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

Αλγόριθμοι και Πολυπλοκότητα

2η Σειρά Γραπτών Ασκήσεων – Χειμερινό Εξάμηνο 2021

Κονταλέξη Μαρίνα – el18022

Άσκηση 1:

Πριν προχωρήσουμε στα βήματα του αλγορίθμου, θα λύσουμε την ανίσωση

$$d\big((a,b),(x,0)\big) \le r$$

Οι λύσεις της οποίας είναι το διάστημα στο οποίο μπορούμε να τοποθετήσουμε κύκλο κέντρου $(x,0)$ για να καλύπτει σίγουρα το σημείο (a,b). Έχουμε:

$$d\big((a,b),(x,0)\big) \le r \Leftrightarrow$$

$$\sqrt{(a-x)^2+b^2} \le r \Leftrightarrow$$

$$(a-x)^2+b^2 \le r^2 \Leftrightarrow$$

$$x^2-2ax+a^2+b^2-r^2 \le 0 \Leftrightarrow$$

$$x \in \left[a-\sqrt{r^2+b^2}, a+\sqrt{r^2+b^2}\right] \ (1)$$

Τα βήματα του αλγορίθμου που θα χρησιμοποιήσουμε φαίνονται παρακάτω.

1. Ταξινομούμε τα n σημεία (x_i, y_i) ως προς την τετμημένη τους x_i.
2. Για κάθε σημείο (a,b) ελέγχουμε εάν καλύπτεται από τον δεξιότερο μέχρι τώρα δίσκο που έχουμε χρησιμοποιήσει. Εάν καλύπτεται προχωράμε στο βήμα 4.
3. Αν δεν καλύπτεται προσθέτουμε έναν δίσκο με κέντρο το $(a+\sqrt{r^2+b^2}, 0)$.
4. Επαναλαμβάνουμε τα βήματα 2, 3 μέχρι να έχουμε εξετάσει όλα τα σημεία.

<u>Απόδειξη ορθότητας</u>

Θα αποδείξουμε με επαγωγή ότι ο αλγόριθμος βρίσκει πάντα τη βέλτιστη λύση. Στο εξής θα συμβολίζουμε με c(n) το πλήθος των κύκλων που χρειάζονται για n δεδομένα σημεία.

Για n = 1, έχουμε ένα μόνο σημείο έστω (a,b) και ο αλγόριθμος προσθέτει ένα δίσκο στο σημείο $(a+\sqrt{r^2+b^2}, 0)$. Ο δίσκος καλύπτει στο σημείο (a,b) αφού το κέντρο του ικανοποιεί την (1) και προφανώς δεν υπάρχει λύση με λιγότερο από 1 δίσκο.

Σύμφωνα με την επαγωγική υπόθεση, ο αλγόριθμος βρίσκει τη βέλτιστη λύση $c(n)$ σε πρόβλημα n σημείων.

Προσθέτουμε στο πρόβλημα ένα ακόμα σημείο με συντεταγμένες (a_{n+1}, b_{n+1}). Για το σημείο n+1 ο αλγόριθμος θα ελέγξει τον δίσκο με το δεξιότερο κέντρο που έχουμε προσθέσει και αν δεν καλύπτει το σημείο (a_{n+1}, b_{n+1}) θα προσθέσει ένα νέο δίσκο με κέντρο $(a_{n+1}+\sqrt{r^2+b_{n+1}^2}, 0)$.

Θα αποδείξουμε ότι αν ο δεξιότερος δίσκος, με τετμημένη d_r, δεν καλύπτει το σημείο (a_{n+1}, b_{n+1}) τότε κανένας άλλος δίσκος που έχουμε χρησιμοποιήσει δεν μπορεί να το καλύψει.

Αφού ο δίσκος d_r δεν το καλύπτει έχουμε $d_r \notin \left[a_{n+1}-\sqrt{r^2+b_{n+1}^2}, a_{n+1}+\sqrt{r^2+b_{n+1}^2}\right]$.

- Αν $d_r < a_{n+1}-\sqrt{r^2+b_{n+1}^2}$ η απόδειξη είναι προφανής, αφού για κάθε άλλο d_i ισχύει:

$$\mathrm{d}_i \le d_r < a_{n+1}-\sqrt{r^2+b_{n+1}^2} \Rightarrow d_i \notin \left[a_{n+1}-\sqrt{r^2+b_{n+1}^2}, a_{n+1}+\sqrt{r^2+b_{n+1}^2}\right]$$

