ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ

ΤΜΗΜΑ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΑΝΑΛΥΣΗ ΑΛΓΟΡΙΘΜΩΝ

Δυναμικός προγραμματισμός

Κωνσταντίνος Γιαννουτάκης
Επίκουρος Καθηγητής

ΣΥΝΟΨΗ ΔΙΑΛΕΞΗΣ

- Δυναμικός Προγραμματισμός

ΕΙΣΑΓΩΓΗ

- *Ορισμοί*
- *Υπολογισμός Ακολουθίας Fibonacci*

ΕΙΣΑΓΩΓΗ

Δυναμικός προγραμματισμός (dynamic programming) είναι μια υπολογιστική μέθοδος που εφαρμόζεται **όταν τα υποπροβλήματα δεν είναι ανεξάρτητα μεταξύ τους**.

Είναι μέθοδος επίλυσης προβλημάτων μέσω του συνδυασμού λύσεων κάποιων **υποπροβλημάτων** τους (όπως η τεχνική Διαίρει και Βασίλευε).

Αναφέρεται σε μια μέθοδο τήρησης στοιχείων μέσω πινάκων και όχι στη σύνταξη κώδικα, όπως υπονοεί ο όρος προγραμματισμός.

ΕΙΣΑΓΩΓΗ

Οι αλγόριθμοι τύπου διαίρει και κυρίευε αναλύουν το πρόβλημα σε **ανεξάρτητα υποπροβλήματα** μεταξύ τους.

Ο **δυναμικός προγραμματισμός** χρησιμοποιείται όταν τα διάφορα **υποπροβλήματα επικαλύπτονται**.

Ένας αλγόριθμος δυναμικού προγραμματισμού **επιλύει μία φορά κάθε υποπρόβλημα** και **αποθηκεύει την λύση του σε έναν πίνακα**, στον οποίο μπορεί να βρίσκει τη λύση κάθε φορά που συναντά κάποιο επιλυμένο υποπρόβλημα.

ΕΙΣΑΓΩΓΗ

- Χρησιμοποιείται κυρίως για προβλήματα βελτιστοποίησης
  - Μπορεί να έχουν πολλές διαφορετικές λύσεις.
  - Θέλουμε να βρούμε μία λύση με τη βέλτιστη (μέγιστη ή ελάχιστη) τιμή.
- Η μέθοδος λειτουργεί ως εξής:
  - Γνωρίζουμε ότι η **απλή αναδρομική λύση** δεν είναι αποδοτική γιατί επιλύει τα ίδια υποπροβλήματα κατ’ επανάληψη.
  - Φροντίζουμε **κάθε υποπρόβλημα να λυθεί μόνο μία φορά**, αποθηκεύοντας τη λύση του.
- Χρησιμοποιούμε **επιπλέον μνήμη για να εξοικονομήσουμε υπολογιστικό χρόνο**.
- Αλγόριθμοι εκθετικού χρόνου μετασχηματίζονται σε πολυωνυμικού χρόνου.

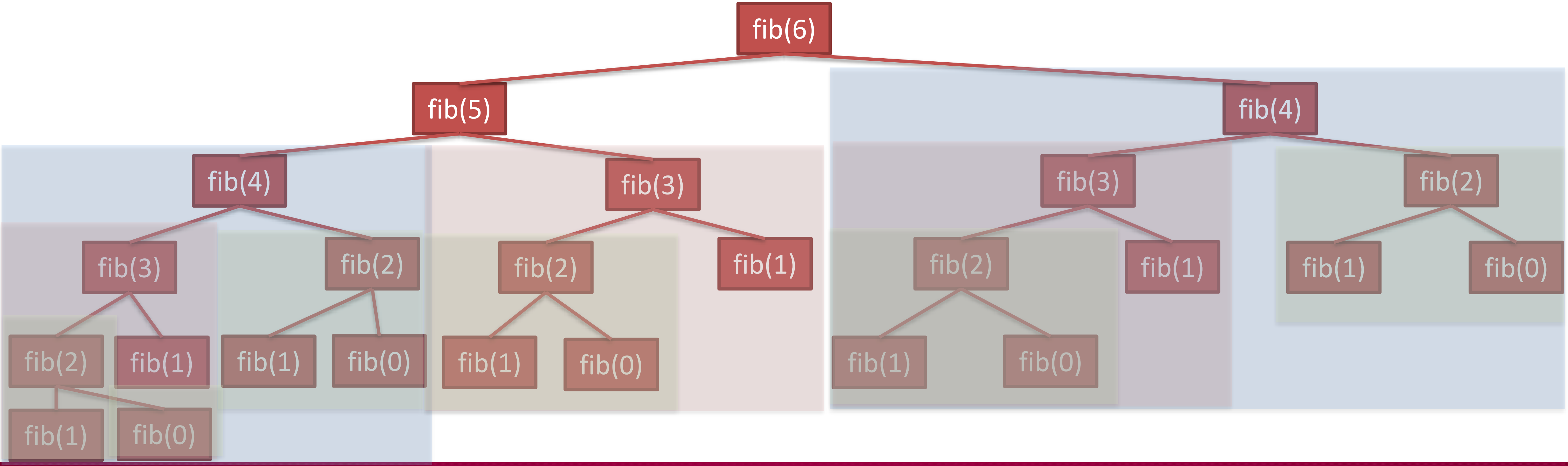
ΑΚΟΛΟΥΘΙΑ FIBONACCI

Υπολογισμός του n-οστού όρου της ακολουθίας Fibonacci:

$$f(n) = \begin{cases} 1, & n \leq 1 \\ f(n-1) + f(n-2), & n > 1 \end{cases}$$

ΥΠΟΛΟΓΙΣΜΟΣ ΑΚΟΛΟΥΘΙΑΣ FIBONACCI

```
int fib(int n) {
   if(n <= 1)  return 1;
   else  return fib(n – 1) + fib(n – 2);
}
```



ΠΟΛΥΠΛΟΚΟΤΗΤΑ ΥΠΟΛΟΓΙΣΜΟΥ FIBONACCI

$$T(n) = \begin{cases} 1, & \alpha\nu\ n \leq 1 \\ T(n-1) + T(n-2) + c, & \delta\iota\alpha\varphi o\rho\varepsilon\tau\iota\kappa\alpha \end{cases}$$

Προφανώς ισχύει
$T(n-1) \leq T(n)$

Έχουμε

$T(n) = T(n-1) + T(n-2) + c \leq 2T(n-1) + c$
$= 2(T(n-2) + T(n-3) + c) + c \leq 2(2T(n-2) + c) + c$
$= 4T(n-2) + 3c = 4(T(n-3) + T(n-4) + c) + 3c$
$\leq 4(2T(n-3) + c) + 3c = 8T(n-3) + 7c \leq 2^k T(n-k) + \left(2^k - 1\right)c$

Για $k = n$, έχουμε $T(n-k) = T(0)$, άρα

$$T(n) \leq 2^n T(0) + (2^n - 1)c = 2^n(1 + c) - c$$

ΠΟΛΥΠΛΟΚΟΤΗΤΑ ΥΠΟΛΟΓΙΣΜΟΥ FIBONACCI

$$T(n) = \begin{cases} 1, & \alpha\nu\ n \leq 1 \\ T(n-1) + T(n-2) + c, & \delta\iota\alpha\varphi o\rho\varepsilon\tau\iota\kappa\alpha \end{cases}$$

Εναλλακτικά, μπορούμε να επιλύσουμε την $T(n)$ ως γραμμική αυτόνομη εξίσωση διαφορών δεύτερης τάξης.

