ΠΛΗ30

ΕΝΟΤΗΤΑ 2: ΣΧΕΔΙΑΣΗ ΑΛΓΟΡΙΘΜΩΝ

Μάθημα 2.2:
Δυναμικός Προγραμματισμός

Δημήτρης Ψούνης

www.psounis.gr

ΠΕΡΙΕΧΟΜΕΝΑ

Α. Σκοπός του Μαθήματος

Οι στόχοι του μαθήματος είναι:

Επίπεδο Α
- (-)

Επίπεδο Β
- Η τεχνική σχεδίασης αλγόριθμων Δυναμικός Προγραμματισμός
- Ο αλγόριθμος Δ.Π. για την ακολουθία Fibonacci

Επίπεδο Γ
- Ο αλγόριθμος Δ.Π. για τον αλυσιδωτό πολλαπλασιασμό πινάκων
- Ο αλγόριθμος Δ.Π. για την εύρεση της Μέγιστης Κοινής Υπακολουθίας

Β. Θεωρία
Τεχνικές Σχεδίασης Αλγορίθμων

- Στην 2η ενότητα του μαθήματος ασχολούμαστε με τεχνικές που έχουν αναπτυχθεί, ως γενικές μεθοδολογίες για την κατασκευή ενός αλγορίθμου:
  - Η τεχνική Διαίρει και Βασίλευε (Μάθημα 2.1)
  - **Η τεχνική του Δυναμικού Προγραμματισμού (Μάθημα 2.2)**
  - Η κατασκευή των Άπληστων Αλγόριθμων (Μάθημα 2.3)
- Υπάρχουν ακόμη δεκάδες τεχνικές κατασκευής αλγορίθμων που είναι εκτός ύλης.

Β. Θεωρία
1. Δυναμικός Προγραμματισμός

Όταν έχουμε ένα πρόβλημα που έχει τις εξής ιδιότητες:

- Ιδιότητα των Βέλτιστων Επιμέρους Δομών: Οτί για να λύσουμε το πρόβλημα αρκεί να υπολογίσουμε την βέλτιστη λύση σε κάποια υποπροβλήματα, συνήθως με αναδρομή.
- Μικρός Αριθμός Υποπροβλημάτων: Το πλήθος των υποπροβλημάτων που πρέπει να λύσουμε είναι μικρό (δηλαδή πολυωνυμικό ως προς το μέγεθος του προβλήματος)
- Επικαλυπτόμενα Επιμέρους Προβλήματα: Ότι λύνουμε πολλές φορές τα ίδια υποπροβλήματα με αποτέλεσμα να χάνουμε χρόνο

Ο Δυναμικός προγραμματισμός είναι η λύση!
- Αντί να λύνουμε από το μεγαλύτερο όλο και μικρότερα προβλήματα
- Ξεκινάμε από το μικρότερο και υπολογίζουμε όλο και μεγαλύτερα προβλήματα!! Αποθηκεύοντας τις ενδιάμεσες λύσεις για να τις αξιοποιήσουμε αργότερα!

Β. Θεωρία
1. Δυναμικός Προγραμματισμός

Για να κατασκευάσουμε έναν αλγόριθμο Δυναμικού Προγραμματισμού κάνουμε τα εξής βήματα:

1. Περιγράφουμε έναν αναδρομικό αλγόριθμο που λύνει το πρόβλημα
2. Δίνουμε την αναδρομική σχέση που υπολογίζει την βέλτιστη λύση (επίλυση από πάνω προς τα κάτω)
3. Διαπιστώνουμε ότι ισχύουν οι τρεις συνθήκες για την κατασκευή του αλγορίθμου δυναμικού προγραμματισμού.
4. Με βάση την αναδρομική σχέση, κατασκευάζουμε την διαδικασία επίλυσης από τα μικρά προβλήματα σε όλο και μεγαλύτερα (επίλυση από κάτω προς τα πάνω)
5. Δίνουμε τον επαναληπτικό αλγόριθμο που κάνει την επίλυσή του προβλήματος
6. Υπολογίζουμε την πολυπλοκότητα του επαναληπτικού αλγορίθμου

Β. Θεωρία
1. Δυναμικός Προγραμματισμός
1. Η ακολουθία Fibonacci

- ΠΡΟΒΛΗΜΑ: Δίνεται ένας φυσικός αριθμός n. Να υπολογιστεί ο n-ός αριθμός Fibonacci.
- Οι πρώτοι 15 αριθμοί Fibonacci:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	2	3	5	8	13	21	34	55	89	144	233	377	610

- Και τυπικά η ακολουθία ορίζεται μέσω της αναδρομικής σχέσης:

$$f_n = \begin{cases} 1, & n=1 \;\; ή \;\; n=2 \\ f_{n-1}+f_{n-2}, & n>2 \end{cases}$$

Β. Θεωρία
1. Δυναμικός Προγραμματισμός
1. Η ακολουθία Fibonacci (1.Διατύπωση Αναδρομικού Αλγορίθμου)

- Ένας αναδρομικός αλγόριθμος που υλοποιεί την παραπάνω διαδικασία είναι ο εξής:

```
procedure FibRec(n)
   if n=1 or n=2 then
      return 1
   else
      a=FibRec(n-1)
      b=FibRec(n-2)
      c=a+b
      return c
   end if
end procedure
```

