ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΑΝΑΛΥΣΗ ΑΛΓΟΡΙΘΜΩΝ

Ενότητα 11: Minimum Spanning Trees

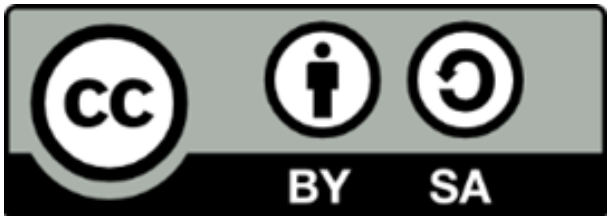
Αλγόριθμος Prim

Αλγόριθμος Kruskal

Μαρία Σατρατζέμη
Τμήμα Εφαρμοσμένης Πληροφορικής

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

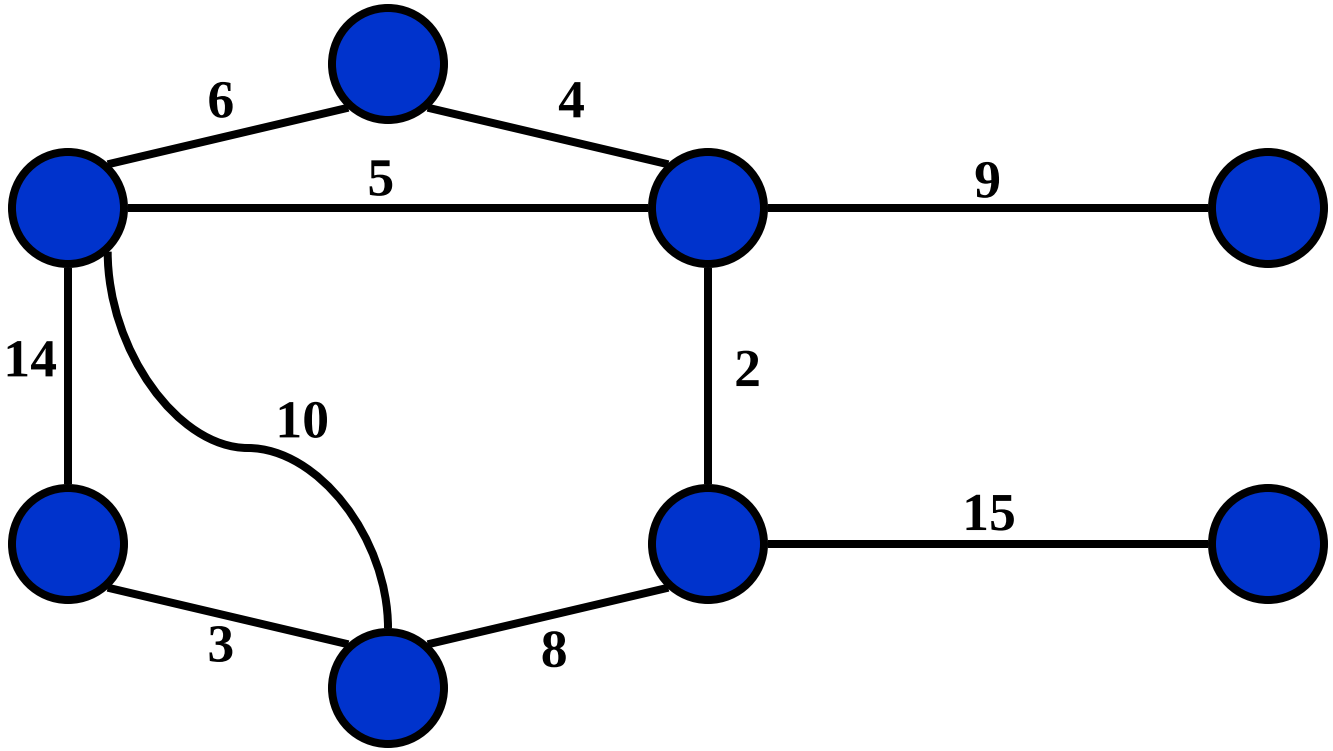


Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Μακεδονίας» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

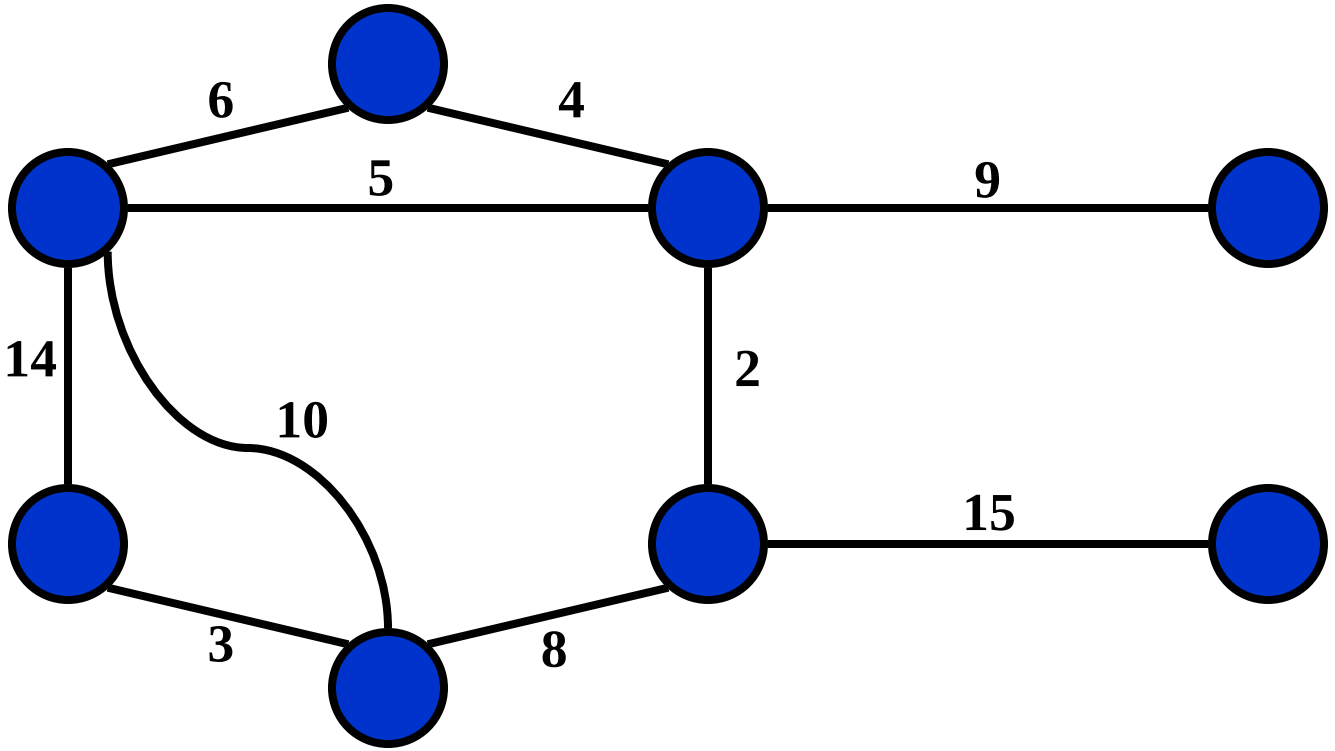
Minimum Spanning Tree

- **Πρόβλημα**: Για δοσμένο συνεκτικό, μη προσανατολισμένο, με βάρος γράφημα.



Minimum Spanning Tree

- **Πρόβλημα**: Για δοσμένο συνεκτικό, μη προσανατολισμένο, με βάρος γράφημα, να βρεθεί ένα *δένδρο κάλυμμα* (*spanning tree)* χρησιμοποιώντας ακμές που ελαχιστοποιούν το συνολικό βάρος.



Ορισμοί

Δένδρο κάλυμμα (*spanning tree)* είναι ένα μερικό γράφημα του συνεκτικού γραφήματος G που είναι δένδρο.

Δένδρο: συνεκτικό άκυκλο γράφημα

Ισοδύναμες προτάσεις

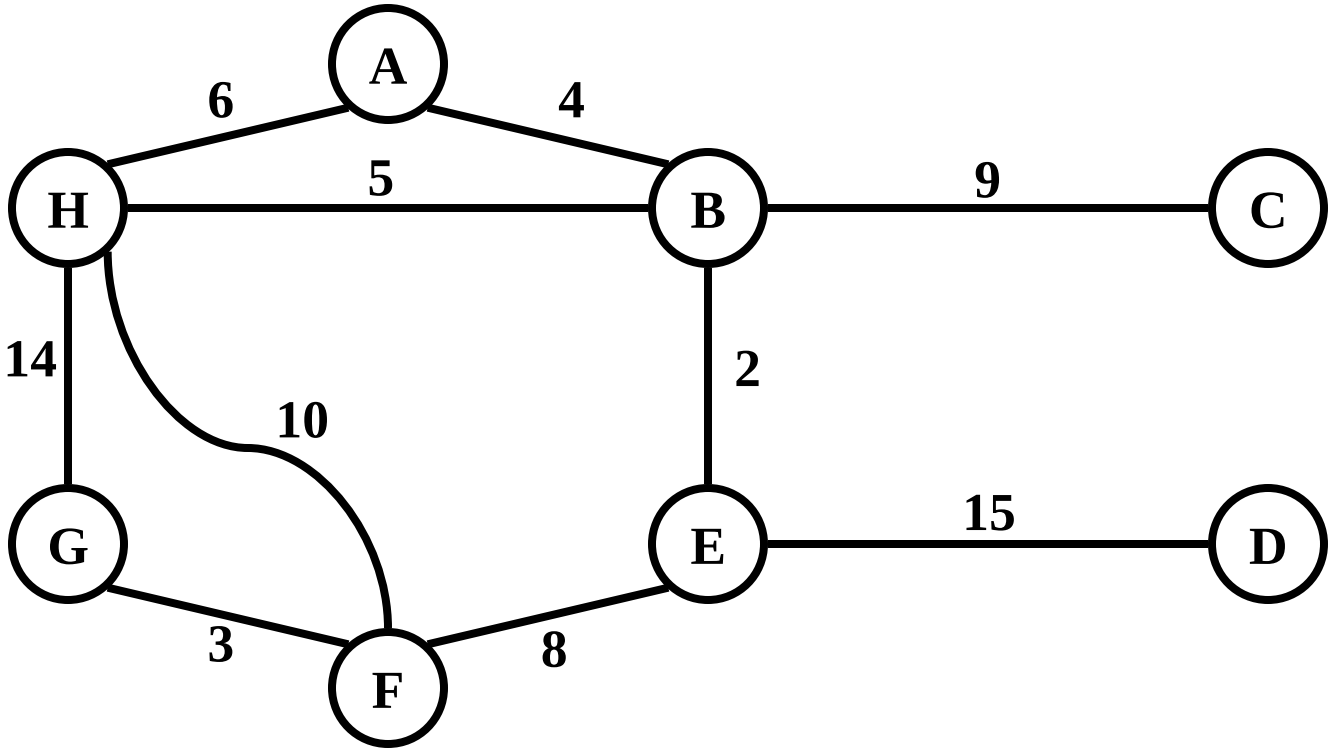
- Συνεκτικό με n-1 ακμές
- Άκυκλο με n-1 ακμές
- Κάθε ζεύγος κορυφών συνδέεται με ένα μόνο μονοπάτι
- Αν ενωθούν 2 μη γειτονικές κορυφές με μια ακμή σχηματίζεται κύκλο.

Ορισμοί

Ένα γράφημα g ονομάζεται *μερικό γράφημα (partial graph)* του γραφήματος G= (V, E) αν οι κορυφές του g είναι οι κορυφές του G και οι ακμές του g είναι υποσύνολο των ακμών του G, δηλαδή g = (V, E'), όπου $E \subset E'$.
Ελάχιστο δένδρο κάλυμμα (minimum spanning tree) το δένδρο κάλυμμα με το ελάχιστο συνολικό βάρος ακμών.

