Διάλεξη 22: Γράφοι ΙΙΙ - Ελάχιστα Γεννητορικά Δέντρα

Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:

- Ελάχιστα Γεννητορικά Δένδρα (ΕΓΔ) – Minimum Spanning Trees

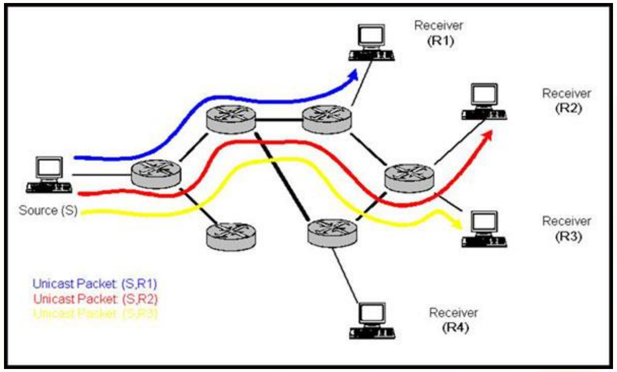
- Ο αλγόριθμος του Prim για εύρεση ΕΓΔ σε γράφους

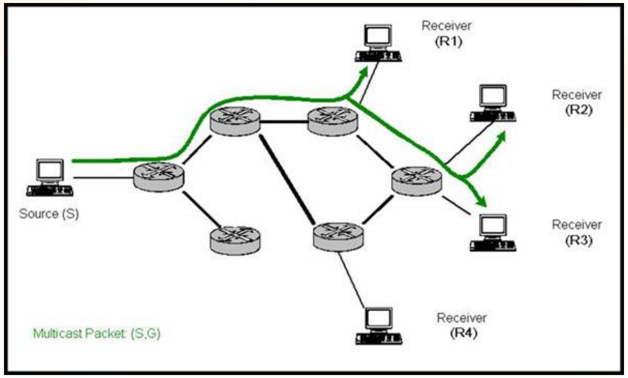
- Ο αλγόριθμος του Kruskal για εύρεση ΕΓΔ σε γράφους

- Παραδείγματα Εκτέλεσης

Ελάχιστα Γεννητορικά Δένδρα (ΕΓΔ) - Το πρόβλημα

- Υποθέστε ότι έχουμε το weighted γράφο G(V,E), ο οποίος εκφράζει τις συνδέσεις σε ένα δίκτυο (το βάρος κάθε ακμής εκφράζει κάποιο κόστος – π.χ. καθυστέρηση μετάδοσης).
- Επίσης υποθέστε ότι θέλουμε να στείλουμε από ένα κόμβο (server) ένα video stream στις υπόλοιπες τερματικές κορυφές του γράφου.
- Ένας τρόπος θα ήταν να στείλουμε ένα video stream ανά κορυφή παραλήπτη (unicast) …Όμως αυτό θα ήταν πολύ ακριβό.
- Ιδανικά θα θέλαμε να φτιάξουμε ένα μονοπάτι (δένδρο) προς όλους τους τερματικούς κόμβους έτσι ώστε το συνολικό άθροισμα των ακμών να είναι ελάχιστο.
- Ένα τέτοιο δένδρο θα μας επέτρεπε να στείλουμε την ταινία προς όλους με το ελάχιστο κόστος. Σήμερα θα μελετήσουμε τέτοια Ελάχιστα Γεννητορικά Δένδρα



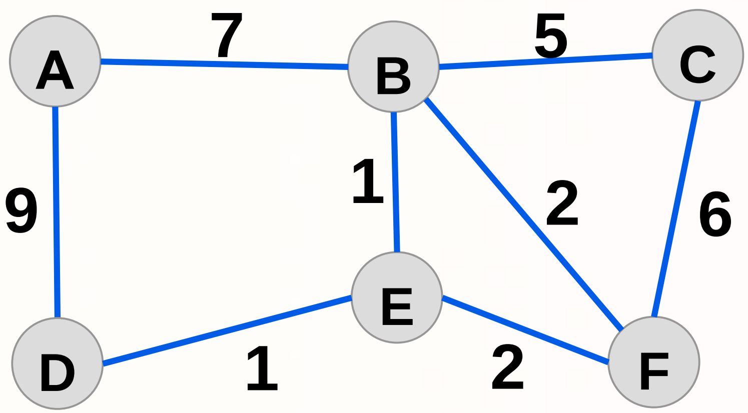


Ελάχιστα Γεννητορικά Δένδρα (ΕΓΔ)

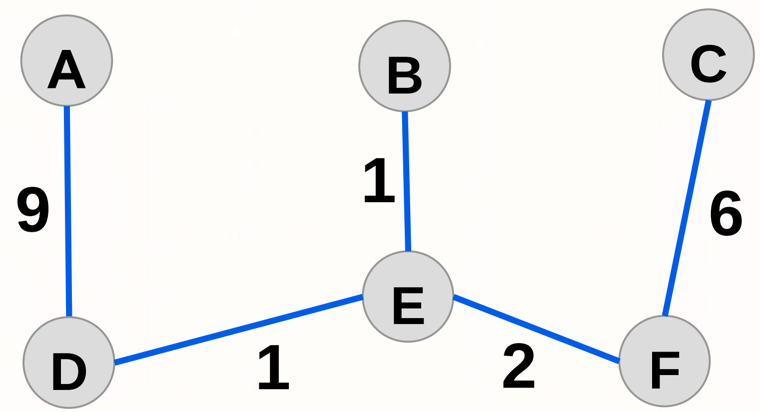
- Έστω ένας **μη-κατευθυνόμενος** γράφος με βάρη, G(V,E).
- **Γεννητορικό δένδρο (spanning tree, ΓΔ)** του **G** ονομάζουμε κάθε **δένδρο T** που περιέχει όλους τους κόμβους του **G** και κάθε ακμή του οποίου είναι και ακμή του **G.**
- Σε ένα ΓΔ, όλες οι κορυφές '**καλύπτονται**', γι' αυτό το δένδρο ονομάζεται και δένδρο σκελετός (**spanning tree**: the tree spans all the vertices)
- Ένα γεννητορικό δένδρο γράφου με **n** κορυφές έχει **n-1** ακμές.
- **Βάρος ενός ΓΔ** είναι το άθροισμα των βαρών όλων των ακμών του
- **Ελάχιστο ΓΔ (ΕΓΔ)** είναι το ΓΔ με το μικρότερο βάρος. Ένας γράφος δυνατό να έχει περισσότερα από ένα ΕΓΔ. (Εάν ο γράφος δεν είχε βάρη τότε οποιονδήποτε δένδρο που ενώνει όλες τις ακμές = ΕΓΔ)
- Το πρόβλημα εύρεσης ενός ΓΔ μπορεί να εκφραστεί και για κατευθυνόμενους γράφους αλλά είναι κάπως δυσκολότερη η υλοποίηση

Παραδείγματα Γεννητορικών Δένδρων

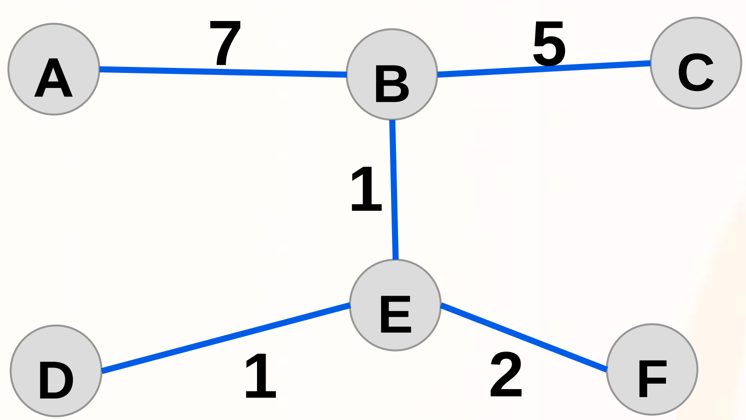
Γράφος G



ΓΔ-1: Βάρος=19



ΓΔ-2: Βάρος=16



Κατ’ ακρίβεια αυτό το δένδρο είναι ένα Ελάχιστο ΓΔ (όπως θα δούμε στην συνέχεια)

Υπάρχουν άλλα;

Ιδιότητες ΕΓΔ

- Υποθέτουμε στην συνέχεια ότι οι γράφοι που μελετάμε είναι **συνεκτικοί** (δηλαδή υπάρχει τουλάχιστο μια διαδρομή μεταξύ όλων των κορυφών).

- Δεν κάνει νόημα να βρούμε ένα ΕΓΔ ενός μη-συνεκτικού γράφου … γιατί ο ορισμός του ΕΓΔ προϋποθέτει ότι το δένδρο καλύπτει όλες τις κορυφές.

- Η εύρεση ΕΓΔ είναι γνωστό και βαθιά μελετημένο πρόβλημα στην επεξεργασία γράφων. Έχει ποικίλες εφαρμογές.

Ο αλγόριθμός του Prim

- Αρχικά το δένδρο περιέχει ακριβώς μία κορυφή, η οποία επιλέγεται τυχαία.
- Για να κτίσουμε το δένδρο, σε κάθε βήμα συνδέουμε ακόμα μια κορυφή στο παρόν δένδρο με την επιλογή και εισαγωγή μιας καινούριας ακμής (από τις ακμές του γράφου).
- Πως μπορούμε να επιλέξουμε την κατάλληλη ακμή;
- Στην περίπτωση αυτού του αλγόριθμου, αν **S** είναι το ***σύνολο των κορυφών του παρόντος δένδρου***, επιλέγουμε

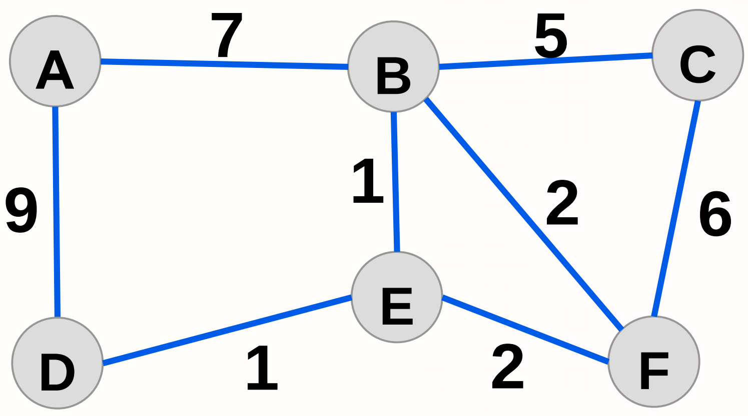
  Την ακμή με το μικρότερο βάρος,

  1. **Την ακμή η οποία μπορεί να μεγαλώσει το δένδρο κατά ένα κόμβο**
  2. **Την ακμή η οποία δεν θα δημιουργήσει κάποιο κύκλο**

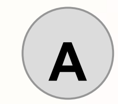
- Ο αλγόριθμος του Prim είναι ένας **Άπληστος Αλγόριθμος (Greedy Algorithm).** Σε κάθε βήμα κάνει την κίνηση που ικανοποιεί όλες τις συνθήκες και έχει το πιο λίγο κόστος.

Παράδειγμα Εκτέλεσης

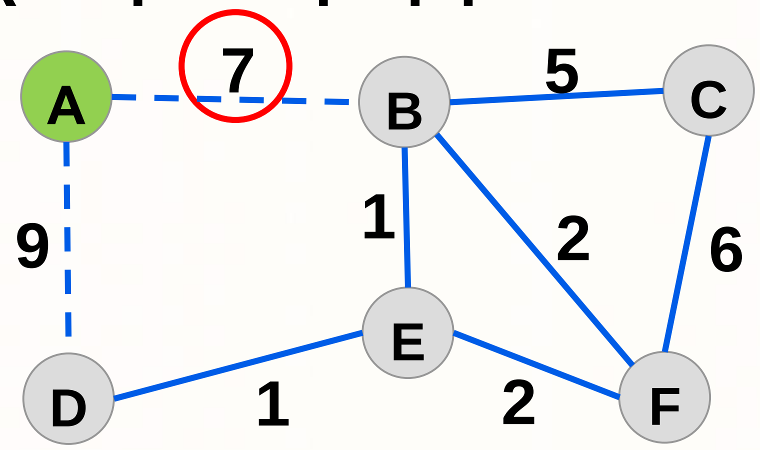
Γράφος G



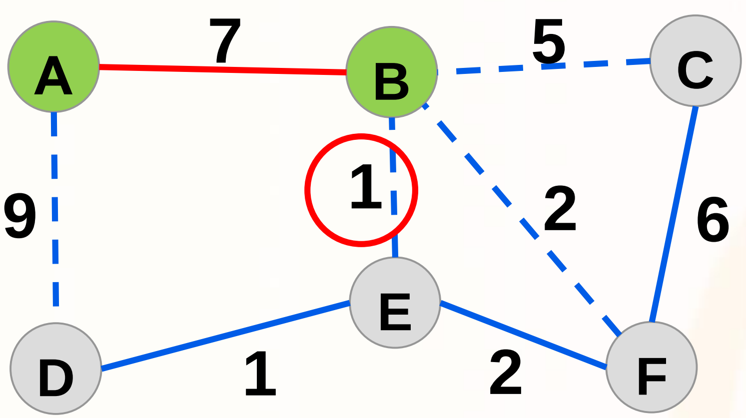
Ξεκινούμε διαλέγοντας τυχαία μια κορυφή



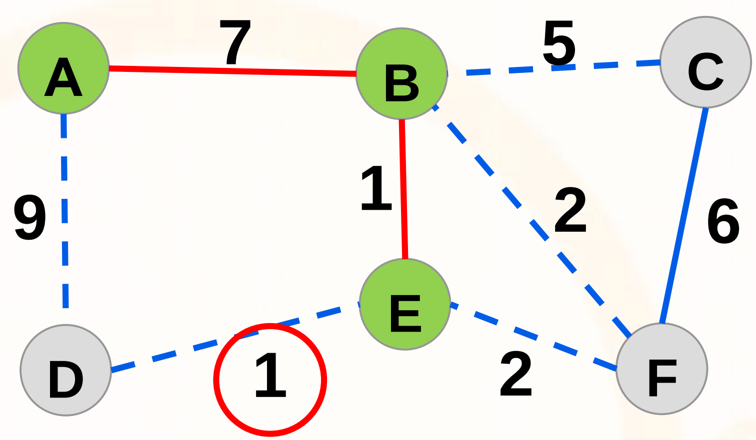
1



2

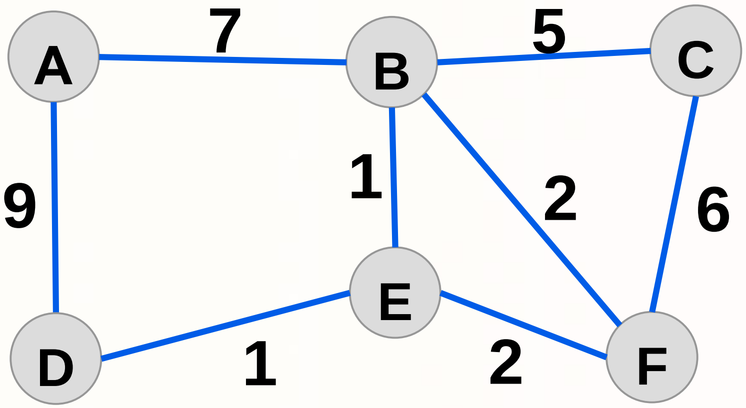


3

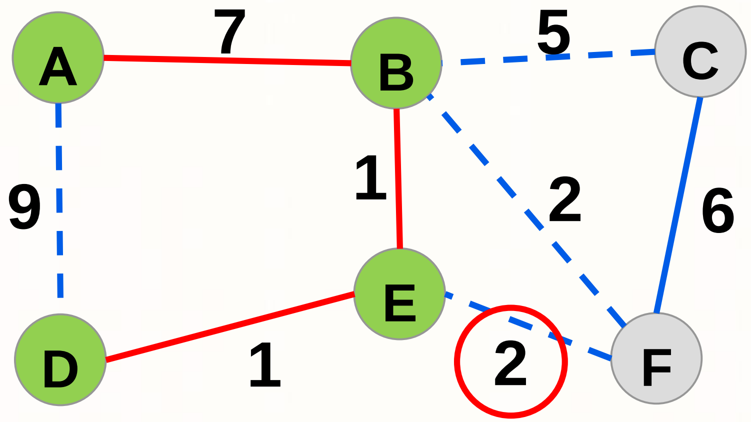


Παράδειγμα Εκτέλεσης (συν.)

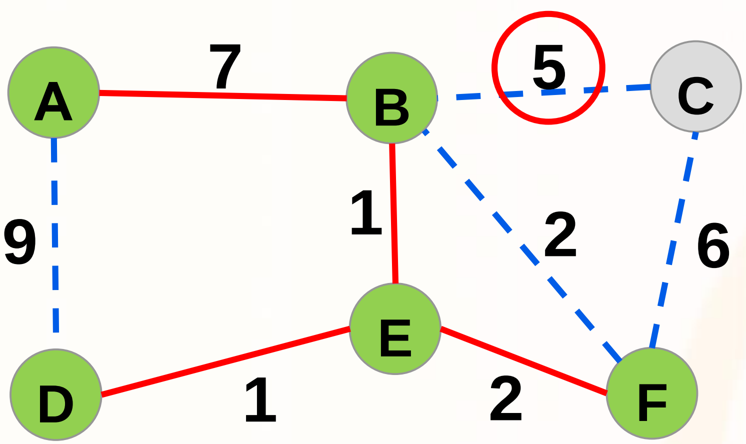
Γράφος G



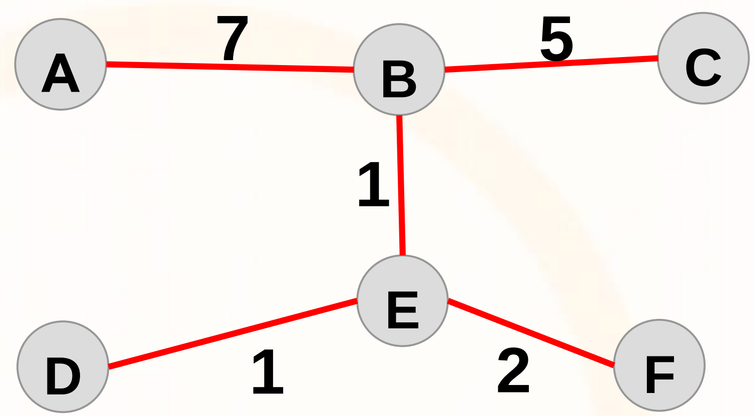
4



5

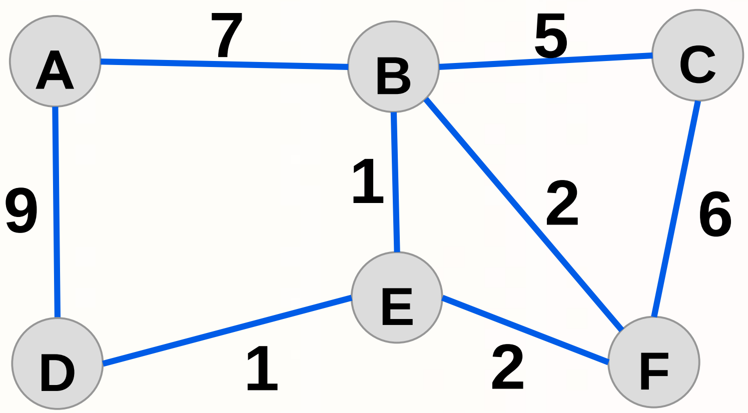


6

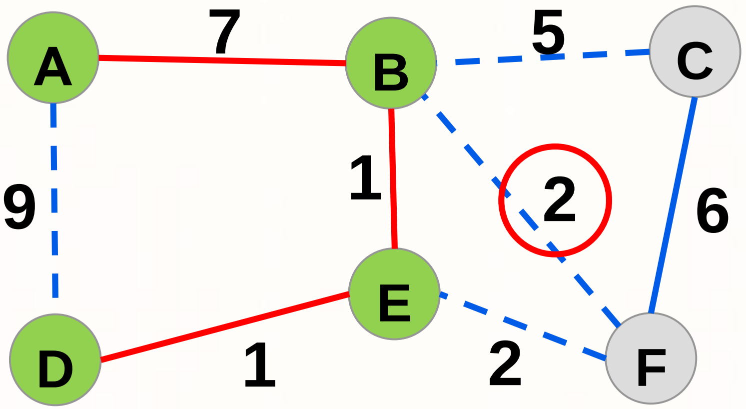


Παράδειγμα Εκτέλεσης – Εναλλακτική Λύση (συν.)