Έστω $t_n = t_{n-1} + t_{n-2} + c \ \Rightarrow\ t_n - t_{n-1} - t_{n-2} = c$ με $t_1 = t_0 = 1$ και με χαρακτηριστική εξίσωση $r^2 - r - 1 = 0$, η οποία έχει ρίζες $r_{1,2} = \frac{1 \pm \sqrt{5}}{2}$, οπότε η λύση της εξίσωσης διαφορών είναι:

$$t_n = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right] - c$$

Από Μαθηματική Ανάλυση

Αν $\Delta > 0$ (πραγματικές και διακεκριμένες ρίζες):
$y_t = C_1 r_1^t + C_2 r_2^t + \frac{b}{1+a_1+a_2}$

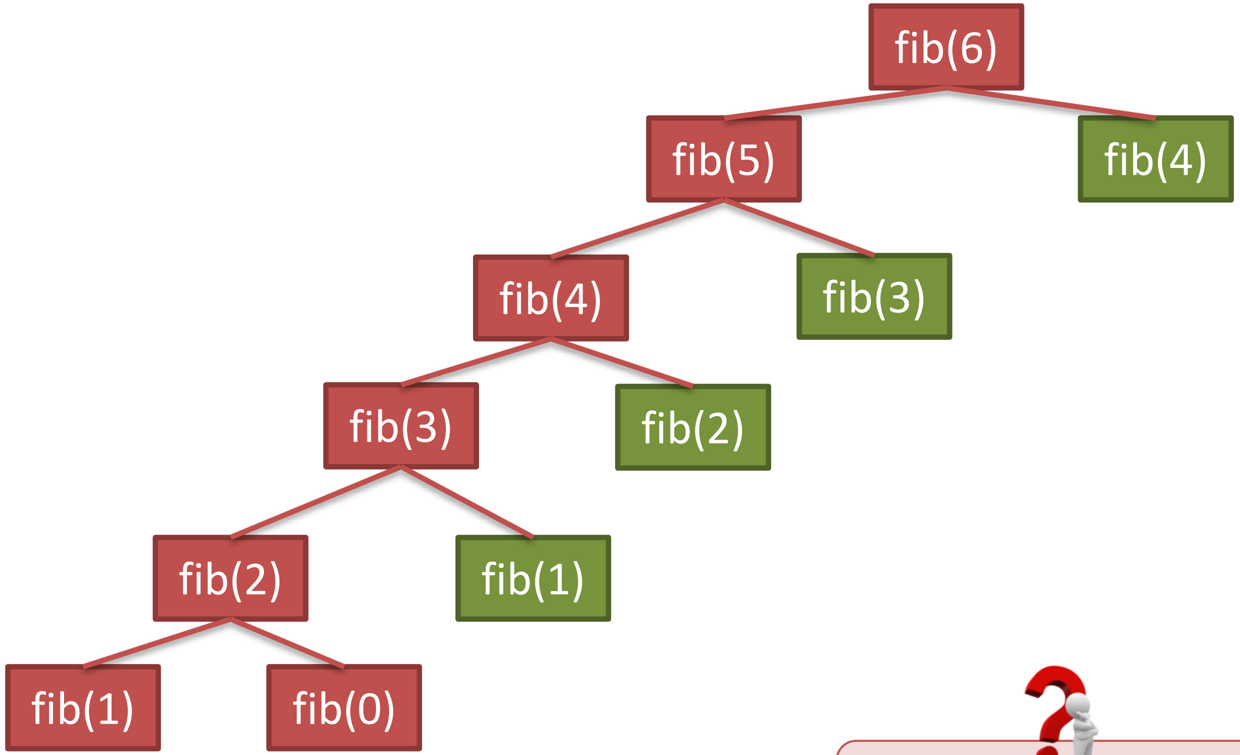
ΥΠΟΛΟΓΙΣΜΟΣ ΑΚΟΛΟΥΘΙΑΣ FIBONACCI

Καταβιβαστική με υπομνηματισμό (top-down)

- Γράφουμε τη διαδικασία αναδρομικά με το συνήθη τρόπο, αλλά τροποποιημένη ώστε να αποθηκεύεται το αποτέλεσμα του κάθε υποπροβλήματος.
- Συνεπώς, «θυμάται» προηγούμενες υπολογισμένες τιμές υπο-προβλημάτων.

ΚΩΔΙΚΑΣ C (ΚΑΤΑΒΙΒΑΣΤΙΚΗ ΜΕ ΥΠΟΜΝΗΜΑΤΙΣΜΟ)

```c
int fib(int n) {
   int t;
   if(knownF[n] != -1)
      return knownF[n];
   if(n <= 1)
      t = 1;
   else
      t = fib(n – 1) + fib(n – 2);
   return knownF[n] = t;
}
```



Πολυπλοκότητα

ΥΠΟΛΟΓΙΣΜΟΣ ΑΚΟΛΟΥΘΙΑΣ FIBONACCI

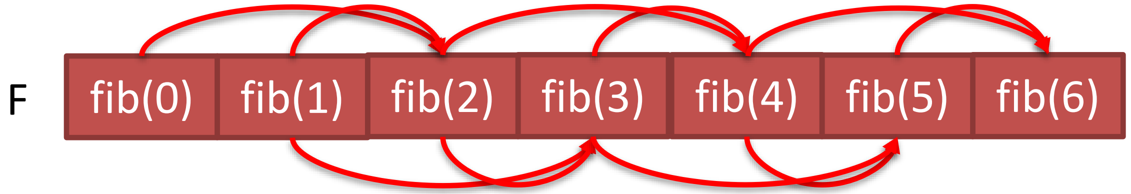
Αναβιβαστική με υπομνηματισμό (bottom-up)

– Ταξινομούμε τα προβλήματα κατά σειρά μεγέθους και έπειτα τα επιλύουμε.

