ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Λύσεις 1ης Σειράς Γραπτών Ασκήσεων για το μάθημα "Αλγόριθμοι και Πολυπλοκότητα"

Δημήτριος Βασιλείου
03119830
7ο Εξάμηνο

Άσκηση 1: Πλησιέστερο ζεύγος σημείων

(α). Η ιδέα για την ανάπτυξη του αλγορίθμου βασίζεται στον αλγόριθμο του ίδιου προβλήματος στις δύο διαστάσεις τροποποιώντας τον κατάλληλα.

Περιγραφή Αλγορίθμου

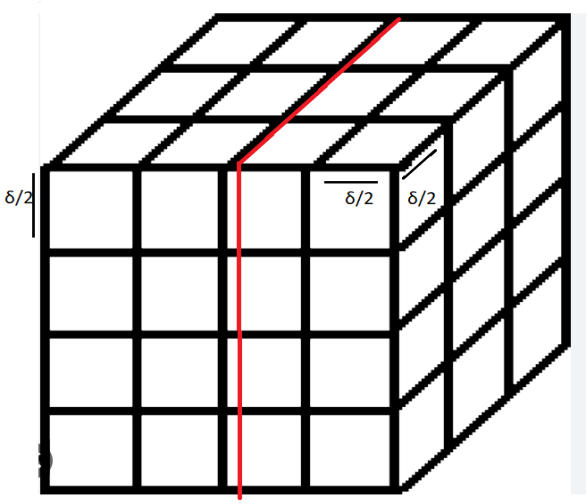
Έστω P το σύνολο των n σημείων του χώρου που μας δίνονται. Θέλουμε κάθε αναδρομική κλήση να διαθέτει μία ταξινόμηση των στοιχείων κατά x, y και z. Συνεπώς πριν ξεκινήσει η αναδρομή δημιουργούμε λίστες P_x, P_Y και P_Z, όπου P_i λίστα των στοιχείων ταξινομημένων σε αύξουσα σειρά κατά την i-συνιστώσα. Η ταξινόμηση αυτή έχει κόστος $O(nlogn)$.

Θεωρούμε επίπεδο L το οποίο χωρίζει «στα δύο» τον χώρο και ορίζουμε τις ακόλουθες λίστες: $Q_x, R_x, Q_y, R_y, Q_z, R_z$ όπου η Q_i περιέχει τα σημεία που βρίσκονται αριστερά του L ταξινομημένα σε αύξουσα σειρά κατά την i-συνιστώσα και η Ri τα σημεία που βρίσκονται δεξιά του L ταξινομημένα σε αύξουσα σειρά κατά την i-συνιστώσα. Ο σχηματισμός αυτών των λιστών γίνεται εύκολα σε χρόνο $O(n)$ διατρέχοντας τις λίστες P_x, P_y και P_z που φτιάξαμε παραπάνω.

Θεωρούμε ότι με το τέλος της αναδρομής γνωρίζουμε ένα πλησιέστερο ζεύγος σημείων στον «αριστερό» χώρο (Q). Έστω q_0, q_1 αυτό το ζεύγος. Όμοια γνωρίζουμε ένα πλησιέστερο ζεύγος r_0, r_1 στο R. Θεωρούμε $\delta = \min\{d(q_0,\ q_1), d(r_0,\ r_1)\}$.

Έχοντας βρει την μικρότερη απόσταση μεταξύ δύο σημείων στον ίδιο «ημιχώρο» μένει να εξετάσουμε τι συμβαίνει με ζεύγη σημείων εκ των οποίων το ένα βρίσκεται στον Q και το άλλο στον R. Αρκεί να περιορίσουμε την αναζήτησή μας σε περιοχή απόστασης δ από το επίπεδο L, διότι οποιαδήποτε άλλα ζεύγη σημείων βρίσκονται σε απόσταση μεγαλύτερη από δ δεν πρόκειται να αποτελούν λύση στο πρόβλημα.

Έστω S το σύνολο των σημείων αυτών. Αν υπάρχουν σημεία s, s' με $d(s, s') < \delta$, τότε αυτά θα ανήκουν στο S. Δημιουργούμε λίστες S_y, S_z δηλαδή λίστες με τα σημεία του S ταξινομημένα ως προς y και z αντίστοιχα. Για την δημιουργία αυτών των λιστών αρκεί να διασχίσουμε από μία φορά τις λίστες P_x και P_z, συνεπώς απαιτείται χρόνος $O(n)$. Τώρα μένει να αποδείξουμε ότι κάθε σημείο στο σύνολο S αρκεί να εξεταστεί για ελάχιστη απόσταση με σταθερό αριθμό σημείων. Χωρίζουμε το σύνολο S σε κύβους ακμής $\delta/2$. Κάθε κύβος περιέχει το πολύ ένα σημείο, διότι αν περιείχε 2 τότε αυτά θα είχαν απόσταση το πολύ όσο η διαγώνιος του κύβου δηλαδή $\sqrt{3}\delta/2 < \delta$, άτοπο επειδή δ είναι η μικρότερη απόσταση μεταξύ σημείων που ανήκουν στον ίδιο ημιχώρο. Ισχυριζόμαστε ότι κάθε σημείο στο S αρκεί να ελεγχθεί με τα επόμενα 63 σημεία του ως προς y και τα επόμενα 63 σημεία του ως προς z.



Απόδειξη του ισχυρισμού:

Έστω ότι τα σημεία s, s' με $d(s, s') < \delta$ χωρίζονται από τουλάχιστον 64 κουτιά. Τότε θα υπάρχουν τουλάχιστον 3 σειρές του S μεταξύ τους, συνεπώς αυτά θα έχουν απόσταση τουλάχιστον ${}^{3\delta}/_{2} > \delta$, άτοπο.■

Τελικά κάθε σημείο του S, αρκεί να ελεγχθεί με τα 63 επόμενά του ως σημεία του ως προς y και τα επόμενα 63 σημεία του ως προς z, διαδικασία η οποία απαιτεί χρόνο $O((63 + 63)n) = O(n)$, καθώς επιτυγχάνεται με προσπέλαση των λιστών S_y, S_z.

