The Master Theorem

Εναλλακτικά:

$$\text{If } f(n) \in \Theta(n^d) \quad d \geq 0$$

$$T(n) \in \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log n) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

Ασκήσεις (1/12)

Να λύσετε την επόμενη αναδρομική σχέση:

$$T(n)=3T(n/2)+n^2$$

Ασκήσεις (2/12)

Να λύσετε την επόμενη αναδρομική σχέση: T(n)=3T(n/2)+n^2

Λύση

$n^{\log_2 3} = n^{1.58}$

Άρα f(n) = Ω($n^{1.58}$)

Κοιτάζουμε την ανίσωση: af(n/b)<=cf(n)

Έχουμε: 3/4 n^2<=c n^2, ισχύει άρα 3η περίπτωση του Master, συνεπώς T(n)=Θ(n^2)

Ασκήσεις (3/12)

Να λύσετε την επόμενη αναδρομική σχέση:

$$T(n)=4T(n/2)+n^2$$

Ασκήσεις (4/12)

Να λύσετε την επόμενη αναδρομική σχέση: T(n)=4T(n/2)+n^2

Λύση

$n^{\log_2 4} = n^2$

Άρα f(n) = Θ(n^2)

2η περίπτωση του Master, συνεπώς T(n)=Θ($n^2 logn$)

Ασκήσεις (5/12)

Να λύσετε την επόμενη αναδρομική σχέση:

$$T(n)=T(n/2)+2^n$$

Ασκήσεις (6/12)

Να λύσετε την επόμενη αναδρομική σχέση: T(n)=T(n/2)+2^n

Λύση

$n^{\log_2 1} = n^0$=1

Άρα f(n) = Ω(n^0)

Κοιτάζουμε την ανίσωση: af(n/b)<=cf(n)

Έχουμε: $2^{n/2}$<=c 2^n, ισχύει άρα 3η περίπτωση του Master, συνεπώς T(n)=Θ(2^n)

Ασκήσεις (7/12)

Να λύσετε την επόμενη αναδρομική σχέση:

$$T(n)=16T(n/4)+n$$

Ασκήσεις (8/12)

Να λύσετε την επόμενη αναδρομική σχέση: T(n)=16T(n/4)+n

Λύση

$n^{\log_4 16} = n^2$

Άρα f(n) = O(n^2)

1η περίπτωση του Master, συνεπώς T(n)=Θ(n^2)

Ασκήσεις (9/12)

Να λύσετε την επόμενη αναδρομική σχέση:

$$T(n)=\sqrt{2}T(n/2)+logn$$

Ασκήσεις (10/12)

Να λύσετε την επόμενη αναδρομική σχέση: T(n)=$\sqrt{2}$T(n/2)+$logn$

Λύση

$n^{\log_2 \sqrt{2}} = n^{1/2}$

Άρα f(n) = O($n^{1/2}$)

1η περίπτωση του Master, συνεπώς T(n)=Θ($n^{1/2}$)

Ασκήσεις (11/12)

Να λύσετε την επόμενη αναδρομική σχέση:

$$T(n)=3T(n/2)+n$$

Ασκήσεις (12/12)

Να λύσετε την επόμενη αναδρομική σχέση: T(n)=3T(n/2)+n

Λύση

$n^{\log_2 3} = n^{1.58}$

Άρα f(n) = O($n^{1.58}$)

1η περίπτωση του Master, συνεπώς T(n)=Θ($n^{1.58}$)

Διαίρει και βασίλευε

(συνέχεια)

Πολ/μός Μεγάλων Αριθμών (1/9)

Κάποιες εφαρμογές (π.χ. κρυπτογραφία) απαιτούν το χειρισμό ακεραίων ακόμη και με 100 ψηφία

Αυτού του είδους οι αριθμοί δεν μπορούν να χωρέσουν σε μια λέξη ενός υπολογιστή

Απαιτούν ειδική μεταχείριση

Αν προσπαθήσουμε να πολλαπλασιάσουμε αριθμούς n ψηφίων χρειαζόμαστε n^2 πολλαπλασιασμούς (κάθε ένα ψηφίο του πρώτου αριθμού πολλαπλασιάζεται με όλα τα ψηφία του δεύτερου)

Πολ/μός Μεγάλων Αριθμών (2/9)

Βασική ιδέα αλγορίθμου

- Παράδειγμα: πολλαπλασιασμός του 23 και 14
- Οι αριθμοί γράφονται ως εξής

$$23 = 2 \cdot 10^1 + 3 \cdot 10^0 \quad \text{and} \quad 14 = 1 \cdot 10^1 + 4 \cdot 10^0$$

- Οπότε

$$\begin{aligned} 23 * 14 &= (2 \cdot 10^1 + 3 \cdot 10^0) * (1 \cdot 10^1 + 4 \cdot 10^0) \\ &= (2 * 1)10^2 + (2 * 4 + 3 * 1)10^1 + (3 * 4)10^0 \end{aligned}$$

Πολ/μός Μεγάλων Αριθμών (3/9)

Ο τελευταίο τύπος εξάγει το σωστό αποτέλεσμα αλλά απαιτεί 4 πολλαπλασιασμούς

Μπορεί να γίνει μια βελτίωση ως εξής:

$$2*4+3*1=(2+3)*(1+4)-2*1-3*4$$

Για δύο αριθμούς $a=a_1a_0$ και $b=b_1b_0$

Το γινόμενό τους μπορεί να δοθεί ως εξής:

$$c=a*b=c_2 10^2+c_1 10^1+c_0,$$ με

$$c_2=a_1*b_1$$
$$c_0=a_0*b_0$$
$$c_1=(a_1+a_0)*(b_1+b_0)-(c_2+c_0)$$

Πολ/μός Μεγάλων Αριθμών (4/9)

Εφαρμόζουμε τον τύπο για τον πολλαπλασιασμό αριθμών με n ψηφία

Διαιρούμε τους αριθμούς στη μέση

α_1 είναι τα πρώτα ψηφία του α και α_0 είναι τα υπόλοιπα

b_1 είναι τα πρώτα ψηφία του b και b_0 είναι τα υπόλοιπα

Συνεπώς:

$$a = a_1 10^{n/2} + a_0$$

$$b = b_1 10^{n/2} + b_0.$$

Πολ/μός Μεγάλων Αριθμών (5/9)

Οπότε

$$c = a * b = (a_1 10^{n/2} + a_0) * (b_1 10^{n/2} + b_0)$$
$$= (a_1 * b_1)10^n + (a_1 * b_0 + a_0 * b_1)10^{n/2} + (a_0 * b_0)$$
$$= c_2 10^n + c_1 10^{n/2} + c_0$$

με

$$c_2 = a_1 * b_1$$
$$c_0 = a_0 * b_0$$
$$c_1 = (a_1 + a_0) * (b_1 + b_0) - (c_2 + c_0)$$