– Έτσι, **όταν λύνουμε ένα συγκεκριμένο υποπρόβλημα, έχουμε ήδη λύσει όλα τα μικρότερα** στα οποία βασίζεται η λύση του και έχουμε αποθηκεύσει τις λύσεις τους.

ΚΩΔΙΚΑΣ C (ΑΝΑΒΙΒΑΣΤΙΚΗ ΜΕ ΥΠΟΜΝΗΜΑΤΙΣΜΟ)

```c
int fib(int n) {
    int i;
    F[0] = 1; F[1] = 1;
    for (i = 2; i <= n; i++)
        F[i] = F[i - 1] + F[i - 2];
    return F[n];
}
```



Πολυπλοκότητα

ΕΚΤΟΣ ΥΛΗΣ ΕΞΕΤΑΣΕΩΝ

ΔΥΝΑΜΙΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

- *Μεθοδολογία*
- *Πολλαπλασιασμός Αλληλουχίας Πινάκων*
- *Πρόβλημα του Σακιδίου*

ΜΕΘΟΔΟΛΟΓΙΑ

1. Χαρακτηρίζουμε τη **δομή** μίας βέλτιστης λύσης (εύρεση βέλτιστης υποδομής).
2. Ορίζουμε **αναδρομικά** την τιμή μίας βέλτιστης λύσης.
3. Υπολογίζουμε την τιμή μίας βέλτιστης λύσης εργαζόμενοι κατά κανόνα «**αναβιβαστικά**» (bottom-up).
4. Κατασκευάζουμε μία **βέλτιστη** λύση από τα δεδομένα που έχουμε υπολογίσει.

ΒΕΛΤΙΣΤΗ ΥΠΟΔΟΜΗ (OPTIMAL SUBSTRUCTURE)

- Θα λέμε ότι ένα πρόβλημα παρουσιάζει **βέλτιστη υποδομή**, εάν η βέλτιστη λύση περιέχει βέλτιστες λύσεις στα υπο-προβλήματά της.
- Αντίστοιχα, εάν η λύση κάποιου **υπο-προβλήματος δεν είναι βέλτιστη**, τότε και η λύση του αρχικού προβλήματος **δεν θα είναι βέλτιστη**.
- Εάν το προς επίλυση πρόβλημα δεν εμφανίζει βέλτιστη υποδομή, τότε η μέθοδος δυναμικού προγραμματισμού δεν είναι συνήθως η κατάλληλη μέθοδος επίλυσης.

ΠΟΛΛΑΠΛΑΣΙΑΣΜΟΣ ΑΛΛΗΛΟΥΧΙΑΣ ΠΙΝΑΚΩΝ (MATRIX CHAIN MULTIPLICATION)

Έστω μια ακολουθία n πινάκων $\langle A_1, A_2, \dots, A_n \rangle$, όπου για κάθε $i = 1,2, \dots n$ ο πίνακας A_i έχει διαστάσεις $p_{i-1} \times p_i$.
Θέλουμε να εκφράσουμε το γινόμενο $A_1 A_2 \cdots A_n$ σε πλήρως παρενθετική μορφή τέτοια ώστε να **ελαχιστοποιήσουμε** το πλήθος των βαθμωτών πολλαπλασιασμών.

Παράδειγμα

- Έστω ότι θέλουμε να εκφράσουμε το γινόμενο των πινάκων $A_1 A_2 A_3 A_4$ σε πλήρως παρενθετική μορφή:
  - $\Big(A_1\big(A_2(A_3A_4)\big)\Big)$, $\big((A_1A_2)(A_3A_4)\big)$, $\Big(\big((A_1A_2)A_3\big)A_4\Big)$, $\Big(A_1\big((A_2A_3)A_4\big)\Big)$, $\Big(\big(A_1(A_2A_3)\big)A_4\Big)$

ΙΔΙΟΤΗΤΕΣ ΠΟΛΛΑΠΛΑΣΙΑΣΜΟΥ

- Για να πολλαπλασιάσουμε δύο πίνακες A και B θα πρέπει ο αριθμός των στηλών του A να είναι ίσος με τον αριθμό των γραμμών του B.
- Εάν $A \in M_{p,q}$ και $B \in M_{q,r}$, τότε το γινόμενό τους $C \in M_{p,r}$

$$c_{i,j} = \sum_{k=1}^{q} a_{i,k} b_{k,j}$$

- Το πλήθος των βαθμωτών πολλαπλασιασμών που χρειάζονται είναι $p \cdot q \cdot r$.

ΠΑΡΑΔΕΙΓΜΑ

- Έστω ο πολλαπλασιασμός τριών πινάκων $\langle A_1, A_2, A_3 \rangle$, με
  - $A_1 \in M_{10,100}$, $A_2 \in M_{100,5}$, $A_3 \in M_{5,50}$.
- Υπάρχουν δύο ομαδοποιήσεις με διαφορετικό κόστος:

$\big((A_1 A_2) A_3\big)$

$B = A_1 A_2 \in M_{10,5}$
$10 \cdot 100 \cdot 5 = 5000$ πολ/σμοί

$C = B A_3 \in M_{100,50}$
$10 \cdot 5 \cdot 50 = 2500$ πολ/σμοί

Συνολικά 7500 βαθμωτοί πολ/σμοί

$\big(A_1 (A_2 A_3)\big)$

$B = A_2 A_3 \in M_{100,50}$
$100 \cdot 5 \cdot 50 = 25000$ πολ/σμοί

$C = A_1 B \in M_{10,50}$
$10 \cdot 100 \cdot 50 = 50000$ πολ/σμοί

Συνολικά 75000 βαθμωτοί πολ/σμοί

ΕΞΑΝΤΛΗΤΙΚΗ ΑΝΑΖΗΤΗΣΗ

- Υπολογίζουμε το πλήθος των δυνατών ομαδοποιήσεων $P(n)$ μιας αλληλουχίας n πινάκων.
- Για $n = 1$ υπάρχει μόνο ένας πίνακας, και για $n \geq 2$ έχουμε

$$P(n) = \begin{cases} 1, & \alpha\nu\ n = 1 \\ \displaystyle\sum_{k=1}^{n-1} P(k)P(n-k), & \delta\iota\alpha\varphi o\rho\varepsilon\tau\iota\kappa\alpha \end{cases}$$

- Αποδεικνύεται ότι η λύση της αναδρομικής σχέσης είναι $\Omega(2^n)$.

