

# Αλγόριθμοι και Πολυπλοκότητα

7ο εξάμηνο  
Σ.Η.Μ.Μ.Υ. & Σ.Ε.Μ.Φ.Ε.

<http://www.corelab.ece.ntua.gr/courses/>

*4η εβδομάδα: Εύρεση k-οστού Μικρότερου Στοιχείου,  
Master Theorem, Τεχνική Greedy: Knapsack, Minimum  
Spanning Tree, Shortest Paths*

Διδάσκοντες:  
Στάθης Ζάχος - Άρης Παγουρτζής

# Η Τεχνική Greedy (Άπληστη)

```
function Greedy (a:set of elements) : solution;  
var  
  x:item;  
begin  
  solution:=empty;  
  for all elements of a do  
    begin  
      (* χρήση των κανόνων βελτιστοποίησης *)  
      x:=OptSelectRemove(a);  
      (* έλεγχος αν πληρούνται οι περιορισμοί *)  
      if feasible(solution+{x}) then solution:=solution+{x}  
      (* η λύση μεγαλώνει και ενημερώνεται η αντικειμενική συνάρτηση *)  
    end;  
  greedy:=solution  
End
```

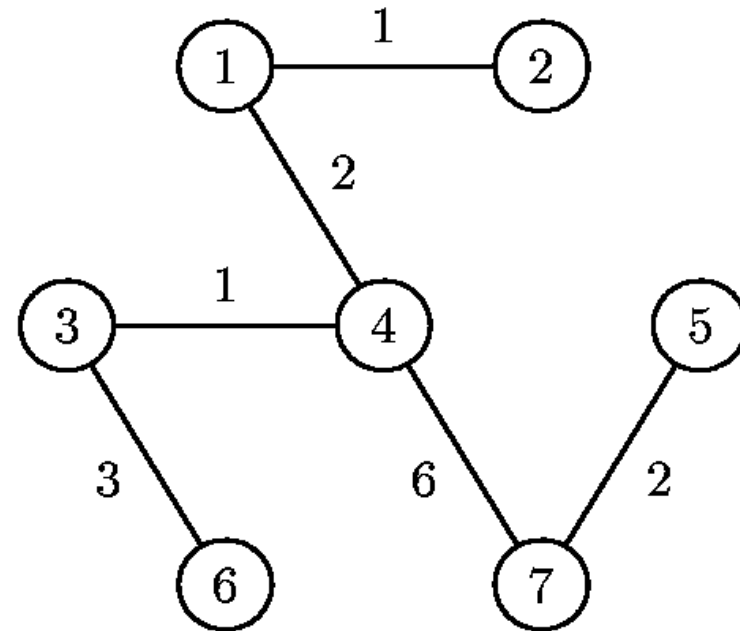
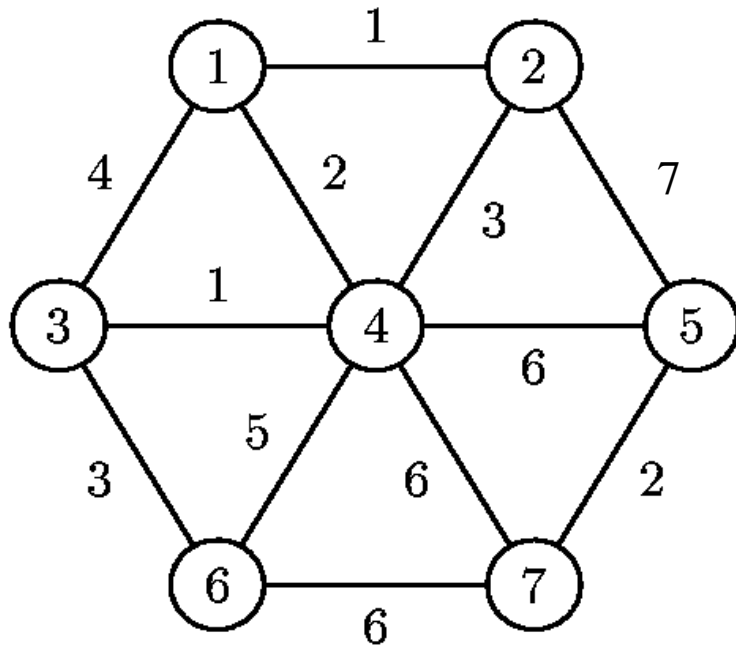
# Πρόβλημα Σακιδίου (Knapsack)

- Είσοδος: σακίδιο χωρητικότητας  $M$  και  $n$  "συνεχή" αντικείμενα βάρους  $w_i$  και αξίας  $p_i$ .
- Αποδεκτή λύση: καθορισμός κλάσματος  $x_i$  για κάθε αντικείμενο ώστε να χωράνε στο σακίδιο.
- Στόχος: μεγιστοποίηση της αξίας των αντικειμένων που μπήκαν στο σακίδιο.

