Ελάχιστα Γεννητορικά Δένδρα

Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:

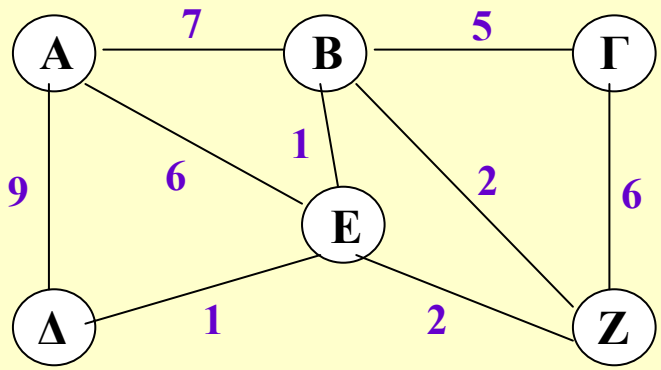
Ο αλγόριθμος του Prim και ο αλγόριθμος του Kruskal για εύρεση Ελάχιστων Γεννητορικών Δένδρων

Ελάχιστα Γεννητορικά Δένδρα (ΕΓΔ)

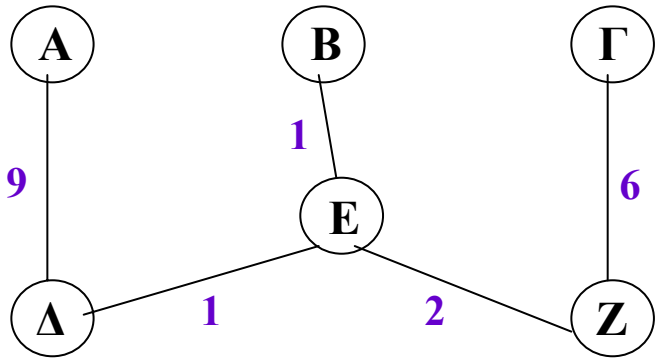
- Έστω ένας μη κατευθυνόμενος γράφος με βάρη, G. ***Γεννητορικό δένδρο*** (spanning tree, ΓΔ) του G ονομάζουμε κάθε δένδρο T που περιέχει όλους τους κόμβους του G και κάθε ακμή του οποίου είναι και ακμή του G.
- Με άλλα λόγια το T ορίζει κάποιο υποσύνολο των ακμών του G το οποίο
  1. εξακολουθεί να κρατά τον γράφο συνεκτικό, και
  2. δεν περιέχει κύκλο.
- *Παρατήρηση*: αφαίρεση ακμής από ένα ΓΔ έχει σαν αποτέλεσμα τη μετατροπή του γράφου σε μη-συνεκτικό γράφο.
- Το ***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.

