Τεχνικές Σχεδιασμού Αλγορίθμων

- Διαίρει και Βασίλευε
- Δυναμικός Προγραμματισμός
- Απληστία

Διαίρει και Βασίλευε

- Βασικά Βήματα
  - **Διαίρει:** Κατάτμηση του αρχικού προβλήματος σε *b*, *όσο το δυνατόν, ιδίου μεγέθους* υποπροβλήματα
  - **Βασίλευε:** Επίλυση των υποπροβλήματων με αναδρομικό τρόπο
  - **Συνδύασε:** Συνδυασμός των επιμέρους λύσεων, εντός πολυωνυμικού χρόνου, για επίτευξη της ολικής λύσης
- Έκφραση της πολυπλοκότητας μέσω αναδρομικών σχέσεων

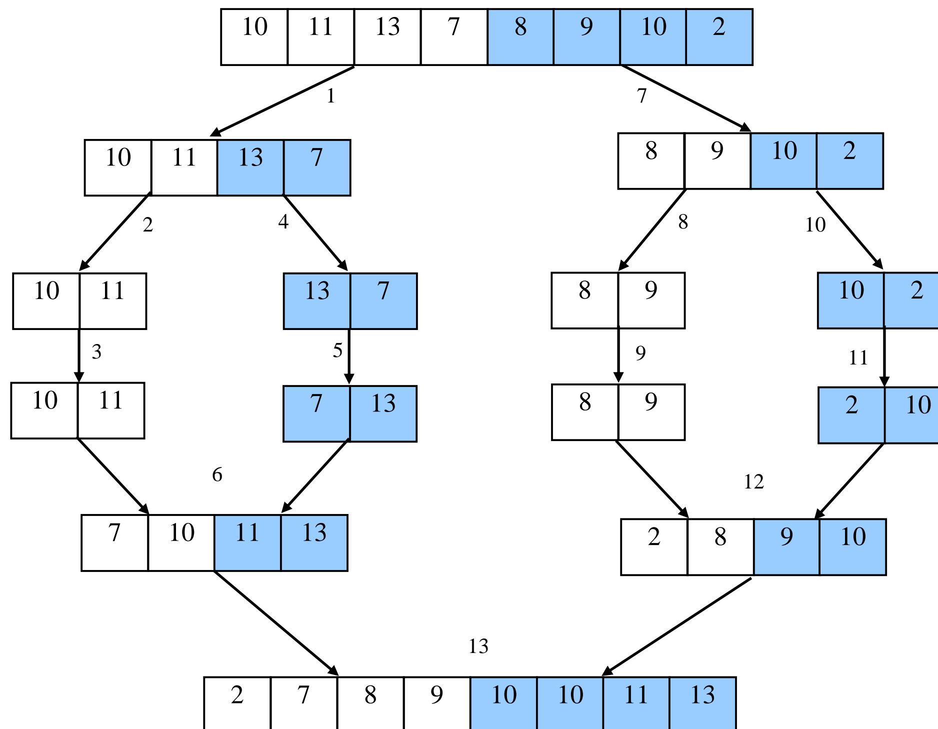
Mergesort

- Κατάτμηση σε δύο ίσα τμήματα A_1 και A_2
- Αναδρομική ταξινόμηση των A_1 και A_2
  - Κόστος: $T(\lceil n/2 \rceil)+T(\lfloor n/2 \rfloor)$
- Συγχώνευση των ταξινομημένων τμημάτων A_1 και A_2 μέσω ενός βοηθητικού πίνακα B
  - Κόστος: cn, c σταθερά

- Πολυπλοκότητα

$$T(n) = 2T(n/2)+cn = \Theta(n\log n)$$

Γενική μέθοδος: *β' περίπτωση*, σχηματίζουμε το $n^{\log_b a}$ με $a=2$, $b=2$, και $f(n)=\Theta(n)=cn^{\log_2 2}$



Πολλαπλασιασμός Ακεραίων

- Δίχως βλάβη της γενικότητας, έστω A, B δύο ακέραιοι, μήκους $n=2^m$
- Η πρόσθεση $A+B$ και η αφαίρεσή τους $A-B$ είναι πράξεις γραμμικές στο πλήθος n των μπιτ
- Ο πολλαπλασιασμός τους, με την απλοϊκή μέθοδο, κοστίζει $O(n^2)$
- Καθώς υπάρχει ανάγκη για γρήγορη τέλεση του (π.χ., κρυπτογραφία), υπάρχει καλύτερη λύση;

