Δυναμικός Προγραμματισμός

Επιμέλεια διαφανειών: **Δ. Φωτάκης**

Τροποποιήσεις /προσθήκες: Α. Παγουρτζής

Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών

Εθνικό Μετσόβιο Πολυτεχνείο

Διωνυμικοί Συντελεστές

- Διωνυμικοί συντελεστές $C(n,k) = \binom{n}{k} = \frac{n!}{k!(n-k)!}$

$$C(n,k) = \begin{cases} C(n-1,k-1) + C(n-1,k) & \text{αν } 0 < k < n \\ 1 & \text{διαφορετικά} \end{cases}$$

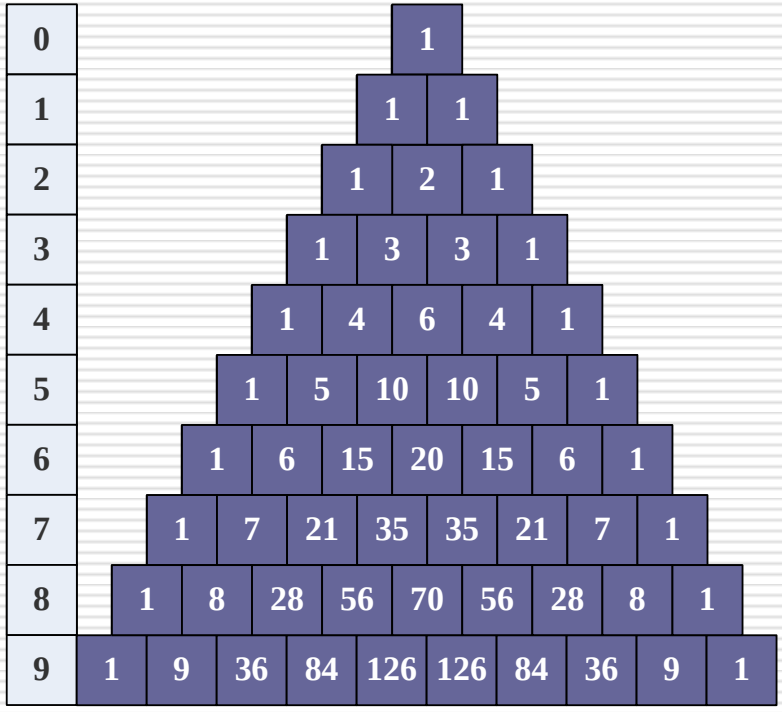
```
long Binom(int n, int k) {
  if ((k == 0) || (k == n)) return(1);
  return(Binom(n - 1, k - 1) + Binom(n - 1, k)); }
```

  - Χρόνος εκτέλεσης δίνεται από την ίδια αναδρομή!
    $T(n,k) = \Theta(C(n,k)) = \Omega((n/e)^k)\,,\ C(30,15) = 155117520$
  - Πρόβλημα οι επαναλαμβανόμενοι υπολογισμοί.
- Όταν έχω επικαλυπτόμενα στιγμιότυπα, χρησιμοποιώ **δυναμικό προγραμματισμό**.

Τρίγωνο του Pascal

$$C(n,k) = \begin{cases} C(n-1,k-1) + C(n-1,k) & \text{αν } 0 < k < n \\ 1 & \text{διαφορετικά} \end{cases}$$

- Όταν έχω επαναλαμβανόμενα στιγμιότυπα, αποθηκεύω τιμές σε πίνακα και τις χρησιμοποιώ χωρίς να τις υπολογίζω πάλι.
  - Θεαματική βελτίωση χρόνου εκτέλεσης!
  - Σημαντικές απαιτήσεις σε μνήμη.



Τρίγωνο του Pascal

$$C(n,k) = \begin{cases} C(n-1,k-1) + C(n-1,k) & \text{αν } 0 < k < n \\ 1 & \text{διαφορετικά} \end{cases}$$

Binomial(n, k)

$C[0,0] = C[1,0] = C[1,1] = 1;$
for $i \leftarrow 2$ **to** n **do**
  $C[i,0] \leftarrow 1;$
  for $j \leftarrow 1$ **to** $\min\{i-1, k\}$ **do**
    $C[i,j] \leftarrow C[i-1,j-1] + C[i-1,j];$
  if $i - 1 < k$ **then** $C[i,i] = 1;$
return$(C[n,k])$;

- Χρόνος εκτέλεσης $\Theta(nk)$ αντί για $\Omega((n/e)^k)$.
  - Μνήμη $\Theta(nk)$. Μπορεί να μειωθεί σε $\Theta(k)$.

Δυναμικός Προγραμματισμός

- Εφαρμόζουμε δυναμικό προγραμματισμό για προβλήματα συνδυαστικής βελτιστοποίησης όπου ισχύει:
  - **Αρχή βελτιστότητας** (βέλτιστες επιμέρους λύσεις).
    - Κάθε τμήμα βέλτιστης λύσης αποτελεί βέλτιστη λύση για αντίστοιχο υποπρόβλημα.
    - π.χ. κάθε τμήμα μιας συντομότερης διαδρομής είναι συντομότερη διαδρομή μεταξύ των άκρων του.
- Έστω βέλτιστες λύσεις για «μικρότερα» προβλήματα. Πως συνδυάζονται για βέλτιστη λύση σε «μεγαλύτερα»;
  - **Αναδρομική εξίσωση** που περιγράφει **τιμή** βέλτιστης λύσης.
  - Υπολογίζουμε λύση από μικρότερα σε μεγαλύτερα **(bottom-up).**

Διακριτό Πρόβλημα Σακιδίου (0-1 KNAPSACK)

- Δίνονται *n* αντικείμενα και **σακίδιο** μεγέθους *B*.
  Αντικείμενο *i* έχει **μέγεθος** και **αξία**: (s_i, p_i)
- Ζητείται συλλογή μέγιστης αξίας που χωράει στο σακίδιο.

$$\begin{array}{rl} \max & \sum_{i=1}^{n} f_i p_i \\ \text{υπό περιορισμούς} & \sum_{i=1}^{n} f_i s_i \leq B \\ & f_i \in \{0,1\} \quad \forall i \in [n] \end{array} \qquad f_i = \begin{cases} 1 & i \text{ εντός} \\ 0 & i \text{ εκτός} \end{cases}$$

  - Αντικείμενα: { (1, 0.5), (2, 5), (2, 5), (3, 9), (4, 8) }
    Μέγεθος σακιδίου: **4**.
  - Βέλτιστη λύση = { (2, 5), (2, 5) }
  - Αντικείμενα: { (3, 5), (2, 7), (4, 4), (6, 8), (5, 4) }
    Μέγεθος σακιδίου: **10**.
  - Βέλτιστη λύση = { (3, 5), (2, 7), (4, 4) }