- Ο αλγόριθμος έχει πολυπλοκότητα που περιγράφεται από την αναδρομική σχέση:

$$T(n) = \begin{cases} \Theta(1), & n=1 \;\; ή \;\; n=2 \\ T(n-1)+T(n-2)+\Theta(1), & n>2 \end{cases}$$

Β. Θεωρία

1. Δυναμικός Προγραμματισμός

1. Η ακολουθία Fibonacci (1.Διατύπωση Αναδρομικού Αλγορίθμου)

- Η αναδρομική σχέση T'(n)=T'(n-1)+T'(n-2)+1 μπορεί να προσεγγιστεί μέσω φραγμάτων

ΥΠΟΛΟΓΙΣΜΟΣ ΑΝΩ ΦΡΑΓΜΑΤΟΣ

Ισχύει $T'(n)=T'(n-1)+T'(n-2)+1\leq T'(n-1)+T'(n-1)+1$

Συνεπώς λύνουμε την αναδρομή $A(n)=2A(n-1)+1$

$$\begin{aligned}A(n)&=2A(n-1)+1=\\&=2^2A(n-2)+2+1=\\&=2^3A(n-3)+2^2+2+1=\\&=\ldots=\\&=2^kA(n-k)+2^{k-1}+\ldots+2^2+2+1=\\&=\ldots=\\&=2^nA(0)+2^{n-1}+\ldots+2^2+2+1=\\&=2^n\Theta(1)+\frac{2^{n-1+1}-1}{2-1}=\Theta(2^n)\end{aligned}$$

άρα $T(n)=O(2^n)$

Β. Θεωρία

1. Δυναμικός Προγραμματισμός

1. Η ακολουθία Fibonacci (1.Διατύπωση Αναδρομικού Αλγορίθμου)

- Η αναδρομική σχέση T'(n)=T'(n-1)+T'(n-2)+1 μπορεί να προσεγγιστεί μέσω φραγμάτων

ΥΠΟΛΟΓΙΣΜΟΣ ΚΑΤΩ ΦΡΑΓΜΑΤΟΣ

Ισχύει $T'(n)=T'(n-1)+T'(n-2)+1\geq T'(n-2)+T'(n-2)+1$

Συνεπώς λύνουμε την αναδρομή $K(n)=2K(n-2)+1$

$$\begin{aligned}K(n)&=2K(n-2)+1=\\&=2^2K(n-4)+2+1=\\&=2^3K(n-6)+2^2+2+1=\\&=\ldots=\\&=2^kK(n-2k)+2^{k-1}+\ldots+2^2+2+1=\\&=\ldots=\\&=2^{n/2}K(0)+2^{(n/2)-1}+\ldots+2^2+2+1=\\&=2^n\Theta(1)+\frac{2^{(n/2)-1+1}-1}{2-1}=\Theta(2^{n/2})\end{aligned}$$

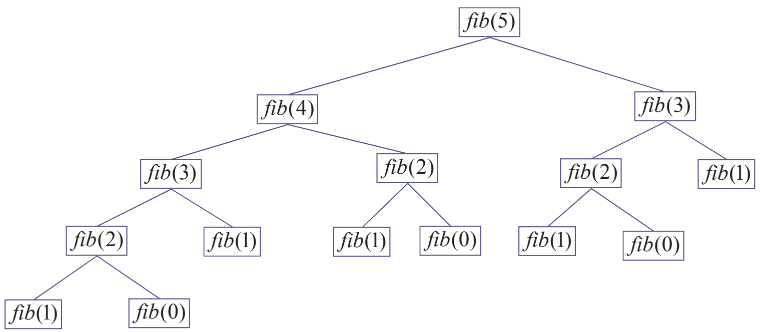
άρα $T(n)=\Omega(2^{n/2})$

Β. Θεωρία

1. Δυναμικός Προγραμματισμός

1. Η ακολουθία Fibonacci (1.Διατύπωση Αναδρομικού Αλγορίθμου)

- Η πολυπλοκότητα του αλγορίθμου είναι $T(n)=\Omega(2^{n/2})$ και $T(n)=O(2^n)$
- Είναι πολύ αργός αλγόριθμος!!
  - Λόγω του τρόπου εκτέλεσης της αναδρομικής διαδικασίας. Π.χ. για n=5:



Β. Θεωρία

1. Δυναμικός Προγραμματισμός

1. Η ακολουθία Fibonacci (3.Συνθήκες του Δ.Π.)

- Επαληθεύουμε ότι ισχύουν οι συνθήκες για την κατασκευή ενός αλγόριθμου Δυναμικού Προγραμματισμού:

ΕΠΑΛΗΘΕΥΣΗ ΤΩΝ ΙΔΙΟΤΗΤΩΝ ΤΟΥ Δ.Π. για την ακολουθία Fibonacci:

- Ιδιότητα των Βέλτιστων Επιμέρους Δομών: Πράγματι για να υπολογίσουμε τη λύση, υπολογίζουμε την λύση στα δύο αμέσως προηγούμενα υποπροβλήματα.
- Μικρός Αριθμός Υποπροβλημάτων: Το πλήθος των υποπροβλημάτων πρέπει να λύσουμε είναι μικρό (συγκεκριμένα πρέπει να λύσουμε n υποπροβλήματα)
- Επικαλυπτόμενα Επιμέρους Προβλήματα: Πράγματι αναγκαζόμαστε να λύσουμε πολλές φορές τα ίδια υποπροβλήματα, λόγω του τρόπου εκτέλεσης της αναδρομικής διαδικασίας.