Πρώτη Προσέγγιση

- Εφαρμογή της τεχνικής διαίρει και βασίλευε:

$$A = A_{n-1}A_{n-2} \cdots A_{n/2}\, A_{n/2-1} \cdots A_1 A_0 = A_L 2^{n/2} + A_R$$

$$B = B_{n-1}B_{n-2} \cdots B_{n/2}\, B_{n/2-1} \cdots B_1 B_0 = B_L 2^{n/2} + B_R$$

$$AB = (A_L 2^{n/2} + A_R)(B_L 2^{n/2} + B_R) = A_L B_L 2^n + A_L B_R 2^{n/2} + A_R B_L 2^{n/2} + A_R B_R$$

- Πολυπλοκότητα

$$T(n) = 4T(n/2) + cn = \Theta(n^{\log_2 4}) = \Theta(n^2)$$

  - 4 αναδρομικοί πολ/σμοί «μισού» μεγέθους, 3 αριστερές ολισθήσεις και 3 προσθέσεις γραμμικού κόστους
  - Γενική Μέθοδος: *α' περίπτωση* με $a=4$, $b=2$ και $f(n)=cn=O(n^{\log_2 4-\varepsilon})$

Δεύτερη Προσέγγιση

- Προκειμένου να επιτευχθεί καλύτερη πολυπλοκότητα, πρέπει να μειωθεί το πλήθος των αναδρομικών κλήσεων:

$$AB = (A_L 2^{n/2} + A_R)(B_L 2^{n/2} + B_R) =$$
$$A_L B_L 2^n + A_L B_R 2^{n/2} + A_R B_L 2^{n/2} + A_R B_R =$$
$$A_L B_L 2^n + (A_L B_R + A_R B_L) 2^{n/2} + A_R B_R =$$
$$A_L B_L 2^n + (\underline{A_L B_R + A_R B_L} + A_L B_L + A_R B_R \underline{- A_L B_L - A_R B_R}) 2^{n/2} + A_R B_R =$$
$$A_L B_L 2^n + ((A_L - A_R)(B_R - B_L) + A_L B_L + A_R B_R)\, 2^{n/2} + A_R B_R$$

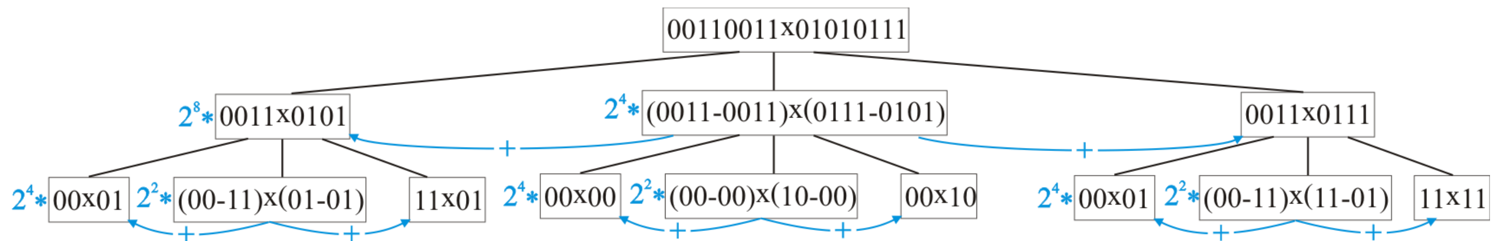
- Πολυπλοκότητα:

$$T(n) = 3T(n/2) + cn = \Theta(n^{\log_2 3}) = \Theta(n^{1.59})$$

  - 3 αναδρομικοί πολ/σμοί «μισού» μεγέθους, 2 αριστερές ολισθήσεις και 6 προσθέσεις/αφαιρέσεις γραμμικού κόστους
  - Γενική Μέθοδος: *α' περίπτωση* με $a=3$, $b=2$ και $f(n)=cn=O(n^{\log_2 3-\varepsilon})$

Σημ. Είναι δυνατόν να επιλυθεί εντός χρόνου $\Theta(n\log n)$, με χρήση Fourier

Παράδειγμα



Δυναμικός Προγραμματισμός

- Εφαρμόζεται σε ***προβλήματα βελτιστοποιήσεως*** (***optimization problems***)**:**

  «*Από ένα σύνολο* ***αποδεκτών λύσεων****, όπου κάθε μία σχετίζεται με κάποιο* ***κόστος****, αιτείται η εύρεση της* ***βέλτιστης*** *(****μεγίστου ή ελαχίστου κόστους****) βάσει ενός συνόλου* ***περιορισμών***»

Δυναμικός Προγραμματισμός (συν.)

