Κατ'οίκον Εργασία 1 – Σκελετοί Λύσεων

1. (α) Το *m*-οστό στοιχείο βρίσκεται στη θέση $A[(m+k) \bmod n]$.

   (β) Μπορούμε να υπολογίσουμε τη θέση του πρώτου στοιχείο (και επομένως τον αριθμό *k*) με τον πιο κάτω αλγόριθμο διαίρει και βασίλευε Στη συνέχεια εφαρμόζουμε τον αλγόριθμο από το μέρος (α).

```
Έστω ο πίνακας A[l, r]
   mid = (l+r)/2
   Αν A[l]≤ A[mid] ≤ A[r]
        επέστρεψε το l
   Διαφορετικά αν A[l]>A[mid]
        κάλεσε αναδρομικά τη διαδικασία
        στον πίνακα A[l, mid]
   διαφορετικά
        κάλεσε αναδρομικά τη διαδικασία
        στον πίνακα A[mid+1, r]
```

   Ο χρόνος εκτέλεσης χείριστης περίπτωσης του αλγορίθμου δίνεται από την αναδρομική διαδικασία

   $T(n) = T(n/2) + 1$

   λύση της οποίας μας δίνει τη χρονική πολυπλοκότητα του αλγορίθμου ως $O(\lg n)$.

   (γ) Μπορούμε να υπολογίσουμε τα k_1 και k_2 σε χρόνο $O(\lg n)$ με τον αλγόριθμο από το μέρος (β). Υποθέτουμε ότι και οι δύο ακέραιοι είναι ίσοι με μηδέν. (Σε αντίθετη περίπτωση ο προτεινόμενος αλγόριθμος μπορεί εύκολα να γενικευθεί.)

```
l = 1;
r = n;
while (r != mid)
      mid = (l+r)/2
      εφάρμοσε δυαδική διερεύνηση για να βρεις το πλήθος των
      στοιχείων x του πίνακα B, έστω j, για τα οποία A[mid] >
      x
      Αν mid + j = m επέστρεψε το A[mid] και τερμάτισε
      Αν i + j > m θέσε r = mid - 1 ; και επανέλαβε το
      βρόχο (δηλαδή, συνέχισε το ψάξιμο στο πρώτο μισό του
      πίνακα A)
      Αν i + j < m θέσε l = mid + 1 ; και επανέλαβε το
      βρόχο (δηλαδή, συνέχισε το ψάξιμο στο δεύτερο  μισό του
      πίνακα A)
Αν r=mid τότε το m-οστό στοιχείο δεν βρίσκεται στον πίνακα A
Εφάρμοσε ξανά τον αλγόριθμο αντιστρέφοντας τους ρόλους των A
και B
```

   Για να υπολογίσουμε τον χρόνο εκτέλεσης του αλγορίθμου, παρατηρούμε ότι σε κάθε επανάληψη του βρόχου εκτελείται μια δυαδική αναζήτηση στον πίνακα B (χρόνος εκτέλεσης $O(\lg n)$). Ο αριθμός επαναλήψεων του βρόχου είναι επίσης $O(\lg n)$ αφού, η

αναζήτηση του m-οστού στοιχείου γίνεται και πάλι δυαδικά. Επομένως η χρονική πολυπλοκότητα του αλγορίθμου είναι της τάξης $O(\lg^2 n)$.

2. (α) Κάθε θέση του πίνακα T υπολογίζεται από την πιο κάτω έκφραση:

$$T(i,j) = \begin{cases} 0, & \text{if } i = j \\ F[i,j] + T(i,j-1) + T(i+1,j) - T(i+1,j-1), & \text{otherwise} \end{cases}$$

Προφανώς υπολογισμός κάθε θέσης απαιτεί χρόνο της $O(1)$ και ολόκληρου του πίνακα χρόνο $O(n^2)$. Ο υπολογισμός γίνεται ξεκινώντας από τη διαγώνιο, και προχωρώντας στις θέσεις, $T(i, i+1)$, $T(i, i+2)$, …, μέχρι την $T(1, n)$

(β) Γράφουμε $M(i,j)$ για τον μέγιστο αριθμό φιλιών που μπορούν να ικανοποιηθούν κατά το μοίρασμα των πρώτων i φοιτητών σε j ομάδες. Η *τελευταία* ομάδα μπορεί να έχει από 2 μέχρι και i-$(2j$-$2)$ φοιτητές. Παρατηρούμε ότι για να πετύχουμε το επιδιωκόμενο βέλτιστο μοίρασμα, πρέπει να η τελευταία κοπή να γίνει μετά από τον φοιτητή l, $(2j\text{-}2) \le l \le i - 2$, όπου μεγιστοποιείται η τιμή $M(l,j\text{-}1) + T(l\text{+}1, j)$. Επομένως το $M(i,j)$ μπορεί να οριστεί αναδρομικά ως εξής:

$$M(i,j) = \begin{cases} 0, & \text{if } i = 0 \\ T(1,i), & \text{if } j = 1 \\ \max\limits_{2j-2 \le l \le i-2} \{M(l,j-1) + T(l+1,j)\}, & \text{otherwise} \end{cases}$$