Στο τέλος, ο αλγόριθμος θα κρατήσει την μικρότερη από τις αποστάσεις μεταξύ σημείων του S, συγκρίνει αυτή την απόσταση με το δ και επιστρέφει την μικρότερη των δύο.

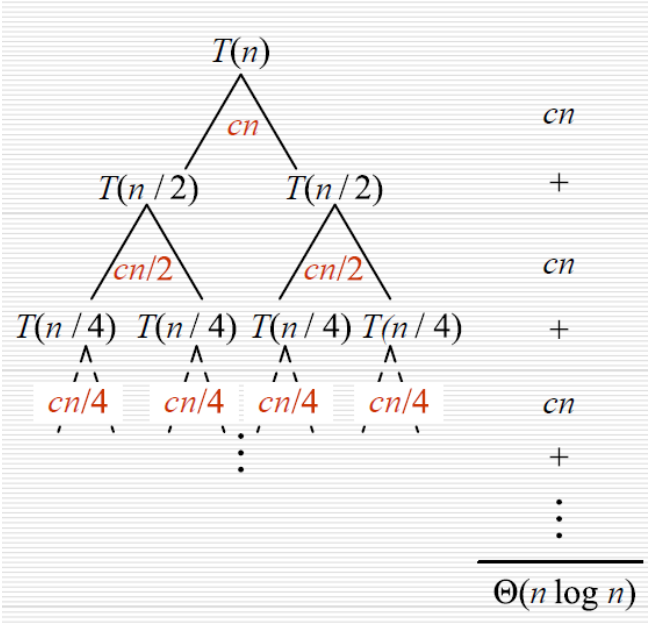
Απόδειξη Ορθότητας

Ο αλγόριθμος αρχικά ψάχνει με αναδρομή για ελάχιστο απόσταση μεταξύ σημείων που ανήκουν στον ίδιο ημιχώρο Q ή R και κρατάει την ελάχιστη απόσταση δ. Έπειτα ψάχνει για ελάχιστη απόσταση μεταξύ σημείων οπού το ένα ανήκει στον χώρο Q και το άλλο στον χώρο R. Ο ισχυρισμός που αποδείξαμε εξασφαλίζει ότι η ελάχιστη απόσταση μεταξύ τέτοιων σημείων βρίσκεται σε γραμμικό χρόνο. Στο τέλος ο αλγόριθμος συγκρίνει την ελάχιστη απόσταση αυτή με το δ και επιστρέφει την μικρότερη από τις δύο αποστάσεις, που είναι και το ζητούμενο. Η ορθότητα έγκειται στο γεγονός ότι θα εξεταστούν για ελάχιστη απόσταση και σημεία που βρίσκονται στον ίδιο ημιχώρο αλλά και σημεία που βρίσκονται σε διαφορετικό ημιχώρο.

Ανάλυση Πολυπλοκότητας

Έστω $T(n)$ ο χρόνος για την εύρεση της ελάχιστης απόστασης. Σε κάθε αναδρομή «μοιράζουμε» τα σημεία σε ${}^{n}/_{2}$ και ${}^{n}/_{2}$, οπότε $T({}^{n}/_{2})$ είναι ο χρόνος που απαιτείται για την εύρεση ελάχιστης απόστασης στον αριστερό «ημιχώρο» και $T({}^{n}/_{2})$ ο αντίστοιχος χρόνος για τον δεξιό. Αφού προσδιορισθούν αυτές οι δύο αποστάσεις ο αλγόριθμος εκτελεί λειτουργίες που κοστίζουν γραμμικό χρόνο για την εξέταση των ζευγών σημείων όπου το ένα ανήκει αριστερά και το άλλο δεξιά. Συνεπώς, η αναδρομική σχέση που περιγράφει τον αλγόριθμο είναι η ακόλουθη:

$$T(n) \leq T({}^{n}/_{2}) + \Theta(n)$$



(πηγή: διαφάνειες κυρίου Φωτάκη)

Με βάση το δένδρο αναδρομής όπως φαίνεται παραπάνω, προκύπτει ότι τελικά ο χρόνος εκτέλεσης του αλγορίθμου είναι $O(nlogn)$.