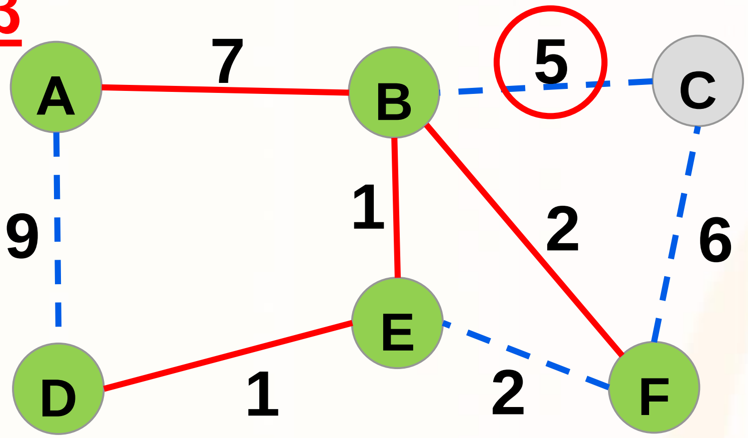
Γράφος G



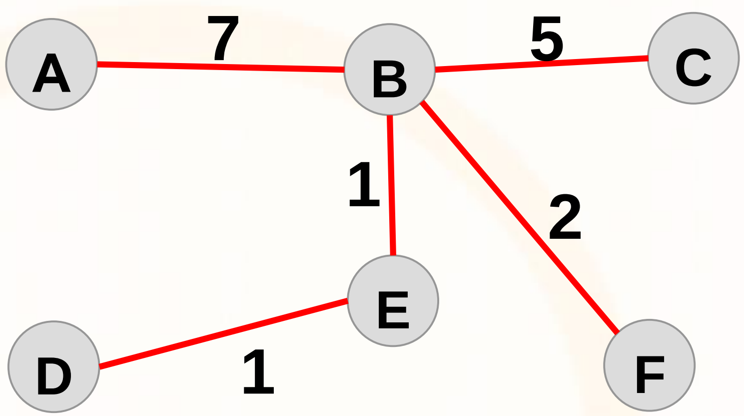
4β



5β



6β



Υλοποίηση Αλγόριθμου Prim

- Για να υλοποιήσουμε τον αλγόριθμο Prim θα χρησιμοποιήσουμε παράλληλους πίνακες
  - A) visited[n] : Κορυφές από τις οποίες περάσαμε.
  - B) closest[n] : Η κοντινότερη κορυφή κάθε κόμβου στο δένδρο (μια δεδομένη στιγμή)
  - C) distance[n] : Η απόσταση του κάθε επί μέρους κόμβου στο (B)

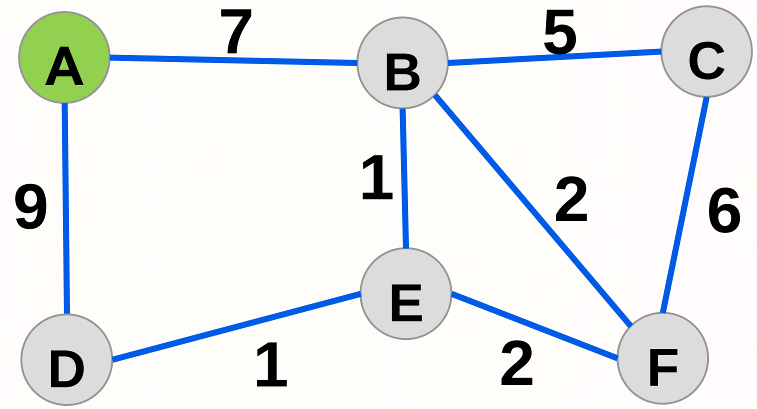
Αρχικοποίηση

	A	B	C	D	E	F
visited:	0	0	0	0	0	0
closest:	0	0	0	0	0	0
distance:	∞	∞	∞	∞	∞	∞

Υλοποίηση Αλγόριθμου Prim (συν.)

- Μέτα την εισαγωγή του **A** στο Δένδρο

	A	B	C	D	E	F
visited:	1	0	0	0	0	0
closest:	0	A	0	A	0	0
distance:	∞	7	∞	9	∞	∞



- Μέτα την εισαγωγή του **B** στο Δένδρο

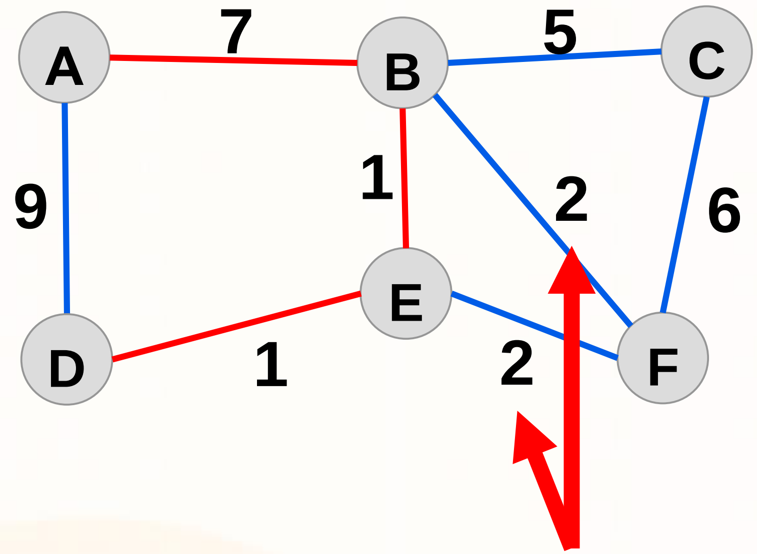
	A	B	C	D	E	F
visited:	1	1	0	0	0	0
closest:	0	A	B	A	B	B
distance:	∞	7	5	9	1	2

Προσπαθούμε να μεγαλώσουμε το δένδρο με άπληστο τρόπο (διατηρώντας το συνδεδεμένο)

Υλοποίηση Αλγόριθμου Prim (συν.)

- Μέτα την εισαγωγή του **E** στο Δένδρο

	A	B	C	D	E	F
visited:	1	1	0	0	1	0
closest:	0	A	B	E	B	**B**
distance:	∞	7	5	1	1	**2**



Δυο τρόποι να πάμε στο F, διατηρούμε τον ένα.

Η Υλοποίηση του Αλγόριθμου Prim

```
Prim(graph G){
int visited[n]={}; // Κορυφές που προστέθηκαν στο δένδρο (Αρχικά όλα "0")
int closest[n]={}; // «Πιο Κοντινός Γείτονας» για κάθε i: Αρχικά κανένας
int distance[n]=∞, // Απόσταση από «Κοντιν. Γείτονα» για κάθε i: Αρχικά άπειρο
Tree = {};    // Το ΕΓΔ που θέλουμε να κτίσουμε (Λίστα από ακμές (α,β))
// επιλογή αρχικής κορυφής
Διάλεξε τυχαία κορυφή v;
visited[v] = 1;      // Τώρα το v ανήκει στο δένδρο

για κάθε κορυφή v {
    // ενημέρωση πινάκων distance & closest
    για κάθε w γείτονα του v {
       if (G.weight(v,w) < distance[w]) {
          distance[w] = G.weight(v,w); // απόσταση κοντινότερου
          closest[w] = v; // ταυτότητα κοντινότερου
       }
     }
     // εύρεση επόμενης κορυφής με μικρότερη απόσταση)
     v = minVertex(visited, distance);
     visited[v]=1;                     // επιλογή κόμβου
     Tree = Tree ∪ {(closest[v],v)}; //προσθήκη ακμής
 }
}
```

weight(a,b) : βάρος ακμής a-b

Η βοηθητική συνάρτηση minVertex

- Η βοηθητική διαδικασία minVertex βρίσκει μεταξύ όλων των κορυφών που δεν προστέθηκαν στο MST (Minimum Spanning Tree) την πιο κοντινή κορυφή. Δηλαδή:

```
vertex minVertex(int visited[], int distance[]){
   min = ∞;
   for (i=0; i<|V|; i++) {
     if (visited[i] == 1) continue; // skip nodes already in MST
     if (distance[i] < distance[min]) min = i;
    }
   return min;       // Return the minimum among all distances
 }
```

	A	B	C	D	E	F
visited:	0	0	0	0	0	0
closest:	0	A	B	A	B	B
distance:	∞	7	5	9	1	2

Ανάλυση Χρόνου Εκτέλεσης

- Η διαδικασία **minVertex** απαιτεί χρόνο O(|V|), όπου |V| είναι ο αριθμός των κορυφών του γράφου.
- Ο χρόνος εκτέλεσης του βρόχου της εντολής **`while`** στον αλγόριθμο Prim είναι και αυτός O(|V|). (Και για υλοποίηση με πίνακα γειτνίασης και για υλοποίηση με λίστα γειτνίασης.)
- **Άρα ο ολικός χρόνος εκτέλεσης είναι Θ(|V|^2).**
- Μπορούμε να βελτιώσουμε τον αλγόριθμο;
- **Ναι** με την χρήση **σωρών (Priority Queue)**
- Με την χρήση σωρών ο αλγόριθμος μπορεί να υλοποιηθεί σε O(|E|·log |E|), όπου E οι ακμές του γράφου.

Υλοποίηση με Σωρούς

- Θεωρούμε υλοποίηση γράφου με λίστα γειτνίασης.

