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

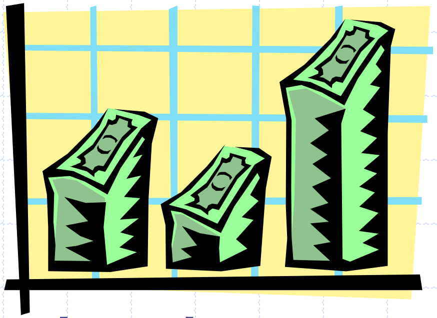
Η άπληστη μέθοδος

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

Εφαρμογή: Δημοπρασίες παγκόσμιου ιστού

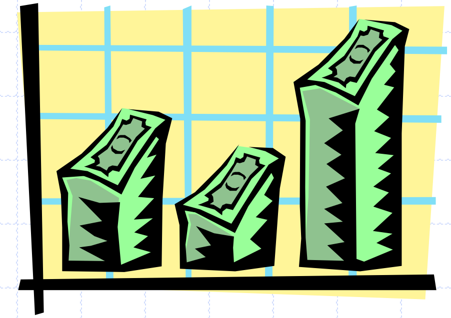
- Έστω ένας νέος **δικτυακός τόπος ηλεκτρονικών δημοπρασιών**, που θα επεξεργάζεται προσφορές για δημοπρασίες πολλών αντικειμένων.
- Για παράδειγμα: Ο δικτυακός τόπος θα πρέπει να είναι σε θέση να χειριστεί μία δημοπρασία 100 μονάδων της ίδιας ψηφιακής κάμερας ή 500 μονάδων του ίδιου smartphone, με τις προσφορές να εμφανίζονται με τη μορφή, **“$y για x μονάδες”**, που σημαίνει ότι ο συμμετέχων στη δημοπρασία θέλει **x** μονάδες από το αντικείμενο που πωλείται και είναι πρόθυμος να πληρώσει **$y** για αυτές.
- Ο δικτυακός τόπος δημοπρασιών θα επιτρέπει σε μεγάλο αριθμό συμμετεχόντων να κάνουν προσφορές πολλών αντικειμένων και θα αποφασίζει ποιες δημοπρασίες θα κατακυρώνονται.
- Προκειμένου να μεγιστοποιήσει τις προμήθειες από τις πωλήσεις, ο δικτυακός τόπος θα πρέπει να επιλέγει τις προσφορές που μεγιστοποιούν το συνολικό χρηματικό ποσό που καταβάλλεται για τα αντικείμενα που δημοπρατούνται.