Β. Θεωρία

1. Δυναμικός Προγραμματισμός

1. Η ακολουθία Fibonacci (4. Διαδικασία Επίλυσης)

- Ο δυναμικός προγραμματισμός προτείνει:
  - Να λύσουμε το πρόβλημα «από κάτω προς τα πάνω», δηλαδή να λύσουμε διαδοχικά τα υποπροβλήματα:
    - Fib(1)
    - Fib(2)
    - …
    - Fib(n)

1	2	3	4	5	6	...	n
1	1					…	fib(n)

Β. Θεωρία

1. Δυναμικός Προγραμματισμός

1. Η ακολουθία Fibonacci (5.Επαναληπτικος Αλγόριθμος και 6.Πολυπλοκότητα)

- Ένας αλγόριθμος δυναμικού προγραμματισμού που υλοποιεί την παραπάνω διαδικασία είναι ο εξής:

```
procedure FibSeq(n)
   A[1]=1
   A[2]=1
   for i=3 to n
      A[i]=A[i-1]+A[i-2]
   end for
   return A[n]
end procedure
```

- Η πολυπλοκότητα του αλγορίθμου είναι Θ(n).

Β. Θεωρία

1. Δυναμικός Προγραμματισμός

2. Αλυσιδωτός Πολλαπλασιασμός Πινάκων

- ΠΡΟΒΛΗΜΑ: Δίδονται οι πίνακες $A_1, A_2, \ldots, A_n$ όπου ο πίνακας A_i είναι διάστασης d_{i-1} x d_i. Με ποια σειρά θα γίνει ο πολλαπλασιασμός του γινομένου $A_1 \times A_2 \times \ldots \times A_n$

- Στιγμιοτύπα:
  - $A_1 \times A_2 \times A_3$ όπου A_1: 5x8, A_2: 8x4, A_3: 4x2
  - $A_1 \times A_2 \times A_3 \times A_4$ όπου A_1: 10x6, A_2: 6x3, A_3: 3x8, A_4: 8x2

- Αλγόριθμοι:
  - Προφανής αλγόριθμος: Κάνε τις πράξεις από αριστερά προς τα δεξιά: Πρώτα $A_1 \times A_2 = C_1$ έπειτα $C_1 \times A_3$ κ.ο.κ. Οι πράξεις που γίνονται είναι:
    $$d_0 d_1 d_2 + d_0 d_2 d_3 + \ldots + d_0 d_{n-1} d_n$$
  - Αλγόριθμος Δυναμικού Προγραμματισμού: Επέλεξε την σειρά με την οποία θα γίνουν οι πολλαπλασιασμοί των πινάκων. Επιτυγχάνεται σηματική βελτίωση στην πολυπλοκότητα.

Β. Θεωρία

1. Δυναμικός Προγραμματισμός

2. Αλυσιδωτός Πολλαπλασιασμός Πινάκων

- Ας δούμε γιατί έχει αξία ένας αλγόριθμος καθορισμού της σειράς των πολλαπλασιασμών. Παράδειγμα:
  - $A_1 \times A_2 \times A_3$ όπου A_1: 5x8, A_2: 8x4, A_3: 4x2 οι πράξεις μπορούν να γίνουν:
    - $(A_1 \times A_2) \times A_3$
      - Για τον υπολογισμό του $A_1 \times A_2$: 5x8x4=160 πράξεις και προκύπτει πίνακας 5x4, έστω C
      - Για τον υπολογισμό του C x A_3: 5x4x2=40 πράξεις και προκύπτει ο πίνακας-γινόμενο 5 x 2

      Σύνολο 200 πράξεις
    - $A_1 \times (A_2 \times A_3)$
      - Για τον υπολογισμό του $A_2 \times A_3$: 8x4x2=64 πράξεις και προκύπτει πίνακας 8x2, έστω C
      - Για τον υπολογισμό του A_1 x C: 5x8x2=80 πράξεις και προκύπτει ο πίνακας-γινόμενο 5 x 2

      Σύνολο 144 πράξεις

      Άρα με άλλη σειρά γινομένου, γλιτώνουμε πολλές πράξεις!

Β. Θεωρία
1. Δυναμικός Προγραμματισμός
2. Αλυσιδωτός Πολλαπλασιασμός Πινάκων

- Για περισσότερους πίνακες υπάρχουν περισσότεροι τρόποι να κάνουμε τον πολλαπλασιασμό, άρα και μεγαλύτερο όφελος στις πράξεις:
  - $A_1 \times A_2 \times A_3 \times A_4$ μπορεί να παρενθετοποιηθεί ως εξής:
    - $((A_1 A_2) A_3) A_4)$
    - $((A_1A_2)(A_3A_4))$
    - $(A_1(A_2A_3))A_4$
    - $A_1((A_2A_3)A_4)$
    - $A_1(A_2(A_3A_4))$
  - Πως θα επιλέξουμε την καλύτερη παρενθετοποίηση χωρίς να χρειαστεί να τις μελετήσουμε την κάθε μία ξεχωριστα;

