Δυναμικός Προγραμματισμός

Διδάσκοντες: **Σ. Ζάχος, Δ. Φωτάκης**
Επιμέλεια διαφανειών: **Δ. Φωτάκης**

Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών

Εθνικό Μετσόβιο Πολυτεχνείο

Τρίγωνο του Pascal

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

n										
0	1									
1	1	1								
2	1	2	1							
3	1	3	3	1						
4	1	4	6	4	1					
5	1	5	10	10	5	1				
6	1	6	15	20	15	6	1			
7	1	7	21	35	35	21	7	1		
8	1	8	28	56	70	56	28	8	1	
9	1	9	36	84	126	126	84	36	9	1

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

- 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.

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

  - 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) }

Πρόβλημα Σακιδίου

- Πρόβλημα συνδυαστικής βελτιστοποίησης:
  - Συλλογή που χωράει **εφικτή λύση**. Αντιστοιχεί σε **αξία**.
  - 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, …, *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 $B - (f_n s_n + f_{n-1} s_{n-1})$.
- Αν γνωρίζουμε βέλτιστη αξία για αντικείμενα $N \setminus \{n\}$ και σακίδια μεγέθους B και $B - s_n$
  - … αποφασίζουμε αν αντικείμενο n στη βέλτιστη λύση!

Αναδρομική Εξίσωση

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

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

- Βέλτιστη αξία με αντικείμενα {1, …, *i*} και σακίδιο μεγέθους *b* : $P(i,b)$

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

Παράδειγμα

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

- 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 ναι, πρέπει πολυώνυμο του μεγέθους εισόδου!
  - Μέγεθος εισόδου: $O(n(\log_2 B + \log_2 P_{\max}))$
  - Χρόνος πολυωνυμικός στο *n* αλλά εκθετικός στο $\log_2 B$
- Αριθμητικά προβλήματα:
  - Μέγεθος αριθμών πολύ μεγαλύτερο (π.χ. εκθετικό) από πλήθος «βασικών συνιστωσών» (ότι συμβολίζουμε με n).
- Αλγόριθμος πολυωνυμικό χρόνου:
  $$O((n \log \text{max\_num})^k),\ \text{σταθερά } k \geq 1$$
- Αλγόριθμος **ψευδο-πολυωνυμικού** χρόνου:
  $$O((n\, \text{max\_num})^k),\ \text{σταθερά } k \geq 1$$

Απληστία vs Δυναμικός Προγρ.

- Διακριτό Πρόβλ. Σακιδίου: **όχι** ιδιότητα άπληστης επιλογής.
  - Π.χ. Αντικείμενα: {(1, 1+ε), (Β, Β)}. Σακίδιο μεγέθους Β.
- **Απληστία** και **Δυναμικός Προγραμματισμός**:
  - Αρχή βελτιστότητας.
- **Δυναμικός Προγραμματισμός**: αναδρομή
  - Βέλτιστη λύση σε **όλα** τα υπο-προβλήματα που εμπλέκονται στην αναδρομή.
    - Διακριτό Σακίδιο: Βέλτιστη λύση με πρώτα *i* αντικείμενα για **όλα** τα μεγέθη σακιδίου!
  - Συνδυάζει «κατάλληλες» επιμέρους λύσεις για βέλτιστη.
  - Λύση **όλων** 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: φαίνεται καλύτερη με βάση τρέχουσα κατάσταση και κάποιο (απλό) κριτήριο.
  - Λύση μόνο «**αναγκαίων**» υπο-προβλημάτων: αποδοτικό υπολογιστικά αλλά δεν δίνει πάντα τη βέλτιστη λύση.
    - Γρήγοροι, απλοί και «φυσιολογικοί» αλγόριθμοι!
    - (Καλές) προσεγγιστικές λύσεις σε πολλά προβλήματα.
    - Βέλτιστη λύση μόνο όταν άπληστη επιλογή (ως προς συγκεκριμένο κριτήριο επιλογής).

