

**Αλγόριθμοι και Στοιχεία Πολυπλοκότητας.**

**Ενότητα 6:** Δυναμικός Προγραμματισμός.

Διδάσκων: Ηλίας Κ Σάββας, Αναπληρωτής Καθηγητής.

Τμήμα Μηχανικών Πληροφορικής, Τεχνολογικής Εκπαίδευσης.

**Άδειες χρήσης.**

* Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons (C C). **Αναφορά δημιουργού (B Y), Μη εμπορική χρήση (N C), Μη τροποποίηση (N D), 3.0, Μη εισαγόμενο.**
* Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.



**Χρηματοδότηση.**

* Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
* Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.



# Σκοποί ενότητας.

Ο αναγνώστης να μπορεί να:

1) χρησιμοποιεί την τεχνική του δυναμικού προγραμματισμού,

2) επιλύει προβλήματα εφαρμόζοντας την παραπάνω τεχνική.

ΠΕΡΙΕΧΌΜΕΝΑ ΕΝΌΤΗΤΑΣ.

[Σκοποί ενότητας. 2](#_Toc368166806)

[ΠΡΟΓΡΆΜΜΑΤΑ. 3](#_Toc368166807)

[ΠΊΝΑΚΕΣ. 3](#_Toc368166808)

[6. ΔΥΝΑΜΙΚΌΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΌΣ. 3](#_Toc368166809)

[6.1 Εισαγωγή. 3](#_Toc368166810)

[6.2 Πολλαπλασιασμός πινάκων. 4](#_Toc368166811)

# ΠΡΟΓΡΆΜΜΑΤΑ.

|  |  |
| --- | --- |
| ***Πρόγραμμα 1****: Πολλαπλασιασμός αλυσίδας πινάκων.* | 7 |

# ΠΊΝΑΚΕΣ.

|  |  |
| --- | --- |
| ***Πίνακας 6.1:*** *Κόστος.* | 9 |
| ***Πίνακας 6.2:*** *Προτεραιότητα.*  | 9 |

# 6. ΔΥΝΑΜΙΚΌΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΌΣ.

## 6.1 Εισαγωγή.

Αυτή η τεχνική προϋποθέτει μία αναδρομική επίλυση προβλημάτων, αλλά με μία bottom-up εκτίμηση των λύσεων. Οι υπό-λύσεις συνήθως αποθηκεύονται για την επαναχρησιμοποίησή τους. Αυτή η τεχνική χρησιμοποιείται κυρίως σε προβλήματα βελτιστοποίησης, τα οποία σχετίζονται με πολλές λύσεις, αλλά και μια τιμή κόστους για το κάθε ένα από αυτά. Τυπικό παράδειγμα αποτελεί η επίλυση της συνάρτησης Fibonacci, η οποία ορίζεται ως εξής:

.

Δηλαδή, η  αναλύεται ως εξής:



Εδώ μία top-down προσέγγιση της λύσης, δεν είναι καθόλου ικανοποιητική, γιατί αναγκαστικά θα επαναλάβει πολλούς ίδιους υπολογισμούς. Για παράδειγμα, στον υπολογισμό του , θα έπρεπε να υπολογισθεί τα  δύο φορές, και το  τρεις φορές. Σε αντίθεση, με την τεχνική του δυναμικού προγραμματισμού, οι υπό-λύσεις υπολογίζονται από μία φορά, και αποθηκεύονται (πχ σε ένα πίνακα), για όλες τις πιθανές φορές που θα πρέπει να χρησιμοποιηθούν.