Β. Θεωρία
1. Δυναμικός Προγραμματισμός
2. Αλυσιδωτός Πολλαπλασιασμός Πινάκων (1. Επίλυση με αναδρομή)

Για την επίλυση με αναδρομή του προβλήματος σκεφτόμαστε ως εξής:

- Θεωρούμε ότι έχουμε μια διαδικασία T(i,j) που έχει ήδη οριστεί και δουλεύει ορθά που όταν παίρνει ως όρισμα μια ακολουθία $A_i \dots A_j$ παράγει την βέλτιστη παρενθετοποίηση.
- Έστω τώρα το γενικό στιγμιότυπο: $A_1A_2 \dots A_{n-1}A_n$. Θα επιλέξουμε το καλύτερο από τα γινόμενα:
  - $A_1(A_2 \dots A_n)$
  - $(A_1A_2)(A_3 \dots A_n)$
  - …
  - $(A_1 \dots A_{n-2})(A_{n-1}A_n)$
  - $(A_1 \dots A_{n-1})A_n$
- Ο υπολογισμός της βέλτιστης παρενθετοποίησης για κάθε υποπρόβλημα γίνεται με κλήση της διαδικασίας T(i,j)
- Οριακή Συνθήκη θα έχουμε όταν έχουμε μόνο έναν πίνακα (0 πράξεις)

Β. Θεωρία
1. Δυναμικός Προγραμματισμός
2. Αλυσιδωτός Πολλαπλασιασμός Πινάκων (2. Αναδρομική Σχέση)

Ο αλγόριθμος που υλοποιεί την παραπάνω διαδικασία με αναδρομή είναι ο ακόλουθος:

```
procedure MatMult(Ai,Ai+1,…,Aj)
   if i=j then
      return 0
   else
      for k=i to j-1
         ck=MatMult(Ai,…,Ak)+MatMult(Ak+1,…,Aj)+di-1dkdj
      end for
   end if
   return το ελάχιστο από ck για k=i,…,j-1
end procedure
```

Με βάση τα παραπάνω το βέλτιστο πλήθος των πολλαπλασιασμών της ακολουθίας πινάκων $A_i, \dots, A_j$ δίδεται από την αναδρομική σχέση:

$$M[i,j] = \begin{cases} 0, & i=j \\ \min\limits_{i \le k < j} \{ M[i,k] + M[k+1,j] + d_{i-1} d_k d_j \}, & i<j \end{cases}$$

Β. Θεωρία
1. Δυναμικός Προγραμματισμός
2. Αλυσιδωτός Πολλαπλασιασμός Πινάκων (3.Συνθήκες του Δ.Π.)

- Επαληθεύουμε ότι ισχύουν οι συνθήκες για την κατασκευή ενός αλγόριθμου Δυναμικού Προγραμματισμού:

<u>ΕΠΑΛΗΘΕΥΣΗ ΤΩΝ ΙΔΙΟΤΗΤΩΝ ΤΟΥ Δ.Π. για τον Αλυσιδωτο Πολ/μό Πινάκων:</u>

- <u>Ιδιότητα των Βέλτιστων Επιμέρους Δομών:</u> Πράγματι για να υπολογίσουμε τη βελτιστη λύση, αρκεί να υπολογίσουμε την βέλτιστη λύση στα υποπροβλήματα που προκύπτουν από την αναδρομική σχέση.
- <u>Μικρός Αριθμός Υποπροβλημάτων:</u> Τα υποπροβλήματα που καλούμαστε να λύσουμε είναι το πολύ n(n+1)/2 (όλες οι τιμές με $1 \le i < j \le n$)
- <u>Επικαλυπτόμενα Επιμέρους Προβλήματα:</u> Πράγματι αναγκαζόμαστε να υπολογίσουμε πολλές φορές για παράδειγμα τα γινόμενα A_iA_{i+1}

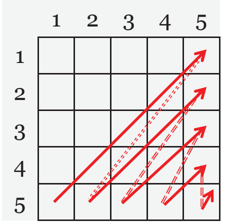
- Συνεπώς θα προσεγγίσουμε το πρόβλημα με Δυναμικό Προγραμματισμό. Θα ξεκινήσουμε από μικρά υποπροβλήματα και θα λύνουμε ολοένα και μεγαλύτερα με σύνθεση των μικρότερων υποπροβλημάτων.

Β. Θεωρία

1. Δυναμικός Προγραμματισμός

2. Αλυσιδωτός Πολλαπλασιασμός Πινάκων (4.Σειρά Επίλυσης Υποπροβ/των)

- Για να αποφασίσουμε με ποια σειρά πρέπει να κάνουμε τις πράξεις παρατηρούμε από ποια υποπροβλήματα εξαρτάται η επίλυση κάθε υποπροβλήματος.
- Π.χ. αν θεωρήσουμε ότι πολλπλασιάζουμε 5 πίνακες, τότε οι πράξεις πρέπι να γίνουν με την εξής σειρά:



- Πρέπει πρώτα να υπολογίσουμε γινόμενα 1 πίνακα, μετά 2 πινάκων, μετά 3 πινάκων κ.ο.κ.
- Το κελί (i,j) θα αποθηκεύει το βέλτιστο πλήθος πράξεων που απαιτούνται στο για τον πολ/σμό της ακολουθίας πινάκων $A_i, A_{i+1}, \ldots A_j$