ΑΠΛΗΣΤΗ ΠΡΟΣΕΓΓΙΣΗ

- Επιλέγουμε σε κάθε βήμα τον πολλαπλασιασμό με το ελάχιστο κόστος.
- Έστω $\langle A_1, A_2, A_3, A_4 \rangle$, με $A_1 \in M_{50,20}$, $A_2 \in M_{20,1}$, $A_3 \in M_{1,10}$, $A_4 \in M_{10,100}$.
  - Μια άπληστη προσέγγιση θα εκτελούσε τους πολλαπλασιασμούς
    - A_2A_3: 200
    - $A_1(A_2A_3)$: 10200
    - $(A_1(A_2A_3))A_4$: 60200
  - Η βέλτιστη λύση είναι $(A_1A_2)(A_3A_4)$ με 7000 πολλαπλασιασμούς

Η άπληστη τεχνική θα αποτύγχανε ακόμα και εάν η επιλογή γινόταν βάσει του πολλαπλασιασμού με το μεγαλύτερο κόστος ή της απαλοιφής της μεγαλύτερης διάστασης.

1. ΕΥΡΕΣΗ ΒΕΛΤΙΣΤΗΣ ΥΠΟΔΟΜΗΣ

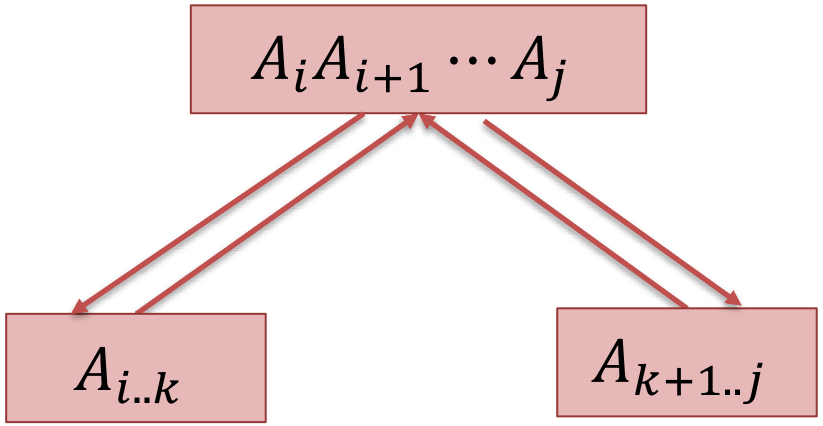
- Αρχικά βρίσκουμε τη **δομή μιας βέλτιστης παρενθετικής ομαδοποίησης** και στη συνέχεια **κατασκευάζουμε μια βέλτιστη λύση** από τις βέλτιστες λύσεις των υποπροβλημάτων.

- Συμβολίζουμε με $A_{i..j}$ τον πίνακα που προκύπτει από τον πολλαπλασιασμό των πινάκων $A_i A_{i+1} \cdots A_j$ (με $i \leq j$).

- Για $i < j$, μπορούμε να διαμερίσουμε το πρόβλημα στο σύνορο ανάμεσα στους πίνακες A_k και A_{k+1} (με $i \leq k < j$).
    - Έτσι υπολογίζουμε τα $A_{i..k}$ και $A_{k+1..j}$, και μετά το γινόμενό τους $A_{i..j}$.
    - Άρα το κόστος είναι ίσο με το κόστος υπολογισμού των $A_{i..k}$ και $A_{k+1..j}$ και του γινομένου τους.

1. ΕΥΡΕΣΗ ΒΕΛΤΙΣΤΗΣ ΥΠΟΔΟΜΗΣ

- Έστω ότι για να ομαδοποιήσουμε με βέλτιστο τρόπο την αλληλουχία $A_i A_{i+1} \cdots A_j$, διαχωρίζουμε στο σύνορο ανάμεσα στον πίνακα A_k και A_{k+1} (με $i \leq k < j$).
- Η ομαδοποίηση των $A_{i..k}$ και $A_{k+1..j}$ πρέπει να είναι **βέλτιστη**.
- Κατασκευάζουμε μια βέλτιστη λύση
  - Διαχωρίζουμε το πρόβλημα σε 2 υποπροβλήματα
  - Προσδιορίζουμε τις βέλτιστες λύσεις σε αυτά
  - Συνδυάζουμε τις επιμέρους λύσεις



2. ΜΙΑ ΑΝΑΔΡΟΜΙΚΗ ΛΥΣΗ

- Ορίζουμε το κόστος μιας βέλτιστης λύσης αναδρομικά.
- Επιλέγουμε ως υποπροβλήματα, τα προβλήματα $A_i A_{i+1} \cdots A_j$.
- Έστω $m[i,j]$ το ελάχιστο πλήθος των βαθμωτών πολλαπλασιασμών που απαιτούνται για τον υπολογισμό του πίνακα $A_{i..j}$.
  - Για $i = j$ το κόστος είναι $m[i,i] = 0$.
  - Για $i < j$ το κόστος είναι ίσο με το ελάχιστο κόστος υπολογισμού των υποδομών $A_{i..k}$ και $A_{k+1..j}$, καθώς και του γινομένου τους:

$$m[i,j] = m[i,k] + m[k+1,j] + p_{i-1} p_k p_j$$

2. ΜΙΑ ΑΝΑΔΡΟΜΙΚΗ ΛΥΣΗ

- Ο αναδρομικός τύπος για το ελάχιστο κόστος ομαδοποίησης του γινομένου $A_i A_{i+1} \cdots A_j$ είναι:

$$m[i,j] = \begin{cases} 0 & \varepsilon\alpha\nu\ i = j \\ \min\limits_{i \le k < j} \{m[i,k] + m[k+1,j] + p_{i-1}p_k p_j\} & \varepsilon\alpha\nu\ i < j \end{cases}$$

- Ορίζουμε $s[i,j]$ μια τιμή k στην οποία διαχωρίζεται το γινόμενο σε μια βέλτιστη ομαδοποίηση, δηλαδή $s[i,j] = k$ τέτοια ώστε

$$m[i,j] = m[i,k] + m[k+1,j] + p_{i-1}p_k p_j$$

2. ΜΙΑ ΑΝΑΔΡΟΜΙΚΗ ΛΥΣΗ

Δύο «βοηθητικοί» πίνακες

$$m = \begin{bmatrix} 0 & m_{1,2} & m_{1,3} & m_{1,4} & m_{1,5} \\ & 0 & m_{2,3} & \boxed{m_{2,4}} & m_{2,5} \\ & & 0 & m_{3,4} & m_{3,5} \\ & & & 0 & m_{4,5} \\ & & & & 0 \end{bmatrix} \qquad s= \begin{bmatrix} 0 & s_{1,2} & s_{1,3} & s_{1,4} & s_{1,5} \\ & 0 & s_{2,3} & \boxed{s_{2,4}} & s_{2,5} \\ & & 0 & s_{3,4} & s_{3,5} \\ & & & 0 & s_{4,5} \\ & & & & 0 \end{bmatrix}$$

Η τιμή του $m_{2,4}$ προσδιορίζει το ελάχιστο κόστος υπολογισμού του γινομένου $A_2A_3A_4$. Η τιμή του $s_{2,4}$ προσδιορίζει τη θέση στην οποία διαχωρίζεται το γινόμενο με το ελάχιστο κόστος υπολογισμού. Για παράδειγμα, εάν $s_{2,4} = 3$, τότε το ελάχιστο κόστος επιτυγχάνεται με το διαχωρισμό $(A_2A_3)A_4$.