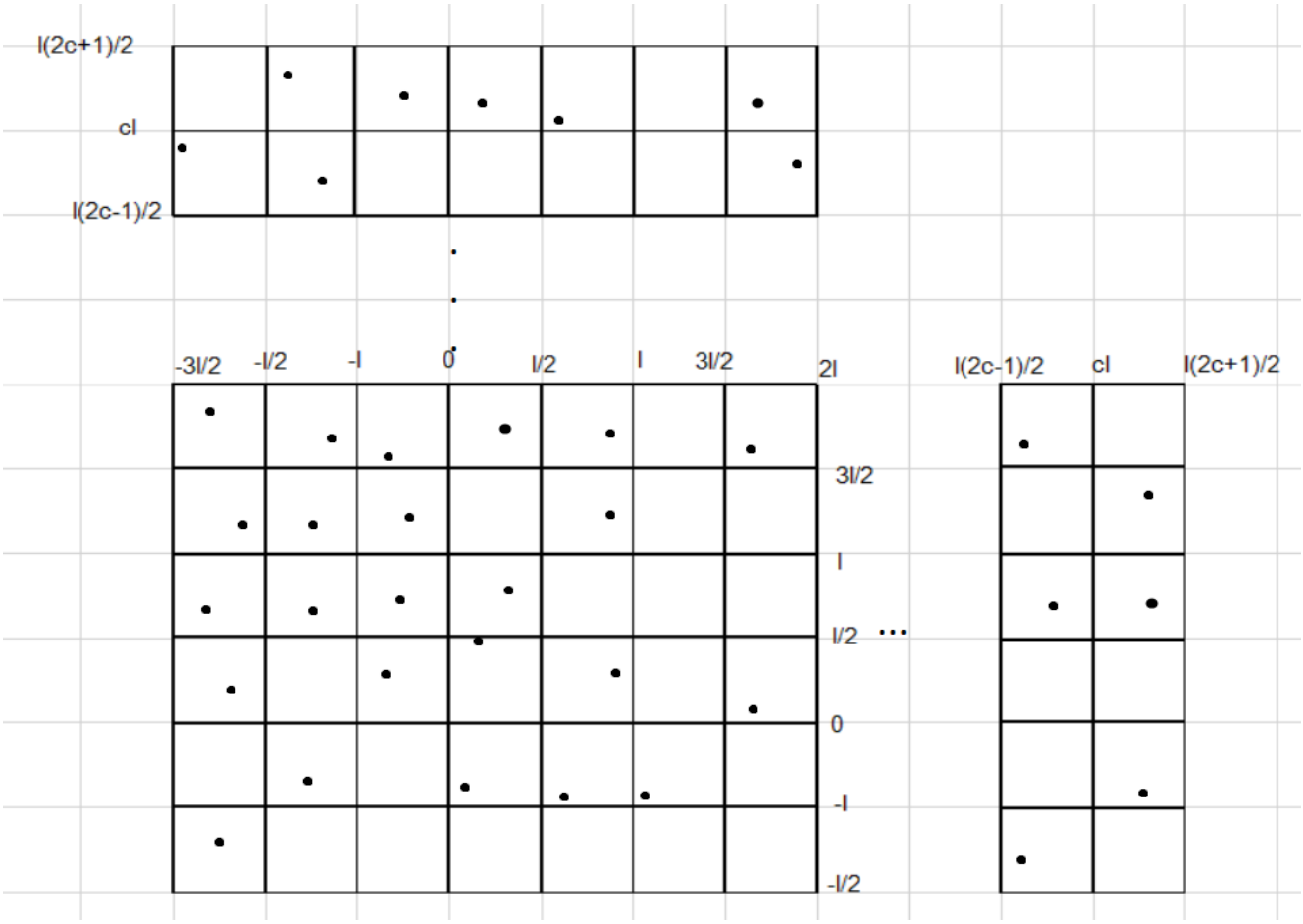
(β). Αρχικά θα διατυπώσουμε λύση του προβλήματος για τις 2 διαστάσεις και στη συνέχεια θα γενικεύσουμε τη λύση για 3 και περισσότερες διαστάσεις.
Θεωρούμε ότι γνωρίζουμε τις συντεταγμένες των n σημείων ως προς x, y, δηλαδή έχουμε $(x_1, x_2, \dots, x_n)$ και $(y_1, y_2, \dots, y_n)$. Εφόσον γνωρίζουμε προσέγγιση του δ^* με κάτω όριο το l θα κατανείμουμε τα σημεία σε buckets ως προς x και y διαιρώντας την κάθε συντεταγμένη με $l/2$. Δηλαδή θα σχηματιστούν buckets ως εξής:

$$x: \left[0, l/2\right), y: \left[0, l/2\right) \rightarrow \textit{περιέχει το σημεία με } x - \textit{συνιστώσα στο διάστημα } \left[0, l/2\right) \textit{ και } y - \textit{συνιστώσα στο διάστημα } \left[0, l/2\right)$$

$$x: \left[0, l/2\right), y: \left[l/2, l\right) \rightarrow \textit{περιέχει τα σημεία με } x - \textit{συνιστώσα στο διάστημα } \left[0, l/2\right) \textit{ και } y - \textit{συνιστώσα στο διάστημα } \left[l/2, l\right)$$

.
.
.

Με αυτόν τον τρόπο εξασφαλίζουμε ότι σε κάθε bucket θα περιέχεται μόνο ένα σημείο, διότι αν περιέχονται 2 τότε η απόστασή τους θα είναι το πολύ όσο η διαγώνιος του κουτιού που θα σχηματιστεί, δηλαδή $\sqrt{2}l/2 < l$, πράγμα άτοπο, διότι η απόσταση δεν μπορεί να είναι κάτω από l.
Αξίζει να σημειωθεί ότι τα buckets φτιάχνονται σε γραμμικό χρόνο, διότι αρκεί ένα πέρασμα στη λίστα των σημείων.
Για τη συνέχεια παραθέτουμε το ακόλουθο πλέγμα που θεωρητικά έχει σχηματιστεί.



Έστω ότι εξετάζουμε το σημείο στο bucket $x: \left[0, {}^{l}/_{2}\right), y: \left[0, {}^{l}/_{2}\right)$. Εφόσον σε κάθε bucket υπάρχει το πολύ ένα σημείο, το σημείο αυτό πρέπει να εξεταστεί με σημεία τα οποία απέχουν x –απόσταση έως cl και y –απόσταση έως cl. Συνεπώς το σημείο πρέπει να εξεταστεί με $4c \cdot 4c = 16c^2$ σημεία. Ουσιαστικά εξετάζουμε όλα τα σημεία που ανήκουν στο «τετράγωνο» πλευράς $4c(2c\ αριστερά, 2c\ δεξιά, 2c\ πάνω\ και\ 2c\ κάτω)$.

Επαναλαμβάνοντας για όλα τα n σημεία τελικά θα προκύψει πολυπλοκότητα $O(n \cdot 16c^2)$, δηλαδή γραμμική ως προς n, αλλά εξαρτώμενη από το c.