Τα προβλήματα που αφορούν τον δυναμικό προγραμματισμό, είναι συνήθως προβλήματα βελτιστοποίησης, δηλαδή εύρεση μίας βέλτιστης λύσης, ενώ ταυτόχρονα η λύση αυτή να υπόκειται και σε κάποιους περιορισμούς. Η τεχνική μοιάζει αρκετά με την μέθοδο του «διαίρει και βασίλευε», σε σχέση με την διαίρεση του προβλήματος σε μικρότερα και απλούστερα υπό-προβλήματα, αλλά είναι δυνατόν να μην είναι του ίδιου τύπου. Τα βασικά συστατικά των αλγορίθμων δυναμικού προγραμματισμού είναι τα:

1. Διαίρεση του προβλήματος σε μικρότερα και απλούστερα μέρη.
2. Αποθήκευση των λύσεων σε ένα πίνακα. Αυτό γίνεται γιατί πολλές αν όχι όλες από τις λύσεις των υπό-προβλημάτων, πρόκειται να ξαναχρησιμοποιηθούν και δεν ενδείκνυται το να επιλυθούν ξανά και ξανά.
3. Συνδυασμός των λύσεων των μικρών κομματιών του προβλήματος, ώστε να οδηγούν σε λύσεις μεγαλύτερων κομματιών, μέχρι να επιλυθεί όλο το πρόβλημα.

## 6.2 Πολλαπλασιασμός πινάκων.

Ένα ενδιαφέρον και ιδιαίτερα χρονοβόρο μαθηματικό πρόβλημα, (ως προς τον απαιτούμενο χρόνο υπολογισμού του), είναι ο πολλαπλασιασμός πολλών πινάκων, πχ, .

Είναι γνωστό ότι το γινόμενο δύο πινάκων, έστω , δίνεται από τον τύπο:

, και .

και απαιτεί  πολλαπλασιασμούς. Δηλαδή, ανάλογα και με τα μεγέθη των πινάκων, απαιτεί ένα μεγάλο αριθμό πράξεων.

Το πρόβλημα έγκειται στο ότι ανάλογα με την σειρά που θα εκτελεσθούν οι πολλαπλασιασμοί, οι απαιτούμενες πράξεις διαφέρουν και μάλιστα κατά πολύ. Για παράδειγμα, έστω ότι ζητείται το γινόμενο των πινάκων:

.

Εάν ο πολλαπλασιασμός γίνει με την σειρά:

.

απαιτεί 100\*1\*100 = 10000 πράξεις για τον πρώτο πολλαπλασιασμό, και 100\*100\*4 = 40000 για τον δεύτερο, άρα σύνολο 50000 πράξεις.

Ενώ αν γίνει με την σειρά:

.

απαιτεί 1\*100\*4 = 400 πράξεις για τον πολλαπλασιασμό των B και C ο οποίος θα εκτελεσθεί πρώτος, και 100\*1\*4 = 400 για τον πολλαπλασιασμό του A με το αποτέλεσμα των άλλων δύο, δηλαδή σύνολο 800 πράξεις. Όπως φαίνεται, η διαφορά είναι παραπάνω από αρκετή, για να δώσει το ερέθισμα του ψαξίματος της σειράς των πολλαπλασιασμών, (ή για το που πρέπει να υπάρχουν παρενθέσεις, ώστε να προσδιορίζουν την προτεραιότητα των πράξεων).

Η τυποποίηση του προβλήματος αυτού είναι η ακόλουθη: Δεδομένης μίας σειράς πινάκων προς πολλαπλασιασμό, ζητείται να προσδιορισθεί η σειρά των πινάκων, ώστε να ελαχιστοποιείται ο αριθμός των πράξεων.

Φυσικά, ο αλγόριθμος που θα επιλύσει το πρόβλημα, δεν ασχολείται με τον πολλαπλασιασμό αυτό καθαυτό, αλλά με την σειρά των πολλαπλασιαστέων. Η προφανής λύση αυτού του προβλήματος, είναι να δοκιμασθούν όλοι οι πιθανοί συνδυασμοί. Εάν το κόστος που απαιτείται, αποδίδεται από την συνάρτηση , όπου *N* το πλήθος των υπό πολλαπλασιασμό πινάκων, τότε (διαίρει και βασίλευε), εάν η αλυσίδα του πολλαπλασιασμού διασπασθεί σε δύο λίστες, η μία *k* πινάκων και η άλλη *N* -*k* (προφανές), αυτό δείχνει ότι  γιατί, για κάθε συνδυασμό της πρώτης λίστας, αντιστοιχούν όλοι οι συνδυασμοί της άλλης. Επομένως, η συνάρτηση κόστους μπορεί να αποδοθεί αναδρομικά ως εξής:

.