Η άπληστη μέθοδος

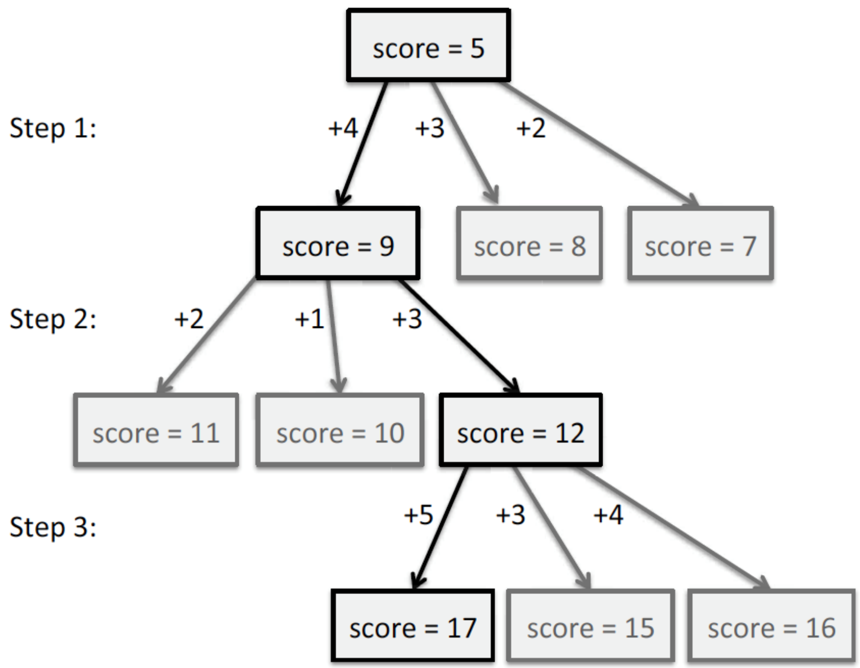


- **Η άπληστη μέθοδος (greedy method)** είναι ένα γενικό παράδειγμα σχεδίασης αλγορίθμων που βασίζεται στα ακόλουθα στοιχεία:
  - **διαμορφώσεις (configuration)**: μια διαμόρφωση ορίζεται από τις τιμές που λαμβάνουν οι παράμετροι που ορίζουν μια λύση
  - **αντικειμενική συνάρτηση (objective function)**: κάθε διαμόρφωση αντιστοιχεί σε μια βαθμολογία, που θέλουμε να μεγιστοποιήσουμε ή να ελαχιστοποιήσουμε
- Η άπληστη μέθοδος λειτουργεί βέλτιστα σε προβλήματα που έχουν την ιδιότητα της **άπληστης-επιλογής (greedy-choice)** δηλαδή προβλήματα για τα οποία ισχύει ότι μια καθολικά βέλτιστη λύση, μπορεί πάντα να βρεθεί με μια σειρά τοπικών βελτιώσεων ξεκινώντας από μια αρχική διαμόρφωση

Η άπληστη μέθοδος



- Η σειρά επιλογών ξεκινά από μια καλά καθορισμένη εναρκτήρια διαμόρφωση και κατόπιν, με επαναληπτικό τρόπο επιλέγεται η καλύτερη διαμόρφωση απ' τις τρέχουσες διαθέσιμες, με γνώμονα τη βελτίωση της αντικειμενικής συνάρτησης.



Εφαρμογή διαδικτυακών δημοπρασιών

- Η άπληστη στρατηγική λειτουργεί αν μπορείτε να ικανοποιήσετε μια προσφορά αγοράς **x** μονάδων για **$y** πουλώντας **k < x** μονάδες για **$y*k/x**.
- Σε αυτήν την περίπτωση το πρόβλημα είναι ισοδύναμο με το **κλασματικό πρόβλημα σακιδίου**.

American GIs recover works of art stolen by the Nazis (NARA/Public Domain)

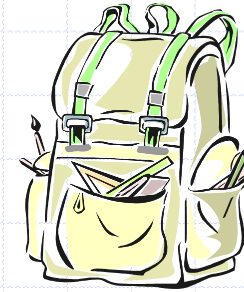
Διαδικτυακές δημοπρασίες και το κλασματικό πρόβλημα σακιδίου

- Στο **πρόβλημα σακιδίου (knapsack problem)**, δίνεται ένα σύνολο n αντικειμένων με το καθένα να έχει κάποιο βάρος και κάποιο όφελος και θέλουμε να επιλέξουμε το σύνολο των αντικειμένων που μεγιστοποιεί το κέρδος χωρίς να υπερβαίνει το βάρος του σακιδίου.
- Στην εφαρμογή των δικτυακών δημοπρασιών, κάθε προσφορά είναι ένα αντικείμενο, με το "βάρος" να είναι ο αριθμός ο μονάδων που ζητούνται και το όφελος να είναι το χρηματικό ποσό της προσφοράς.
- Στην περίπτωση που οι προσφορές μπορούν να ικανοποιηθούν μερικά, τότε είναι ισοδύναμο του **κλασματικού** προβλήματος σακιδίου, για το οποίο η άπληστη μέθοδος βρίσκει την καθολικά βέλτιστη λύση.
- Έχει ενδιαφέρον ότι για την έκδοση "0-1" του προβλήματος , όπου δεν επιτρέπονται κλασματικές επιλογές, η άπληστη μέθοδος ενδέχεται να μην δίνει τη βέλτιστη λύση και το πρόβλημα να είναι πολύ δύσκολο να λυθεί σε πολυωνυμικό χρόνο.