Παραδείγματα ΕΓΔ

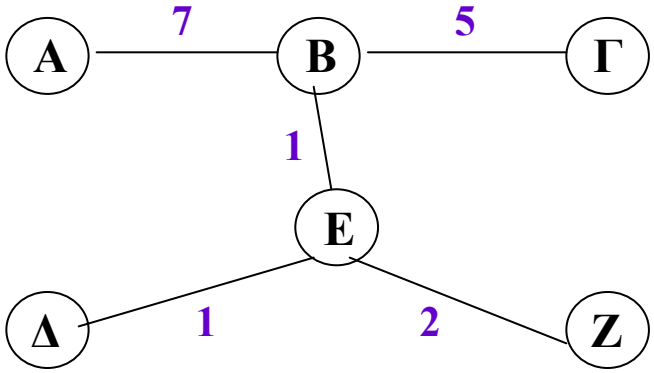
Γράφος G



ΓΔ1



ΓΔ2

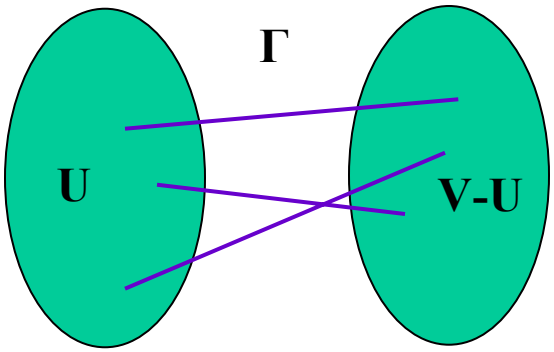


Βάρος: 19

Βάρος: 16

Ιδιότητες ΕΓΔ

- Όλες οι κορυφές 'καλύπτονται', γι'αυτό το δένδρο ονομάζεται και δένδρο σκελετός (spanning tree).
- Ένα γεννητορικό δένδρο γράφου με n κορυφές έχει n-1 ακμές.
- ***Ορισμός:*** Έστω γράφος G=(V,E), και U υποσύνολο του V. Τότε ονομάζουμε ***γέφυρα του U***, το σύνολο των ακμών που συνδέουν κορυφές του U με κορυφές του V-U.



Ιδιότητες ΕΓΔ

- Αν Γ είναι μια γέφυρα οποιουδήποτε υποσυνόλου των κορυφών ενός γράφου, τότε κάθε ΓΔ θα περιέχει τουλάχιστον μια ακμή από τη Γ.

- Γνωστό και βαθιά μελετημένο πρόβλημα στην επεξεργασία γράφων είναι η εύρεση ΕΓΔ. Έχει ποικίλες εφαρμογές.

- Κύρια ιδέα: αφαιρούμε όσες ακμές μπορούμε, ελαχιστοποιώντας το συνολικό βάρος και διατηρώντας τη συνεκτικότητα.

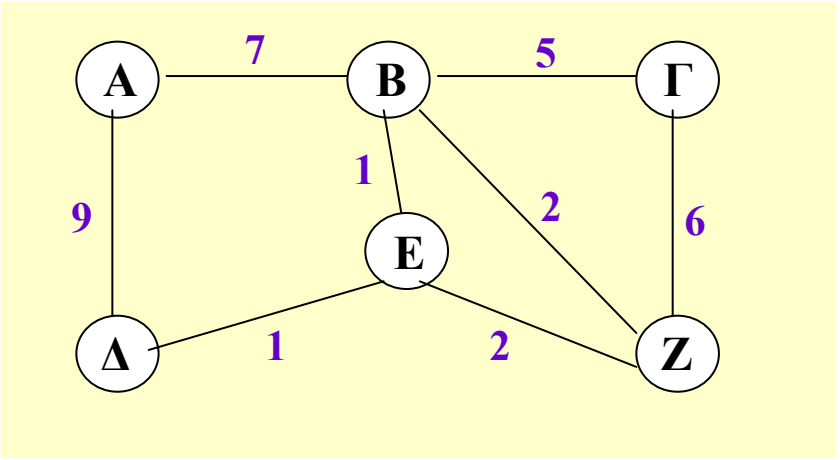
Ο αλγόριθμός του Prim

- 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 από τις ακμές που ανήκουν στη γέφυρα του S.

Παράδειγμα Εκτέλεσης

Γράφος G



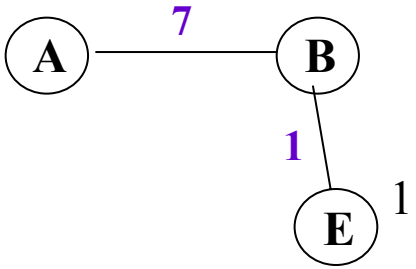
1:

A	B : 7, Δ : 9

2:

A	Δ : 9
B	Γ : 5, E : 1, Z : 2

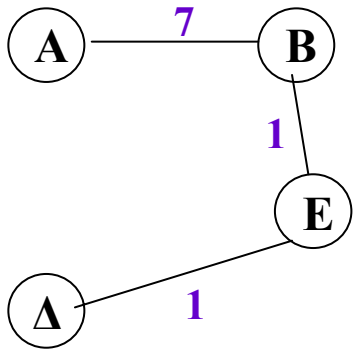
3:



A	Δ : 9
B	Γ : 5, Z : 2
E	Δ : 1, Z : 2

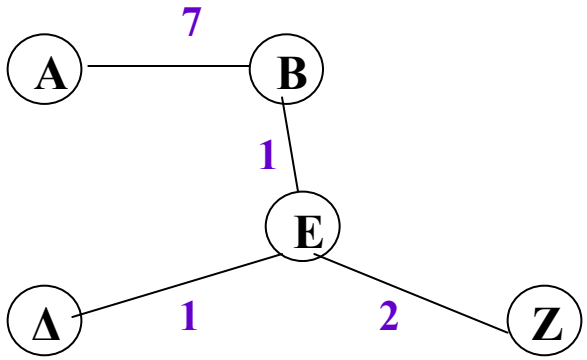
Παράδειγμα Εκτέλεσης (συν.)

4:



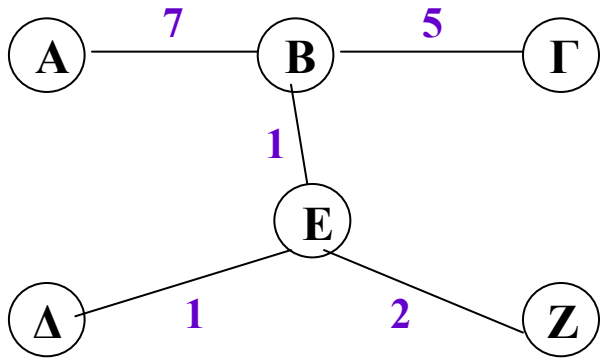
B	Γ : 5, Z : 2
E	Z : 2

5:



B	Γ : 5
Z	Γ : 6

6:



Η Υλοποίηση

- Διατηρούμε τον πίνακα **S** όπου για κάθε κορυφή u, **S**[u]=0 σημαίνει ότι η u δεν ανήκει μέχρι στιγμής στο δένδρο και **S**[u]=1 ότι ανήκει, και...
- Δύο πίνακες, **P** και **C** για κάθε κορυφή u που δεν βρίσκεται ήδη στο δένδρο όπου
  - στη θέση **P**[u] φυλάγουμε την κορυφή του δένδρου που βρίσκεται πιο κοντά στην u, και
  - στη θέση **C**[u] φυλάγουμε το βάρος της ακμής (u, **P**[u]), το οποίο συμβολίζουμε ως d(u, **P**[u]).
- Διαλέγουμε την επόμενη ακμή του δένδρου βρίσκοντας το μικρότερο **C**[u] από τις κορυφές u που δεν βρίσκονται στο δένδρο ως εξής:

```
minVertex(int S[], int C[]){
   επίστρεψε την κορυφή j για την οποία S[j] = 0
   και για κάθε κορυφή k,
        if S[k]=0 then C[j] <= C[k]
}
```

Η Υλοποίηση

- Με την επιλογή της ακμής (u,v), προσθέτουμε την ακμή και την κορυφή u στο δένδρο.
- Με κάθε εισαγωγή κόμβου u στο δένδρο οι πίνακες **C** και **P** πιθανό να χρειαστούν αλλαγές:

```
για κάθε w γείτονα του u
  if (d(u,w) < C[w])
    P[w] = u; C[w] = d(u,w);
```

Η Υλοποίηση

```
Prim(graph G){

  int C[n]=∞, P[n];
  int S[n]=0;

  διάλεξε τυχαία κορυφή v;
  S[v] = 1;
  Tree = {};

  for (i=1; i<|V|; i++){
     για κάθε w γείτονα του v
        if (d(v,w) < C[w])
           P[w] = v; C[w] = d(v,w);
     v = minVertex (S, C);
     S[v]=1;
     Tree = Tree ∪ {(P[v],v)};
  }
}
```

Ανάλυση Χρόνου Εκτέλεσης

- Η διαδικασία minVertex μπορεί να υλοποιηθεί χρησιμοποιώντας ακριβώς ένα **`while`**-loop: απαιτεί χρόνο O(|V|), όπου |V| είναι ο αριθμός των κορυφών του γράφου.
- Ο χρόνος εκτέλεσης του βρόχου της εντολής **`for`** της διαδικασίας είναι O(|V|). (Και για υλοποίηση με πίνακα γειτνίασης και για υλοποίηση με λίστα γειτνίασης.)
- Άρα ο ολικός χρόνος εκτέλεσης είναι $\Theta(|V|^2)$.
- Μπορούμε να βελτιώσουμε τον αλγόριθμο;

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 διαδικασία 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.