Διακριτό Πρόβλημα Σακιδίου (0-1 KNAPSACK)

- Δίνονται *n* αντικείμενα και **σακίδιο** μεγέθους *B*.
  Αντικείμενο *i* έχει **μέγεθος** και **αξία**: (s_i, p_i)
- Ζητείται συλλογή μέγιστης αξίας που χωράει στο σακίδιο.

$$\begin{array}{rl} \max & \sum_{i=1}^{n} f_i p_i \\ \text{υπό περιορισμούς} & \sum_{i=1}^{n} f_i s_i \le B \\ & f_i \in \{0,1\} \quad \forall i \in [n] \end{array} \qquad f_i = \begin{cases} 1 & i \text{ εντός} \\ 0 & i \text{ εκτός} \end{cases}$$

- Λύνεται με την άπληστη στρατηγική;
  - Αντικείμενα: { (10, 100), (10, 100), (11, 111) }
    Μέγεθος σακιδίου: **20**.
  - Λύση απέχει πολύ από βέλτιστη.
- Μήπως με divide-and-conquer;

Διακριτό Πρόβλημα Σακιδίου (0-1 KNAPSACK)

- Πρόβλημα συνδυαστικής βελτιστοποίησης:
  - **Εφικτή λύση**: συλλογή που χωράει στο σακίδιο.
  - **Αξία λύσης**: άθροισμα αξιών στοιχείων συλλογής.
  - Ζητούμενο: (**βέλτιστη**) συλλογή που χωράει με μέγιστη αξία.
- Εξαντλητική αναζήτηση:
  - #συλλογών = 2^n . Χρόνος $\Omega(n2^n)$
- Πρόβλημα Σακιδίου είναι **NP**-δύσκολο και δεν υπάρχει «γρήγορος» (πολυωνυμικός) αλγόριθμος.
  - Εφαρμογή δυναμικού προγραμματισμού.
  - Χρόνος $\Theta(n\,B)$. **Δεν** είναι πολυωνυμικός(;)!

Αρχή Βελτιστότητας

- Αντικείμενα $N = \{1, \ldots, n\}$, σακίδιο μεγέθους B.
  Βέλτιστη λύση $A^* \subseteq \{1, \ldots, n\}$.
- Αγνοούμε αντικείμενο n :
  - $A^* \setminus \{n\}$ βέλτιστη λύση για $N \setminus \{n\}$ με σακίδιο $B - (f_n\, s_n)$.
- Αγνοούμε αντικείμενα $\{n, n-1\}$:
  - $A^* \setminus \{n, n-1\}$ βέλτιστη λύση για $N \setminus \{n, n-1\}$ με σακίδιο $B - (f_n\, s_n + f_{n-1}\, s_{n-1})$.
- **2 περιπτώσεις**: αντικ. n εντός ή εκτός βέλτιστης λύσης.
- Αν γνωρίζουμε βέλτιστες λύσεις για αντικείμενα $N \setminus \{n\}$ και σακίδια μεγέθους $B - s_n$ και B
  - … αποφασίζουμε αν αντικείμενο n στη βέλτιστη λύση για N

Αναδρομική Εξίσωση

- $P(n\text{-}1, B)$ βέλτιστη αξία για $N \setminus \{n\}$ σε σακίδιο B
  $P(n\text{-}1, B - s_n)$ βέλτιστη αξία για $N \setminus \{n\}$ σε σακίδιο $B - s_n$

$$P(n, B) = \max\{P(n-1, B), P(n-1, B - s_n) + p_n\}$$

- Προσέγγιση "Bottom-up": ορίζουμε $P(i, b)$: βέλτιστη αξία με αντικείμενα $\{1, \ldots, i\}$ και σακίδιο μεγέθους b

$$P(i,b) = \begin{cases} 0 & \text{αν } b \leq 0 \\ 0 & \text{αν } i = 0 \\ \max\{P(i-1,b), P(i-1,b-s_i) + p_i\} & \text{για } i = 1, \ldots, n \\ P(i-1,b) \quad \text{if } b < s_i & \text{και } b = 1, \ldots, B \end{cases}$$

Παράδειγμα

$$P(i,b) = \begin{cases} 0 & \text{αν } b \leq 0 \\ 0 & \text{αν } i = 0 \\ \max\{P(i-1,b), P(i-1,b-s_i)+p_i\} & \text{για } i = 1,\ldots,n \\ P(i-1,b) \quad \text{if } b < s_i & \text{και } b = 1,\ldots,B \end{cases}$$

- Αντικείμενα: { (3, 5), (2, 7), (4, 4), (6, 8), (5, 4) }
  Μέγεθος σακιδίου: **10**.

$i \mid b$	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	5	**5**	5	5	5	5	5	5
2	0	0	7	7	7	12	**12**	12	12	12	12
3	0	0	7	7	7	12	12	12	12	16	**16**
4	0	0	7	7	7	12	12	12	15	16	16
5	0	0	7	7	7	12	12	12	15	16	16

Υλοποίηση

Αναδρομική υλοποίηση;

```
Knapsack(B, (s_1, p_1), ..., (s_n, p_n))
    for i ← 0 to n do P[i, 0] ← 0;
    for b ← 1 to B do P[0, b] ← 0;
    for i ← 1 to n do
        for b ← 1 to B do
            if b − s_i ≥ 0 then
                t ← P[i − 1, b − s_i] + p_i;
            else t ← 0;
            if P[i − 1, b] ≥ t then
                P[i, b] ← P[i − 1, b];
            else P[i, b] ← t;
    return(P[n, B]);
```

Χρόνος $O(nB)$

Μνήμη $O(nB)$

Ψευδοπολυωνυμικοί Αλγόριθμοι

- Το πρόβλημα του σακιδίου είναι **NP**-δύσκολο.
- Αλγόριθμος $O(n\,B)$ δεν είναι πολυωνυμικού χρόνου(;)
  - Για να είναι, πρέπει πολυώνυμο του μεγέθους εισόδου!
  - Μέγεθος εισόδου: $\mathrm{O}(n(\log_2 B + \log_2 P_{\max}))$
  - Χρόνος πολυωνυμικός στο n αλλά εκθετικός στο $\log_2 B$
- Αριθμητικά προβλήματα:
  - Μέγεθος αριθμών πολύ μεγαλύτερο (π.χ. εκθετικό) από πλήθος «βασικών συνιστωσών» (ό,τι συμβολίζουμε με n).
- Αλγόριθμος πολυωνυμικού χρόνου:

$$\mathrm{O}((n\ \log \mathrm{max\_num})^k),\ \text{σταθερά}\ k \geq 1$$

- Αλγόριθμος **ψευδο-πολυωνυμικού** χρόνου:

$$\mathrm{O}((n\,\mathrm{max\_num})^k),\ \text{σταθερά}\ k \geq 1$$

Βελτιώσεις ΔΠ για 0-1 KNAPSACK (I)