Γενικεύοντας στις 3 διαστάσεις, πλέον τα buckets θα περιέχουν και την z –συνιστώσα. Τώρα αντί για κουτιά, τα buckets σχηματίζονται ως κύβοι ακμής ${}^{l}/_{2}$ όπου πάλι κάθε κύβος θα περιέχει το πολύ ένα σημείο, διότι αν περιείχε δύο τότε θα είχαμε απόσταση το πολύ ${}^{\sqrt{3}l}/_{2} < l$, άτοπο. Επίσης τώρα κάθε σημείο πρέπει να εξεταστεί με $4c \cdot 4c \cdot 4c = 64c^3$ σημεία και προκύπτει πολυπλοκότητα $O(n \cdot 64c^3)$.
Για $d \geq 2$ διαστάσεις λοιπόν, η πολυπλοκότητα είναι $O(n \cdot 4c^d)$.

Άσκηση 2: Πόρτες ασφαλείας στο κάστρο

Περιγραφή Αλγορίθμου

Θεωρούμε αρχική διαμόρφωση των διακοπτών ως $0\ 0\ 0 \ldots 0$. Φτιάχνουμε νέα διαμόρφωση $1\ 1\ 1 \ldots 1$ και εστιάζουμε την προσοχή μας στην 1η πόρτα. Η 1η πόρτα αρχικά είναι ανοιχτή ή κλειστή. Έστω χωρίς βλάβη της γενικότητας ότι είναι κλειστή. Αν με την αρχική διαμόρφωση με μηδενικά ανοίγει τότε γνωρίζουμε ότι η πόρτα ανοίγει με 0, αλλιώς αν ανοίγει με την διαμόρφωση με άσσους τότε ανοίγει με 1. Έστω ότι η πόρτα ανοίγει με 1. Στη συνέχεια δοκιμάζουμε διαμόρφωση $0\ 0\ 0 \ldots 0\ 1\ 1\ 1 \ldots 1$ με ${}^{n}/_{2}$ 0 και ${}^{n}/_{2}$ 1. Αν η πόρτα κλείσει (αφού προηγουμένως ήταν ανοιχτή) τότε γνωρίζουμε ότι ο διακόπτης βρίσκεται στο πρώτο «μισό» των διακοπτών, αλλιώς στο δεύτερο μισό. Αντίστοιχα εργαζόμαστε αν η πόρτα ανοίγει με 0. Στη συνέχεια επικεντρωνόμαστε στο «μισό» που έφερε την αλλαγή στην κατάσταση της πόρτας οπότε ουσιαστικά εξετάζουμε ${}^{n}/_{2}$ διακόπτες (οι υπόλοιποι δεν παίζουν ρόλο στην εύρεση του διακόπτη που ανοίγει την 1η πόρτα, οπότε στις επόμενες διαμορφώσεις δεν τους αλλάζουμε κατάσταση). Συνεχίζουμε αναδρομικά την ίδια διαδικασία έως ότου καταλήξουμε στον διακόπτη που ανοίγει την 1η πόρτα. Συνεχίζουμε την ίδια διαδικασία για όλες τις πόρτες.

Απόδειξη Ορθότητας

Η 1η πόρτα αρχικά είναι είτε ανοιχτή είτε κλειστή. Έστω ότι είναι ανοιχτή. Δοκιμάζοντας διαδοχικά διαμορφώσεις $0\ 0\ 0 \ldots 0$ και $1\ 1\ 1 \ldots 1$ μπορούμε να αποφανθούμε για το αν η πόρτα κλείνει με 0 και ανοίγει με 1 ή ανάποδα, διότι αν κλείσει τότε ξέρουμε ότι κλείνει με 0 και ανοίγει με 1, ενώ αν παραμείνει ανοιχτή ξέρουμε ότι ανοίγει με 0 και κλείνει με 1. Εξετάζουμε την περίπτωση όπου η πόρτα κλείνει με 0 και ανοίγει με 1 (η ανάλυση της περίπτωσης η πόρτα να ανοίγει με 0 είναι συμμετρική με την ανάλυση που θα κάνουμε όταν η πόρτα κλείνει με 0).
Ξέρουμε ότι το 0 κλείνει την πόρτα, το 1 την ανοίγει και ότι η πόρτα τώρα είναι ανοιχτή, διότι η τελευταία διαμόρφωση που εφαρμόσαμε ήταν η $1\ 1\ 1 \ldots 1$. Φτιάχνουμε διαμόρφωση $0\ 0\ 0 \ldots 0\ 1\ 1\ 1 \ldots 1$ με ${}^{n}/_{2}$ 0 και ${}^{n}/_{2}$ 1. Αν η πόρτα κλείσει, εφόσον ξέρουμε ότι κλείνει με 0, τότε ο διακόπτης βρίσκεται στο 1ο μισό των διακοπτών οπότε δεν έχει νόημα να εξετάσουμε το δεύτερο μισό. Άρα οι διαμορφώσεις μας τώρα θα μεταβάλουν το πρώτο μισό των διακοπτών ενώ το δεύτερο παραμένει

αμετάβλητο με $1\,1\,1\,\ldots 1$. Αν παραμείνει ανοιχτή εργαζόμαστε ανάλογα, διότι ο διακόπτης βρίσκεται στο δεύτερο μισό.
Παρατηρούμε δηλαδή αναδρομή, αφού πλέον έχουμε ουσιαστικά $^{n}/_{2}$ διακόπτες που πρέπει να εξετάσουμε και η πόρτα είναι κλειστή ή ανοιχτή. Η διαδικασία αυτή επαναλαμβάνεται για όλες τις πόρτες.