Δεδομένης της ύπαρξης του πίνακα $T(i,j)$, ο αλγόριθμος υπολογισμού του πίνακα $M(i,j)$ έχει ως εξής:

```
for (j = 1; j ≤ k; j++)
     for (i = 1; i ≤ n; i++)
     υπολόγισε το M(i,j) βάσει της πιο πάνω εξίσωσης
     επέστρεψε το M(n,k)
```

Για υπολογισμό κάθε μιας από τις nk θέσεις του πίνακα απαιτείται χρόνος της τάξης $O(n)$. Επομένως η χρονική πολυπλοκότητα του αλγορίθμου είναι της τάξης $O(k \cdot n^2)$

Τέλος, για να επιστρέψουμε τα ακριβή σημεία των $k - 1$ κοπών που αντιστοιχούν στη βέλτιστη λύση, θα πρέπει να επεκτείνουμε τον αλγόριθμο έτσι ώστε να διατηρεί πληροφορίες για τη δομή της βέλτιστης λύσης κάθε υπο-προβλήματος. Συγκεκριμένα, εκτός από τον πίνακα $M(i,j)$ θα χρησιμοποιήσουμε ένα δεύτερο πίνακα $LastCut(i,j)$ όπου θα φυλάγουμε τον αριθμό της τελευταίας κοπής l που μεγιστοποιεί την τιμή $M(l,j\text{-}1) + T(l\text{+}1, j)$. Για επανάκτηση των κοπών από την τελευταία μέχρι την πρώτη, θα επιτρέψουμε τις τιμές του πίνακα, $l_{k\text{-}1} = LastCut(n, k)$, $l_{k\text{-}2} = LastCut(l_{k\text{-}1}, k\text{-}1)$, …, $l_1 = \text{LastCut}(l_2, 2)$.

3. Για κάθε άτομο x της εταιρείας μας ενδιαφέρει να υπολογίσουμε δύο τιμές:

max1(*x*), τον μέγιστο αριθμό καλεσμένων που μπορεί να επιτευχθεί από το υπόδενδρο που ριζώνει στο x, *αν περιλάβουμε το άτομο x*, και

max2(*x*), τον μέγιστο αριθμό καλεσμένων που μπορεί να επιτευχθεί από το υπόδενδρο που ριζώνει στο x, *αν δεν περιλάβουμε το άτομο x*.

Οι δύο αυτές τιμές υπολογίζονται αναδρομικά ως εξής:

$$\max 1(x) = 1 + \sum_{y \in child(x)} \max 2(y)$$

$$\max 2(x) = \sum_{y \in child(x)} \max\{\max 1(y), \max 2(y)\}$$

Η πρώτη εξίσωση εκφράζει ότι ο βέλτιστος τρόπος για να επιλέξουμε εργαζόμενους περιλαμβανομένου του x από το υπόδενδρο του x είναι να επιλέξουμε εργαζόμενους από τα υπόδενδρα κάθε παιδιού του x, y, χωρίς να περιλάβουμε τον y. Η δεύτερη εξίσωση εκφράζει ότι ο βέλτιστος τρόπος για να επιλέξουμε εργαζόμενους μη-περιλαμβανομένου του x από το υπόδενδρο του x είναι να επιλέξουμε εργαζόμενους από τα υπόδενδρα κάθε παιδιού του x, y, όπου ο y μπορεί να περιληφθεί ή όχι ανάλογα με το τι θα επιφέρει τον μεγαλύτερο αριθμό ατόμων.

Άρα ο αλγόριθμος θα πρέπει να υπολογίσει για κάθε κόμβο τις δύο τιμές, ξεκινώντας από το τελευταίο επίπεδο του δένδρου και προχωρώντας προς τα πάνω. Παρατηρούμε ότι για οποιοδήποτε φύλλο y, $\max1(y)=1$, $\max2(y)=0$.

Το ζητούμενο είναι η τιμή $\max\{\max1(r), \max2(r)\}$, όπου r η ρίζα του δένδρου.

Κάθε κόμβος εξετάζεται κάποιο σταθερό αριθμό φορών και επεξεργασία του απαιτεί σταθερό χρόνο, επομένως, η πολυπλοκότητα του αλγόριθμου είναι $\Theta(n)$, όπου n είναι ο αριθμός των κόμβων του δένδρου.