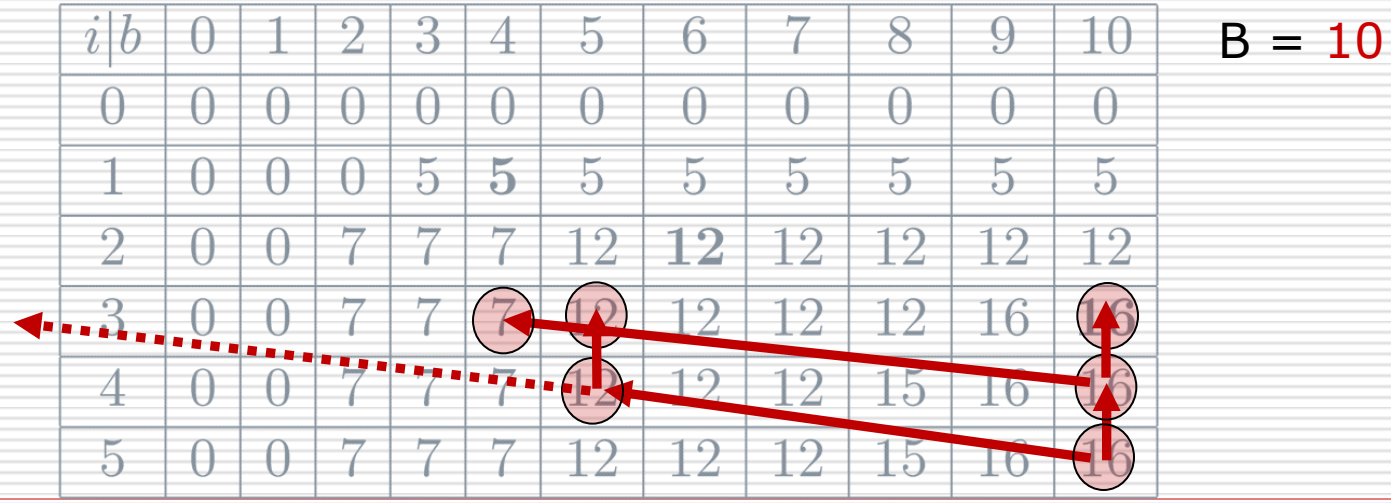
- Ζητούμενο: μείωση #υπολογιζόμενων θέσεων
- Αναδρομή: δεν χρειάζεται όλον τον πίνακα!

s_i, p_i	$i \mid b$	0	1	2	3	4	5	6	7	8	9	10
	0	0	0	0	0	0	0	0	0	0	0	0
3, 5	1	0	0	0	5	**5**	5	5	5	5	5	5
2, 7	2	0	0	7	7	7	12	**12**	12	12	12	12
4, 4	3	0	0	7	7	7	12	12	12	12	16	**16**
6, 8	4	0	0	7	7	7	12	12	12	15	16	16
5, 4	5	0	0	7	7	7	12	12	12	15	16	16

B = 10

Βελτιώσεις ΔΠ για 0-1 KNAPSACK (I)

- Ζητούμενο: μείωση #υπολογιζόμενων θέσεων
- Αναδρομή: δεν χρειάζεται όλον τον πίνακα!
- Χρήση αναδρομής με απομνημόνευση (memoization)



$i \mid b$	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	5	**5**	5	5	5	5	5	5
2	0	0	7	7	7	12	**12**	12	12	12	12
3	0	0	7	7	7	12	12	12	12	16	16
4	0	0	7	7	7	12	12	12	15	16	16
5	0	0	7	7	7	12	12	12	15	16	16

Βελτιώσεις ΔΠ για 0-1 KNAPSACK (I)

- Ζητούμενο: μείωση #υπολογιζόμενων θέσεων
- Αναδρομή: δεν χρειάζεται όλον τον πίνακα!
- Χρήση αναδρομής με απομνημόνευση (memoization)
  - θέσεις πίνακα υπολογίζονται μία φορά μόνο
- πολυπλ/τα: $O(\min(nB, 2^n))$, πρακτικά καλύτερος για $n << B$

$i \mid b$	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	5	**5**	5	5	5	5	5	5
2	0	0	7	7	7	12	12	12	12	12	12
3	0	0	7	7	7	12	12	12	12	16	16
4	0	0	7	7	7	12	12	12	15	16	16
5	0	0	7	7	7	12	12	12	15	16	16

Παράδειγμα ΔΠ-memoized

Δυναμικός Προγραμματισμός

```
15 25   0:    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
14 24   1:    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  25  25  25  25  25  25  25  25  25  25  25
16 70   2:    0   0   0   0   0   0   0   0   0   0   0   0   0   0  24  25  25  25  25  25  25  25  25  25  25  25
19 28   3:    0   0   0   0   0   0   0   0   0   0   0   0   0   0  24  25  70  70  70  70  70  70  70  70  70  70
 6 75   4:    0   0   0   0   0   0   0   0   0   0   0   0   0   0  24  25  70  70  70  70  70  70  70  70  70  70
 6 31   5:    0   0   0   0   0   0  75  75  75  75  75  75  75  75  75  75  75  75  75  75  75  99 100 145 145 145 145
12 11   6:    0   0   0   0   0   0  75  75  75  75  75  75 106 106 106 106 106 106 106 106 106 106 106 145 145 145 145
 2 85   7:    0   0   0   0   0   0  75  75  75  75  75  75 106 106 106 106 106 106 106 106 106 106 106 145 145 145 145
16 41   8:    0   0  85  85  85  85  85  85 160 160 160 160 160 160 191 191 191 191 191 191 191 191 191 191 230 230
 5 73   9:    0   0  85  85  85  85  85  85 160 160 160 160 160 160 191 191 191 191 191 191 191 191 191 191 230 230
       10:    0   0  85  85  85  85  85 158 160 160 160 160 160 233 233 233 233 233 233 264 264 264 264 264 264 264

Max Profit: 264
          0:    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
15 25     1:    0   0   0   0   0   0   0   0   0   0  -1   0   0   0   0  -1  -1  25  25  25  25  -1  -1  25  -1  25
14 24     2:    0   0   0   0   0   0   0   0   0   0  -1   0   0   0  24  -1  -1  25  25  25  25  -1  -1  25  -1  25
16 70     3:    0   0   0   0   0   0   0   0   0   0  -1   0   0   0  24  -1  -1  70  70  70  70  -1  -1  70  -1  70
19 28     4:    0   0   0   0   0   0   0   0   0   0  -1   0   0   0  24  -1  -1  70  70  70  70  -1  -1  70  -1  70
 6 75     5:    0   0   0   0   0   0  75  75  75  75  -1  75  75  75  75  -1  -1  75  75  75  99  -1  -1 145  -1 145
 6 31     6:    0  -1   0  -1   0  -1  75  75  75  75  -1  75  -1 106  -1  -1  -1  -1 106  -1 106  -1  -1 145  -1 145
12 11     7:    0  -1   0  -1   0  -1  -1  75  -1  75  -1  -1  -1  -1  -1  -1  -1  -1 106  -1 106  -1  -1 145  -1 145
 2 85     8:    0  -1  -1  -1  85  -1  -1  -1  -1 160  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1 191  -1  -1  -1  -1 230
16 41     9:    0  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1 191  -1  -1  -1  -1 230
 5 73    10:    0  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1 264

#clocks elapsed iterative: 6.000000
#clocks elapsed recursive: 2.000000
```