Πολ/μός Μεγάλων Αριθμών (6/9)

Αν το n/2 είναι άρτιος μπορούμε να συνεχίσουμε με την ίδια μέθοδο για τα c_2, c_1, c_0

Συνεπώς αν το n είναι δύναμη του 2 μπορούμε εύκολα να καταλήξουμε στην αναδρομική σχέση

Η αναδρομική σχέση καταλήγει όταν n = 1

Μπορεί να καταλήξει όταν το n είναι πολύ μικρό

Απαιτούνται 3 πολλαπλασιασμοί αριθμών με n/2 ψηφία

Πολ/μός Μεγάλων Αριθμών (7/9)

Υπολογισμοί

$$M(n) = 3M(n/2) \quad \text{for } n > 1, \quad M(1) = 1.$$

$$M(2^k) = 3M(2^{k-1}) = 3[3M(2^{k-2})] = 3^2 M(2^{k-2})$$

$$= \cdots = 3^i M(2^{k-i}) = \cdots = 3^k M(2^{k-k}) = 3^k.$$

$$k = \log_2 n$$

$$a^{\log_b c} = c^{\log_b a}$$

$$M(n) = 3^{\log_2 n} = n^{\log_2 3} \approx n^{1.585}$$

Πολ/μός Μεγάλων Αριθμών (8/9)

Προσθέσεις και αφαιρέσεις

- Απαιτούνται πέντε προσθέσεις και μια αφαίρεση
- Παίρνουμε την ακόλουθη αναδρομική σχέση

$$A(n) = 3A(n/2) + cn \quad \text{for } n > 1, \quad A(1) = 1.$$

- Και καταλήγουμε ότι

$$A(n) \in \Theta(n^{\log_2 3}).$$

Πολ/μός Μεγάλων Αριθμών (9/9)

Η διαίρει και βασίλευε είναι γρηγορότερη ακόμα και για αριθμούς 8 ψηφίων σε σχέση με τις συμβατικές μεθόδους

Για αριθμούς με 300 ψηφία είναι δύο φορές πιο γρήγορη

Στις γλώσσες Java, C++ υπάρχουν ειδικές κλάσεις για χειρισμό μεγάλων αριθμών

Ερωτήματα:

- **Τι γίνεται αν οι δύο αριθμοί δεν είναι του ιδίου μεγέθους?**
- **Τι γίνεται αν το πλήθος των ψηφίων n δεν είναι πολλαπλάσιο του 2?**

Matrix Multiplication (1/6)

Αν $A = (a_{ij})$ και $B = (b_{ij})$ είναι nxn πίνακες τότε το γινόμενό τους είναι

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

Αλγόριθμος

```
SQUARE-MATRIX-MULTIPLY(A, B)
1  n = A.rows
2  let C be a new n × n matrix
3  for i = 1 to n
4      for j = 1 to n
5          c_ij = 0
6          for k = 1 to n
7              c_ij = c_ij + a_ik · b_kj
8  return C
```

Matrix Multiplication (2/6)

Προσέγγιση με διαίρει και βασίλευε

- Υποθέτουμε ότι σπάμε τους πίνακες σε 4 n/2 x n/2 πίνακες

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$$\begin{aligned} C_{11} &= A_{11} \cdot B_{11} + A_{12} \cdot B_{21} \\ C_{12} &= A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \\ C_{21} &= A_{21} \cdot B_{11} + A_{22} \cdot B_{21} \\ C_{22} &= A_{21} \cdot B_{12} + A_{22} \cdot B_{22} \end{aligned}$$

Matrix Multiplication (3/6)

Για κάθε μια από τις προηγούμενες 4 εξισώσεις απαιτούνται 2 πολ/μοι και η πρόσθεση των n/2 x n/2 γινομένων

Νέος αλγόριθμος

```
SQUARE-MATRIX-MULTIPLY-RECURSIVE(A, B)
 1  n = A.rows
 2  let C be a new n × n matrix
 3  if n == 1
 4      c11 = a11 · b11
 5  else partition A, B, and C as in equations (4.9)
 6      C11 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B11)
              + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B21)
 7      C12 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B12)
              + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B22)
 8      C21 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B11)
              + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B21)
 9      C22 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B12)
              + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B22)
10  return C
```

Matrix Multiplication (4/6)

Χωρίζουμε τους πίνακες χωρίς να αντιγράφουμε στοιχεία

Χρησιμοποιούμε index calculations

Αναγνωρίζουμε ένα υπο-πίνακα από ένα εύρος γραμμών και στηλών στον αρχικό πίνακα

Έστω T(n) είναι ο χρόνος για τον πολλαπλασιασμό των πινάκων

Όταν n=1 εκτελούμε ένα πολλαπλασιασμό, οπότε $T(1)=\Theta(1)$

Η αναδρομική σχέση ισχύει για n>1

Matrix Multiplication (5/6)

Το κόστος της γραμμής 5 του αλγορίθμου είναι Θ(1)

Στις γραμμές 6-9 εκτελούμε 8 κλήσεις της αναδρομής

Κάθε κλήση συνεισφέρει με T(n/2) στο συνολικό χρόνο

Άρα το συνολικό κόστος των κλήσεων είναι 8T(n/2)

Στις γραμμές 6-9 εκτελούμε 4 προσθέσεις πινάκων

Κάθε υπο-πίνακας περιέχει $n^2/4$ στοιχεία

Οι 4 προσθέσεις έχουν πολυπλοκότητα $\Theta(n^2)$

```
SQUARE-MATRIX-MULTIPLY-RECURSIVE(A, B)
1  n = A.rows
2  let C be a new n × n matrix
3  if n == 1
4      c11 = a11 · b11
5  else partition A, B, and C as in equations (4.9)
6      C11 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B11)
           + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B21)
7      C12 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B12)
           + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B22)
8      C21 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B11)
           + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B21)
9      C22 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B12)
           + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B22)
10 return C
```

Matrix Multiplication (6/6)

Ο συνολικός χρόνος για τις αναδρομικές κλήσεις και τους πολλαπλασιασμούς – προσθέσεις είναι:

$$\begin{aligned} T(n) &= \Theta(1) + 8T(n/2) + \Theta(n^2) \\ &= 8T(n/2) + \Theta(n^2) \end{aligned}$$

Αν στη διάσπαση των πινάκων επιλέξουμε την αντιγραφή (κόστος $\Theta(n^2)$) η αναδρομική σχέση δεν θα αλλάξει

Συνεπώς $$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 8T(n/2) + \Theta(n^2) & \text{if } n > 1 \end{cases}$$

Πολυπλοκότητα: $O(n^3)$ από την $1^η$ περίπτωση του Θ.Κ.