- Αν $d_r > a_{n+1} + \sqrt{r^2 + b_{n+1}^2}$ τότε για κάποιο k με $a_k \leq a_{n+1}$ ισχύει:

$$d_r = a_k + \sqrt{r^2 + b_k^2} \Longrightarrow \nexists d_i \in \left[a_k - \sqrt{r^2 + b_k^2}, a_k + \sqrt{r^2 + b_k^2}\right] \qquad (2)$$

λόγω βελτιστότητας του αλγορίθμου, αφού δεν θα χρησιμοποιούσαμε δίσκο για να καλύψουμε ένα σημείο που είχε ήδη καλυφθεί από κύκλο d_i.
Επίσης ισχύει ότι

$$\begin{cases} a_k \leq a_{n+1} \ \ και \\ d_r = a_k + \sqrt{r^2 + b_k^2} > a_{n+1} - \sqrt{r^2 + b_{n+1}^2} \end{cases} \Longrightarrow \begin{cases} a_k \leq a_{n+1} \\ \sqrt{r^2 + b_k^2} > \sqrt{r^2 + b_{n+1}^2} \end{cases}$$

$$\Longrightarrow a_k - \sqrt{r^2 + b_k^2} < a_{n+1} - \sqrt{r^2 + b_{n+1}^2} \Longrightarrow$$

$$\left[a_{n+1} - \sqrt{r^2 + b_{n+1}^2}, a_{n+1} + \sqrt{r^2 + b_{n+1}^2}\right] \subseteq \left[a_k - \sqrt{r^2 + b_k^2}, a_k + \sqrt{r^2 + b_k^2}\right]$$

Άρα από (2) $\nexists d_i \in \left[a_{n+1} - \sqrt{r^2 + b_{n+1}^2}, a_{n+1} + \sqrt{r^2 + b_{n+1}^2}\right]$. Αποδείξαμε ότι αν ο δεξιότερος δίσκος, με τετμημένη d_r, δεν καλύπτει το σημείο (a_{n+1}, b_{n+1}) τότε κανένας άλλος δίσκος που έχουμε χρησιμοποιήσει δεν μπορεί να το καλύψει.

Άρα, αποδείξαμε ότι ο αλγόριθμος δεν προσθέτει δίσκους εάν το πρόβλημα με n+1 σημεία λυνόταν με c(n) δίσκους.

Πριν προχωρήσουμε σημειώνουμε πως ισχύει ότι $c(n) \leq c(n+1)$, δηλαδή για δεδομένο πρόβλημα n σημείων, το πλήθος δίσκων που θα χρησιμοποιήσουμε για να καλύψουμε τα n σημεία και ένα ακόμη που προστέθηκε στο πρόβλημα αποκλείεται να είναι μικρότερο από το πλήθος δίσκων που χρειάστηκαν για να καλύψουμε τα n σημεία.

Ο αλγόριθμός μας δίνει $c(n+1) = c(n)$ ή $c(n+1) = c(n) + 1$, όταν το πρόβλημα δεν λύνεται με $c(n)$ δίσκους όπως αποδείξαμε παραπάνω.

Γνωρίζουμε ότι $c(n) \leq c(n+1) \Longrightarrow c(n+1) \in [c(n), \infty)$. Επομένως η τιμή $c(n+1)$ δεν θα μπορούσε να πάρει μικρότερη τιμή από αυτή που υπολόγισε ο αλγόριθμός μας. Άρα, είναι βέλτιστη.

Αποδείξαμε, λοιπόν, ότι ο αλγόριθμος βρίσκει τη βέλτιστη λύση για n+1 σημεία. Άρα επαληθεύτηκε η επαγωγική υπόθεση.

Η πολυπλοκότητα του αλγορίθμου είναι $O(nlogn + n)$ για την ταξινόμηση των σημείων και τη διάσχισή τους. Άρα τελικά έχουμε $O(nlogn)$.

Άσκηση 2:

1. Θα λύσουμε το πρόβλημα χρησιμοποιώντας τον greedy αλγόριθμο που περιγράφεται παρακάτω.

   Ταξινομούμε τους πελάτες με κριτήριο τον λόγο βάρους ανά μονάδα χρόνου $\frac{w_i}{p_i} \equiv \lambda_i$. Η ταξινομημένη ακολουθία πελατών με φθίνοντα λόγο λ_i είναι η λύση του προβλήματος ελάχιστης βεβαρυμμένης εξυπηρέτησης.

   Απόδειξη ορθότητας

   Θα χρησιμοποιήσουμε το παρακάτω επιχείρημα ανταλλαγής για να αποδείξουμε την ορθότητα του αλγορίθμου μας.

   Έστω μία optimal μετάθεση των πελατών της μορφής $\{o_1, o_2, \dots, o_n\}$, και G η ταξινομημένη ακολουθία πελατών που προκύπτει από τον αλγόριθμό μας της μορφής $\{g_1, g_2, \dots, g_n\}$.