- Ομοιότητα με ***Διαίρει και Βασίλευε***

  Διάσπαση αρχικού προβλήματος σε μικρότερου μεγέθους υποπροβλήματα, τα οποία επιλύονται αναδρομικά
- Διαφορά με ***Διαίρει και Βασίλευε***

  Τα υποπροβλήματα δεν είναι κατ᾽ ανάγκη ανεξάρτητα μεταξύ τους, αλλά είναι δυνατόν να διαμοιράζονται κοινά – μικρότερα υποπροβλήματα.
- Το τελευταίο επιβάλλει την επίλυση κάθε υποπροβλήματος άπαξ και την αποθήκευση της λύσεως, για συνακόλουθη χρήση, σε έναν πίνακα. Έτσι προκύπτει το όνομα ***Δυναμικός Προγραμματισμός...***

Δυναμικός Προγραμματισμός (συν.)

- Τυπικά χαρακτηριστικά:
  - **Απλά υποπροβλήματα:** *το πρόβλημα διασπάται σε ένα σύνολο υποπροβλημάτων, συνήθως με περισσότερους του ενός τρόπους (σε αντιδιαστολή με το «διαίρει και βασίλευε»)*
  - **Αρχή του βέλτιστου:** *η ολική βέλτιστη λύση είναι αποτέλεσμα συνδυασμού βέλτιστων λύσεων υποπροβλημάτων*
  - **Επικάλυψη υποπροβλημάτων:** *καθώς φαινομενικά ασυσχέτιστα υποπροβλήματα είναι δυνατόν να διαμοιράζονται βέλτιστες λύσεις κοινών υποπροβλημάτων, κάθε βέλτιστη λύση αποθηκεύεται για μελλοντική ενδεχόμενη χρήση-το πολύ πολυωνυμικές, στο πλήθος, τέτοιες λύσεις*

Πολλαπλασιασμός Πινάκων

- Δοθείσης μίας ακολουθίας πινάκων:

$$A_0, A_1, \ldots, A_{n-1},$$

αιτείται να βρεθεί ο φθηνότερος, από απόψεως υπολογισμών, τρόπος υπολογισμού του γινομένου:

$$A_0 \cdot A_1 \cdot \; \cdot \; \cdot A_{n-1}.$$

- Εάν ο A_i είναι $d_i \times d_{i+1}$, οπότε το κόστος του γινομένου $A_i \cdot A_{i+1}$ είναι $d_i \cdot d_{i+1} \cdot d_{i+2}$ (γιατί: $d_i \{ \underbrace{[\;]}_{d_{i+1}} \overbrace{[\;]}^{d_{i+2}} \}^{d_{i+1}}$)

Πολλαπλασιασμός Πινάκων (συν.)

- Π.χ., για τρεις πίνακες διαστάσεων 5×4, 4×6, 6×2 έχουμε δύο διαφορετικούς τρόπους τοποθετήσεως των παρενθέσεων:

  $A_0 \cdot (A_1 \cdot A_2)$, κόστους $4 \cdot 6 \cdot 2 + 5 \cdot 4 \cdot 2 = 88$, και

  $(A_0 \cdot A_1) \cdot A_2$, κόστους $5 \cdot 4 \cdot 6 + 5 \cdot 6 \cdot 2 = 180$

Επίλυση

- Πρώτη προσπάθεια (brute force) με αναδρομή:
  - Θα εξετάσουμε όλες τις δυνατές τοποθετήσεις
  - Υπάρχουν n-1 θέσεις για να τοποθετήσουμε τις παρενθέσεις:

$$(A_0 \cdot A_1 \cdot \cdot \cdot A_{k-1}) \cdot (A_k A_{k+1} \cdot \cdot \cdot A_{n-1})$$

  - Το # παραγόμενων διασπάσεων δίδεται από την αναδρομική εξίσωση:

$$\Pi(n) = \sum_{1 \le k \le n} \Pi(k) \cdot \Pi(n-k)$$

Λύση: $C(n\text{-}1) = \Omega(4^n / n^{1.5})$ ***ΑΠΟΓΟΡΕΥΤΙΚΑ ΜΕΓΑΛΗ***

Επίλυση (συν.)

- Η σωστή αντιμετώπιση:

  Έστω $A_{i,j} = A_i \cdot A_{i+1} \cdot \cdot \cdot A_j$, με κόστος $M_{i,j}$.

  Τότε: $A_{0,n\text{-}1} = A_{0,k\text{-}1} \cdot A_{k,n\text{-}1}$

  και: $M_{0,n\text{-}1} = M_{0,k\text{-}1} + M_{k,n\text{-}1} + d_0 \cdot d_k \cdot d_n$
- Ισχύουν οι προϋποθέσεις του Δυναμικού Προγραμματισμού:
  - Τα $M_{0,k\text{-}1}$ και $M_{k,n\text{-}1}$ πρέπει είναι βέλτιστα λυμένα: *εάν τα υποπροβλήματα δεν λύνονταν βέλτιστα, ενώ η ολική λύση $M_{0,n\text{-}1}$ ήταν βέλτιστη, τότε με «τοπική αντικατάσταση» με τις βέλτιστες λύσεις, θα προέκυπτε καλύτερη λύση!*

Επίλυση (συν.)

