ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Αλγόριθμοι και Πολυπλοκότητα

1η Σειρά Γραπτών Ασκήσεων

Ανδρέας Χατζησάββας 03118701

Άσκηση 1: Πλησιέστερο Ζεύγος Σημείων

(α)

Θα χρησιμοποιηθεί η ίδια ιδέα που χρησιμοποιήθηκε για το 2D πρόβλημα.\
Ταξινομούμε τα σημεία στην x-διάσταση.
Θα διαιρέσουμε(devide) με βάση τη συντεταγμένη x με ένα επίπεδο το χώρο ώστε τα μισά σημεία να βρίσκονται στο αριστερό μέρος και τα άλλα μισά στο δεξί.
Θα βρούμε αναδρομικά(recursively) το κοντινότερο ζεύγος σε κάθε πλευρά και ορίζουμε δ τη μικρότερη απόσταση ζευγαριών και απο τις δύο πλευρές(conquer).
Μένει να ελέγξουμε τα ζεύγη που ένα σημείο βρίσκεται στην μια πλευρά και το άλλο στην άλλη.
Αντί για λωρίδα(strip) με πλάτος 2δ θα έχουμε πλάκα(slab) με πλάτος 2δ και άπειρο ύψος και βάθος(στις y και z διαστάσεις). Χωρίζουμε την πλάκα σε κύβους διαστάσεων δ/2 ώστε κανένας κύβος να μην διαπερνα το x0.
Στο 2D επίπεδο κάθετο στον άξονα x προβάλουμε όλα τα σημεία που βρίσκονται εντός της πλάκας.
Εξετάζουμε μόνο τα ζεύγη των οποίων οι προβολές βρίσκονται σε απόσταση το πολύ δ και τώρα εφαρμόζουμε τον αλγόριθμο 2D.
Ο αλγόριθμος χωρίζει τα σημεία σε 2 μέρη και στη συνέχεια τα ενώνει(merge). Άρα έχουμε πολυπλοκότητα $T(n) = 2T(n/2) + \Theta(n\log n)$ και απο το Master Theorem καταλήγουμε στο $T(n) = \Theta(n\log^2 n)$.

(β)

Διαλέγω μια αρχική τιμή για το δ* ως προς ℓ.
Χωρίζω τον χώρο σε κουτιά διαστάσεων δ* με τέτοιο τρόπο ώστε κάθε κουτί να περιέχει ένα υποσύνολο απο σημεία και κάνω sort τα σημεία με βάση τη συντεταγμένη x.
Για κάθε σημείο συγκρίνω με τα γειτονικά του εντός του ίδιου κουτιού και των γειτονικών κουτιών. Αφού τα σημεία είναι sorted χρειάζεται μόνο να ελέγξω τα διαδοχικά για να βρω το κοντινότερο ζεύγος.
Κρατάω την μικρότερη απόσταση(minimum_distance) που βρίσκω κάθε φορα κατα την διαδικασία και αν είναι μικρότερη από το δ* αναβαθμίζω την τιμή του δ*.
Στο τέλος έχουμε στο minimum_distance την μικρότερη απόσταση.

Σε $d \geq 2$ διαστάσεις:

Στον διαχωρισμό του χώρου με d διαστάσεις τα κουτιά είναι d-διάστατα αντι για 2d.
Κάθε πλευρά των κουτιών έχει δ* διαστάσεις και αυτό εφαρμόζεται σε $d \geq 2$ διαστάσεις.

Ορθότητα:

Ο αλγόριθμος είναι σωστός γιατι μέσο του διαχωρισμού του χώρου σε κουτιά εξετάζει όλα τα σημεία μέσα σε απόσταση εύρους ζώνης $[\ell, c\ell]$. Έτσι δεν μπορεί να “χάσει το κοντινότερο ζεύγος.
Αναβαθμίζει την τιμή του δ* αν βρεθεί ζεύγος με απόσταση μικρότερη του δ*. Έτσι το σωστό δ* θα εντοπιστεί.
Εξετάζονται όλα τα ζεύγη σημείων μέσα στα κουτιά και στα γειτονικά άρα είναι εξαντλητικός.