   Έστω ότι η optimal δεν συμβαδίζει με τη λύση που προκύπτει από τον αλγόριθμό μας στο σημείο i. Αυτό σημαίνει ότι ο i-οστός πελάτης που θα έπρεπε να εξυπηρετηθεί σύμφωνα με τον αλγόριθμό μας, δηλαδή ο πελάτης g_i, εξυπηρετείται σε μεταγενέστερο χρόνο από την optimal λύση. Δηλαδή $\exists j > i: o_j = g_i$.

   Υποστηρίζουμε ότι μπορούμε να ανταλλάξουμε διαδοχικά τον πελάτη o_j με τους ενδιάμεσους $o_k, με\ k \in [i, j-1]$ και να φέρουμε την optimal λύση στη μορφή της δικής μας.

   Σημειώνουμε πως $\forall\ o_k: k \geq i$ ισχύει ότι $\lambda_{o_k} \leq \lambda_{o_j}$ καθώς ο πελάτης o_j είναι ο πελάτης με τον μεγαλύτερο λόγο λ που δεν έχει εξυπηρετηθεί πριν τον πελάτη o_i. (1)

   - Ανταλλάσσουμε τους πελάτες o_j, o_{j-1}.
     Τότε ο χρόνος γίνεται
     $$T' = p_1(w_1 \dots + w_n) + p_2(w_2 + \cdots + w_n) + \cdots + p_J\big(w_J + w_{J-1} + \cdots + w_n\big) + p_{J-1}\big(w_{J-1} + \cdots + w_n\big) + p_{J+1}\big(w_{J+1} + \cdots w_n\big) + \cdots + p_n w_n$$
     Οι δείκτες ακολουθούν την optimal μετάθεση και $J = o_j, (J-1) = o_{j-1}$, κοκ.

     Θα αποδείξουμε ότι $T' \leq T$,

     $$\text{όπου } T = p_1(w_1 \dots + w_n) + p_2(w_2 + \cdots + w_n) + \cdots + p_{J-1}\big(w_{J-1} + w_J + \cdots + w_n\big) + p_J\big(w_J + \cdots + w_n\big) + p_{J+1}\big(w_{J+1} + \cdots w_n\big) + \cdots + p_n w_n$$

     Έχουμε:
     $$T' \leq T \Leftrightarrow$$
     $$p_J\big(w_J + w_{J-1} + \cdots + w_n\big) + p_{J-1}\big(w_{J-1} + \cdots + w_n\big) \leq p_{J-1}\big(w_{J-1} + w_J + \cdots + w_n\big) + p_J\big(w_J + \cdots + w_n\big) \Leftrightarrow$$
     $$p_J w_{J-1} \leq p_{J-1} w_J \Leftrightarrow$$
     $$\frac{w_{J-1}}{p_{J-1}} \leq \frac{w_J}{p_J} \Leftrightarrow \lambda_{J-1} \leq \lambda_J\ \textit{που ισχύει από}\ (1)$$

- Συνεχίζουμε με διαδοχικές ανταλλαγές του o_j μέχρι να φτάσει στη θέση i της optimal μετάθεσης. Κάθε φορά ο χρόνος είναι μικρότερος σύμφωνα με το παραπάνω επιχείρημα.

Επομένως, κάθε φορά που η λύση μας δεν συμφωνεί με την optimal μπορούμε μέσω διαδοχικών ανταλλαγών να τις φέρουμε στο ίδιο σημείο και οι μεταθέσεις τους εν τέλει να ταυτίζονται.

Άρα, η λύση μας είναι επίσης optimal.

Υπολογιστική πολυπλοκότητα

Ο χρόνος που απαιτείται για τον υπολογισμό της λύσης του προβλήματος είναι εκείνος της ταξινόμησης των πελατών βάσει του λόγου λ. Επομένως, έχουμε πολυπλοκότητα O(nlogn).

2. Θα λύσουμε το πρόβλημα με δυναμικό προγραμματισμό. Η αναδρομική σχέση είναι η παρακάτω:

$$\mathrm{d}p(i,P) = min\begin{cases} dp(i+1, P+p_i) + w_i(P+p_i) \\ dp(i+1,P) + w_i\left(\sum_{j=0}^{i} p_j - P\right) \end{cases}$$

