

**Εθνικό Μετσόβιο Πολυτεχνείο**



Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Αλγόριθμοι και Πολυπλοκότητα 2016-2017  
2η Σειρά Γραπτών Ασκήσεων

Όνομα: Παναγιώτης Κωστοπαναγιώτης  
A.M: 03115196

## Άσκηση 1: Επιλογή και Κάλυψη Μαθημάτων

(α.1)

1. Το συγκεκριμένο κριτήριο πράγματι μας εγγυάται την βέλτιστη λύση
2. Θα δείξουμε ότι το συγκεκριμένο κριτήριο δεν μας οδηγεί σε βέλτιστη λύση κατασκευάζοντας ένα απλό αντιπαράδειγμα. Έστω τα διαστήματα [ 1, 5 ), [ 4, 6 ), [ 6, 7 ), σε αυτό το παράδειγμα, εφ' όσον υπάρχουν επικαλύψεις ο αλγόριθμός μας θα αφαιρέσει το διάστημα [ 1, 5 ) το οποίο είναι και το μεγαλύτερο σε διάρκεια. Ωστόσο, συνεχίζουν να υπάρχουν επικαλύψεις και άρα ο αλγόριθμός μας θα αφαιρέσει το διάστημα [ 4, 6 ) μένοντας με το [ 6, 7 ) και άρα η απάντηση θα είναι 1. Ωστόσο, η βέλτιστη απάντηση είναι 2 αφού τα διαστήματα [ 1,5 ) και [ 6, 7 ) δεν επικαλύπτονται.
3. Το συγκεκριμένο κριτήριο πράγματι μας εγγυάται την βέλτιστη λύση.

(α.2) Εφ' όσον επιτρέψουμε κόστη σε κάθε διάστημα, δεν φαίνεται να υπάρχει κάποιο άπληστο κριτήριο το οποίο να μας επιτρέπει να λύσουμε το πρόβλημα εύκολα. Συνεπώς, θα χρησιμοποιήσουμε δυναμικό προγραμματισμό. Αρχικά, ταξινομούμε τα διαστήματα σε αύξουσα σειρά ως προς το τέλος τους. Έπειτα, ορίζουμε ως  $dp(i)$  το βέλτιστο κέρδος που μπορούμε να επιτύχουμε με τα πρώτα  $i$  διαστήματα,. Εφ' όσον έχουμε υπολογίσει το παραπάνω για κάθε  $i$ , μπορούμε να λάβουμε την τελική λύση μας η οποία θα είναι απλά η  $dp(n)$ . Θα εξηγήσουμε αρχικά πως μπορούμε να υπολογίσουμε την βέλτιστη λύση και έπειτα θα παραλλάξουμε την λύση μας ώστε να βρίσκει και ένα βέλτιστο σύνολο διαστημάτων.

Για τον υπολογισμό του  $dp(i)$  για κάποιο τυχαίο  $i$  έχουμε την εξής επιλογή, αν θα επιλέξουμε ή όχι το  $i$ -οστό διάστημα στην λύση μας. Στην περίπτωση που δεν θα το συμπεριλάβουμε, η λύση μας είναι η βέλτιστη αν εξαιρέσουμε το  $i$ -οστό διάστημα, δηλαδή το  $dp(i-1)$ . Σε περίπτωση τώρα που επιλέξουμε να συμπεριλάβουμε το  $i$ -οστό διάστημα στην λύση μας, αρκεί να βρούμε το καλύτερο υποσύνολο διαστημάτων, όπου όλα τα στοιχεία του είναι πριν το  $i$ , το οποίο είναι συμβατό με το  $i$ -οστό διάστημα. Πρέπει δηλαδή, να βρούμε ένα  $k \leq i$ , τέτοιο ώστε  $f_k \leq s_i$  και το  $dp(k)$  να είναι όσο το δυνατόν μεγαλύτερο. Λαμβάνουμε λοιπόν την εξής αναδρομική σχέση για το  $dp(i)$ .

$$dp(1) = w_1$$

$$dp(i) = \max(dp(i-1), w_i + \max_{k \leq i \wedge f_k \leq s_i} dp(k))$$

Έχοντας αυτή την λύση μπορούμε να την χρησιμοποιήσουμε για να βρούμε και ένα βέλτιστο σύνολο διαστημάτων. Εστω  $\text{choice}(i)$  η βέλτιστη επιλογή προηγούμενου διαστήματος που πρέπει να κάνουμε προκειμένου να επιτύχουμε την βέλτιστη λύση. Το  $\text{choice}(i)$  υπολογίζεται ταυτόχρονα μαζί με το  $dp(i)$  και χρησιμοποιείται όπως ακολούθως.

