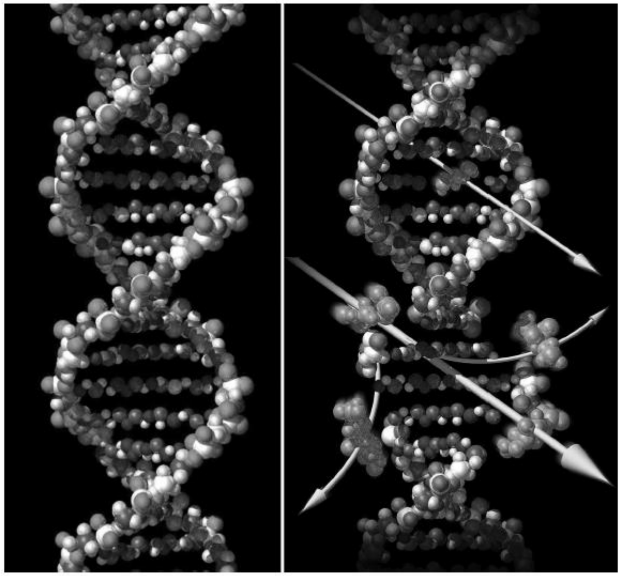
Παρουσίαση για χρήση με το σύγγραμμα, Αλγόριθμοι Σχεδίαση και Εφαρμογές, των M. T. Goodrich and R. Tamassia, Wiley, 2015 (στα ελληνικά από εκδόσεις Μ. Γκιούρδας)

Δυναμικός προγραμματισμός



Effects of radiation on DNA's double helix, 2003. U.S. government image. NASA-MSFC.

Εφαρμογή: Αντιστοίχιση ακολουθιών DNA

- Οι ακολουθίες DNA μπορούν να θεωρηθούν ως συμβολοσειρές αποτελούμενες από τους χαρακτήρες **A**, **C**, **G**, **T**, οι οποίες αναπαριστούν νουκλεοτίδια.
- Η εύρεση ομοιοτήτων ανάμεσα σε δύο ακολουθίες DNA αποτελεί μια σημαντική πράξη 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 ακολουθιών DNA

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

```
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, GTCGTCGGAAGCCGGCCGAA, that is common to these two strings.

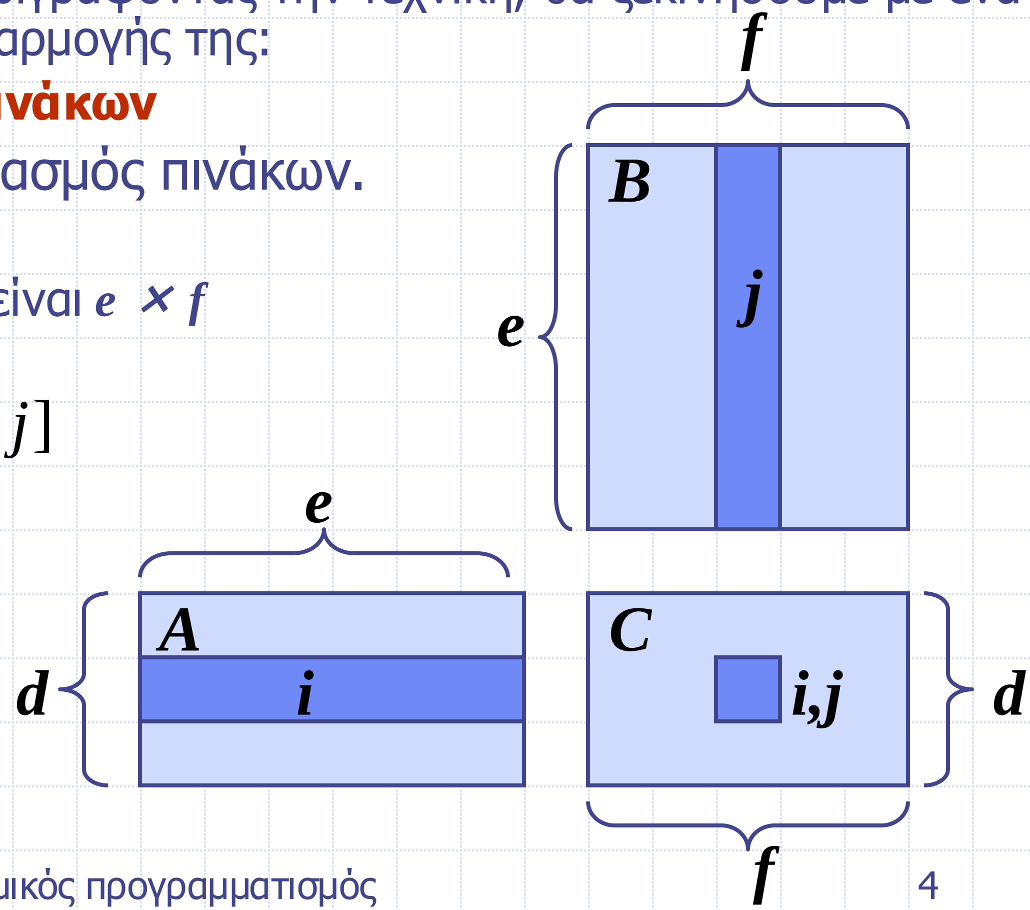
- Μία αναζήτηση ωμής δύναμης (brute force) θα απαιτούσε εκθετικό χρόνο αλλά μπορούμε να πετύχουμε πολύ καλύτερα αποτελέσματα χρησιμοποιώντας **δυναμικό προγραμματισμό**.

Προθέρμανση: Γινόμενα αλυσίδας πινάκων

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

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

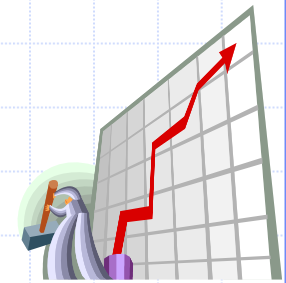