Ο παραπάνω τύπος όμως, συσχετίζεται με μία γνωστή συνάρτηση της Συνδυαστικής Ανάλυσης, την συνάρτηση των *Catalan Numbers*, με *P*(*N*) = *C*(*N-1*) όπου,

.

(Ένα κλασσικό πρόβλημα που σχετίζεται με τους *Catalan numbers* είναι το περίφημο πρόβλημα του *Euler*, όπου τίθεται το ερώτημα με πόσους τρόπους ένα κανονικό πολύγωνο *Ν* κορυφών, διαιρείται σε *Ν*-*2* τρίγωνα).

Η πολυπλοκότητα αυτής της μεθόδου είναι της τάξης , και κατά συνέπεια απαγορευτική εκτός και εάν το *N* είναι ένας πολύ μικρός αριθμός. Επομένως, θα πρέπει να βρεθεί μία άλλη μέθοδος.

Η λύση που προκύπτει με χρήση των μεθόδων του δυναμικού προγραμματισμού, είναι η διάσπαση του προβλήματος σε μικρότερα, και στη συνέχεια ο συνδυασμός αυτών των λύσεων, ώστε να οδηγήσει στην συνολική λύση του προβλήματος.

Έστω ότι , είναι η λίστα των προς πολλαπλασιασμό πινάκων και έστω ότι , είναι το αποτέλεσμα του πολλαπλασιασμού των πινάκων με δείκτες από *i* μέχρι *j*. Τότε, το ζητούμενο είναι:

.

Εάν το πρόβλημα διασπασθεί σε δύο επιμέρους προβλήματα, αυτό σημαίνει ότι:

, για 1 ≤ *k* ≤ *N.*

Επομένως το πρόβλημα εστιάζεται στον υπολογισμό του βέλτιστου δείκτη *k*. Προφανώς, για να βρεθεί αυτός ο βέλτιστος δείκτης πρέπει να ψαχθούν όλοι οι πιθανοί συνδυασμοί, δηλαδή όλοι οι πιθανοί δείκτες και στη συνέχεια επιλογή του βέλτιστου. Εάν αυτό επιτευχθεί, τότε δεν χρειάζεται τίποτα άλλο παρά την διάσπαση των δύο σειρών σε μικρότερες ξανά και ξανά, μέχρι η σειρά να αποτελείται από μόνο ένα πίνακα, όπου δεν τίθεται θέμα κανενός υπολογισμού, (δεν θα υπάρχουν τουλάχιστον δύο πίνακες, οπότε δεν θα υπάρχει τίποτα να πολλαπλασιασθεί). Τέλος, από τον συνδυασμό των λύσεων από κάτω προς τα πάνω, θα προκύψει και η συνολική λύση.

Ανάλυση: Προφανώς οι επιμέρους λύσεις των υπό-προβλημάτων θα αποθηκεύονται σε πίνακες (αρχή της μεθόδου του δυναμικού προγραμματισμού). Έστω λοιπόν, δύο πίνακες *N*x*N*, m και s όπου θα αποθηκεύονται τα κόστη των γινομένων, και η προτεραιότητα των πράξεων (θέσεις των παρενθέσεων), αντίστοιχα. Το βέλτιστο κόστος του γινομένου , θα αποθηκεύεται στην θέση , (και επομένως ο πίνακας θα πρέπει να αρχικοποιηθεί στο 0). Ο πίνακας m θα πρέπει να ορίζεται αναδρομικά ως εξής:

.

Η διαδικασία δίνεται στην συνάρτηση του *Προγράμματος 1*. Οι αντίστοιχες τιμές του πίνακα s περιγράφουν το σημείο «αποκοπής» της αλυσίδας, ώστε να προκύψει η βέλτιστη λύση.