- Οι πίνακες C και P μπορεί να αντικατασταθούν από ένα σωρό που περιέχει στοιχεία της μορφής (d, u, v), όπου d είναι το βάρος της ακμής (u,v).

- Κάθε φορά που προστίθεται μια καινούρια κορυφή στο δένδρο, προσθέτουμε στον σωρό πληροφορία για κάθε γειτονική της ακμή, η οποία συγκρατεί το βάρος της ακμής. Άρα ο σωρός θα κρατά περισσότερες από μια πληροφορία για κάθε κορυφή.

- Ο σωρός θα πρέπει να έχει μήκος .

- Η επιλογή καινούριας ακμής θα γίνεται με τη μέθοδο σωρών **DeleteMin**. Αφού όμως υπάρχουν περισσότερες από μια πληροφορίες για κάθε ακμή, και αφού η **DeleteMin** αφαιρεί μόνο την πληροφορία που αντιστοιχεί στην ακμή με το μικρότερο βάρος, θα πρέπει να είμαστε προσεκτικοί ούτως ώστε να μην προσθέτουμε ακμές μεταξύ κορυφών που ήδη γνωρίζουμε.

Η Υλοποίηση του Αλγόριθμου Prim 2

```
Prim(graph G){
int visited[n]={}; // Κορυφές που προστέθηκαν στο δένδρο (Αρχικά όλα "0")
int closest[n]={}; // «Πιο Κοντινός Γείτονας» για κάθε i: Αρχικά κανένας
int distance[n]=∞, // Απόσταση από «Κοντιν. Γείτονα» για κάθε i: Αρχικά άπειρο
Tree = {};    // Το ΕΓΔ που θέλουμε να κτίσουμε (περιέχει ακμές (α,β))
// επιλογή αρχικής κορυφής
Διάλεξε τυχαία κορυφή v;
Heap H = Heap();
visited[v] = 1;      // Τώρα το v ανήκει στο δένδρο

για κάθε κορυφή v {
    // ενημέρωση πινάκων distance & closest
    για κάθε w γείτονα του v {
        H.insert((G.weight(w,v),w,v));
       (d,v,u) = H.DeleteMin();
       while (visited[u] == 1)
           (d,v,u) = H.DeleteMin();
       visited[v]=1;

       //προσθήκη ακμής
       Tree = Tree ∪ {(closest[v],v)};
 }
}
```

G.weight(a,b) : βάρος ακμής a-b

Αν |E| = αριθμός των ακμών του G, ο χρόνος εκτέλεσης των Insert και DeleteMin είναι log |E|.

Ο ολικός χρόνος εκτέλεσης είναι O(|E|·log |E|)= O(|E|·log |V|).

Ορθότητα του Αλγόριθμου

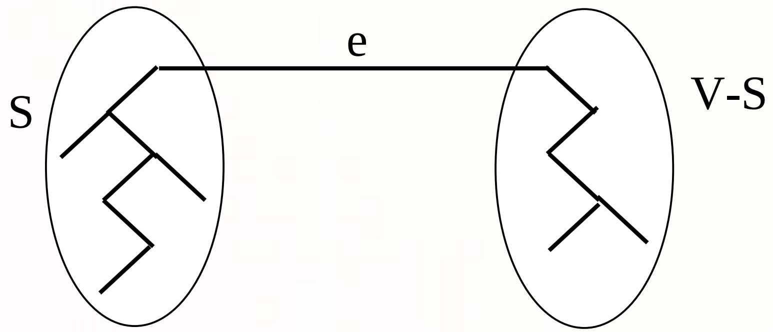
- Είναι εύκολο να δούμε ότι ο αλγόριθμος επιστρέφει ένα ΓΔ.
- Είναι λιγότερο εύκολο να αποφασίσουμε κατά πόσο το δένδρο που επιστρέφεται είναι ΕΓΔ.

Απόδειξη με αντίφαση:

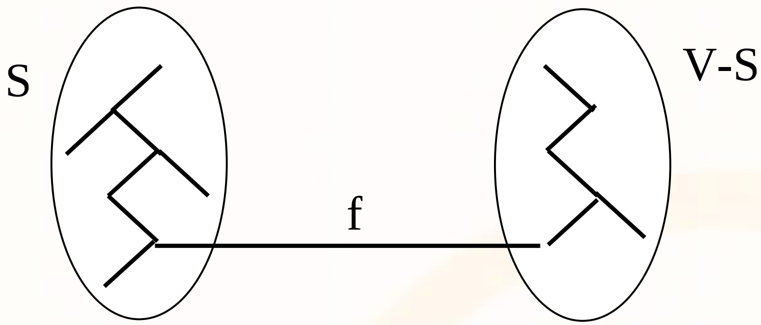
- Έστω ότι το δένδρο δεν είναι ΕΓΔ.
- Έστω ότι ο αλγόριθμος προσθέτει τις ακμές στο δένδρο με τη σειρά e_1, e_2, …, e_n. Διαλέγουμε την πρώτη ακμή στην ακολουθία, $f = e_j$, για την οποία η ακολουθία e_1,…, e_j δεν είναι μέρος κανενός ΕΓΔ του γράφου.
- Έστω S το σύνολο των κορυφών που βρίσκονται στις j-1 πρώτες ακμές. Τότε η f είναι η ακμή με το μικρότερο βάρος από τις ακμές στη γέφυρα του S.
- Έστω ότι T είναι ΕΓΔ που περιέχει τις j-1 πρώτες ακμές. Το T πρέπει να περιέχει τουλάχιστον μια ακμή e από τη γέφυρα του S.

Ορθότητα του Αλγόριθμου

- Το T έχει τη μορφή:



- Έστω T΄ ο γράφος (δένδρο;) που λαμβάνεται από το T με αντικατάσταση της ακμής e με την ακμή f:



- Το T΄ είναι ΓΔ και αφού το T είναι ΕΓΔ τότε και το T΄ είναι ΕΓΔ.
- Αυτό δίνει αντίφαση στην υπόθεση μας ότι οι πρώτες j ακμές δεν περιέχονται σε κανένα ΕΓΔ. Επομένως ο αλγόριθμος είναι ορθός.

Ο αλγόριθμος του Kruskal

- Ακόμα ένας άπληστος (greedy) αλγόριθμος που υπολογίζει το Ελάχιστο Γεννητορικό Δένδρο (ΕΓΔ).

- Ενώ ο αλγόριθμος του Prim επεξεργάζεται μια-μια τις ***κορυφές***, ο αλγόριθμος του Kruskal επεξεργάζεται μια-μια τις ***ακμές*** του γράφου.

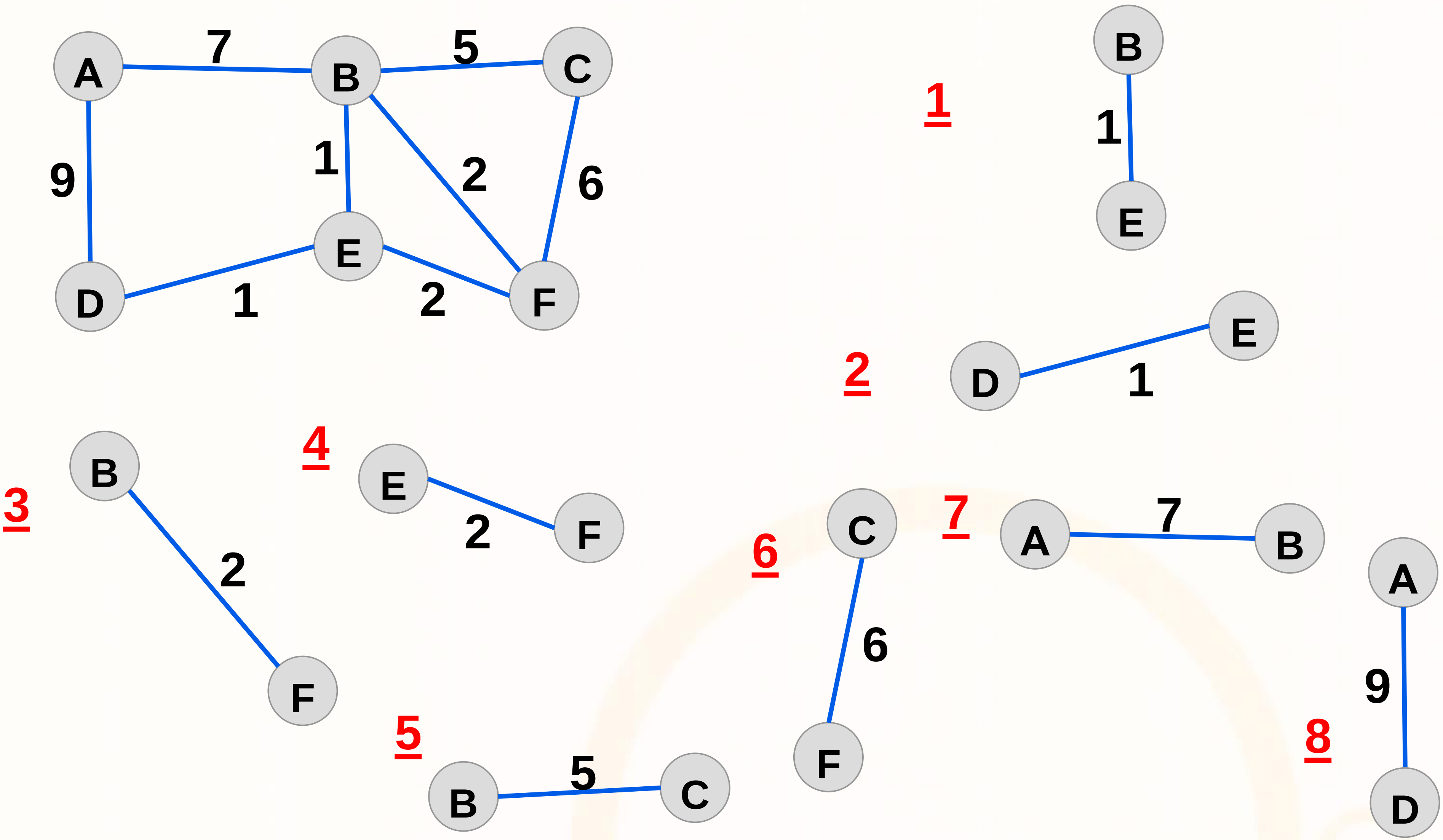
- Επίσης, ενώ σε κάθε βήμα του αλγόριθμου του Prim οι επιλεγμένες ακμές σχηματίζουν ένα **δένδρο**, στην περίπτωση του αλγόριθμου Kruskal, σχηματίζουν ένα **δάσος** (ένα σύνολο από δένδρα).