Βελτιώσεις ΔΠ για 0-1 KNAPSACK (II)

- 2ος τρόπος: αλγόριθμος Nemhauser-Ullmann
- Pareto-optimal θέσεις: (0,0), (1,3), (2,2), (2,5), (3,9), (4,8)
  - αντ/χούν σε «χρήσιμους» συνδ/σμούς αντικ/νων από [1.. *i*]
  - ... που δεν «κυριαρχούνται» από άλλους από αντικ/να [1.. *i*]
- #P.o.θ. σε γραμμή *i* ≤ Σ#P.o.θ. σε γραμμές [0.. *i*-1]

B = 10

s_i, p_i	$i \mid b$	0	1	2	3	4	5	6	7	8	9	10
	0	0	0	0	0	0	0	0	0	0	0	0
3, 5	1	0	0	0	5	**5**	5	5	5	5	5	5
2, 7	2	0	0	7	7	7	12	**12**	12	12	12	12
4, 4	3	0	0	7	7	7	12	12	12	12	16	**16**
6, 8	4	0	0	7	7	7	12	12	12	15	16	16
5, 4	5	0	0	7	7	7	12	12	12	15	16	16

Βελτιώσεις ΔΠ για 0-1 KNAPSACK (II)

- Ρ.ο.θ. αντιστοιχούν σε αθροίσματα υποσυνόλων
- Ρ.ο.θ. γραμμής *i*: «άθροιση» αντ. *i* σε Ρ.ο.θ. προηγ. γραμμής
- απόρριψη θέσεων με ≤ αξία, ≥ μέγεθος από άλλες
- πολυπλ/τα: $O(\min(nB, 2^n))$, *πολυων/κός* αναμεν. χρόνος!
- υλοποίηση με λίστες ζευγών (s_i, p_i) για κάθε γραμμή *i*

s_i p_i	$i \mid b$	0	1	2	3	4	5	6	7	8	9	10
	0	0	0	0	0	0	0	0	0	0	0	0
3, 5	1	0	0	0	5	**5**	5	5	5	5	5	5
2, 7	2	0	0	7	7	7	12	**12**	12	12	12	12
4, 4	3	0	0	7	7	7	12	12	12	12	16	**16**
6, 8	4	0	0	7	7	7	12	12	12	15	16	16
5, 4	5	0	0	7	7	7	12	12	12	15	16	16

B = 10

Απληστία vs Δυναμικός Προγρ.

- Διακριτό Πρόβλ. Σακιδίου: **όχι** ιδιότητα άπληστης επιλογής.
  - Π.χ. Αντικείμενα: {(1, 1+ε), (B, B)}. Σακίδιο μεγέθους B.
- **Απληστία** και **Δυναμικός Προγραμματισμός**:
  - Αρχή βελτιστότητας υπο-λύσεων κοινό χαρακτηριστικό.
- **Δυναμικός Προγραμματισμός**: αναδρομή
  - Χρειάζεται βέλτιστη λύση σε **πολλά** υπο-προβλήματα που εμπλέκονται στην αναδρομή.
    - Διακριτό Σακίδιο: υπολ/σμός βέλτιστης λύσης για πρώτα *i* αντικείμενα για *κάθε i* και **όλα** τα δυνατά μεγέθη σακιδίου!
  - Συνδυάζει «κατάλληλα» επιμέρους λύσεις για βέλτιστη. Προσέγγιση "bottom-up" (συνήθως).
  - Λύση **πολλών** υπο-προβλημάτων εγγυάται βέλτιστη λύση αλλά κοστίζει σημαντικά σε υπολογιστικό χρόνο.

Απληστία vs Δυναμικός Προγρ.

- **Απληστία:** επανάληψη
  - Ταξινόμηση ως προς κάποιο (εύλογο) κριτήριο.
  - Σε κάθε βήμα **αμετάκλητη** άπληστη επιλογή.
  - Άπληστη επιλογή: η καλύτερη με βάση τρέχουσα κατάσταση και κάποιο (απλό) κριτήριο.
  - Λύση **λίγων** (συχνά **ενός**) «**αναγκαίων**» υπο-προβλημάτων: αποδοτικό υπολογιστικά αλλά δεν δίνει πάντα βέλτιστη λύση.
  - Γρήγοροι, απλοί και «φυσιολογικοί» αλγόριθμοι!
  - (Καλές) προσεγγιστικές λύσεις σε πολλά προβλήματα.
  - Βέλτιστη λύση μόνο όταν ισχύει ιδιότητα άπληστης επιλογής (ως προς συγκεκριμένο κριτήριο επιλογής).

Subset Sum και Partition

- Πρόβλημα Subset Sum (Άθροισμα Υποσυνόλου):
  - Σύνολο φυσικών $A = \{s_1, \ldots, s_n\}$ και B, $0 < B < s(A)$.
  - Υπάρχει $X \subseteq A$ με $s(X) = \sum_{i \in X} s_i = B$;
  - γενίκευση Subset Sum .
  - Πρόβλημα Partition (Διαμέριση): όταν $B = s(A) / 2$
- $S(i, b)$ είναι TRUE ανν υπάρχει $X \subseteq \{1, \ldots, i\}$ με $s(X) = b$.

$$S(i,b) = \begin{cases} 1 & \text{αν } b = 0 \\ 0 & \text{αν } b < 0 \\ 0 & \text{αν } i = 0 \text{ και } b > 0 \\ S(i-1,b) \vee S(i-1,b-s_i) & \text{για } i = 1,\ldots,n \\ & \text{και } b = 1,\ldots,B \end{cases}$$

  - Η τιμή του $S(n, B)$ δίνει την απάντηση. Αποδοτικότητα;

Το Πρόβλημα του Περιπτερά

- Κέρματα αξίας 1, 12, και 20 λεπτών.
- Ρέστα ποσό *x* με **ελάχιστο** #κερμάτων.
  - Δυναμικός προγραμματισμός.
  - Πώς; Βρείτε την αναδρομή!
  - Πώς μπορεί να φτιαχτεί ο Πίνακας ΔΠ;