Υλοποίηση με Σωρούς

```
Prim2(graph G){
   int S[n]=0;

   MakeEmptyHeap(H);

   διάλεξε τυχαία κορυφή v;
   S[v] = 1;
   Tree = {};

   for (i=1; i<|V|; i++)
      για κάθε w γείτονα του v
          Insert((d(w,v),w,v), H);

      (d,v,u) = DeleteMin(H);

      while (S[v] == 1)
          (d,v,u) = DeleteMin(H);
      S[v]=1;
      Tree = Tree ∪ {(u,v)};
}
```

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

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

Ορθότητα του Αλγόριθμου

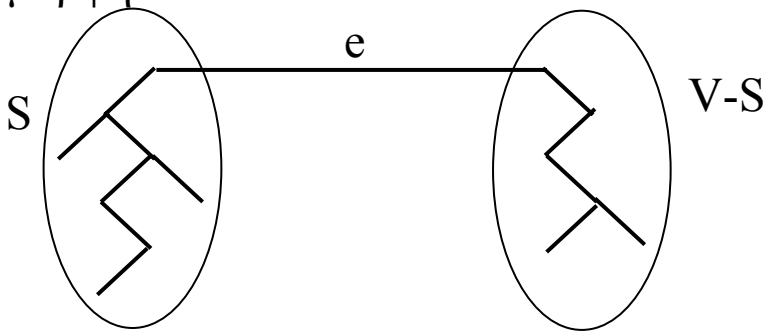
- Είναι εύκολο να δούμε ότι ο αλγόριθμος επιστρέφει ένα ΓΔ.
- Είναι λιγότερο εύκολο να αποφασίσουμε κατά πόσο το δένδρο που επιστρέφεται είναι ΕΓΔ.

Απόδειξη με αντίφαση:

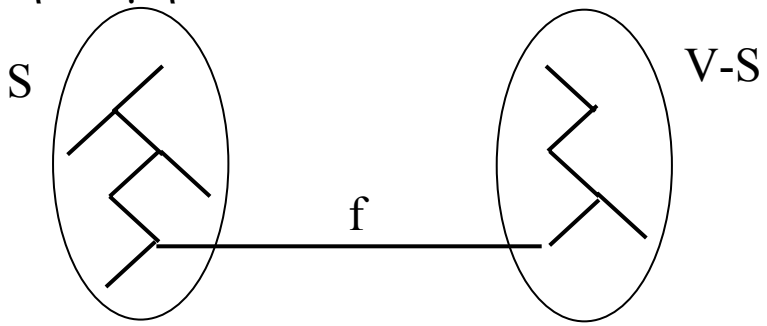
- Έστω ότι το δένδρο δεν είναι ΕΓΔ.
- Έστω ότι ο αλγόριθμος προσθέτει τις ακμές στο δένδρο με τη σειρά e_1, e_2, …, e_n. Διαλέγουμε την πρώτη ακμή στην ακολουθία, $f = e_j$, για την οποία η ακολουθία e_1,…, e_j δεν είναι μέρος κανενός ΕΓΔ του γράφου.
- Έστω S το σύνολο των κορυφών που βρίσκονται στις 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 Τ έχει τη μορφή:



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



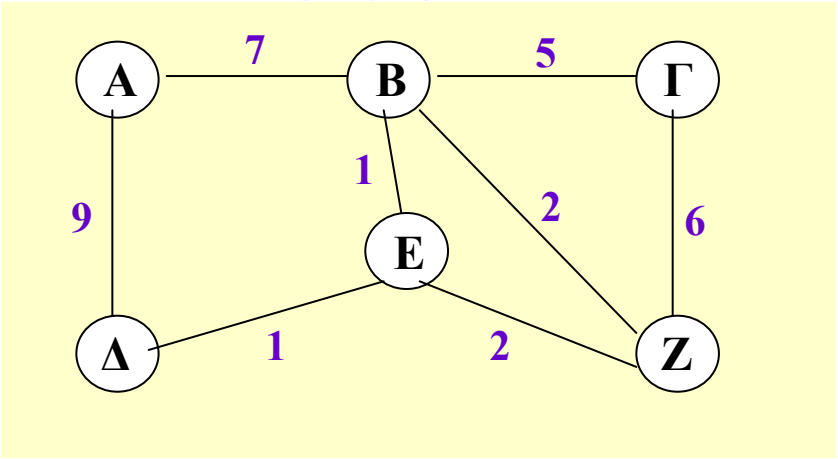
- Το Τ΄ είναι ΓΔ και αφού το Τ είναι ΕΓΔ τότε και το Τ΄ είναι ΕΓΔ.
- Αυτό δίνει αντίφαση στην υπόθεση μας ότι οι πρώτες j ακμές δεν περιέχονται σε κανένα ΕΓΔ. Επομένως ο αλγόριθμος είναι ορθός.

Ο αλγόριθμος του Kruskal

- Ακόμα ένας αλγόριθμος που υπολογίζει 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 T είναι άδειο.
  - επεξεργαζόμαστε μια-μια τις ακμές, σε αύξουσα σειρά βάρους:
  - αν η ακμή e έχει το ελάχιστο βάρος από αυτές που δεν έχουμε μέχρι στιγμής επεξεργασθεί, ελέγχουμε αν η εισαγωγή της e στο T δεν προκαλεί κύκλο. Αν η απάντηση είναι θετική, προσθέτουμε την e στο T, δηλ. $T := T \cup \{e\}$.

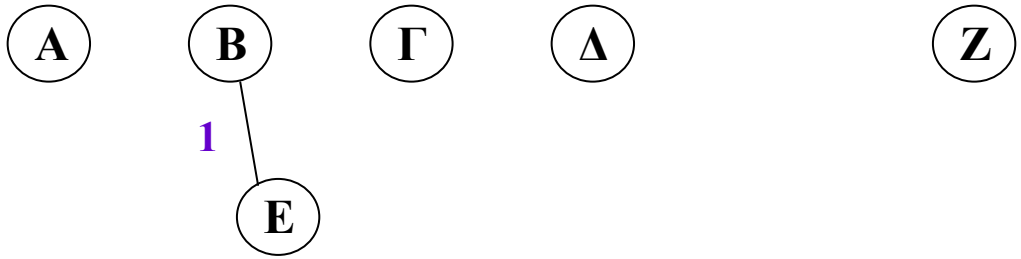
Παράδειγμα Εκτέλεσης

Γράφος G



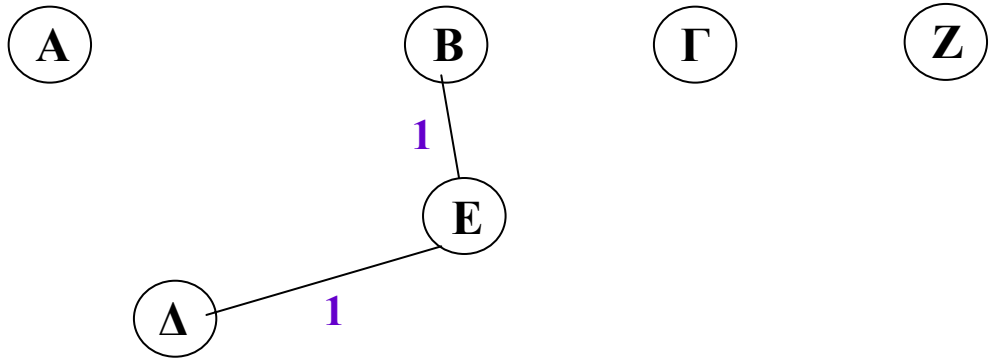
Αρχική Κατάσταση

Μετά από επιλογή της πρώτης ακμής (B,E)

