

Παρουσίαση για χρήση με το σύγγραμμα, **Αλγόριθμοι Σχεδίαση και Εφαρμογές**, των Μ. Τ. Goodrich and R. Tamassia, Wiley, 2015 (στα ελληνικά από εκδόσεις Μ. Γκιούρδας)

Δυναμικός προγραμματισμός: Πρόβλημα σακιδίου 0-1



Civil War Knapsack. U.S. government image. Vicksburg National Military Park. Public domain.

The Πρόβλημα σακιδίου 0-1 Problem



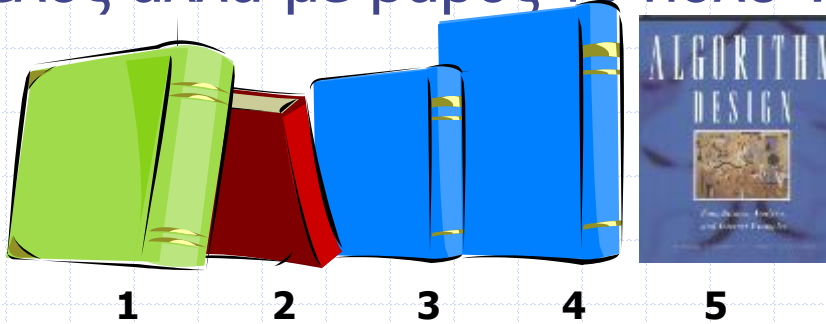
- ◆ Δεδομένα: Ένα σύνολο S η αντικειμένων, με κάθε αντικείμενο i να έχει
 - w_i – ένα θετικό βάρος
 - b_i – μία θετική τιμή οφέλους
- ◆ Στόχος: Επιλογή αντικειμένων με το μέγιστο συνολικό όφελος αλλά με βάρος το πολύ W .
- ◆ Εάν **δεν** επιτρέπεται να πάρουμε κλασματικά ποσά τότε αυτό είναι το **πρόβλημα σακιδίου 0-1**.
 - Σε αυτήν την περίπτωση όπου το T δηλώνει το σύνολο των αντικειμένων που θα πάρουμε
 - Σκοπός: μεγιστοποίηση του $\sum_{i \in T} b_i$
 - Περιορισμός: $\sum_{i \in T} w_i \leq W$

Παράδειγμα



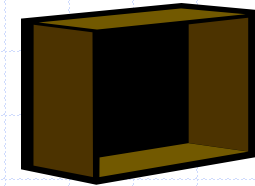
- ◆ Δεδομένα: Ένα σύνολο S η αντικειμένων, με κάθε αντικείμενο i να έχει
 - w_i – ένα θετικό βάρος
 - b_i – μία θετική τιμή οφέλους
- ◆ Στόχος: Επιλογή αντικειμένων με το μέγιστο συνολικό όφελος αλλά με βάρος το πολύ W .

Items:



Weight:	4 in	2 in	2 in	6 in	2 in
Benefit:	\$20	\$3	\$6	\$25	\$80

“knapsack”

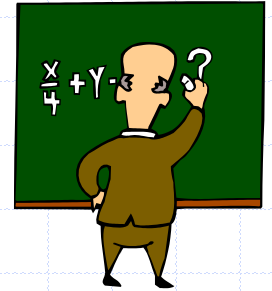


κουτί πλάτους 9 ιντσών

Solution:

- item 5 (\$80, 2 in)
- item 3 (\$6, 2in)
- item 1 (\$20, 4in)

Η Γενική τεχνική δυναμικού προγραμματισμού



- ◆ Εφαρμόζεται σε προβλήματα που αρχικά φαίνεται να απαιτούν πολύ χρόνο (πιθανότατα εκθετικό), αρκεί να έχουμε:
 - **Απλά υπό-προβλήματα:** τα υπό-προβλήματα μπορούν να οριστούν με λίγους μόνο δείκτες, όπως j, k, l, m , κ.ο.κ..
 - **Άριστα υπό-προβλήματα:** η καθολική βέλτιστη λύση μπορεί είναι μία σύνθεση των βέλτιστων λύσεων στα υπό-προβλήματα.
 - **Επικάλυψη υπό-προβλημάτων:** τα υπό-προβλήματα δεν είναι ανεξάρτητα, αλλά επικαλύπτονται (οπότε πρέπει να κατασκευαστούν από κάτω προς τα πάνω).