  - $O(def)$ χρόνος



Γινόμενα αλυσίδας πινάκων

- **Γινόμενα αλυσίδας πινάκων:**
  - Υπολογισμός $A=A_0*A_1*\ldots*A_{n-1}$
  - 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 3*100*5 + 3*5*5 = 1500 + 75 = 1575 ops (ops = αριθμός λειτουργιών)
  - Το B*(C*D) θέλει 3*100*5 + 100*5*5 = 1500 + 2500 = 4000 ops

Προσέγγιση απαρίθμησης



- **Αλγόριθμος γινομένου αλυσίδας πινάκων:**
  - Δοκιμή όλων των πιθανών τρόπων για τοποθέτηση παρενθέσεων στο $A=A_0*A_1*\ldots*A_{n-1}$
  - Υπολογισμός του αριθμού των λειτουργιών (ops) για κάθε τρόπο.
  - Επιλογή του καλύτερου τρόπου τοποθέτησης παρενθέσεων.
- Χρόνος εκτέλεσης:
  - Ο αριθμός των πιθανών τρόπων για να τοποθετηθούν οι παρενθέσεις είναι ίσος με τον αριθμό δυαδικών δένδρων που έχουν n φύλλα.
  - Είναι **εκθετικό**!
  - Ονομάζεται αριθμός Catalan, και είναι σχεδόν 4^n
  - Είναι πολύ κακός αλγόριθμος!

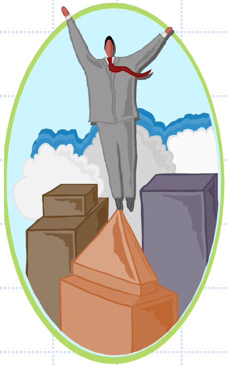
Μία άπληστη προσέγγιση

- Ιδέα #1: συνεχόμενη επιλογή του γινομένου που απαιτεί **τις περισσότερες** λειτουργίες.
- Αντι-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
- Η άπληστη προσέγγιση δεν μας δίνει τη βέλτιστη λύση.

Ακόμη μία άπληστη προσέγγιση

- Ιδέα #2: συνεχόμενη επιλογή του γινομένου που απαιτεί **τις λιγότερες** λειτουργίες.