όπου i είναι ο i-οστός πελάτης που εξετάζουμε σε ποιον υπάλληλο θα ανατεθεί, και P είναι η αναμονή που υπάρχει ήδη στον έναν από τους δύο υπαλλήλους, έστω στον πρώτο. Γνωρίζοντας την αναμονή P μπορούμε να υπολογίσουμε την αναμονή που έχουμε μέχρι στιγμής στον υπάλληλο 2. Είναι ίση με $P_2 = \sum_{j=0}^{i} p_j - P$, δηλαδή είναι ίση με τη συνολική αναμονή μέχρι τον i-οστό πελάτη μειωμένη κατά την αναμονή των πελατών που έχουν ανατεθεί στον υπάλληλο 1.
Εξηγούμε σύντομα την αναδρομή. Έχουμε ότι η βέλτιστη λύση είναι η ελάχιστη τιμή
1. του βεβαρυμμένου χρόνου εξυπηρέτησης αν ο πελάτης i ανατεθεί στον υπάλληλο 1 (και ο χρόνος P αυξηθεί κατά τον χρόνο εξυπηρέτησης του i) αυξημένου κατά τον όρο $w_i(P+p_i)$, δηλαδή τη βεβαρυμμένη αναμονή του πελάτη i στον υπάλληλο 1 και
2. Του βεβαρυμμένου χρόνου εξυπηρέτησης αν ο πελάτης i ανατεθεί στον υπάλληλο 2 (και ο χρόνος P μείνει σταθερός) αυξημένου κατά τον όρο $w_i\left(\sum_{j=0}^{i} p_j - P\right)$, δηλαδή τη βεβαρυμμένη αναμονή του πελάτη i στον υπάλληλο 2.

Η λύση του προβλήματός μας είναι η τιμή $\mathrm{d}p(1,0)$. Οι «αρχικές» συνθήκες που θα χρησιμοποιήσουμε για να τερματίσει ο αλγόριθμός μας είναι $\forall j: dp(n,j) = 0$. Επομένως, ο αλγόριθμος θα τερματίσει μόλις φτάσει σε κάποιο «κελί» της μορφής $\mathrm{d}p(n,i)$. Προφανώς, το i θα είναι κάποια valid τιμή αναμονής, αφού θα έχει προκύψει από διαδοχικές αθροίσεις p_i.

Οι τιμές $\sum_{j=0}^{i} p_j$ είναι τα prefix του πίνακα p και μπορούν να υπολογιστούν σε χρόνο $O(n)$. Επομένως, η αναδρομή κάνει $n * \sum_{j=1}^{n} p_j$ επαναλήψεις και σε κάθε επανάληψη κάνει σταθερό αριθμό από συγκρίσεις. Επομένως, η συνολική πολυπλοκότητα που προκύπτει είναι $O\left(n * \sum_{j=1}^{n} p_j\right)$.

Η παραπάνω σχέση για $\mathrm{m} \geq 3$ υπαλλήλους γίνεται:

$$dp(i, P_1, \dots P_{n-1}) = min \begin{cases} min_{j=1}^{n} \left(dp\left(i, \dots, P_j + p_i, \dots\right) + w_i\left(P_j + p_i\right)\right) \\ dp(i, P_1, \dots, P_{n-1}) + w_i \left(\sum_{j=1}^{i} p_j - \sum_{j=1}^{n-1} P_j \right) \end{cases}$$

Τα ίδια με πριν ισχύουν για τους χρόνους$\sum_{j=1}^{i} p_j$, $\sum_{j=1}^{n-1} P_j$. Ο αλγόριθμός μας τώρα έχει πολυπλοκότητα $O\left(n * \left(\sum_{j=1}^{n} p_j\right)^{n-1}\right)$.

Άσκηση 3:

(α) Θα λύσουμε το πρόβλημα χρησιμοποιώντας δυναμικό προγραμματισμό. Η ελάχιστη τιμή για την κάλυψη του f-οστού σημείου είναι:

$$c(f) = \min_{0<i\leq f}\left\{c(i-1) + \left(x_f - x_i\right)^2 + C\right\}$$

Με αρχική συνθήκη $c(0) = 0$.

Δηλαδή, η βέλτιστη λύση είναι κάθε φορά το ελάχιστο κόστος για να καλυφθούν τα σημεία μέχρι και το $(i-1)$ αυξημένο κατά το κόστος να καλυφθούν με ένα υπόστεγο τα σημεία που έμειναν από το i μέχρι το f.

Η πολυπλοκότητα του αλγορίθμου είναι $O(n^2)$ καθώς θα εκτελέσει n επαναλήψεις και σε κάθε μία θα χρειάζεται να κάνει n το πολύ συγκρίσεις. Προφανώς, θα χρειαστεί memoization ώστε να μην επανυπολογίσουμε κάποια τιμή $c(i)$.

(β) Το πρόβλημα λύνεται εάν βρούμε ποια ευθεία έχει τη μικρότερη τιμή σε κάθε διάστημα και αντιστοιχίσουμε κάθε x_i στο διάστημα που ανήκει.

