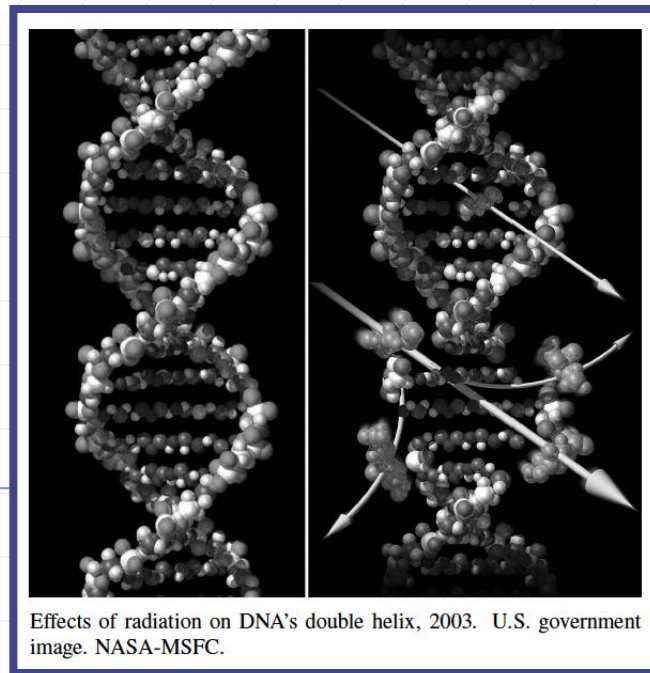


Παρουσίαση για χρήση με το σύγγραμμα, **Αλγόριθμοι Σχεδίαση και Εφαρμογές**, των Μ. Τ. Goodrich and R. Tamassia, Wiley, 2015 (στα ελληνικά από εκδόσεις Μ. Γκιούρδας)

Δυναμικός προγραμματισμός



Εφαρμογή: Αντιστοίχιση αλληλουχιών DNA

- ◆ Οι αλληλουχίες DNA μπορούν να θεωρηθούν συμβολοσειρές χαρακτήρων **A**, **C**, **G**, και **T**, οι οποίες αναπαριστούν νουκλεοτίδια.
- ◆ Η εύρεση των ομοιοτήτων μεταξύ δύο αλληλουχιών DNA είναι μια σημαντική πράξη, που εκτελείται στη βιοπληροφορική.
 - Για παράδειγμα, όταν συγκρίνουμε το DNA διαφορετικών οργανισμών, τέτοιες αντιστοιχίσεις μπορούν να επισημάνουν τα σημεία, στα οποία αυτοί οι οργανισμοί έχουν παρόμοια μοτίβα DNA.

Εφαρμογή: Αντιστοίχιση αλληλουχιών DNA

- ◆ Η εύρεση της καλύτερης αντιστοίχισης συμβολοσειρών DNA αφορά την ελαχιστοποίηση του αριθμού των αλλαγών για να μετατρέψουμε την μία συμβολοσειρά στην άλλη.other.

```
X: ACCGGTCGAGTGCGCGGAAGCCGGCCGAA
    |  |  |  |  |  |  |  |  |  |  |  |  |
    G TC  GT CG G AAGCCGGCCGAA
    GTCGT CGGAA GCCG  GC C G AA
    | | | | | | | | | | | | | | |
Y:  GTCGTTCGGAATGCCGTTGCTCTGTAA
```

Figure 12.1: Two DNA sequences, X and Y, and their alignment in terms of a longest subsequence, GTCGTTCGGAAGCCGGCCGAA, that is common to these two strings.