Υπολογιστική πολυπλοκότητα:

Ο αλγόριθμος έχει O(n) πολυπλοκότητα γιατι επισκεπτόμαστε κάθε σημείο 1 φορά για ένα σύνολο πράξεων και εξαρτάται επίσης απο την διάσταση d

Άσκηση 2: Πόρτες Ασφαλείας στο Κάστρο

Ξεκινάμε θέτοντας όλους τους διακόπτες στην ίδια θέση(πχ κάτω θέση).

Μέχρι να βρούμε όλες τις αντιστοιχίες πόρτα-διακόπτη εκτελούμε τα παρακάτω:

2.1 Βρίσκουμε την πρώτη πόρτα στο διάδρομο που παραμένει κλειστή. Αλλάζουμε την θέση στους πρώτους μισούς διακόπτες.

2.2 Αν υπάρχει μόνο ένας διακόπτης για έλεγχο αλλάζουμε τη θέση του και ψάχνουμε την πόρτα που άλλαξε κατάσταση και την αντιστοιχούμε με το διακόπτη. Κρατάμε την πόρτα ανοιχτή και αφαιρούμε από το σύνολο των διακόπτη που γνωρίζουμε πλέον την αντιστοιχη πόρτα του. Αν δεν υπάρχουν άλλες αντιστοιχίες που εκκρεμούν τελειώσαμε.

2.3 Αλλάζουμε την θέση τους πρώτους μισούς διακόπτες και ψάχνουμε την πρώτη κλειστή πόρτα.

2.4 Αν η πρώτη κλειστή πόρτα είναι διαφορετική από το βήμα 2.1. Σημαίνει πως ο διακόπτης αυτής βρίσκεται στο μισό που αλλάξαμε. Επαναλαμβάνουμε το βήμα 2.2 λαμβάνοντας υπόψην μόνο αυτούς που αλλάξαμε.

2.5 Αν η πρώτη κλειστή πόρτα παραμένει η ίδια αυτό σημαίνει πως ο διακόπτης αυτής βρίσκεται στο μισό που δεν αλλάξαμε. Επαναλαμβάνουμε το βήμα 2.2 λαμβάνοντας υπόψην μόνο αυτούς που δεν αλλάξαμε.

2.6 Αν όλες οι πόρτες είναι ανοιχτές τότε δοκιμάζουμε ένα ένα τους διακόπτες για να δούμε ποια πόρτα κλείνει κάθε φορα.

Η λογική πίσω από τον αλγόριθμο χρησιμοποιεί την δυαδική αναζήτηση. Ψάχνουμε την αντιστοιχία μιας κλειστης πόρτα. Αναζητούμε τον διακόπτη της μεταβάλλοντας τους μισούς διακόπτες. Αν μετά την αλλαγή η πόρτα αλλάξει κατάσταση σημαίνει πως ο διακόπτης της βρίσκεται στο μισό των διακοπτών που αλλάξαμε, αν όχι τότε ο διακόπτης της βρίσκεται στο άλλο μισό των διακοπτών που δεν αλλάξαμε την κατάσταση τους.
Για μια πόρτα θέλουμε O(logn) για να βρούμε την λύση της(binary search) και για n πόρτες θέλουμε O(nlogn).