Το κλασματικό πρόβλημα σακιδίου

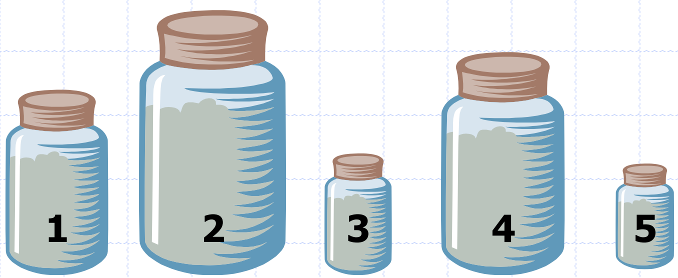
- Δεδομένα: Ένα σύνολο **S** από **n** αντικείμενα, με κάθε αντικείμενο **i** να έχει:
  - $\mathbf{b_i}$ – μια θετική τιμή οφέλους
  - $\mathbf{w_i}$ – μια θετική τιμή βάρους
- Στόχος: Η επιλογή των αντικειμένων με το μέγιστο συνολικό όφελος αλλά με βάρος το πολύ **W**.
- Όταν επιτρέπεται να έχουμε κλασματικές ποσότητες αντικειμένων, τότε προκύπτει το **κλασματικό πρόβλημα σακιδίου**.
  - Σε αυτήν την περίπτωση, το $\mathbf{x_i}$ υποδηλώνει την ποσότητα που επιλέγουμε από το αντικείμενο **i**
  - Σκοπός: μεγιστοποίηση του $\sum_{i \in S} b_i (x_i / w_i)$
  - Περιορισμός: $\sum_{i \in S} x_i \leq W$

Παράδειγμα



- Δεδομένα: Ένα σύνολο **S** από **n** αντικείμενα, με κάθε αντικείμενο **i** να έχει:
  - $\mathbf{b_i}$ – μια θετική τιμή οφέλους
  - $\mathbf{w_i}$ – μια θετική τιμή βάρους
- Στόχος: Η επιλογή αντικειμένων με το μέγιστο συνολικό όφελος αλλά με βάρος το πολύ B.

Αντικείμενα:



Βάρος:	4 ml	8 ml	2 ml	6 ml	1 ml
Όφελος:	$12	$32	$40	$30	$50
Τιμή: ($ ανά ml)	3	4	20	5	50

10 ml

“σακίδιο”

Λύση:
- 1 ml του 5
- 2 ml του 3
- 6 ml του 4
- 1 ml του 2

Ο αλγόριθμος του κλασματικού προβλήματος σακιδίου

- Άπληστη επιλογή: Επιλογή του αντικειμένου με την υψηλότερη τιμή (αναλογία όφελος-προς-βάρος)
  - Επειδή $\sum_{i \in S} b_i (x_i / w_i) = \sum_{i \in S} (b_i / w_i) x_i$
  - Χρόνος: O(n log n)

Algorithm *fractionalKnapsack*(*S*, *W*)
    Input: set *S* of items with benefit b_i and weight w_i; max. weight *W*
    Output: amount x_i of each item *i* to maximize benefit with weight at most *W*
    for ***each item i in S***
        $x_i \leftarrow 0$
        $v_i \leftarrow b_i / w_i$ {value}
    $w \leftarrow 0$ {total weight}
    while $w < W$
        remove item i with highest v_i
        $x_i \leftarrow \min\{w_i, W - w\}$
        $w \leftarrow w + \min\{w_i, W - w\}$

Ανάλυση του άπληστου αλγόριθμου για το κλασματικό πρόβλημα σακιδίου

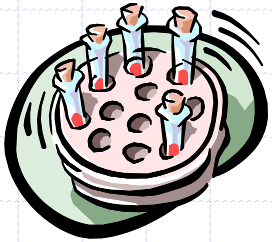
- Μπορούμε να ταξινομήσουμε τα αντικείμενα με τις όφελος-προς-βάρος τιμές και κατόπιν να τα επεξεργαστούμε μ' αυτήν τη σειρά.
- Θα χρειαστεί **O(n log n)** χρόνος για την ταξινόμηση των αντικειμένων και επιπλέον χρόνο **O(n)** για την επεξεργασία τους στο βρόχο while.
- Για να αποδείξουμε ότι ο αλγόριθμος μας είναι σωστός, υποθέτουμε, ώστε να φτάσουμε σε αντίφαση, ότι υπάρχει μια βέλτιστη λύση, που είναι καλύτερη από εκείνη που επιλέχθηκε από τον άπληστο αλγόριθμο.
- Τότε πρέπει να υπάρχουν δύο αντικείμενα **i** και **j** τέτοια ώστε