- Αντι-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 τη βέλτιστη λύση.

Μία «αναδρομική» προσέγγιση

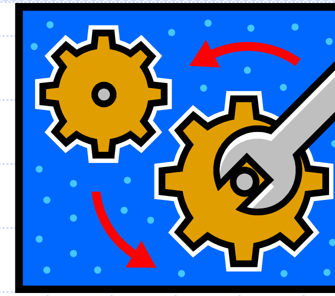


- Ορισμός **υπό-προβλημάτων**:
  - Εύρεση του καλύτερου τρόπου για τοποθέτηση παρενθέσεων στο $A_i*A_{i+1}*\ldots*A_j$.
  - Το $N_{i,j}$ υποδηλώνει τον αριθμό των λειτουργιών που απαιτεί το συγκεκριμένο υπό-πρόβλημα.
  - Η βέλτιστη λύση για το συνολικό πρόβλημα είναι η $N_{0,n-1}$.
- **Βέλτιστα υπό-προβλήματα**: Η βέλτιστη λύση μπορεί να εκφραστεί με όρους βέλτιστων υπό-προβλημάτων.
  - Πρέπει να υπάρχει ένας τελικός πολλαπλασιασμός για τη βέλτιστη λύση.
  - Έστω ότι ο τελικός πολλαπλασιασμός είναι στο i:

    $$(A_0*\ldots*A_i)*(A_{i+1}*\ldots*A_{n-1})$$

  - Τότε η βέλτιστη λύση $N_{0,n-1}$ είναι το άθροισμα δύο βέλτιστων υπό-προβλημάτων, του $N_{0,i}$ και του $N_{i+1,n-1}$ συν το χρόνο για τον τελευταίο πολλαπλασιασμό.
  - Εάν το καθολικό βέλτιστο δεν είχε αυτά τα βέλτιστα υπό-προβλήματα θα μπορούσαμε να βρούμε μία ακόμα καλύτερη «βέλτιστη» λύση.

Μία χαρακτηριστική εξίσωση

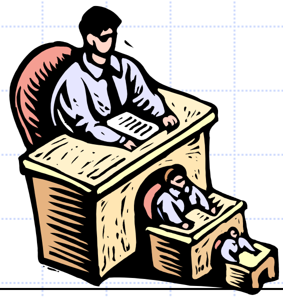


- Το καθολικό βέλτιστο θα πρέπει να οριστεί με όρους βέλτιστων υπό-προβλημάτων.
- Ας σκεφτούμε όλες τις πιθανές θέσεις για τον τελικό πολλαπλασιασμό:
  - Θυμηθείτε ότι ο A_i είναι ένας πίνακας διαστάσεων $d_i \times d_{i+1}$.
  - Έτσι, μια χαρακτηριστική εξίσωση για το $N_{i,j}$ είναι η εξής:

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

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