Β. Θεωρία

1. Δυναμικός Προγραμματισμός

2. Αλυσιδωτός Πολλαπλασιασμός Πινάκων (4.Σειρά Επίλυσης Υποπροβ/των)

- Παράδειγμα Εκτέλεσης για τον πολλαπλασιασμό πινάκων A1A2A3A4, όταν
  - A_1: 5x3, A_2: 3x4, A_3: 4x8, A_4:8x2, A_5: 2x3

	1	2	3	4
5				
4				A4 0
3			A3 0	A3A4 4x8x2=64
2		A2 0	A2A3 3x4x8=96	A2A3A4 (A2A3)A4=96+3x8x2=96+48=144 A2(A3A4)=64+3x4x2=64+24=88 88
1	A1 0	A1A2 5x3x4=60	A1A2A3 (A1A2)A3=60+5x4x8=220 A1(A2A3)=96+5x3x8=216 216	A1A2A3A4 A1(A2A3A4):88+5x3x2=88+30=118 (A1A2)(A3A4):60+64+5x4x2=164 (A1A2A3)A4:216+5x8x2=296 118

Β. Θεωρία

1. Δυναμικός Προγραμματισμός

2. Αλυσιδωτός Πολλαπλασιασμός Πινάκων (5. Επαναληπτικός Αλγόριθμος)

Ο αλγόριθμος που υλοποιεί την παραπάνω διαδικασία σειράς πράξεων είναι ο ακόλουθος:

```
procedure DP_MatMult(A1,A2,…,An)
   for i=1 to n
      m[i,i]=0
   end for
   for p=2 to n
      for i=2 to n-p+1
         j=i+p-1
         m[i,j]=+∞
         for k=1 to j-1
            q=M[i,k]+M[k+1,j]+d[i-1]*d[k]*d[j]
            if (q>M[i,j]) then M[i,j]=q
         end for
      end for
   end for
   return M[1,n]
end procedure
```


Β. Θεωρία

1. Δυναμικός Προγραμματισμός

2. Αλυσιδωτός Πολλαπλασιασμός Πινάκων (6. Ανάλυση Πολυπλοκότητας)

Ακολουθεί ανάλυση πολυπλοκότητας:

- Το πρώτο for έχει πολυπλοκότητα Θ(n)
- Τα δύο επόμενα for υλοποιούν την διαδικασία υπολογισμού των κελιών κατά διαγώνιους όπως είδαμε στην διαδικασία επίλυσης. Τα κελιά αυτά είναι $n^2/2$, άρα οι πράξεις αυτών των for, θα εκτελεστούν $Θ(n^2)$ φορές.
  - Οι πράξεις που γίνονται σε κάθε επανάληψη είναι O(n)
- Συνεπώς η τελική πολυπλοκότητα είναι $O(n^3)$

B. Θεωρία

1. Δυναμικός Προγραμματισμός

2. Αλυσιδωτός Πολλαπλασιασμός Πινάκων (Απομνημόνευση Λύσης)

Ο αλγόριθμος που έχουμε κατασκευάσει:

- Υπολογίζει την βέλτιστη λύση, αποθηκεύοντας την τιμή για την βέλτιστη παρενθετοποιήση κάθε ακολουθίας πινάκων που αντιστοιχεί σε ένα ορισμένο κελί του πίνακα.
- ΔΕΝ επιστρέφει την βέλτιστη λύση, ούτε καταγράφει πως γίνεται η βέλτιστη παρενθετοποιήση

Για το λόγο αυτό τροποποιούμε ελαφρά τον κώδικά μας, ώστε να κάνει και απομνημόνευση της βέλτιστης λύσης κάθε κελιού.

B. Θεωρία

1. Δυναμικός Προγραμματισμός

2. Αλυσιδωτός Πολλαπλασιασμός Πινάκων (Απομνημόνευση Λύσης)

Τροποποιούμε τον αλγόριθμο ως εξής:

```
procedure DP_MatMult(A1,A2,…,An)
   for i=1 to n
      m[i,i]=0
   end for
   for p=2 to n
      for i=2 to n-p+1
         j=i+p-1
         m[i,j]=+∞
         for k=1 to j-1
            q=M[i,k]+M[k+1,j]+d[i-1]*d[k]*d[j]
            if (q>M[i,j]) then M[i,j]=q , s[i,j]=k
         end for
      end for
   end for
   return M[1,n]
end procedure
```

- Η εντολή που προστέθηκε συμβολίζει ότι το σημείο του διαχωρισμού του πολλαπλασιασμού πινάκων $A_iA_{i+1}\dots A_j$ γίνεται στο σημείο s[i,j]=k, δηλαδή ότι η βέλτιστη παρνεθετοποίηση είναι η $(A_iA_{i+1}..A_k)(A_{k+1}\dots A_j)$

B. Θεωρία

1. Δυναμικός Προγραμματισμός

2. Αλυσιδωτός Πολλαπλασιασμός Πινάκων (Απομνημόνευση Λύσης)