Αλγόριθμος: maximum score of non-overlapping intervals

Input: ένας πίνακας μεγέθους  $n$  από διαστήματα

Output: ένα σύνολο  $A$  από διαστήματα

```
sort( $s_i, f_i$ ) by  $f_i$ ,  $\forall i \leq n$ 
 $dp(1) \leftarrow w_1$ 
 $\text{choice}(1) \leftarrow -1$ 
for  $i \leq n$ 
     $dp(i) \leftarrow w_i$ 
     $\text{choice}(i) \leftarrow -1$ 
    for  $k < i$ 
        if  $dp(k) + w_i > dp(i) \wedge f_k \leq s_i$ 
             $dp(i) \leftarrow dp(k) + w_i$ 
             $\text{choice}(i) \leftarrow k$ 
```

```

choice(i) ← k
A = ∅
idx ← n
while idx > 0
    if choice(idx) = -1
        idx ← idx - 1
    else
        A = A ∪ (sidx, fidx)
        idx = choice(idx)

```

Ορθότητα: Αρχικά, είναι σαφές πως άπαξ και καταφέρουμε να υπολογίσουμε σωστά το  $dp(i)$   $\forall i$ , τότε το  $dp(n)$  είναι η απάντηση στο πρόβλημά μας. Για να αποδείξουμε την ορθότητα του αλγορίθμου μας, εφαρμόζουμε επαγωγή. Αρχικά, για  $i=1$  ο αλγόριθμός μας δουλεύει αφού πράγματι η βέλτιστη λύση για ένα διάστημα είναι το διάστημα αυτό καθ' αυτό. Έστω τώρα, πως έχουμε υπολογίσει ορθά την βέλτιστη απάντηση για κάθε  $k < i$ . Θα δείξουμε ότι ο αλγόριθμός μας υπολογίζει σωστά το  $dp(i)$ . Αρχικά, σε αυτό το πρόβλημα ισχύει η αρχή της βελτιστότητας, προφανώς δηλαδή η τελική μας λύση, καθώς και κάθε ενδιάμεση, θα αποτελείται από βέλτιστες επιμέρους λύσεις, αφού αν δεν αποτελούνταν θα μπορούσαμε να την βελτιώσουμε για να πάρουμε μια καλύτερη τελική λύση.

Επομένως, διαμερίζουμε τον χώρο των δυνατών επιλογών που έχουμε σε 2 συμπληρωματικά ενδεχόμενα. Είτε θα επιλέξουμε το  $i$ -οστό διάστημα, είτε δεν θα το επιλέξουμε. Αν καταφέρουμε να υπολογίσουμε την βέλτιστη απάντηση για κάθε μας από αυτές τις περιπτώσεις, θα έχουμε και την τελική αφού η λύση μας θα είναι το μέγιστο των μοναδικών επιλογών μας.

Έστω πως δεν επιλέγουμε το  $i$ -οστό διάστημα, τότε προφανώς η λύση μας θα είναι το  $dp(i-1)$  αφού οι τιμές του  $dp$  διατηρούν αύξουσα διάταξη.

Έστω πως επιλέγουμε το  $i$ -οστό διάστημα τότε, δεδομένου πως το διάστημα που επιλέξαμε δεν είναι το πρώτο της λύσης μας, θα υπάρχει και κάποιο προηγούμενο. Θεωρούμε την τετριμμένη περίπτωση που το διάστημά μας είναι το πρώτο ξεχωριστά στον αλγόριθμό μας. Πάλι, διαμερίζουμε τον χώρο των επιλογών μας σε συμπληρωματικά και ξένα μεταξύ τους περιεχόμενα. Από τα διαστήματα που δεν τέμνουν το  $i$ -οστό, κάποιο πρέπει να επιλέξουμε. Λόγω της αρχής της βελτιστότητας, θα επιλέξουμε αυτό που μας μεγιστοποιεί την λύση.

