετήσουμε έναν έναν τους πρώτους φοιτητές στο πρώτο λεωφορείο και όταν ο δείκτη ευαισθησίας ξεπεράσει τον μέσο, θα βγάλουμε τον τελευταίο και θα τον βάλουμε στο δεύτερο λεωφορείο κ.ο.κ. Εάν γεμίσουν όλα τα λεωφορεία χωρίς να έχουν εξαντληθεί οι φοιτητές επαναλαμβάνουμε για μεγαλύτερη τιμή δείκτη ευαισθησίας, ενώ αν η διάταξη είναι έγκυρη δοκιμάζουμε μικρότερες τιμές ευαισθησίας κρατώντας στο δεξί άκρο του διαστήματος τον επιτυχή μέσο για να βρούμε καλύτερη διάταξη.

Άσκηση 5η – Το Σύνολο των Συνδετικών Δέντρων

α) Προφανώς, εάν από το T_1 αφαιρέσουμε την ακμή $e = \{u, v\}$, δηλαδή έχουμε το δέντρο $T_1 \setminus \{e\}$, τότε το T_1 θα χωριστεί σε δύο συνεκτικές συνιστώσες. Με DFS μπορούμε να διατρέξουμε αυτές τις συνιστώσες και να σημειώσουμε τους κόμβους που ανήκουν σε κάθε μία από αυτές. Στην συνέχεια, με άλλο ένα DFS στο T_2 ξεκινώντας από τον κόμβο u μπορούμε να βρούμε το μονοπάτι που συνδέει τους κόμβους u και v στο T_2. Η ακμή e' που αναζητάμε βρίσκεται σε αυτό το μονοπάτι. Συνεπώς, διατρέχουμε αυτό το μονοπάτι και ψάχνουμε δύο κορυφές που η μία να ανήκει στην πρώτη συνεκτική συνιστώσα και η άλλη στη δεύτερη συνεκτική συνιστώσα. Η ακμή που ψάχνουμε θα είναι η e', η οποία ενώνει τα δύο συνεκτικά δέντρα αφού αφαιρέσαμε την ακμή e από το T_1. Συνολική πολυπλοκότητα $O(|V|)$ λόγω του DFS.

β) Χρησιμοποιώντας τον αλγόριθμο του πρώτου ερωτήματος μπορούμε να μεταβούμε από οποιοδήποτε συνδετικό δέντρο σε οποιοδήποτε άλλο συνδετικό δέντρο βγάζοντας μία ακμή e και προσθέτοντας μία άλλη ακμή e'. Εάν έχουμε τα συνδετικά δέντρα T_1, T_2 που ανήκουν στο H και η $d(T_1, T_2)$ η απόσταση τους στο γράφημα H. Μπορούμε να δείξουμε επαγωγικά ότι αν $d(T_1, T_2) = k$ τότε $|T_1 \setminus T_2| = k$.

- Για $k = 1$, λόγω του ορισμού του H προφανώς ισχύει η υπόθεση

- Έστω ότι ισχύει και για k, θα δείξουμε ότι ισχύει και για $k+1$
- Αφού $d(T_1, T_2) = k+1$ θα υπάρχει δέντρο T' που να ανήκει στο H και $d(T_1, T) = 1$, $d(T', T_2) = k$. Από επαγωγική υπόθεση ξέρουμε ότι $|T' \setminus T_2| = k$ και από τη βάση της επαγωγή ότι $|T_1 \setminus T'| = k-1$. Άρα είτε $|T_1 \setminus T_2| = k-1$ είτε $|T_1 \setminus T_2| = k+1$. Όμως, εάν $|T_1 \setminus T_2| = k-1$ τότε αναγκαστικά από την υπόθεση της επαγωγής ισχύει $d(T_1, T_2) = k-1$, που είναι άτοπο καθώς $d(T_1, T_2) = k+1$. Οπότε $|T_1 \setminus T_2| = k+1$.

Για να βρούμε τώρα το συντομότερο μονοπάτι μεταξύ των T_1 και T_2, θα βρούμε το σύνολο των ακμών του $T_2 \setminus T_1$. Έπειτα, κάθε ακμή στο $T_2 \setminus T_1$ θα την προσθέτουμε στο T_1 και θα σβήνουμε μία ακμή από το T_1 που ανήκει στο $T_1 \setminus T_2$ έτσι ώστε να μετατρέψουμε το T_1 στο T_2. Η διαδικασία αυτή θα χρειαστεί $|T_2 \setminus T_1|$ βήματα. Άρα στο H, η ελάχιστη απόσταση μεταξύ των δέντρων T_1 και T_2 είναι ίση με k.

Όσον αφορά την πολυπλοκότητα, εάν για να ελέγξουμε εάν μία ακμή του T_2 ανήκει και στο T_1 χρειαζόμαστε $O(1)$, και αφού πρέπει για κάθε κόμβο να εφαρμόσουμε k φορές τον αλγόριθμο του πρώτου ερωτήματος, αφού θα κάνουμε k ανανεώσεις, η συνολική πολυπλοκότητα είναι $O(k\,|V|)$.

γ) Αφού κατασκευάσουμε μία φορά το ελάχιστο συνδετικό δέντρο μπορούμε με βάση αυτό να βρίσκουμε κάθε φορά το ζητούμενο MST ελαχίστου βάρους για κάθε ακμή $e = \{u, v\}$. Μπορούμε να παρατηρήσουμε ότι εάν προσθέσουμε μία ακμή που δεν ανήκει στο MST σε αυτό και αφαιρέσουμε τη βαρύτερη ακμή στον βρόχο που θα δημιουργηθεί, βρίσκουμε το ζητούμενο δέντρο. Έτσι, αφού υπολογίσουμε το συνολικό βάρος w_{mst} ενός MST, πρέπει να μπορούμε για τις ακμές που δεν συμπεριλήφθηκαν σε αυτό το δέντρο να βρίσκουμε την ακμή με το μεγαλύτερο βάρος στον κύκλο που δημιουργούν και να προσθέσουμε τη διαφορά των βαρών τους στο συνολικό βάρος για να πάρουμε το ζητούμενο αποτέλεσμα. Με Kruskal, μπορούμε να βρούμε ένα MST σε χρόνο $O(m \log m)$ και στη συνέχεια για κάθε ακμή μεταξύ δύο κορυφών u, v που δεν ανήκουν στο MST, θα τρέξουμε ένα DFS από την πρώτη κορυφή στη δεύτερη για να βρούμε τη βαρύτερη ακμή στο μονοπάτι αυτό. Έτσι θα έχουμε πρόσβαση σε $O(1)$ στην πληροφορία που χρειαζόμασταν, δηλαδή τη βαρύτερη ακμή στον κύκλο που δημιουργείται, την οποία και πρέπει να αφαιρέσουμε από το συνολικό βάρος του MST. Άρα η συνολική πολυπλοκότητα ανέρχεται σε $O(n^2)$ λόγω του DFS που εκτελέσαμε κατά την προεπεξεργασία του προβλήματος.