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

Εύρεση Μικρότερου-Μεγαλύτερου

- Χώρισε σε 2 υποακολουθίες
- Βρες μεγαλύτερο και μικρότερο
- Σύγκρινε μεγαλύτερα και μικρότερα

$$T(n) = \begin{cases} 0 & , \text{για } n = 1 \\ 1 & , \text{για } n = 2 \\ 2T(\frac{n}{2}) + 2 & , \text{για } n > 2 \end{cases}$$

$$T(n) = \frac{n}{2}T(2) + 2^{k+1} - 2 = \frac{3}{2}n - 2 \qquad (\frac{n}{2} = 2^k)$$

Εύρεση *k*-οστού Μικρότερου

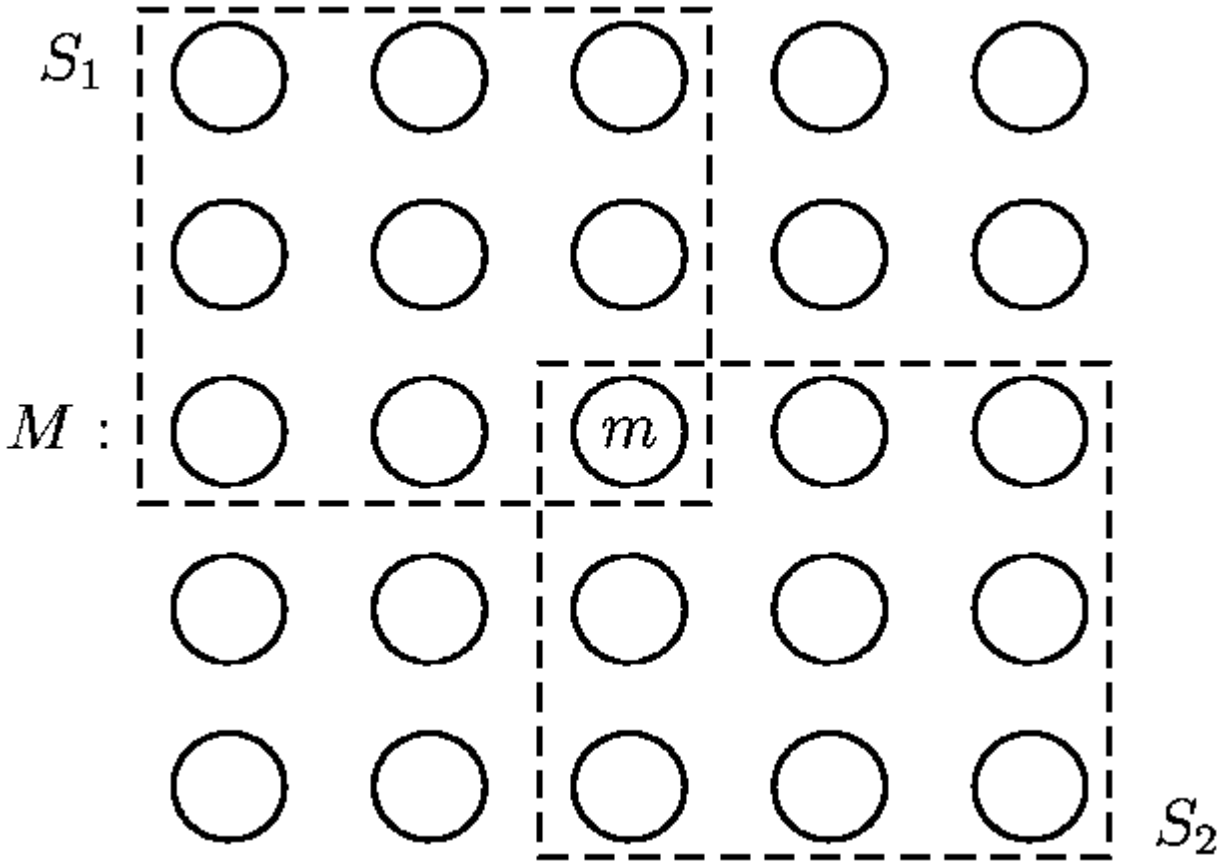
```
procedure Selection (p, q, k);
begin
  while p<q do
  begin
    choose t from [p..q]; partition (p, q, t);
    if k<t then q:=t−1 else begin p:=t; k:=k−p+1 end
  end
end
```

Πολυπλοκότητα: $O(n^2)$ χειρότερη, $O(n)$ μέση

Βελτίωση Πολυπλοκότητας

- Με επιλογή κατάλληλου σημείου partition
- Χωρισμός σε n/5 ακολουθίες 5 στοιχείων
- Ταξινόμηση 5άδων και σχηματισμός *M* από τα μεσαία στοιχεία: O(n)
- Εύρεση του μεσαίου *m* της *M*: T(n/5)
- Τουλάχιστον $\frac{1}{4}$ στοιχεία μεγαλύτερα του m και $\frac{1}{4}$ στοιχεία μικρότερα του m

Η μέθοδος σχηματικά



Πολυπλοκότητα Βελτίωσης

$$T(n) = \left\{ \begin{array}{ll} d & , \text{για } n \leq i \\ T(\frac{n}{5}) + T(\frac{3n}{4}) + cn & , \text{για } n > i \end{array} \right\} = O(n)$$

Θεώρημα 6.8.1. *Αν ισχύει ότι* $T(1) = d$ *και* $T(n) = T(an) + T(bn) + cn$ *με* $a + b < 1$ *τότε η συνάρτηση* $T(n)$ *είναι γραμμική, ισχύει δηλαδή ότι* $T(n) = O(n)$.

Το Κύριο Θεώρημα (Master Theorem)

Θεώρημα 6.9.1. Master Theorem *Εστω* $a \geq$ *και* $b > 1$ *σταθερές,* $f(n)$ *μια συνάρτηση, και η* $T(n)$ *ορίζεται στους μη αρνητικούς ακεραίους από την αναδρομή*

$$T(n) = aT(n/b) + f(n)$$

(το n/b *σημαίνει είτε* $\lfloor n/b \rfloor$ *είτε* $\lceil n/b \rceil$*). Τότε η* $T(n)$ *μπορεί να φραχτεί ασυμπτωτικά ως εξής:*

- $T(n) = \Theta(f(n))$, *αν* $f(n) = \Omega(n^{\log_b a + \epsilon})$ *για κάποια σταθερά* $\epsilon > 0$, *και αν* $af(n/b) \leq cf(n)$ *για κάποια σταθερά* $c > 1$ *και όλα τα αρκετά μεγάλα* n
- $T(n) = \Theta(f(n) log_2 n)$, *αν* $f(n) = \Theta(n^{\log_b a})$
- $T(n) = \Theta(n^{\log_b a})$, *αν* $f(n) = O(n^{\log_b a - \epsilon})$ *για κάποια σταθερά* $\epsilon > 0$

Υπενθύμιση: απλή εκδοχή του Master Theorem

$$\textit{Αν ισχύει } T(1) = d \textit{ και } T(n) = aT(\tfrac{n}{b}) + cn \textit{ τότε:}$$

$$T(n) = \begin{cases} O(n) & , \textit{για } a < b \\ O(n \log n) & , \textit{για } a = b \\ O(n^{\log_b a}) & , \textit{για } a > b \end{cases}$$

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.