Strassen's Method(1/7)

1. Divide the input matrices A and B and output matrix C into $n/2 \times n/2$ submatrices . This step takes $\Theta(1)$ time by index calculation, just as in SQUARE-MATRIX-MULTIPLY-RECURSIVE.
2. Create 10 matrices $S_1, S_2, \ldots, S_{10}$, each of which is $n/2 \times n/2$ and is the sum or difference of two matrices created in step 1. We can create all 10 matrices in $\Theta(n^2)$ time.
3. Using the submatrices created in step 1 and the 10 matrices created in step 2, recursively compute seven matrix products $P_1, P_2, \ldots, P_7$. Each matrix P_i is $n/2 \times n/2$.
4. Compute the desired submatrices $C_{11}, C_{12}, C_{21}, C_{22}$ of the result matrix C by adding and subtracting various combinations of the P_i matrices. We can compute all four submatrices in $\Theta(n^2)$ time.

Strassen's Method (2/7)

Αναδρομική σχέση

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 7T(n/2) + \Theta(n^2) & \text{if } n > 1 \end{cases}$$

Με τη χρήση του master θεωρήματος παίρνουμε ότι $T(n) = \Theta(n^{\lg 7})$.

Strassen's Method (3/7)

Δημιουργία 10 υπο-πινάκων στο 2ο βήμα

$$
\begin{aligned}
S_1 &= B_{12} - B_{22} \\
S_2 &= A_{11} + A_{12} \\
S_3 &= A_{21} + A_{22} \\
S_4 &= B_{21} - B_{11} \\
S_5 &= A_{11} + A_{22} \\
S_6 &= B_{11} + B_{22} \\
S_7 &= A_{12} - A_{22} \\
S_8 &= B_{21} + B_{22} \\
S_9 &= A_{11} - A_{21} \\
S_{10} &= B_{11} + B_{12}
\end{aligned}
$$

Κόστος $\Theta(n^2)$

Strassen's Method (4/7)

Πολλαπλασιασμός στο 3ο βήμα

$$
\begin{aligned}
P_1 &= A_{11} \cdot S_1 &&= A_{11} \cdot B_{12} - A_{11} \cdot B_{22} \\
P_2 &= S_2 \cdot B_{22} &&= A_{11} \cdot B_{22} + A_{12} \cdot B_{22} \\
P_3 &= S_3 \cdot B_{11} &&= A_{21} \cdot B_{11} + A_{22} \cdot B_{11} \\
P_4 &= A_{22} \cdot S_4 &&= A_{22} \cdot B_{21} - A_{22} \cdot B_{11} \\
P_5 &= S_5 \cdot S_6 &&= A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22} \\
P_6 &= S_7 \cdot S_8 &&= A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22} \\
P_7 &= S_9 \cdot S_{10} &&= A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12}
\end{aligned}
$$

Strassen's Method (5/7)

Προσθέσεις στο 4ο βήμα

$$C_{11} = P_5 + P_4 - P_2 + P_6$$

$$\begin{array}{l}
A_{11}\cdot B_{11} + A_{11}\cdot B_{22} + A_{22}\cdot B_{11} + A_{22}\cdot B_{22} \\
\qquad\qquad - A_{22}\cdot B_{11} \qquad\qquad + A_{22}\cdot B_{21} \\
\quad - A_{11}\cdot B_{22} \qquad\qquad\qquad\qquad - A_{12}\cdot B_{22} \\
\qquad\qquad\qquad\qquad - A_{22}\cdot B_{22} - A_{22}\cdot B_{21} + A_{12}\cdot B_{22} + A_{12}\cdot B_{21} \\
\hline
A_{11}\cdot B_{11} \qquad\qquad\qquad\qquad\qquad\qquad + A_{12}\cdot B_{21}
\end{array}$$

Strassen's Method (6/7)

$$C_{12} = P_1 + P_2$$

$$\begin{array}{r} A_{11} \cdot B_{12} - A_{11} \cdot B_{22} \\ + A_{11} \cdot B_{22} + A_{12} \cdot B_{22} \\ \hline A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \end{array}$$

$$C_{21} = P_3 + P_4$$

$$\begin{array}{r} A_{21} \cdot B_{11} + A_{22} \cdot B_{11} \\ - A_{22} \cdot B_{11} + A_{22} \cdot B_{21} \\ \hline A_{21} \cdot B_{11} + A_{22} \cdot B_{21} \end{array}$$

Strassen's Method (7/7)

$$C_{22} = P_5 + P_1 - P_3 - P_7$$

$$\begin{array}{llll}
A_{11}\cdot B_{11} + A_{11}\cdot B_{22} + A_{22}\cdot B_{11} + A_{22}\cdot B_{22} & & & \\
\quad - A_{11}\cdot B_{22} & & + A_{11}\cdot B_{12} & \\
& - A_{22}\cdot B_{11} & & - A_{21}\cdot B_{11} \\
- A_{11}\cdot B_{11} & & - A_{11}\cdot B_{12} + A_{21}\cdot B_{11} + A_{21}\cdot B_{12} & \\
\hline
& A_{22}\cdot B_{22} & & + A_{21}\cdot B_{12}
\end{array}$$

Brute Force and Exhaustive Search

Εισαγωγή

Πρόκειται για μια άμεση μέθοδο για την επίλυση προβλημάτων συχνά βασισμένη στη διατύπωση του προβλήματος και στους ορισμούς των εννοιών που εμπλέκονται

Το όνομα της μεθόδου σημαίνει: Just do it!

Πολλές φορές είναι η πιο απλή μέθοδος

Παράδειγμα:

- Υπολογισμός της δύναμης α^{ν}

$$a^n = \underbrace{a * \cdots * a}_{n \text{ times}}$$

Χαρακτηριστικά

Εφαρμόζεται σε ένα μεγάλο εύρος προβλημάτων

- Μάλλον είναι η μέθοδος που δύσκολα κάποιος βρίσκει προβλήματα που να μην μπορεί να τα χειριστεί

Οδηγεί σε ‘λογικούς’ αλγορίθμους με πρακτική σημασία

Παρά το γεγονός ότι γενικά δεν είναι αποδοτική μέθοδος, μπορεί να χρησιμοποιηθεί για μικρού μεγέθους προβλήματα

Μπορεί να αποτελέσει μια θεωρητική βάση για τη σύγκριση με άλλες μεθοδολογίες

Εφαρμογές

Αριθμητικά προβλήματα

- Αποδεκτή επίδοση
- Μπορεί να χρησιμοποιηθεί για μεγάλου μεγέθους προβλήματα

Συνδυαστικά Προβλήματα

Εύρεση Διαιρετών

Δοσμένου ενός αριθμού n να βρεθούν οι διαιρέτες του

Απαριθμούμε όλους τους ακέραιους από το 1 μέχρι το n (υποψήφιοι διαιρέτες)