3. ΥΠΟΛΟΓΙΣΜΟΣ ΕΛΑΧΙΣΤΟΥ ΚΟΣΤΟΥΣ

Αναβιβαστική προσέγγιση με χρήση πινάκων.

Είσοδος: Η ακολουθία $p = \langle p_0, p_1, \ldots, p_n \rangle$ με τις διαστάσεις των πινάκων

```
ΑΛΓΟΡΙΘΜΟΣ ΔΙΑΤΑΞΗ ΑΛΛΗΛΟΥΧΙΑΣ ΠΙΝΑΚΩΝ (p)
n=p.length-1
for i=1 to n
   m[i,i]=0
for l=2 to n
   for i=1 to n-l+1
      j=i+l-1
      m[i,j]=∞
      for k=i to j-1
         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
return m,s
```

Χρόνος εκτέλεσης

$\mathcal{O}(n^3)$

Χώρος αποθήκευσης (πίνακες m και s)

$\mathcal{O}(n^2)$

3. ΥΠΟΛΟΓΙΣΜΟΣ ΕΛΑΧΙΣΤΟΥ ΚΟΣΤΟΥΣ

```
ΑΛΓΟΡΙΘΜΟΣ ΔΙΑΤΑΞΗ ΑΛΛΗΛΟΥΧΙΑΣ ΠΙΝΑΚΩΝ (p)
n=p.length-1
for i=1 to n
   m[i,i]=0
for l=2 to n
   for i=1 to n-l+1
      j=i+l-1
      m[i,j]=∞
      for k=i to j-1
         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
return m,s
```

Υπολογισμός αλληλουχιών μοναδιαίου μήκους

Παράδειγμα

$A_1A_2A_3A_4A_5$

$p = [20,10,15,40,20,5]$

$$m = \begin{bmatrix} 0 & & & & \\ & 0 & & & \\ & & 0 & & \\ & & & 0 & \\ & & & & 0 \end{bmatrix}$$

3. ΥΠΟΛΟΓΙΣΜΟΣ ΕΛΑΧΙΣΤΟΥ ΚΟΣΤΟΥΣ

```
ΑΛΓΟΡΙΘΜΟΣ ΔΙΑΤΑΞΗ ΑΛΛΗΛΟΥΧΙΑΣ ΠΙΝΑΚΩΝ (p)
n=p.length-1
for i=1 to n
    m[i,i]=0
for l=2 to n
    for i=1 to n-l+1
        j=i+l-1
        m[i,j]=∞
        for k=i to j-1
            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
return m,s
```

1η επανάληψη (l=2)

Υπολογισμός κόστους για αλληλουχίες μήκους 2

Παράδειγμα

$A_1A_2A_3A_4A_5$

$p = [20,10,15,40,20,5]$

$$m = \begin{array}{c} \\ \\ \end{array}\begin{array}{ccccc} & A_1A_2 & A_2A_3 & A_3A_4 & A_4A_5 \end{array}$$

$$m = \begin{bmatrix} 0 & 3000 & & & \\ & 0 & 6000 & & \\ & & 0 & 12000 & \\ & & & 0 & 4000 \\ & & & & 0 \end{bmatrix}$$

$$s = \begin{bmatrix} 1 & & & \\ & 2 & & \\ & & 3 & \\ & & & 4 \end{bmatrix}$$

m[1,2]=m[1,1]+m[2,2]+20*10*15=3000

m[2,3]=m[2,2]+m[3,3]+10*15*40=6000

m[3,4]=m[3,3]+m[4,4]+15*40*20=12000

m[4,5]=m[4,4]+m[5,5]+40*20*5=4000

3. ΥΠΟΛΟΓΙΣΜΟΣ ΕΛΑΧΙΣΤΟΥ ΚΟΣΤΟΥΣ

```
ΑΛΓΟΡΙΘΜΟΣ ΔΙΑΤΑΞΗ ΑΛΛΗΛΟΥΧΙΑΣ ΠΙΝΑΚΩΝ (p)
n=p.length-1
for i=1 to n
    m[i,i]=0
for l=2 to n
    for i=1 to n-l+1
        j=i+l-1
        m[i,j]=∞
        for k=i to j-1
            q=m[i,k]+m[k+1,j]+p[i-1]*p[k]*p[j]
            if (q<m[i,j])
                m[i,j]=q

return m,s
```

2η επανάληψη (l=3)
Υπολογισμός κόστους για αλληλουχίες μήκους 3

$$m[1,3]=\min\begin{cases}\mathbf{m[1,1]+m[2,3]+20*10*40=14000}\\ m[1,2]+m[3,3]+20*15*40=15000\end{cases}$$

$$m[2,4]=\min\begin{cases}m[2,2]+m[3,4]+10*15*20=15000\\ \mathbf{m[2,3]+m[4,4]+10*40*20=14000}\end{cases}$$

$$m[3,5]=\min\begin{cases}\mathbf{m[3,3]+m[4,5]+15*40*5=7000}\\ m[3,4]+m[5,5]+15*20*5=13500\end{cases}$$

Παράδειγμα

$A_1A_2A_3A_4A_5$

$p=[20,10,15,40,20,5]$

$A_1(A_2A_3)$

$(A_2A_3)A_4$

$A_3(A_4A_5)$

$$m=\begin{bmatrix}0 & 3000 & 14000 & & \\ & 0 & 6000 & 14000 & \\ & & 0 & 12000 & 7000\\ & & & 0 & 4000\\ & & & & 0\end{bmatrix}$$

$$s=\begin{bmatrix}1 & 1 & & \\ & 2 & 3 & \\ & & 3 & 3\\ & & & 4\end{bmatrix}$$

3. ΥΠΟΛΟΓΙΣΜΟΣ ΕΛΑΧΙΣΤΟΥ ΚΟΣΤΟΥΣ

```
ΑΛΓΟΡΙΘΜΟΣ ΔΙΑΤΑΞΗ ΑΛΛΗΛΟΥΧΙΑΣ ΠΙΝΑΚΩΝ (p)
n=p.length-1
for i=1 to n
    m[i,i]=0
for l=2 to n
    for i=1 to n-l+1
        j=i+l-1
        m[i,j]=∞
        for k=i to j-1
            q=m[i,k]+m[k+1,j]+p[i-1]*p[k]*p[j]
            if (q<m[i,j])
                m[i,j]=q

return m,s
```