$$\mathbf{x_i < w_i,\ x_j > 0}, \text{ και } \mathbf{v_i > v_j}$$

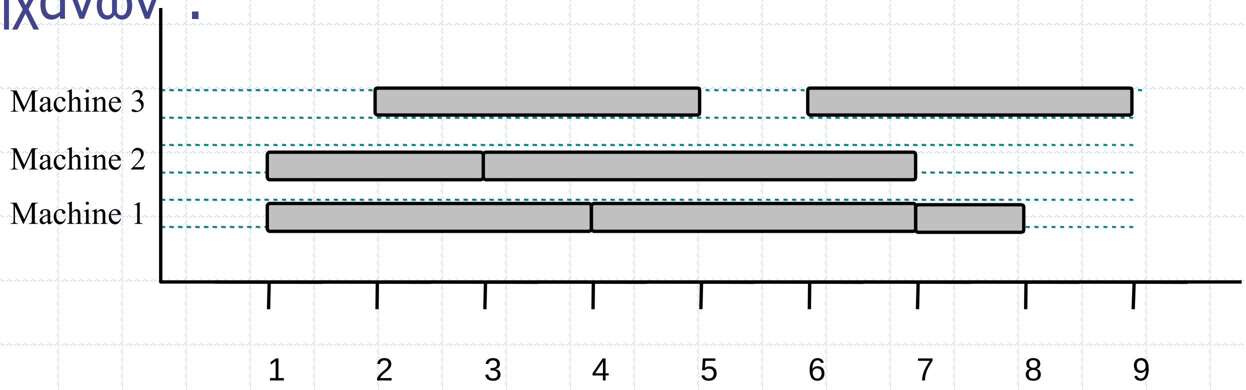
δηλαδή, αν και το αντικείμενο **i** έχει υψηλότερη αξία από το αντικείμενο **j**, υπάρχει ακόμα διαθέσιμη ποσότητα του **i**, ενώ στη λύση βρίσκεται ποσότητα του **j**

- Έστω ότι $\mathbf{y = \min\{w_i - x_i, x_j\}}$.
- Θα μπορούσαμε να αντικαταστήσουμε ποσότητα **y** του αντικειμένου **j** με ίση ποσότητα του αντικειμένου **i**, αυξάνοντας έτσι το συνολικό όφελος χωρίς να μεταβάλλουμε το συνολικό βάρος, κάτι που αντιβαίνει στην παραδοχή ότι η υποθετική λύση είναι η βέλτιστη.

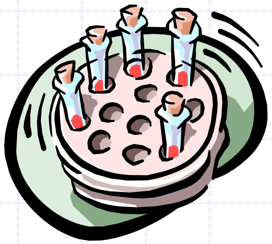
Παράδειγμα



- Δεδομένα: ένα σύνολο **T** από **n** εργασίες, με καθεμία να έχει:
  - Ώρα έναρξης, $\mathbf{s_i}$
  - Ώρα λήξης, $\mathbf{f_i}$ (όπου $\mathbf{s_i < f_i}$)
  - [1,4], [1,3], [2,5], [3,7], [4,7], [6,9], [7,8] (ταξινομημένες βάσει χρονικής στιγμής έναρξης)
- Στόχος: Εκτέλεση όλων των εργασιών ελαχιστοποιώντας τον αριθμό “μηχανών”.



Αλγόριθμος χρονοπρογραμματισμού εργασιών



- Άπληστη επιλογή: εργασίες ταξινομημένες κατά χρόνο έναρξης και χρήση όσο λιγότερων μηχανών με αυτήν τη σειρά.
  - Χρόνος: **O(n log n)**.
- Ορθότητα: Υποθέτουμε ότι υπάρχει καλύτερο πρόγραμμα που χρησιμοποιεί **k-1** μηχανές, ενώ ο αλγόριθμος taskSchedule χρησιμοποιεί **k**.
  - Έστω ότι **i** είναι η πρώτη εργασία που έχει προγραμματιστεί στην μηχανή **k**
  - Η εργασία **i** θα συγκρούεται με **k-1** άλλες εργασίες
  - Αυτό όμως σημαίνει ότι δεν υπάρχει πρόγραμμα χωρίς συγκρούσεις που χρησιμοποιεί **k-1** μηχανές

Algorithm ***taskSchedule*(*T*)**
    Input: set *T* of tasks with start time s_i and finish time f_i
    Output: non-conflicting schedule with minimum number of machines
    $m \leftarrow 0$ {no. of machines}
    while *T is not empty*
        remove task i with smallest s_i
        if ***there's a machine j for i*** **then**
            schedule i on machine j
        else
            $m \leftarrow m + 1$
            schedule i on machine m