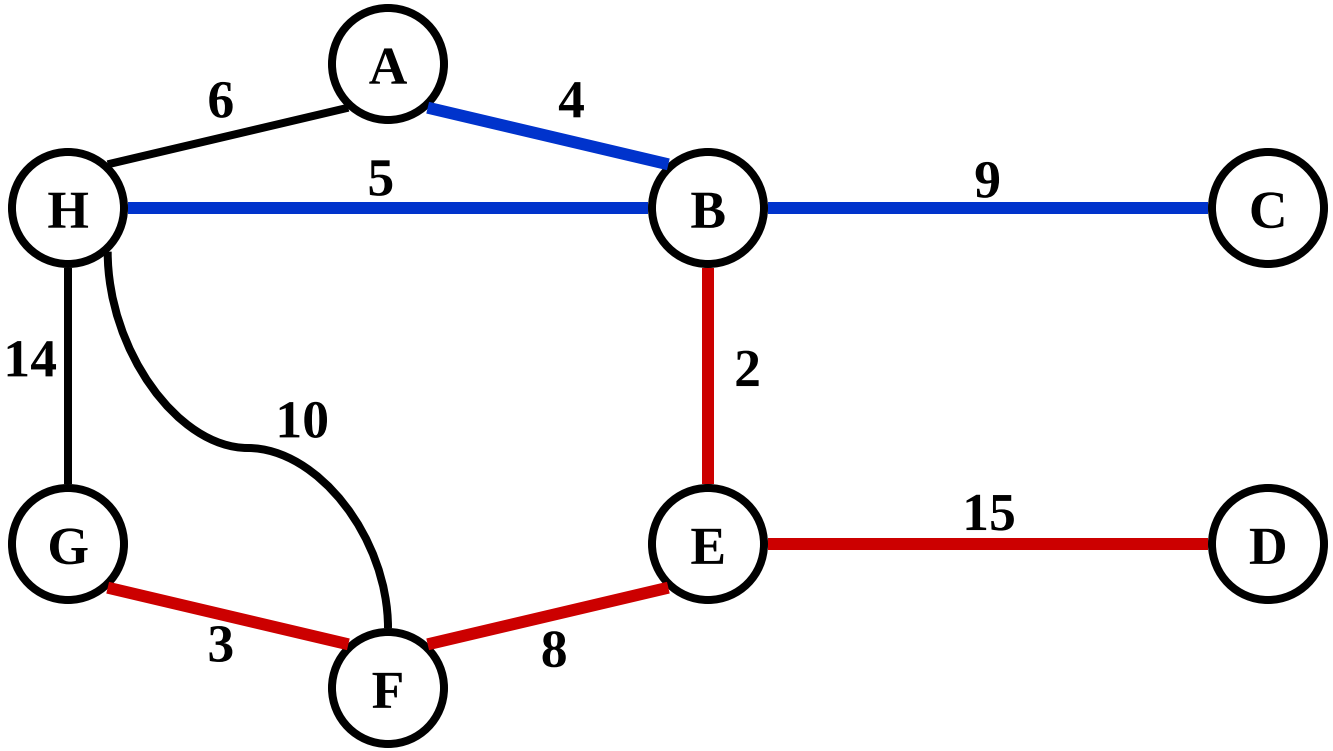
Minimum Spanning Tree

- *Ποιες ακμές σχηματίζουν Ελάχιστο Δένδρο Κάλυμμα (Minimum Spanning Tree MST) στο παρακάτω γράφημα?*



Minimum Spanning Tree

- Απάντηση:



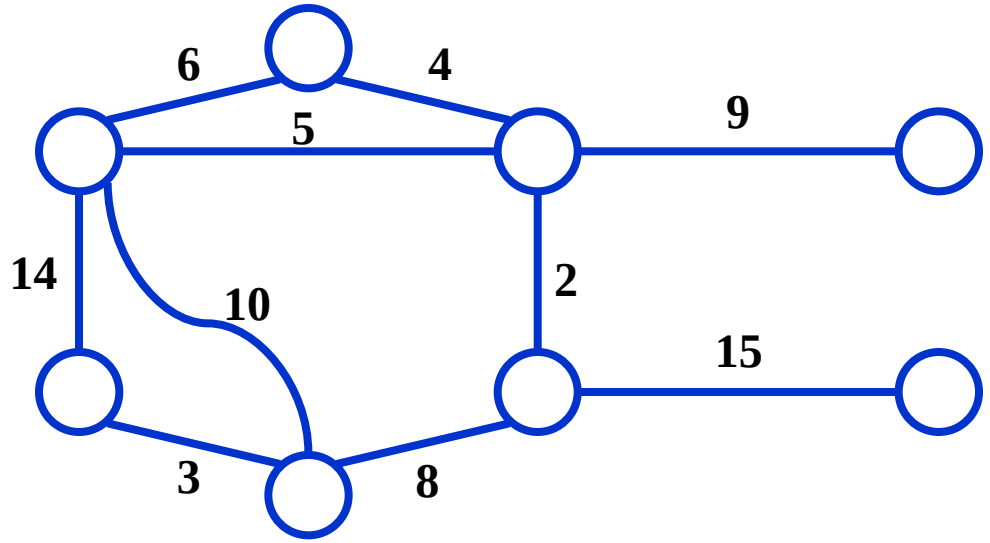
Prim's Algorithm

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```

w(u, v) βάρος ακμής (u, v)

Prim's Algorithm

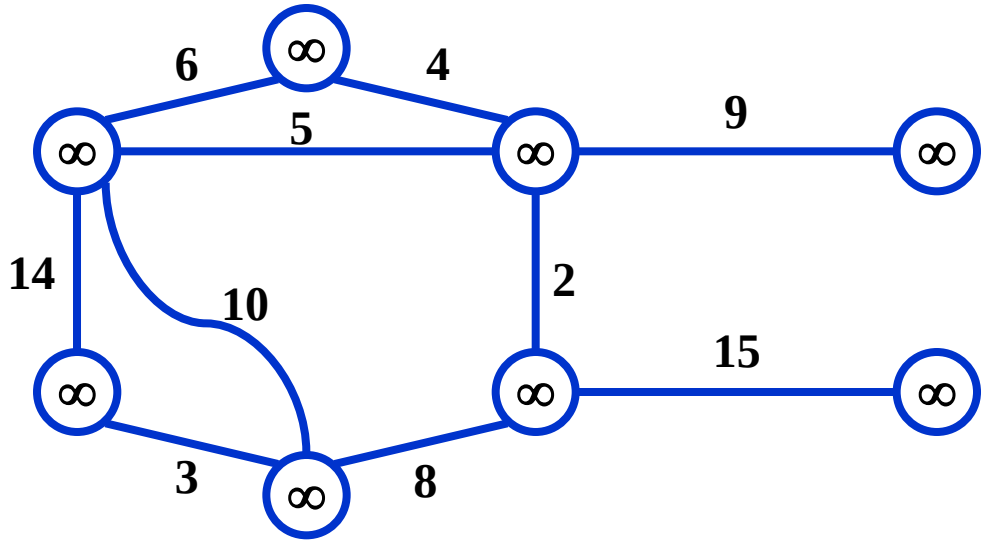
```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```



Τρέχοντας ένα παράδειγμα

Prim's Algorithm

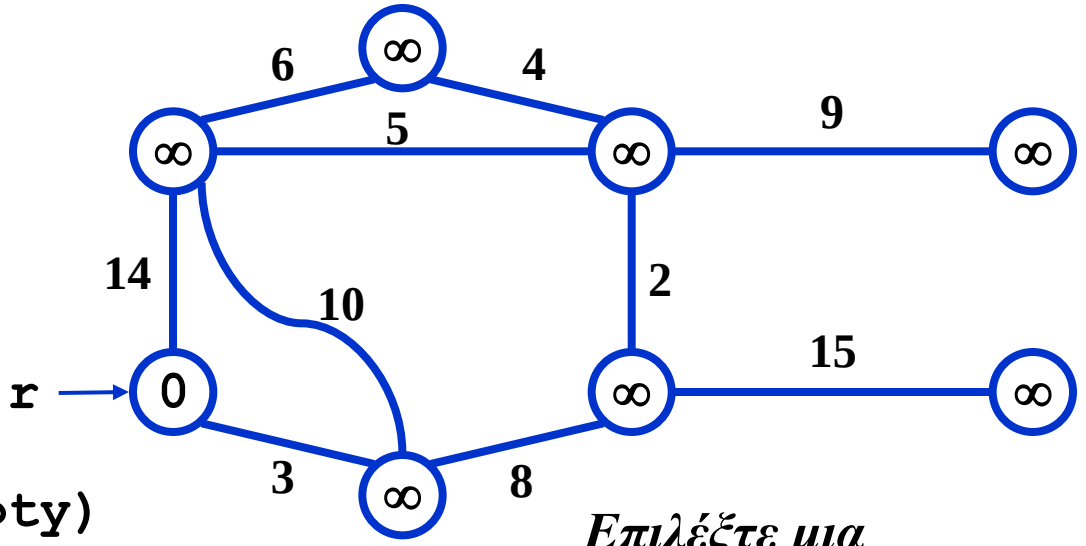
```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```



Τρέχοντας ένα παράδειγμα

Prim's Algorithm

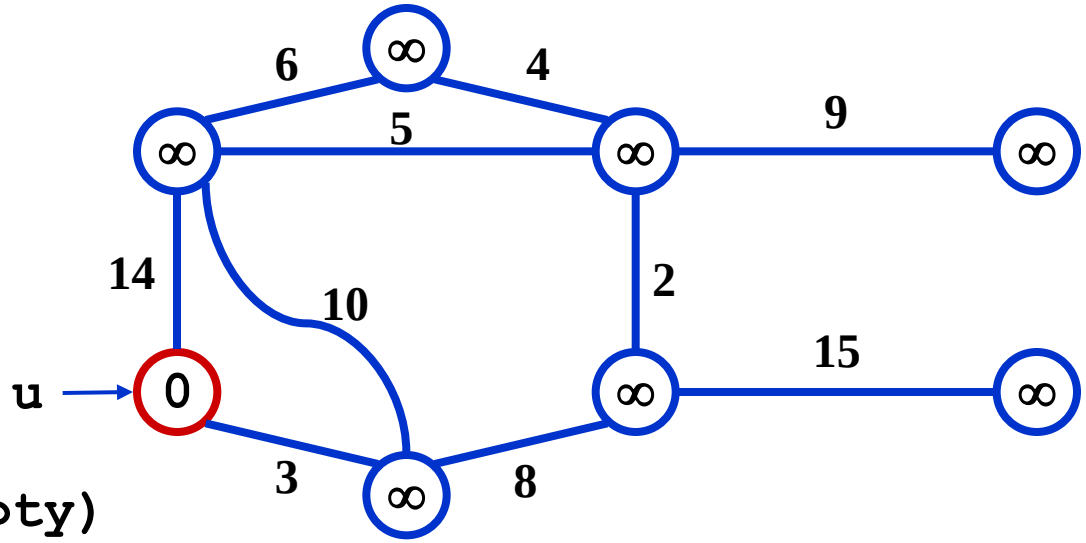
```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```



Επιλέξτε μια οποιαδήποτε κορυφή ως αρχική r

Prim's Algorithm

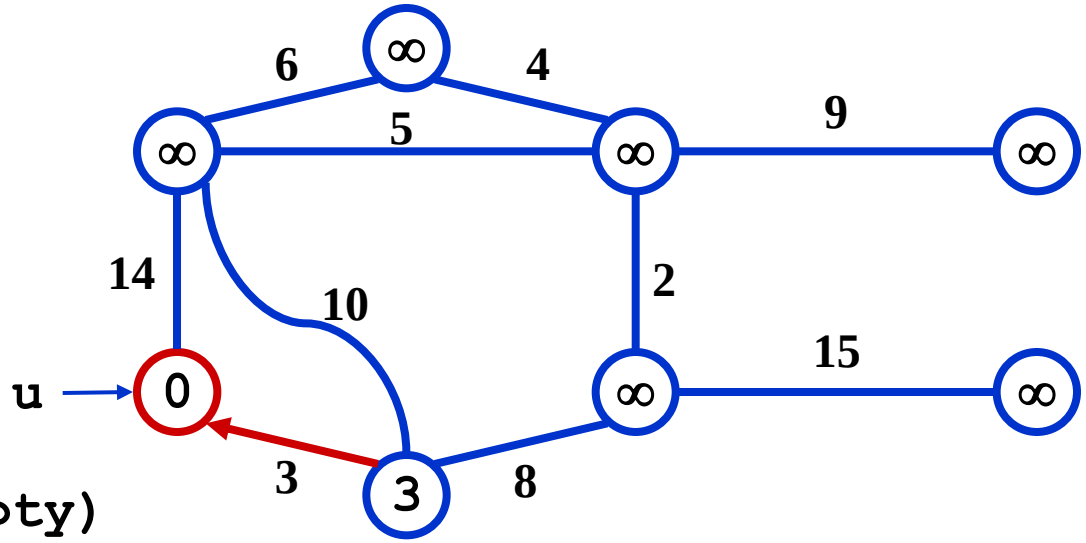
```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```