3η επανάληψη (l=4)

Υπολογισμός κόστους για αλληλουχίες μήκους 4

$$\text{m[1,4]=min}\begin{cases}\mathbf{m[1,1]+m[2,4]+20*10*20=18000} \\ \text{m[1,2]+m[3,4]+20*15*20=21000} \\ \text{m[1,3]+m[4,4]+20*40*20=30000}\end{cases}$$

$$\text{m[2,5]= min}\begin{cases}\mathbf{m[2,2]+m[3,5]+10*15*5=7750} \\ \text{m[2,3]+m[4,5]+10*40*5=12000} \\ \text{m[2,4]+m[5,5]+10*20*5=15000}\end{cases}$$

Παράδειγμα

$A_1A_2A_3A_4A_5$

$p = [20,10,15,40,20,5]$

$A_1(A_2A_3)A_4$

$A_2(A_3(A_4A_5))$

$$m = \begin{bmatrix} 0 & 3000 & 14000 & 18000 & \\ & 0 & 6000 & & 7750 \\ & & 0 & 12000 & 7000 \\ & & & 0 & 4000 \\ & & & & 0 \end{bmatrix}$$

$$s = \begin{bmatrix} 1 & 1 & 1 & \\ & 2 & 3 & 2 \\ & & 3 & 3 \\ & & & 4 \end{bmatrix}$$

3. ΥΠΟΛΟΓΙΣΜΟΣ ΕΛΑΧΙΣΤΟΥ ΚΟΣΤΟΥΣ

```
ΑΛΓΟΡΙΘΜΟΣ ΔΙΑΤΑΞΗ ΑΛΛΗΛΟΥΧΙΑΣ ΠΙΝΑΚΩΝ (p)
n=p.length-1
for i=1 to n
    m[i,i]=0
for l=2 to n
    for i=1 to n-l+1
        j=i+l-1
        m[i,j]=∞
        for k=i to j-1
            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
return m,s
```

4η επανάληψη (l=5)

Υπολογισμός κόστους για αλληλουχίες μήκους 5

Παράδειγμα

$A_1A_2A_3A_4A_5$

$p=[20,10,15,40,20,5]$

$A_1(A_2(A_3(A_4A_5)))$

$$m=\begin{bmatrix} 0 & 3000 & 14000 & 18000 & 8750 \\ & 0 & 6000 & 14000 & 7750 \\ & & 0 & 12000 & 7000 \\ & & & 0 & 4000 \\ & & & & 0 \end{bmatrix}$$

$$s=\begin{bmatrix} 1 & 1 & 1 & 1 \\ & 2 & 3 & 2 \\ & & 3 & 3 \\ & & & 4 \\ & & & \end{bmatrix}$$

$$m[1,5]=\min\begin{cases} \mathbf{m[1,1]+m[2,5]+20*10*5=8750} \\ m[1,2]+m[3,5]+20*15*5=11500 \\ m[1,3]+m[4,5]+20*40*5=22000 \\ m[1,4]+m[5,5]+20*20*5=20000 \end{cases}$$

4. ΚΑΤΑΣΚΕΥΗ ΜΙΑΣ ΒΕΛΤΙΣΤΗΣ ΛΥΣΗΣ

- Ο πίνακας s προσδιορίζει τον τρόπο πολλαπλασιασμού των πινάκων

$$s=\begin{bmatrix} 1 & 1 & 1 & 1 \\ & 2 & 3 & 2 \\ & & 3 & 3 \\ & & & 4 \\ & & & \end{bmatrix}$$

- Η τιμή του κάθε στοιχείου $s[i,j]$ αντιστοιχεί σε ένα σημείο διαχωρισμού k μεταξύ των πινάκων A_k και A_{k+1} για μια βέλτιστη ομαδοποίηση της αλληλουχίας $A_i A_{i+1} \cdots A_j$.

4. ΚΑΤΑΣΚΕΥΗ ΜΙΑΣ ΒΕΛΤΙΣΤΗΣ ΛΥΣΗΣ

- Η αναδρομική διαδικασία εκτυπώνει μια βέλτιστη ομαδοποίηση της αλληλουχίας λαμβάνοντας ως είσοδο τον πίνακα s και τις τιμές των i και j.