Ανάλυση Πολυπλοκότητας
Για κάθε πόρτα φτιάχνουμε διαμορφώσεις πλήθους $logn$, καθώς κάθε φορά επικεντρωνόμαστε στο μισό του πλήθους των διακοπτών που αλλάζουν την κατάσταση της πόρτας. Εφόσον έχουμε n πόρτες συνολικά, η πολυπλοκότητα είναι $O(nlogn)$.

Άσκηση 3: Φόρτιση Ηλεκτρικών Αυτοκινήτων

Περιγραφή Αλγορίθμου
Για κάθε αυτοκίνητο i γνωρίζουμε το χρόνο άφιξης του a_i συνεπώς είναι χρήσιμο να φτιάξουμε έναν πίνακα ο οποίος θα περιέχει όλες τις χρονικές στιγμές άφιξης για κάθε αυτοκίνητο. Έστω A αυτός ο πίνακας, ο οποίος έχει μέγεθος n και a_i η στιγμή άφιξης του αυτοκινήτου i. Ορίζουμε επίσης έναν πίνακα $SoFar$ μεγέθους n, όπου $SoFar[i]$ είναι ο αριθμός των αυτοκινήτων που έχουν φτάσει στον σταθμό μέχρι την χρονική στιγμή i. Προφανώς ο πίνακας $SoFar$ φτιάχνεται σε γραμμικό χρόνο. Έχοντας φτιάξει τους βοηθητικούς πίνακες, ασχολούμαστε με τον αριθμό των εξυπηρετητών. Σίγουρα ο βέλτιστος αριθμός εξυπηρετητών θα βρίσκεται στο διάστημα 1 μέχρι n. Αυτό που χρειάζεται να κάνουμε είναι για κάθε χρονική στιγμή, ξεκινώντας με την ενδιάμεση τιμή $\lceil {}^{n}/_{2} \rceil$ να ελέγχουμε αν «φτάνουν» οι εξυπηρετητές που έχουμε και αν δεν φτάνουν, με binary search να επαναλαμβάνουμε την διαδικασία μέχρι να φτάσουμε στη βέλτιστη λύση.
Ο αλγόριθμος δηλαδή έχει ως εξής:

- ➔ Ξεκινάμε θέτοντας $s^* = \lceil {}^{n}/_{2} \rceil$
- ➔ Για κάθε χρονική στιγμή, ελέγχουμε αν ισχύει $d < {}^{SoFar[i]}/_{s^*}$. Η δεύτερη ποσότητα στην ανισότητα εκφράζει τον αριθμό χρονικών μονάδων όπου κάθε εξυπηρετητής μένει απασχολημένος. Αν αυτή η ποσότητα είναι μικρότερη από d, σημαίνει ότι οι $SoFar[i]$ πελάτες που έχουν φτάσει μέχρι στιγμής δεν μπορούν να εξυπηρετηθούν και χρειαζόμαστε περισσότερους εξυπηρετητές, συνεπώς εφαρμόζουμε binary search στο δεξί μισό του πίνακα (από ${}^{n}/_{2} + 1$ μέχρι n) και ενημερώνουμε το s^*. Αν όμως είναι $d \geq {}^{SoFar[i]}/_{s^*}$ σημαίνει ότι το s^* που έχουμε διαλέξει αρκεί για την εξυπηρέτηση των i πελατών, συνεπώς το κρατάμε και συνεχίζουμε στα επόμενα βήματα.
- ➔ Τώρα ελέγχουμε την επόμενη χρονική στιγμή όπου έχουμε άφιξη, δηλαδή την στιγμή $A[i+1]$. Αν $A[i+1] \geq A[i] + d$, δηλαδή αν η στιγμή της επόμενης άφιξης, απέχει το πολύ d χρονικές μονάδες από την στιγμή που εξετάσαμε, τότε όλα τα αυτοκίνητα της στιγμής i εξυπηρετούνται με τον αριθμό εξυπηρετητών που έχουμε. Στην αντίθετη περίπτωση εξυπηρετούμε όσα αυτοκίνητα μπορούμε, και τα υπόλοιπα $SoFar[i+1] - SoFar[i]$ που δεν θα εξυπηρετηθούν τα βάζουμε στην αναμονή και θα φροντίσουμε για την εξυπηρέτησή τους στον επόμενο έλεγχο για $d < {}^{SoFar[i]}/_{s^*}$.
- ➔ Επαναλαμβάνουμε τα παραπάνω βήματα (εκτός του πρώτου. Στο πρώτο βήμα το s^*θα είναι αυτό που έχουμε βρει μέχρι στιγμής) για όλες τις χρονικές στιγμές άφιξης και καταλήγουμε στο βέλτιστο s^*.

Απόδειξη Ορθότητας

Ο αλγόριθμος αρχικά θα επιλέξει ένα s^* το οποίο μπορεί και να μην είναι βέλτιστο, διότι η δυαδική αναζήτηση θα τερματίσει και θα επιστρέψει το πρώτο s^* για το οποίο παύει να ισχύει η σχέση $d < {SoFar[i]}/{s^*}$. Ωστόσο στην επόμενη χρονική στιγμή, όπου έρχονται νέα αυτοκίνητα, ο αλγόριθμος θα δοκιμάσει με το ίδιο s^* αν μπορεί να τα εξυπηρετήσει, δηλαδή θα ελέγξει πάλι την σχέση $d < {SoFar[i]}/{s^*}$ για το νέο i. Αν τα αυτοκίνητα εξυπηρετούνται με το ίδιο s^* τότε το κρατάμε και επαναλαμβάνουμε, αλλιώς ξανακάνουμε binary search μέχρι να βρεθεί νέο s^* που εξυπηρετεί τα αυτοκίνητα. Η διαδικασία αυτή συνεχίζεται για όλες τις στιγμές που φτάνει ένα νέο αυτοκίνητο και τελικά το s^* που θα προκύψει στο τέλος συγκλίνει στο βέλτιστο.