Δεδομένου αυτού του πίνακα, μπορούμε να παράγουμε την κατασκευή της βέλτιστης λύσης με την βέλτιστη παρενθετοποίηση, ως εξής:

```
procedure MatrixChainProduct(Ai,A2,…,Aj,s[1…n])
   if i<j
      X=MatrixChainProduct(Ai,…As[i,j],s)
      Y=MatrixChainProduct(As[i,j+1],…Aj,s)
      C=MatMult(X,Y)
      return C
   else
      return Ai
   end if
end procedure
```

- Όπου MatMult(X,Y) είναι ο γνωστός αλγόριθμος πολλαπλασιασμού πινάκων που μελετήσαμε στο Μάθημα 2.1

B. Θεωρία

1. Δυναμικός Προγραμματισμός

3. Μέγιστη Κοινή Υπακολουθία

- ΠΡΟΒΛΗΜΑ: Δίδονται ακολουθίες χαρακτήρων $X=x_1x_2x_3\dots x_n$ και $Y=y_1y_2\dots y_m$. Να βρεθεί η μέγιστη κοινή υπακολουθία τους.
  - Ορισμός: Δεδομένης μιας ακολουθίας $X=x_1x_2\dots x_n$, υπακολουθία της X ονομάζεται οποιαδήποτε ακολουθία μπορεί να παραχθεί με διαγραφή κάποιων στοιχείων της.
- Στιγμιοτύπα:
  - X=abcdf Y=dbdaf, τότε Z=bdf
  - X=aabaabb Y=abacba τότε Z=abab
- Αλγόριθμοι:
  - Προφανής αλγόριθμος: Υπολογίζουμε όλες τις υπακολουθίες των δύο και τις συγκρίνουμε ανά δύο. Πολυπλοκότητα $O(2^m2^n(m+n))$
  - Αλγόριθμος Δυναμικού Προγραμματισμού: Επιτυγχάνεται πολυπλοκότητα $\Theta(mn)$.

Β. Θεωρία
1. Δυναμικός Προγραμματισμός
3. Μέγιστη Κοινή Υπακολουθία (1. Επίλυση με αναδρομή)

Για την επίλυση με αναδρομή του προβλήματος σκεφτόμαστε ως εξής:

- Θεωρούμε ότι έχουμε μια διαδικασία $T(x_1x_2\ldots x_i,\ y_1y_2\ldots y_j)$ που έχει ήδη οριστεί και δουλεύει ορθά που όταν παίρνει ως όρισμα δύο ακολουθίες X,Y παράγει την μεγαλύτερη υπακολουθία τους Z

- Παίρνουμε τώρα το γενικό στιγμιότυπο: $X=x_1x_2\ldots x_{n-1}x_n$ και $Y=y_1y_2\ldots y_{n-1}y_m$ Θα επιλέξουμε το καλύτερο από τα ακόλουθα:
  - Αν ισχύει $x_n=y_m$ τότε θέτουμε σαν μεγαλύτερη υπακολουθία την

    $T(x_1\ldots x_{n-1},\ y_1\ldots y_{m-1})x_n$
  - Αν δεν ισχύει $x_n=y_m$ τότε θέτουμε σαν μεγαλύτερη υπακολουθία όποια είναι μεγαλύτερη από τις:

    $T(x_1\ldots x_n,\ y_1\ldots y_{m-1})$

    και

    $T(x_1\ldots x_{n-1},\ y_1\ldots y_m)$

Β. Θεωρία
1. Δυναμικός Προγραμματισμός
3. Μέγιστη Κοινή Υπακολουθία (2. Αναδρομική Σχέση)

- Η αναδρομική σχέση που υπολογίζει την βέλτιστη λύση είναι:

$$f_n=\begin{cases} 0, & i=0\ \acute{\eta}\ \ j=0 \\ c[i-1,j-1]+1, & i,j>0\ \ \kappa\alpha\iota\ x_i=y_j \\ \max\{c[i,j-1],c[i-1,j]\}, & i,j>0\ \ \kappa\alpha\iota\ x_i\neq y_j \end{cases}$$

Β. Θεωρία
1. Δυναμικός Προγραμματισμός
3. Μέγιστη Κοινή Υπακολουθία (3.Συνθήκες του Δ.Π.)

- Επαληθεύουμε ότι ισχύουν οι συνθήκες για την κατασκευή ενός αλγόριθμου Δυναμικού Προγραμματισμού:

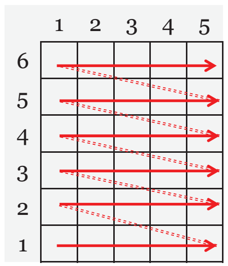
> ΕΠΑΛΗΘΕΥΣΗ ΤΩΝ ΙΔΙΟΤΗΤΩΝ ΤΟΥ Δ.Π. για την Μεγιστη Κοινή Υπακολουθία:
> - Ιδιότητα των Βέλτιστων Επιμέρους Δομών: Πράγματι για να υπολογίσουμε τη βελτιστη λύση, αρκεί να υπολογίσουμε την βέλτιστη λύση στα υποπροβλήματα που προκύπτουν από την αναδρομική σχέση.
> - Μικρός Αριθμός Υποπροβλημάτων: Τα υποπροβλήματα που καλούμαστε να λύσουμε είναι το πολύ nm (όλες οι τιμές με 1≤i,j ≤n)
> - Επικαλυπτόμενα Επιμέρους Προβλήματα: Λόγω της λειτουργίας της αναδρομής, υπολογίζονται πολλές φορές οι ίδιες υπακολουθίες.