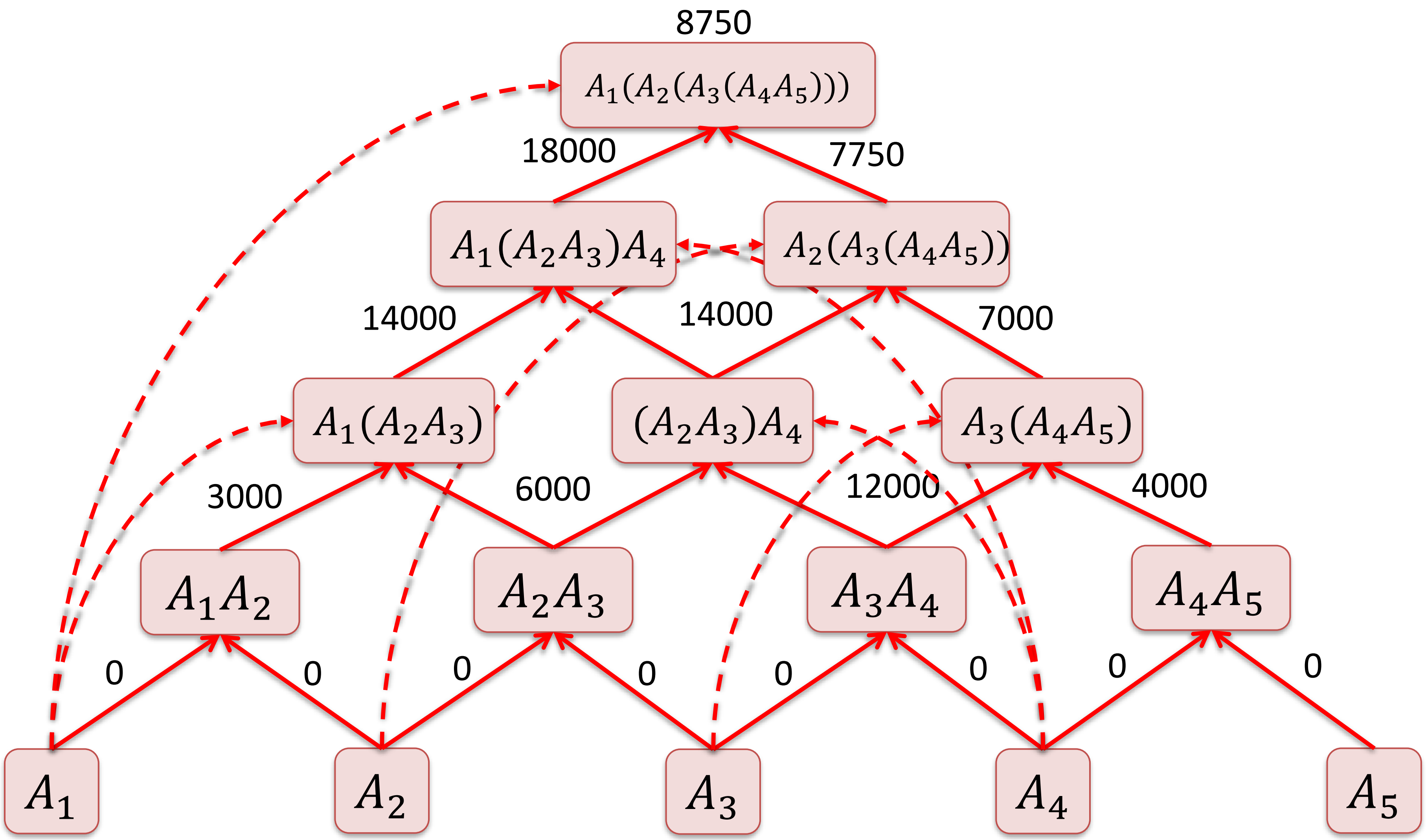
$$s = \begin{bmatrix} 1 & 1 & 1 & 1 \\ & 2 & 3 & 2 \\ & & 3 & 3 \\ & & & 4 \\ & & & \end{bmatrix}$$

```
ΑΛΓΟΡΙΘΜΟΣ ΕΚΤΥΠΩΣΗ ΒΕΛΤΙΣΤΗΣ ΟΜΑΔΟΠΟΙΗΣΗΣ (s,i,j)
if (i==j)
   print “A_”I
else
   print “(“
   ΕΚΤΥΠΩΣΗ ΒΕΛΤΙΣΤΗΣ ΟΜΑΔΟΠΟΙΗΣΗΣ (s,i,s[i,j])
   ΕΚΤΥΠΩΣΗ ΒΕΛΤΙΣΤΗΣ ΟΜΑΔΟΠΟΙΗΣΗΣ (s,s[i,j]+1,j)
   print “)”
```

Βέλτιστη Ομαδοποίηση

$A_1(A_2(A_3(A_4A_5)))$

ΣΧΗΜΑΤΙΚΗ ΑΝΑΠΑΡΑΣΤΑΣΗ



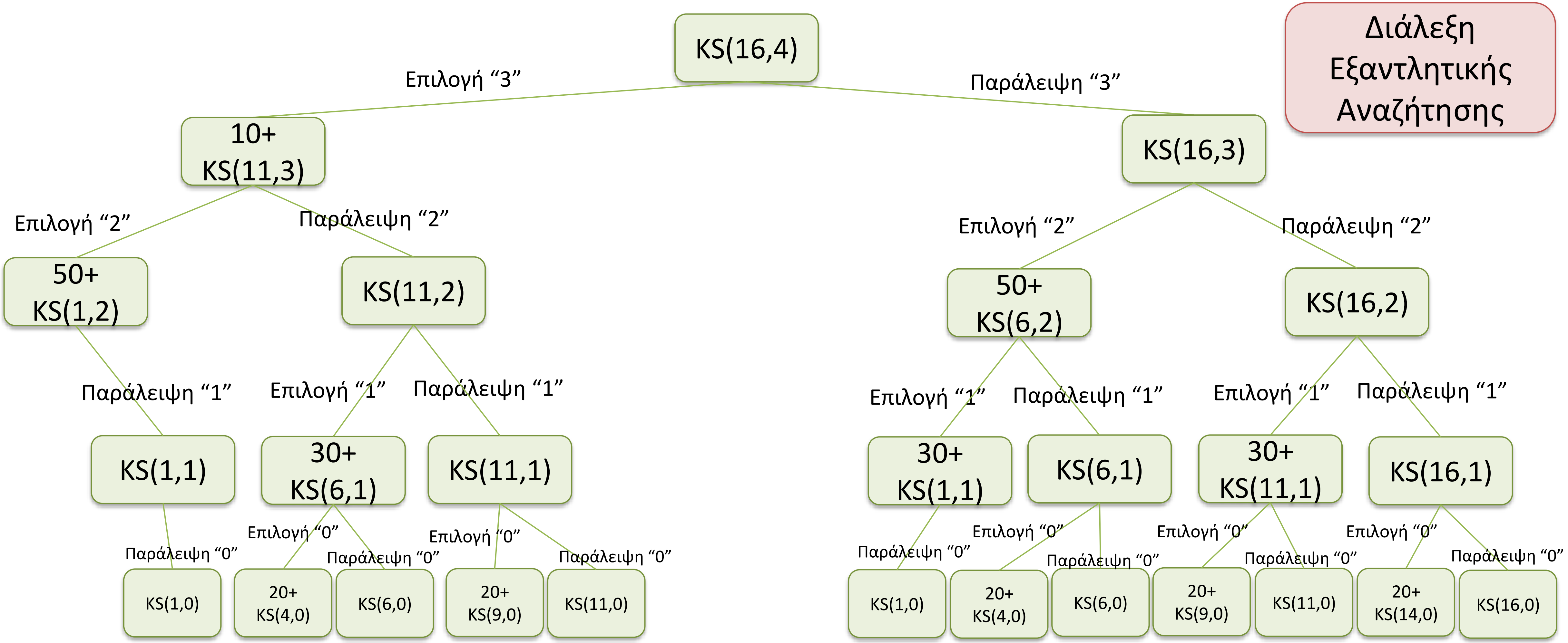
ΠΡΟΒΛΗΜΑ ΤΟΥ ΣΑΚΙΔΙΟΥ (KNAPSACK PROBLEM)

Δοσμένων ***n* αντικειμένων** με **βάρη** $w_1, w_2, \ldots, w_n$ και **αξίες** $v_1, v_2, \ldots, v_n$ και **ενός σάκου χωρητικότητας** W, να βρεθεί το πολυτιμότερο υποσύνολο των αντικειμένων που ταιριάζουν στο σάκο.

