Φροντιστήριο 3

Άσκηση 1

Ένα d-διάστατο 'κουτί', διαστάσεων (x[1],…,x[d]) ***χωρεί*** σε κάποιο άλλο κουτί, διαστάσεων (y[1],…,y[d]) αν υπάρχει κάποια μετάθεση π του συνόλου {1,…d}, για την οποία x[π(1)] < y[1], …, x[π(d)]<y[d].

a. Να σχεδιασθεί αλγόριθμος ο οποίος με δεδομένα εισόδου δύο d-διάστατα κουτιά, αποφασίζει αν το ένα χωρεί μέσα στο άλλο.

b. Μας δίδονται n d-διάστατα κουτιά

$$B_1,...,B_n$$

Να σχεδιασθεί αλγόριθμος ο οποίος βρίσκει και επιστρέφει τη μεγαλύτερη ακολουθία κουτιών

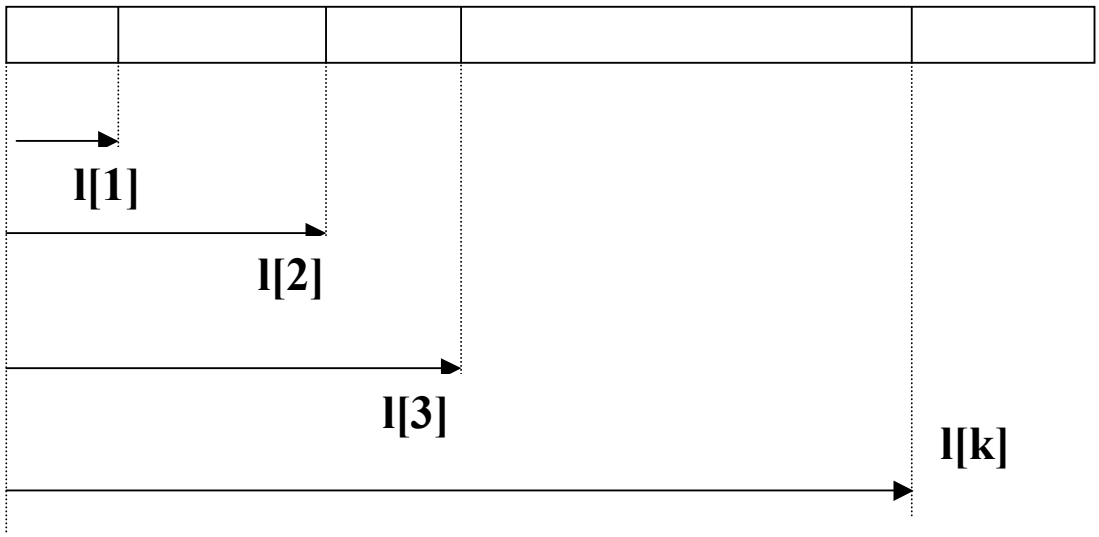
$$B_{i_1},...,B_{i_k}$$

για την οποία το κουτί B_{i_m} χωρεί μέσα στο $B_{i_{m+1}}$ για κάθε $1 \le j \le k-1$.

Να αναλυθεί η χρονική πολυπλοκότητα του αλγόριθμού σας ως προς n και d.

Άσκηση 2

Έχετε ένα καυσόξυλο μήκους l το οποίο θέλετε να τεμαχισθεί σε k+1 κομμάτια ως εξής: οι θέσεις κατατεμαχισμού να απέχουν κατά l[1], l[2], …, l[k], από το αριστερό άκρο το ξύλου, όπου $0 < l[1] < l[2] < \ldots < l[k] < l$.



Παίρνετε το ξύλο σε ένα ξυλουργό, ο οποίος για κάθε κοπή στα δύο ενός καυσόξυλου μήκους X, χρεώνει £X. Ως ποια ακολουθία κοπών θα πρέπει να γίνει ο κατατεμαχισμός ώστε να ελαχιστοποιείται το κόστος του ενώ να ικανοποιούνται οι προδιαγραφές του;

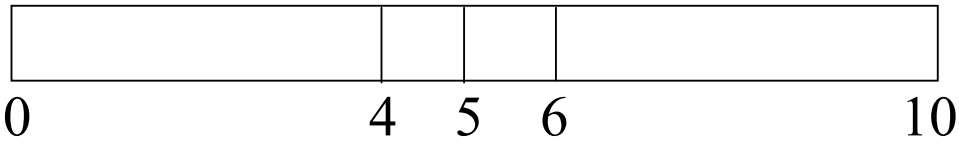
a. Αποφασίστε αν ο πιο κάτω άπληστος αλγόριθμος λύνει το πρόβλημα:

Κόψε σε εκείνη τη θέση για την οποία το μεγαλύτερο από τα δύο κομμάτια είναι το ελάχιστο δυνατό και συνέχισε αναδρομικά.

b. Να δώσετε ένα αλγόριθμο δυναμικού προγραμματισμού ο οποίος επιλύνει το πρόβλημα.

c. Να συζητήσετε την καταλληλότητα χρήσης αλγόριθμου οπισθοδρόμησης για λύση του προβλήματος.

a. Αντιπαράδειγμα:



Άπληστος Αλγόριθμος
Κοπή: 5,4,6
Κόστος: 10 + 5 + 5 = 20
Εναλλακτική λύση
Κοπή: 6,4,5
Κόστος: 10 + 6 + 2 =18

b. Θέτουμε l[0]=0 και l[k+1]=l.
Έστω c[i,j] το ελάχιστο κόστος για κατατεμαχισμό του υποκαυσοξύλου του οποίου του αριστερό άκρο απέχει l[i] και το δεξί του άκρο απέχει l[j] από το αριστερό άκρο του ξύλου.

Τότε

$$c[i,j]=\begin{cases}(l[j]-l[i])+ \\ \quad \min_{i<m<j}(c[i,m]+c[m,j]), & \text{if } j>i+1 \\ 0, & \text{if } j\le i+1\end{cases}$$

Το ζητούμενο είναι το c[0,k+1] το οποίο υπολογίζουμε από τα κάτω προς τα πάνω:

```
for (m=1; m≤k; m++)
    for (i=0; i≤k+1-m; i++)
       c[i,i+m]=…
```

Ο αλγόριθμος λύνει το κάθε υποπρόβλημα μόνο μια φορά, καταχωρεί τη λύση του σε ένα πίνακα από όπου το χρησιμοποιεί κάθε φορά που το υποπρόβλημα καλείται από ένα μεγαλύτερο υποπρόβλημα.

Χρόνος Εκτέλεσης:
Υπάρχουν τόσα υποπροβλήματα, όσα και τα ζεύγη (i,j), 0≤i<j≤k+1:

$$\binom{k+2}{2} = \frac{(k+1)(k+2)}{2} \in \Theta(k^2)$$

Για κάθε υποπρόβλημα χρειαζόμαστε χρόνο:

Συνολικός χρόνος: