



# Διάλεξη 31: Ελάχιστα Γεννητορικά Δέντρα Ο Αλγόριθμος Prim

---

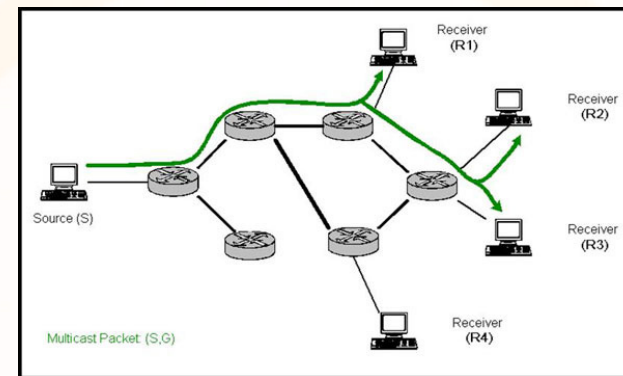
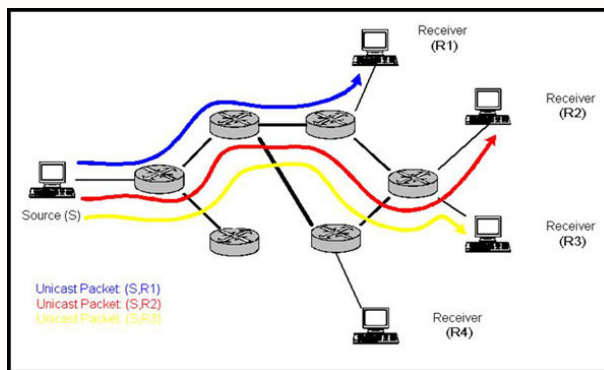
**Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:**

- Ελάχιστα Γεννητορικά Δένδρα (ΕΓΔ) – Minimum Spanning Trees
- Ο αλγόριθμος του Prim για εύρεση ΕΓΔ σε γράφους

**Διδάσκων: Παναγιώτης Ανδρέου**

# 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

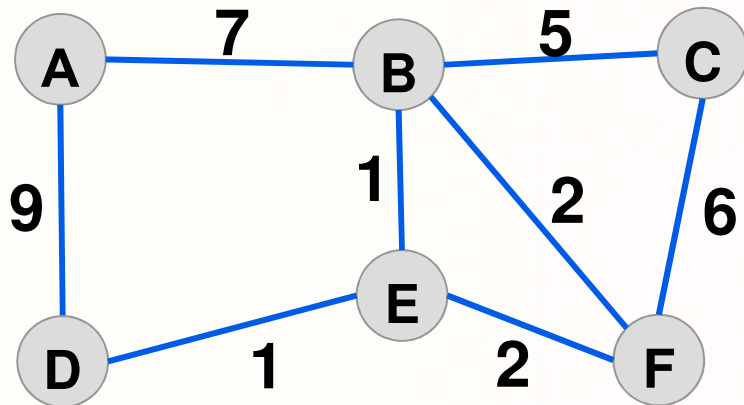


# Ελάχιστα Γεννητορικά Δένδρα (ΕΓΔ)

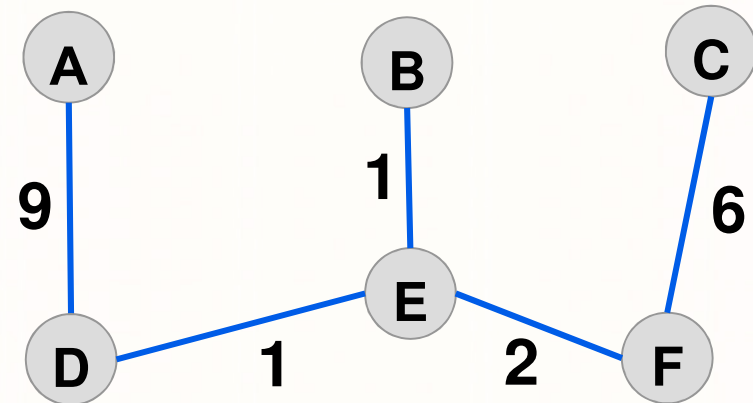
- Έστω ένας μη-κατευθυνόμενος γράφος με βάρη,  $G(V,E)$ .
- Γεννητορικό δένδρο (spanning tree, ΓΔ) του  $G$  ονομάζουμε κάθε δένδρο  $T$  που περιέχει όλους τους κόμβους του  $G$  και κάθε ακμή του οποίου είναι και ακμή του  $G$ .
- Σε ένα ΓΔ, όλες οι κορυφές **‘καλύπτονται’**, γι’ αυτό το δένδρο ονομάζεται 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

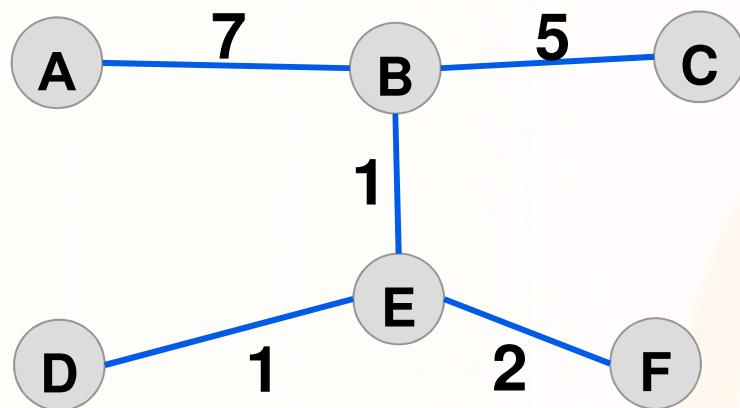
Γράφος G



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

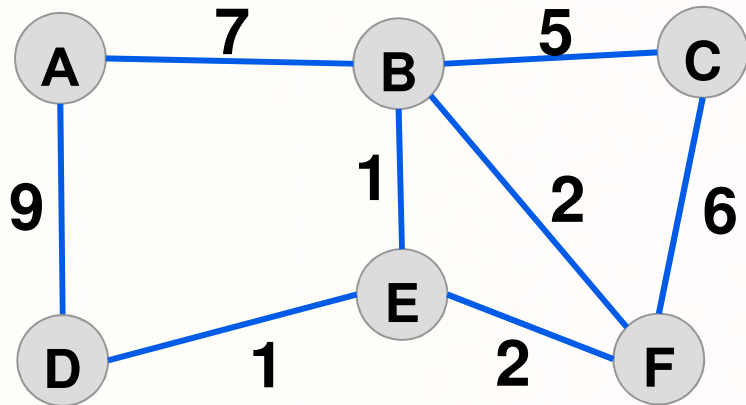
- Υποθέτουμε στην συνέχεια ότι οι γράφοι που μελετάμε είναι **συνεκτικοί** (δηλαδή υπάρχει τουλάχιστο μια διαδρομή μεταξύ όλων των κορυφών).
- Δεν κάνει νόημα να βρούμε ένα ΕΓΔ ενός μη-συνεκτικού γράφου ... γιατί ο ορισμός του ΕΓΔ προϋποθέτει ότι το δένδρο καλύπτει όλες τις κορυφές.
- Η εύρεση ΕΓΔ είναι γνωστό και βαθιά μελετημένο πρόβλημα στην επεξεργασία γράφων. Έχει ποικίλες εφαρμογές.