- Συνεπώς θα προσεγγίσουμε το πρόβλημα με Δυναμικό Προγραμματισμό. Θα ξεκινήσουμε από μικρά υποπροβλήματα και θα λύνουμε ολοένα και μεγαλύτερα με σύνθεση των μικρότερων υποπροβλημάτων.

Β. Θεωρία
1. Δυναμικός Προγραμματισμός
3. Μέγιστη Κοινή Υπακολουθία (4.Σειρά Επίλυσης Υποπροβ/των)

- Για να αποφασίσουμε με ποια σειρά πρέπει να κάνουμε τις πράξεις παρατηρούμε από ποια υποπροβλήματα εξαρτάται η επίλυση κάθε υποπροβλήματος.

- Π.χ. αν θεωρήσουμε ότι έχουμε 2 ακολουθίες με μήκος 6 και 5. Το στοιχείο M[i,j] συμβολίζει ότι συγκρίνω τις ακολουθίες $x_1x_2\ldots x_i$ και $y_1y_2\ldots y_j$



Β. Θεωρία

1. Δυναμικός Προγραμματισμός

3. Μέγιστη Κοινή Υπακολουθία (4.Σειρά Επίλυσης Υποπροβ/των)

- Παράδειγμα εκτέλεσης για X=abcdf Y=dbdaf

	1	2	3	4	5
5	X=abcdf Y=d c=1	X=abcdf Y=db c=1	X=abcdf Y=dbd c=2	X=abcdf Y=dbda c=2	X=abcdf Y=dbdaf c=3
4	X=abcd Y=d c=1	X=abcd Y=db c=1	X=abcd Y=dbd c=2	X=abcd Y=dbda c=2	X=abcd Y=dbdaf c=2
3	X=abc Y=d c=0	X=abc Y=db c=1	X=abc Y=dbd c=1	X=abc Y=dbda c=1	X=abc Y=dbdaf c=1
2	X=ab Y=d c=0	X=ab Y=db c=1	X=ab Y=dbd c=1	X=ab Y=dbda c=1	X=ab Y=dbdaf c=1
1	X=a Y=d c=0	X=a Y=db c=0	X=a Y=dbd c=0	X=a Y=dbda c=1	X=a Y=dbdaf c=1

Β. Θεωρία

1. Δυναμικός Προγραμματισμός

3. Μέγιστη Κοινή Υπακολουθία (5. Επαναληπτικός Αλγόριθμος)

Ο αλγόριθμος που υλοποιεί την παραπάνω διαδικασία σειράς πράξεων είναι ο ακόλουθος:

```
procedure LCS(X,Y)
   for i=1 to n : c[i,0]=0
   for j=1 to m : c[0,j]=0
   for i=1 to n
      for j=1 to m
         if xi=yj then
            c[i,j]=c[i-1,j-1]+1
         else
            if (c[i-1,j]>c[i,j-1]) then
               c[i,j]=c[i-1,j]
            else
               c[i,j]=c[i,j-1]
            end if
         end if
      end for
   end for
   return c[n,m]
end procedure
```


Β. Θεωρία

1. Δυναμικός Προγραμματισμός

3. Μέγιστη Κοινή Υπακολουθία (Κατασκευή Βέλτιστης Λύσης)

- Δεδομένου πίνακα C[M,N] που κατασκευάσαμε μπορούμε να εξάγουμε την βέλτιστη λύση σε χρόνο Θ(m+n) με την ακόλουθη αναδρομική ρουτίνα:

```
procedure printLCS(X[i],Y[j])
   if i>0 and j>0
      if xi=yj then
         printLCS(X[i-1],Y[j-1])
         print(x[i])
      else if c[i,j]=c[i-1,j] then
         printLCS(X[i-1],Y[j])
      else
         printLCS(X[i],Y[j-1])
      end if
   end if
end procedure
```


Β. Θεωρία

1. Δυναμικός Προγραμματισμός

3. Μέγιστη Κοινή Υπακολουθία (Εκτέλεση του αλγόριθμου κατασκευής Β.Λ.)

- Βλέπουμε πως θα τρέξει ο αναδρομικός αλγόριθμος για την κατασκευή της βέλτιστης λύσης:

	1	2	3	4	5
5	X=abcdf Y=d c=1	X=abcdf Y=db c=1	X=abcdf Y=dbd c=2	X=abcdf Y=dbda c=2	X=abcd**f** Y=dbda**f** c=3
4	X=abcd Y=d c=1	X=abcd Y=db c=1	X=abc**d** Y=db**d** c=2	X=abcd Y=dbda c=2	X=abcd Y=dbdaf c=2
3	X=abc Y=d c=0	X=abc Y=db c=1	X=abc Y=dbd c=1	X=abc Y=dbda c=1	X=abc Y=dbdaf c=1
2	X=ab Y=d c=0	X=a**b** Y=d**b** c=1	X=ab Y=dbd c=1	X=ab Y=dbda c=1	X=ab Y=dbdaf c=1
1	X=a Y=d c=0	X=a Y=db c=0	X=a Y=dbd c=0	X=a Y=dbda c=1	X=a Y=dbdaf c=1

Β. Θεωρία
1. Δυναμικός Προγραμματισμός
3. Μέγιστη Κοινή Υπακολουθία (6.Πολυπλοκότητα Αλγορίθμου ΔΠ)

Υπολογισμός Πολυπλοκότητας:

- Οι πρώτες 2 for έχουν πολυπλοκότητα Θ(n)
- Έπειτα για nm φορές γίνονται Θ(1) πράξεις. Συνεπώς η πολυπλοκότητα είναι Θ(nm).