Ελέγχουμε αν ο καθένας από αυτούς είναι διαιρέτης του n

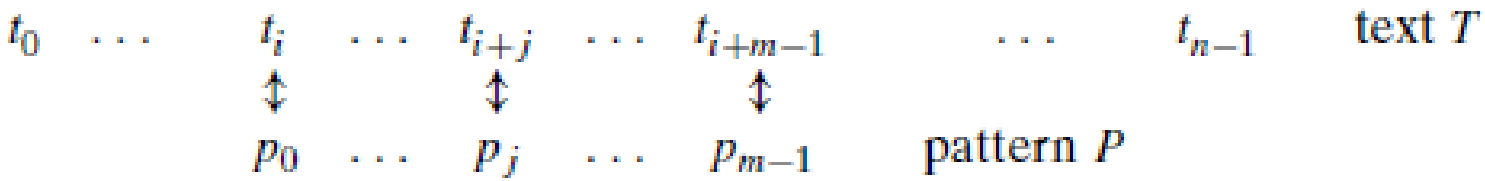
```
d ← 1
while ( d < n + 1 ) do
        if ( n mod d == 0 )
        then output (d)
        d ← d + 1
```

String Matching (1/6)

Δεδομένου ενός αλφαριθμητικού n χαρακτήρων (text) και ενός αλφαριθμητικού m χαρακτηρων (pattern) με m <= n να βρεθεί ένα υπο-αλφαριθμητικό του text που ταιριάζει με το pattern

Θέλουμε να βρούμε το i-index του πιο αριστερά χαρακτήρα τέτοιο ώστε

$$t_i = p_0, \ldots, t_{i+j} = p_j, \ldots, t_{i+m-1} = p_{m-1}:$$



Μπορεί ο αλγόριθμος να συνεχίζει για πολλαπλές εμφανίσεις

String Matching (2/6)

Η προσέγγιση brute force είναι προφανής:

- ‘Ευθυγραμίζουμε’ το pattern με τους πρώτους m χαρακτήρες του text και συγκρίνουμε τα αντίστοιχα ζεύγη
- Συνεχίζουμε μέχρι να ταιριάξουν όλοι οι χαρακτήρες ή να βρούμε μια διαφορά
- Αν βρούμε διαφορά, τότε μετακινούμε μια θέση το pattern προς τα δεξιά και ξεκινούμε από την αρχή
- Η τελευταία θέση στην οποία μπορεί να υπάρχει το pattern είναι η n – m
- Πέρα από αυτή τη θέση δεν υπάρχουν αρκετοί χαρακτήρες

String Matching (3/6)

Αλγόριθμος

ALGORITHM *BruteForceStringMatch*$(T[0..n-1], P[0..m-1])$

```
//Implements brute-force string matching
//Input: An array T[0..n − 1] of n characters representing a text and
//       an array P[0..m − 1] of m characters representing a pattern
//Output: The index of the first character in the text that starts a
//        matching substring or −1 if the search is unsuccessful
for i ← 0 to n − m do
    j ← 0
    while j < m and P[j] = T[i + j] do
        j ← j + 1
    if j = m return i
return −1
```

String Matching (4/6)

Παράδειγμα εκτέλεσης:

```
N  O  B  O  D  Y  _  N  O  T  I  C  E  D  _  H  I  M
N  O  T
   N  O  T
      N  O  T
         N  O  T
            N  O  T
               N  O  T
                  N  O  T
                     N  O  T
                        N  O  T
```

String Matching (5/6)

Η χειρότερη περίπτωση είναι να γίνουν m συγκρίσεις προτού μετακινηθεί το pattern προς τα δεξιά

Ο μέγιστος αριθμός προσπαθειών είναι $n - m + 1$

Στη χειρότερη περίπτωση γίνονται $m (n - m + 1)$ συγκρίσεις

Πολυπλοκότητα $O(m \cdot n)$

Η μέση περίπτωση έχει πολυπλοκότητα $\Theta(n)$

Προχωρημένοι αλγόριθμοι για string matching θα συζητηθούν σε επόμενα μαθήματα

String Matching (6/6)

Άσκηση

Καθορίστε το πλήθος των συγκρίσεων (χαρακτήρες) που γίνονται από τον brute force αλγόριθμο για string matching και για το pattern **GANDHI** ως προς το text

THERE_IS_MORE_TO_LIFE_THAN_INCREASING_ITS_SPEED

Closest Pair Problem (1/4)

Αναζητούμε τα κοντινότερα σημεία μέσα σε ένα πλήθος n σημείων

Ανήκει στη κατηγορία της υπολογιστικής γεωμετρίας

Παραδείγματα:

- θέσεις αεροπλάνων
- θέσεις γραφείων
- εγγραφές σε βάσεις δεδομένων
- ανάλυση συστάδων (cluster)
- ....

Closest Pair Problem (2/4)

Υποθέτουμε σημεία στο επίπεδο

Απόσταση σημείων

$$d(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

Υπολογίζουμε την απόσταση μεταξύ κάθε ενός ζεύγους σημείων και παίρνουμε το ζεύγος με τη μικρότερη απόσταση

Για να μην μετρήσουμε την απόσταση του ίδιου ζεύγους δυο φορές εστιάζουμε σε ζεύγη (p_i, p_j) με $i < j$

Closest Pair Problem (3/4)

Αλγόριθμος

ALGORITHM *BruteForceClosestPair*(P)

//Finds distance between two closest points in the plane by brute force
//Input: A list P of n $(n \geq 2)$ points $p_1(x_1, y_1), \ldots, p_n(x_n, y_n)$
//Output: The distance between the closest pair of points
$d \leftarrow \infty$
for $i \leftarrow 1$ **to** $n - 1$ **do**
    for $j \leftarrow i + 1$ **to** n **do**
        $d \leftarrow \min(d, sqrt((x_i - x_j)^2 + (y_i - y_j)^2))$ //*sqrt* is square root
return d

Closest Pair Problem (4/4)

Η βασική λειτουργία του αλγορίθμου είναι ο υπολογισμός της τετραγωνικής ρίζας

Ο υπολογισμός της ρίζας είναι προσεγγιστικός πολλές φορές

Αποφεύγουμε τη σύγκριση της ρίζας αν συγκρίνουμε τα $(x_i - x_j)^2 + (y_i - y_j)^2$

Το πλήθος των εκτελέσεων της βασικής λειτουργίας είναι:

$$C(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2 = 2 \sum_{i=1}^{n-1} (n-i)$$

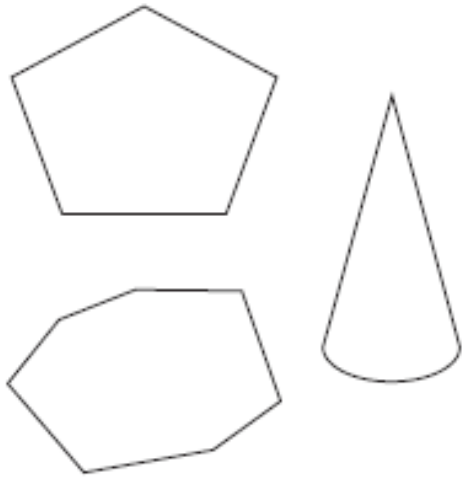
$$= 2[(n-1) + (n-2) + \cdots + 1] = (n-1)n \in \Theta(n^2)$$