# Ο αλγόριθμός του Prim

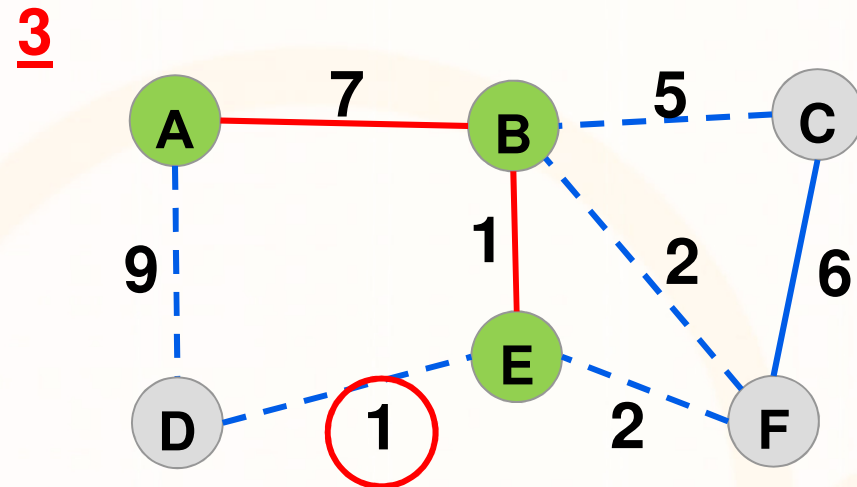
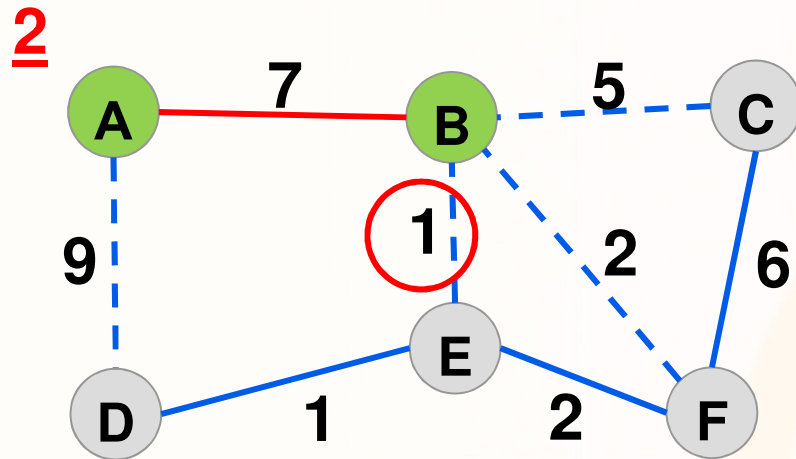
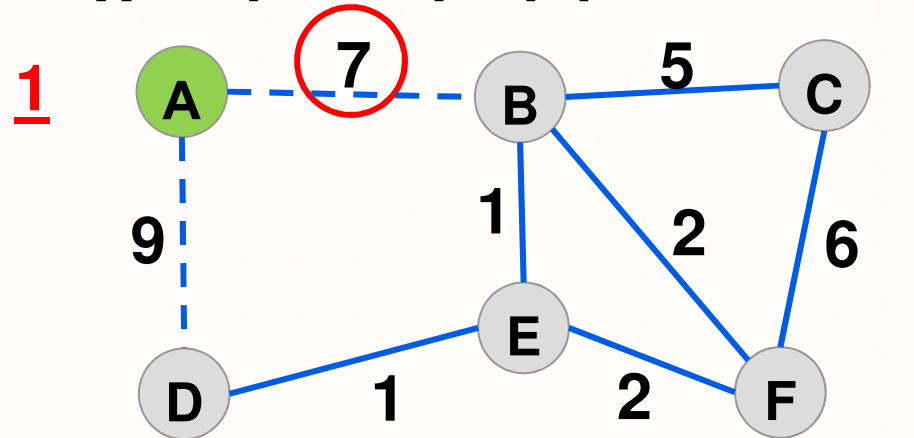
- Αρχικά το δένδρο περιέχει ακριβώς μία κορυφή, η οποία επιλέγεται τυχαία.
- Για να κτίσουμε το δένδρο, σε κάθε βήμα συνδέουμε ακόμα μια κορυφή στο παρόν δένδρο με την επιλογή και εισαγωγή μιας καινούριας ακμής (από τις ακμές του γράφου).
- Πως μπορούμε να επιλέξουμε την κατάλληλη ακμή;
- Στην περίπτωση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