- Γενικό βήμα:
  Εύρεση του βέλτιστου *k, αναδρομικά,* αποθηκεύοντας τα ενδιάμεσα αποτελέσματα.
- Επίλυση του $A_{i,j}$ με κόστος:

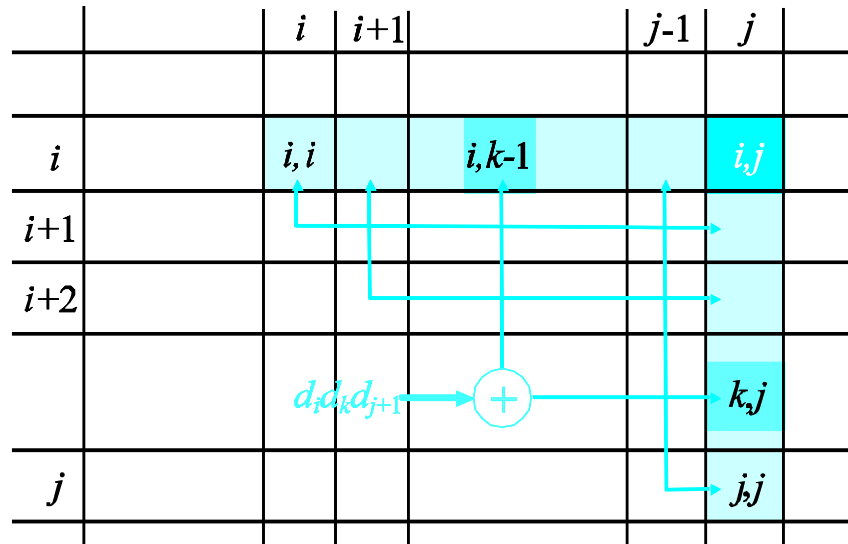
$$M_{i,j} = \begin{cases} \min_{i<k\leq j}\{M_{i,k-1} + M_{k,j} + d_i \cdot d_k \cdot d_j\} & i < j \\ 0 & i = j \end{cases}$$

- Από τον ανωτέρω τύπο φαίνεται πως πρέπει να είναι διαθέσιμες οι βέλτιστες λύσεις $M_{i,k-1}$ και $M_{k,j}$ όταν θα ασχοληθούμε με τον υπολογισμό του $M_{i,j}$
- Άρα, πρέπει να υπολογιστεί ***από κάτω προς τα πάνω. Δηλαδή, πρώτα όλες οι "αλυσίδες" μήκους*** **1*****, μετά οι μήκους*** **2** ***κοκ.***
- Εάν l είναι το τρέχον μήκος αλυσίδας, τότε πρέπει να ισχύει (γιατί;):

$$j\text{-}i+1 = l \Rightarrow j = l+i-1 \quad \text{και} \quad i \leq n-l$$

Επίλυση (συν.)

- Σχηματικά, έχουμε:



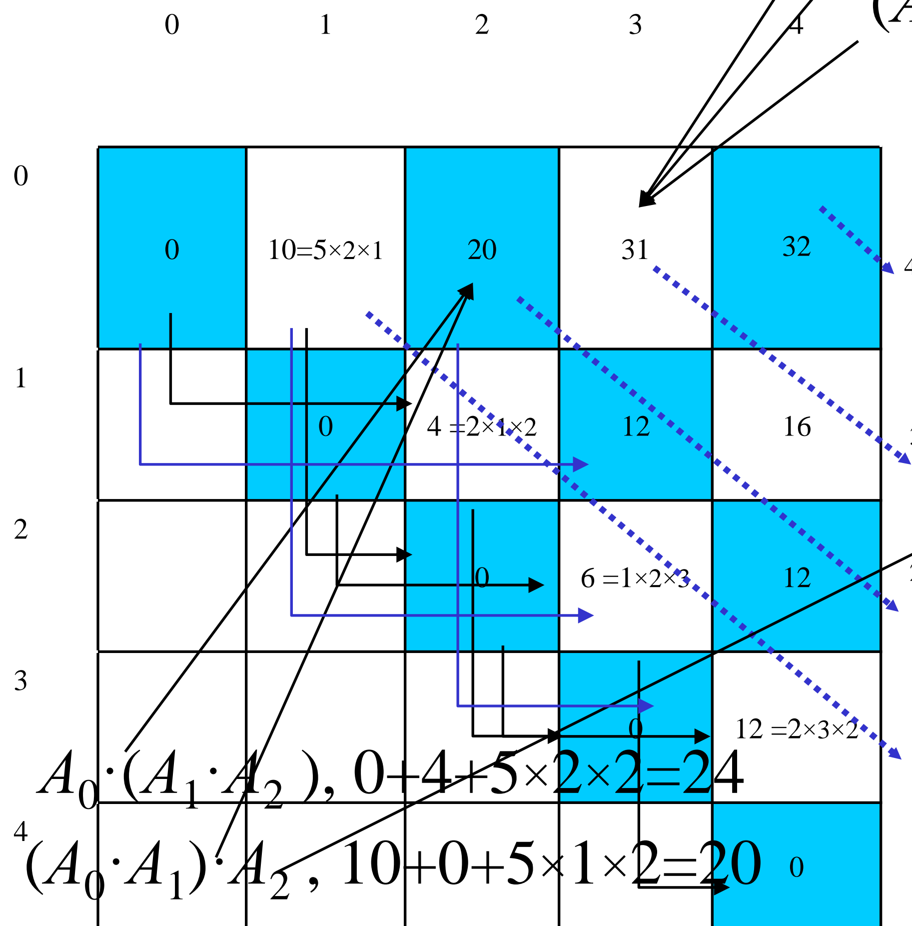
Algorithm chainMatrixMultiplication(**int** []d)

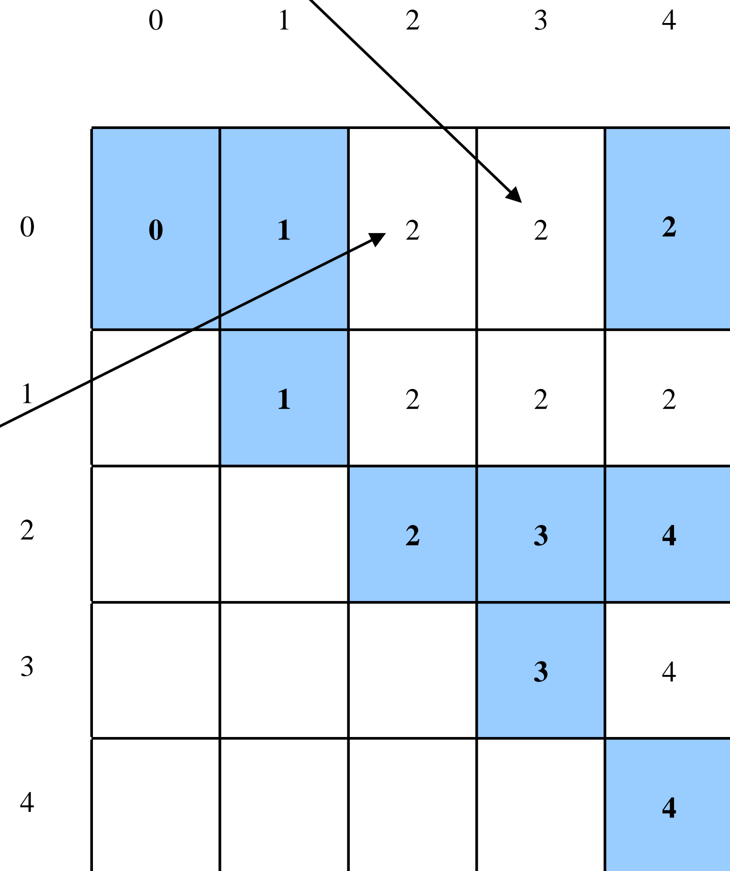
```
1.  for (i = 0; i < n; i++)
2.     M[i][i]=0;
3.  for (l = 2; l < n+1; l++){
4.    for (i = 0; i < n-l+1; i++){
5.      j = i+l-1;
6.      M[i][j] = infty;   // αρχικοποίηση σε άπειρη τιμή
7.      for (k = i+1; k <= j; k++){
8.        temp = M[i][k-1]+M[k][j]+d[i]*d[k]*d[j+1];
9.        if (temp < M[i][j]){
10.          M[i][j] = temp;
11.          S[i][j] = k;
12.        }
13.     }
14.   }
15. }
16. return M[0][n-1], S;
```

- Χρόνος $O(n^3)$

Παράδειγμα για πίνακες διαστάσεως
5×2, 2×1, 1×2, 2×3, 3×2 ,
Λύση: $(A_0 \cdot A_1)\cdot((A_2 \cdot A_3)\cdot A_4)$