Άσκηση 3: Κρυμμένος Θησαυρός

Θέτουμε ως d την απόσταση στην οποία βρίσκεται ο θησαυρος απο εμάς.
Στον αλγόριθμο τον οποίο θα ακολουθήσουμε έχουμε τις αποστάσεις $f_0, f_1, f_2, \ldots$, όπου $f_{i \bmod 2}$ αναπαριστά την απόσταση από την αρχική μας θέση που έχουμε ταξιδέψει στην κατεύθυνση $P_{i \bmod 2}$(Όπου P1 η κατεύθυνση στα θετικά και P0 η κατεύθυνση στα αρνητικά).
Ουσιαστικά ταξιδεύουμε προς την μια κατεύθυνση και έπειτα ταξιδεύουμε στην αντίθετη αλλά θέλουμε $f_i < f_{i+2}$ ώστε κάθε φορά που ξανα ταξιδεύουμε σε μια κατεύθυνση να διανύουμε μεγαλύτερη απόσταση από την τελευταία φορα.
Για αυτό επιλέγουμε $f_i = 2^{i+1}$ for i = 1, 2, 3, ...
Στην χειρότερη περίπτωση $d = f_i + \epsilon$, για κάποιο μικρό ϵ και κάποιο $i \geq 1$ όπου περπατάμε στην κατεύθυνση που βρίσκεται ο θησαυρός στο βήμα i αλλά τον χάνουμε για απόσταση ϵ. Τότε στο επόμενο βήμα περπατάμε στην αντίθετη κατεύθυνση και στο βήμα i+2 επιστρέφουμε

ξανα στην σωστή κατεύθυνση και βρίσκουμε τον θησαυρό. Άρα η απόσταση που περπατάμε είναι:

$$2 * \sum_{j=1}^{i+1} fi \; + \; d \; = \; 2 * \sum_{j=2}^{i+2} 2^{j} \; + \; d$$

$$= \; 2 * (2^{i+3} - 3) \; + \; 2^{i+1} \; + \; \epsilon$$

$$= \; 9 * \; 2^{i+1} - 6 \; + \; \epsilon$$

$$= \; 9 * \; f_{i} \; + \; \epsilon \; - 6$$

$$\leq \; 9 * (f_{i} \; + \; \epsilon) = 9d$$

Άσκηση 4: Μη Επικαλυπτόμενα Διαστήματα Μέγιστου Συνολικού Μήκους

1.1) Άπληστος αλγόριθμος επιλογής μέγιστου μήκους $f_i - s_i$ σε κάθε επανάληψη:

Το αντιπαράδειγμα που θα χρησιμοποιήσω είναι αυτό του παραδείγματος που δίνεται όπου n=5 και τα διαστήματα είναι [1, 3), [2, 6), [4, 7), [5, 8), [7, 8) .

Στην πρώτη επανάληψη θα επιλεγεί το διάστημα [2,6) με διάστημα 4 και στην δεύτερη επανάληψη θα επιλεγεί το διάστημα [7,8) με διάστημα 1 και θα τερματίσει. Το συνολικό διάστημα είναι 5 το οποίο είναι μικρότερο από το 6(δηλαδη το βέλτιστο).

1.2) Άπληστος αλγόριθμος επιλογής ελάχιστου χρόνου ολοκλήρωσης f_i σε κάθε επανάληψη:

Το αντιπαράδειγμα που θα χρησιμοποιήσω είναι το [1,3), [4,7), [5,8), [7,8), [2,10).

Στην πρώτη επανάληψη θα επιλεγεί το διάστημα [1,3), στην δεύτερη επανάληψη το διάστημα [4,7) και στην τρίτη επανάληψη θα επιλεγεί το [7,8) με συνολικό διάστημα 6. Το συνολικό διάστημα είναι μικρότερο από το βέλτιστο το οποίο είναι το [2,10) με συνολικό διάστημα 8.

2) Για την επίλυση του προβλήματος θα χρησιμοποιήσουμε δυναμικό αλγόριθμο.

Αρχικά θα κάνουμε sort τα διαστήματα με βάση τον χρόνο ολοκλήρωσης τους.

Στον πίνακα T[i] με μέγεθος n(=σύνολο διαστημάτων) θα αποθηκεύουμε τις καλύτερες λύσεις για τα διαστήματα που έχουν ελεγχθεί μέχρι το διάστημα i.