Στο εξής θα συμβολίζουμε με c_{ij} το σημείο τομής των ευθειών $(a_i, b_i), \left(a_j, b_j\right)$. Προφανώς $c_{ij} = c_{ji}$.

Από το γεγονός ότι οι κλίσεις των ευθειών δίνονται σε φθίνουσα σειρά από την είσοδο, συμπεραίνουμε ότι η ευθεία (a_i, b_i) βρίσκεται «κάτω» από κάθε ευθεία $\left(a_j, b_j\right)$ $με$ $j < i$ στο διάστημα $\left(c_{ij}, \infty\right)$. Άρα, για κάθε (a_i, b_i) αρκεί να βρούμε το σημείο τομής της c_i' με κάποια από τις ευθείες που σχηματίζουν -μέχρι στιγμής- το κάτω φράγμα (convex hull). Για κάθε σημείο δεξιότερα αυτού, το κάτω φράγμα θα είναι η ευθεία (a_i, b_i).

Ο αλγόριθμος φαίνεται παρακάτω σε ψευδοκώδικα.

```
conv_hull = [1]                                  //αρχικό φράγμα μόνο η ευθεία (a_1, b_1)
start = [−∞]                                     //ξεκινάει από το −∞
for i in range(2,n):
        j = conv_hull_end
        c_ij = (b_j − b_i) / (a_i − a_j)         //σημείο τομής
        while (j ≥ 0 && c_ij < start[j]):        /*αν την τέμνει αριστερότερα τότε πρέπει
                                                 να βρω σημείο τομής με την προηγούμενη*/
                j = j-1
                c_ij = (b_j − b_i) / (a_i − a_j)
        delete conv_hull[j+1, …], start[j+1, …]  //όταν το βρω διαγράφω τις επόμενες
        conv_hull[j+1] = i                       //κρατάω μόνο την καινούρια
        start[j+1] = c_ij
```

Μετά τον τερματισμό του αλγορίθμου έχουμε την τεθλασμένη γραμμή που αποτελεί το κάτω φράγμα που αναζητούμε, καθώς και τα σημεία στα οποία αλλάζει κλίση.

Η παραπάνω διαδικασία απαιτεί χρόνο $O(n)$ καθώς κάθε ευθεία μία το πολύ φορά προστίθεται στο conv_hull και μία το πολύ φορά διαγράφεται. Επομένως, εξετάζουμε κάθε ευθεία 2 το πολύ φορές. Επίσης μπορούμε να θεωρήσουμε πως κάθε προσθήκη ευθείας στο κάτω φράγμα έχει πολυπλοκότητα περίπου $O(1)$.

Έχοντας τώρα τους πίνακες conv_hull και start, ακολουθούμε τον παρακάτω αλγόριθμο για να βρούμε τους δείκτες j(i). Διατρέχουμε τις θέσεις x_i και τις συγκρίνουμε με τις τιμές

της δομής start. Για κάθε x_i ψάχνουμε με τη δυάδα τιμών που ισχύει $start[m] \leq x_i \leq start[m+1]$. Μόλις τη βρούμε αντιστοιχίζουμε $j(i) = conv\_hull[m]$.

Η διαδικασία αυτή απαιτεί μία προσπέλαση των τιμών x_i κόστους $O(k)$ και μία το πολύ προσπέλαση της δομής start με κόστος $O(n)$. Είναι προφανές, ότι κάθε φορά συνεχίζουμε να διατρέχουμε το start από εκεί που σταματήσαμε για το προηγούμενο x_i μιας και τόσο τα x_i όσο και τα σημεία του start είναι σε αύξουσα σειρά. Άρα, μπορούμε να θεωρήσουμε πως η πολυπλοκότητα εύρεσης κάθε δείκτη j(i) είναι περίπου O(1).

Επομένως η συνολική πολυπλοκότητα του αλγορίθμου είναι $O(n+k)$.

Θα λύσουμε το πρόβλημά μας χρησιμοποιώντας τον παραπάνω αλγόριθμο. Αλλάζουμε τη μορφή της αναδρομικής σχέσης ως εξής:

$$c(f) = \min_{0<i\leq f}\left\{c(i-1) + \left(x_f - x_i\right)^2 + C\right\} \Leftrightarrow$$
$$c(f) = x_f^2 + C + \min_{0<i\leq f}\left\{c(i-1) + x_i^2 - 2x_i x_f\right\}$$

Δηλαδή, για κάθε $c(f)$ αρκεί να βρούμε για ποιο i η ποσότητα $c(i-1) + x_i^2 - 2x_i x_f$ είναι ελάχιστη. Ισοδύναμα ποια από τις ευθείες $(a_i, b_i) = \left(-2x_i, c(i-1) + x_i^2\right)$ έχει τη μικρότερη τιμή στο σημείο x_f.