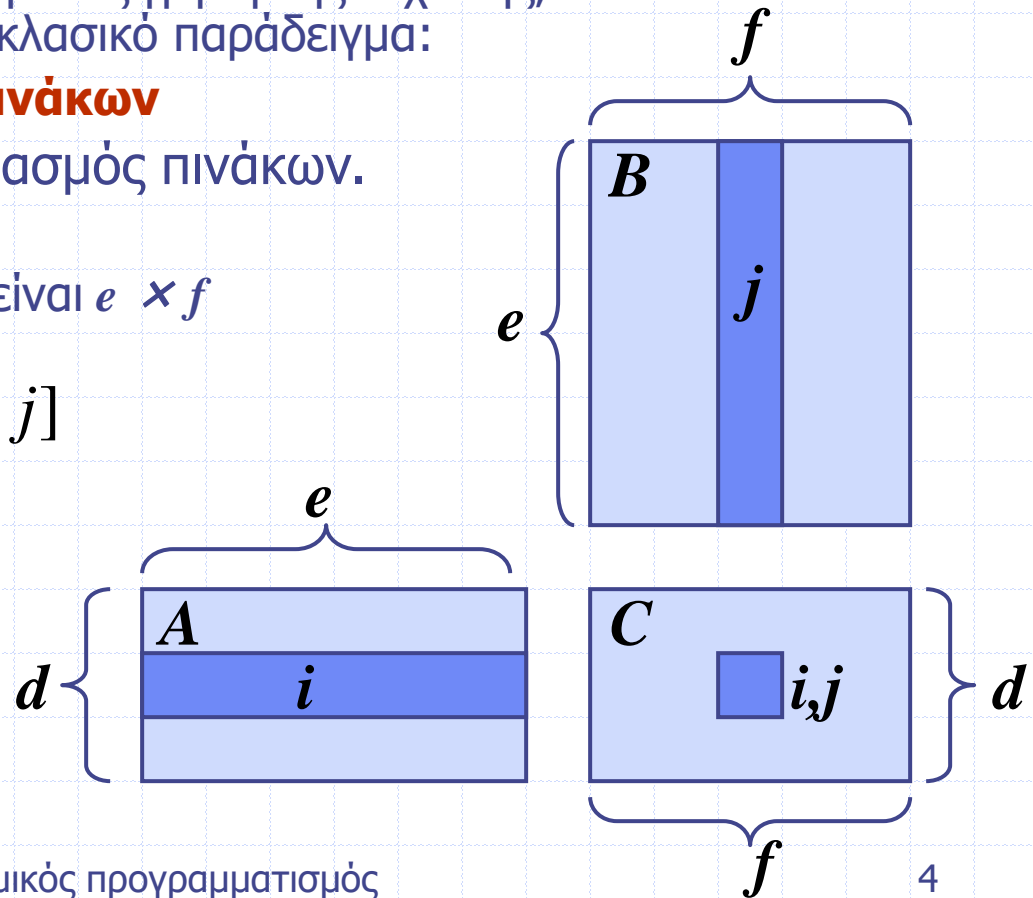
- ◆ Μία αναζήτηση ωμής δύναμης θα απαιτούσε εκθετικό χρόνο αλλά μπορούμε να το επιτύχουμε πολύ καλύτερα χρησιμοποιώντας **δυναμικό προγραμματισμό**.

Προθέρμανση: Γινόμενα αλυσίδας πινάκων

- ♦ **Ο Δυναμικός προγραμματισμός** είναι μία τεχνική αλγοριθμικής σχεδίασης.
 - Αντί να ξεκινήσουμε με μία εξήγηση της τεχνικής, θα ξεκινήσουμε με ένα κλασικό παράδειγμα:
 - **Γινόμενα αλυσίδας πινάκων**
- ♦ Επανάληψη: Πολλαπλασιασμός πινάκων.
 - $C = A * B$
 - Ο A είναι $d \times e$ και ο B είναι $e \times f$

$$C[i, j] = \sum_{k=0}^{e-1} A[i, k] * B[k, j]$$

- $O(def)$ χρόνος



Γινόμενα αλυσίδας πινάκων

◆ Γινόμενα αλυσίδας πινάκων:

- Υπολογισμός $A = A_0 * A_1 * \dots * A_{n-1}$
- Ο A_i είναι $d_i \times d_{i+1}$
- Πρόβλημα: Που θα μπουν οι παρενθέσεις?

◆ Παράδειγμα

- Ο B είναι 3×100
- Ο C είναι 100×5
- Ο D είναι 5×5
- Το $(B * C) * D$ θέλει $1500 + 75 = 1575$ ops
- Το $B * (C * D)$ θέλει $1500 + 2500 = 4000$ ops

Προσέγγιση απαρίθμησης



◆ Αλγόριθμος γινόμενου αλυσίδας πινάκων:

- Δοκιμή όλων πιθανών τρόπων για παρενθετοποίηση του $TA = A_0 * A_1 * \dots * A_{n-1}$
- Υπολογισμός του αριθμού των λειτουργιών (ops) για κάθε τρόπο
- Επιλογή του καλύτερου

◆ Χρόνος εκτέλεσης:

- Ο αριθμός των πιθανών τρόπων για να μπουν οι παρενθέσεις είναι ίσος με τον αριθμό δυαδικών δένδρων με n κόμβου
- Είναι **εκθετικό**!
- Ονομάζεται αριθμός Catalan, και είναι σχεδόν 4^n .
- Είναι απαίσιος αλγόριθμος!