Έστω profit[i] είναι το κέρδος από το διάστημα i και η συνάρτηση dontoverlap(i,j) ελέγχει αν δύο διαστήματα ειναι μη επικαλυπτόμενα.

Ο αλγόριθμος είναι ο εξής:

```
for i=1 to n:
        for j=0 to i-1:
                if (dontoverlap(i,j)):
                        T[i] = max(T[i], T[j] + profit[i])
```

Η επανάληψη γίνεται n φορές και σε κάθε step μπορούμε να βρούμε το j χρησιμοποιώντας binary search κάνοντας την πολυπλοκότητα 0(nlogn).

Άσκηση 5: Παραλαβή Πακέτων

1. Μόνο ένας υπάλληλος:

   Η ταξινόμηση των πελατών θα γίνει με βάση την φθίνουσα ακολουθία των θετικών βαρών των πελατών ανά τον χρόνο εξυπηρέτησης τους wi / pi χρησιμοποιώντας τον αλγόριθμο ταξινόμησης radix sort και σε χρόνο O(n).

Ο αλγόριθμος έχει πολυπλοκότητα O(n) .
Αποδεικνύουμε την ορθότητα του αλγορίθμου με το εξής παράδειγμα. Ας υποθέσουμε ότι ο βέλτιστος αλγόριθμος δεν είναι ο radix sort τότε θα υπάρξουν έστω 2 πελάτες i και j οι οποίοι εξυπηρετούνται πρώτα ο i και αμέσως μετά ο j και ισχύει wi/pi < wj/pj. Συμβολίζουμε με t τον χρόνο εξυπηρέτησης πριν εξυπηρετηθούν οι 2 πελάτες και T τον βεβαρημένο χρόνο. Ο χρόνος εξυπηρέτησης του i θα είναι t0i = t + pi ενώ ο χρόνος εξυπηρέτησης του j θα είναι t0j = t + ti + tj. Ο συνολικός βεβαρημένος χρόνος θα είναι: T0 = T + wi(t+pi) + wj(t+pi+pj) Αν αντιστρέψουμε την σειρά των δύο πελατών τότε ο χρόνος εξυπηρέτησης του j θα είναι t1j = t + pj και του i θα είναι t1i = t + pj + pi. Ο συνολικός βεβαρημένος χρόνος θα είναι:
T1= T + wj(t+pj) + wi(t+pj+pi) Η αλλαγή της σειράς των δύο πελατών δεν επηρεάζει τον βεβαρημένο χρόνο των υπολοίπων πελατών.
Έχουμε T0 – T1 = wjpi – wipj = pipj (wj/pj – wi/pi) >0 => T0 > T1 κάτι το οποίο είναι άτοπο αφού T0 είναι η βέλτιστη λύση και άρα πρέπει να χρησιμοποιείται radix sort.

2. Δύο υπάλληλοι:
   Σε αυτή τη περίπτωση θα πρέπει να χρησιμοποιήσουμε δυναμικό προγραμματισμό. Πρώτα ταξινομούμε τους πελάτες όπως στο προηγούμενο ερώτημα. Έτσι προκύπτει η αναδρομική εξίσωση που περιγράφει την τιμή της βέλτιστης λύσης:

$$T(i,t_1) = min\begin{cases} T(i-1,t_1-p_i) + w_i t_1 \\ T(i-1,t_1) + w_i(\sum_{j=1}^{i} p_j - t_1) \end{cases}$$

   όπου t_1 είναι το άθροισμα των χρόνων των πελατών που εξυπηρετούνται απο τον πρώτο υπάλληλο. Σε περίπτωση περισσότερων υπαλλήλων θα εκφράζαμε τον χρόνο αυτό ως συνάρτηση περισσότερων παραμέτρων όπως $T(t_1, t_2, t_3, \ldots)$.