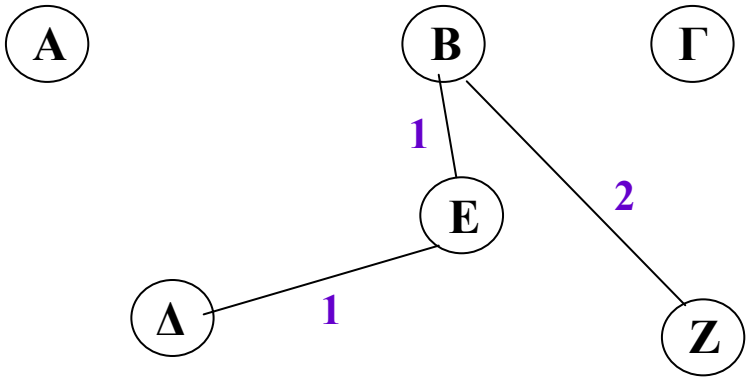


Παράδειγμα Εκτέλεσης

Μετά από επιλογή της δεύτερης ακμής (Δ,Ε)

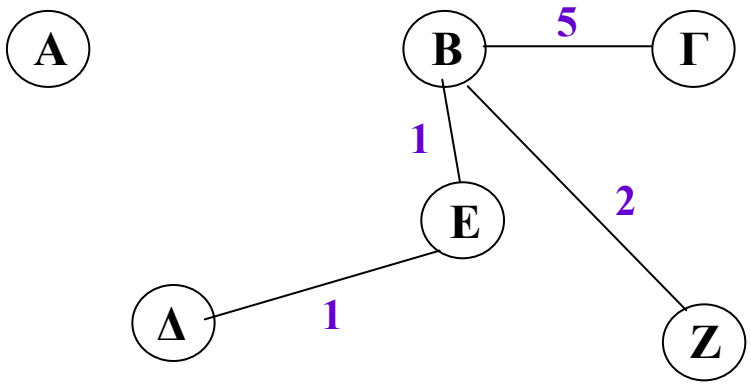


Μετά από επιλογή της τρίτης ακμής (Β,Ζ)

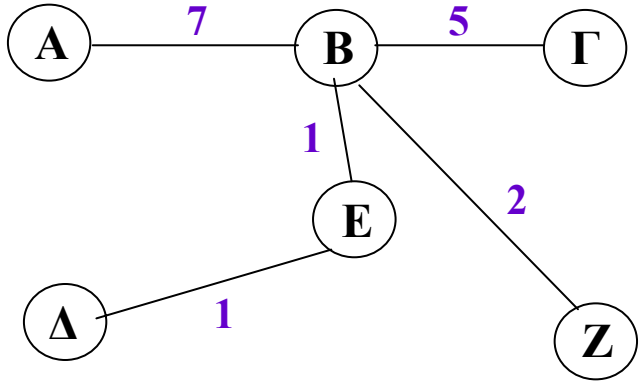


Παράδειγμα Εκτέλεσης

Μετά από επιλογή της τέταρτης ακμής (Β,Γ)



Μετά από επιλογή της πέμπτης ακμής (Α,Β)



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.

Ο αλγόριθμος Kruskal – ψευδοκώδικας

```
kruskal (graph G){

buildHeap(H); (που περιέχει όλες τις ακμές του G)

partition A = {{v1},{v2},…,{vn}};
i = 1;
while (i < n){
   (u,v) = DeleteMin(H);
   U = find(A,u);
   V = find(A,v);
   if (U!=V)
        T = T∪{(u,v)};
        union(A, U, V);
        i++;
}
```

Χρόνος Εκτέλεσης:

- Η διαδικασία `find(A,u)` βρίσκει και επιστρέφει το υποσύνολο του A που περιέχει το στοιχείο u και η διαδικασία `union(A,U,V)` επιστρέφει το σύνολο A΄ = A – {U,V} ∪{ U∪V}.