# Άπληστος Αλγόριθμος για το Πρόβλημα Σακιδίου

- Αν το άθροισμα όλων των βαρών είναι μικρότερο από  $M$ , θέσε  $x_i=1$  για όλα.
- Αλλιώς, διάτρεξε τα αντικείμενα κατά φθίνουσα σειρά  $p_i/w_i$ , παίρνοντας τη μέγιστη δυνατή ποσότητα.
- Πολυπλοκότητα:  $O(n \log n)$

# Συνδετικό Δένδρο Ελαχίστου Κόστους (Minimum Cost Spanning Tree)



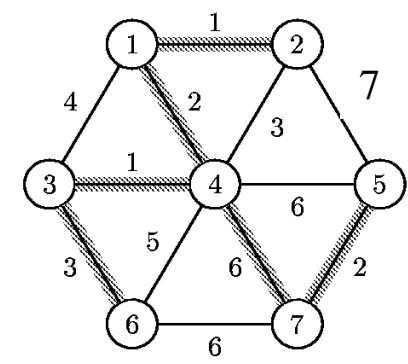
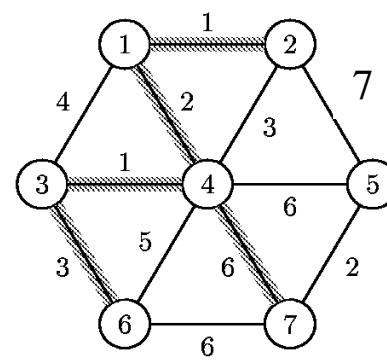
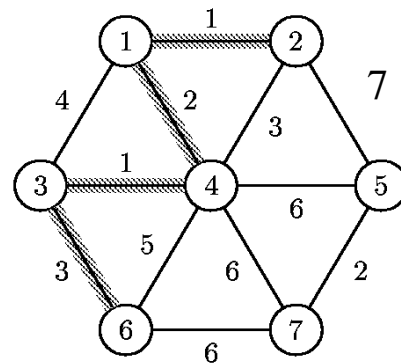
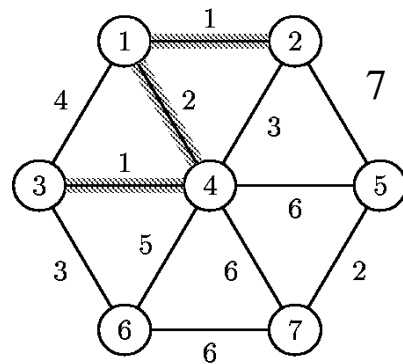
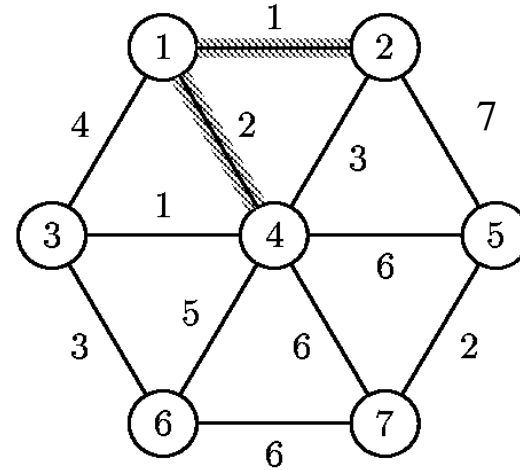
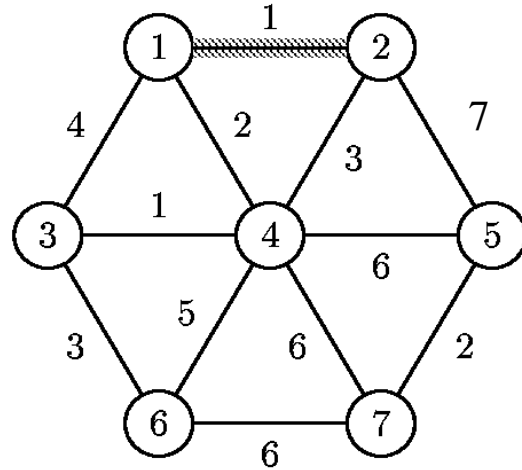
# Δύο μέθοδοι