Αλυσιδωτός Πολ/μός Πινάκων

- Γινόμενο πινάκων Α ($p \times q$) επί Β ($q \times r$) σε χρόνο $\Theta(p\,q\,r)$
  ($p\,q\,r$ = πλήθος πολλαπλ/σμών)
- Ποιος ο συντομότερος τρόπος υπολογισμού γινομένου:

$$\underset{(d_0 \times d_1)}{A_1}\ \underset{(d_1 \times d_2)}{A_2}\ \underset{(d_2 \times d_3)}{A_3}\ \ldots\ \underset{(d_{n-2} \times d_{n-1})}{A_{n-1}}\ \underset{(d_{n-1} \times d_n)}{A_n}$$

  - Ο πολλαπλασιασμός πινάκων είναι πράξη προσεταιριστική (αποτέλεσμα ανεξάρτητο από υπολ/μό επιμέρους γινομένων)
  - Ο χρόνος υπολογισμού εξαρτάται από τη σειρά!

$$\underset{(1 \times 100)}{A_1}\ \underset{(100 \times 3)}{A_2}\ \underset{(3 \times 1)}{A_3}$$

$$\underset{(1 \times 100 \times 3) + (1 \times 3 \times 1) = 303}{(A_1 A_2)}\ A_3$$

$$A_1\ \underset{(1 \times 100 \times 1) + (100 \times 3 \times 1) = 400}{(A_2 A_3)}$$

Πολλαπλασιασμός Πινάκων

$$\underset{(13\times 5)}{A_1}\quad \underset{(5\times 89)}{A_2}\quad \underset{(89\times 3)}{A_3}\quad \underset{(3\times 34)}{A_4}$$

Σειρά Υπολογισμού	**Αριθμός Πολλαπλασιασμών**
$(((A_1A_2)A_3)A_4)$	$13\times 5\times 89+13\times 89\times 3+13\times 3\times 34=10582$
$((A_1A_2)(A_3A_4))$	54201
$((A_1(A_2A_3))A_4)$	2856
$(A_1((A_2A_3)A_4))$	4055
$(A_1(A_2(A_3A_4)))$	26418

Πολλαπλασιασμός Πινάκων

- Δίνονται *n* πίνακες:

$$\underset{(d_0 \times d_1)}{A_1}\ \underset{(d_1 \times d_2)}{A_2}\ \underset{(d_2 \times d_3)}{A_3}\ \ldots\ \underset{(d_{n-2} \times d_{n-1})}{A_{n-1}}\ \underset{(d_{n-1} \times d_n)}{A_n}$$

  Με ποια σειρά θα υπολογιστεί το γινόμενο $A_1 A_2 \ldots A_n$ ώστε να ελαχιστοποιηθεί #πολ/σμών;

- Πρόβλημα συνδυαστικής βελτιστοποίησης:
  - Κάθε σειρά υπολογισμού υπολογίζει γινόμενο πινάκων εκτελώντας διαφορετικό #πολ/σμών.
  - Ζητείται η σειρά με ελάχιστο #πολ/σμών.
- Υπάρχει αποδοτικός αλγόριθμος για υπολογισμό καλύτερης σειράς;

Εξαντλητική Αναζήτηση

- … δοκιμάζει όλες τις σειρές υπολογισμού και βρίσκει καλύτερη.
  - Κάθε σειρά αντιστοιχεί σε δυαδικό δέντρο με n φύλλα.
  - Χρόνος ανάλογος #δυαδικών δέντρων με n φύλλα:

$$P(n) = \sum_{k=1}^{n-1} P(k)P(n-k), \; P(1) = 1$$

  - Λύση (n-1)-οστός αριθμός Catalan:

$$P(n) = \frac{1}{n}\binom{2(n-1)}{n-1} = \Omega\left(\frac{4^n}{n^{3/2}}\right)$$

- Θα εφαρμόσουμε δυναμικό προγραμματισμό.

Αρχή Βελτιστότητας

- Συμβολίζουμε $A_{i..j} = A_i \times \cdots \times A_j$
- Βέλτιστη λύση υπολογίζει $A_{1..i}$, $(d_0 \times d_i)$, και $A_{i+1..n}$, $(d_i \times d_n)$, για κάποιο $i, 1 \leq i < n,$ και τελειώνει με $A_{1..i} \times A_{i+1..n}$.
  - #πολ/μών = $d_0 \times d_i \times d_n$ + #πολ/μών($A_{1..i}$) + #πολ/μων($A_{i+1..n}$)
  - Επιμέρους γινόμενα $A_{1..i}$ και $A_{i+1..n}$ υπολογίζονται **βέλτιστα**.
- Συμβολίζουμε $m[i,j] = \text{βέλτιστος } \#\text{πολ/μών}(A_{i..j})$
- Έστω για κάθε $i, 1 \leq i < n,$ γνωρίζουμε $m[1,i]$ και $m[i+1,n]$
- Τότε $m[1,n] = \min_{1 \leq i < n}\{m[1,i] + m[i+1,n] + d_0 d_i d_n\}$
- Γενική **αναδρομική σχέση**:

$$m[i,j] = \begin{cases} \min_{i \leq k < j}\{m[i,k] + m[k+1,j] + d_{i-1}\, d_k\, d_j\} & \text{αν } i < j \\ 0 & \text{αν } i = j \end{cases}$$