***Πρόγραμμα 1****: Πολλαπλασιασμός αλυσίδας πινάκων.*

#include <stdio.h>

#define N 100

/\* Πολλαπλασιασμός Αλυσίδας Πινάκων \*/,

int p[N];

int s[N][N];

long m[N][N];

void matrix\_chain(int);

void mult(int, int);

main()

{

 int x;

 int i, j;

 printf("\n Πλήθος πινάκων:");

 scanf("%d", &x);

 for (i=0; i<=x; i++) {

printf("\n Είσοδος %d διάστασης:", i);

scanf("%d", &p[i]);

 }

 matrix\_chain(x);

 printf("\n\n Πίνακας Κόστους");

 for (i=1; i<x; i++) {

printf("\n");

for (j=1; j<=x; j++)

printf("%8d", m[i][j]);

 }

 printf("\n\n Πίνακας Προτεραιοτήτων");

 for (i=1; i<x; i++) {

printf("\n");

for (j=1; j<=x; j++)

printf("%8d", s[i][j]);

 }

 }

**void matrix\_chain(int n)**

**{**

 **int i, j, k, L, q;**

 **for (i=1; i<=n; i++)**

**m[i][i] = 0;**

 **for (L=2; L<=n; L++) {**

**for (i=1; i<= n -L+1; i++) {**

**j = i + L -1;**

**m[i][j] = 1000000; /\* Υποκαθιστά το άπειρο \*/,**

**for (k=i; k<=j-1; k++) {**

**q = m[i][k] + m[k+1][j] + p[i-1] \* p[k] \* p[j];**

**if (q < m[i][j]) {**

**m[i][j] = q;**

**s[i][j] = k;**

**}**

**}**

**}**

 **}**

**}**

Έστω, ότι ζητείται να γίνει ο πολλαπλασιασμός των πινάκων:



Η είσοδος δεδομένων στο *Πρόγραμμα 1* θα είναι η εξής:

* Πλήθος πινάκων: **5.**
* Είσοδος 0 διάστασης: **3.**
* Είσοδος 1 διάστασης: **4.**
* Είσοδος 2 διάστασης: **2.**
* Είσοδος 3 διάστασης: **7.**
* Είσοδος 4 διάστασης: **3.**
* Είσοδος 5 διάστασης: **4.**

Και για τους δύο πίνακες, το αποτέλεσμα θα είναι:

***Πίνακας 6.1:*** *Κόστος.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 24 | 66 | 84 | **114** |
| 0 | 0 | 56 | 66 | 98 |
| 0 | 0 | 0 | 42 | **66** |
| 0 | 0 | 0 | 0 | 84 |

Προφανώς ο βέλτιστος αριθμός πράξεων δίνεται στη θέση 1, 5, και είναι 114, (γιατί ο ζητούμενος πολλαπλασιασμός αποδίδεται σαν ).

***Πίνακας 6.2:*** *Προτεραιότητα.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 2 | **2** |
| 0 | 0 | 2 | 2 | 2 |
| 0 | 0 | 0 | 3 | **4** |
| 0 | 0 | 0 | 0 | 4 |

Σε συνδυασμό με τον πίνακα s (και στις αντίστοιχες θέσεις), παρατηρείται ότι ο πρώτος διαχωρισμός πρέπει να γίνει στον δεύτερο πίνακα, δηλαδή . Άρα, εάν χρειάζεται να βρεθεί το κόστος αυτού του πρώτου γινομένου, αυτό αποδίδεται στον πρώτο πίνακα και στην θέση 1, 2, δηλαδή 24 πράξεις. Στην συνέχεια, για να βρεθεί η σειρά και το κόστος του γινομένου , αρκεί να αναζητηθούν στους πίνακες m και s πάλι στις αντίστοιχες θέσεις. Δηλαδή, κόστος 66 και 4. Τελικά, η σειρά εκτέλεσης των πράξεων είναι:

.

Τέλος έκτης ενότητας.

 