Συνεπώς, έχοντας διαχειριστεί και τις δύο δυνατές περιπτώσεις συμπαιρένουμε πως ο αλγόριθμος υπολογισμού της λύσης μας είναι βέλτιστος. Ο αλγόριθμος εύρεσης του συνόλου των διαστημάτων είναι τετριμμένα βέλτιστος λόγω της παραπάνω λύσης.

Πολυπλοκότητα: Στην παραπάνω λύση έχουμε  $O(n)$  state-space καθώς και  $O(n)$  χρόνο για τον υπολογισμό κάθε κατάστασης. Συνεπώς ο αλγόριθμός μας έχει πολυπλοκότητα  $O(n^2)$ . Ωστόσο, μπορούμε να βελτιστοποιήσουμε ορισμένα πράγματα κάνοντας την εξής παρατήρηση: Τα διαστήματα που δεν τέμνουν το διάστημα που μας ενδιαφέρει σε κάθε βήμα, αποτελούν ένα prefix του πίνακα μας, εφ' όσον βέβαια τα έχουμε ταξινομήσει κάτα  $f_i$ . Συνεπώς, ορίζουμε ως  $dpm_{max}(i)$  ως το μέγιστο  $dp(k)$  για κάποιο  $k \leq i$ , για κάθε  $i$  για το οποίο υπολογίζουμε το  $dp(i)$  εφαρμόζουμε δυαδική αναζήτηση προκειμένου να βρούμε το πρώτο διάστημα, έστω  $j$ , που δεν τέμνει το  $i$ -οστό και έπειτα η απάντηση είναι απλά  $w_i + dpm_{max}(j)$ . Έτσι, καταφέρουμε να ρίξουμε την πολυπλοκότητα του αλγορίθμου μας σε  $O(n \log n)$ , όσο δηλαδή και του αντίστοιχου άπληστου για την περίπτωση που τα κόστη είναι μοναδιαία.

(β) Ο αλγόριθμος που θα εφαρμόσουμε είναι ο εξής. Ταξινομούμε τα διαστήματα κατά τον χρόνο έναρξης. Έπειτα εισάγουμε συνεχώς διαστήματα μέχρις ότου ο χρόνος έναρξης του διαστήματος που πάμε να βάλουμε να ξεπερνάει τον ελάχιστο χρόνο λήξης των διαστημάτων που είναι στο σύνολό μας. Άμα συμβεί αυτό, μετράμε μια χρονική στιγμή και ξεκινάμε από την αρχή

Αλγόριθμος: minimum number of sets

Input: ένας πίνακας διαστημάτων μεγέθους  $n$   
Output: ένα σύνολο  $A$  από χρονικές στιγμές

```
sort( $s_i, f_i$ ) by  $s_i \forall i$ 
 $A = \emptyset$ 
 $min_{time} = \infty$ 
for  $i \leq n$ 
  if  $s_i \leq min_{time}$ 
     $min_{time} = \min(min_{time}, f_i)$ 
  else
     $A = A \cup min_{time}$ 
     $min_{time} = f_i$ 
 $A = A \cup min_{time}$ 
```

Πολυπλοκότητα: Αρχικά ταξινομούμε τα διαστήματα και έπειτα εφαρμόζουμε μια γραμμική διαδικασία. Συνεπώς η συνολική πολυπλοκότητα είναι  $O(n \log n)$ .

## Άσκηση 2: Πομποί και Δέκτες

(α) Μπορούμε να σκεφτούμε το πρόβλημα ως εξής. Έστω πως αντικαθιστούμε κάθε πομπό με μια ανοιχτή παρένθεση και κάθε δέκτη με μια κλειστή παρένθεση. Το ζητούμενο πλέον είναι βρούμε όσο το δυνατόν περισσότερα ζευγάρια ζυγισμένων παρενθέσεων. Εφαρμόζουμε έναν άπληστο αλγόριθμο. Κάθε στιγμή διατηρούμε έναν μετρητή ανοιχτών παρενθέσεων, κάθε φορά που συναντάμε μια κλειστή παρένθεση και μετρητής μας είναι θετικός, μετράμε ένα έξτρα ζευγάρι και μειώνουμε κατά ένα τον μετρητή μας. Στο τέλος, θα έχουμε τον μέγιστο αριθμόν ζευγών πομπού-δέκτη που μπορούμε να κατασκευάσουμε. Συνολικά ο αλγόριθμός μας έχει πολυπλοκότητα  $O(n)$ .