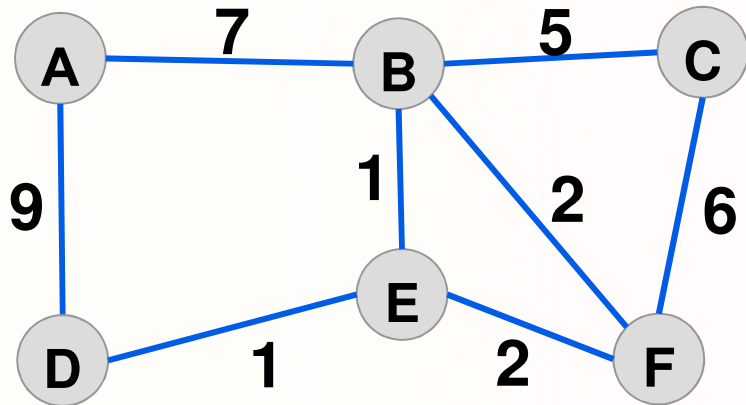


Ξεκινούμε διαλέγοντας τυχαία μια κορυφή

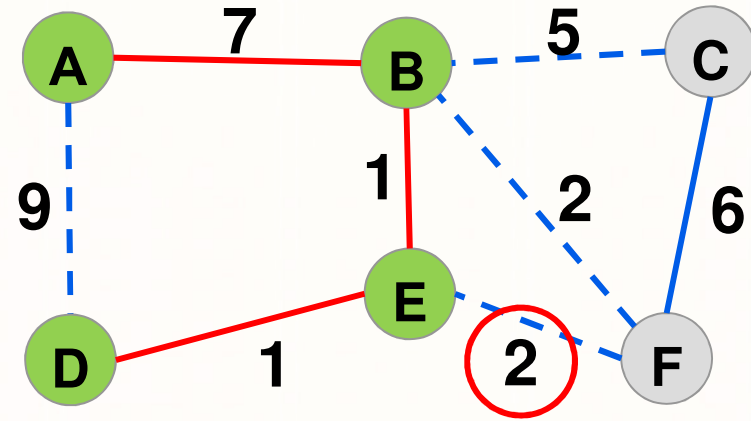


# Παράδειγμα Εκτέλεσης (συν.)

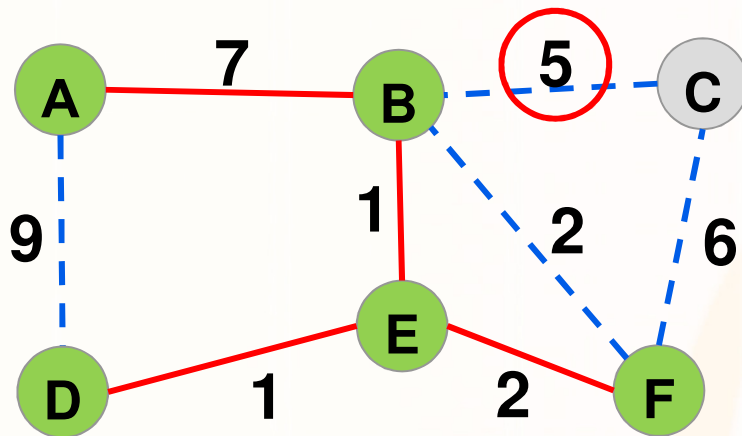
Γράφος G



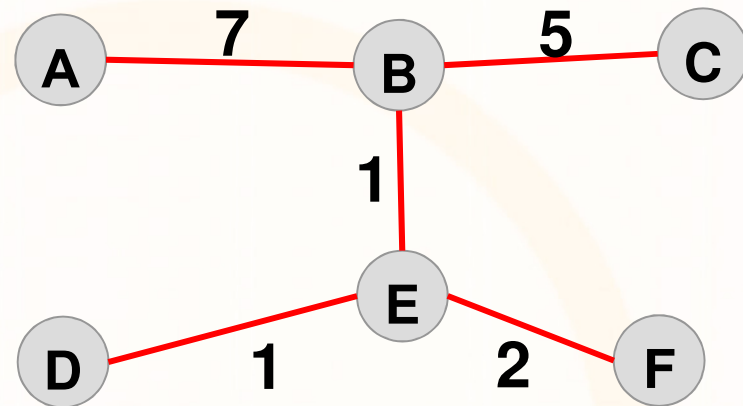
4



5



6





# 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

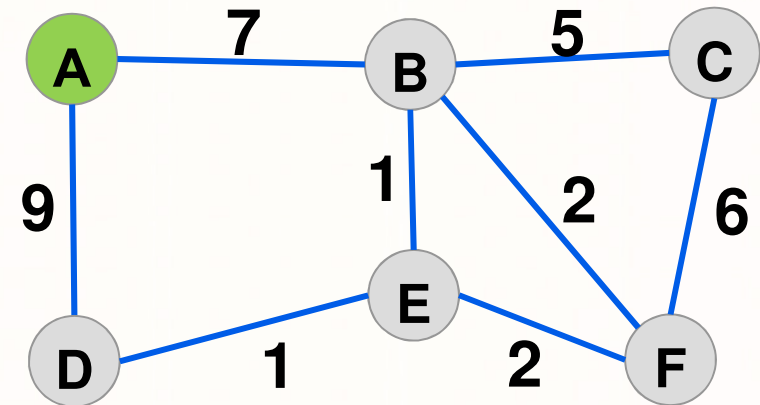
closest:	0	0	0	0	0	0
----------	---	---	---	---	---	---

distance:	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
-----------	----------	----------	----------	----------	----------	----------