Convex Hull Problem (1/10)

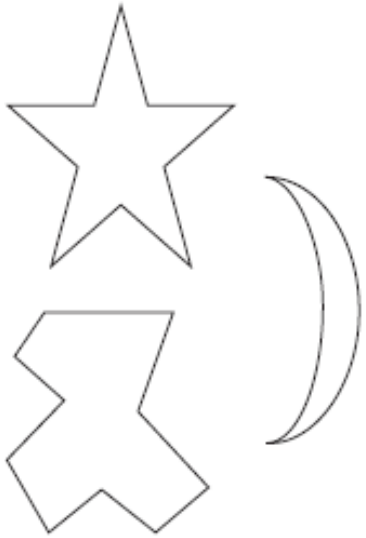
Ορισμός

Ένα σύνολο σημείων καλείται κυρτό όταν για δύο οποιαδήποτε σημεία p και q του συνόλου, ολόκληρη το ευθύγραμμο τμήμα με τελικά σημεία p και q ανήκει στο σύνολο.

Κυρτά σύνολα



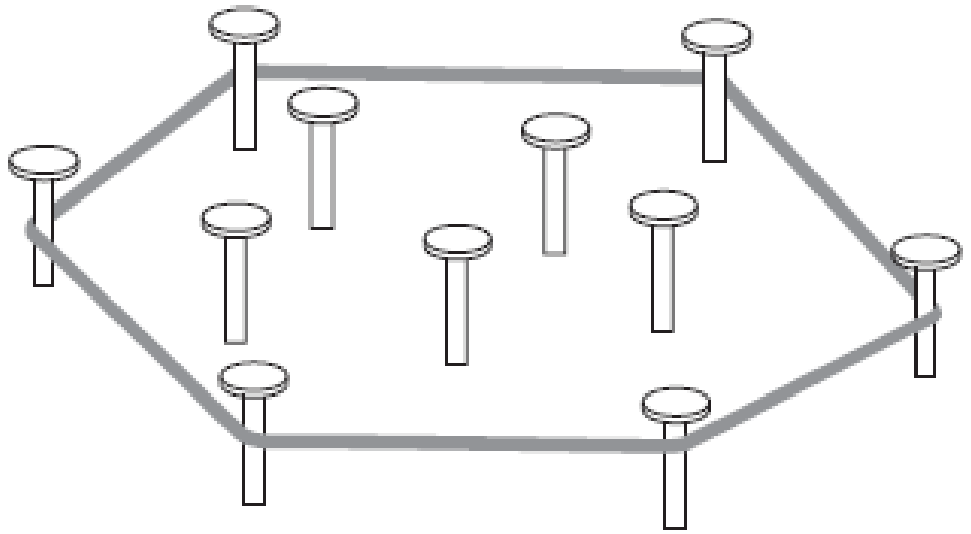
Μη κυρτά σύνολα



Convex Hull Problem (2/10)

Ορισμός

Το convex hull ενός συνόλου σημείων S είναι το μικρότερο κυρτό σύνολο που περιέχει το S



Convex Hull Problem (3/10)

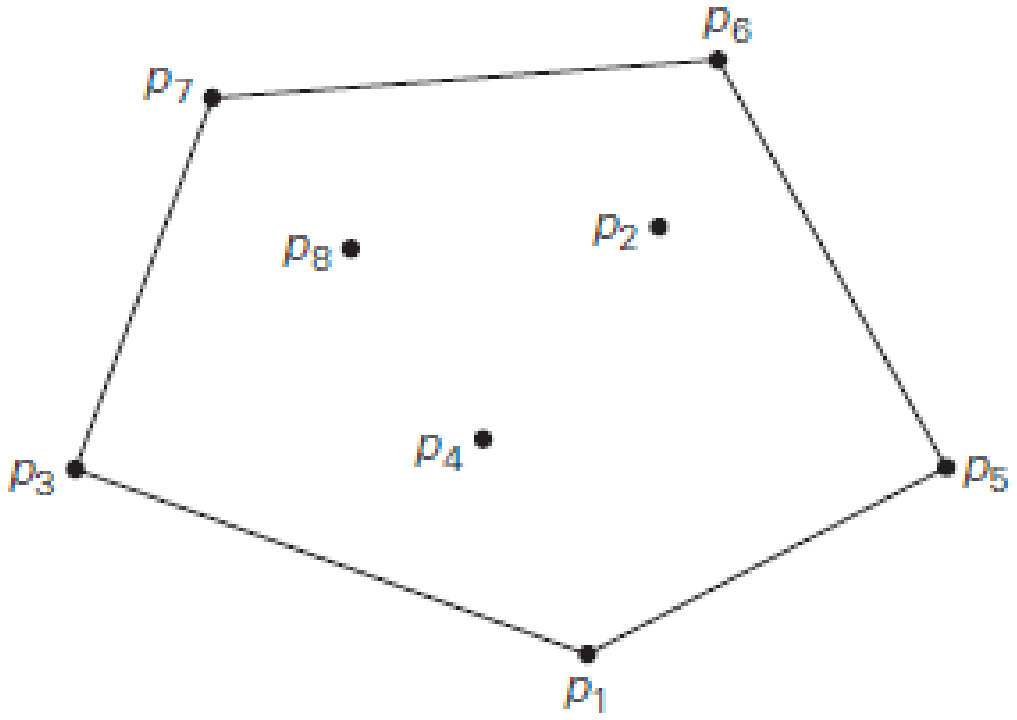
Αν το S είναι κυρτό τότε το convex hull είναι το ίδιο το S

Αν το S αποτελείται από δύο σημεία τότε το convex hull είναι η ευθεία που ενώνει τα δύο αυτά σημεία

Αν το S αποτελείται από τρία σημεία που δεν ανήκουν στην ίδια ευθεία, τότε το convex hull είναι το τρίγωνο με κορυφές στα τρία σημεία του S; Αν τα τρία σημεία ανήκουν στην ίδια ευθεία, τότε το convex hull είναι η ευθεία που έχει ως άκρες τα πιο απομακρυσμένα σημεία

Convex Hull Problem (4/10)

Το convex hull για περισσότερα σημεία φαίνεται στην ακόλουθη εικόνα



Convex Hull Problem (5/10)

Θεώρημα

Το convex hull ενός συνόλου σημείων S με πλήθος σημείων n > 2 τα οποία δεν ανήκουν όλα στην ίδια ευθεία γραμμή είναι ένα κυρτό πολύγωνο με κορυφές σε κάποια από τα σημεία του S. Αν όλα τα σημεία είναι στην ίδια γραμμή, το πολύγωνο εκφυλίζεται σε ένα ευθύγραμμο τμήμα αλλά με άκρες πάλι δύο σημεία του S.