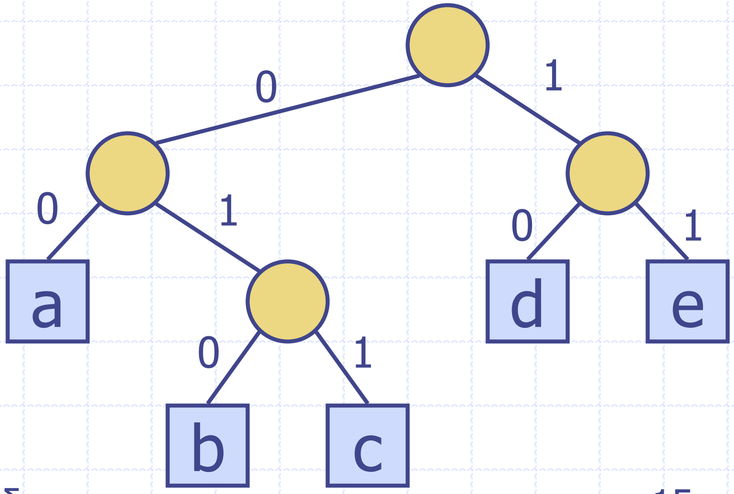
Συμπίεση κειμένου

- Δεδομένης μίας συμβολοσειράς **X**, κωδικοποιείστε το **X** σε μία μικρότερη συμβολοσειρά **Y**
  - Εξοικονόμηση μνήμης και/ή εύρος ζώνης
- Μία καλή προσέγγιση: **Κωδικοποίηση Huffman**
  - Υπολογισμός της συχνότητας **f(c)** για κάθε χαρακτήρα **c**.
  - Κωδικοποίηση των χαρακτήρων με υψηλή συχνότητα εμφάνισης με κωδικούς μικρού μήκους
  - Κανένας κωδικός δεν θα πρέπει να είναι πρόθεμα κάποιου άλλου κωδικού
  - Η κωδικοποίηση Huffman χρησιμοποιεί ένα βέλτιστο δένδρο κωδικοποίησης το οποίο καθορίζει τους κωδικούς

Παράδειγμα δένδρου κωδικοποίησης

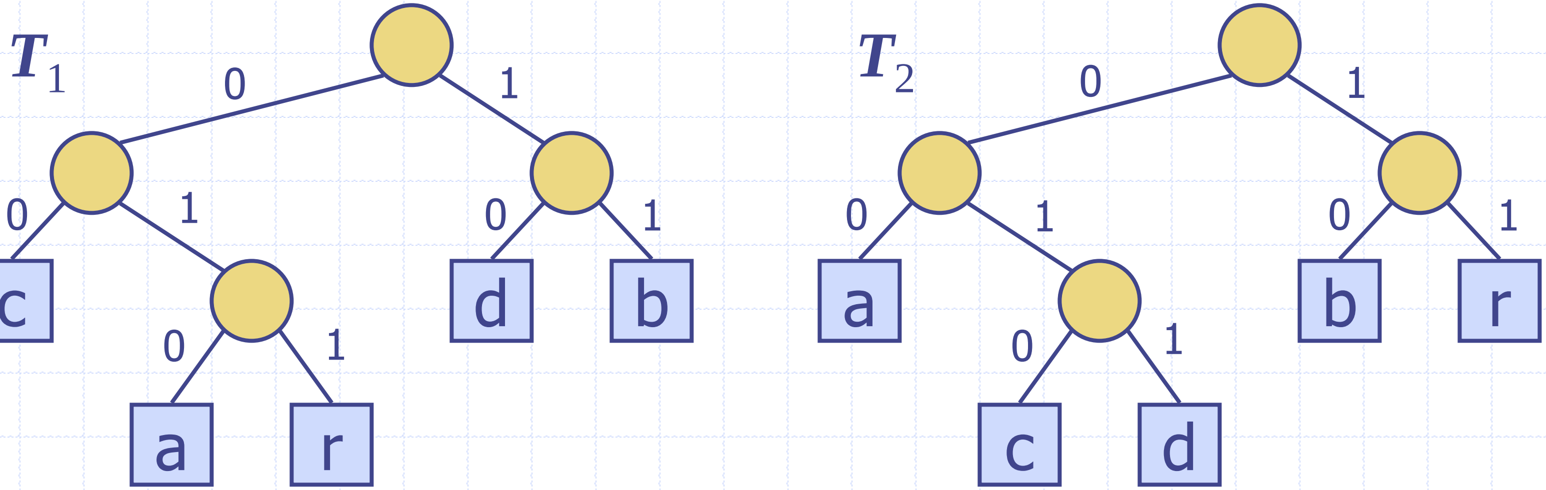
- Μια **κωδικοποίηση** είναι μία αντιστοίχιση κάθε χαρακτήρα του αλφαβήτου σε έναν δυαδικό κωδικό
- Μια **προθεματική κωδικοποίηση** είναι μια δυαδική κωδικοποίηση τέτοια ώστε κανένας κωδικός να μην είναι το πρόθεμα κάποιου άλλου κωδικού
- Ένα **δένδρο κωδικοποίησης** αντιπροσωπεύει μια προθεματική κωδικοποίηση όπου:
  - Κάθε φύλλο περιέχει ένα χαρακτήρα
  - Ο κωδικός κάθε χαρακτήρα δίνεται από το μονοπάτι από τη ρίζα προς τον φύλλο του δένδρου που αποθηκεύει τον χαρακτήρα (0 για κάθε αριστερό παιδί και 1 για κάθε δεξί παιδί)