$A_0\cdot(A_1\cdot A_2\cdot A_3)$, 0+12+5×2×3=32
$(A_0\cdot A_1)\cdot(A_2\cdot A_3)$, 10+6+5×1×3=31
$(A_0\cdot A_1\cdot A_2)\cdot A_3$, 20+0+5×2×3=50





Σακκίδιο 0-1

- Δοθέντος ενός συνόλου n αντικειμένων $S=\{a_1,a_2,\ldots,a_n\}$, με (ακέραια) βάρη $\{w_1,w_2,\ldots,w_n\}$ και αξίες $\{v_1,v_2,\ldots,v_n\}$, αιτείται η εύρεση του υποσυνόλου $S' \subseteq S$ ώστε:
  - η αξία τους Σv_j να μεγιστοποιείται
  - ενώ το βάρος τους Σw_j να μην ξεπερνά μία τιμή W

Σακκίδιο 0-1(συν.)

- Η ονομασία προέρχεται από το γεγονός ότι τα αντικείμενα επιλέγονται ακέραια.
- Έχει εφαρμογές στην καθημερινότητα, π.χ., μεταφορικές εταιρίες
- Η λύση σχηματισμού όλων των δυνατών υποσυνόλων και επιλογής του βέλτιστου έχει κόστος $O(2^n)$

Σακκίδιο 0-1(συν.)

- Λύση δυναμικού προγραμματισμού με δύο παραμέτρους. Έστω :
  - $S_k = \{a_1, a_2, \ldots, a_k\}$ και
  - $S_{k,w} \subseteq S_k$ το υποσύνολό του, βάρους w με μέγιστη ωφέλεια $V_{k,w}$

  Τότε, λύση:
  - $S_{n,w}$, $w \leq W$, με μέγιστη ωφέλεια $V_{n,w}$
  - Ολική λύση: $v_{\max} = \max_{w \leq W} \{V_{n,w}\}$

Σακκίδιο 0-1(συν.)

- Αντίστοιχη σχέση, $w \leq W$:

$$V_{k,w} = \begin{cases} 0 & k = 0 \\ V_{k\text{-}1,w} & w < w_k \\ \max\{V_{k\text{-}1,w}, V_{k\text{-}1,w\text{-}w_k} + v_k\} & \text{αλλιώς} \end{cases}$$

Δηλαδή, είτε τα πρώτα k-1 είτε τα πρώτα k, *εφ' όσον το k-στό αντικείμενο χωρά*

Algorithm knapSack01(item[] S, **int** W)

```
1.  for (j = 0; j <= W; j++)
2.     V[0][j]=0;
3.  for (k=1; k<S.length; k++){
4.    for (w=0; j<=W; j++){
5.      withoutk = V[k-1][w];
6.      if (w >= S[k].w)
7.         withk = V[k-1][w-S[k].w]+S[k].v;
8.      else withk = -INFTY;
9.      V[k][w] = max(withk,withoutk);
10.     A[k][w] = (withk < withoutk ? 0 : 1);
11.   }
12. }
13. return V[S.length,W], A;
```

- Χρόνος O(nW): **ΨΕΥΔΟΠΟΛΥΩΝΥΜΙΚΟΣ**

Παράδειγμα

- S = {(3, 2), (5, 3), (7, 3), (4, 4), (3, 4), (9, 5), (2, 7), (11, 8), (5, 8)} –(βάρος, αξία)– και W= 15
- Βέλτιστη λύση: {(4, 4), (3, 4), (2, 7), (5, 8)} βάρους 14 και αξίας 23

$k\backslash w$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	0	0	2	2	2	2	2	2	2	2	2	2	2	2	2
2	0	0	0	2	2	3	3	3	5	5	5	5	5	5	5	5
3	0	0	0	2	2	3	3	3	5	5	5	5	6	6	6	8
4	0	0	0	2	4	4	4	6	6	7	7	7	9	9	9	9
5	0	0	0	4	4	4	6	8	8	8	10	10	11	11	11	13
6	0	0	0	4	4	4	6	8	8	8	10	10	11	11	11	13
7	0	0	7	7	7	11	11	11	13	15	15	15	17	17	18	18
8	0	0	7	7	7	11	11	11	13	15	15	15	17	17	18	18
9	0	0	7	7	7	11	11	15	15	15	19	19	19	21	23	23

$k\backslash w$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
2	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1
3	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
4	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
5	0	0	0	1	0	0	1	1	1	1	1	1	1	1	1	1
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	1	1	0	1	1	1	1	1	1

Απληστία

- Υπάρχουν περιπτώσεις που ο Δυναμικός Προγραμματισμός δίδει «ακριβές» λύσεις
- Για ορισμένα προβλήματα, είναι δυνατόν μία και μόνο επιλογή, αυτή που, *άπληστα* ή *με τελείως τοπικά κριτήρια*, κάποιος διαλέγει να οδηγεί σε βέλτιστη λύση
- Καθημερινότητα: τρόπος που δίνουμε ρέστα

Απληστία (συν.)