Παρατηρούμε ότι οι ευθείες αυτές έχουν $0 \geq a_1 \geq a_2 \geq \cdots \geq a_n$, αφού τα x_i είναι σε αύξουσα σειρά και θετικά.

Άρα, ξεκινάμε από το $c(1)$ και για κάθε $c(i)$ βρίσκουμε σε $O(1)$ τον δείκτη $j(i)$, δηλαδή: $c(i) = x_i^2 + C + \boldsymbol{c(j(i)-1) + x_{j(i)}^2 - 2x_{j(i)}x_i}$.

Σε χρόνο $O(1)$ ανανεώνουμε το convex hull προσθέτοντας (αν χρειάζεται) την ευθεία (a_i, b_i) στο κάτω φράγμα. Συνεχίζουμε μέχρι το $c(f)$.

Άρα, χρειάστηκαν n αναζητήσεις δεικτών j(i) και n το πολύ προσθήκες στο conv_hull. Συνολικά, λοιπόν ο απαιτούμενος χρόνος είναι $O(n)$.

Άσκηση 4:

(α) Θα λύσουμε το πρόβλημα με δυναμικό προγραμματισμό. Η αναδρομική σχέση που λύνει το πρόβλημα φαίνεται παρακάτω.

$$dp(i,j,start) = \begin{cases} 0, & i = j \\ \min\big(dp(i-1,j,start) + sum(start,i), dp(i-1,j-1,i)\big), & else \end{cases}$$

όπου το i είναι το πλήθος των φοιτητών που δεν έχουν ανατεθεί σε κάποιο λεωφορείο, το j είναι το πλήθος των υπολειπόμενων λεωφορείων και το start ο φοιτητής που ανατέθηκε πρώτος στο τελευταίο λεωφορείο. Ως $sum(i,j)$ ορίζουμε το άθροισμα $\sum_{m=i+1}^{j} A_{im}$, δηλαδή την ευαισθησία που προσθέτει ο i-οστός φοιτητής στο εάν μπει στο λεωφορείο που βρίσκονται ήδη οι φοιτητές $[i+1, j]$.

Εξηγούμε σύντομα τις τρεις επιλογές της αναδρομής.
Αν ο αριθμός των υπολειπόμενων λεωφορείων είναι ίσος με τον αριθμό των φοιτητών που υπολείπονται, τότε η βέλτιστη λύση είναι να μπει ο καθένας μόνος του, συνεπώς μηδενική συνολική επιβάρυνση.
Σε άλλη περίπτωση είναι το ελάχιστο του ενδεχομένου να προσθέσουμε τον i-οστό φοιτητή στο τελευταίο λεωφορείο με αυξημένο δείκτη ευαισθησίας κατά την ευαισθησία που οφείλεται στον φοιτητή i, και του ενδεχομένου να αναθέσουμε τον i-οστό φοιτητή στο j-οστό λεωφορείο χωρίς η συνολική ευαισθησία να αυξηθεί.
Η λύση του προβλήματος είναι η τιμή $dp(n-1, k-1, n)$ με αρχικές συνθήκες $dp(i,-1,p) = \infty,\ dp(0,m,p) = 0,\ m \geq 0$.

Υπολογιστική πολυπλοκότητα
Είναι προφανές ότι με χρήση memoization ο αλγόριθμος θα χρειαστεί $n * k * n = kn^2$ επαναλήψεις για να υπολογίσει κάθε στοιχείο $dp(x,y,z)$. Υποστηρίζουμε πως ο υπολογισμός των τιμών αυτών μπορεί να γίνει σε $O(1)$ εάν έχουμε προϋπολογίσει τα αθροίσματα $sum(i,j)$.
Πράγματι ο υπολογισμός μπορεί να γίνει χρησιμοποιώντας τις βοηθητικές τιμές $S(i,j)$ οι οποίες αντιστοιχούν στην ευαισθησία που προσθέτει ο φοιτητής i εάν μπει στο λεωφορείο που είναι ήδη οι φοιτητές $[1,j]$, δηλαδή $S(i,j) = \sum_{m=1}^{j} A_{im}$.
Χωρίς βλάβη της γενικότητας θεωρούμε $i < j$. Έχουμε:

$$S(i,j) - S(i,i) = \sum_{m=1}^{j} A_{im} - \sum_{m=1}^{i} A_{im} = \sum_{m=i+1}^{j} A_{im} = sum(i,j)$$

Οι τιμές $S(i,j)$ είναι τα prefix sums του πίνακα A_{ij} και υπολογίζονται σε χρόνο $O(n^2)$.
Επομένως η συνολική πολυπλοκότητα του αλγορίθμου είναι $O(n^2 + kn^2) = O(kn^2)$.