00	010	011	10	11
a	b	c	d	e



Βελτιστοποίηση δένδρου κωδικοποίησης

- Δεδομένης μίας συμβολοσειράς X, θέλουμε να βρούμε μια προθεματική κωδικοποίηση του X που θα παράγει μία κωδικοποίηση μικρού μήκους για το X
  - Οι συχνοί χαρακτήρες πρέπει να έχουν κωδικούς μικρού μήκους
  - Οι σπάνιοι χαρακτήρες πρέπει να έχουν κωδικούς μεγάλου μήκους
- Παράδειγμα
  - X = abracadabra
  - Το T_1 κωδικοποιεί το X σε 29 bits
  - Το T_2 κωδικοποιεί το X σε 24 bits



Ο αλγόριθμος Huffman

- Δεδομένης μιας συμβολοσειράς X, ο αλγόριθμος Huffman δημιουργεί μια προθεματική κωδικοποίηση που ελαχιστοποιεί το μήκος της κωδικοποίησης της X
- Εκτελείται σε χρόνο $O(n + d \log d)$, όπου n το μήκος του X και d ο αριθμός των διακριτών χαρακτήρων του X
- Μία ουρά προτεραιότητας (με εσωτερική υλοποίηση σωρού) μπορεί να λειτουργήσει ως βοηθητική δομή για την επιτάχυνση της εκτέλεσης του αλγορίθμου

Ο αλγόριθμος Huffman

Algorithm Huffman(X):
  Input: String X of length n with d distinct characters
  Output: Coding tree for X

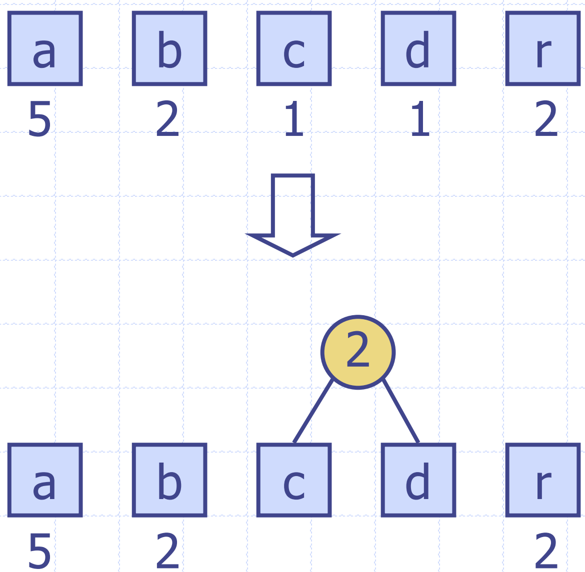
Compute the frequency $f(c)$ of each character c of X.
Initialize a priority queue Q.
for each character c in X **do**
  Create a single-node binary tree T storing c.
  Insert T into Q with key $f(c)$.
while len(Q) > 1 **do**
  $(f_1, T_1) = Q$.remove_min()
  $(f_2, T_2) = Q$.remove_min()
  Create a new binary tree T with left subtree T_1 and right subtree T_2.
  Insert T into Q with key $f_1 + f_2$.
$(f, T) = Q$.remove_min()
return tree T