- **Κεντρική ιδέα.**
  - Αρχικά το δάσος Τ είναι άδειο.
  - Επεξεργαζόμαστε μια-μια τις ακμές, **σε αύξουσα σειρά βάρους.**
  - Αν η εισαγωγή της e στο Τ **δεν προκαλεί κύκλο**, τότε προσθέτουμε την e στο Τ, δηλαδή $T := T \cup \{e\}$.

Παράδειγμα Εκτέλεσης

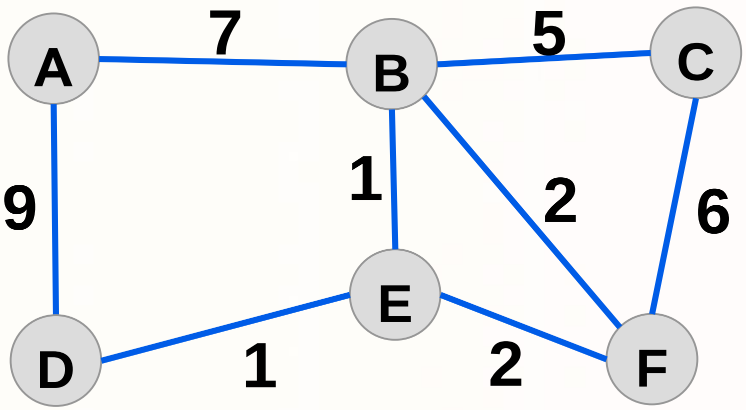
Γράφος G

Ταξινομημένες Ακμές με το Βάρος

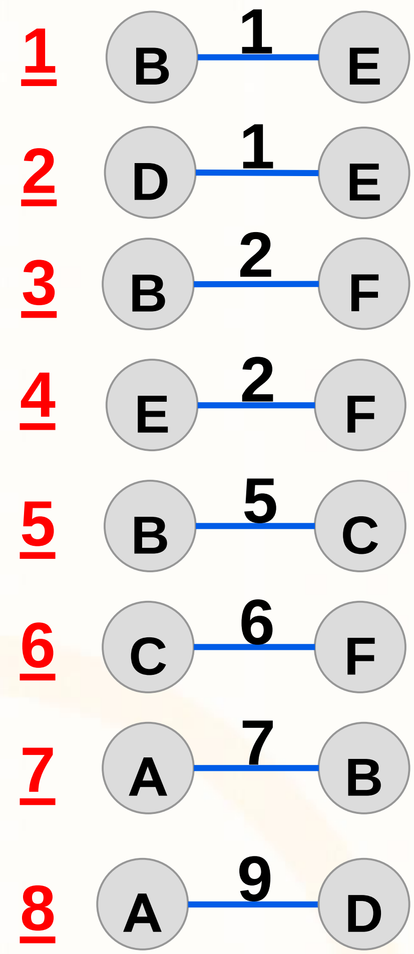


Παράδειγμα Εκτέλεσης

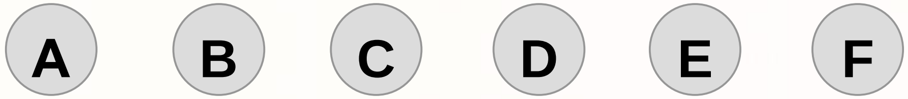
Γράφος G



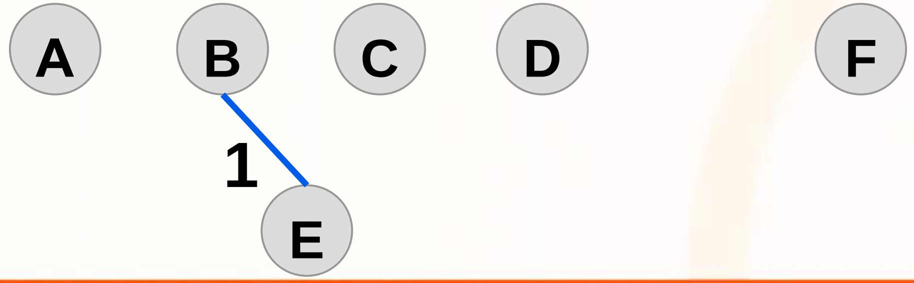
Ταξινομημένες Ακμές



Αρχική Κατάσταση

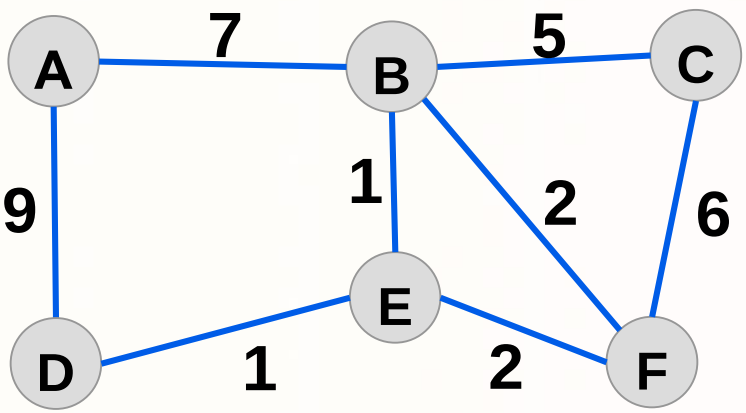


Μετά από επιλογή της πρώτης ακμής (B,E)

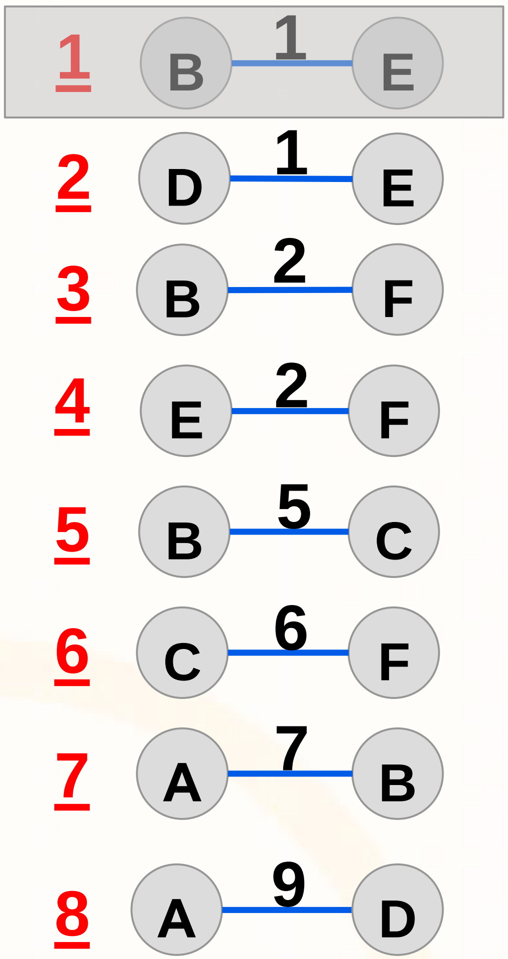


Παράδειγμα Εκτέλεσης

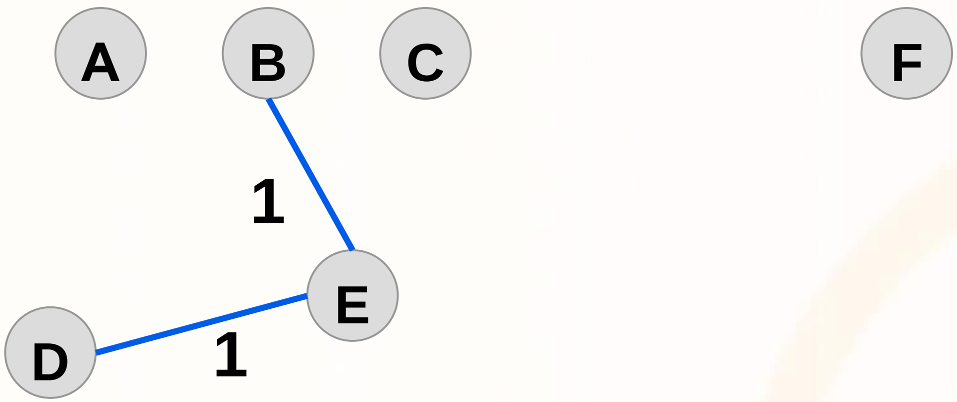
Γράφος G



Ταξινομημένες Ακμές

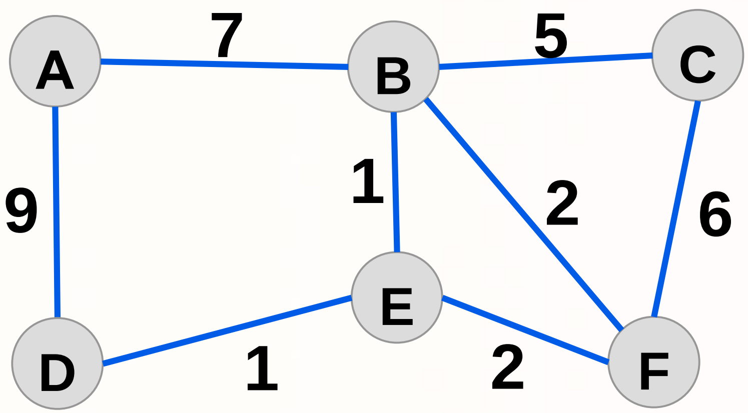


Μετά από επιλογή της πρώτης ακμής (D,E)