Δυναμικός Προγραμματισμός

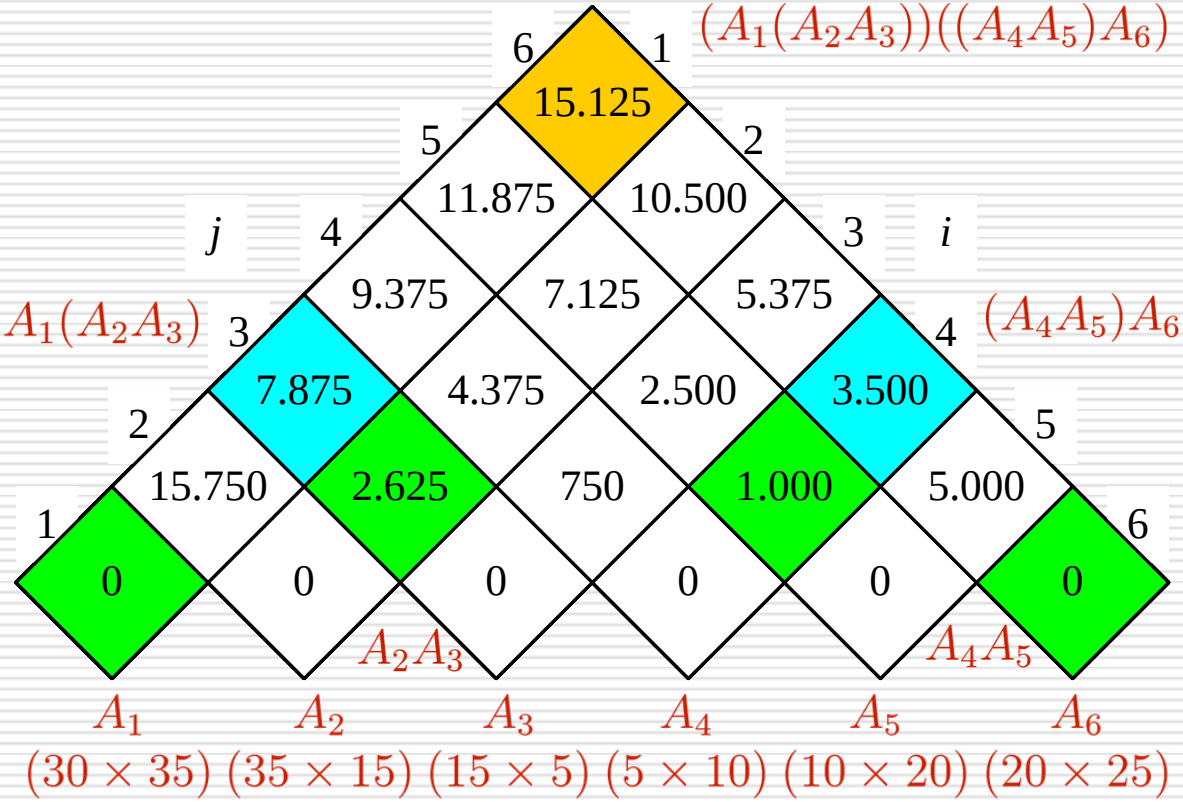
- Bottom-up υπολογισμός $m[1, n]$ από αναδρομική σχέση:

$$m[i,j] = \begin{cases} \min_{i \le k < j} \{ m[i,k] + m[k+1,j] + d_{i-1}\, d_k\, d_j \} & \text{αν } i < j \\ 0 & \text{αν } i = j \end{cases}$$

- Υπολογίζω $n(n-1)/2$ τιμές $m[i, j]$.
  - Κάθε $m[i, j]$ υπολογίζεται σε χρόνο $O(n)$ από τιμές γινομένων μικρότερου εύρους.
  - Τιμές $m[i, j]$ αποθηκεύονται σε πίνακα.

$$m[i,j] = \begin{cases} \min_{i \le k < j} \{m[i,k] + m[k+1,j] + d_{i-1}\, d_k\, d_j\} & \text{αν } i < j \\ 0 & \text{αν } i = j \end{cases}$$

Παράδειγμα



$$m[i,j] = \begin{cases} \min_{i \le k < j} \{m[i,k] + m[k+1,j] + d_{i-1}\, d_k\, d_j\} & \text{αν } i < j \\ 0 & \text{αν } i = j \end{cases}$$

Υλοποίηση με επανάληψη (bottom-up)

```
MatrixChainMultiplication(d[0, 1, ..., n])   /* A_i διάστασης d[i − 1] × d[i] */
    for i ← 1 to n do
        m[i, i] ← 0;
    for p ← 2 to n do
        for i ← 1 to n − p + 1 do
            j ← i + p − 1; m[i, j] ← ∞;
            for k ← i to j − 1 do
                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;
    return(m[1, n]);
```

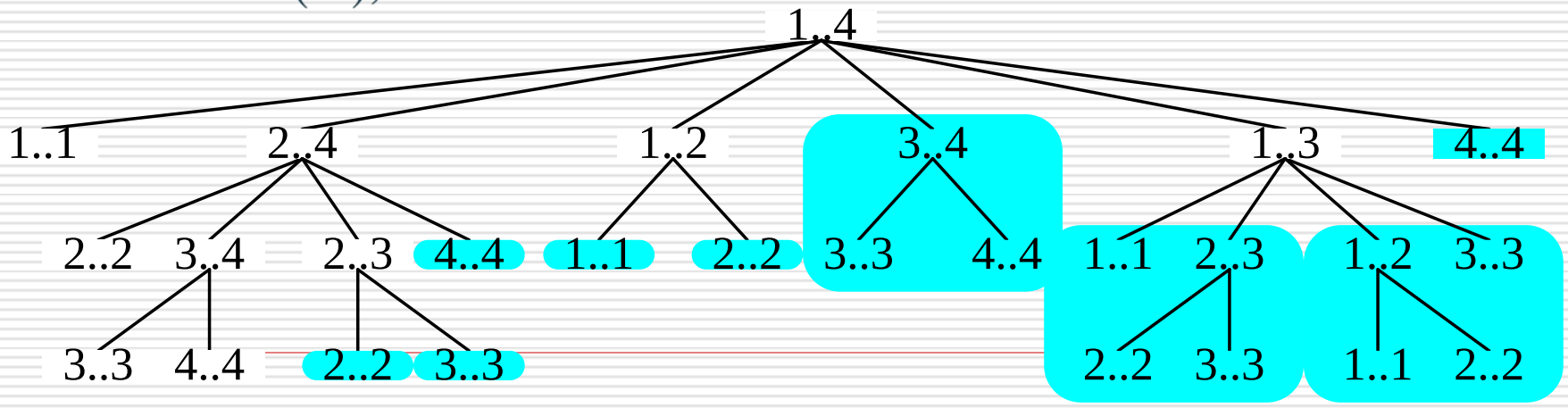
- Χρόνος $O(n^3)$ και μνήμη $O(n^2)$, μειώνεται σε $O(n)$.

$$m[i,j] = \begin{cases} \min_{i \le k < j} \{ m[i,k] + m[k+1,j] + d_{i-1}\, d_k\, d_j \} & \text{αν } i < j \\ 0 & \text{αν } i = j \end{cases}$$

Υλοποίηση με αναδρομή (top-down)

```
RecMatrixChain(d[i − 1, . . . , j])
    if i = j then return(0);
    m ← ∞;    /* Το m θα πάρει την τιμή m[i, j] */
    for k ← i to j − 1 do
        q ← RecMatrixChain(d[i − 1, . . . , k]) +
            RecMatrixChain(d[k, . . . , j]) + d[i − 1]d[k]d[j];
        if q < m then m ← q;
    return(m);
```

Χρόνος **Ω(2ⁿ)** !