(β) Θα εφαρμόσουμε δυναμικό προγραμματισμό. Για κάθε κεραία έχουμε 2 επιλογές, είτε θα την κάνουμε πομπό, ή δέκτη με το αντίστοιχο κόστος. Οι περιορισμοί μας είναι στο τέλος το πλήθος των δεκτών να είναι ίσο το πλήθος των δεκτών να είναι ίσο με το πλήθος των πομπών, καθώς και σε πρόθεμα το πλήθος των πομπών να είναι τολάχιστον όσο το πλήθος των δεκτών.

Ας συμβολίσουμε ως  $dp[i, k]$  ως το ελάχιστο κόστος για να αντικαταστήσουμε τις πρώτες  $i$  κεραίες με το πλήθος των πομπών να είναι ίσο με  $k$  (σε αυτή την περίπτωση το πλήθος των δεκτών είναι προφανώς  $i - k$ ), με  $i < 2k$ . Η αναδρομική σχέση προκύπτει άμεσα λόγω του παραπάνω δυισμού, είτε η  $i$ -οστή κεραία θα είναι πομπός, είτε δέκτης. Έχουμε δηλαδή:

Βάση της αναδρομής:  $dp[1, 0] = \infty, dp[1, 1] = T_1$

Αναδρομική σχέση:  $dp[i, k] = \min(dp[i-1, k-1] + T_i, dp[i-1, k] + R_i), \text{Av } i-1 < 2(k-1)$   
 $dp[i, k] = dp[i-1, k] + R_i, \text{Av } i-1 \geq 2(k-1)$

Η τελική απάντηση μας δίνεται προφανώς από το  $dp[n, n/2]$ .

Ορθότητα: Στο πρόβλημα μας ισχύει η αρχή της βελτιστότητας. Σε κάθε βήμα της αναδρομικής σχέσης διαμερίζουμε τον χώρο μας σε 2 συμπληρωματικά ενδεχόμενα για τα οποία γνωρίζουμε την βέλτιστη λύση. Συνεπώς, η ελάχιστη εκ των 2 ενδεχομένων αποτελεί την βέλτιστη λύση για το υποπρόβλημα που εξετάζουμε στο επαγγεικό μας βήμα. Τέλος, η απάντηση  $dp[n, n/2]$  πράγματι μας δίνει την βέλτιστη απάντηση αφού εμπεριέχει όλη την πληροφορία που χρειαζόμαστε προκειμένου η λύση μας να είναι έγκυρη.

Πολυπλοκότητα: Το μέγεθος του state-space μας είναι  $O(n^2)$  και ο υπολογισμός κάθε υποπρόβληματος γίνεται σε σταθερό χρόνο. Συνεπώς η πολυπλοκότητα της λύσης μας είναι  $O(n^2)$ .

### Άσκηση 3: Διαγωνισμός Χωρού

(α) Θα μπορούσαμε να αντιμετωπίσουμε κάθε κομμάτι ως ένα διάστημα  $[i, i + \text{rest}(i) - 1]$  με score,  $\text{score}(i)$  και να εργαστούμε ακριβώς όπως στο πρόβλημα 1(β) καταλήγοντας σε μια λύση  $O(n\log n)$ .

Ωστόσο, μπορούμε να λύσουμε το πρόβλημα πιο αποδοτικά καταλήγοντας σε αλγόριθμο γραμμικού χρόνου. Εφαρμόζουμε πάλι δυναμικό προγραμματισμό, ωστόσο, αντί να υπολογίσουμε την λύση ξεκινώντας από την αρχή, ξεκινάμε από το τέλος και ορίζουμε ως  $dpr[i]$  ότι για το score που μπορούμε να έχουμε αν διαλέξουμε κάποια από τα κομμάτια  $i, i+1, \dots, n$ . Έτσι, τα διαστήματά μας βρίσκονται σε φθίνουσα σειρά ως προς το τέλος  $i$ .