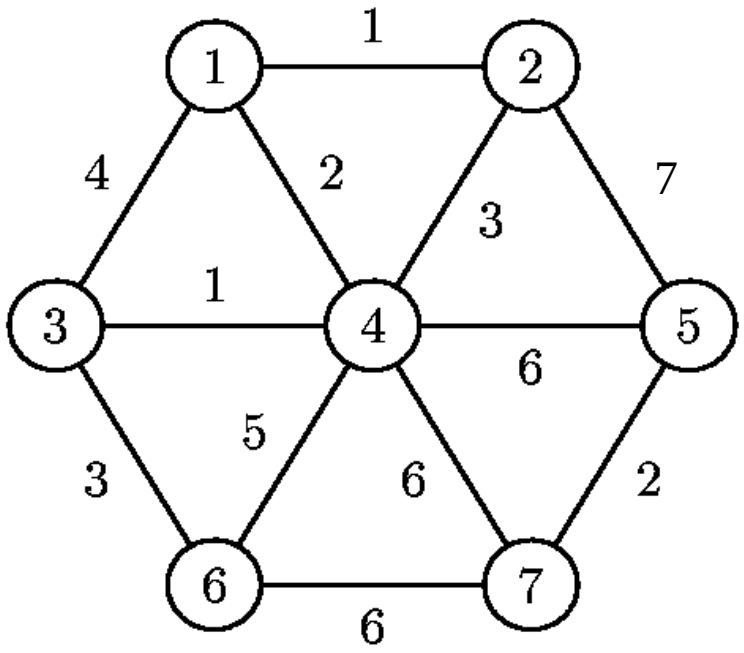
Άπληστος Αλγόριθμος για το Πρόβλημα Σακιδίου

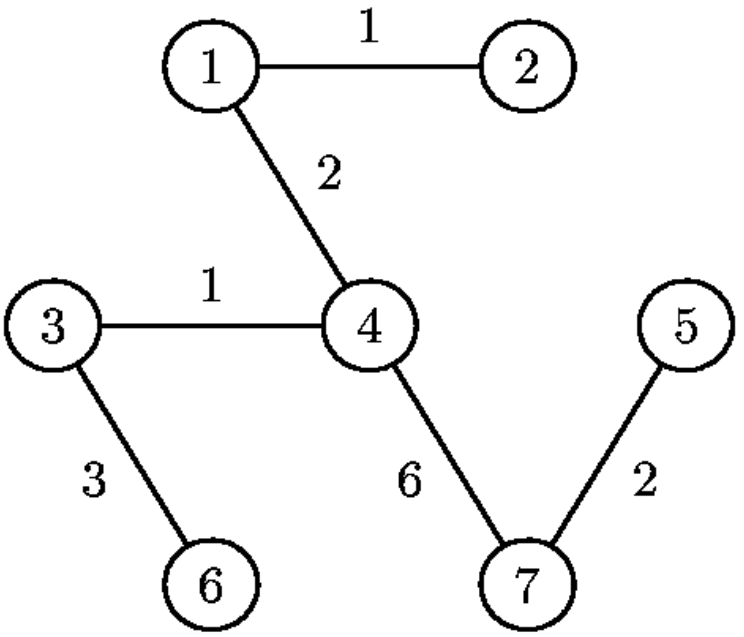
- Αν το άθροισμα όλων των βαρών είναι μικρότερο από Μ, θέσε x_i:=1 για όλα.

- Αλλιώς, διάτρεξε τα αντικείμενα κατά φθίνουσα σειρά p_i/w_i , παίρνοντας τη μέγιστη δυνατή ποσότητα.

- Πολυπλοκότητα: O(n logn)

Συνδετικό Δένδρο Ελαχίστου Κόστους (Minimum Cost Spanning Tree)





Δύο μέθοδοι

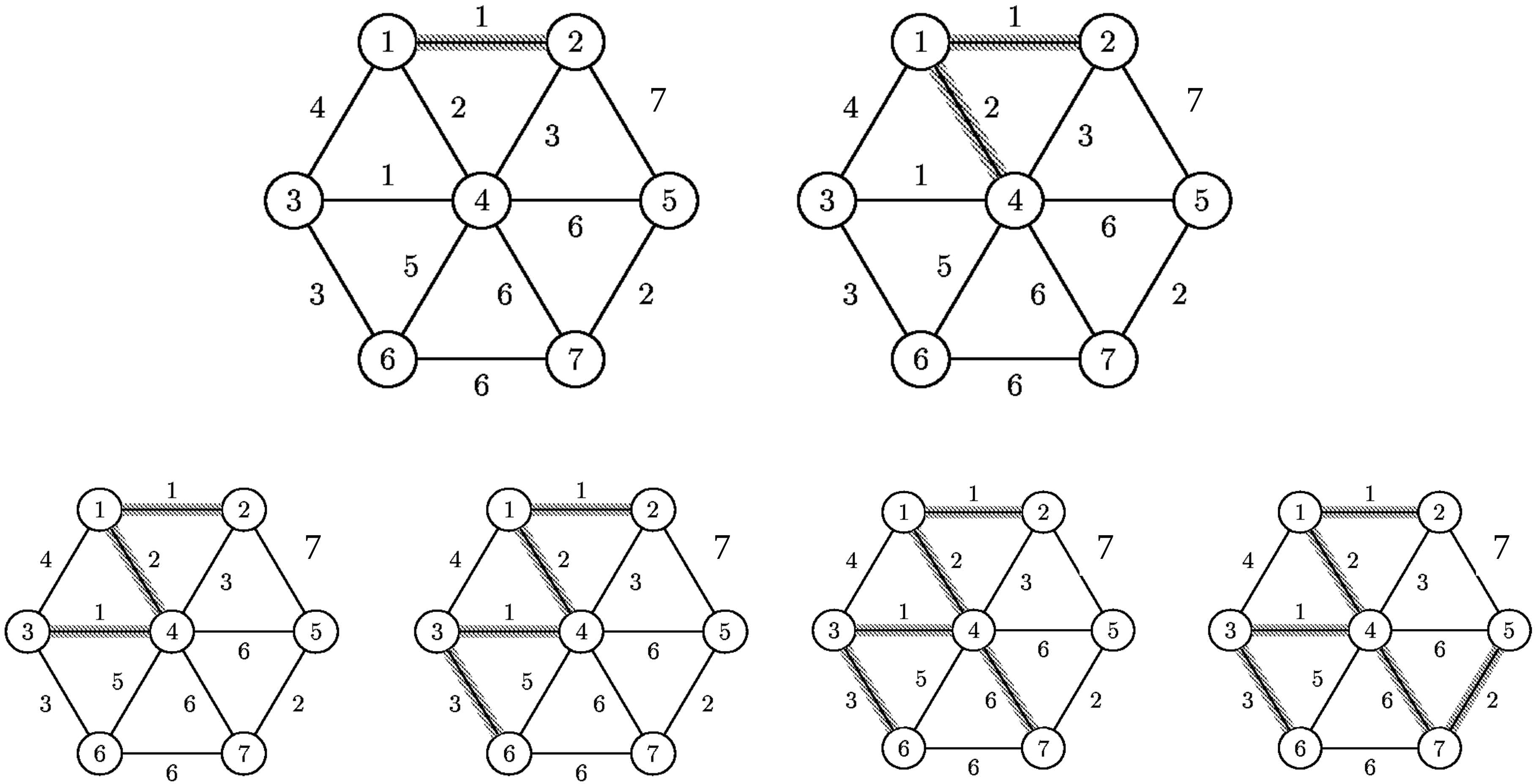
- Κριτήριο Prim: *επιλογή πλευράς ελαχίστου κόστους ώστε ο υπογράφος να παραμένει δέντρο*