Άσκηση 5:

(α) Το γράφημα $T_1 \setminus \{e\}$ προκύπτει από την αφαίρεση μίας ακμής από ένα συνδετικό δέντρο. Όμως ένα συνδετικό δέντρο είναι ελαχιστικά συνεκτικό, άρα αποτελείται από 2 συνεκτικές συνιστώσες έστω H_1, H_2.
Το γράφημα T_2 είναι επίσης συνδετικό. Επομένως, λόγω συνεκτικότητας υπάρχει τουλάχιστον μία ακμή e' που ενώνει τις συνιστώσες H_1, H_2. Προφανώς $e' \notin T_1$, γιατί τότε το $T_1 \setminus \{e\}$ θα παρέμενε συνεκτικό εξαιτίας της e'.
Άρα, το γράφημα $(T_1 \setminus \{e\}) \cup \{e'\}$ είναι και πάλι συνεκτικό με $|E| = |E|_T = |V| - 1$.
Άρα, $(T_1 \setminus \{e\}) \cup \{e'\}$ συνδετικό δέντρο.

<u>Αλγόριθμος εύρεσης e'</u>
Έστω $e = (u, v)$.
Διασχίζοντας τις ακμές του $T_1 \setminus \{e\}$ σε $O(|V|)$ θέτουμε $EquivClass(i) = \begin{cases} H_1, & i \in H_1 \\ H_2, & i \in H_2 \end{cases}$.
Εξετάζουμε τις ακμές του T_2, όπου $|E| = |V|$.
Μόλις συναντήσουμε ακμή $(u', v') \in T_2$: $EquivClass(u') \neq EquivClass(v')$, έχουμε βρει την ακμή $e' = (u', v')$.
Η συνολική πολυπλοκότητα του αλγορίθμου είναι $O(|V|)$.

(β) Ο ισχυρισμός ότι το γράφημα H είναι συνεκτικό είναι ισοδύναμος με την πρόταση ότι $\forall T, T' \in H$: $\exists$ μονοπάτι $p = (T, \dots, T')$. Έστω ότι τα T, T' διαφέρουν κατά m' ακμές.

Θα αποδείξουμε τον ισοδύναμο ισχυρισμό με επαγωγή στο πλήθος m'. Ταυτόχρονα θα αποδείξουμε ότι το μονοπάτι p έχει μήκος $|T \setminus T'| = m'$.

Για $m' = 1$ έχουμε ότι:
$$|T \setminus T'| = |T' \setminus T| = 1 \Longrightarrow \exists e \in H: e = (T, T') \Longrightarrow p = (T, T') \; και \; |p| = 1.$$
Έστω ότι ο ισχυρισμός είναι αληθής για πλήθος m'. Τότε $\exists p = (T, \dots, T') \; με \; |p| = m'$.

Εξετάζουμε εάν ο ισχυρισμός αληθεύει για $m'' = m' + 1$. Δηλαδή εάν υπάρχει μονοπάτι που να ενώνει τα δέντρα T, T'' για τα οποία $|T \setminus T''| = |T'' - T| = m'' = m' + 1$.
Από ερώτημα (α) για τα T, T'' έχουμε ότι $\exists e \in T \; και \; \exists e' \in T''$:
Το δέντρο $T_m = (T'' \setminus \{e'\}) \cup \{e\}$ είναι συνδετικό.
Προφανώς $|T'' \setminus T_m| = 1 \; και \; |T - T_m| = m'' - 1 = m'$ (φέραμε τα δέντρα κατά 1 ακμή «πιο κοντά»).
Άρα $\exists T_m$: $\begin{cases} T_m \in H \; και & (1) \\ |T'' \setminus T_m| = 1 \; και & (2) \\ |T \setminus T_m| = m' & (3) \end{cases}$
Από (1), (3) και από την υπόθεση της επαγωγής μας έχουμε ότι $\exists p = (T, \dots, T_m)$.
Επίσης, από (2) και από τη βάση της επαγωγής ξέρουμε ότι $\exists p_m = (T_m, T'')$.
Επομένως, το μονοπάτι $p'' = p + p_m = (T, \dots, T_m, T'')$ ενώνει τα δέντρα T, T'' και έχει μήκος $|p''| = m' + 1$.