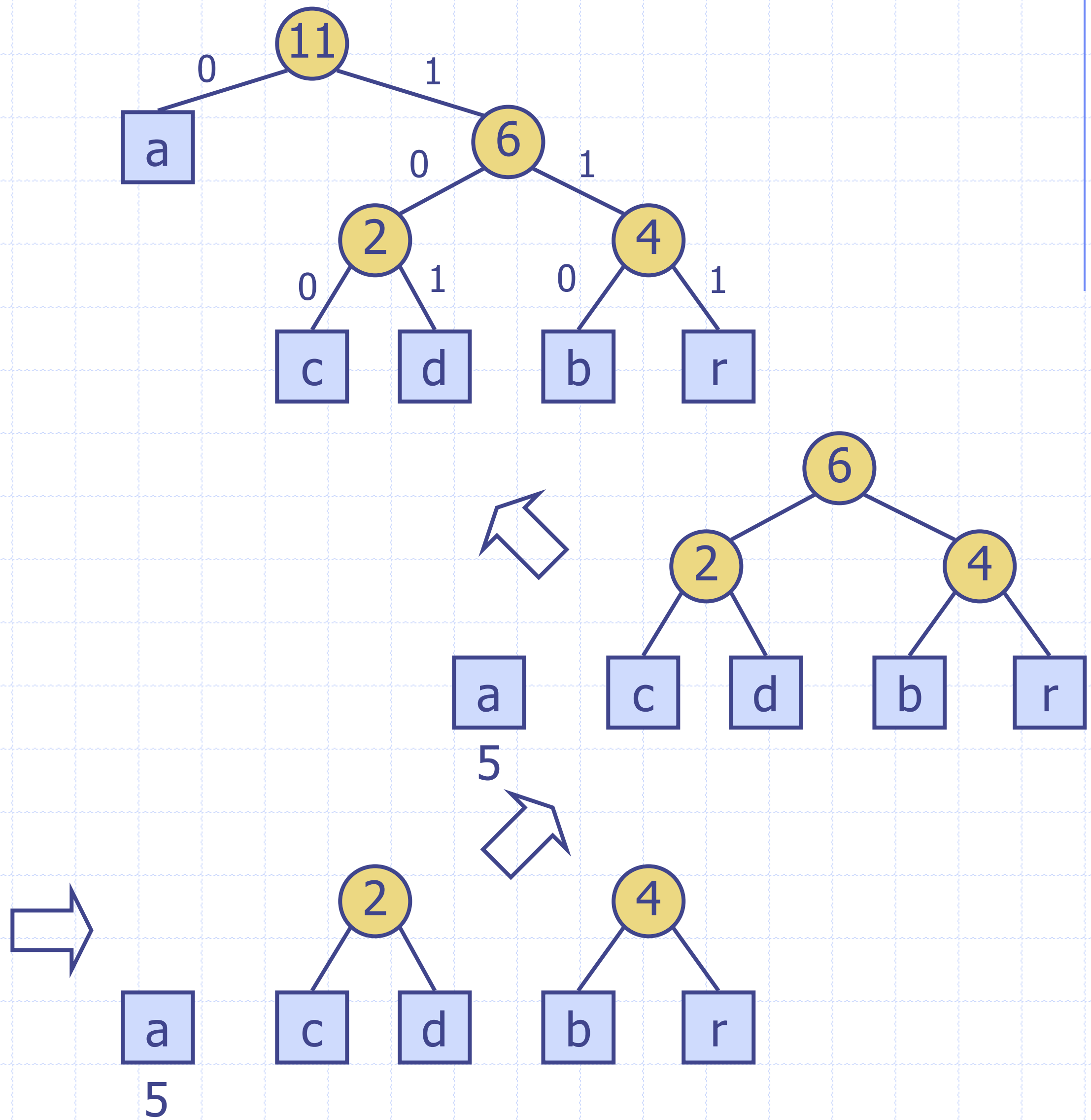
Παράδειγμα

X = abracadabra

Συχνότητες

a	b	c	d	r
5	2	1	1	2





Μεγαλύτερο παράδειγμα δένδρου Huffman

String: **a fast runner need never be afraid of the dark**

Character		a	b	d	e	f	h	i	k	n	o	r	s	t	u	v
Frequency	9	5	1	3	7	3	1	1	1	4	1	5	1	2	1	1