Συνεπώς η συνολική πολυπλοκότητα είναι Θ(n)+Θ(nm)=Θ(nm)

Γ. Ασκήσεις
Εφαρμογή 1

Δίδεται η ακολουθία ακεραίων αριθμών που δημιουργείται από την
$X(n) = 4X(n-2) + 2\,X(n-4)$, για $n > 3$, και $X(n) = 1$, $n=0,1,2,3$

1. Σχεδιάστε έναν αναδρομικό αλγόριθμο που θα υπολογίζει τον n-οστό όρο της ακολουθίας. Γράψτε την αναδρομική σχέση που δίνει τον χρόνο εκτέλεσης του αλγόριθμου. Βρείτε συμβατά άνω και κάτω όρια για τον χρόνο εκτέλεσης του αλγορίθμου (δηλαδή ακριβές όριο Θ).
2. Σχεδιάστε έναν αλγόριθμο δυναμικού προγραμματισμού που λύνει το ίδιο πρόβλημα Ποιος είναι τώρα ο χρόνος εκτέλεσης?

Γ. Ασκήσεις
Εφαρμογή 2

Έστω ένας πίνακας ακεραίων αριθμών A που αποτελείται από m γραμμές και n στήλες. Θεωρούμε ένα παιχνίδι όπου ξεκινώντας από οποιοδήποτε κελί της κάτω γραμμής του πίνακα, προσπαθούμε να φτάσουμε σε κάποιο κελί της πάνω γραμμής του πίνακα περνώντας από κελιά ελάχιστου συνολικού κόστους. Σαν κόστος A(i,j) ενός κελιού (i,j) θεωρούμε τον αριθμό που αναγράφεται στο συγκεκριμένο κελί. Από ένα κελί του πίνακα μπορούμε να κινηθούμε είτε προς το κελί που βρίσκεται ακριβώς από πάνω του, είτε διαγώνια αριστερά, είτε διαγώνια δεξιά.

Για παράδειγμα, στον πίνακα του πιο κάτω σχήματος, από το κελί (2,2) μπορούμε να κινηθούμε προς κάποιο από τα κελιά (3,1), (3,2) ή (3,3). Σημειώνουμε ότι σαν γραμμή με αριθμό 1 θεωρείται η κάτω γραμμή του πίνακα.

3	9	9	**6**	9
5	6	7	**3**	4
6	8	7	7	**2**
4	3	6	**5**	9

Τα κελιά με έντονη γραφή δείχνουν τη βέλτιστη διαδρομή στον συγκεκριμένο πίνακα, συνολικού κόστους ίσου με 5+2+3+6=16.

Γ. Ασκήσεις
Εφαρμογή 2

α) Να περιγράψετε μία λύση του πιο πάνω προβλήματος με τη βοήθεια του Δυναμικού Προγραμματισμού.
(β) Να δώσετε τον αντίστοιχο αλγόριθμο (ψευδοκώδικα) υπολογισμού του ελάχιστου κόστους μετάβασης από την πρώτη γραμμή του πίνακα προς την τελευταία.
(γ) Ποια η πολυπλοκότητα χρόνου και χώρου του αλγόριθμου αυτού; Θα μπορούσε να εφαρμοστεί ισοδύναμος αλγόριθμος με μικρότερη πολυπλοκότητα χώρου; Για το τελευταίο ερώτημα αρκεί να περιγράψετε τον τρόπο μείωσης της πολυπλοκότητας χώρου, χωρίς να δώσετε ακριβή περιγραφή του νέου αλγόριθμου.
(δ) Να εφαρμόσετε τον αλγόριθμο που περιγράψατε στα προηγούμενα σκέλη στον πιο κάτω πίνακα:

4	3	4
2	3	4
4	1	3
4	4	3

Γ. Ασκήσεις
Εφαρμογή 3

Οι διοργανωτές ερασιτεχνικού ποδηλατικού αγώνα θέλουν να σχεδιάσουν, για ευνόητους λόγους μειωμένης φυσικής κατάστασης των συμμετεχόντων ερασιτεχνών ποδηλάτων, την οδική διαδρομή μικρότερης συνολικής ανωφέρειας (ανηφορικής κλίσεως). Ο χάρτης με τη διαδρομή (άκυκλο κατευθυνόμενο γράφημα) εμφανίζεται παρακάτω, όπου κάθε δρόμος αναγράφει την ανωφέρεια του. Η αφετηρία του αγώνα είναι η πόλη P_0 και τέλος η πόλη P_8.
(1) Σχεδιάστε αλγόριθμο δυναμικού προγραμματισμού που υπολογίζει την συνολική ελάχιστη ανωφέρεια.
(2) Τρέξτε τον αλγόριθμο στο παράδειγμα του χάρτη.