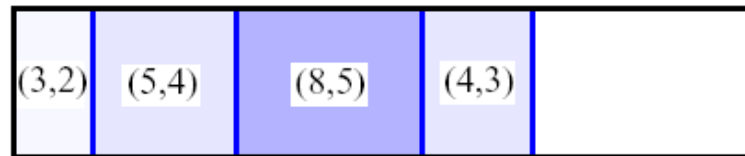
Πρόβλημα σακιδίου 0-1

Αλγόριθμος, Πρώτη προσπάθεια

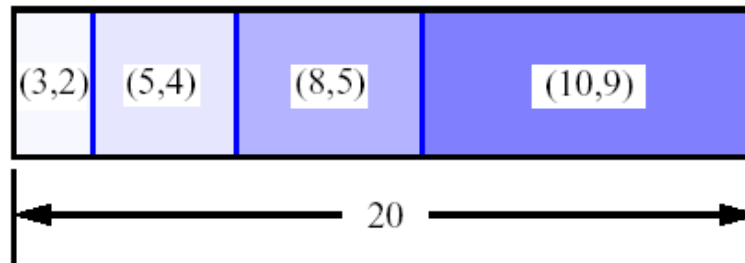


- ◆ S_k : Σετ αντικειμένων από το 1 μέχρι το k .
- ◆ Ορισμός $B[k]$ = η καλύτερη επιλογή από το S_k .
- ◆ Πρόβλημα: δεν υπάρχουν άριστα υπό-προβλήματα:
 - Σκεφτείτε το σύνολο $S = \{(3,2), (5,4), (8,5), (4,3), (10,9)\}$ με ζεύγη (όφελος, βάρος) και συνολικό βάρος $W = 20$

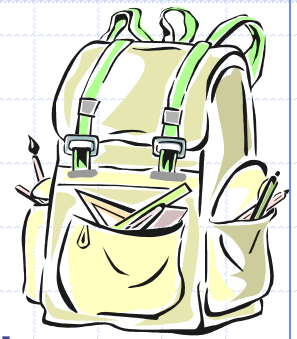
Καλύτερο για S_4 :



Καλύτερο για S_5 :



Πρόβλημα σακιδίου 0-1 Αλγόριθμος, Δεύτερη (καλύτερη) προσπάθεια



- ◆ S_k : Σύνολο αντικειμένων αριθμημένα από το 1 έως το k .
- ◆ Ορισμός του $B[k, w]$ να είναι η καλύτερη επιλογή από το S_k με βάρος το πολύ w
- ◆ Καλά νέα: εδώ υπάρχουν άριστα υπό-προβλήματα.

$$B[k, w] = \begin{cases} B[k-1, w] & \text{if } w_k > w \\ \max\{B[k-1, w], B[k-1, w - w_k] + b_k\} & \text{else} \end{cases}$$

- ◆ Π.χ., το καλύτερο υποσύνολο του S_k με βάρος το πολύ w είναι ένα από τα:
 - το καλύτερο υποσύνολο του S_{k-1} με βάρος το πολύ w ή
 - το καλύτερο υποσύνολο του S_{k-1} με βάρος το πολύ $w - w_k$ συν το αντικείμενο k

Πρόβλημα σακιδίου 0-1 Αλγόριθμος



$$B[k, w] = \begin{cases} B[k-1, w] & \text{if } w_k > w \\ \max\{B[k-1, w], B[k-1, w - w_k] + b_k\} & \text{else} \end{cases}$$

- ◆ Θυμηθείτε τον ορισμό του $B[k, w]$
- ◆ Επειδή το $B[k, w]$ ορίζεται με όρου του $B[k-1, *]$, μπορούμε να χρησιμοποιήσουμε δύο πίνακες αντί για πλέγμα
- ◆ Χρόνος εκτέλεσης: $O(nW)$.
- ◆ Δεν είναι αλγόριθμος πολυωνυμικού χρόνου καθώς το W μπορεί να είναι μεγάλο
- ◆ Είναι αλγόριθμος **ψευδό-πολυωνυμικού** χρόνου

Algorithm *01Knapsack*(S, W):

Input: set S of n items with benefit b_i and weight w_i ; maximum weight W

Output: benefit of best subset of S with weight at most W

let A and B be arrays of length $W + 1$

for $w \leftarrow 0$ **to** W **do**

$B[w] \leftarrow 0$

for $k \leftarrow 1$ **to** n **do**

copy array B into array A

for $w \leftarrow w_k$ **to** W **do**

if $A[w - w_k] + b_k > A[w]$ **then**

$B[w] \leftarrow A[w - w_k] + b_k$

return $B[W]$