Κόκκινες κορυφές έχουν απομακρυνθεί από την Q

Prim's Algorithm

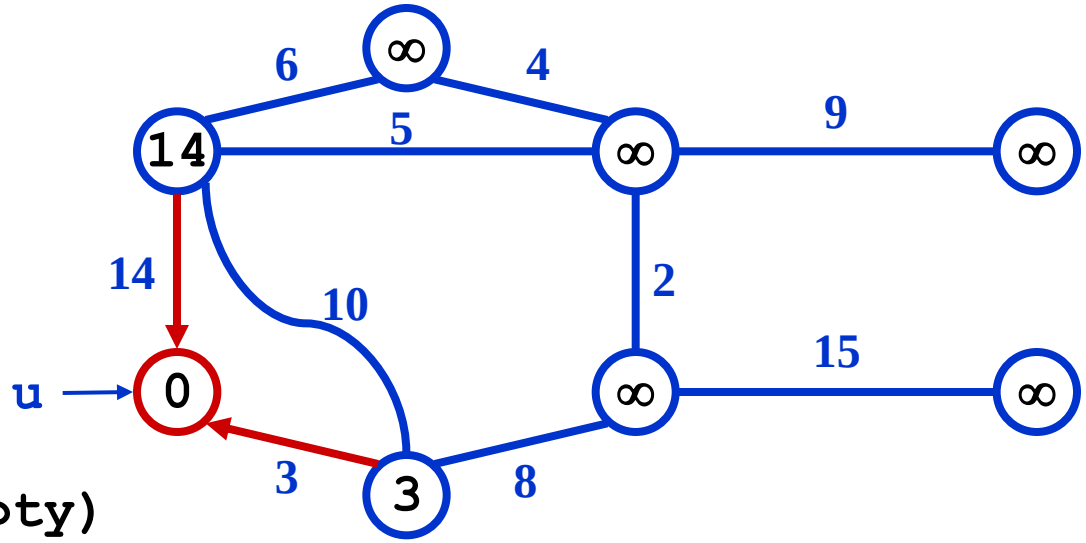
```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```



Κόκκινες κορυφές έχουν απομακρυνθεί από την Q

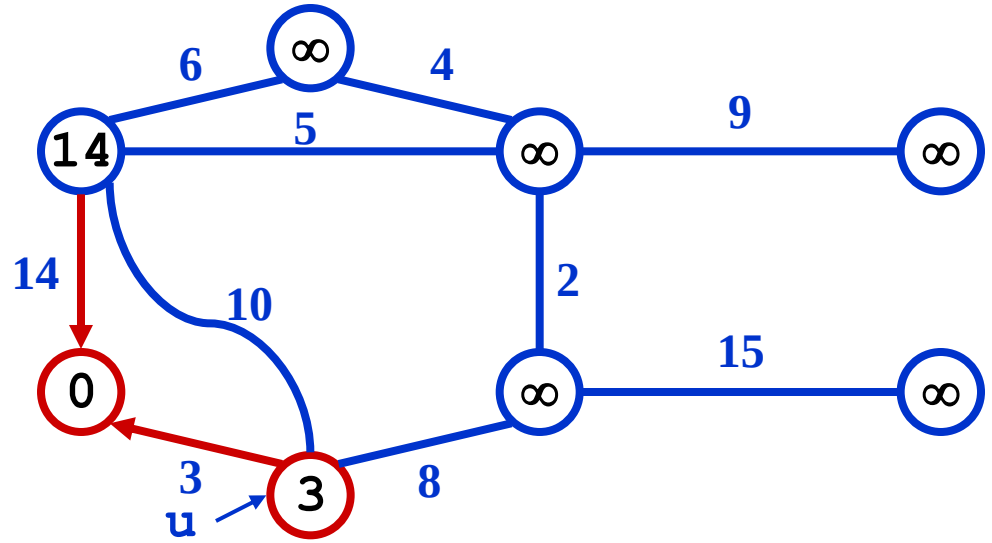
Prim's Algorithm

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```



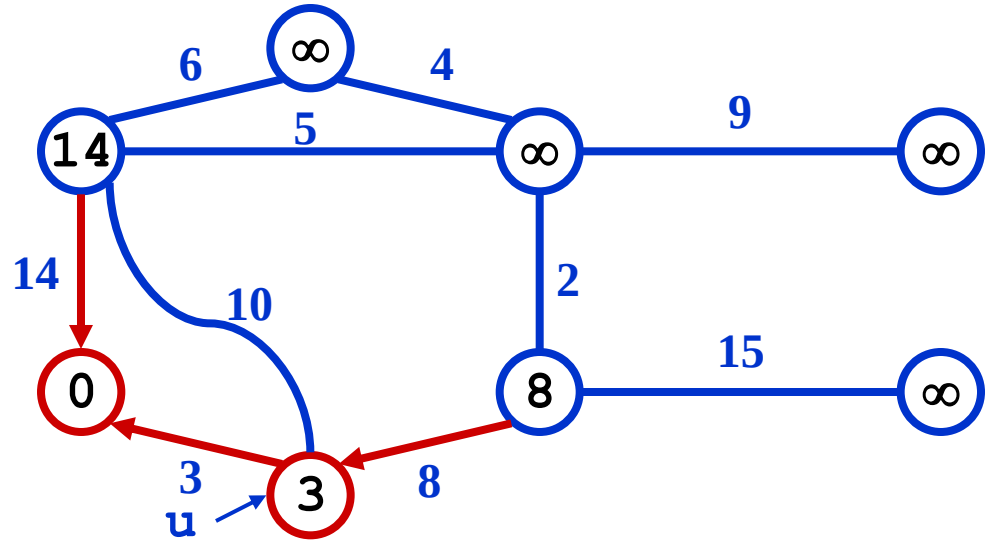
Prim's Algorithm

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```



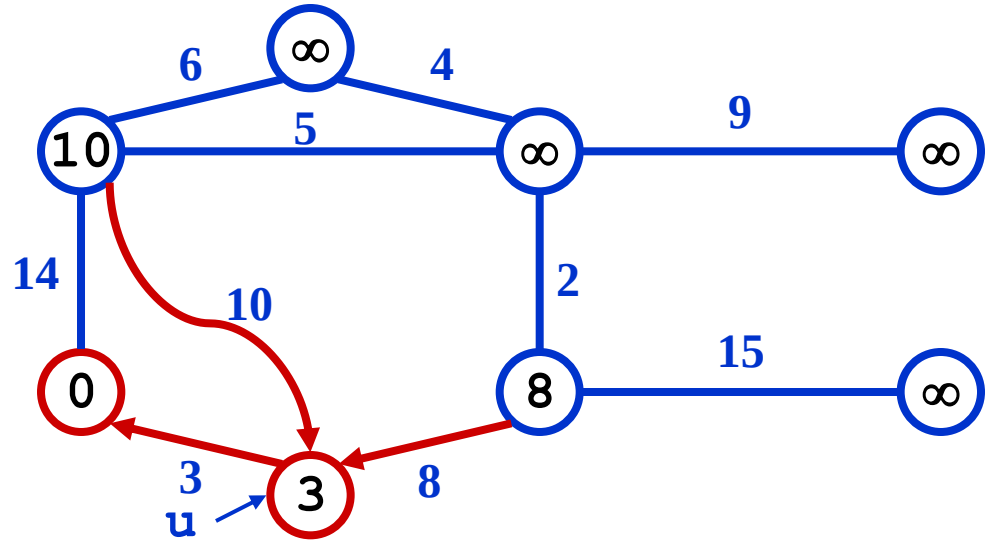
Prim's Algorithm

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```



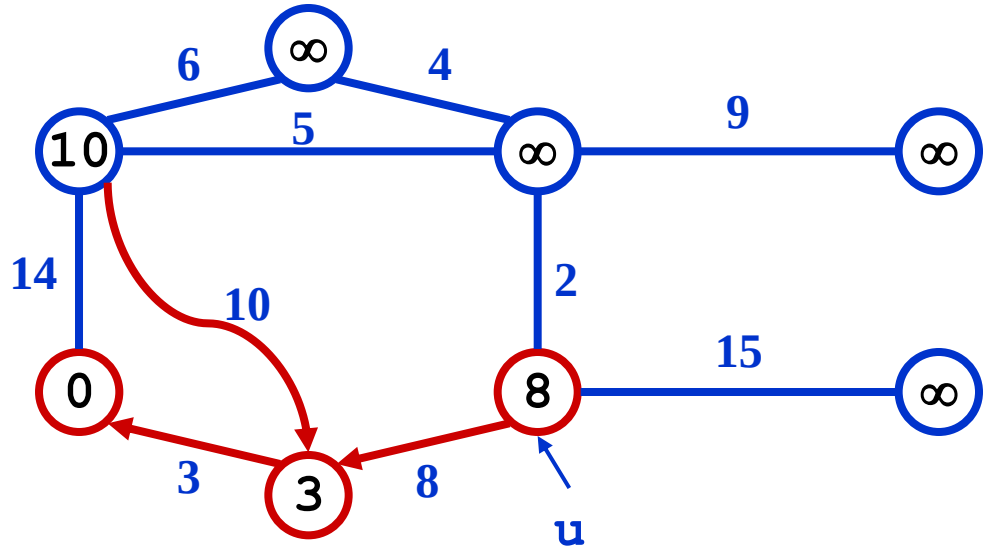
Prim's Algorithm

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```



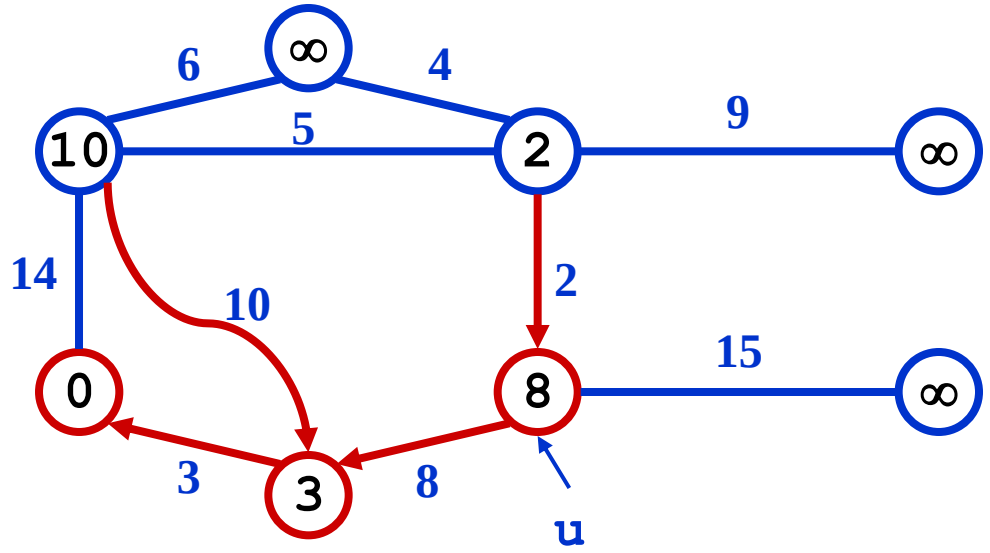
Prim's Algorithm

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```



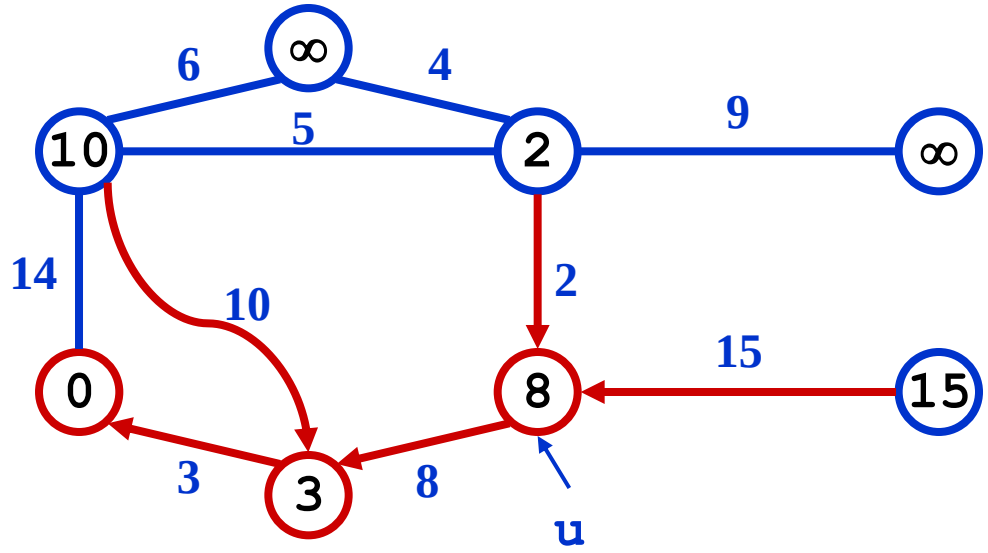
Prim's Algorithm

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```



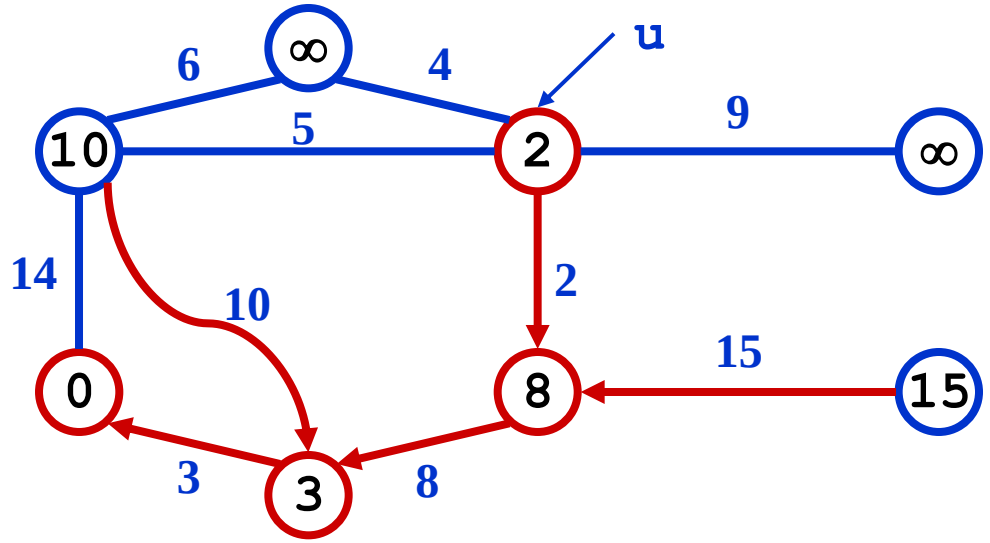
Prim's Algorithm

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```



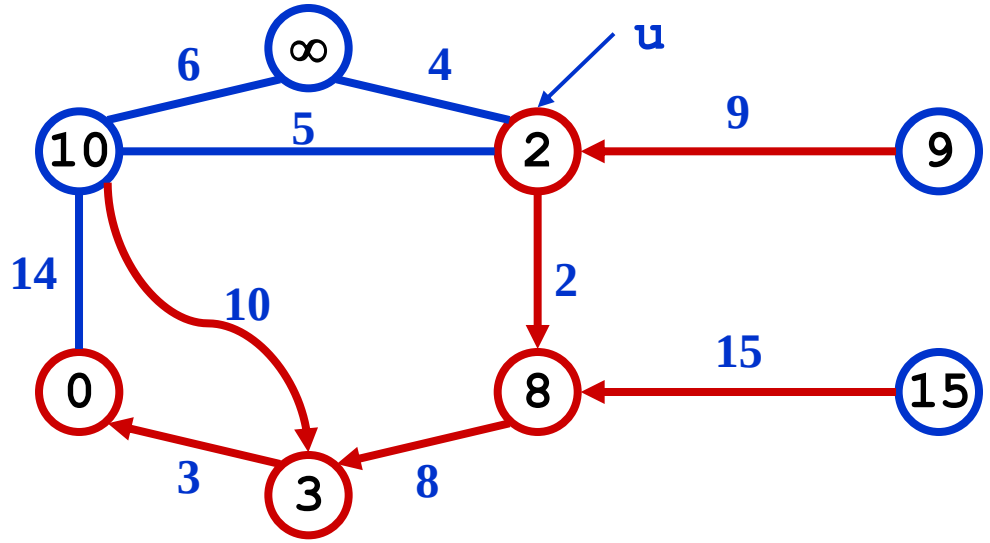
Prim's Algorithm