- Τα γνωρίσματα ενός προβλήματος άπληστα επιλύσιμου:
  - **Ιδιότητα Άπληστης Επιλογής:** *Πάντοτε* ο διαχωρισμός σε υποπροβλήματα, κατά άπληστο τρόπο, ανήκει σε όλους τους δυνατούς διαχωρισμούς που δίδουν βέλτιστες λύσεις. Δηλαδή, *αρκεί μόνο μία κατάτμηση του προβλήματος με τοπικά κριτήρια*
  - **Βέλτιστη Υποδομή:** Η άπληστη επιλογή, μαζί με την βέλτιστη λύση του μοναδικού παραγόμενου υποπροβλήματος, οδηγεί σε ολικά βέλτιστη λύση

Απληστία (συν.)

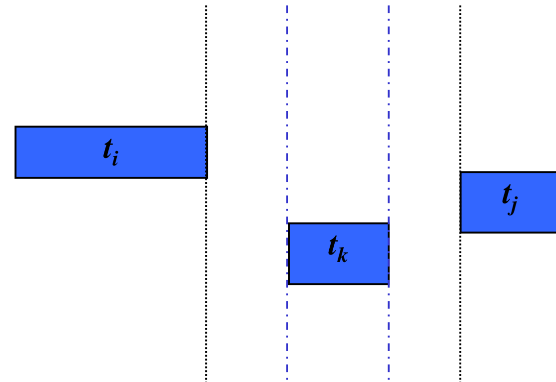
- Δυστυχώς, λίγα είναι τα προβλήματα που επιδέχονται άπληστης λύσεως. Θα εξετάσουμε δύο από αυτά:
  - Προγραμματισμός Εργασιών
  - Κλασματικό σακκίδιο

Απληστία (συν.)

- Προγραμματισμός Εργασιών
  - Έστω σύνολο εργασιών $T=\{t_0,t_1,\ldots,t_{n\text{-}1}\}$ όπου η t_i χαρακτηρίζεται από τον χρόνο ενάρξεως s_i και τον χρόνο περατώσεως f_i. Εάν έχουμε στην διάθεσή μας μονοεπεξεργαστική μηχανή, αιτείται η δρομολόγηση του μέγιστου δυνατού πλήθους εργασιών
  - Υπάρχουν προβλήματα της καθημερινότητας που έχουν την ανωτέρω αφαίρεση. Π.χ., προγραμματισμός μαθημάτων σε μία αίθουσα

Απληστία (συν.)

- Λύση Δυναμικού Προγραμματισμού:
  - Ταξινομούμε τις εργασίες κατά αύξουσα τιμή χρόνου περατώσεως
  - $T_{i,j} = \{t_k \in T: f_i \leq s_k < f_k \leq s_j\}$
  - $T_{i,j} = T_{i,k} \cup \{t_k\} \cup T_{k,j}$
  - $M_{i,j} = \max_{i<k<j} \{M_{i,k} + M_{k,j} + 1\}$, $T_{i,j} \neq \emptyset$
  - Δύο τεχνητές διεργασίες:
    - T_{-1}, με $f_{-1} = 0$
    - T_n, με $s_n = \infty$
  - Αρκεί να βρεθεί το $T_{-1,n}$ πληθαρίθμου $M_{-1,n}$



Κόστος $O(n^3)$ (ομοιάζει με την σχέση της αλυσίδας πινάκων)

Απληστία (συν.)