Ένας αλγόριθμος δυναμικού προγραμματισμού



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

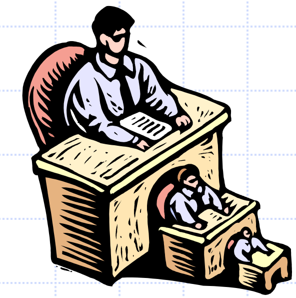
Algorithm ***matrixChain***(***S***):

Input: sequence ***S*** of ***n*** matrices to be multiplied

Output: number of operations in an optimal parenthesizing of ***S***

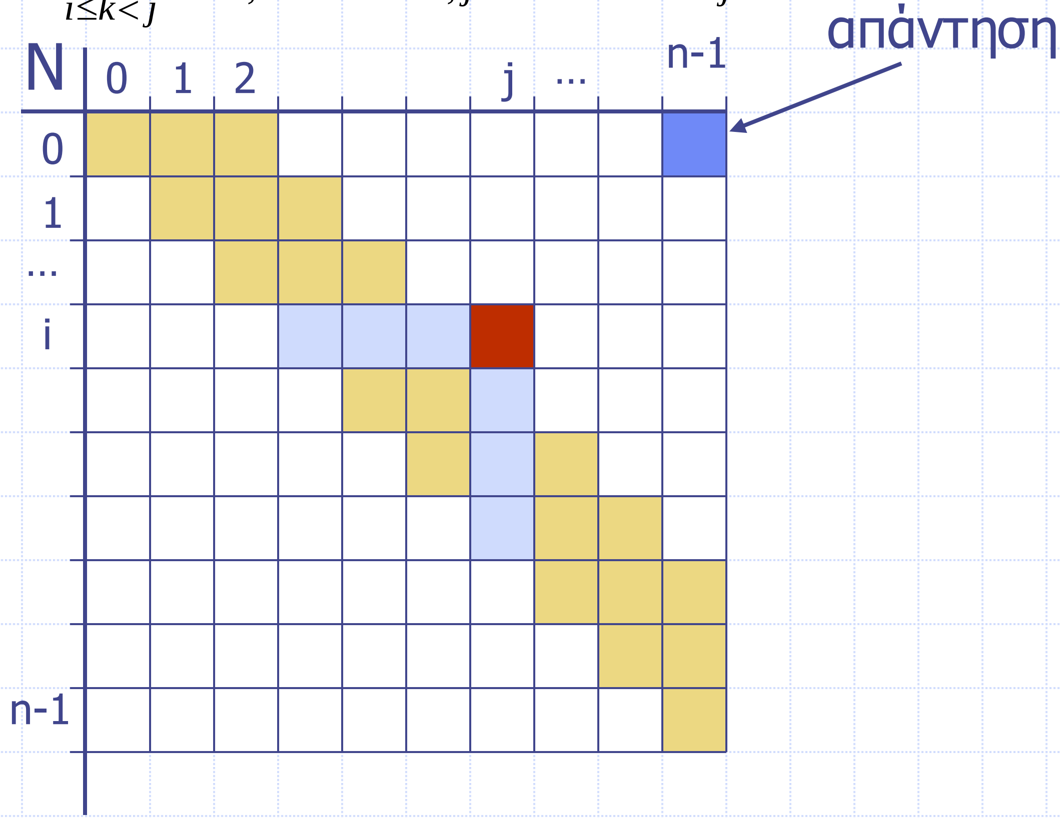
for $i \leftarrow 0$ **to** $n-1$ **do**
  $N_{i,i} \leftarrow 0$
for $b \leftarrow 1$ **to** $n-1$ **do**
  for $i \leftarrow 0$ **to** $n-b-1$ **do**
    $j \leftarrow i+b$
    $N_{i,j} \leftarrow +\infty$
    for $k \leftarrow i$ **to** $j-1$ **do**
      $N_{i,j} \leftarrow \min\{N_{i,j}, N_{i,k} + N_{k+1,j} + d_i d_{k+1} d_{j+1}\}$

Οπτικοποίηση αλγόριθμου δυναμικού προγραμματισμού

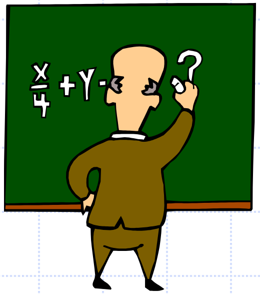


$$N_{i,j} = \min_{i \le k < j} \{ N_{i,k} + N_{k+1,j} + d_i d_{k+1} d_{j+1} \}$$

- 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 του τελικά ζητούμενου τρόπου τοποθέτησης παρενθέσεων γίνεται με την καταγραφή του κατάλληλου “k” για κάθε τιμή του N



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 να μπορούν να οριστούν χρησιμοποιώντας λίγες μόνο μεταβλητές, όπως j, k, l, m, κ.ο.κ.
  - **Βελτιστότητα υπό-προβλημάτων:** η καθολικά βέλτιστη λύση να μπορεί να οριστεί με όρους βέλτιστων λύσεων υπό-προβλημάτων.
  - **Επικάλυψη υπό-προβλημάτων:** τα υπό-προβλήματα να μην είναι ανεξάρτητα, αλλά να επικαλύπτονται (οπότε να πρέπει να κατασκευαστούν από κάτω προς τα πάνω).