Άρα, αποδείξαμε επαγωγικά πως πράγματι υπάρχει μονοπάτι στο H που να ενώνει τα δέντρα T, T'' για τα οποία $|T \setminus T''| = |T'' - T| = m'' = m' + 1$ με μήκος ίσο με τη διαφορά των δέντρων T, T''. Επομένως, το γράφημα H είναι συνεκτικό και ισχύει $d(T, T') = |T \setminus T'|$.

Ο αλγόριθμος για να βρούμε ένα μονοπάτι p στο H που να ενώνει τα δέντρα T_1, T_2 φαίνεται στα παρακάτω βήματα. Ορίζουμε $G = T_1 \setminus T_2$.

1. Έστω ακμή e: $e \in G$. Εφαρμόζουμε τον αλγόριθμο του (α) στο T_1 για την ακμή e και προκύπτει ένα δέντρο T'.
2. Προσθέτουμε το T' στο μονοπάτι p και επαναλαμβάνουμε το βήμα 1 για $T_1 = T'$, $G = G \setminus \{e\}$.

Ο αλγόριθμος θα τερματίσει όταν G = {} και θα εκτελέσει $|T_1 \setminus T_2|$ επαναλήψεις. Προφανώς, όταν G = {} θα ισχύει ότι $T' = T_2$.

(γ) Πριν παρουσιάσουμε τον αλγόριθμο, σημειώνουμε ότι τα δέντρα $T(e_i)$ που θα προκύψουν θα απέχουν από το MST(G) κατά μία ακμή. Αυτό συμβαίνει επειδή, εάν στο MST(G) προσθέσουμε την ακμή e_i τότε δημιουργείται κύκλος (μεγιστικά ακυκλικό). Αν από τον κύκλο αφαιρέσουμε τη βαρύτερη ακμή έστω e_i', το γράφημα που θα προκύψει θα είναι και πάλι συνδετικό με το ελάχιστο δυνατό κόστος δεδομένου του περιορισμού να περιέχει την ακμή e_i.

Ο αλγόριθμος περιγράφεται στα παρακάτω βήματα.

1. Ταξινομούμε τις ακμές του G και υπολογίζουμε το $T = MST(G)$.
2. Για κάθε ακμή e_i του T (ξεκινώντας από τη βαρύτερη και συνεχίζοντας στις ταξινομημένες ακμές) εκτελούμε τον αλγόριθμο του ερωτήματος (α) για τις τιμές $T_1 = T, T_2 = G \setminus T, e = e_i$. Η ορθότητα του αλγορίθμου δεν επηρεάζεται εάν το T_2 δεν είναι MST, καθώς εάν δεν βρεθεί ακμή e′ αυτό θα σημαίνει ότι καμία ακμή του G δεν θα μπορούσε να ανταλλαχθεί με την e_i στην απόδειξη που κάναμε παραπάνω. Σε άλλη περίπτωση ονομάζουμε e_i' την ακμή που βρέθηκε από τον αλγόριθμο του (α).
   Σημείωση: Από τις ακμές του T_2 ελέγχουμε μόνο όσες έχουν w $\geq w(e_i)$, καθώς εάν μία ακμή ήταν ελαφρύτερη της e_i και μπορούσε να ανταλλαχθεί με αυτήν στο MST, θα είχαμε χρησιμοποιήσει αυτή στη θέση της εξ' αρχής.
3. Το δέντρο $T(e_i) = (T \setminus \{e_i\}) \cup \{e_i'\}$, σύμφωνα με την απόδειξη που κάναμε. Για το κόστος έχουμε $w\big(T(e_i)\big) = w(T) - w(e_i) + w(e_i')$.
4. Αν εξετάσαμε όλες τις ακμές του T, τερματίζουμε αλλιώς επιστρέφουμε στο βήμα 2 με $T_2' = T_2 \setminus \{e_i\}$.

<u>Απόδειξη ορθότητας</u>

Ο αλγόριθμος «ανταλλάσσει» κάθε ακμή με την βαρύτερη ακμή που δεν «σπάει» τη συνεκτικότητα του συνδετικού δέντρου. Επομένως, επιλέγοντας μεγάλο w(e_i) ελαχιστοποιούμε το κόστος $w\big(T(e_i)\big) = w(T) - w(e_i) + w(e_i')$.

Το T_2 που προκύπτει σε κάθε επανάληψη έχει το πολύ $|E|$ ακμές, επομένως σε κάθε επανάληψη έχουμε κόστος $O(|V| + |E|) = O(|E|)$ από ερώτημα (α). Κάνουμε |V| τέτοιες επαναλήψεις, μία για κάθε ακμή του MST. Επομένως, η συνολική πολυπλοκότητα του αλγορίθμου είναι $O(|E| \log|E| + |V||E|) = O(|VE|)$.