# Υλοποίηση Αλγόριθμου Prim (συν.)

- Μετά 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:	$\infty$	7	$\infty$	9	$\infty$	$\infty$



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

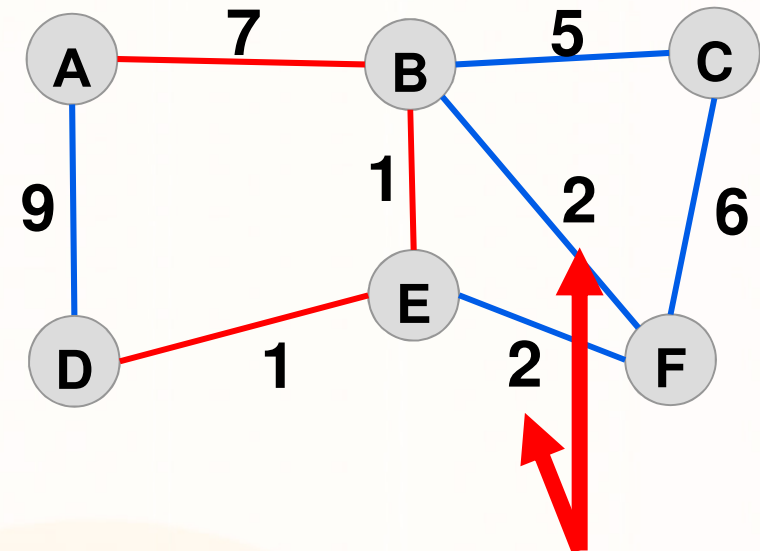
	A	B	C	D	E	F
visited:	1	1	0	0	0	0
closest:	0	A	B	A	B	B
distance:	$\infty$	7	5	9	1	2

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

# Υλοποίηση Αλγόριθμου Prim (συν.)

- Μετά 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>B</b>
distance:	$\infty$	7	5	1	1	2

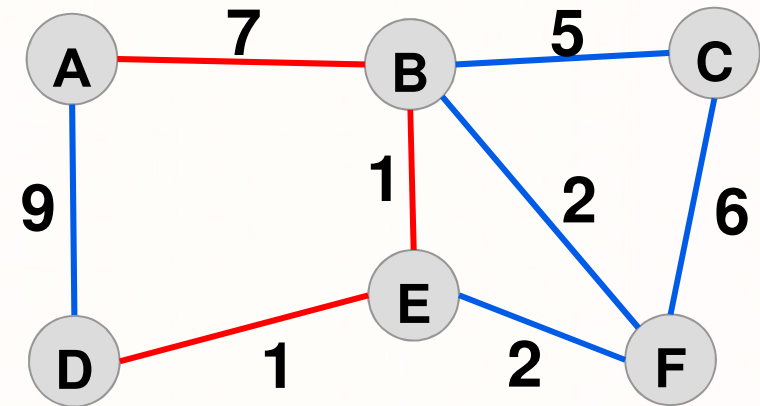


Δυο τρόποι να πάμε στο **F**, διατηρούμε τον ένα.

# Υλοποίηση Αλγόριθμου Prim (συν.)

- Μετά την εισαγωγή του **D** στο Δένδρο

	A	B	C	D	E	F
visited:	1	1	0	1	1	0
closest:	0	A	B	E	B	B
distance:	$\infty$	7	5	1	1	2



# 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 (περιέχει 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 (weight(v,w) < distance[w]) {  
        distance[w] = 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  $\Theta(|V|^2)$ .
- Μπορούμε να βελτιώσουμε τον αλγόριθμο;
- Ναι με την χρήση σωρών
- Με την χρήση σωρών ο αλγόριθμος μπορεί να υλοποιηθεί σε  $O(|E| \cdot \log |E|)$ , όπου  $E$  οι ακμές του γράφου.
- Ωστόσο δεν θα μελετήσουμε αυτή την υλοποίηση