- 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
     συνδετικό 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<>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
```

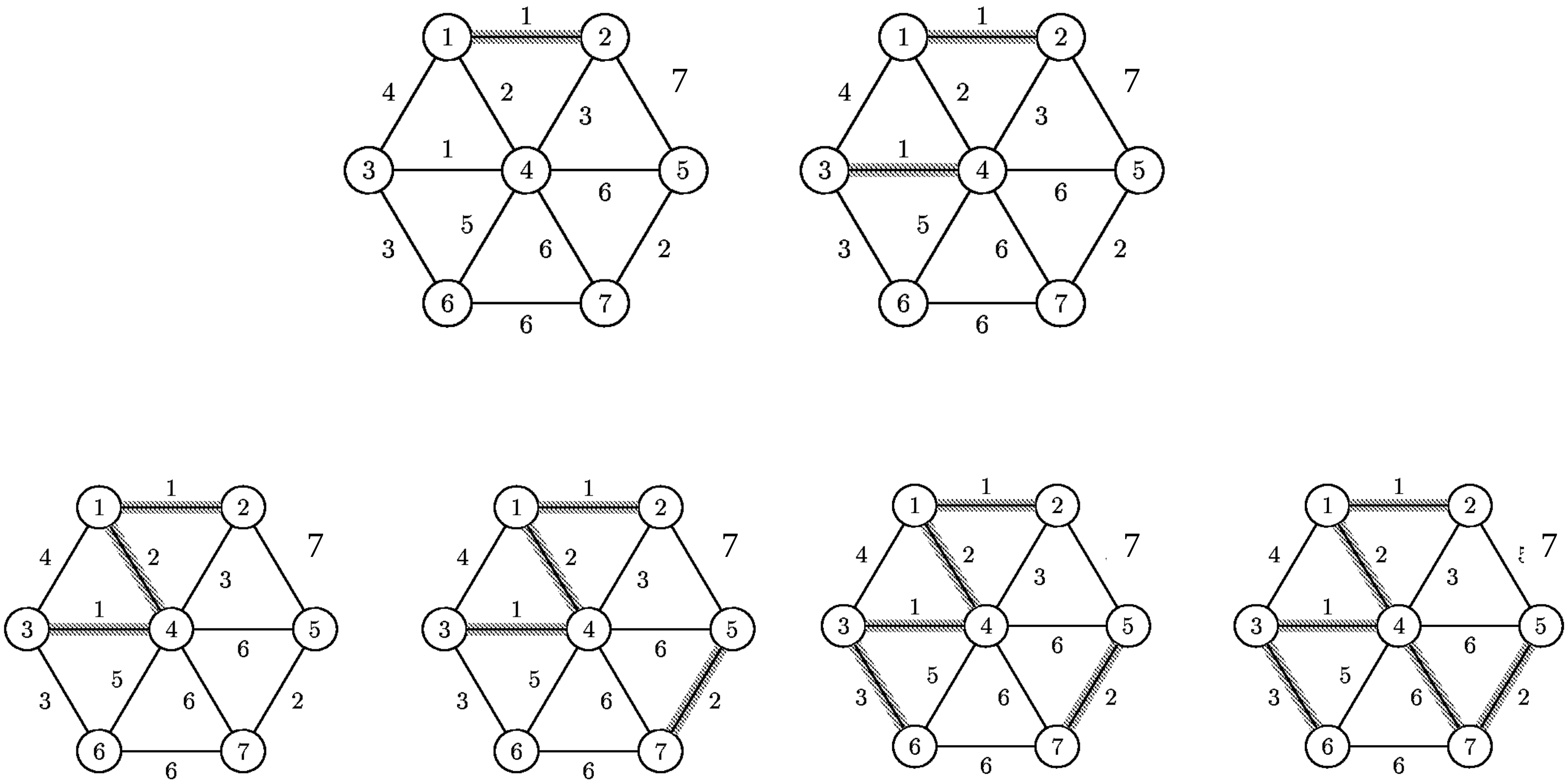
Παράδειγμα (με Prim)



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
```

Παράδειγμα (με Kruskal)



Πολυπλοκότητα - Βελτιστότητα

- Έστω n=|V|, e=|E|
- Prim: $O(n^2)$
- Kruskal: O(e logn)
- Συνεκτικοί γράφοι:

$$n-1 <= e <= n(n-1)/2$$

- Ο αλγόριθμος Kruskal καλύτερος σε αραιούς (dense) γράφους
- Θεώρημα: ο Kruskal δίνει βέλτιστη λύση

Ορθότητα Kruskal

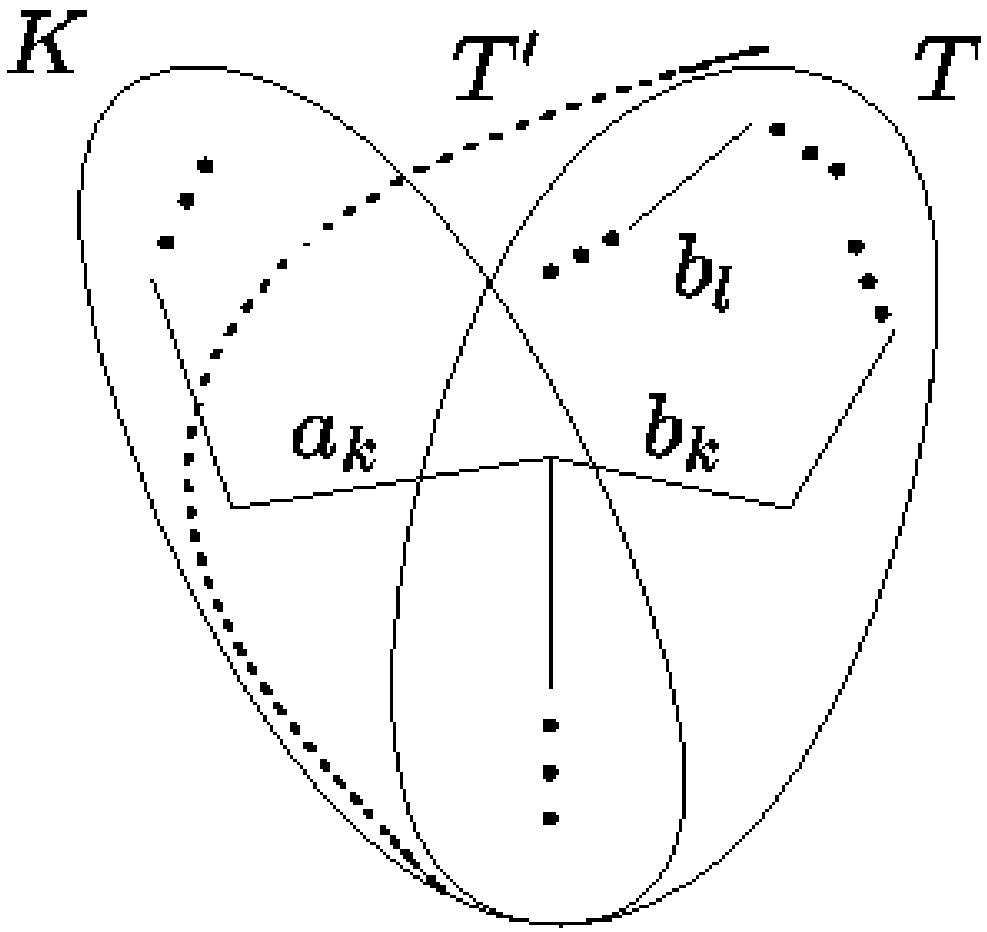
Θεώρημα 8.4.1. *Η λύση που δίνει ο αλγόριθμος του Kruskal στο πρόβλημα του συνδετικού δέντρου είναι η βέλτιστη.*

Απόδειξη. Έστω $a_1, a_2, \ldots, a_n$ οι ακμές σε αύξουσα σειρά βάρους στο δέντρο K, το οποίο προκύπτει από τον αλγόριθμο του *Kruskal*. Έστω $b_1, b_2, \ldots, b_n$ 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.

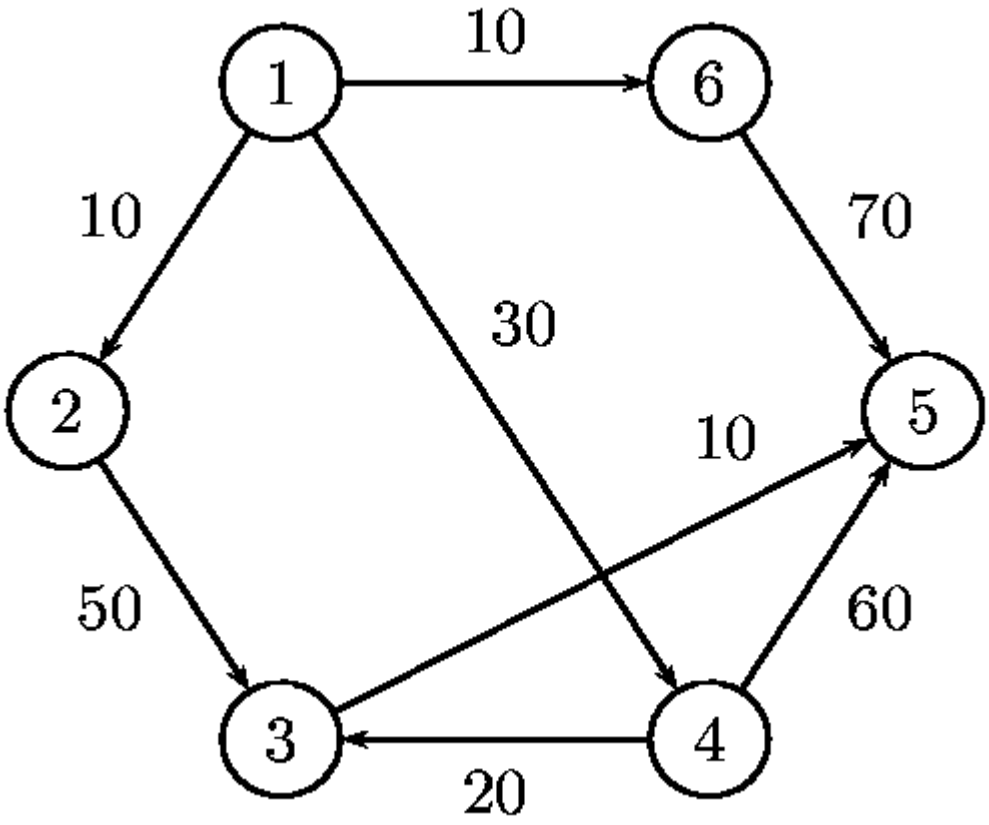
Ορθότητα Kruskal (συν.)