Ανάλυση Πολυπλοκότητας

Ο αλγόριθμος αρχικά χρειάζεται χρόνο $O(n)$ για τη δημιουργία των δύο πινάκων. Στη συνέχεια εξετάζει n χρονικές στιγμές και σε κάθε εξέταση, η πιο «βαριά» δουλειά είναι το binary search, όπου στην χειρότερη περίπτωση θα χρειαστεί χρόνο $O(logn)$ για κάθε χρονική στιγμή. Συνεπώς η συνολική πολυπλοκότητα είναι $O(nlogn)$.

Άσκηση 4: Παραλαβή Πακέτων

(1).

Περιγραφή Αλγορίθμου

Ο αναδρομικός αλγόριθμος δοκιμάζει όλες τις πιθανές διατεταγμένες n-άδες για την σειρά εξυπηρέτησης πελατών. Θα λύσουμε το πρόβλημα με greedy τρόπο. Στην «άπληστη» λύση ταξινομούμε τα πηλίκα ${p_i}/{w_i}$ κατά αύξουσα σειρά και επιλέγουμε κάθε φορά το πακέτο με το μικρότερο πηλίκο για εξυπηρέτηση. Χωρίς βλάβη της γενικότητας υποθέτουμε ότι μετά από ταξινόμηση επαναπροσδιορίζουμε τους δείκτες i ώστε να ισχύει ότι ${p_1}/{w_1} \leq {p_2}/{w_2} \leq \cdots \leq {p_n}/{w_n}$.

Απόδειξη Ορθότητας

Ο άπληστος αλγόριθμος επιλέγει πρώτο για εξυπηρέτηση το δέμα του 1ου πελάτη. Ο συνολικός βεβαρυμένος χρόνος εξυπηρέτησης θα ισούται με

$G_1 = w_1p_1 + w_2(p_1 + p_2) + \cdots + w_n(p_1 + p_2 + \cdots p_n)(1).$

Έστω ότι σε κάποιο βήμα, χωρίς βλάβη της γενικότητας στο πρώτο, ο αλγόριθμος δεν επιλέγει το δέμα του 1ου πελάτη αλλά του 2ου. Ο νέος χρόνος θα είναι

$$G_2 = w_2p_2 + w_1(p_1 + p_2) + \cdots + w_n(p_1 + p_2 + \cdots p_n)(2).$$

Αφαιρούμε την (1) από την (2) και παίρνουμε

$G_2 - G_1 = w_1(p_1 + p_2 - p_1) + w_2(p_2 - p_1 - p_2) = w_1p_2 - w_2p_1 \geq 0,$

διότι ισχύει ${p_1}/{w_1} \leq {p_2}/{w_2}$. Συνεπώς, η εναλλαγή αυτή οδηγεί σε αύξηση του χρόνου εξυπηρέτησης και προκύπτει το συμπέρασμα ότι ο άπληστος αλγόριθμος είναι βέλτιστος. Παρόμοια δουλεύουμε για να αποδείξουμε την αύξηση του χρόνου με εναλλαγή οποιονδήποτε δύο διαδοχικών πελατών i, j με ${p_i}/{w_i} \leq {p_j}/{w_j}$ και εξυπηρέτηση πρώτα του j.

Ανάλυση Πολυπλοκότητας

Ο αλγόριθμος αρχικά ταξινομεί τους πελάτες και στη συνέχεια εκτελεί ένα πέρασμα της λίστας n πηλίκων ${}^{p_1}/_{w_1}, {}^{p_2}/_{w_2}, \dots, {}^{p_n}/_{w_n}$. Λόγω της ταξινόμησης η οποία υποθέτουμε ότι γίνεται με τη $mergesort$ συνολική πολυπλοκότητα είναι $O(nlogn)$.

(2).

Περιγραφή Αλγορίθμου

Για αυτό το πρόβλημα, το άπληστο κριτήριο δεν θα δουλέψει, διότι αν και ταξινομούμε κατά αύξουσα σειρά, δεν γνωρίζουμε αν είναι βέλτιστο, το δέμα του πελάτη i να τοποθετηθεί στον 1ο ή στον 2ο εξυπηρετητή. Συνεπώς θα χρησιμοποιήσουμε δυναμικό προγραμματισμό και για κάθε δέμα, θα αποφασίζουμε αν αυτό θα δρομολογηθεί στον 1ο ή στον 2ο εξυπηρετητή.

Ταξινομούμε τα δέματα και πάλι με βάση την προηγούμενη ταξινόμηση και εξετάζουμε το n-οστό, δηλαδή το τελευταίο δέμα. Έστω C ο ελάχιστος βεβαρυμένος χρόνος εξυπηρέτησης. Για το C χρειαζόμαστε δύο παραμέτρους, το πλήθος των δεμάτων και τον μέγιστο χρόνο εξυπηρέτησης στον πρώτο εξυπηρετητή, έστω t_1, μόλις δρομολογηθεί το n-οστό πακέτο. Επίσης ορίζουμε ανάλογα τον μέγιστο χρόνο εξυπηρέτησης στον δεύτερο εξυπηρετητή, t_2. Προφανώς, εφόσον τα t_1, t_2 αποτελούνται από p_i, τότε ισχύει $t_1 + t_2 = \sum_{i=1}^{n} p_i$.

Για το n-οστό αντικείμενο λοιπόν έχουμε 2 επιλογές όπως είπαμε.