Μία άπληστη μέθοδος

◆ Ιδέα #1: συνεχόμενη επιλογή του γινομένου που απαιτεί τις περισσότερες λειτουργίες.

◆ Αντί-παράδειγμα:

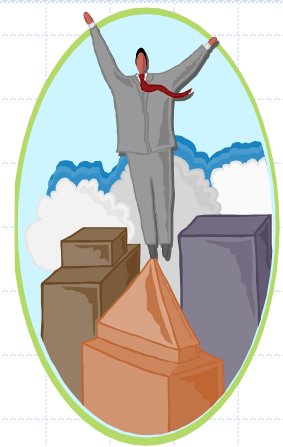
- Ο A είναι 10×5
- Ο B είναι 5×10
- Ο C είναι 10×5
- Ο D είναι 5×10
- Η άπληστη ιδέα #1 δίνει $(A*B)*(C*D)$, που θέλει $500+1000+500 = 2000$ ops
- Το $A*((B*C)*D)$ θέλει $500+250+250 = 1000$ ops

Ακόμη μία άπληστη προσέγγιση



- ◆ Ιδέα #2: συνεχόμενη επιλογή του γινομένου που χρησιμοποιεί τις περισσότερες λειτουργίες.
- ◆ **Αντί-παράδειγμα:**
 - Ο A είναι 101×11
 - Ο B είναι 11×9
 - Ο C είναι 9×100
 - Ο D είναι 100×99
 - Η άπληστη ιδέα #2 δίνει $A*((B*C)*D)$, που θέλει $109989+9900+108900=228789$ ops
 - Το $(A*B)*(C*D)$ θέλει $9999+89991+89100=189090$ ops
- ◆ Η άπληστη προσέγγιση δεν μας δίνει την βέλτιστη λύση.

Μία «αναδρομική» προσέγγιση



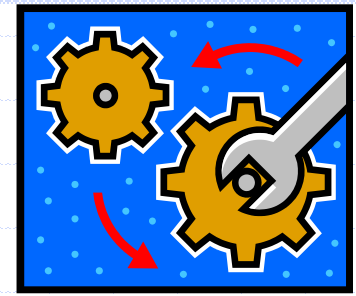
◆ Ορισμός **υπό-προβλημάτων**:

- Εύρεση του καλύτερου τρόπου για παρενθετοποίηση του $A_i * A_{i+1} * \dots * A_j$.
- Το $N_{i,j}$ υποδηλώνει τον αριθμό των λειτουργιών που απαιτεί το υπό-πρόβλημα.
- Η βέλτιστη λύση για το συνολικό πρόβλημα είναι $N_{0,n-1}$.

◆ **Άριστα υπό-προβλήματα**: Η βέλτιστη λύση μπορεί να εκφραστεί με όρους άριστων υπό-προβλημάτων

- Πρέπει να υπάρχει ένας τελικός πολλαπλασιασμός για την βέλτιστη λύση.
- Έστω ότι ο τελικός πολλαπλασιασμός είναι στο i : $(A_0 * \dots * A_i) * (A_{i+1} * \dots * A_{n-1})$.
- Τότε η βέλτιστη λύση $N_{0,n-1}$ είναι το άθροισμα δύο άριστων υπό-προβλημάτων, το $N_{0,i}$ και το $N_{i+1,n-1}$ συν τον χρόνο για τον τελευταίο πολλαπλασιασμό.
- Εάν το καθολικό βέλτιστο δεν είχε αυτά τα άριστα υπό-προβλήματα θα μπορούσαμε να βρούμε μία ακόμη καλύτερη «βέλτιστη» λύση.

Μία εξίσωση χαρακτηρισμού

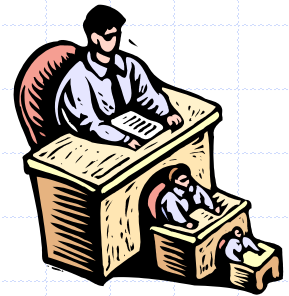


- ◆ Το καθολικό βέλτιστο θα πρέπει να οριστεί σε όρους άριστων υπό-προβλημάτων.
- ◆ Ας σκεφτούμε όλες τις πιθανές θέσεις για τον τελικό πολλαπλασιασμό:
 - Θυμηθείτε ότι ο A_i είναι ένας πίνακας διαστάσεων $d_i \times d_{i+1}$.
 - Έτσι, μια χαρακτηριστική εξίσωση για το $N_{i,j}$ είναι η εξής:

$$N_{i,j} = \min_{i \leq k < j} \{ N_{i,k} + N_{k+1,j} + d_i d_{k+1} d_{j+1} \}$$

- ◆ Σημειώστε ότι τα υπό-προβλήματα δεν είναι ανεξάρτητα
τα **υπάρχει αλληλοεπικάλυψη στα υπό-προβλήματα.**

Ένας αλγόριθμος δυναμικού προγραμματισμού



- ◆ Επειδή υπάρχει επικάλυψη στα υπό-προβλήματα, δεν χρησιμοποιούμε αναδρομή.
- ◆ Αντί αυτού, δημιουργούμε άριστα υπό-προβλήματα «από κάτω προς τα πάνω»
- ◆ Τα $N_{i,i}$ είναι εύκολα, έτσι ξεκινάμε από αυτά
- ◆ Μετά υπό-προβλήματα μεγέθους 2,3,... και έτσι στην συνέχεια.
- ◆ Ο χρόνος εκτέλεσης είναι $O(n^3)$

Algorithm *matrixChain*(S):

Input: sequence S of n matrices to be multiplied

Output: number of operations in an optimal paranethization of S

for $i \leftarrow 1$ **to** $n-1$ **do**

$N_{i,i} \leftarrow 0$

for $b \leftarrow 1$ **to** $n-1$ **do**

for $i \leftarrow 0$ **to** $n-b-1$ **do**

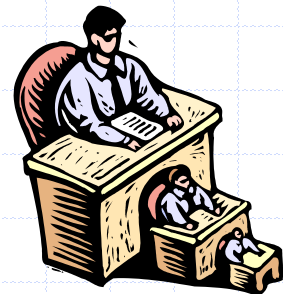
$j \leftarrow i+b$

$N_{i,j} \leftarrow +\text{infinity}$

for $k \leftarrow i$ **to** $j-1$ **do**

$N_{i,j} \leftarrow \min\{N_{i,j}, N_{i,k} + N_{k+1,j} + d_i d_{k+1} d_{j+1}\}$

Οπτικοποίηση αλγόριθμου δυναμικού προγραμματισμού



$$N_{i,j} = \min_{i \leq k < j} \{N_{i,k} + N_{k+1,j} + d_i d_{k+1} d_{j+1}\}$$

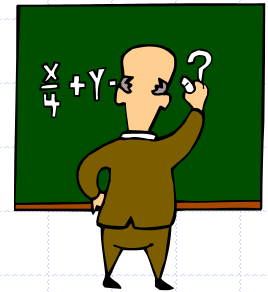
- ◆ Η κατασκευή από κάτω προς τα πάνω γεμίζει τον πίνακα N κατά διαγώνιους
- ◆ Το $N_{i,j}$ παίρνει τις τιμές από την γραμμή i και την στήλη j
- ◆ Το γέμισμα κάθε κελιού στον πίνακα N θέλει χρόνο $O(n)$.
- ◆ Συνολικός χρόνος: $O(n^3)$
- ◆ Η εύρεση της παρενθετοποίησης γίνεται με το να θυμάται το “k” για κάθε κελί στον N

N	0	1	2			j	...	n-1
0								
1								
...								
i								
n-1								

answer



Η γενική τεχνική δυναμικού προγραμματισμού



- ◆ Εφαρμόζεται σε προβλήματα που αρχικά φαίνεται να απαιτούν πολύ χρόνο (πιθανότατα εκθετικό), αρκεί να έχουμε:
 - **Απλά υπό-προβλήματα:** τα υπό-προβλήματα μπορούν να οριστούν με λίγους μόνο δείκτες, όπως j, k, l, m , κ.ο.κ..
 - **Άριστα υπό-προβλήματα:** η καθολική βέλτιστη λύση μπορεί είναι μία σύνθεση των βέλτιστων λύσεων στα υπό-προβλήματα.
 - **Επικάλυψη υπό-προβλημάτων:** τα υπό-προβλήματα δεν είναι ανεξάρτητα, αλλά επικαλύπτονται (οπότε πρέπει να κατασκευαστούν από κάτω προς τα πάνω).