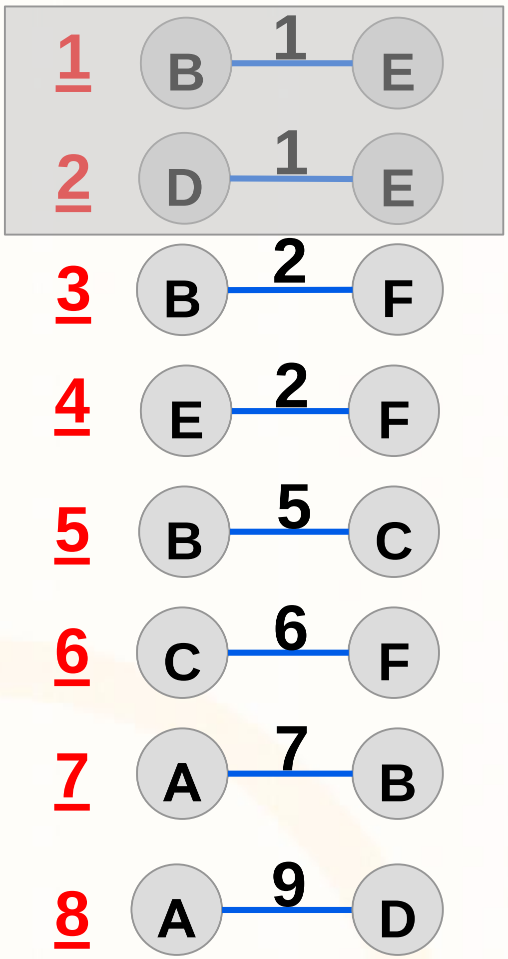


Παράδειγμα Εκτέλεσης

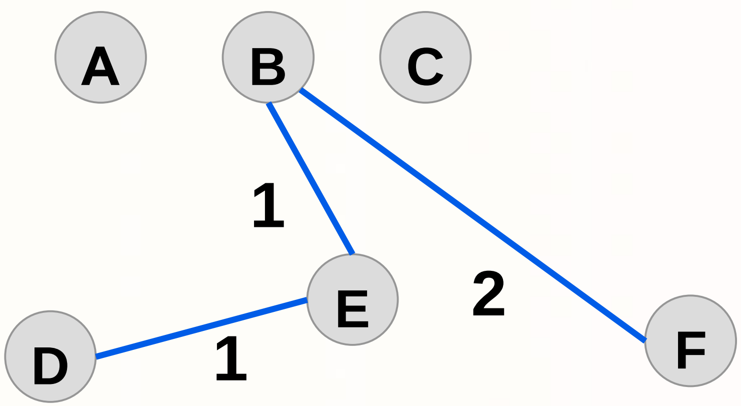
Γράφος G



Ταξινομημένες Ακμές

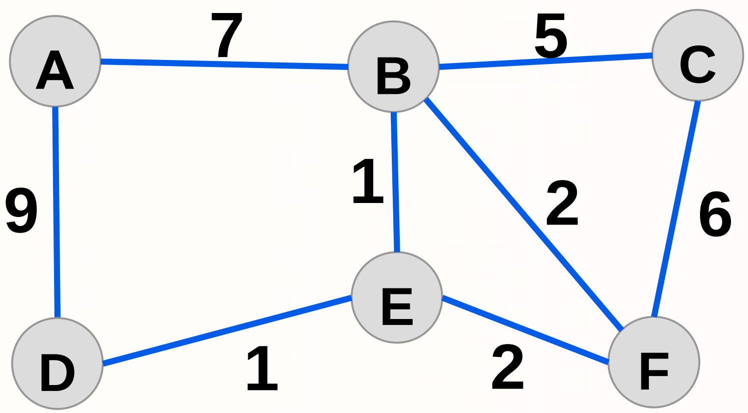


Μετά από επιλογή της τρίτης ακμής (B,F)

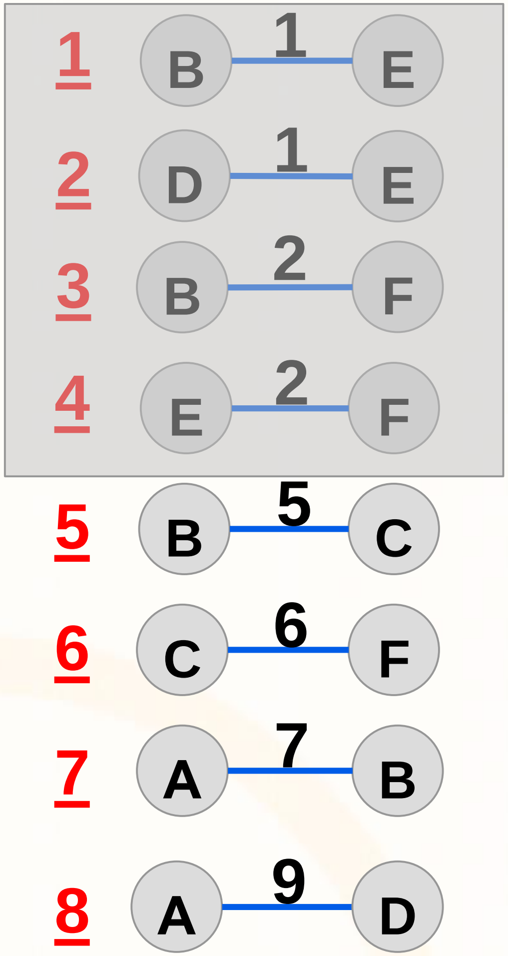


Παράδειγμα Εκτέλεσης

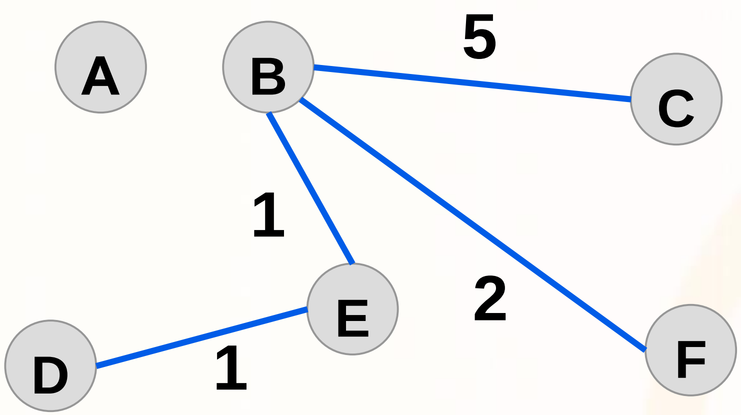
Γράφος G



Ταξινομημένες Ακμές

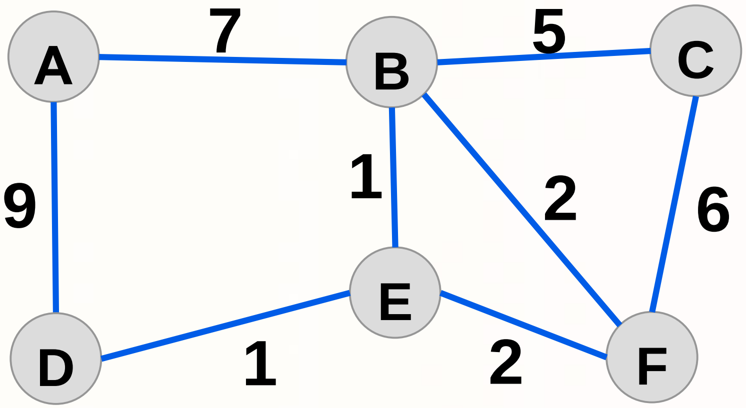


Μετά από επιλογή της τέταρτης ακμής (B,C)

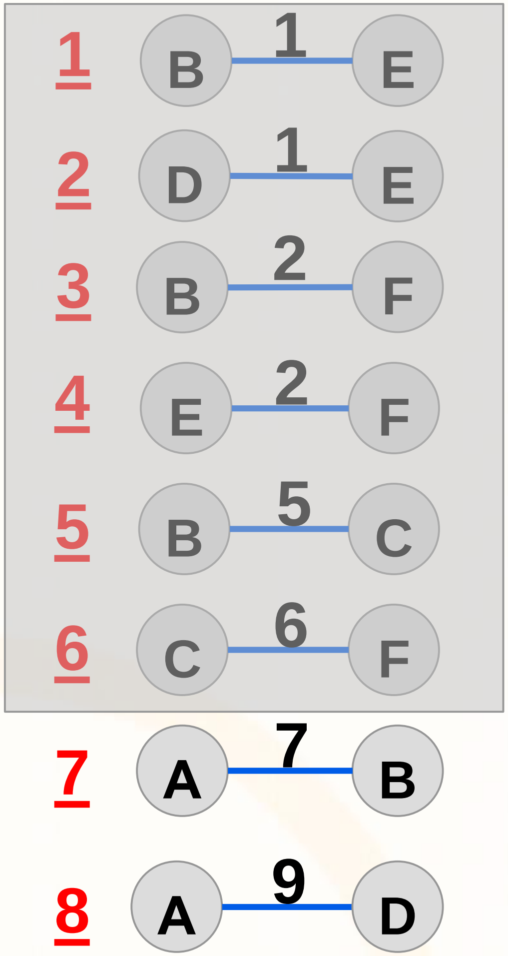


Παράδειγμα Εκτέλεσης

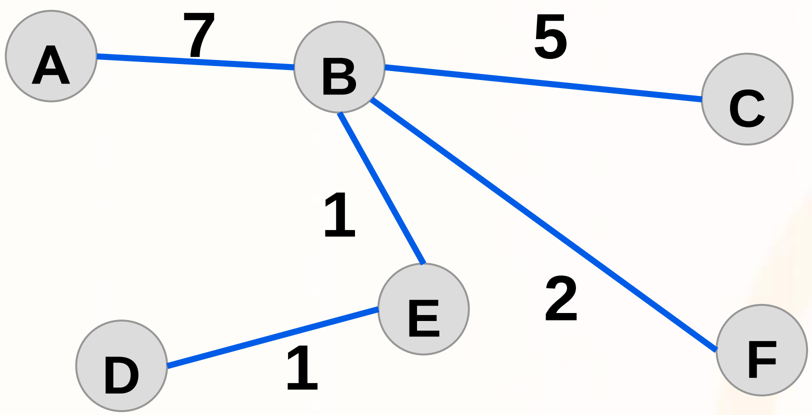
Γράφος G



Ταξινομημένες Ακμές

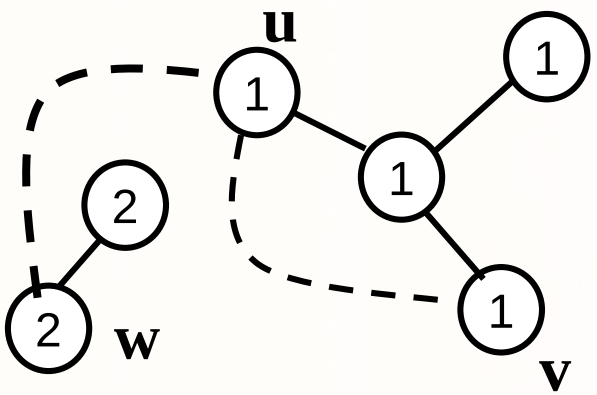


Μετά από επιλογή της τέταρτης ακμής (B,C)