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. □

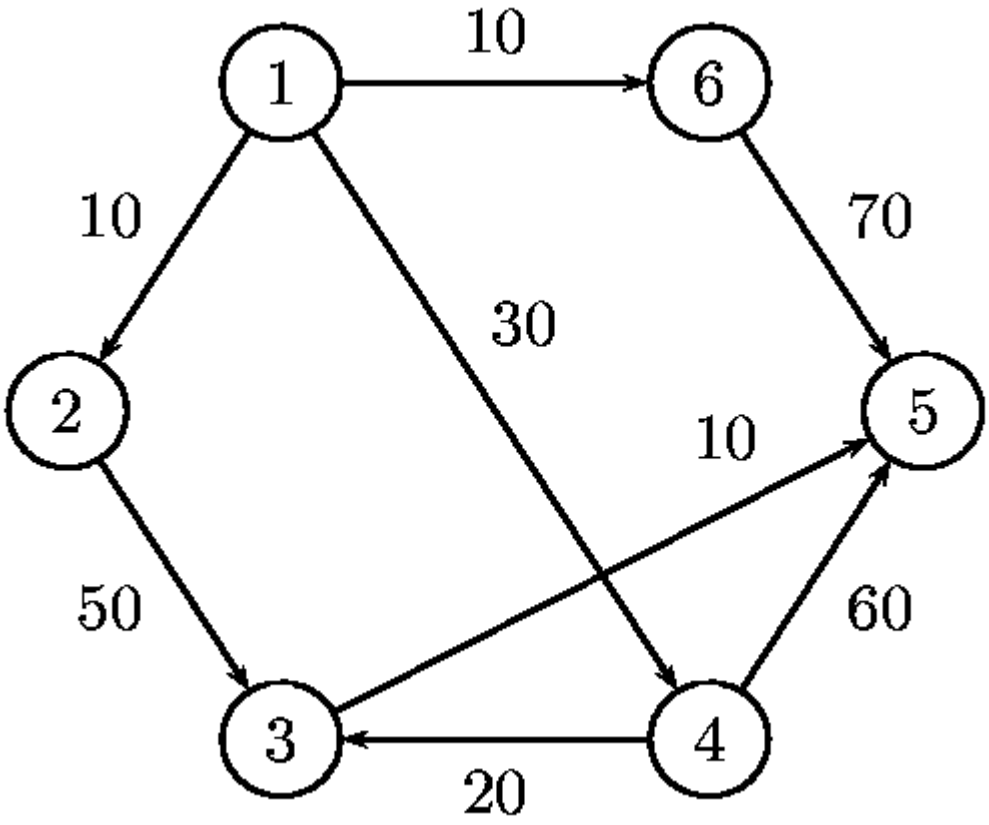
Πρόβλημα Συντομότερων Μονοπατιών (Single Source Shortest Paths)



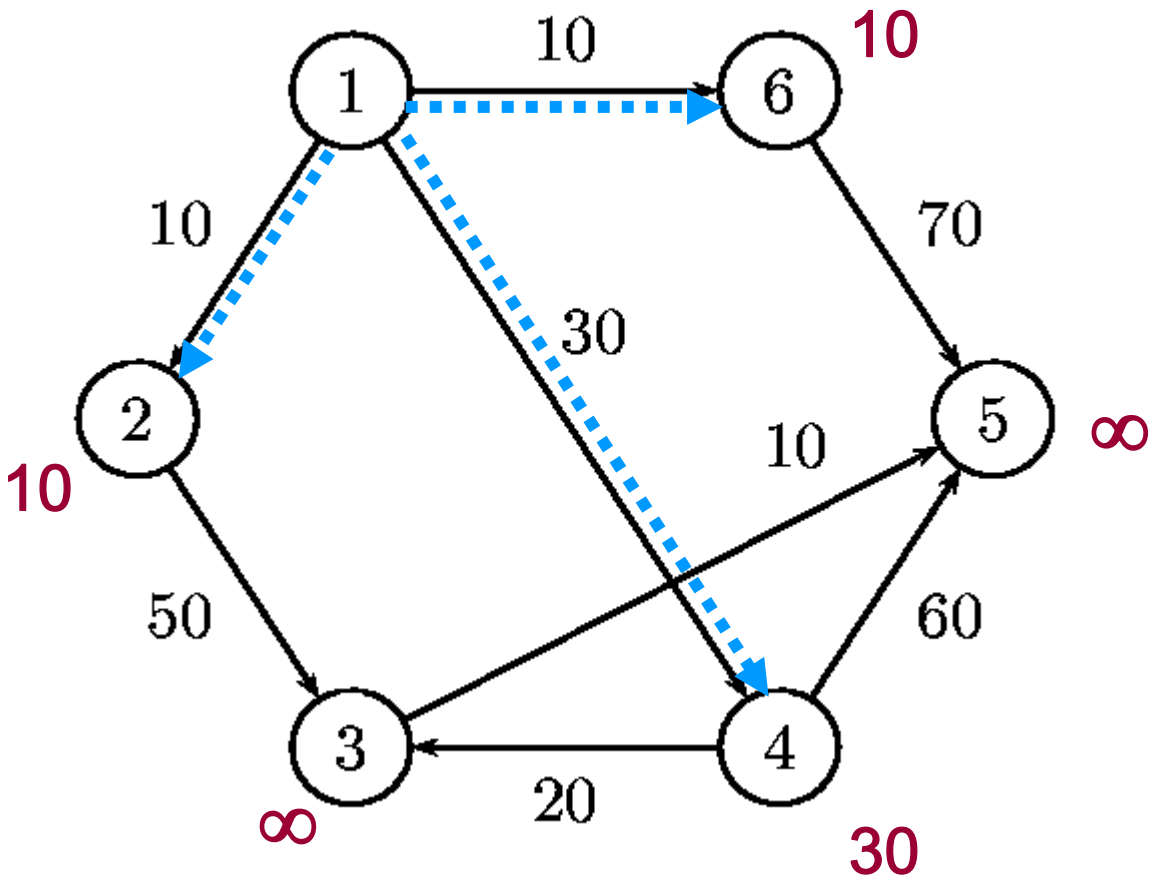
Αλγόριθμος του Dijkstra

- Δημιουργεί δέντρο ελαχίστων αποστάσεων από τον κόμβο 1 επαναλαμβάνοντας τα παρακάτω βήματα n-1 φορές:
- Σε κάθε επανάληψη επιλέγεται ο κόμβος w με την ελάχιστη απόσταση από τον κόμβο 1 (θεωρώντας μόνο διαδρομές που περνούν από κόμβους του ήδη σχηματισμένου δέντρου) και προστίθεται στο δέντρο.
- Ενημερώνονται οι αποστάσεις που επηρεάζονται λόγω της προσθήκης του w.