Η λύση μας τώρα είναι ακριβώς ανάλογη με την λύση του 1(β) με την διαφορά ότι για τον υπολογισμό του κάθε state  $dpr[i]$  δεν χρειάζεται να κάνουμε binary search, απλώς παίρνουμε το μέγιστο  $dpr[k]$  με  $k \geq i + \text{rest}(i) - 1$ . Η πληροφορία αυτή μας δίνεται από τον πίνακα  $dpmx$ .

**Οοθότητα:** Η οοθότητα του αλγορίθμου μας προκύπτει επαγωγικά, ακριβώς ανάλογα με την λύση για την 1(β).

**Πολυπλοκότητα:** Η πολυπλοκότητα του αλγορίθμου μας είναι γραμμική, αφού ο υπολογισμός κάθε state γίνεται σε σταθερό χρόνο.

(β) Μπορούμε να εργαστούμε αντίστοιχα με την παραπάνω περίπτωση. Ορίζουμε ως  $dpr(i)$  ότι για την βέλτιστη απάντηση που μπορούμε να έχουμε από το κομμάτι  $i$  εώς το  $n$ . Για κάθε κομμάτι έχουμε 2 επιλογές, είτε θα το επιλέξουμε είτε όχι. Λαμβάνουμε τις παρακάτω περιπτώσεις:

- Αν δεν επιλέξουμε το  $i$ -οστό κομμάτι, τότε δεν έχουμε κάποιον περιορισμό και μπορούμε απλά να βρούμε την λύση για τα τα κομμάτια από το  $i + 1$  εώς το  $n$ . Η απάντηση είναι αποθηκευμένη στο  $dpr(i + 1)$ .
- Αν επιλέξουμε το  $i$ -οστό κομμάτι, τότε λαμβάνουμε αρχικά το κέρδος  $\text{score}(i)$ . Έπειτα, πρέπει να βρούμε ποιό κομμάτι θα είναι το προηγούμενο που διαλέξαμε. Για να το κάνουμε αυτό, ελέγχουμε κάθε δυνατό κομμάτι από το  $i + 1$  εώς το  $n$  με το οποίο δεν υπάρχει επικάλυψη και επιλέγουμε αυτό που μας μεγιστοποιεί το κέρδος μας.

Η αναδρομική σχέση είναι η εξής:

$$\begin{aligned} dp(n) &= \text{score}(n), \quad dp(n+1) = 0 \\ dp(i) &= \max(dp(i+1), \text{score}(i) + \max_{i+rest(i) \leq j \leq n+1 : i < j - \text{prep}(j)}(dp(j))) \end{aligned}$$

**Οοθότητα:** Θα δείξουμε ότι ο αλγόριθμός μας υπολογίζει πράγματι σωστά την απάντηση. Εφαρμόζουμε μαθηματική επαγωγή πάνω στο μήκος του suffix. Αρχικά, για το στοιχειώδες suffix μεγέθους 1, ο αλγόριθμός μας λειτουργεί τετριμμένα σωστά, αφού επιλέγει το  $\text{score}(n)$ .

Έστω πως ο αλγόριθμός μας λειτουργεί έχει υπολογίσει ορθά την απάντηση για  $k = n, n - 1, \dots, i + 1$ , θα δείξουμε ότι υπολογίζει σωστά την απάντηση και για  $k = i$ . Οι 2 επιλογές, όπως δείξαμε παραπάνω διαμερίζουν πλήρως τον χώρο των πιθανών επιλογών, άρα αρκεί να υπολογίσουμε την απάντηση και στις 2 περιπτώσεις και να επιλέξουμε από αυτές την μέγιστη.

- Περίπτωση 1: Σε περίπτωση που δεν επιλέξουμε το  $i$ -οστό κομμάτι, τότε η απάντηση για το  $dpr(i)$  θα είναι σαφώς η ίδια με το  $dpr(i+1)$ , αφού το  $i$ -οστό κομμάτι δεν επηρεάζει την λύση μας.
- Περίπτωση 2: Σε αυτή την περίπτωση, αφού επιλέγουμε το  $i$ -οστό, δοκιμάζουμε κάθε πιθανό κομμάτι για επόμενο. Το επόμενο στην σειρά μπορεί να είναι ένα εκ των  $i+1, \dots, n$ , εφ' όσον ικανοποιεί τις συνθήκες που περιγράφει το πρόβλημα, ή να μην υπάρχει. Σε περίπτωση που υπάρχει, έχοντας υπολογίσει την λύση για κάθε  $k > i$ , τις δοκιμάζουμε όλες και από αυτές

επιλέγουμε αυτή που μας δίνει το βέλτιστο score. Έτσι, υπολογίζουμε την λύση για το i.