Για υπολογισμό της λίστας καλεσμένων καλούμε την πιο κάτω αναδρομική διαδικασία με παράμετρο τη ρίζα του δένδρου:

```
Invite(x)
     if max1(x) > max2(x)
          invite x to the party and
          for all grandchildren y of x
               Invite(y)
     else
          for all grandchildren y of x
               Invite(y)
```

4. (α) Έστω οι εργασίες {(0,3), (2,5), (6,9), (4,10)}. Ο προτεινόμενος αλγόριθμος θα επέλεγε το σύνολο εργασιών

   {(0,3), (6,9)} για την πρώτη αίθουσα,
   {(2,5)} για τη δεύτερη αίθουσα, και
   {(4,10)} για την τρίτη αίθουσα.

   Η βέλτιστη λύση όμως χρησιμοποιεί μόνο δύο αίθουσες ως εξής:

   Αίθουσα 1: {(0,3), (4,10)}
   Αίθουσα 2: {(2,5), (6,9)}

   (β) Άπληστος αλγόριθμος για το πρόβλημα είναι ο εξής:
   - Ταξινόμησε τις εργασίες σε αύξουσα σειρά χρόνου εκκίνησης.
   - Ανάθεσε την πρώτη κράτηση στην πρώτη αίθουσα.
   - Για κάθε επόμενη εργασία x

- Εφόσον η x είναι συμβατή με το σύνολο κρατήσεων κάποιας από τις αίθουσες που χρησιμοποιούνται μέχρι στιγμής, ανάθεσε την κράτηση στην αίθουσα, και προχώρησε στην επόμενη κράτηση. Διαφορετικά, ανάθεσε την κράτηση σε μία νέα αίθουσα.

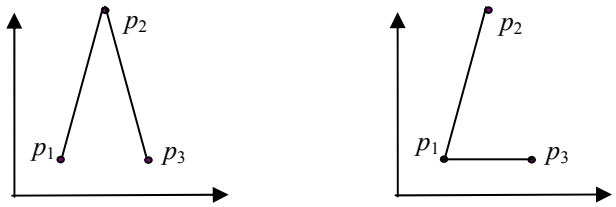
Ορθότητα:

Μπορούμε να δούμε ότι το σύνολο κρατήσεων για κάθε αίθουσα είναι συμβατό. Παραμένει να ελέγξουμε κατά πόσο ο αλγόριθμος χρησιμοποιεί τον ελάχιστο αριθμό αιθουσών.

Έστω a_k ο αριθμός κρατήσεων με τις οποίες επικαλύπτεται η κράτηση k. Προφανώς ο ελάχιστος αριθμός αιθουσών που απαιτείται για την ικανοποίηση όλων των κρατήσεων είναι τουλάχιστον $m=\max(a_1, a_2, \ldots, a_n) + 1$. Θα δείξουμε ότι ο αριθμός αιθουσών που φέρνει εις χρήση ο πιο πάνω αλγόριθμος είναι m.

Η απόδειξη γίνεται με αντίφαση. Ας υποθέσουμε, ότι κατά την επεξεργασία της κράτησης k, ο αλγόριθμος φέρνει εις χρήση την $m+1$ αίθουσα. Αυτό συνεπάγεται ότι η κράτηση k συγκρούεται με τις κρατήσεις και των m αιθουσών εν χρήση. Κατά συνέπεια $a_k \geq m$. Αυτό έρχεται σε αντίφαση με τον ορισμό του m, για το οποίο γνωρίζουμε $m > a_i$ για κάθε i, και εν προκειμένω, $m > a_k$. Συνεπώς, ο αλγόριθμος δεν θα φέρει εις χρήση περισσότερες από m αίθουσες, και αφού ο αριθμός αυτός είναι απαραίτητος για την ορθή ικανοποίηση των κρατήσεων, ο αλγόριθμος θα χρησιμοποιήσει ακριβώς τον ελάχιστο δυνατό αριθμός αιθουσών.

5. (i) Ο αλγόριθμος δεν είναι ορθός. Για παράδειγμα, με δεδομένο εισόδου τα σημεία του πιο κάτω σχήματος θα επέστρεφε το γεννητορικό δένδρο $\{(p_1, p_2), (p_2, p_3)\}$, που δεν είναι ελάχιστο (το $\{(p_1, p_2), (p_1, p_3)\}$, έχει μικρότερο συνολικό βάρος).



(ii) Ο αλγόριθμος δεν είναι ορθός. Για παράδειγμα, στα σημεία του πιο κάτω σχήματος θα επέστρεφε το δένδρο $\{(p_1, p_2), (p_2\ p_3), (p_3\ p_4)\}$, ενώ το δένδρο $\{(p_1, p_2), (p_2\ p_3), (p_1, p_4)\}$ έχει μικρότερο βάρος.