Convex Hull Problem (6/10)

Το πρόβλημα είναι η κατασκευή του convex hull για ένα σύνολο σημείων S

Εύρεση των σημείων που θα αποτελέσουν τις κορυφές του πολυγώνου (extreme points)

Τα extreme points είναι σημεία τα οποία δεν βρίσκονται ενδιάμεσα σε ένα ευθύγραμμο τμήμα με τελικά σημεία στο S

Προσπαθούμε επίσης να βρούμε ποια σημεία στα άκρα του συνόλου S θα είναι αυτά που θα συνδεθούν μεταξύ τους

Convex Hull Problem (7/10)

Απλός αλγόριθμος εύρεσης convex hull (μη αποδοτικός όμως)

- Ένα ευθύγραμμο τμήμα που συνδέει δύο σημεία p_i και p_j ενός συνόλου n σημείων είναι τμήμα του convex hull όταν και μόνο όταν όλα τα υπόλοιπα σημεία του συνόλου βρίσκονται στην ίδια πλευρά της ευθείας των δύο σημείων
- Επαναλαμβάνουμε αυτό το σενάριο για όλα τα ζεύγη σημείων και βρίσκουμε τα ευθύγραμμα τμήματα που αποτελούν το όριο του convex hull
- Εύρεση ευθύγραμμου τμήματος

$$ax + by = c \qquad a = y_2 - y_1, b = x_1 - x_2, c = x_1 y_2 - y_1 x_2.$$

Convex Hull Problem (8/10)

- Ένα ευθύγραμμο τμήμα που συνδέει δύο σημεία χωρίζει το επίπεδο σε δύο τμήματα
- Στο ένα τμήμα ισχύει

$$ax + by \geq c$$

- Ενώ στο άλλο

$$ax + by < c$$

- Για τα σημεία πάνω στο ευθύγραμμο τμήμα ισχύει

$$ax + by = c$$

- Για να ελέγξουμε αν δυο σημεία βρίσκονται στο ίδιο τμήμα ελέγχουμε το πρόσημο της $ax + by - c$

Convex Hull Problem (9/10)

Επίδοση του αλγορίθμου: $O(n^3)$

Για κάθε ένα από τα $n(n-1)/2$ ζεύγη, πρέπει να βρούμε το πρόσημο της προηγούμενης παράστασης για κάθε ένα από τα υπόλοιπα $n-2$ σημεία

Αποδοτικότεροι αλγόριθμοι για την επίλυση του προβλήματος θα συζητηθούν στην πορεία του μαθήματος

Convex Hull Problem (10/10)

Πρόβλημα υπολογιστικής γεωμετρίας

Δεδομένης μιας λίστας n σημείων να βρεθεί ο μικρότερος κυρτός χώρος που περικλείει όλα τα σημεία (convex hull)

Παράδειγμα:

- Computer animation: η αντικατάσταση αντικειμένων από το κυρτό τους χώρο οδηγεί στην ταχύτατη αναγνώριση πιθανών συγκρούσεων
- Γεωγραφικά πληροφοριακά συστήματα
- Outliers detection σε στατιστικές μεθόδους

Exhaustive Search (1/4)

Πρόκειται για μια brute force τεχνική για συνδυαστικά προβλήματα

Οδηγεί στη δημιουργία όλων των στοιχείων ενός προβλήματος

Επιλέγει τα στοιχεία που ικανοποιούν κάποια κριτήρια

Τελικά, βρίσκει το επιθυμητό στοιχείο

Κλασικά προβλήματα:

- the Traveling Salesman Problem
- the Knapsack Problem
- the Assignment Problem

Exhaustive Search (2/4)

Στρατηγική:

- Απαριθμούμε όλες τις πιθανές υποψήφιες λύσεις
- Ελέγχουμε αν μια λύση ικανοποιεί τις συνθήκες του προβλήματος
- Αν απαιτείται, επιλέγουμε μια λύση από το σύνολο των εφικτών λύσεων

Πως θα ελέγξουμε όλες τις πιθανές λύσεις?

Exhaustive Search (3/4)

Βασικός αλγόριθμος

```
c ← generate a first candidate solution
while ( c is a candidate ) do
   if ( c is a valid solution )
   then output (c)
   c ← generate the next candidate solution, if any
```

Μπορεί να τερματιστεί όταν:

- Βρει την πρώτη εφικτή λύση
- Βρει ένα αριθμό αποδεκτών λύσεων
- Ελέγξει ένα αριθμό υποψήφιων λύσεων
- ‘Ξοδέψει’ μια ποσότητα χρόνου ή resources

Exhaustive Search (4/4)

Χαρακτηριστικά:

- Απλή τεχνική
- Πάντα βρίσκει μια λύση εφόσον υπάρχει
- Το κόστος είναι ανάλογο με το πλήθος των υποψήφιων λύσεων
- Εκτοξεύεται το πλήθος και ο χρόνος εκτέλεσης
- Είναι πρακτική μόνο για μικρού μεγέθους προβλήματα

Επιτάχυνση Υπολογισμών

Μείωση του χώρου των πιθανών λύσεων

- Heuristics / Analysis

Ανακατανομή του χώρου των λύσεων

- Χρήσιμη όταν ψάχνουμε μια λύση
- Ο αναμενόμενος χρόνος εξαρτάται από τη σειρά των υποψήφιων λύσεων
- Ελέγχουμε τις πιο πολλά υποσχόμενες λύσεις πρώτα

The Travelling Salesman Problem (1/4)

Πρέπει να βρεθεί το συντομότερο μονοπάτι ανάμεσα σε n πόλεις με μια επίσκεψη σε κάθε πόλη και με τελικό προορισμό την πόλη - αφετηρία