Τελικά, η απάντησή μας θα είναι η  $dp(i)$ . Το παραπάνω ολοκληρώνει την απόδειξη ορθότητας του αλγορίθμου μας.

**Πολυπλοκότητα:** Το state-space είναι μεγέθος  $n$  και για τον υπολογισμό του κάθε state ο αλγόριθμός μας έχει γραμμική πολυπλοκότητα. Συνεπώς, η τελική πολυπλοκότητα του αλγορίθμου μας είναι  $O(n^2)$ .

#### Άσκηση 4: Βγάζοντας Βόλτα το Σκύλο

Είναι σαφές, πως αν έχουμε ένα υποψήφιο μήκος λουριού και μια δεδομένη αλληλουχία μεταβάσεων, αρκεί να ελέγξουμε αν το μήκος του λουριού είναι μεγαλύτερο ή ίσο από την μέγιστη απόσταση μεταξύ κάποιων σημείων της καμπύλης και όχι κάποιου ενδιάμεσου.

Εφαρμόζουμε δυναμικό προγραμματισμό. Ας συμβολίσουμε ως  $dp[i, j]$  ως το ελάχιστο μήκος λουριού που χρειάζεται για να περάσουμε από τα σημεία  $p_1, p_2, \dots, p_i$  καταλήγοντας στο  $p_j$ , ενώ ο σκύλος έχει διασχίσει τα σημεία  $q_1, q_2, \dots, q_j$  καταλήγοντας στο  $q_j$ . Για κάθε κατάσταση έχουμε τις εξής επιλογές:

1. Θα μετακινηθούμε στο επόμενο σημείο, ενώ ο σκύλος μένει στο σημείο που βρίσκεται.
2. Ο σκύλος θα μετακινηθεί στο επόμενο σημείο, ενώ εμείς παραμένουμε ακίνητοι.
3. Μετακινούμαστε ταυτόχρονα με τον σκύλο στο επόμενο σημείο.

Συνεπώς, λαμβάνουμε την ακόλουθη αναδρομική σχέση:

Βάση της αναδρομής:

$$dp[1, 1] = 0, dp[i, 1] = \max(d(i, 1), dp[i-1, 1]), \text{ για } 1 < i \leq n, dp[1, i] = \max(d(1, i), dp[1, i-1]), \text{ για } 1 < i \leq m.$$

Αναδρομική σχέση:

$$dp[i, j] = \min(\max(dp[i-1, j], d(i, j)), \max(dp[i, j-1], d(i, j)), \max(dp[i-1, j-1], d(i, j)))$$

Στο τέλος, το βέλτιστο μήκος θα δίνεται από το  $dp[n, m]$ .

**Ορθότητα:** Στο πρόβλημά μας ισχύει η αρχή της βελτιστότητας. Η βέλτιστη λύση αποτελείται από επιμέρους βέλτιστες διαδρομές. Στο πρόβλημά μας διαμερίζουμε ξανά τον χώρο των επιλογών μας σε συμπληρωματικά ενδεχόμενα και λαμβάνουμε το ελάχιστο από αυτά. Πράγματι, το να ελέγξουμε κάθε φορά ποια ήταν η τελευταία κίνηση που έγινε είναι αρκετό διότι όλες οι υπόλοιπες κινήσεις εμπεριέχονται μέσα στις λύσεις των υποπροβλημάτων που ορίζονται. Επίσης, οι 3 κινήσεις που περιγράφαμε παραπάνω είναι πράγματι οι μοναδικές που μπορούν να προκύψουν.

Συνεπώς, καταλήγουμε ότι η απάντηση  $dp[n, m]$  αποτελεί πράγματι την βέλτιστη λύση στο πρόβλημά μας.

**Πολυπλοκότητα:** Το μέγεθος του χώρου των υποπροβλημάτων είναι μεγέθους  $O(nm)$ . Ο υπολογισμός του κάθε υποπροβλήματος γίνεται σε σταθερό χρόνο, συνεπώς η συνολική πολυπλοκότητα του αλγορίθμου μας είνα  $O(nm)$ .