- ➔ Θα ανατεθεί στον 1ο εξυπηρετητή, συνεπώς ο ελάχιστος χρόνος θα προκύψει από τον ελάχιστο χρόνο για $n-1$ αντικείμενα με μέγιστο χρόνο εξυπηρέτησης στον 1ο εξυπηρετητή $t_1 - p_n$ και τον βεβαρυμένο χρόνο του n-οστού δέματος, $w_n t_1$.
- ➔ Δεν θα ανατεθεί στον 1ο εξυπηρετητή συνεπώς θα ανατεθεί στον 2ο εξυπηρετητή. Ο ελάχιστος χρόνος θα προκύψει από τον ελάχιστο χρόνο για $n-1$ αντικείμενα με μέγιστο χρόνο εξυπηρέτησης στον 1ο εξυπηρετητή t_1 και τον βεβαρυμένο χρόνο του n-οστού δέματος, $w_n t_2$.

Καταλήγουμε λοιπόν στην ακόλουθη σχέση για τον βεβαρυμένο χρόνο εξυπηρέτησης για n πελάτες:

$$C(n, t_1) = min \begin{cases} C(n-1, t_1 - p_n) + w_n t_1 \\ C(n-1, t_1) + w_n t_2 \end{cases}$$

Η παραπάνω σχέση γενικεύεται εύκολα για οποιοδήποτε πακέτο i και λύνεται χρησιμοποιώντας πίνακα παρόμοιο με αυτόν που χρησιμοποιήθηκε για το διακριτό πρόβλημα σακιδίου που λύσαμε στην τάξη.

Απόδειξη Ορθότητας

Ο αλγόριθμος σε κάθε βήμα αποφασίζει με βέλτιστο τρόπο για το αν το δέμα i τοποθετείται στον 1ο ή στον 2ο εξυπηρετητή, εξετάζοντας όλες τις περιπτώσεις. Συνεπώς, σχηματίζει την βέλτιστη λύση για το πρόβλημα.

Ανάλυση Πολυπλοκότητας

Με βάση τα προηγούμενα η πολυπλοκότητα του αλγορίθμου είναι $O(nt_1)$ και αφού ισχύει $t_1 + t_2 = \sum_{i=1}^{n} p_i$, τελικά είναι $O(n \sum_{i=1}^{n} p_i)$.

Γενίκευση για $m \geq 3$ υπαλλήλους

Τώρα έχουμε $m \geq 3$ υπαλλήλους οπότε για κάθε δέμα έχουμε m αποφάσεις, δηλαδή πρέπει να αποφασίσουμε σε ποιον από τους υπαλλήλους θα δρομολογηθεί το δέμα. Η αναδρομική σχέση για τον συνολικό βεβαρυμένο χρόνο εξυπηρέτησης γενικεύεται ως εξής:

$$C(n, t_1) = min \begin{cases} C(n-1, t_1 - p_n) + w_n t_1 \\ C(n-1, t_1) + w_n t_2 \\ \vdots \\ C(n-1, t_1) + w_n t_m \end{cases}$$

Η πολυπλοκότητα τώρα θα είναι $O(n(\sum_{i=1}^{n} p_i)^{m-1})$.

Άσκηση 5: Ελάχιστη Διαταραχή Ακολουθίας

(α). Έστω ότι το τελευταίο στοιχείο της βέλτιστης ακολουθίας β^* δεν είναι το ελάχιστο αλλά είναι κάποιο στοιχείο t. Θεωρούμε ότι το ελάχιστο βρίσκεται σε θέση $i < n$. Η ακολουθία β^* θα μοιάζει κάπως έτσι: $\dots, min, \dots, t$.

Θεωρούμε την ακολουθία γ^* η οποία είναι ίδια με την β^*, με την διαφορά ότι στη θέση i του ελαχίστου έχει το στοιχείο t, ενώ το ελάχιστο στοιχείο έχει τοποθετηθεί στη θέση του t, δηλαδή στην τελευταία θέση. Η ακολουθία γ^* θα μοιάζει κάπως έτσι: $\dots, t, \dots, min$.

Μέχρι την θέση $i-1$ τα στοιχεία των δύο ακολουθιών είναι ίδια οπότε η «συνεισφορά» στην ελάχιστη διαταραχή μέχρι την θέση $i-1$ είναι ίδια και για τις δύο ακολουθίες. Για τα στοιχεία από i μέχρι n ισχύουν τα εξής:

- Στην ακολουθία β^*, η συνολική διαταραχή κάθε προθέματος θα προκύπτει ως η αφαίρεση του min από το μέγιστο στοιχείο σε αυτές τις θέσεις, έστω m.
- Στην ακολουθία γ^*, η συνολική διαταραχή κάθε προθέματος θα προκύπτει ως η αφαίρεση κάποιου στοιχείου $x \neq min$ από το m.

Με βάση τις δύο παραπάνω παρατηρήσεις, αφού θα είναι πάντα $min < x$ τότε στο τμήμα i μέχρι n η συνεισφορά στην συνολική διαταραχή είναι μικρότερη για την ακολουθία γ^*. Καταλήγουμε σε άτοπο, διότι υποθέσαμε αρχικά ότι η ακολουθία β^* είναι βέλτιστη.

Για την παραπάνω διαδικασία υποθέσαμε ότι το τελευταίο στοιχείο της β^* δεν είναι το ελάχιστο. Τώρα υποθέτουμε ότι το τελευταίο στοιχείο της β^* δεν είναι το μέγιστο και με συμμετρικό με τον παραπάνω τρόπο αποδεικνύουμε ότι τότε η β^* δεν είναι βέλτιστη και καταλήγουμε πάλι σε άτοπο. Η διαφορά τώρα είναι ότι στην ακολουθία β^* οι συνεισφορές μετά από την θέση i θα είναι $max - \text{ελάχιστο μέχρι την τρέχουσα θέση}$ ενώ στην νέα ακολουθία γ^* που θα σχηματιστεί θα είναι $x - \text{ελάχιστο μέχρι την τρέχουσα θέση}$, και αφού $max > x$ οι συνεισφορές στην γ^* είναι μικρότερες.