Αποδοτική Εξάλειψη Κύκλων

- Η ταξινόμηση των Άκμων είναι απλή, δηλαδή ταξινομούμε μια φορά όλες τις ακμές με κάποιο αλγόριθμο ταξινόμησης.
- **Βασικό Πρόβλημα:** Το πρόβλημα που απομένει είναι πως θα βρίσκουμε αποδοτικά εάν μια ακμή μπορεί να δημιουργήσει κύκλο
- **Λύση**
  - Θα χρησιμοποιήσουμε ένα πίνακα **TID[n]** (TreeID) ο οποίος μας υποδεικνύει για κάθε κορυφή v σε πιο δένδρο ανήκει η v.
  - **Π.χ. εάν** θέλω να **προσθέσω** μια ακμή **(u,v)** και u & v ανήκουν στο ίδιο δένδρο **(TID[u]==TID[v]),** τότε αυτή η ακμή θα δημιουργήσει κύκλο.
  - **Επομένως δε θα προσθέσω την (u,v)**



- **Το (u,v) θα δημιουργήσει κύκλο γιατί και τα δυο ανήκουν στο ίδιο TID (i.e., 1)**
- **Το (u,w) δεν θα δημιουργήσει κύκλο γιατί οι δυο κόμβο ανήκουν σε διαφορετικά TID (i.e., 1 και 2)**

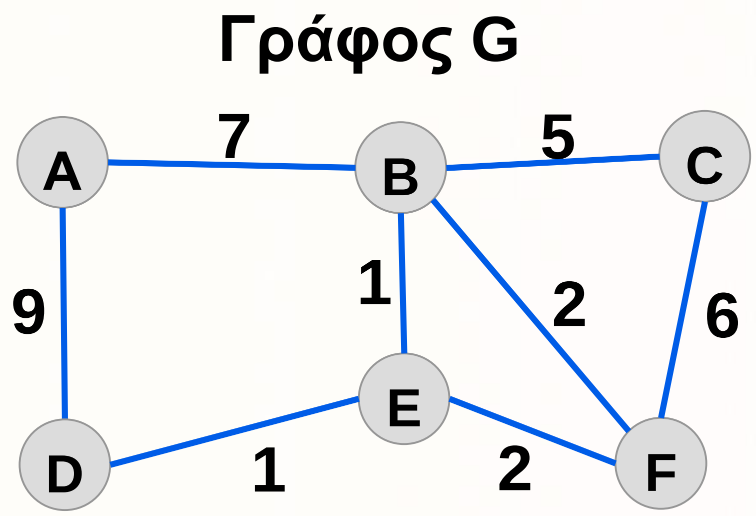
Υλοποίηση του Αλγορίθμου Kruskal

```
Kruskal(graph G(V,E)){
   Tree = {};  // Το ΕΓΔ: Ένα σύνολο ακμών (αρχικά κενό)
   TID[|V|] = {} // Πίνακας που κρατά το TreeID του κάθε κόμβου
   Count = 0;     // Μετρητής που κρατά από πόσες κορυφές πέρασα

   sortEdges(E);  // Ταξινόμηση Άκμων σε χρόνο O(|E|.log|E|)

   // Δημιουργούμε ένα δάσος από δένδρα μεγέθους 1
    for (i=0; i<|V|; i++)     // χρόνο Θ(|V|)
      TID[i] = i;

    for (i=0; i<|E|; i++) {    // χρόνο O(|E|)
      // ανάκτηση επόμενης (μικρότερης) ακμής
      (u,v) = nextEdge();          // χρόνο Θ(1)

      If (TID[u] != TID[v]) {    // Αν ανήκουν σε διαφορετικά δένδρα
          Tree = Tree ∪ {(u,v)};          // Προσθήκη Ακμής

         // Μετρούμε ποιος από τους u,v εμφανίζεται περισσότερο στο TID
         // και επιστρέφουμε τον μεγαλύτερο σαν TreeID x
         x = occurence(TID, u, v);    // χρόνο O(|V|)

         // Ανάθεση TreeID x σε όλους που έχουν  TID[u] ή TID[v]
         count = merge(TID[u],TID[v],x);
      }
      if (count == |V|) break;
    }
}
```

Συνολικός χρόνος: O(|E|.log|E| + |V| + |E|*|V|)