Αλυσιδωτός Πολ/μός Πινάκων

- Γινόμενο πινάκων Α (p × q) επί Β (q × r) σε χρόνο Θ(p q r).
  (μετράμε μόνο πολ/μούς μεταξύ αριθμών).
- Συντομότερος τρόπος υπολογισμού γινομένου

$$\begin{array}{cccccc} A_1 & A_2 & A_3 & \ldots & A_{n-1} & A_n \\ (d_0 \times d_1) & (d_1 \times d_2) & (d_2 \times d_3) & \ldots & (d_{n-2} \times d_{n-1}) & (d_{n-1} \times d_n) \end{array}$$

  - Πολλαπλασιασμός πινάκων είναι πράξη προσεταιριστική (αποτέλεσμα ανεξάρτητο από υπολογισμό επιμέρους γινομέν.)
  - Ο χρόνος υπολογισμού εξαρτάται από τη σειρά!

$$\begin{array}{ccc} A_1 & A_2 & A_3 \\ (1 \times 100) & (100 \times 3) & (3 \times 1) \end{array}$$

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

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

Πολλαπλασιασμός Πινάκων

$$\begin{array}{cccc} A_1 & A_2 & A_3 & A_4 \\ (13 \times 5) & (5 \times 89) & (89 \times 3) & (3 \times 34) \end{array}$$

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:

$$\begin{array}{ccccc} A_1 & A_2 & A_3 & \ldots & A_{n-1} & A_n \\ (d_0 \times d_1) & (d_1 \times d_2) & (d_2 \times d_3) & \ldots & (d_{n-2} \times d_{n-1}) & (d_{n-1} \times d_n) \end{array}$$

  Με ποια σειρά θα υπολογιστεί το γινόμενο $A_1 A_2 \ldots A_n$ ώστε να ελαχιστοποιηθεί #αριθμ. πολ/μών.
- Πρόβλημα συνδυαστικής βελτιστοποίησης:
  - Κάθε σειρά υπολογισμού υπολογίζει γινόμενο πινάκων με κάποιο #αριθμ. πολ/μών.
  - Ζητείται η σειρά με ελάχιστο #αριθμ. πολ/μών.
- Αποδοτικός αλγόριθμος για υπολογισμό καλύτερης σειράς για αλυσιδωτό πολ/μου n πινάκων.

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:

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

  - Λύση (n-1)-οστός αριθμός Catalan: $P(n) = \frac{1}{n}\binom{2(n-1)}{n-1} = \Omega(\frac{4^n}{n^{3/2}})$
- 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 $m[i,j] = \text{βέλτιστος \#πολ/μών}(A_{i..j})$
- 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**:

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

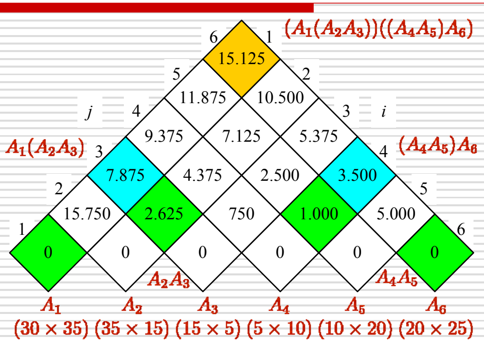
Δυναμικός Προγραμματισμός

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

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

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

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 (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]);
```

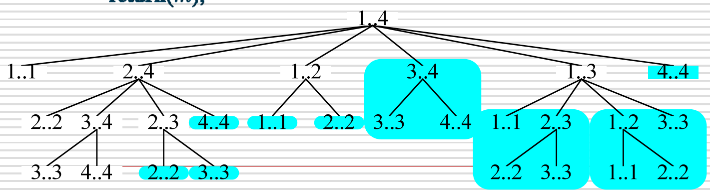
- Χρόνος $O(n^3)$ και μνήμη $O(n^2)$, μειώνεται σε $O(n)$.

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

Υλοποίηση (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 **$\Omega(2^n)$** !



Αναδρομή με Μνήμη

- Ο αναδρομικός αλγόριθμος αποθηκεύει τιμές σε πίνακα. Κάθε τιμή υπολογίζεται μία φορά.
  - 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).
- ΔκΒ επιλύει υπο-προβλήματα ανεξάρτητα.
  - Εφαρμόζεται όταν παράγονται ανεξάρτητα υπο-προβ/τα.
  - Ειδάλλως ίδια υπο-προβλήματα λύνονται πολλές φορές: Σπατάλη υπολογιστικού χρόνου.

ΔΠ vs ΔκΒ

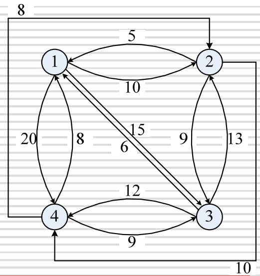
- ΔΠ «κτίζει» βέλτιστη λύση προβ/τος από βέλτιστες λύσεις υπο-προβ/των (bottom-up).
  - ΔΠ ξεκινά με στοιχειώδη στιγμιότυπα.
  - Συνδυάζει λύσεις για να βρει λύσεις σε μεγαλύτερα.
- ΔΠ εφαρμόζεται όταν υπο-προβ/τα επικαλύπτονται. Αποθηκεύει επιμέρους λύσεις για να μην υπολογίζει πάλι.
  - «Προγραμματισμός» διαδικασία συμπλήρωσης πίνακα με ενδιάμεσα αποτελέσματα (Bellman, 50's).
- ΔΠ εφαρμόζεται όταν ισχύει αρχή βελτιστότητας.
  - Διατύπωση αναδρομικής εξίσωσης για βέλτιστη λύση.
  - Αναδρομική εξίσωση λύνεται bottom-up για βέλτιστη **τιμή**.
  - Επιλογές κατά την επίλυση απαρτίζουν βέλτιστη λύση.

Πρόβλημα Πλανόδιου Πωλητή

- 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. $1\,2\,3\,4\,5\,6\,7\,8$
      $1\,5\,2\,7\,3\,8\,4\,6$
  - Μήκος περιοδείας π: $L(\pi) = d_{\pi(n)1} + \sum_{i=1}^{n-1} d_{\pi(i)\pi(i+1)}$

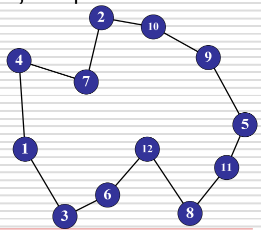
Πρόβλημα Πλανόδιου Πωλητή

- Πρόβλημα συνδυαστικής βελτιστοποίησης:
  - Κάθε περιοδεία **εφικτή λύση** και αντιστοιχεί σε **μήκος**.
  - Ζητούμενο: (**βέλτιστη**) περιοδεία ελάχιστου μήκους.
- Εξαντλητική αναζήτηση:
  - #περιοδειών = (n – 1)!
  - Χρόνος Ω(n!)
- ΠΠΠ 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 $1 \to j$ και συνεχίζει …
  - … από $j \to$ όλα τα σημεία $N \setminus \{1, j\} \to 1$.
  - Αυτό το τμήμα βέλτιστο με αυτή την ιδιότητα.
    - Διαφορετικά, βελτιώνω τμήμα και περιοδεία συνολικά!
- Έστω $L(i, S)$ ελάχιστο μήκος για να ξεκινήσω από $i \to$ όλο το $S \to 1$, ($i \notin S$).
  - Αν $S \subset N \setminus \{1\}$, τότε $i \neq 1$ (το 1 προστίθεται τελευταίο).
  - Εύκολα $L(i,\emptyset) = d_{i1}, \forall i \neq 1$
  - $L(i,\{j\}) = d_{ij} + d_{j1}$



Αρχή Βελτιστότητας

- Έστω $L(i, S)$ ελάχιστο μήκος για να ξεκινήσω από $i \to$ όλο το $S \to 1$, ($i \notin S$).
  - Αν $S \subset N \setminus \{1\}$, τότε $i \neq 1$ (το 1 προστίθεται τελευταίο).
  - Εύκολα για $|S| = 0, 1, 2$:

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

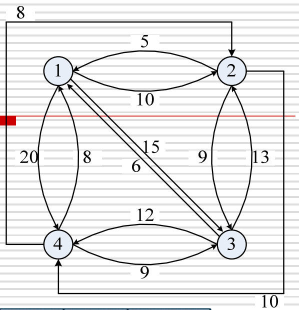
  - Υπολογίζω $L(i, S)$, $|S| = k$, αν γνωρίζω όλα τα $L(j, S \setminus \{j\})$:

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

  - Υπολογίζω όλες τις βέλτιστες «υπο-περιοδείες» που τελειώνουν στο 1 και έχουν μήκος 1, 2, 3, 4, …

Παράδειγμα

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



$i \vert S$	$\emptyset$	$\{2\}$	$\{3\}$	$\{4\}$	$\{3,4\}$	$\{2,4\}$	$\{2,3\}$	$\{2,3,4\}$
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:

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

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