Αναδρομή με Απομνημόνευση

- Memoization: ο αναδρομικός αλγόριθμος αποθηκεύει τιμές σε πίνακα. Κάθε τιμή υπολογίζεται μία φορά μόνο.
  - Συνδυάζει απλότητα top-down προσέγγισης με ταχύτητα bottom-up.

```
RecMemMatrixChain(d[0, ..., n])
    for i ← 1 to n do
        for j ← 1 to n do
            m[i, j] ← ∞;
    return(RecCM(d[0, ..., n]));
```

```
RecCM(d[i − 1, ..., j]);
    if m[i, j] < ∞ then return(m[i, j]);
    if i = j then m[i, j] = 0;
    else
        for k ← i to j − 1 do
            q ← RecCM(d[i − 1, ..., k]) +
                RecCM(d[k, ..., j]) +
                d[i − 1]d[k]d[j];
            if q < m[i, j] then m[i, j] ← q;
    return(m[i, j]);
```

ΔΠ vs ΔκΒ

- Δυναμικός Προγραμματισμός και Διαίρει-και-Βασίλευε επιλύουν προβλήματα συνδυάζοντας λύσεις κατάλληλα επιλεγμένων υπο-προβλημάτων.
- ΔκΒ είναι φύσει αναδρομική μέθοδος (top-down). Υλοποίηση με επανάληψη: σπάνια (μπορεί και πιο αργή).
- ΔκΒ επιλύει ανεξάρτητα υπο-προβλήματα.
  - Εφαρμόζεται όταν το πρόβλημα αναλύεται σε ανεξάρτητα υπο-προβλήματα.
  - Ανεξάρτητα υποπροβλήματα: συνήθως σημαντικά μικρότερου μεγέθους.
  - Απότομη μείωση μεγέθους ⇒ δέντρο αναδρομής **λογαριθμικού ύψους**

ΔΠ vs ΔκΒ

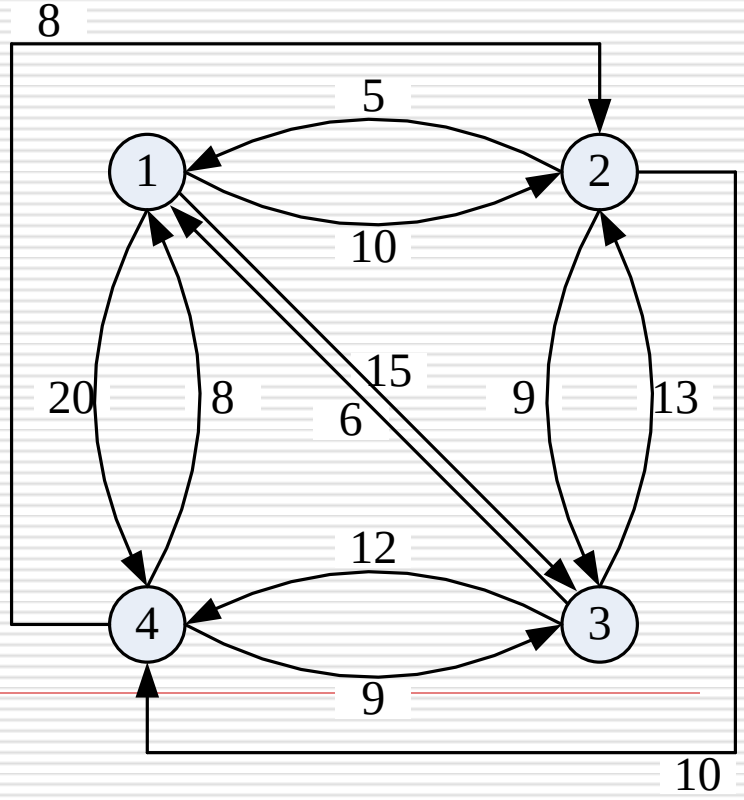
- ΔΠ «κτίζει» βέλτιστη λύση προβλήματος από βέλτιστες λύσεις υπο-προβλημάτων (bottom-up).
  - ΔΠ ξεκινά με στοιχειώδη στιγμιότυπα.
  - Συνδυάζει λύσεις για να βρει λύσεις σε μεγαλύτερα.
- ΔΠ εφαρμόζεται όταν υπο-προβλήματα επικαλύπτονται. Αποθηκεύει επιμέρους λύσεις για να μην υπολογίζει πάλι.
  - «Προγραμματισμός»: διαδικασία συμπλήρωσης πίνακα με ενδιάμεσα αποτελέσματα (Bellman, ~1950).
- ΔΠ εφαρμόζεται όταν ισχύει αρχή βελτιστότητας.
  - Διατύπωση αναδρομικής εξίσωσης για βέλτιστη λύση.
  - Βέλτιστη λύση υπολογίζεται bottom-up για αποδοτικότητα.
  - Βέλτιστος συνδυασμός υπο-λύσεων προσδιορίζει βέλτιστη λύση.
  - Top-down επίλυση: memoization.

Πρόβλημα Πλανόδιου Πωλητή (TSP)

- Δίνονται *n* σημεία $N = \{1, 2, \ldots, n\}$ και αποστάσεις τους $d : N \times N \mapsto \mathbb{R}_+$
  - Απόσταση i → j = d_{ij} , απόσταση j → i = d_{ji}
  - Γενική περίπτωση: όχι συμμετρικές αποστάσεις, όχι τριγωνική ανισότητα.
- Ζητείται μια **περιοδεία ελάχιστου** συνολικού **μήκους**.
  - Περιοδεία: κύκλος που διέρχεται από κάθε σημείο μία φορά.
  - Περιοδεία: μετάθεση σημείων $\pi : N \mapsto N$, $\pi(1) = 1$
    Μετάθεση (permutation): 1-1 και επί αντιστοιχία N με N.
    - Π.χ. $\begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 5 & 2 & 7 & 3 & 8 & 4 & 6 \end{matrix}$
  - Μήκος περιοδείας *π*: $L(\pi) = d_{\pi(n)1} + \sum_{i=1}^{n-1} d_{\pi(i)\pi(i+1)}$

Πρόβλημα Πλανόδιου Πωλητή

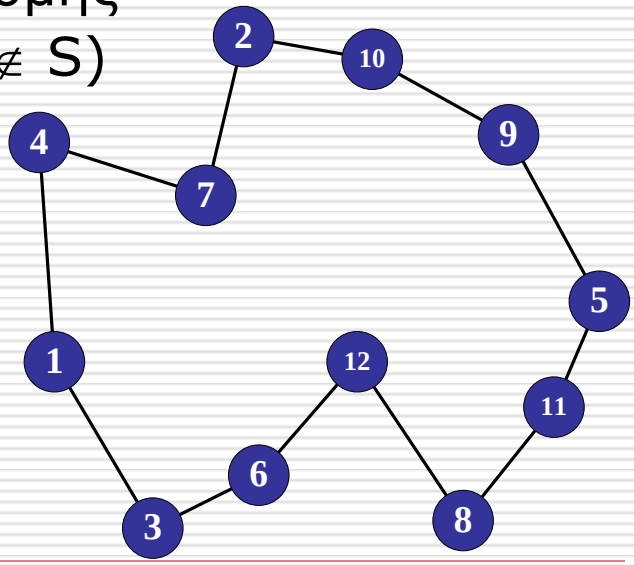
- Πρόβλημα συνδυαστικής βελτιστοποίησης:
  - Κάθε περιοδεία **εφικτή λύση** με αντικειμ. τιμή **μήκος**.
  - Ζητούμενο: περιοδεία ελάχιστου μήκους (**βέλτιστη**).
- Εξαντλητική αναζήτηση:
  - #περιοδειών = (n – 1)!
  - Χρόνος Ω(n!)
- TSP είναι **NP**-δύσκολο και δεν υπάρχει «γρήγορος» (πολυωνυμικός) αλγόριθμος.
  - Δυναμικός προγραμματισμός λύνει γενική περίπτωση σε χρόνο $\Theta(n^2\, 2^n)$.