Αλγόριθμος του Dijkstra

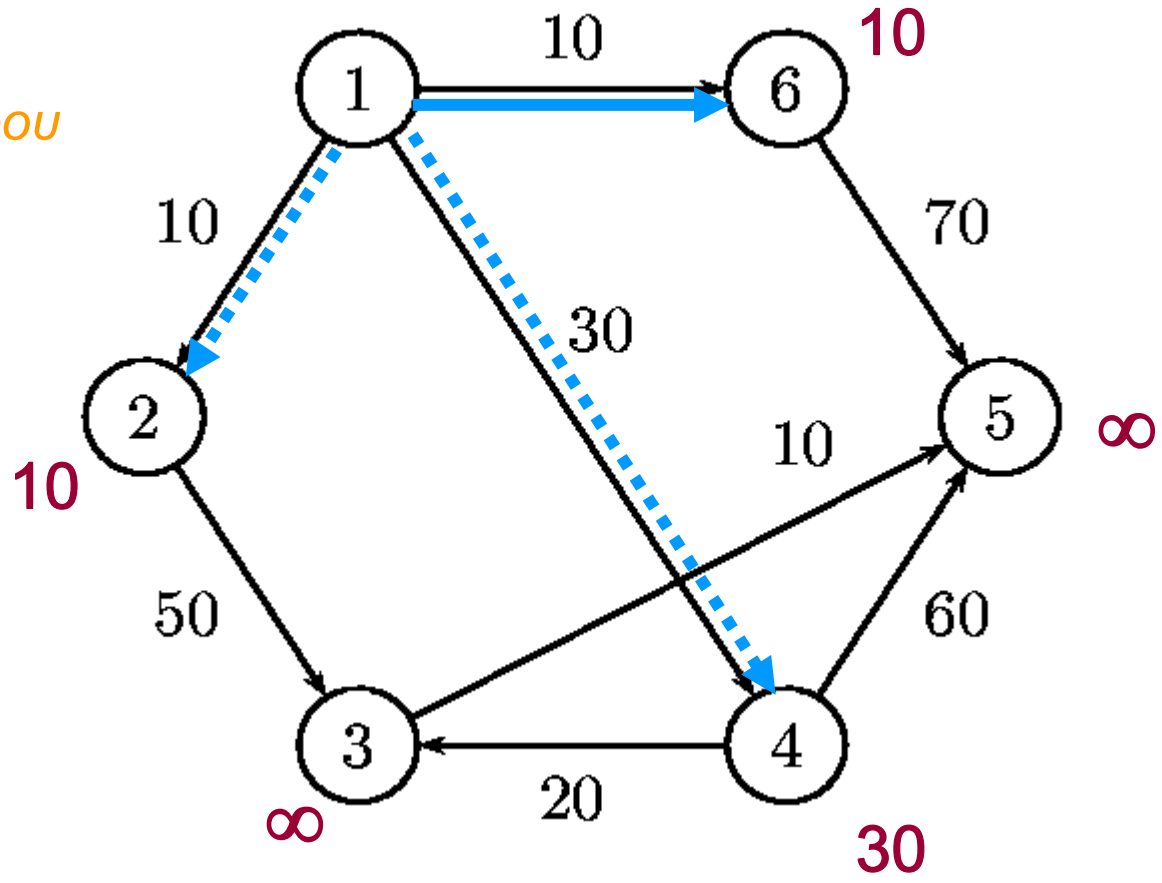


Αλγόριθμος του Dijkstra



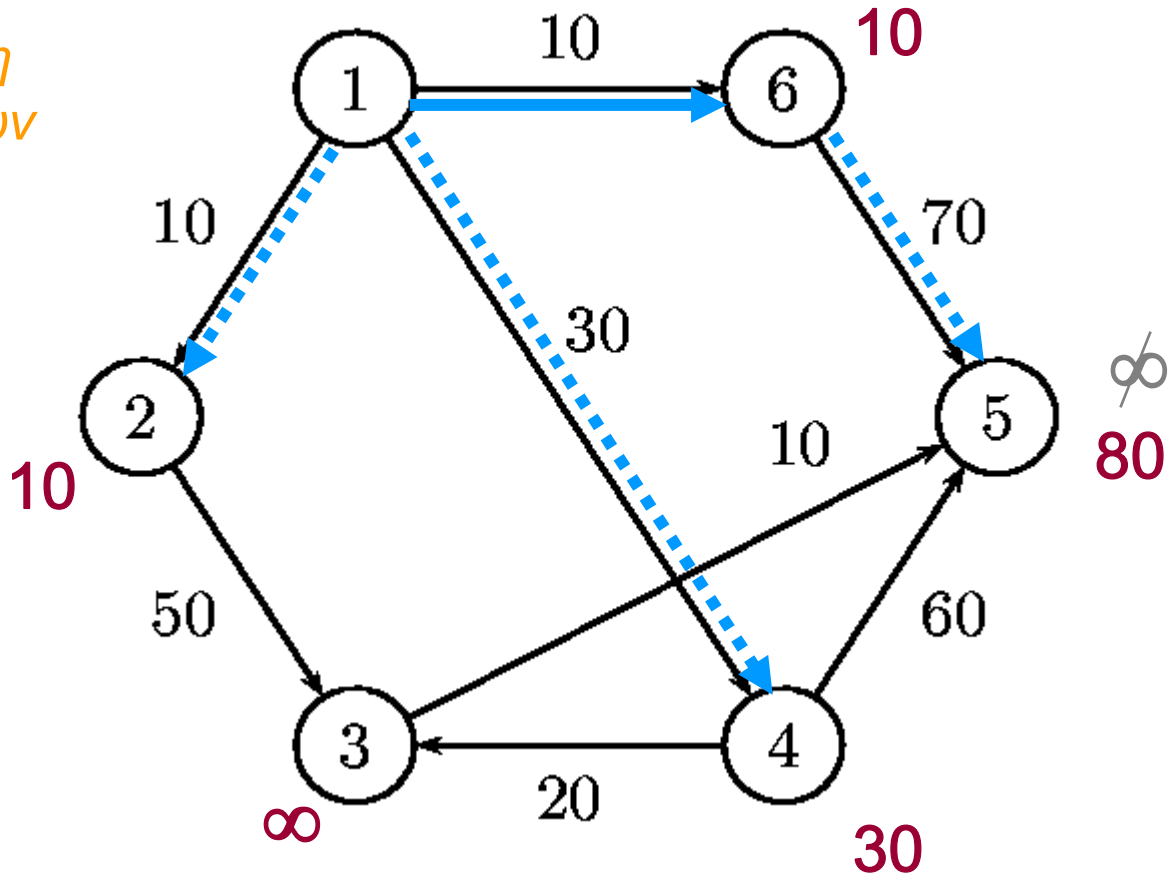
Αλγόριθμος του Dijkstra

Επιλογή πλησιέστερου κόμβου

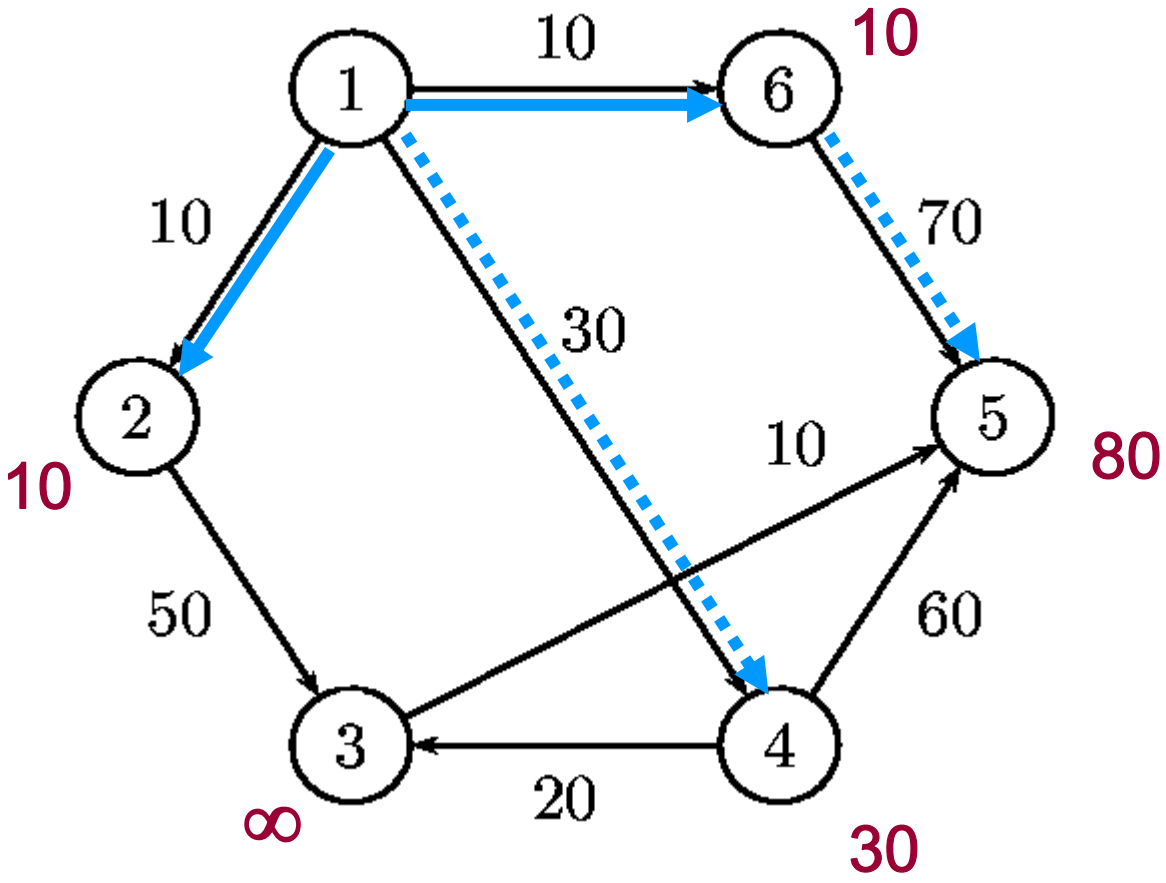


Αλγόριθμος του Dijkstra

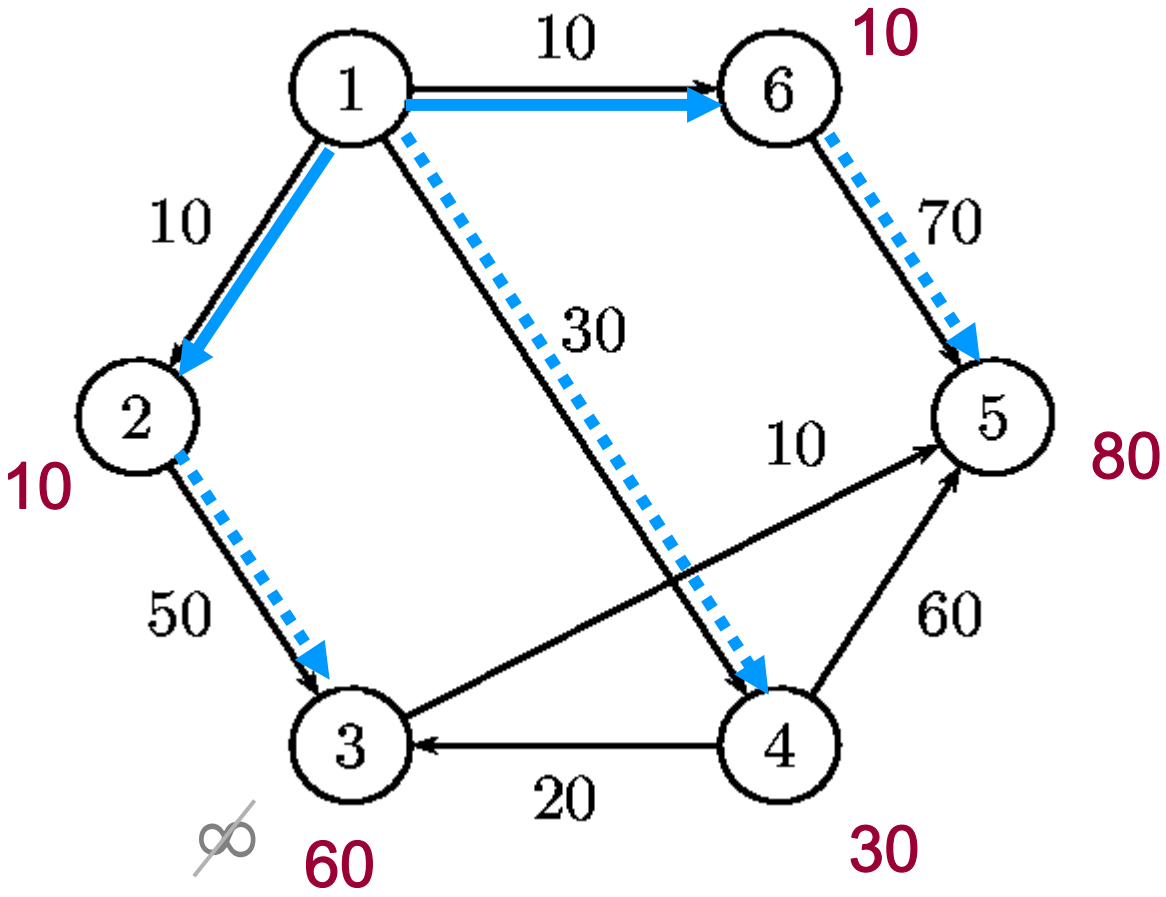
Ενημέρωση αποστάσεων



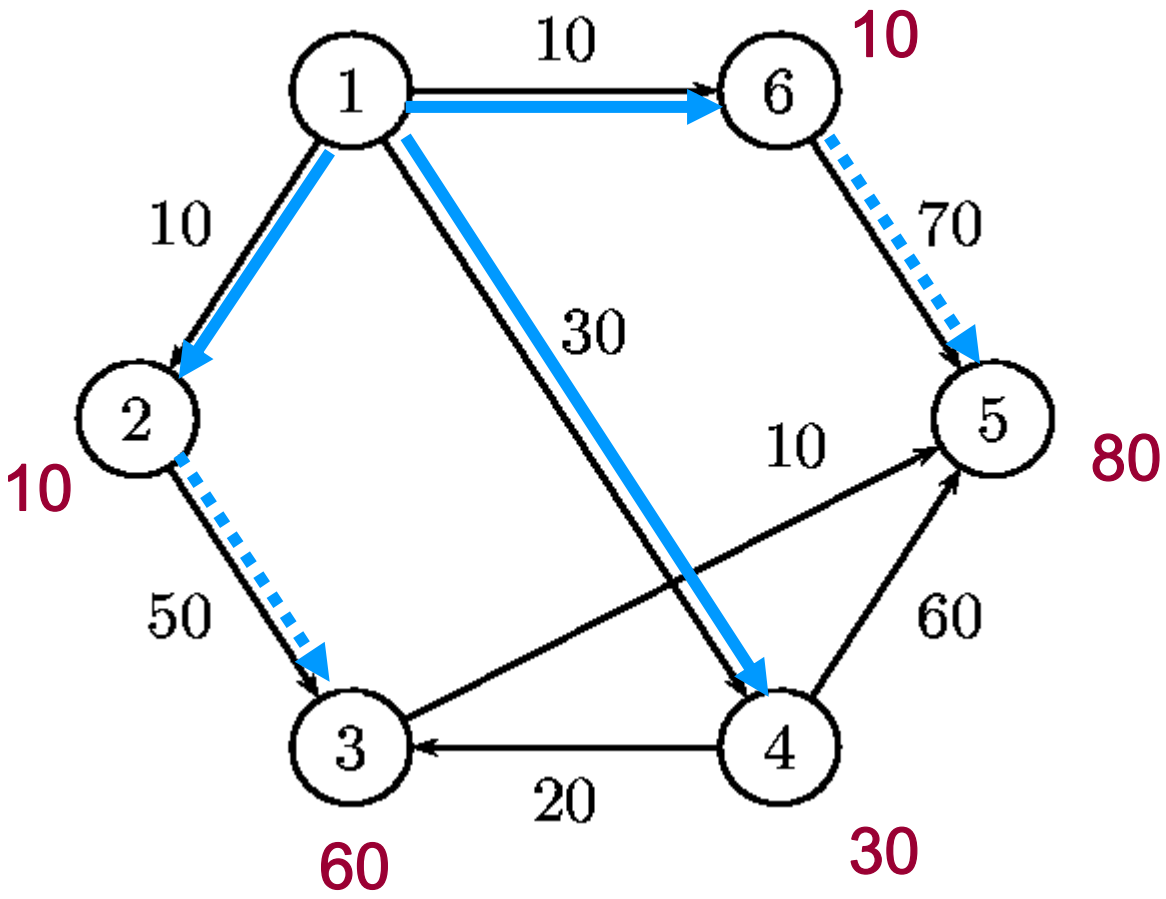
Αλγόριθμος του Dijkstra



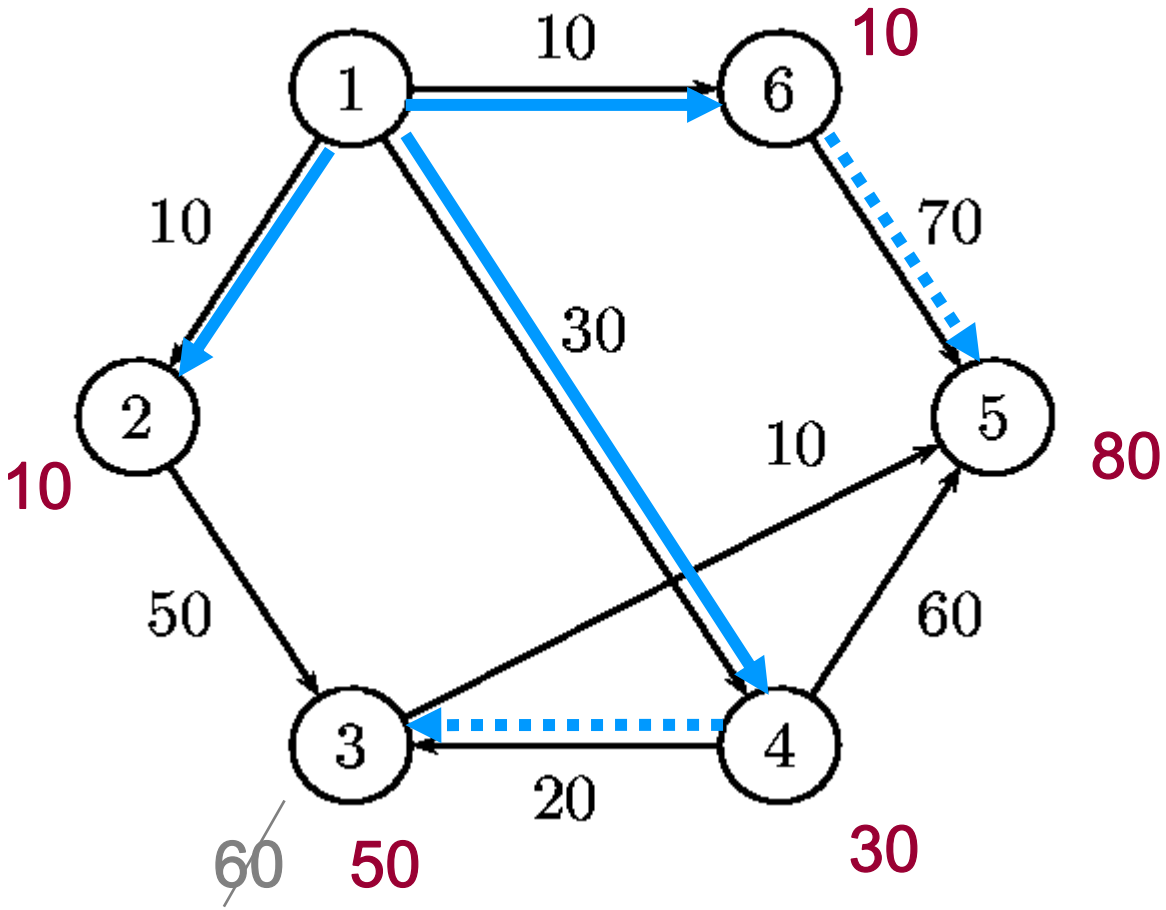
Αλγόριθμος του Dijkstra



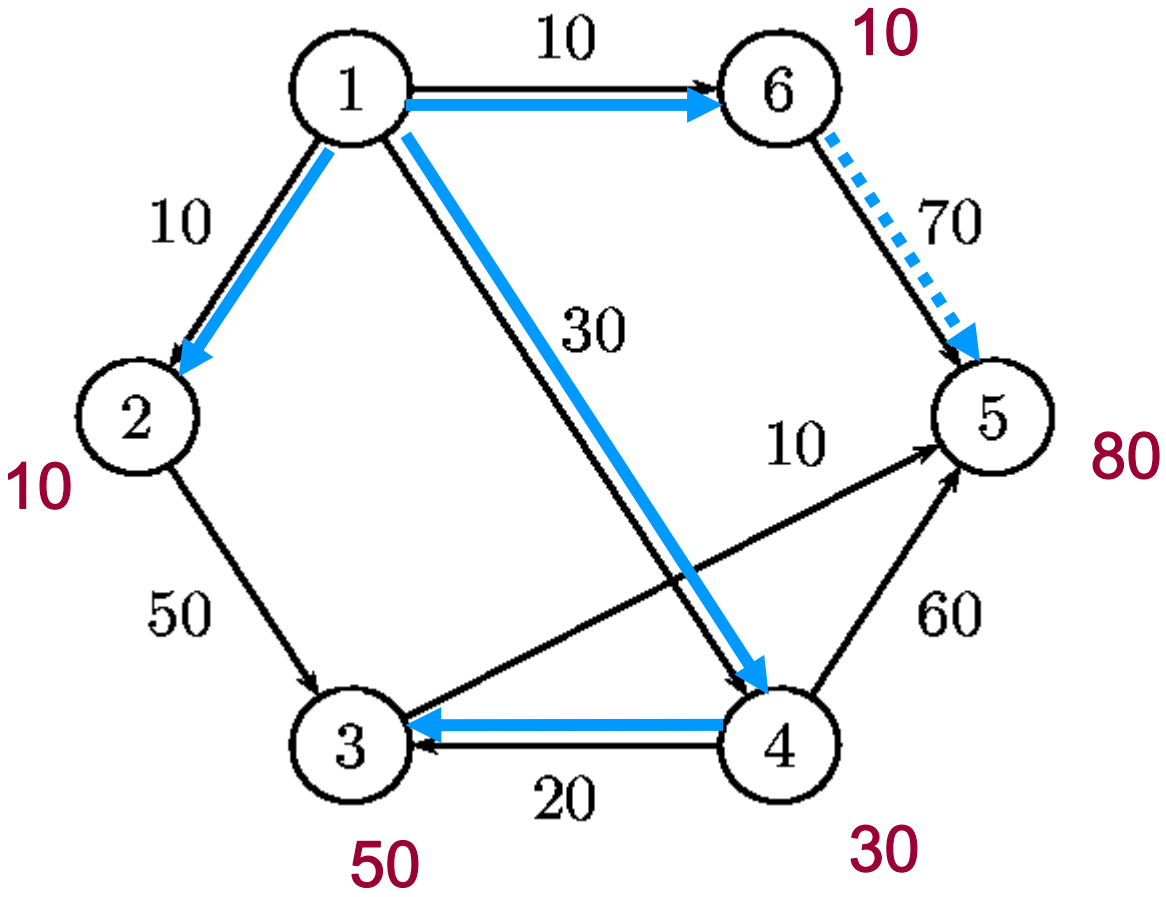
Αλγόριθμος του Dijkstra



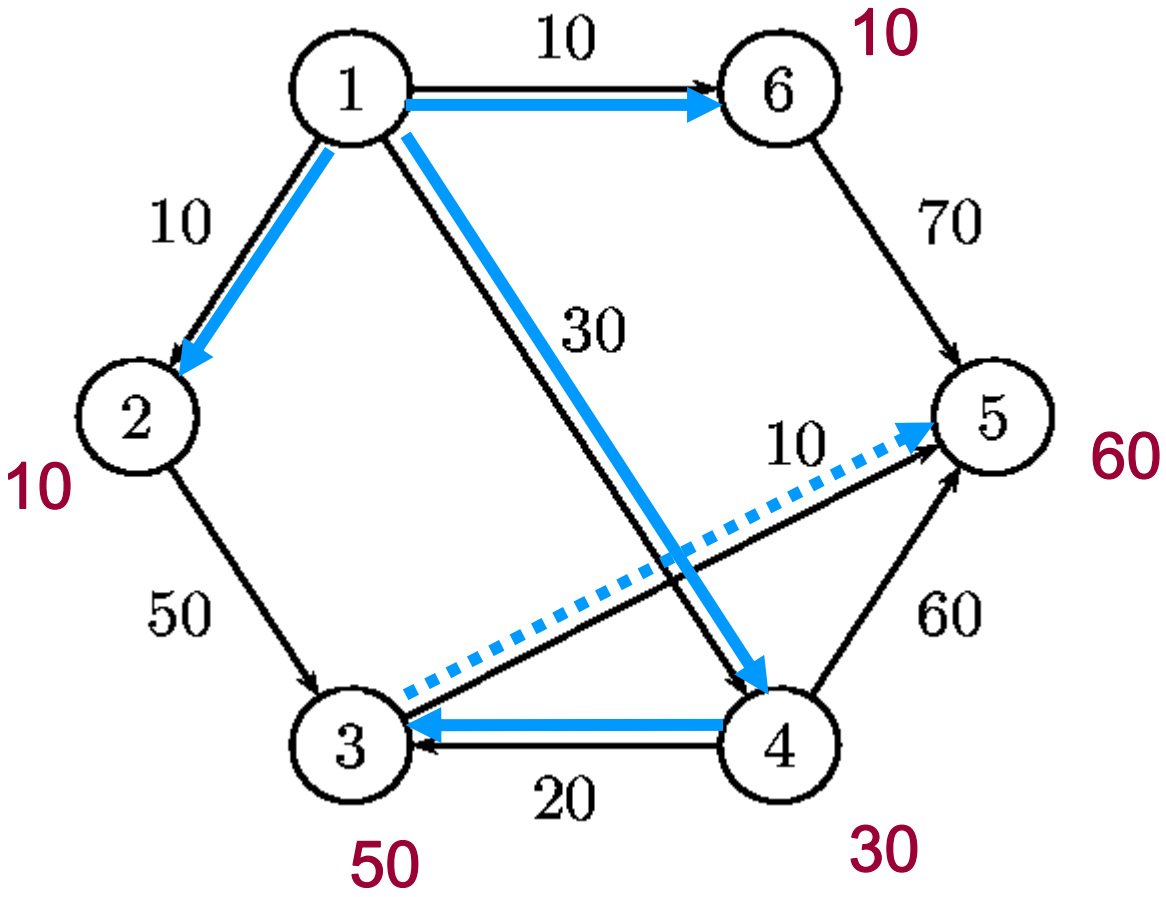
Αλγόριθμος του Dijkstra



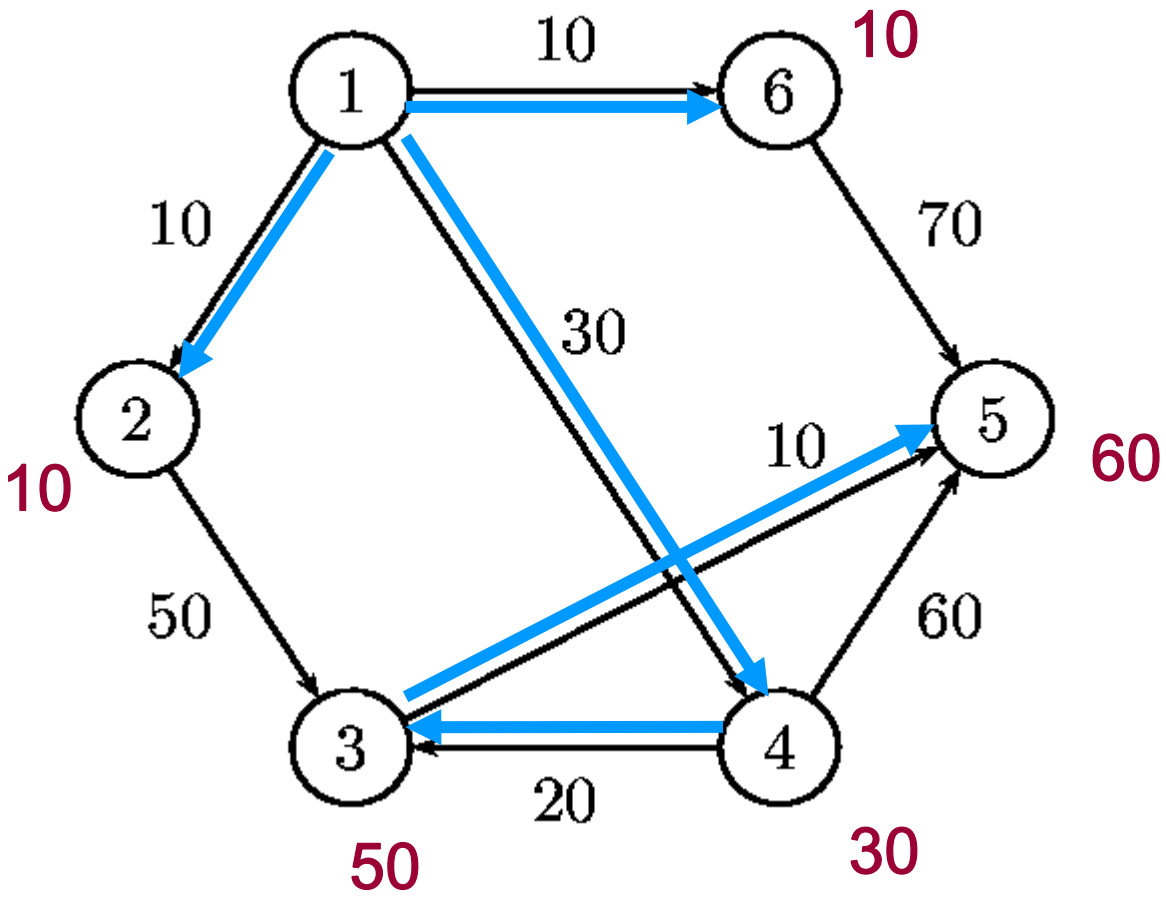
Αλγόριθμος του Dijkstra



Αλγόριθμος του Dijkstra



Αλγόριθμος του Dijkstra



Αλγόριθμος Dijkstra (συν.)

```
procedure Dijkstra;
begin (* Αρχικοποίηση *)
  S := {1};