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```



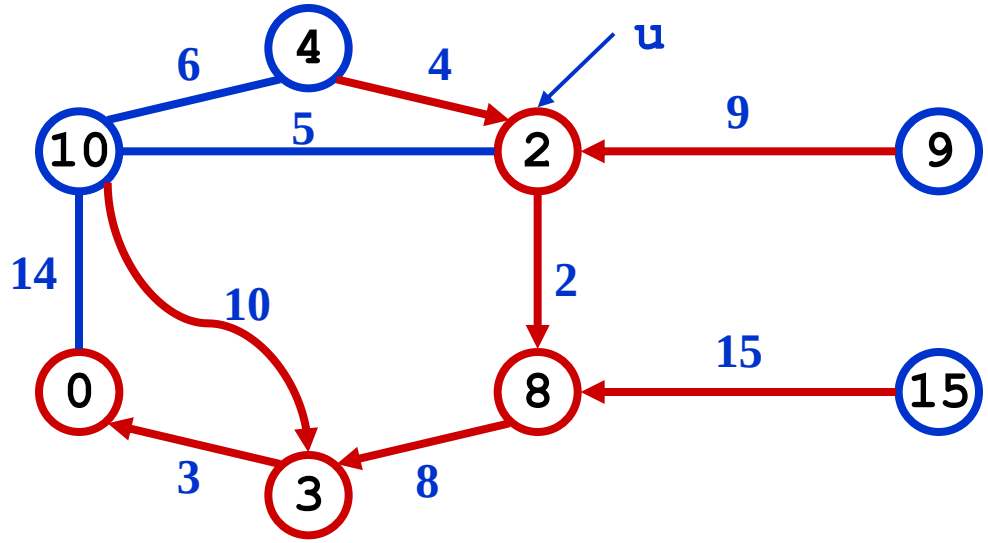
Prim's Algorithm

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```



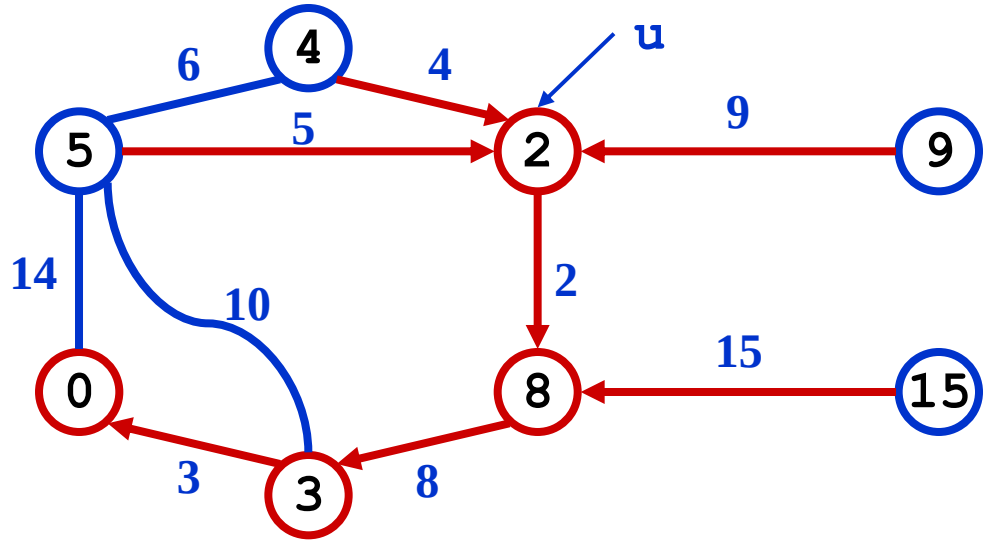
Prim's Algorithm

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```



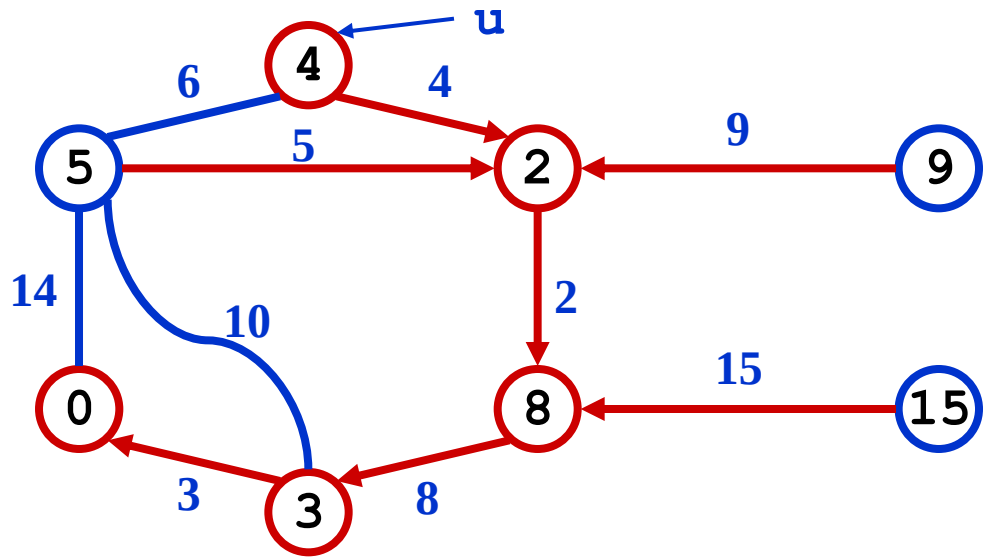
Prim's Algorithm

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```



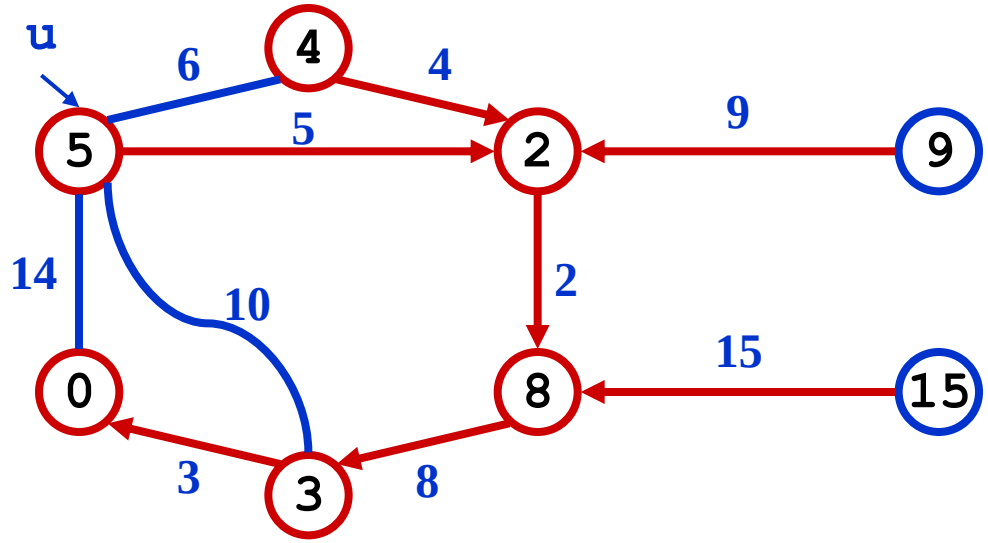
Prim's Algorithm

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```



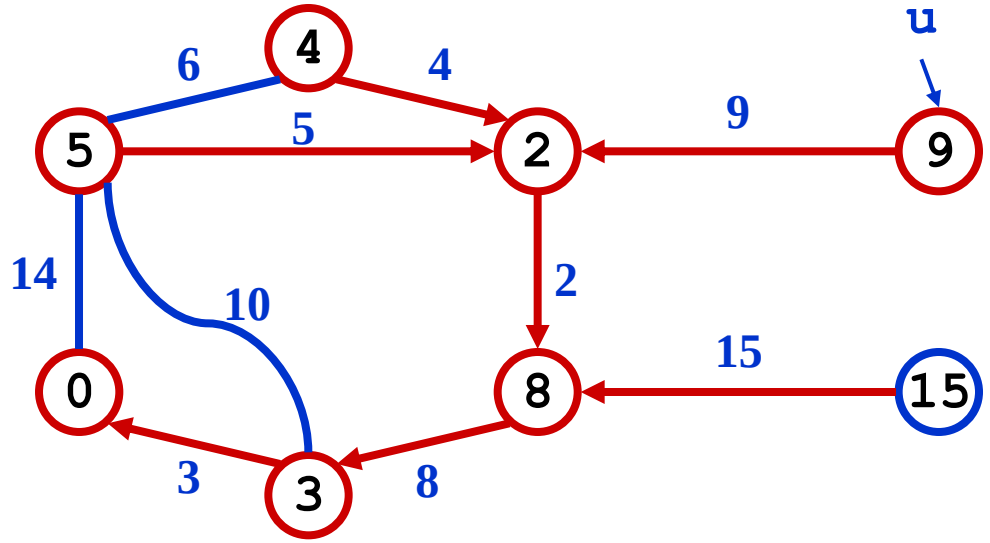
Prim's Algorithm

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```



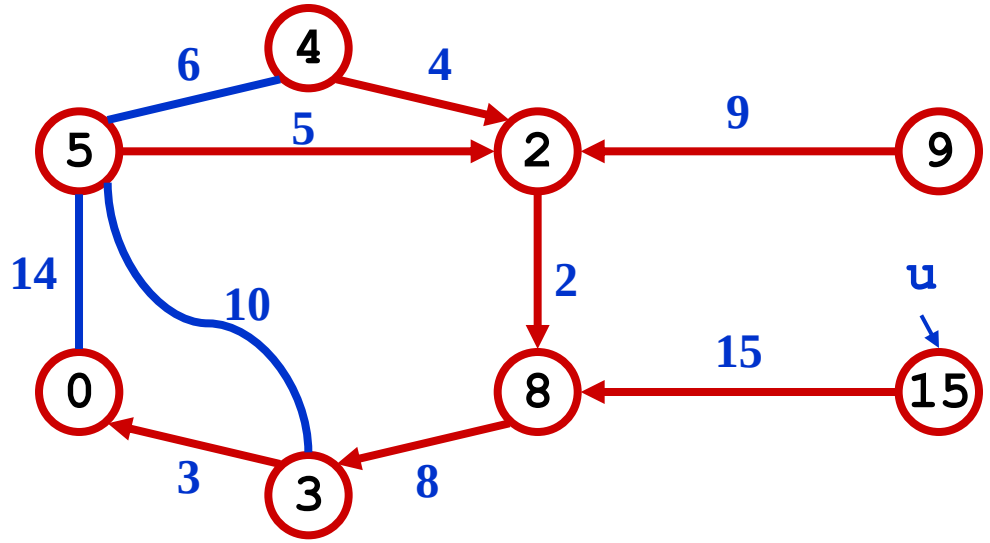
Prim's Algorithm

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```



Prim's Algorithm

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```



Ανάλυση: Prim's Algorithm

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```

Ποιο είναι το κρυμμένο κόστος στον αλγόριθμο?

Ανάλυση: Prim's Algorithm

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                DecreaseKey(v, w(u,v));
```

Ανάλυση : Prim's Algorithm

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                DecreaseKey(v, w(u,v));
```

Πόσο συχνά καλείται η ExtractMin()?
Πόσο συχνά καλείται η DecreaseKey()?

Ανάλυση: Prim's Algorithm

Θ(V)
συνολικά

```
MST-Prim(G, w, r)
1     Q = V[G];
2     for each u ∈ Q
3         key[u] = ∞;
4     key[r] = 0;
5     p[r] = NULL;
6     while (Q not empty)
7         u = ExtractMin(Q);
8         for each v ∈ Adj[u]
9             if (v ∈ Q and w(u,v) < key[v])
10                p[v] = u;
11                key[v] = w(u,v);
```

Ανάλυση : Prim's Algorithm

Θ(V)
συνολικά

|V|
φορές

```
MST-Prim(G, w, r)
1     Q = V[G];
2     for each u ∈ Q
3         key[u] = ∞;
4     key[r] = 0;
5     p[r] = NULL;
6     while (Q not empty)
7         u = ExtractMin(Q);
8         for each v ∈ Adj[u]
9             if (v ∈ Q and w(u,v) < key[v])
10                p[v] = u;
11                key[v] = w(u,v);
```

Ανάλυση : Prim's Algorithm

Θ(V) *συνολικά*

|V| *φορές*

degree(u) *φορές*

```
MST-Prim(G, w, r)
1    Q = V[G];
2    for each u ∈ Q
3        key[u] = ∞;
4    key[r] = 0;
5    p[r] = NULL;
6    while (Q not empty)
7        u = ExtractMin(Q);
8        for each v ∈ Adj[u]
9            if (v ∈ Q and w(u,v) < key[v])
10               p[v] = u;
11               key[v] = w(u,v);
```

Ανάλυση : Prim's Algorithm

```
MST-Prim(G, w, r)
1     Q = V[G];
2     for each u ∈ Q
3         key[u] = ∞;
4     key[r] = 0;
5     p[r] = NULL;
6     while (Q not empty)
7         u = ExtractMin(Q);
8         for each v ∈ Adj[u]
9             if (v ∈ Q and w(u,v) < key[v])
10                p[v] = u;
11                key[v] = w(u,v);
```

Θ(V) *συνολικά*

|V| *φορές*

degree(u) *φορές*

$$\sum_v |Adj(v)| = \sum_v \deg(v) = 2|E|$$

Θ(E) **DecreaseKey**

Ανάλυση : Prim's Algorithm

Χρόνος = $\Theta(V)\cdot T_{\text{ExtractMin}} + \Theta(E)\cdot T_{\text{DecreaseKey}}$

Q	$T_{\text{ExtractMin}}$	$T_{\text{DecreaseKey}}$	Συνολικά
πίνακας	$O(V)$	$O(1)$	$O(V^2)$
Διωνυμικός σωρός	$O(\log_2 V)$	$O(\log_2 V)$	$O(E \log_2 V)$
Σωρός Fibonacci	$O(\log_2 V)$	$O(1)$	$O(E+V \log_2 V)$

Ανάλυση : Prim's Algorithm

Η επίδοση του αλγορίθμου του Prim εξαρτάται από τον τρόπο υλοποίησης της ουράς προτεραιότητας ελαχίστου Q. Αν η Q υλοποιηθεί ως διωνυμικός σωρός ελαχίστου η απόδοση αρχικών τιμών στις γραμμές 1-5 μπορεί να επιτευχθεί σε χρόνο $O(V)$ μέσω της διαδικασίας *Κατασκευή Σωρού Ελαχίστου.*

Το σώμα του βρόχου **while** εκτελείται $|V|$ φορές και δεδομένου ότι κάθε πράξη *Εξαγωγής Ελαχίστου* διαρκεί χρόνο $O(\log_2 V)$ ο συνολικός χρόνος για όλες τις κλήσεις Εξαγωγής Ελαχίστου είναι $O(V \log_2 V)$.

Ο βρόχος **for** στις γραμμές 8-11 εκτελείται συνολικά $O(E)$ φορές αφού το άθροισμα των μηκών όλων των λιστών γειτνίασης είναι $2|E|$. Εντός του βρόχου **for** ο έλεγχος «συμμετοχής» στη ουρά Q στη γραμμή 9 μπορεί να πραγματοποιηθεί σε σταθερό χρόνο μέσω μιας μοναδικής δύφιας μεταβλητής για κάθε κορυφή, που δείχνει αν η κορυφή ανήκει ή όχι στην Q και ενημερώνεται όταν η κορυφή αφαιρείται από την Q.

Η ανάθεση τιμής στη γραμμή 11 περιλαμβάνει μια σιωπηρή πράξη *Μείωσης Κλειδιού* στο σωρό ελαχίστου μπορεί να υλοποιηθεί σε χρόνο $O(\log_2 V)$.

Ο συνολικός χρόνος εκτέλεσης του αλγόριθμου του Prim είναι $O(V \log_2 V + E \log_2 V) = O(E \log_2 V)$.

Ανάλυση : Prim's Algorithm

```
MST-Prim(G, w, r)
1     Q = V[G];
2     for each u ∈ Q
3         key[u] = ∞;
4     key[r] = 0;
5     p[r] = NULL;
6     while (Q not empty)
7         u = ExtractMin(Q);
8         for each v ∈ Adj[u]
9             if (v ∈ Q and w(u,v) < key[v])
10                p[v] = u;
11                key[v] = w(u,v);
```

Ποιος είναι ο χρόνος εκτέλεσης?

Α: εξαρτάται από την υλοποίηση της *Q*

- με διωνυμικό σωρό ελαχίστων: $O(E \log_2 V)$

- με σωρό Fibonacci : $O(V \log_2 V + E)$

Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T ∪ {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

Kruskal's Algorithm

```
Kruskal()
{
1.      T = ∅;
2.      for each v ∈ V
3.          MakeSet(v);
4.      sort E by increasing edge weight w
5.      for each (u,v) ∈ E (in sorted order)
6.          if FindSet(u) ≠ FindSet(v)
7.              T = T U {{u,v}};
8.              Union(FindSet(u), FindSet(v));
}
```

Στις γραμμές 1-3 ορίζεται ένα αρχικό σύνολο Τ κενό και δημιουργούνται |V| δένδρα καθένα απ' αυτά περιέχει 1 κόμβο.

Στην 4 διατάσσονται οι ακμές του σε αύξουσα διάταξη ως προς το βάρος.

Kruskal's Algorithm

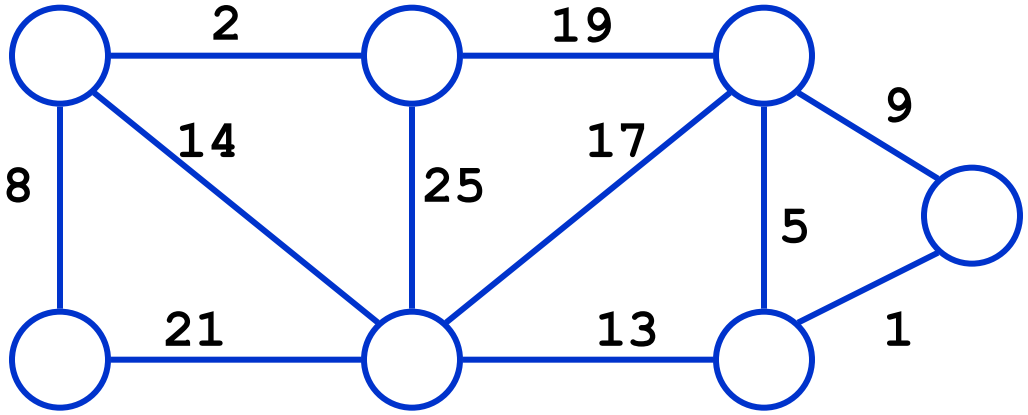
```
Kruskal()
{
1.      T = ∅;
2.      for each v ∈ V
3.          MakeSet(v);
4.      sort E by increasing edge weight w
5.      for each (u,v) ∈ E (in sorted order)
6.          if FindSet(u) ≠ FindSet(v)
7.              T = T U {{u,v}};
8.              Union(FindSet(u), FindSet(v));
}
```

Ο βρόχος στις γραμμές 5-8 ελέγχει για κάθε (u, v) αν τα άκρα της u και v ανήκουν στο ίδιο δένδρο. Αν ανήκουν τότε η προσθήκη της (u, v) θα δημιουργήσει κύκλο και άρα απορρίπτεται. Διαφορετικά οι 2 κόμβοι ανήκουν σε διαφορετικά δένδρα η ακμή (u, v) προστίθεται στο Τ στη γραμμή 7 και οι κόμβοι των 2 δένδρων συγχωνεύονται στη γραμμή 8.

Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

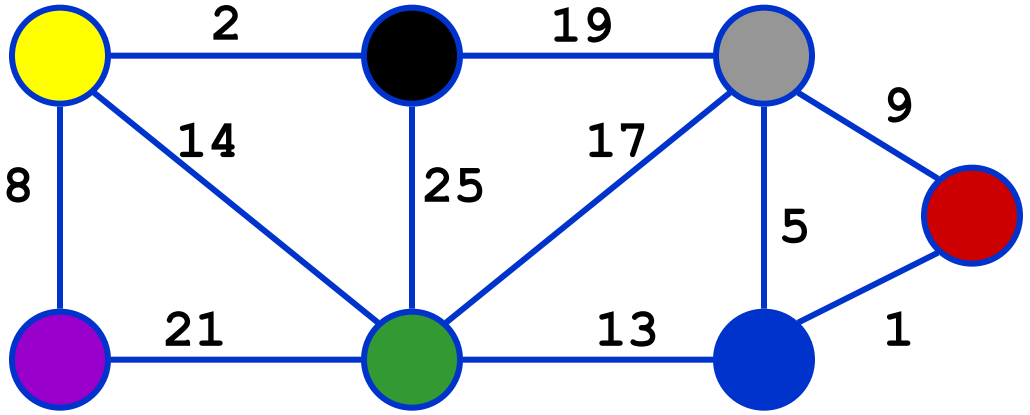
Εκτέλεση αλγόριθμου:



Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

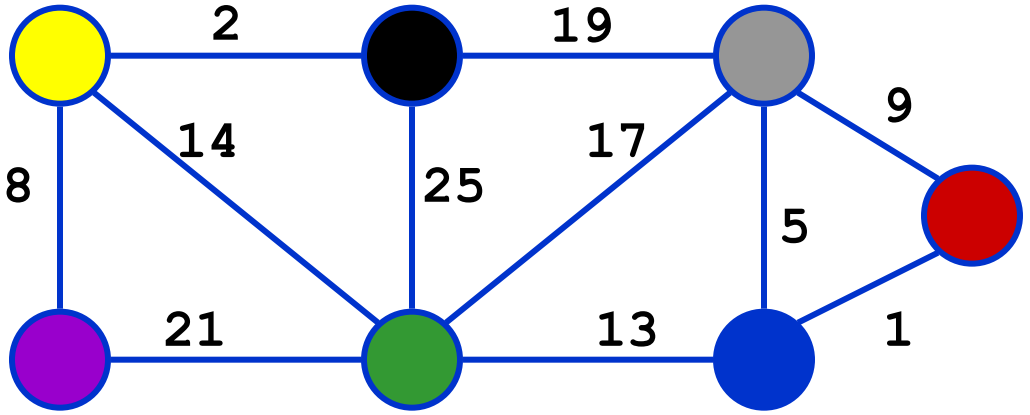
Εκτέλεση αλγόριθμου:



Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
  { sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

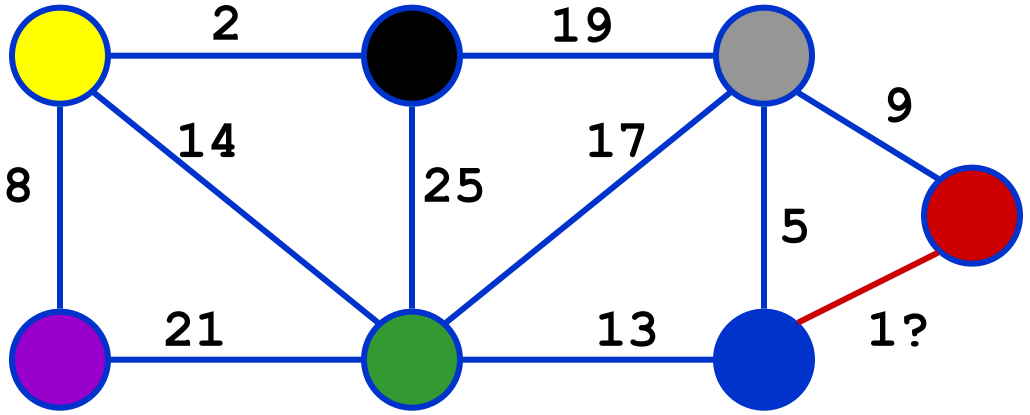
Εκτέλεση αλγόριθμου:



Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

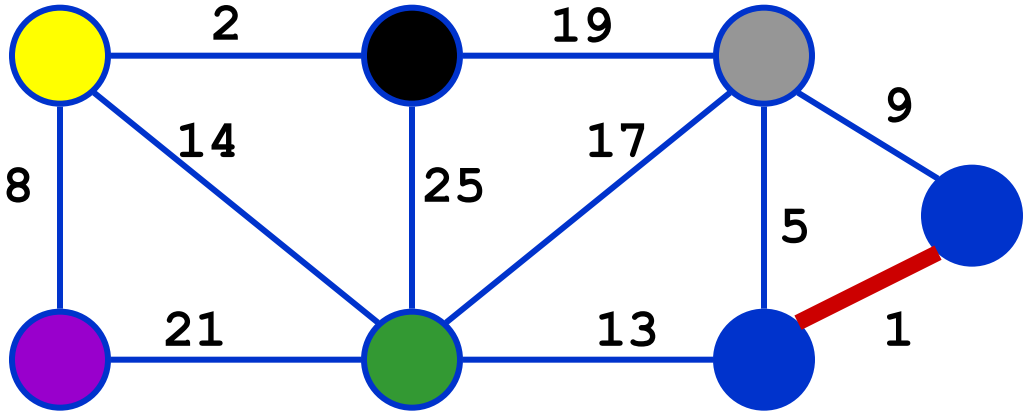
Εκτέλεση αλγόριθμου:



Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

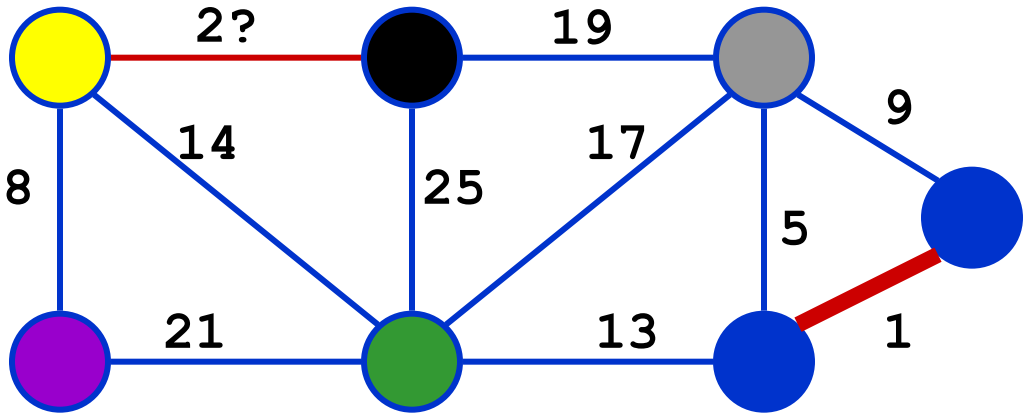
Εκτέλεση αλγόριθμου:



Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

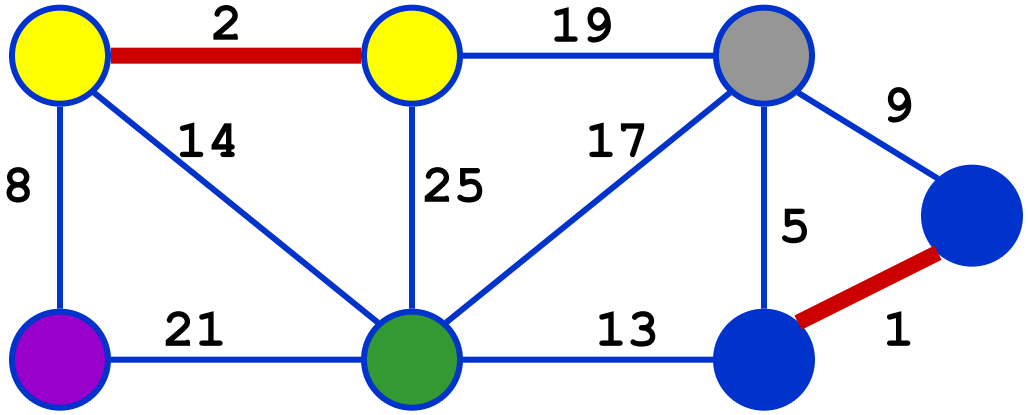
Εκτέλεση αλγόριθμου:



Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

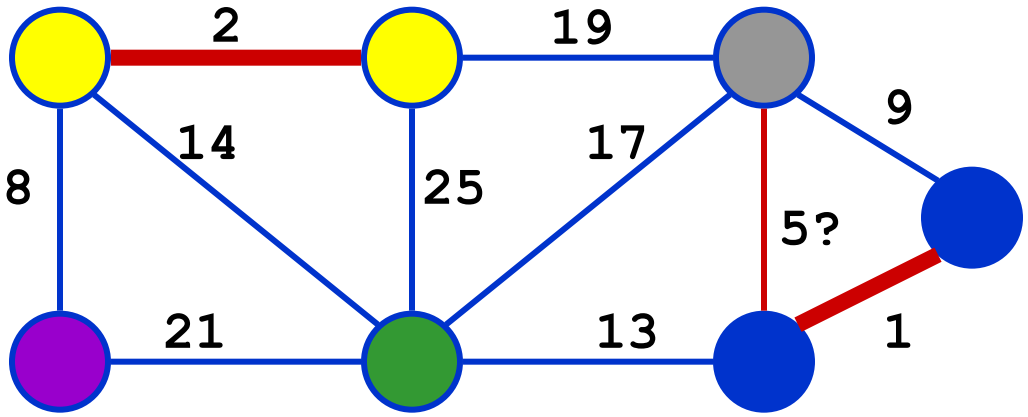
Εκτέλεση αλγόριθμου:



Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

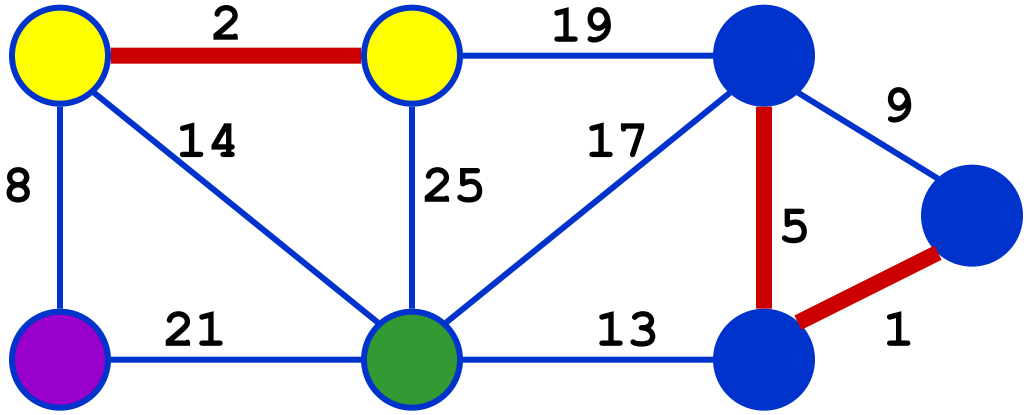
Εκτέλεση αλγόριθμου:



Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

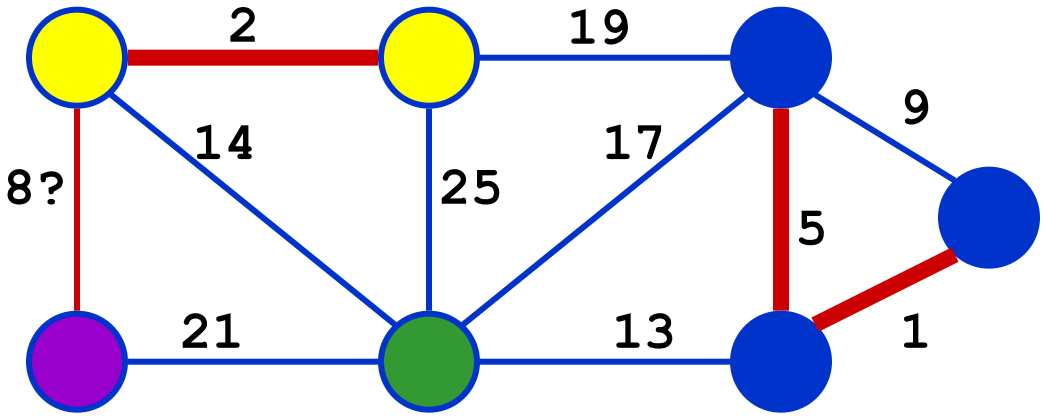
Εκτέλεση αλγόριθμου:



Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

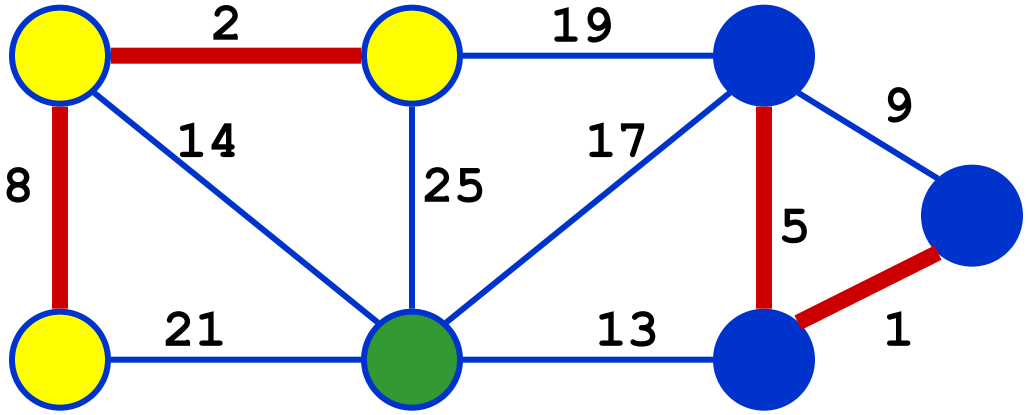
Εκτέλεση αλγόριθμου:



Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

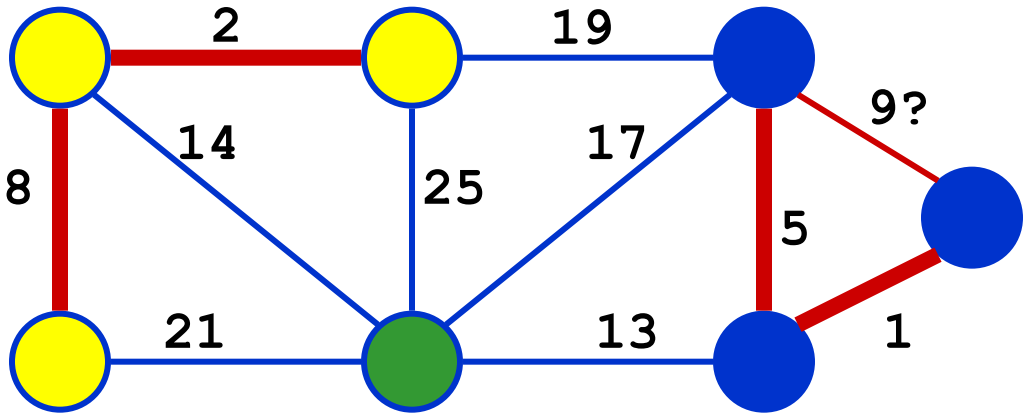
Εκτέλεση αλγόριθμου:



Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

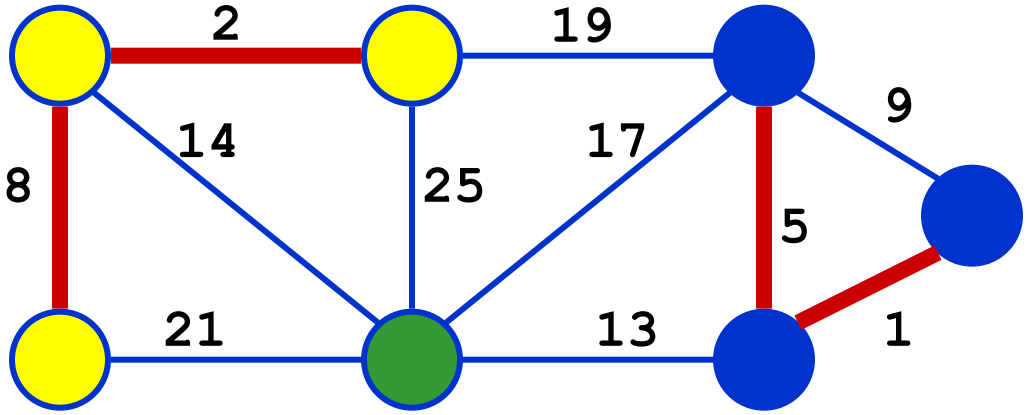
Εκτέλεση αλγόριθμου:



Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

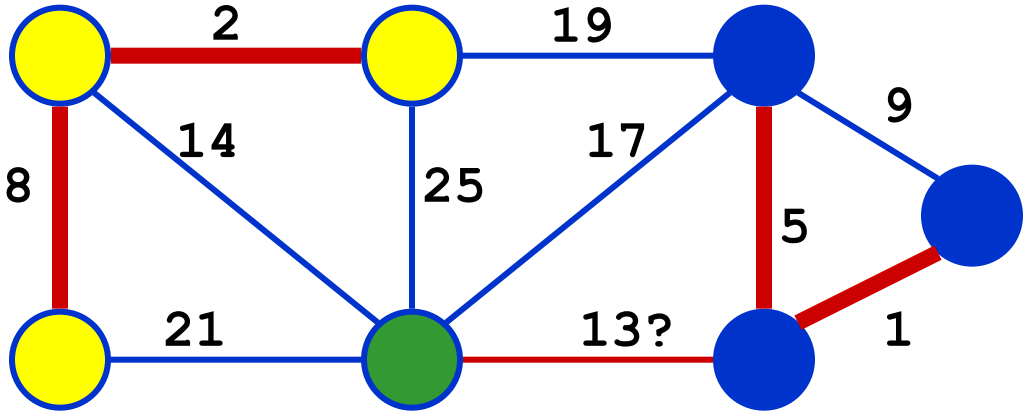
Εκτέλεση αλγόριθμου:



Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

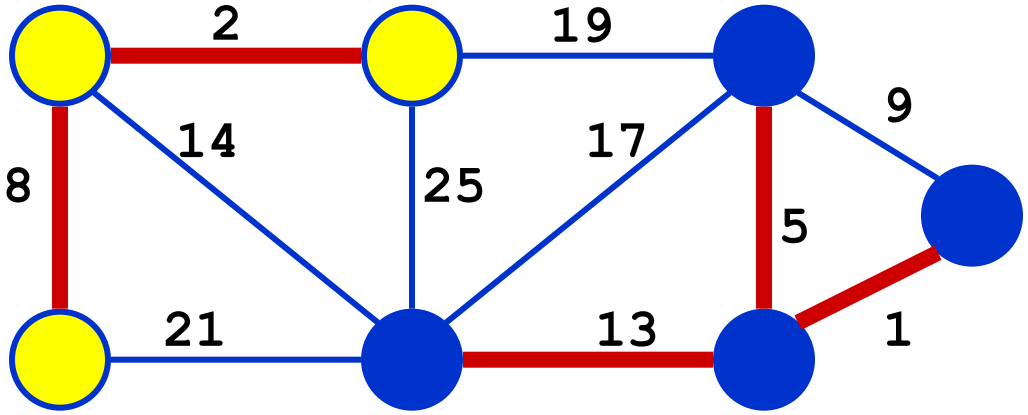
Εκτέλεση αλγόριθμου:



Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

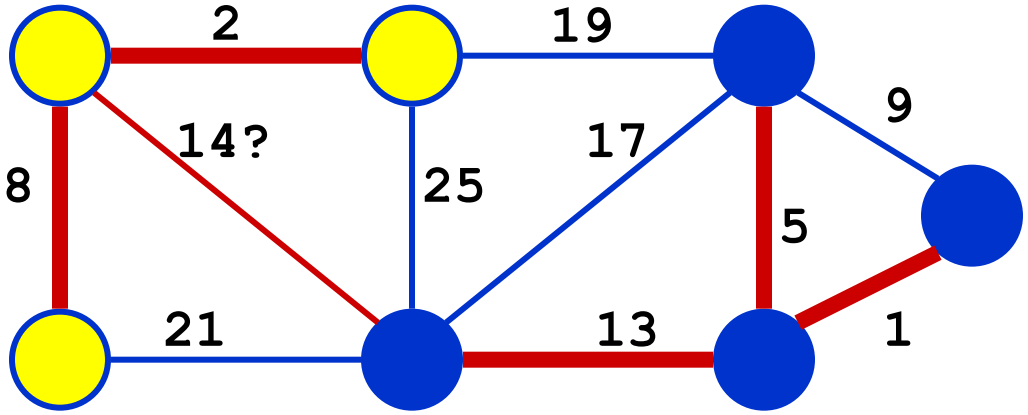
Εκτέλεση αλγόριθμου:



Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

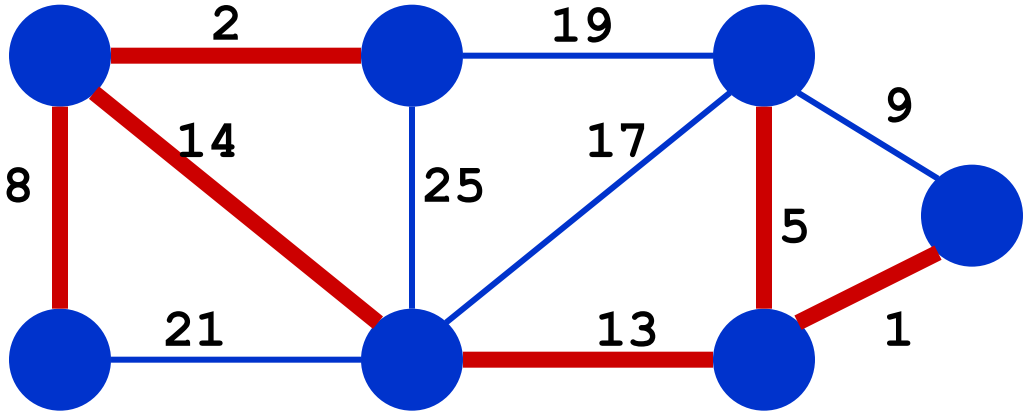
Εκτέλεση αλγόριθμου:



Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

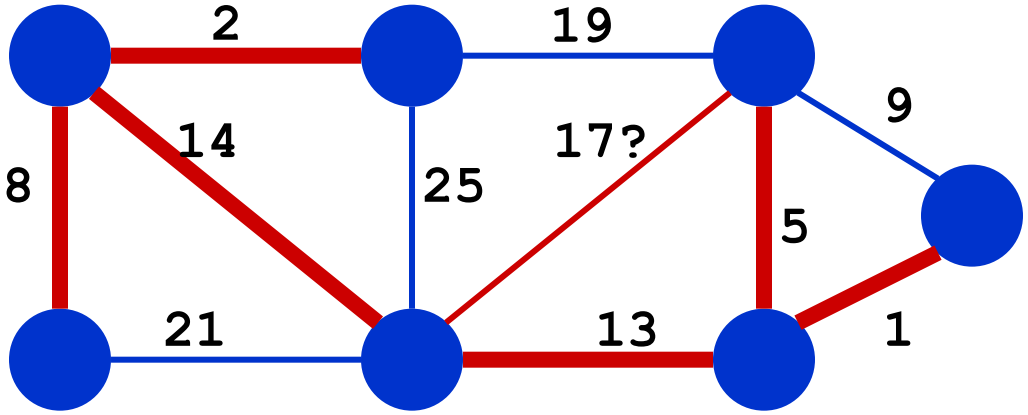
Εκτέλεση αλγόριθμου:



Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

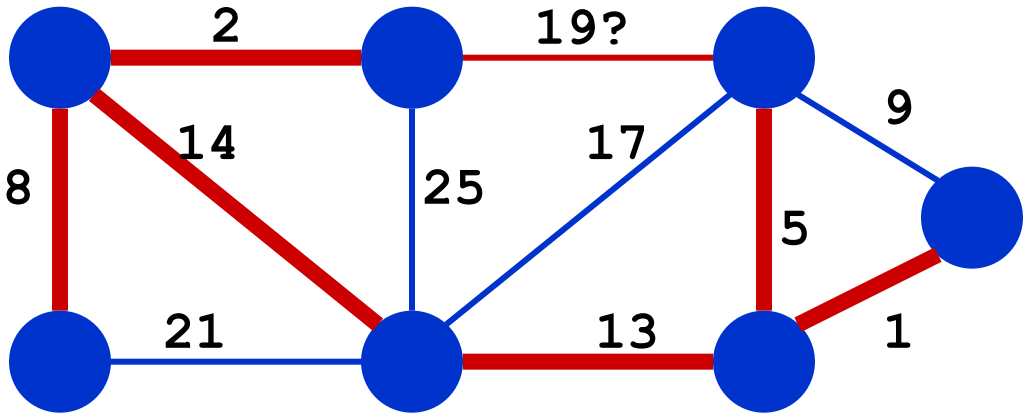
Εκτέλεση αλγόριθμου:



Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

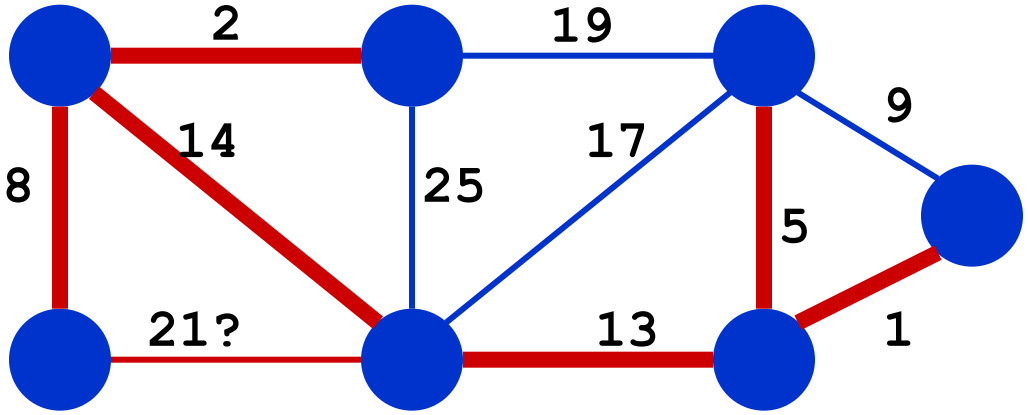
Εκτέλεση αλγόριθμου:



Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

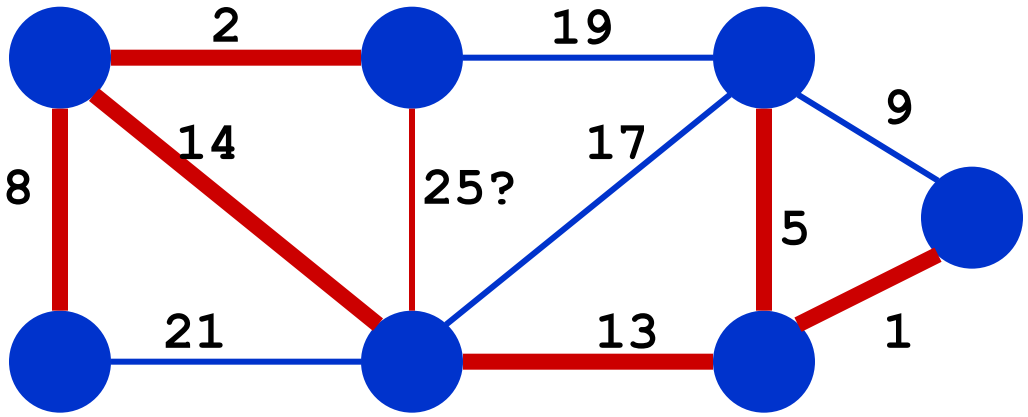
Εκτέλεση αλγόριθμου:



Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

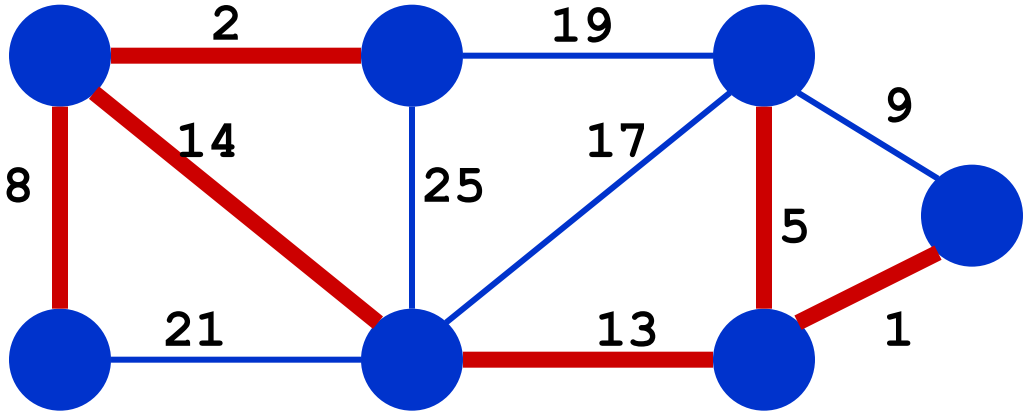
Εκτέλεση αλγόριθμου:



Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

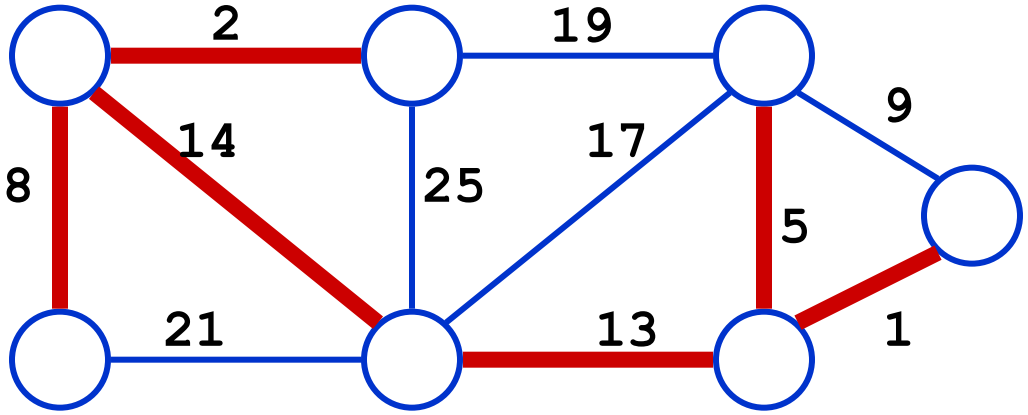
Εκτέλεση αλγόριθμου:



Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

Εκτέλεση αλγόριθμου:



Kruskal's Algorithm: Χρόνος Εκτέλεσης

Τι επηρεάζει το χρόνο εκτέλεσης?

```
Kruskal()
{
1    T = ∅;
2    for each v ∈ V
3       MakeSet(v);
4    sort E by increasing edge weight w
5    for each (u,v) ∈ E (in sorted order)
6       if FindSet(u) ≠ FindSet(v)
7          T = T U {{u,v}};
8          Union(FindSet(u), FindSet(v));
}
```

Kruskal's Algorithm: Χρόνος Εκτέλεσης

Ο χρόνος εκτέλεσης του αλγόριθμου του Kruskal εξαρτάται από την υλοποίηση της δομής δεδομένων ξένων συνόλων. Υποθέτουμε ότι ακολουθείται η υλοποίηση του δάσους ξένων συνόλων.

Ο αρχικός ορισμός του συνόλου T στην γραμμή 1 απαιτεί χρόνο $O(1)$ και η ταξινόμηση των ακμών στην 4 απαιτεί $O(E\log_2 E)$.

Ο βρόχος στις γραμμές 5-8 εκτελεί O(E) πράξεις **FindSet** και **Union** στο δάσος ξένων συνόλων. Μαζί με τις $|V|$ πράξεις **MakeSet** οι εργασίες αυτές απαιτούν συνολικό χρόνο $O(V + E)\ \alpha(V)$ όπου α βραδύτατα αύξουσα συνάρτηση που ορίζεται στην υλοποίηση του δάσους ξένων συνόλων (§21.4 CLRS).

Δεδομένου ότι το G είναι συνεκτικό έχουμε $E \geq |V| - 1$ και επομένως οι πράξεις ξένων συνόλων απαιτούν $O(E\ \alpha(V))$.

Επιπλέον $\alpha(V) = O(\log_2 V)=O(\log_2 E)$ ο συνολικός χρόνος εκτέλεσης του Kruskal είναι $O(E \log_2 E)$. Δεδομένου ότι $E < V^2$. έχουμε $\log_2 |E|=O(\log_2 V)$ και επομένως ο χρόνος εκτέλεσης του αλγορίθμου Kruskal μπορεί να γραφεί $O(E \log_2 V)$

Kruskal's Algorithm: Χρόνος Εκτέλεσης

```
Kruskal()
{
1    T = ∅;
2    for each v ∈ V
3       MakeSet(v);
4    sort E by increasing edge weight w
5    for each (u,v) ∈ E (in sorted order)
6       if FindSet(u) ≠ FindSet(v)
7          T = T U {{u,v}};
8          Union(FindSet(u), FindSet(v));
}
```

Τι επηρεάζει το χρόνο εκτέλεσης?

1 Sort

***O*(*V*) MakeSet() κλήσεις**

***O*(*E*) FindSet() κλήσεις**

***O*(*V*) Union() κλήσεις**

(*πόσες ακριβώς Union()?*)

Kruskal's Algorithm: Χρόνος Εκτέλεσης

- Συνοψίζοντας:
  - Sort edges: $O(E \log_2 E)$
  - $O(V)$ MakeSet()'s
  - $O(E)$ FindSet()'s
  - $O(V)$ Union()'s
- Τελικό συμπέρασμα:
  - Ο καλύτερος αλγόριθμος για disjoint-set union εκτελεί τις παραπάνω 3 πράξεις σε $O(E \cdot \alpha(E,V))$, α σχεδόν σταθερή
  - Τελικώς $O(E \log_2 E) \approx O(E \log_2 V)$

Τέλος Ενότητας