Χωρητικότητα *W*=16		
Είδος	Βάρος	Αξία (€)
1	2	20
2	5	30
3	10	50
4	5	10

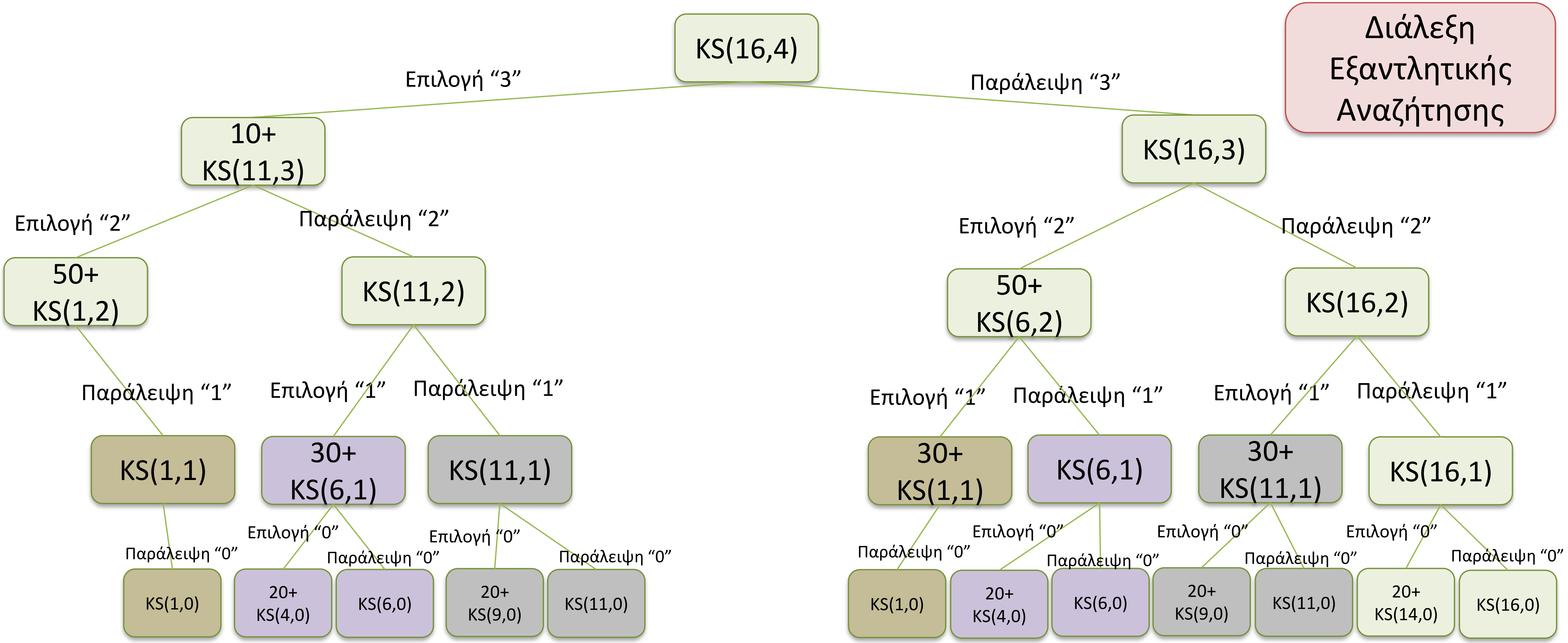
ΑΛΓΟΡΙΘΜΟΣ ΕΞΑΝΤΛΗΤΙΚΗΣ ΑΝΑΖΗΤΗΣΗΣ

Διάλεξη Εξαντλητικής Αναζήτησης



ΑΛΓΟΡΙΘΜΟΣ ΕΞΑΝΤΛΗΤΙΚΗΣ ΑΝΑΖΗΤΗΣΗΣ

Διάλεξη Εξαντλητικής Αναζήτησης



ΕΠΙΛΥΣΗ ΜΕ ΔΥΝΑΜΙΚΟ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ

- Θα πρέπει να κατασκευάσουμε μια αναδρομική σχέση που να εκφράζει μια λύση μιας έκφανσης του προβλήματος, σε σχέση με τις λύσεις μικρότερων εκφάνσεών του.
- Έστω $F[i, j]$ η τιμή μιας βέλτιστης λύσης για μια έκφανση του προβλήματος, που ορίζεται από **τα πρώτα $\boldsymbol{i}$ αντικείμενα** με βάρη $w_1, \ldots, w_i$ και αξίες $v_1, \ldots, v_i$, και ένας **σάκος με χωρητικότητα $\boldsymbol{j}$**.

ΕΠΙΛΥΣΗ ΜΕ ΔΥΝΑΜΙΚΟ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ

- Διαχωρίζουμε τα υποσύνολα των πρώτων i αντικειμένων σε δύο κατηγορίες
  - Αυτά που δεν συμπεριλαμβάνουν το i-οστό αντικείμενο
    - Η αξία ενός βέλτιστου υποσυνόλου είναι $F[i-1,j]$
  - Αυτά που συμπεριλαμβάνουν το i-οστό αντικείμενο
    - Η αξία ενός βέλτιστου υποσυνόλου είναι $v_i + F[i-1, j-w_i]$
- Μπορούμε να ορίσουμε την αναδρομή

$$F[i,j] = \begin{cases} \max\{F[i-1,j], v_i + F[i-1,j-w_i]\}, & \varepsilon\alpha\nu\ j - w_i \geq 0 \\ F[i-1,j], & \varepsilon\alpha\nu\ j - w_i < 0 \end{cases}$$

με αρχικές συνθήκες

$F[0,j] = 0, \forall\, j \geq 0$ και $F[i,0] = 0, \forall\, i \geq 0.$

ΕΦΑΡΜΟΓΗ

Πολυπλοκότητα

$$F[i,j] = \begin{cases} \max\{F[i-1,j], v_i + F[i-1,j-w_i]\}, & \varepsilon\alpha\nu\ j - w_i \ge 0 \\ F[i-1,j], & \varepsilon\alpha\nu\ j - w_i < 0 \end{cases}$$

Χωρητικότητα *W*=16		
Είδος	Βάρος	Αξία (€)
1	2	20
2	5	30
3	10	50
4	5	10

		Χωρητικότητα j																
Βάρος	Αξία	i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$w_1=2$	$v_1=20$	1	0	0	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
$w_2=5$	$v_2=30$	2	0	0	20	20	20	30	30	50	50	50	50	50	50	50	50	50	50
$w_3=10$	$v_3=50$	3	0	0	20	20	20	30	30	50	50	50	50	50	70	70	70	80	80
$w_4=5$	$v_4=10$	4	0	0	20	20	20	30	30	50	50	50	50	50	70	70	70	80	80

Κωνσταντίνος Γιαννουτάκης

Επίκ. Καθηγητής

kgiannou@uom.edu.gr

ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