Μοντελοποιείται σαν γράφος με βάρη

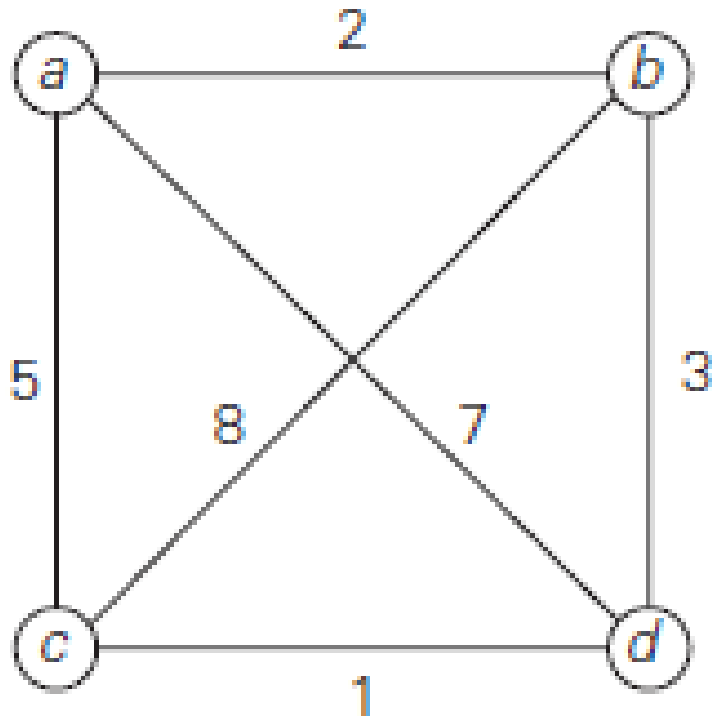
Οι κόμβοι είναι οι πόλεις και οι ακμές είναι οι διαδρομές

Τα βάρη είναι οι αποστάσεις των πόλεων

Μπορούμε να πάρουμε όλες τις διαδρομές των n-1 πόλεων, να υπολογίσουμε το μήκος των συνολικών διαδρομών και να βρούμε τη συντομότερη

The Travelling Salesman Problem (2/4)

Παράδειγμα



Tour	Length	
$a \rightarrow b \rightarrow c \rightarrow d \rightarrow a$	$l = 2 + 8 + 1 + 7 = 18$	
$a \rightarrow b \rightarrow d \rightarrow c \rightarrow a$	$l = 2 + 3 + 1 + 5 = 11$	optimal
$a \rightarrow c \rightarrow b \rightarrow d \rightarrow a$	$l = 5 + 8 + 3 + 7 = 23$	
$a \rightarrow c \rightarrow d \rightarrow b \rightarrow a$	$l = 5 + 1 + 3 + 2 = 11$	optimal
$a \rightarrow d \rightarrow b \rightarrow c \rightarrow a$	$l = 7 + 3 + 8 + 5 = 23$	
$a \rightarrow d \rightarrow c \rightarrow b \rightarrow a$	$l = 7 + 1 + 8 + 2 = 18$	

The Travelling Salesman Problem (3/4)

Εύρεση του συντομότερου Hamiltonian circuit του γράφου

- Ακολουθία από n + 1 γειτονικών κορυφών
- Η πρώτη είναι ίδια με την τελευταία

Απλός αλγόριθμος:

- Επέλεξε μια κορυφή ως αφετηρία
- Δημιούργησε (n-1)! permutations για τις ενδιάμεσες κορυφές
- Για κάθε κύκλο, υπολόγισε το συνολικό κόστος
- Υπολόγισε το μικρότερο κόστος

The Travelling Salesman Problem (4/4)

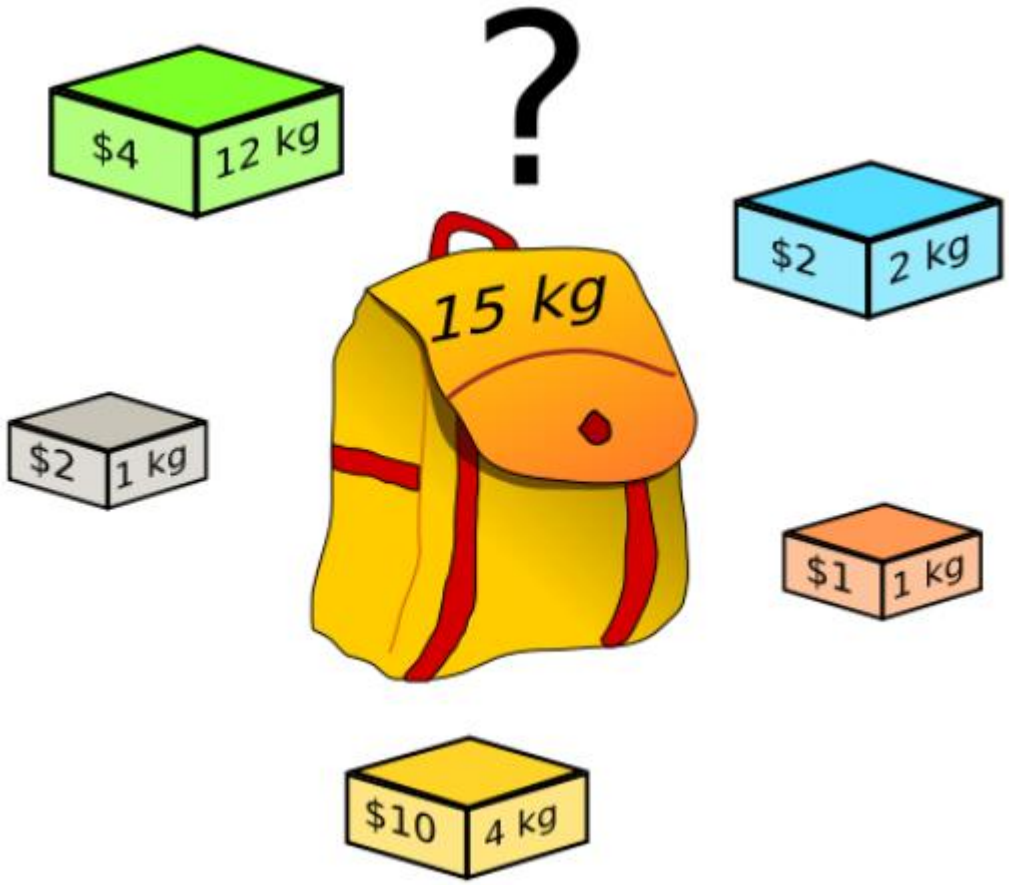
Επίδοση

- O(n!)
- Μπορεί να εφαρμοστεί μόνο για μικρού μεγέθους προβλήματα

The Knapsack Problem (1/4)

Δεδομένων n αντικειμένων με βάρη w_i , αξία $υ_i$ και ένα σακούλι χωρητικότητας W, να βρεθούν τα μεγαλύτερης αξίας αντικείμενα που χωράνε στο σακούλι.