  for i:=2 to n do begin D[i]:=cost[1,i]; P[i]:=1 end;
  for i:=2 to n−1 do
  begin
    Select w from V − S such that D[w] is minimum;
    S := S + {w};
    for all v in V − S do
    begin
      if D[v] > D[w] + C[w,v] then
      begin
        P[v] := w; D[v] := D[w] + C[w,v]
      end
    end
  end
end
```

Παράδειγμα Εκτέλεσης Dijkstra (με πίνακα)

Βήμα	S	w	D					P				
			2	3	4	5	6	2	3	4	5	6
-	{1}	-	10	∞	30	∞	10	1	1	1	1	1
2	{1,2}	2		60	30	∞	10		2			
3	{1,2,6}	6		60	30	80					6	
4	{1,2,6,4}	4		50		80			4			
5	{1,2,6,4,3}	3				60					3	
6	{1,2,6,4,3,5}	5										

Αλγόριθμος Bellman-Ford

Εκτελείται σε $|V| - 1$ στάδια.

Στο στάδιο i ενημερώνεται κάθε κόμβος v (που βρίσκεται σε απόσταση το πολύ i ακμών από τον αρχικό) με το συντομότερο μονοπάτι από τον s στον v που έχει το πολύ i ακμές.

Αυτό επιτυγχάνεται με εκτέλεση για κάθε ακμή $(w, v) \in E$ της εντολής:

if $D[v] > D[w] + C[w, v]$ then
begin
  $P[v] := w;$
  $D[v] := D[w] + C[w, v]$
end

Πολυπλοκότητα: $O(|V||E|)$

Παρατήρηση: δεν δουλεύει αν υπάρχουν κύκλοι αρνητικού βάρους. (Μπορεί όμως να τους εντοπίζει με κατάλληλη τροποποίηση.)

Πρόβλημα Μέγιστης Ροής (max flow)

Δίνεται ένα δίκτυο (network): κατευθυνόμενος γράφος με βάρη που αντιπροσωπεύουν χωρητικότητες, και δύο κόμβοι του s και t.

Ζητείται να βρεθεί ποια είναι η μέγιστη ροή που μπορεί να δρομολογηθεί από τον s στον t.

Αλγόριθμος Ford-Fulkerson

- Επιλογή ενός μονοπατιού από s σε t.
- Δρομολόγηση όσο το δυνατόν μεγαλύτερης ροής στο μονοπάτι (ελάχιστη ακμή).
- Κατασκευή εναπομείναντος δικτύου (residual network).
- Επανάληψη διαδικασίας εάν υπάρχει ακόμη μονοπάτι από s σε t.

