Αλγόριθμοι και Πολυπλοκότητα
1η Σειρά Ασκήσεων

Παπαδόπουλος Χαράλαμπος

03120199

Άσκηση 1:

Για κάθε θέση θα επιχειρήσουμε, μέσω στοίβας, να αξιοποιήσουμε την «προεργασία» της προηγούμενης θέσης. Δηλαδή, για μια θέση i θα ξεκινάμε ελέγχοντας την κυρίαρχη θέση της i-1 κ.ο.κ.

```
void dominant_positions(int A[], int n, int result[]) {
        stack<int> s;

        dominant[0] = 0;
        s.push(0);

        for (int i = 1; i < n; i++) {
                while(!s.empty() && A[s.top()] <= A[i]) {
                        s.pop();
                }
                if(s.empty()) result[i] = 0;
                else result[i] = s.top;
                s.push(i);
        }
}
```

Πρακτικά η στοίβα διατηρεί τις «υποψήφιες» κυρίαρχες θέσεις ανά πάσα στιγμή και αν είναι άδεια αυτό σημαίνει ότι δεν υπάρχει κυρίαρχη θέση στον πίνακα.

Ένα στοιχείο προστίθεται στην στοίβα μόνο αφού ελεγχθεί και αν γίνει pop δεν ξαναμπαίνει. Δηλαδή, κάθε θέση μπορεί να ελεγχθεί το πολύ μία φορά, το οποίο μας οδηγεί σε πολυπλοκότητα Θ(n).

Άσκηση 2:

α) Θα χρησιμοποιήσουμε δυαδική αναζήτηση. Γνωρίζοντας την Μέγιστη τιμή Μ θα κάνουμε κάθε φορά δυαδική αναζήτηση στο μέσο, δηλαδή $\frac{high+low}{2}$, αρχικοποιώντας high = M, low = 1. Έπειτα, σε κάθε επανάληψη θα συγκρίνουμε Fs(mid) με το k. Έχουμε λοιπόν δύο περιπτώσεις.

I) Αν Fs(mid) >= k, τότε high = mid

II) Αν Fs(mid) < k, τότε low = mid + 1

Και επαναλβάνουμε έως ότου low = high = k-οστό μικρότερο στοιχείο.

Ορθότητα:

Έχουμε ότι $\forall x1, x2 \rightarrow Fs(x1) \leq Fs(x2)$, αυτό σημαίνει ότι μπορούμε να εφαρμόσουμε δυαδική αναζήτηση.

Πολυπλοκότητα:

Ως απλή εφαρμογή της δυαδικής αναζήτησης σε διάστημα μεγέθους Μ έχουμε $O(logM)$.

β) Αρχικά θα ταξινομήσουμε το σύνολο Α σε χρόνο O(nlogn).

Έπειτα, θα βρούμε την κατανομή Fs(x) με τον εξής τρόπο:
Για κάθε στοιχείο A[i] κάνουμε δυαδική αναζήτηση για να βρούμε το μέγιστο A[j] για το οποίο ισχύει $A[i] - A[j] \leq x$.

Για το τελικό αποτέλεσμα θα εφαρμόσουμε την Fs(x) όπως προηγουμένως.

Ορθότητα:

Βασιζόμαστε και πάλι στην ιδιότητα της Fs ως μη-φθίνουσα συνάρτηση.

Πολυπλοκότητα:

Για κάθε στοιχείο πρέπει αρχικά να κατασκευάσουμε την Fs που είναι logn, άρα συνολικά nlogn, και όπως είδαμε προηγουμένως η εφαρμογή της Fs απαιτεί logM. Συνολικά (μαζί με τον χρόνο της αρχικής ταξινόμησης) έχουμε: $O(n * logn + n * logn * logM \approx n * logn * logM$

Άσκηση 3:

α) Θα χρησιμοποιήσουμε δυναμικό προγραμματισμό με έναν πίνακα ο οποίος πέρα από το άθροισμα θα αποθηκεύει και πόσα στοιχεία χρησιμοποίησα.