Δημιουργούμε όλα τα υπο-σύνολα αντικειμένων, υπολογίζουμε το συνολικό βάρος τους, αναγνωρίζουμε τα υπο-σύνολα που χωράνε στο σακούλι και επιλέγουμε το υπο-σύνολο με τη μεγαλύτερη αξία



The Knapsack Problem (2/4)

Το πλήθος των υπο-συνόλων είναι 2^n

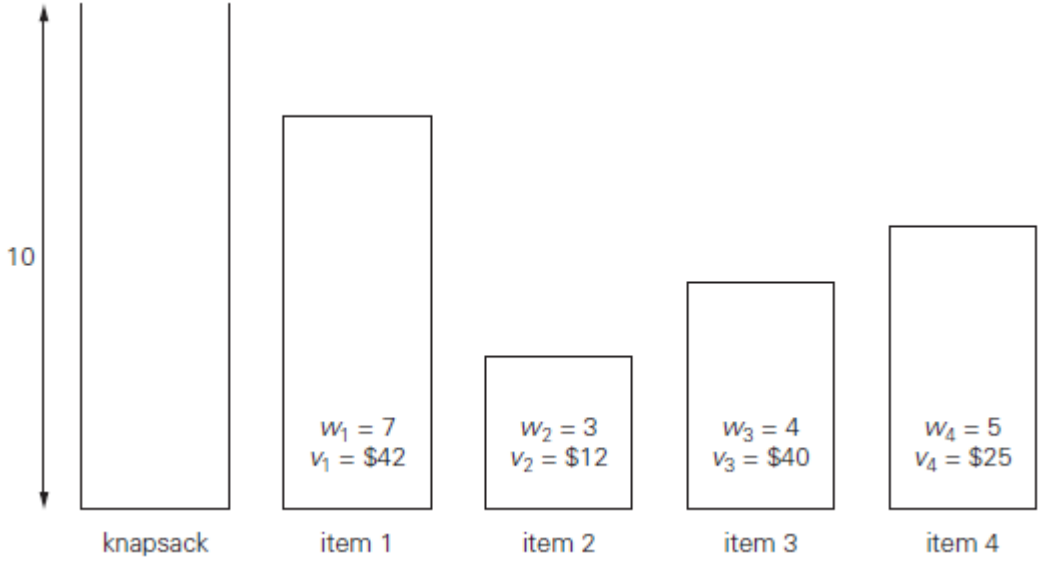
Πολυπλοκότητα: $\Omega(2^n)$

Μαζί με το traveling salesman problem ανήκουν στην κατηγορία των NP-hard προβλημάτων

Τα NP-hard προβλήματα δεν μπορούν να περιγραφούν από πολυωνυμικούς αλγορίθμους και θα συζητηθούν σε επόμενα μαθήματα

The Knapsack Problem (3/4)

Παράδειγμα:



Subset	Total weight	Total value
∅	0	$ 0
{1}	7	$42
{2}	3	$12
{3}	4	$40
{4}	5	$25
{1, 2}	10	$54
{1, 3}	11	not feasible
{1, 4}	12	not feasible
{2, 3}	7	$52
{2, 4}	8	$37
{3, 4}	**9**	**$65**
{1, 2, 3}	14	not feasible
{1, 2, 4}	15	not feasible
{1, 3, 4}	16	not feasible
{2, 3, 4}	12	not feasible
{1, 2, 3, 4}	19	not feasible

The Knapsack Problem (4/4)

Φορμαλισμός

$$\max \sum x_i \, v_i$$

$$\text{subject to} \quad \sum x_i \, w_i \leq W$$

$$\text{with} \quad x_i \text{ in } \{0, 1\}$$

The Assignment Problem (1/5)

Υπάρχουν n άνθρωποι στους οποίους πρέπει να ανατεθούν n εργασίες. Το κόστος ανάθεσης μιας εργασίας σε κάποιον είναι C[i,j] για κάθε ζεύγος. Να βρεθεί η λύση με το μικρότερο συνολικό κόστος.

	Job 1	**Job 2**	**Job 3**	**Job 4**
Person 1	9	2	7	8
Person 2	6	4	3	7
Person 3	5	8	1	8
Person 4	7	6	9	4

The Assignment Problem (2/5)

Ο πίνακας κόστους καθορίζει την τελική λύση

Οι πιθανές λύσεις περιγράφονται με tuples $\langle j_1, \ldots, j_n \rangle$

Στην i θέση περιγράφεται η στήλη του στοιχείου που έχει επιλεγεί στην i γραμμή

Το tuple <2, 3, 4, 1> υποδηλώνει ότι στο άτομο 1 θα ανατεθεί η εργασία 2, στο άτομο 2 η εργασία 3, στο άτομο 3 η εργασία 4, κ.ο.κ.

	Job 1	Job 2	Job 3	Job 4
Person 1	9	2	7	8
Person 2	6	4	3	7
Person 3	5	8	1	8
Person 4	7	6	9	4

The Assignment Problem (3/5)

Παράδειγμα

$$C = \begin{bmatrix} 9 & 2 & 7 & 8 \\ 6 & 4 & 3 & 7 \\ 5 & 8 & 1 & 8 \\ 7 & 6 & 9 & 4 \end{bmatrix}$$

<1, 2, 3, 4> cost = 9 + 4 + 1 + 4 = 18
<1, 2, 4, 3> cost = 9 + 4 + 8 + 9 = 30
<1, 3, 2, 4> cost = 9 + 3 + 8 + 4 = 24
<1, 3, 4, 2> cost = 9 + 3 + 8 + 6 = 26
<1, 4, 2, 3> cost = 9 + 7 + 8 + 9 = 33
<1, 4, 3, 2> cost = 9 + 7 + 1 + 6 = 23

etc.

The Assignment Problem (4/5)

Φορμαλισμός

$$\min \sum \sum x[i,j]\, c[i,j]$$

subject to

$$\sum x[i,j] = 1, \text{ for every } i$$

$$\sum x[i,j] = 1, \text{ for every } j$$

with

$$x[i,j] \text{ in } \{0, 1\}$$

The Assignment Problem (5/5)

Απλός αλγόριθμος:

- Generate all permutations
- Compute the total cost for each permutation
- Find the smallest cost

Μια και το πλήθος των αντιμεταθέσεων (permutations) είναι n!, η μέθοδος exhaustive search δεν είναι καθόλου πρακτική

Πολυπλοκότητα Ω(n!)

Υπάρχει πιο αποδοτικός αλγόριθμος: Hungarian method

The Hungarian Method (1/2)

[1]. Identify least element in row, subtract value of element from all elements in row. (This creates at least one zero). Repeat for all rows.

[2]. If a column contains more than one zero (this implies that two objects can be assigned at equal weight, and one assignment has no determined lowest value), repeat [1] for all columns.

[3]. Select elements in columns for which a distinct minimum weight has been determined and add to solution.

[4]. If full solution has not been achieved (implying indeterminate assignments still present), flag rows without solutions. Flag all columns in flagged rows that contain a zero. Flag all rows with a previously determined solution in previously flagged columns.

[5]. From elements remaining in UNFLAGGED columns and FLAGGED rows, determine least element. Subtract from every unflagged element, and add to every element that has been flagged twice.

[6]. Repeat [3] – [5] until full solution has been achieved.

The Hungarian Method (2/2)

Επίδοση

- Αρχικά $O(n^4)$
- Μετά από βελτιώσεις $O(n^3)$