Αλγόριθμος Ford-Fulkerson (συν.)

Κατασκευή εναπομείναντος δικτύου (residual network):

- Αφαίρεση ροής από χωρητικότητες ακμών (διαγραφή ακμής αν χωρητικότητα = 0).
- Αύξηση της χωρητικότητας στην αντίθετη κατεύθυνση κατά την ίδια ποσότητα (δημιουργία ακμής αν δεν υπάρχει).

Αλγόριθμος Ford-Fulkerson

Ορολογία: Τα μονοπάτια που χρησιμοποιεί ο αλγόριθμος λέγονται συνήθως μονοπάτια επαύξησης (augmenting paths).

Πολυπλοκότητα: $O(|f^*||E|)$, f^* η μέγιστη ροή.

Βελτιώσεις: Αλγόριθμος Edmonds-Karp $O(|V||E|^2)$ (shortest paths), αλγόριθμος Goldberg $O(|V|^2|E|)$ και $O(|V|^3)$ (preflow-push).

Θεώρημα Max Flow – Min Cut

Η μέγιστη ροή ισούται με την ελάχιστης χωρητικότητας τομή (σύνολο ακμών) που διαχωρίζει τον κόμβο s από τον κόμβο t.

Ο αλγόριθμος Ford-Fulkerson επιτυγχάνει τη μέγιστη ροή.