- Κριτήριο Prim: επιλογή πλευράς ελαχίστου κόστους ώστε ο υπογράφος να παραμένει δέντρο
- Κριτήριο Kruskal: επιλογή πλευράς ελαχίστου κόστους ώστε ο υπογράφος να παραμένει ακυκλικός

# Άπληστος Αλγόριθμος με Κριτήριο Prim

```
procedure Prim (E: set of edges;  
                cost: array[1..n,1..n] of real;  
                var tree: array[1..n-1] of edges;  
                var mincost : real);  
var DistFromTree:array[1..n] of real;  
    Edge:array[1..n] of edges; i,j,k,l : integer;  
(* Το DistFromTree[i] περιέχει το ελάχιστο από τα κόστη των πλευρών  
    που συνδέουν τον κόμβο i με το μέχρι στιγμής κατασκευασμένο  
    συνδυαστικό δέντρο και το Edge[i] περιέχει την πλευρά με αυτό  
    το κόστος *)  
begin  
    tree[1]:=(k,l); mincost:=cost[k,l]; (* (k,l)= edge with minimum cost *)  
    for i:=1 to n do  
        begin DistFromTree[i]:=min(cost[k,i],cost[l,i]); update Edge[i] end;  
        for i:=2 to n-1 do  
            begin  
                select j such that DistFromTree[j] is minimum but  $\neq 0$ ;  
                tree[i]:=Edge[j]; mincost:=mincost+DistFromTree[j];  
                DistFromTree[j]:=0;  
                for k:=1 to n do begin update DistFromTree[k]; update Edge[k] end  
            end  
    end  
end
```

# Παράδειγμα (με Prim)

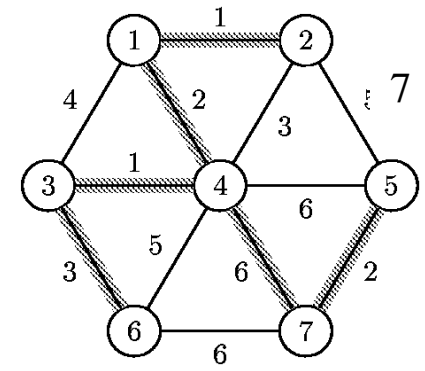
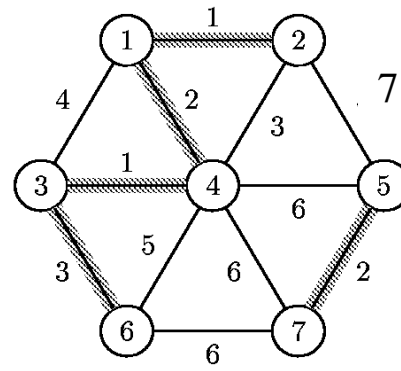
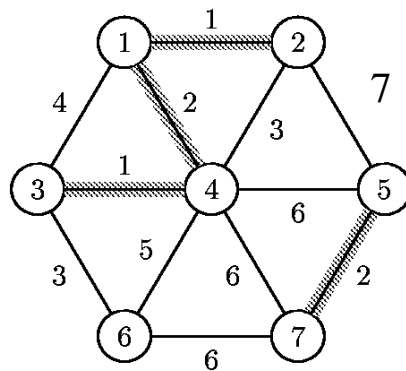
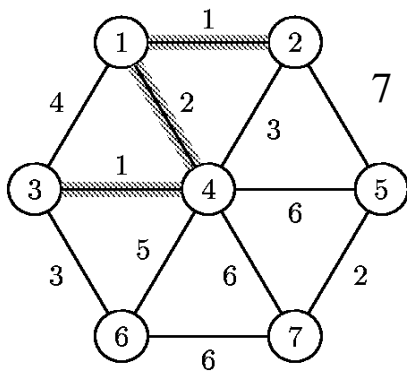
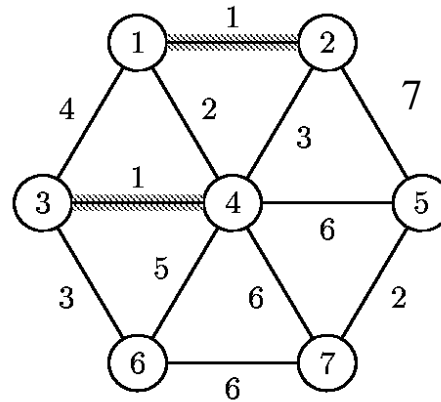
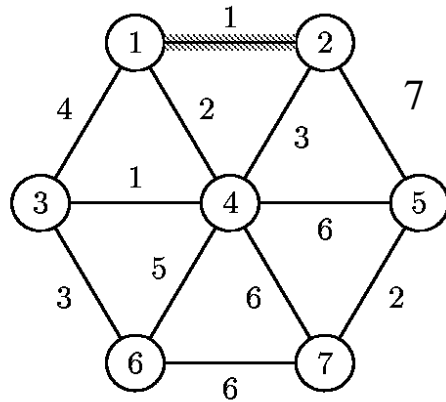