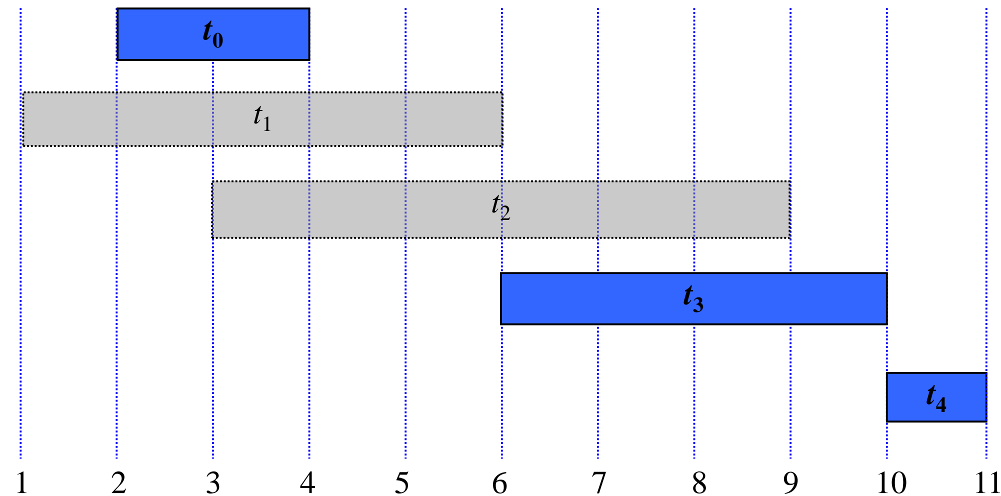
- Όμως...
  - Πάντοτε η διάσπαση που επιλέγει την διεργασία t_m με τον μικρότερο χρόνο περατώσεως οδηγεί σε βέλτιστη λύση:
    - Η πρώτη εργασία t_a κάθε βέλτιστου διαχωρισμού ***πάντοτε*** μπορεί να αντικατασταθεί από την t_m, καθώς σίγουρα δεν δημιουργεί συγκρούσεις

t_m t_a

  - Επιπλέον, το $T_{i,m}$ είναι κενό –αλλιώς, θα υπήρχε άλλη διεργασία με μικρότερο χρόνο περατώσεως
  - Άρα, παράγεται μόνο ένα υποπρόβλημα!

Απληστία (συν.)

- Παράδειγμα...



Πολυπλοκότητα; $O(n\log n)$ για την ταξινόμηση των διεργασιών βάσει χρόνου περατώσεως...

Απληστία (συν.)

- Κλασματικό Σακκίδιο
  - Παραλλαγή του σακκιδίου 0-1, όπου, όμως, υπάρχει η δυνατότητα χρησιμοποιήσεως ***τμήματος*** ή ***κλάσματος*** από κάθε αντικείμενο
  - Λύση:
    - Τα αντικείμενα ταξινομούνται κατά φθίνοντα λόγο αξίας προς βάρος

      $r_i = v_i / w_i$
    - Κατόπιν, τα λαμβάνουμε, κατά αυτήν την σειρά, καθ' ολοκληρίαν, όσο είναι δυνατόν
    - Στην συνέχεια, από το τελευταίο παίρνουμε όση ποσότητα περισσεύει από το W
    - Έτσι έχουμε βέλτιστη λύση!

Παράδειγμα

- $S = \{a_1 = (3, 2), a_2 = (5, 3), a_3 = (7, 3), a_4 = (4, 4), a_5 = (3, 4), a_6 = (9, 5), a_7 = (2, 7), a_8 = (11, 8), a_9 = (5, 8)\}$ –(βάρος, αξία)– και $W = 15$
- $r_1 = 2/3 = 0.67$, $r_2 = 3/5 = 0.6$, $r_3 = 3/7 = 0.43$, $r_4 = 4/4 = 1$, $r_5 = 4/3 = 1.33$, $r_6 = 5/9 = 0.56$, $r_7 = 7/2 = 3.5$, $r_8 = 8/11 = 0.73$, $r_9 = 8/5 = 1.6$
- Ταξινομούμε τους λόγους σε φθίνουσα διάταξη
  $r_7 = 7/2 = 3.5$, $r_9 = 8/5 = 1.6$, $r_5 = 4/3 = 1.33$, $r_4 = 4/4 = 1$, $r_8 = 8/11 = 0.73$, $r_1 = 2/3 = 0.67$, $r_2 = 3/5 = 0.6$, $r_6 = 5/9 = 0.56$, $r_3 = 3/7 = 0.43$
- Παίρνουμε τα αντικείμενα κατά φθίνοντα λόγο
  $a_7 = (2, 7)$, $a_9 = (5, 8)$, $a_5 = (3, 4)$, $a_4 = (4, 4)$, $a_8 = (1, 1{\times}0.73)$ (από το a_8 παίρνουμε κλάσμα)
- Η βέλτιστη λύση είναι αξίας 23.73 και βάρους 15

Απληστία (συν.)

– Γιατί δουλεύει;
  - Έστω πως, σε κάποιο βήμα i, δεν λήφθηκε το μεγαλύτερο, ως προς τον λόγο αξία ανά βάρος, αντικείμενο a_k, αλλά κάποιο άλλο μικρότερο a_l
  - Τότε, στην ολική λύση, είναι δυνατή η αντικατάσταση του a_l, κατά την ποσότητα:

    $\min\{w_k - f_k, f_l\}$,

    από το a_k
  - Συνεπώς, αυξάνεται η αξία της βέλτιστης λύσεως (άτοπον)

Πολυπλοκότητα $O(n\log n)$