						|V|
TID	0	1	2	3	4	5

Εκτέλεση της Υλοποίησης Kruskal



Ταξινομημένες Ακμές

{ B-E=1, D-E=1, B-F=2, E-F=2, B-C=5, C-F=6, A-B=7, A-D=9 }

TID	0	1	2	3	4	5
	A	B	C	D	E	F

1. Nextedge => (B,E)
2. TID[B] != TID[E]? => YES, Επομένως Tree = {} ∪ {(B,E)};
3. Merge(TID[B],TID[E],1);

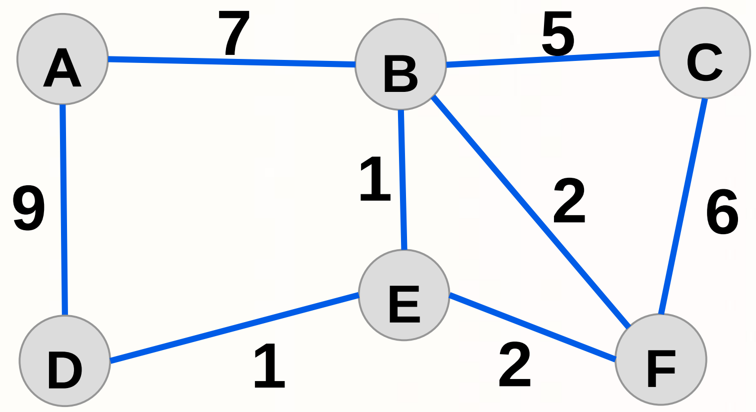
TID	0	1	2	3	1	5
	A	B	C	D	E	F

4. Nextedge => (D,E)
5. TID[D] != TID[E]? => YES, Επομένως Tree={(B,E)} ∪ {(D,E)};
6. Merge(TID[D],TID[E],1);

TID	0	1	2	1	1	5
	A	B	C	D	E	F

Εκτέλεση της Υλοποίησης Kruskal

Γράφος G



Ταξινομημένες Ακμές

{ B-E=1, D-E=1, B-F=2, E-F=2, B-C=5, C-F=6, A-B=7, A-D=9 }

TID	0	1	2	1	1	5
	A	B	C	D	E	F

7. Nextedge => (B,F)
8. TID[B] != TID[F]? => YES,Επομένως Tree = {(B,E),(D,E)} ∪ {(B,F)};
9. Merge(TID[B],TID[F],1);

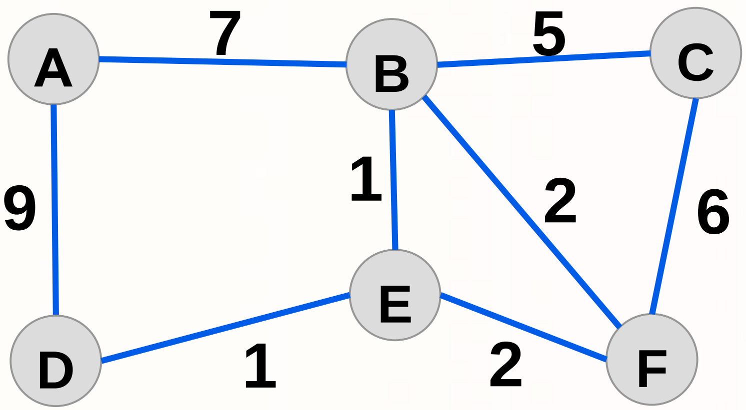
TID	0	1	2	1	1	1
	A	B	C	D	E	F

10. Nextedge => (E,F)
11. TID[E] != TID[F]? => NO, Επομένως δεν χρησιμοποιούμε το (E,F);

TID	0	1	2	1	1	1
	A	B	C	D	E	F

Εκτέλεση της Υλοποίησης Kruskal

Γράφος G



Ταξινομημένες Ακμές

{ B-E=1, D-E=1, B-F=2, E-F=2, B-C=5, C-F=6, A-B=7, A-D=9 }

TID	0	1	2	1	1	1
	A	B	C	D	E	F

12. Nextedge => (B,C)

13. TID[B]!=TID[C]? => YES,Επομένως Tree = {(B,E),(D,E), (B,F)} ∪ {(B,C)};

14. Merge(TID[B],TID[C],1);

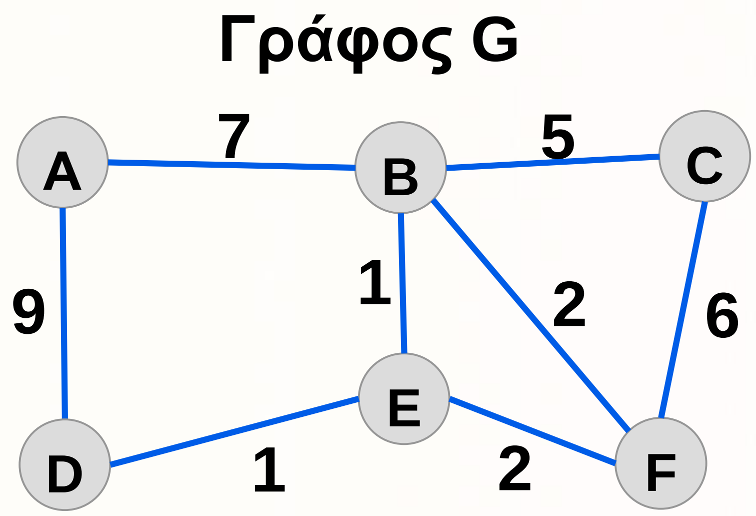
TID	0	1	1	1	1	1
	A	B	C	D	E	F

15. Nextedge => (C,F)

16. TID[E] != TID[F]? => NO, Επομένως δεν χρησιμοποιούμε το (C,F);

TID	0	1	1	1	1	1
	A	B	C	D	E	F

Εκτέλεση της Υλοποίησης Kruskal



Ταξινομημένες Ακμές

{ B-E=1, D-E=1, B-F=2, E-F=2, B-C=5, C-F=6, A-B=7, A-D=9 }

TID	0	1	1	1	1	1
	A	B	C	D	E	F

17. Nextedge => (A,B)

18. TID[A] != TID[B]? => YES,

Επομένως Tree = {(B,E),(Δ,E), (B,Z), (B,Γ)} ∪ {(A,B)};

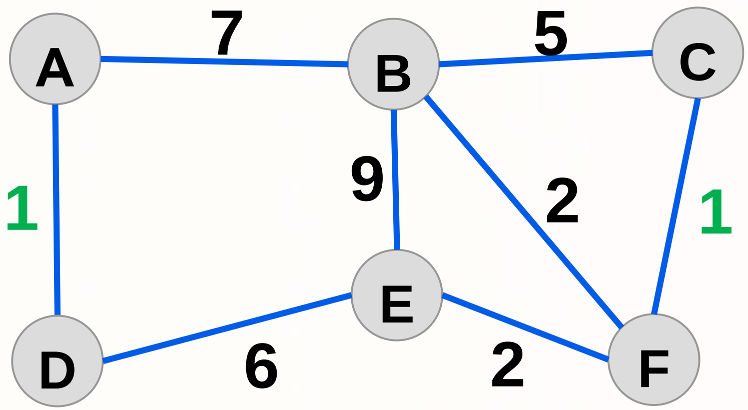
TID	1	1	1	1	1	1
	A	B	C	D	E	F

Εδώ βρήκαμε |V| vertices, επομένως διακόπτουμε την αναζήτηση.

ΤΟ ΕΓΔ είναι το {(B,E),(Δ,E), (B,Z), (B,Γ),(A,B)};

Παράδειγμα Εκτέλεσης 2

Γράφος G



Αρχική Κατάσταση

A B C D E F

Ταξινομημένες Ακμές

{ A-D=1, C-F=1, B-F=2, E-F=2, B-C=5, D-E=6, A-B=7, B-E=9 }

0.

TID	0	1	2	3	4	5
	A	B	C	D	E	F

1. ➔{A-D}

TID	0	1	2	0	4	5
	A	B	C	D	E	F

2. ➔{C-F}

TID	0	1	2	0	4	2
	A	B	C	D	E	F

3. ➔{B-F}

TID	0	2	2	0	4	2
	A	B	C	D	E	F

4. ➔{E-F}

TID	0	2	2	0	2	2
	A	B	C	D	E	F

5. ➔{B-C} = τίποτα

6. ➔{D-E}

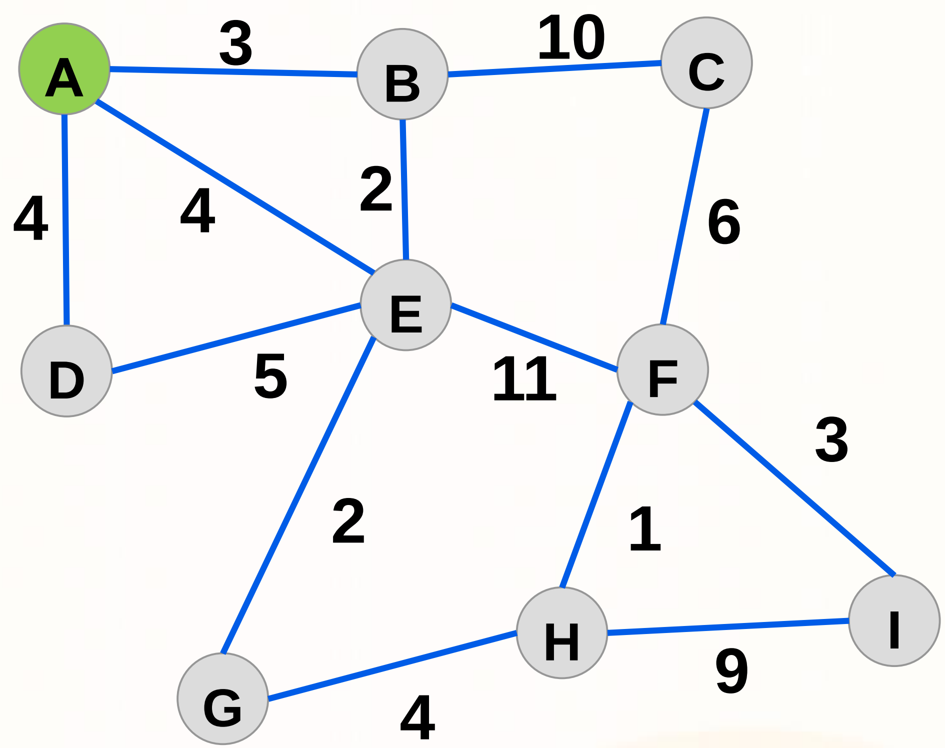
TID	2	2	2	2	2	2
	A	B	C	D	E	F

7. break

Μερικά Σχόλια

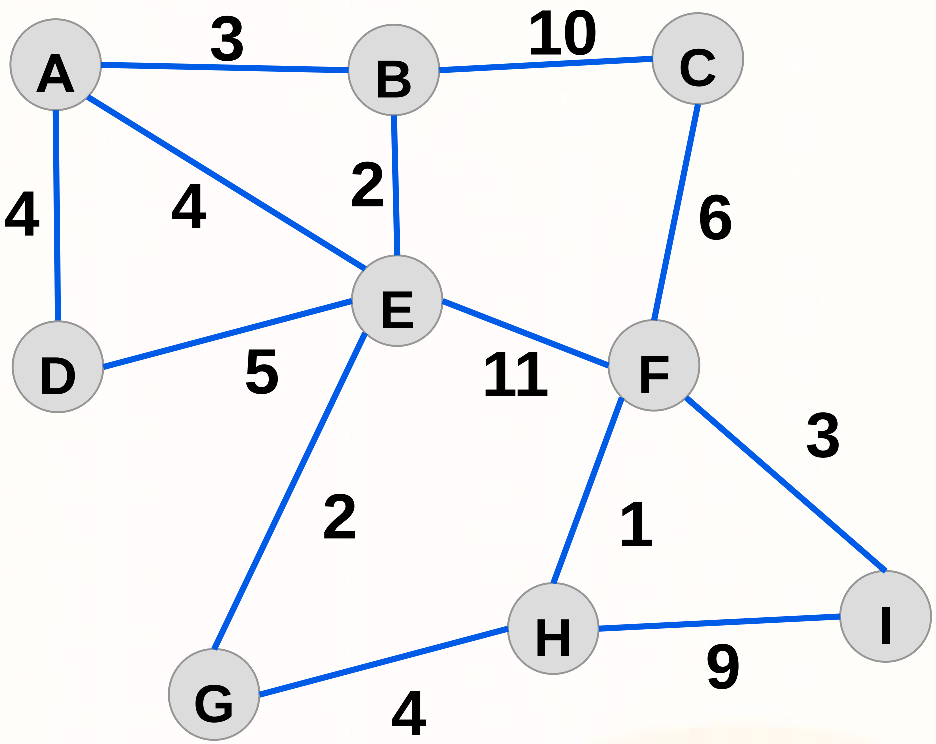
- Η ορθότητα του αλγόριθμου μπορεί να αποδειχθεί όπως και στην περίπτωση του αλγόριθμου του Prim.
- Το σύνολο των ακμών μπορεί να διατηρηθεί ως μία σωρός.
- Για να ελέγξουμε αν μια ακμή μπορεί να προστεθεί, διατηρούμε ένα partition Π όλων των κορυφών:
    - Δυο κορυφές βρίσκονται στο ίδιο υποσύνολο του Π αν υπάρχει μονοπάτι μεταξύ τους.
    - Αρχικά κάθε υποσύνολο του Π περιέχει ακριβώς μια κορυφή.
    - Στη συνέχεια, μια ακμή (u,v) μπορεί να προστεθεί αν οι κορυφές u και v βρίσκονται σε διαφορετικά υποσύνολα του Π. Με την προσθήκη μιας ακμής (u,v), υποσύνολα του Π που περιέχουν τις κορυφές u και v, ενώνονται.

ΕΓΔ: Ασκήσεις – Εκτελέστε τον αλγόριθμο Prim



	A	B	C	D	E	F	G	H	I
visited:	1	0	0	0	0	0	0	0	0
closest:	0	0	0	0	0	0	0	0	0
distance:	∞	∞	∞	∞	∞	∞	∞	∞	∞

ΕΓΔ: Ασκήσεις - Εκτελέστε τον αλγόριθμο Kruskal



Ταξινομημένες Ακμές

{ F-H=1, B-E=2, E-G=2, A-B=3, F-I=3, A-D=4, A-E=4, G-H=4, D-E=5, C-F=6, H-I=9, B-C=10, E-F=11 }

TID	0	1	2	3	4	5	6	7	8
	A	B	C	D	E	F	G	H	I