# Άπληστος Αλγόριθμος με Κριτήριο Kruskal

```
procedure Kruskal (...);  
(*θεωρεί ακμές ταξινομημένες κατά βάρος, αρκεί όμως να είναι σε heap*)  
begin  
  forest:=empty;  
  while |forest| < n-1 do  
    begin  
      select Min_Cost edge (u,w) and delete it from E;  
      if (u,w) does not create a cycle in forest then  
        add it to forest  
      else discard it  
    end  
  end  
end
```

# Παράδειγμα (με Kruskal)



# Πολυπλοκότητα - Βελτιστότητα

- Έστω  $n=|V|$ ,  $e=|E|$
- Prim:  $O(n^2)$
- Kruskal:  $O(e \log n)$
- Συνεκτικοί γράφοι:  
$$n-1 \leq e \leq n(n-1)/2$$
- Ο αλγόριθμος Kruskal καλύτερος σε αραιούς (dense) γράφους
- Θεώρημα: ο Kruskal δίνει βέλτιστη λύση

# Ορθότητα Kruskal

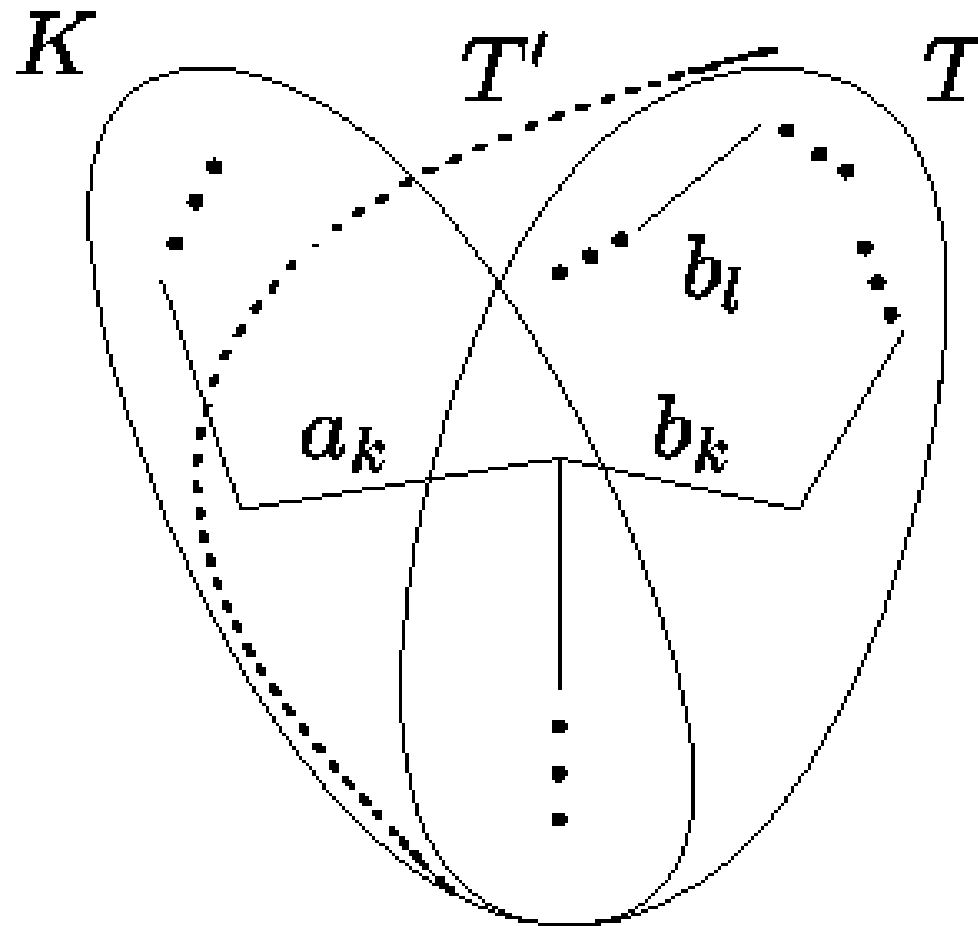
**Θεώρημα 8.4.1.** Η λύση που δίνει ο αλγόριθμος του Kruskal στο πρόβλημα του συνδετικού δέντρου είναι η βέλτιστη.