Αρχή Βελτιστότητας

- Βέλτιστη περιοδεία ξεκινάει $1 \rightarrow i$ και συνεχίζει …
  - … από $i \rightarrow$ όλα τα σημεία $N \setminus \{ 1, i \} \rightarrow 1$.
  - Αυτό το τμήμα βέλτιστο με αυτή την ιδιότητα.
    - Διαφορετικά, βελτιώνω τμήμα και περιοδεία συνολικά!
- Έστω $L(i, S)$ ελάχιστο μήκος διαδρομής

  $i \rightarrow$ όλο το $S \rightarrow 1$ $(i, 1 \notin S)$
  - Για|S| = 0: $L(i, \emptyset) = d_{i1}, \forall i \neq 1$
  - Για|S| = 1: $L(i, \{j\}) = d_{ij} + d_{j1}$



Αρχή Βελτιστότητας

- Έστω $L(i, S)$ ελάχιστο μήκος διαδρομής
  $i \rightarrow$ όλο το $S \rightarrow 1$, $(i, 1 \notin S)$.
  - Για $S \subset N \setminus \{1\}$ θεωρούμε $i \neq 1$ (το 1 τελικό σημείο εκκίν.).
  - Για $|S| = 2$:

$$L(i, \{j, \ell\}) = \min\{d_{ij} + d_{j\ell} + d_{\ell 1}, d_{i\ell} + d_{\ell j} + d_{j1}\}$$
$$L(i, \{j, \ell\}) = \min\{d_{ij} + L(j, \{\ell\}), d_{i\ell} + L(\ell, \{j\})\}$$

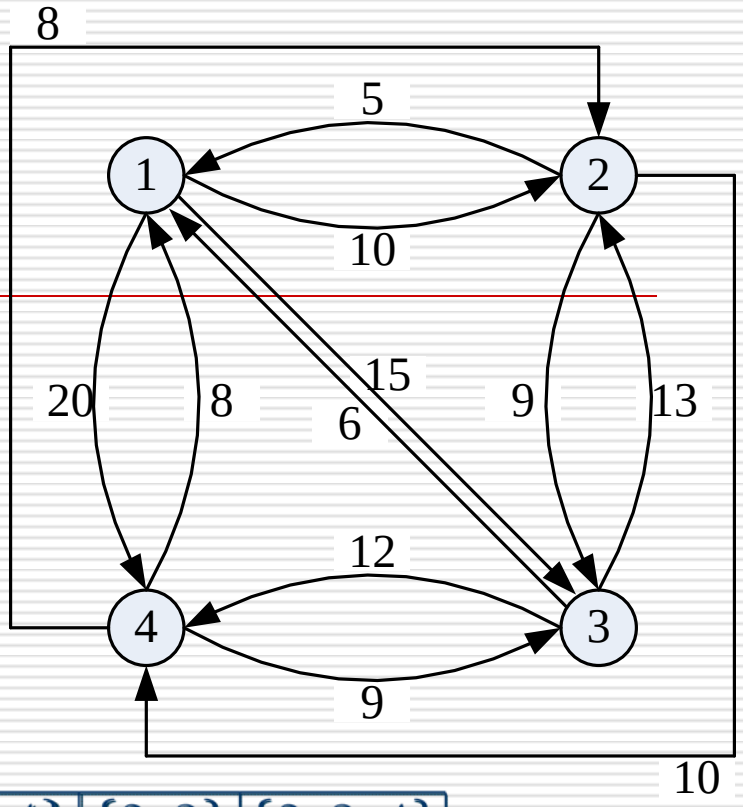
  - Υπολογίζω $L(i, S)$, $|S| = k$, αν γνωρίζω όλα τα $L(j, S \setminus \{j\})$:

$$L(i, S) = \min_{j \in S}\{d_{ij} + L(j, S \setminus \{j\})\}$$

  - Υπολογίζω όλες τις βέλτιστες «υπο-περιοδείες» που τελειώνουν στο 1 και έχουν μήκος 1, 2, 3, 4, …, εξετάζοντας όλα τα υποσύνολα κατάλληλου μεγέθους.

Παράδειγμα

$$L(i, S) = \min_{j \in S}\{d_{ij} + L(j, S \setminus \{j\})\}$$



$i \mid S$	$\emptyset$	$\{2\}$	$\{3\}$	$\{4\}$	$\{3,4\}$	$\{2,4\}$	$\{2,3\}$	$\{2,3,4\}$
1								**35 (2)**
2	5		15 (3)	18 (4)	**25 (4)**			
3	**6**	18 (2)		20 (4)		25 (4)		
4	8	13 (2)	**15 (3)**				23 (2)	

- Βέλτιστη περιοδεία **1, 2, 4, 3** μήκους **35.**

Υλοποίηση

$$L(i,S) = \min_{j \in S}\{d_{ij} + L(j, S \setminus \{j\})$$

```
TSP(d[1...n][1...n])
    for i ← 2 to n do L(i, ∅) ← d[i, 1];
    for k ← 1 to n − 2 do
        for all S ⊂ N \ {1}, |S| = k do
            for all i ∈ (N \ {1}) \ S do
                q ← ∞;
                for all j ∈ S do
                    if d[i, j] + L(j, S \ {j}) < q then
                        q ← d[i, j] + L(j, S \ {j});   t ← j;
                L(i, S) ← q;   J(i, S) ← t;
    q ← ∞;
    for j ← 2 to n do
        if d[1, j] + L(j, N \ {1, j}) < q then
            q ← d[1, j] + L(j, N \ {1, j});    t ← j;
    L(1, N \ {1}) ← q;   J(1, N \ {1}) ← t;
    return(L(1, N \ {1}), J);
```

Μνήμη $\Theta(n\,2^n)$

Χρόνος $\Theta(n^2\,2^n)$

20 σημεία:

$20! = 2.4 \times 10^{18}$

$20^2\,2^{20} = 4.2 \times 10^8$