S(i, b, l).

i: τα πρώτα στοιχεία που χρησιμοποιούμε

l: το πλήθος στοιχείων που έχουμε χρησιμοποιήσει

b: το άθροισμα των l στοιχείων

Άρα, η αναδρομική σχέση είναι της μορφής:

$$\boldsymbol{S(i,b,l)} = \begin{cases} \mathbf{1}, & \boldsymbol{b = 0} \text{ ή } \boldsymbol{l = 0} \\ \mathbf{0}, & \boldsymbol{b < 0} \text{ ή } \boldsymbol{l < 0} \\ \mathbf{0}, & \boldsymbol{i = 0 \ και \ b > 0} \text{ ή } \boldsymbol{l = 0 \ και \ b > 0} \\ \boldsymbol{S(i-1,b,l) + S(i-1,b-s,l-1)}, & \boldsymbol{i = [1,n]}, \quad \boldsymbol{b = [1,B]}, \quad \boldsymbol{l = [1,L]} \end{cases}$$

Η απάντηση μας είναι S(n, B, L).

Πολυπλοκότητα

Ως εφαρμογή δυναμικού προγραμματισμού η πολυπλοκότητα θα είναι

$$O(n * B * L)$$

Άσκηση 4:

α)

1. Έστω τα διαστήματα [1, 3), [2, 7), [5, 9). Η βέλτιστη επιλογή είναι 6 ενώ με το κριτήριο άπληστου της εκφώνησης βρίσκουμε 5.

2. Έστω τα διαστήματα [1, 3), [2, 10). Η βέλτιστη επιλογή είναι 8 ενώ με το κριτήριο άπληστου της εκφώνησης βρίσκουμε 2.

β)

Αρχικά θα ταξινομήσουμε τον πίνακα βάσει του χρόνου περαίωσης f_i και εφαρμόζουμε δυναμικό προγραμματισμό. Η αναδρομική μας σχέση θα είναι:

$$OPT(j) = \begin{cases} 0, & j = 0 \\ \max\{OPT(j-1), (OPT(j) + u_i)\} \end{cases}$$

Όπου p(j) είναι ο μεγαλύτερος δείκτης i < j έτσι ώστε τα διαστήματα i και j να είναι ξένα, δηλαδή αμέσως προηγούμενο μη επικαλυπτόμενο διάστημα.

Πολυπλοκότητα:

Έχουμε μια αρχική ταξινόμηση με quicksort σε $\mathrm{O}(\mathrm{n} * \log\mathrm{n})$.

Έπειτα για την εύρεση των p(j) θα πραγματοποιήσουμε δυαδική αναζήτηση σε $\mathrm{O}(\mathrm{n} * \log\mathrm{n})$ για να βρούμε $f_j \leq s_i$

Για τον υπολογισμό του OPT(n) έχουμε χρόνο O(n) [Kleinberg & Tardos, σχέση 6.4]

Τελικά, έχουμε

$$O(n * logn + n * logn + n) \cong O(n * logn)$$

Άσκηση 5:

α) Αρχικά θα ταξινομήσουμε τα μεγέθη κατά d_i με quicksort σε O(n*logn) και θα εφαρμόσουμε δυναμικό προγραμματισμό

$$S(l) = \begin{cases} 0, & l = 0 \\ \max\{S(l-1), \max\limits_{i:d_i \le l-1}(S(l-1)+u_i)\} \end{cases}$$

Συγκεκριμένα ξεκινάμε από την τελευταία θέση του πίνακα και ελέγχουμε ποια είναι η βέλτιστη επιλογή που μπορεί να γίνει βάσει των διαθέσιμων αυτοκινήτων.

Πολυπλοκότητα:

Η αρχική ταξινόμηση απαιτεί O(n*logn).

Ύστερα για κάθε θέση μπορεί να κάνουμε έως και n ελέγχους, δηλαδή O(n * L).

Συνολικά έχουμε,

$$O(n * logn + n * L)$$

β) Θα ακολουθήσουμε ίδια ακριβώς λογική με πριν, μόνο που τώρα δεν έχουμε αμάξια μήκους 1, αλλά s_i, οπότε:

$$S(l) = \begin{cases} 0, & l = 0 \\ \max\{S(l-1), \max\limits_{i:d_i \le l-s_i}(S(l-s_i)+u_i)\} \end{cases}$$

Αξίζει να σημειωθεί πως το σκέλος της αναδρομικής σχέσης που εκφράζει την περίπτωση που δεν επιλέγουμε κανένα αυτοκίνητο (δηλαδή το S(l – 1)) σημαίνει πως δεν υπήρχε κανένα διαθέσιμο αυτοκίνητο για την συγκεκριμένη θέση. Οπότε, παρότι τα αυτοκίνητα δεν έχουν καθορισμένο μήκος, εμείς πάλι θα προχωρήσουμε με βήμα 1, πηγαίνοντας δηλαδή στο «αμέσως προηγούμενο» di.

Πολυπλοκότητα:

Θα έχουμε πάλι

$$O(n * logn + n * L)$$