*Απόδειξη.* Έστω  $a_1, a_2, \dots, a_n$  οι ακμές σε αύξουσα σειρά βάρους στο δέντρο  $K$ , το οποίο προκύπτει από τον αλγόριθμο του Kruskal. Έστω  $b_1, b_2, \dots, b_n$  οι ακμές σε αύξουσα σειρά βάρους στο δέντρο  $T$  το οποίο είναι το βέλτιστο συνδετικό δέντρο ελαχιστού κόστους. Έστω ότι τα δύο δέντρα δεν ταυτίζονται, δηλαδή ότι υπάρχει  $i$  για το οποίο  $a_i \neq b_i$ . Έστω  $k$  το ελάχιστο τέτοιο  $i$ , δηλαδή:

$$\begin{cases} a_j = b_j, \forall j < k \\ a_k \neq b_k \end{cases}$$

Περίπτωση 1: Το κόστος της  $a_k$  είναι μεγαλύτερο από το κόστος της  $b_k$ . Τότε στο  $k$ -οστό βήμα ο αλγόριθμος θα προτιμούσε την πλευρά  $b_k$  από την πλευρά  $a_k$ , διότι, αφού το μέχρι στιγμής δέντρο είναι το ίδιο δεν σχηματίζεται κύκλος ούτε με το  $b_k$ . Άτοπο.

# Ορθότητα Kruskal (συν.)



# Ορθότητα Kruskal (συν.)

Περίπτωση 2: Το κόστος της  $a_k$  είναι μικρότερο ή ίσο από το κόστος της  $b_k$ . Στην περίπτωση αυτή προσθέτουμε στο δέντρο  $T$  την ακμή  $a_k$  κι έτσι σχηματίζεται ένας κύκλος. Ο κύκλος αυτός πρέπει να περιέχει μια ακμή  $b_l$  με  $l \geq k$  η οποία δεν ανήκει στο δέντρο  $K$ , αλλιώς ο κύκλος θα υπήρχε στο δέντρο  $K$ . Είναι φανερό ότι το κόστος της  $b_l$  είναι μεγαλύτερο ή ίσο από το κόστος της  $a_k$ , διότι ο αλγόριθμος Kruskal προτίμησε την  $a_k$  από την  $b_l$ . Αν αφαιρέσουμε από το  $T'$  (βλέπε σχήμα 8.4) την  $b_l$  τότε προκύπτει ένα δέντρο με μικρότερο ή ίσο βάρος από το ελάχιστο. Αν το βάρος είναι μικρότερο έχουμε καταλήξει σε άτοπο. Αν είναι ίσο τότε επαναλαμβάνουμε την ίδια διαδικασία έχοντας τώρα ένα βέλτιστο δέντρο για το οποίο το  $k$  είναι μεγαλύτερο. Μετά από έναν αριθμό επαναλήψεων ή καταλήγουμε σε άτοπο, ή τα δέντρα ταυτίζονται.  $\square$