(β).

Περιγραφή Αλγορίθμου

Από το προηγούμενο ερώτημα γνωρίζουμε ότι στο τέλος της βέλτιστης ακολουθίας θα τοποθετηθεί είτε το μέγιστο είτε το ελάχιστο στοιχείο της αρχικής ακολουθίας, συνεπώς πρέπει να εκμεταλλευτούμε αυτό το «άπλητο» κριτήριο για τη λύση μας. Θα χρησιμοποιήσουμε για τη λύση μας δυναμικό προγραμματισμό.

Ξεκινώντας, ταξινομούμε την αρχική ακολουθία σε χρόνο $O(nlogn)$ και δημιουργούμε μια νέα λίστα

$$a_{sorted} = a_1, a_2, \dots, a_n$$

όπου τα a_i βρίσκονται σε αύξουσα σειρά.

Θεωρούμε δύο δείκτες i,j πάνω στην ακολουθία a_{sorted} όπου ο δείκτης i δείχνει στο ελάχιστο στοιχείο της βέλτιστης λύσης και ο δείκτης j στο μέγιστο (προφανώς θα είναι $i \leq j$ αφού η a_{sorted} είναι ταξινομημένη). Η ελάχιστη συνολική διαταραχή D θα έχει προφανώς $i = 1$ και $j = n$, αφού η βέλτιστη ακολουθία έχει ελάχιστο το στοιχείο a_1 της a_{sorted} και μέγιστο το a_n. Προσπαθώντας να εκφράσουμε αναδρομικά το ελάχιστο κόστος παρατηρούμε ότι θα προκύψει μέσα από δύο επιλογές:

- ➔ Είτε η βέλτιστη ακολουθία θα έχει στο τέλος το μέγιστο στοιχείο, συνεπώς το ελάχιστο κόστος θα προκύψει από αυτό της ακολουθίας που έχει μέγιστο το a_{n-1} της a_{sorted} και ελάχιστο το a_1.
- ➔ Είτε η βέλτιστη ακολουθία θα έχει στο τέλος το ελάχιστο στοιχείο, συνεπώς το ελάχιστο κόστος θα προκύψει από αυτό της ακολουθίας που έχει μέγιστο το a_n της a_{sorted} και ελάχιστο το a_2.
- ➔ Προφανώς για να σχηματιστεί το ελάχιστο κόστος, προσθέτουμε και στις δύο περιπτώσεις την διαταραχή του προθέματος που περιλαμβάνει όλο την ακολουθία, δηλαδή την ποσότητα $a_n - a_1$.

Σχηματίζουμε την ακόλουθη αναδρομική σχέση, η οποία περιγράφει τα παραπάνω:

$$D(1,n) = \ min\begin{cases} D(1,n-1) + \ a_n - a_1 \\ D(2,n) + a_n - a_1 \end{cases}$$

Για να δουλέψει ο δυναμικός προγραμματισμός και να σχηματιστούν όλες οι βέλτιστες «υπολύσεις», γενικεύουμε την παραπάνω σχέση για κάθε ζεύγος i,j όπου $i \leq j$:

$$D(i,j) = min\begin{cases} D(i,j-1) + \ a_j - a_i \\ D(i+1,j) + a_j - a_i \end{cases}$$

Ο πίνακας δυναμικού προγραμματισμού που θα σχηματιστεί θα έχει προφανώς μηδενικά στις διαγώνιους, διότι για $i = j$ θα ισχύει $a_j - a_i = 0$ ενώ δεν θα έχει στοιχεία για $i < j$. Συνεπώς το μέγεθός του θα είναι $O\left({}^{n^2}/_{2}\right) = O(n^2)$ όσο και η πολυπλοκότητα ανεύρεσης της ελάχιστης συνολικής διαταραχής.

Σε αυτό το σημείο έχουμε υπολογίσει την ελάχιστη συνολική διαταραχή, αλλά δεν έχουμε υπολογίσει την βέλτιστη ακολουθία, δηλαδή αυτή που δίνει αυτή τη διαταραχή. Για να το κάνουμε αυτό, ξεκινάμε από το ζεύγος i,j που δίνει τη βέλτιστη λύση, δηλαδή $i = 1$ και $j = n$, εξετάζουμε όλα τα υπόλοιπα ζεύγη από τα οποία είναι δυνατόν να φτάσαμε στο ζεύγος i,j και κρατάμε το ζεύγος για το οποίο το D είναι ελάχιστο. Σημειώνουμε ότι συμπληρώνοντας τον πίνακα δυναμικού προγραμματισμού, κρατούσαμε σε κάθε «κουτί» την ελάχιστη συνολική διαταραχή, αλλά και αν στο τέλος μπήκε το μέγιστο ή το ελάχιστο, έτσι ώστε να «χτίσουμε» την βέλτιστη ακολουθία από το τέλος προς την αρχή. Συνεχίζουμε αυτή την διαδικασία για κάθε ζεύγος i,j. Η πολυπλοκότητα για αυτή τη διαδικασία είναι $O(n^2)$.

<u>Απόδειξη Ορθότητας</u>

Ο αλγόριθμος αξιοποιεί την ιδιότητα που αποδείξαμε στο προηγούμενο ερώτημα και ουσιαστικά δοκιμάζει όλα τα πιθανά ζευγάρια μεγίστου-ελαχίστου, τοποθετώντας στο τέλος της ακολουθίας είτε το μέγιστο είτε το ελάχιστο και χτίζοντας με αυτόν τον τρόπο τη βέλτιστη λύση.

<u>Ανάλυση Πολυπλοκότητας</u>

Με βάση τα παραπάνω η συνολική πολυπλοκότητα του αλγορίθμου είναι $O(n^2)$.