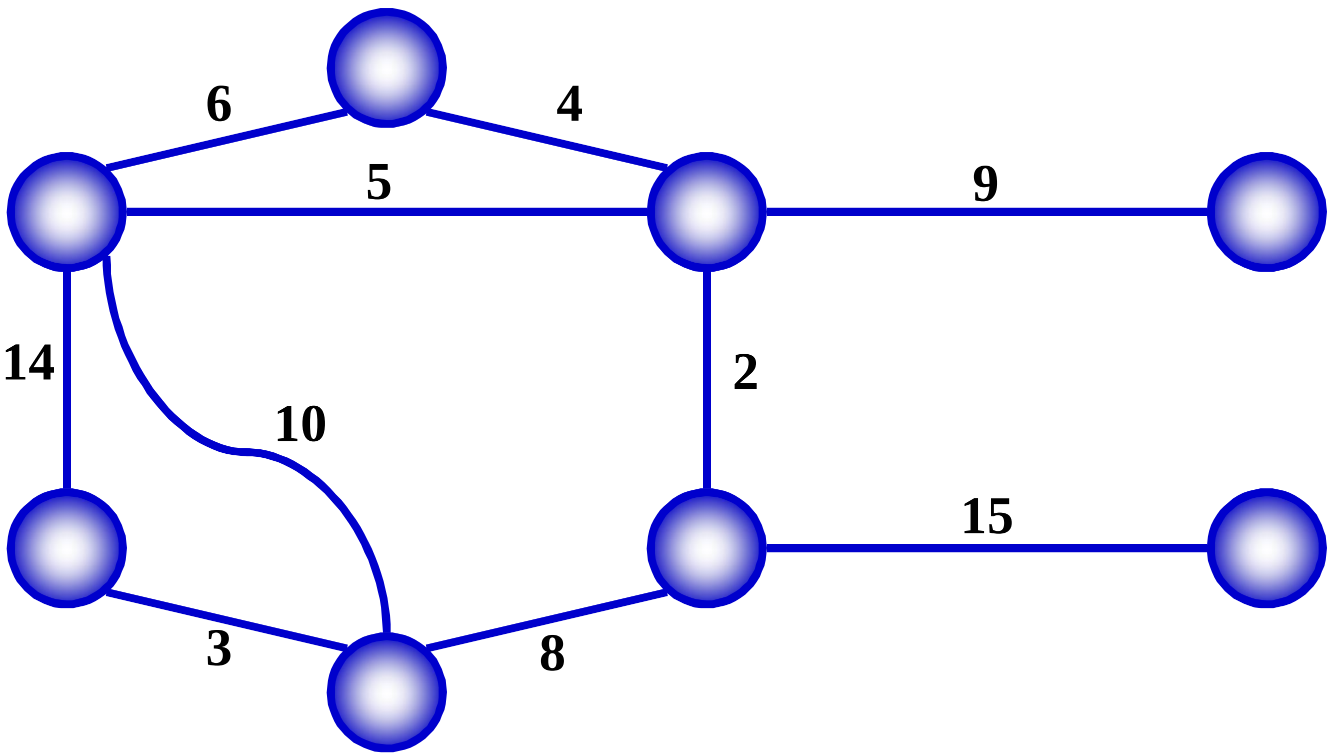
Algorithms:
Greedy Method

Minimum Spanning Tree

Dr. Md. Abul Kashem Mia, Professor, CSE Dept, BUET

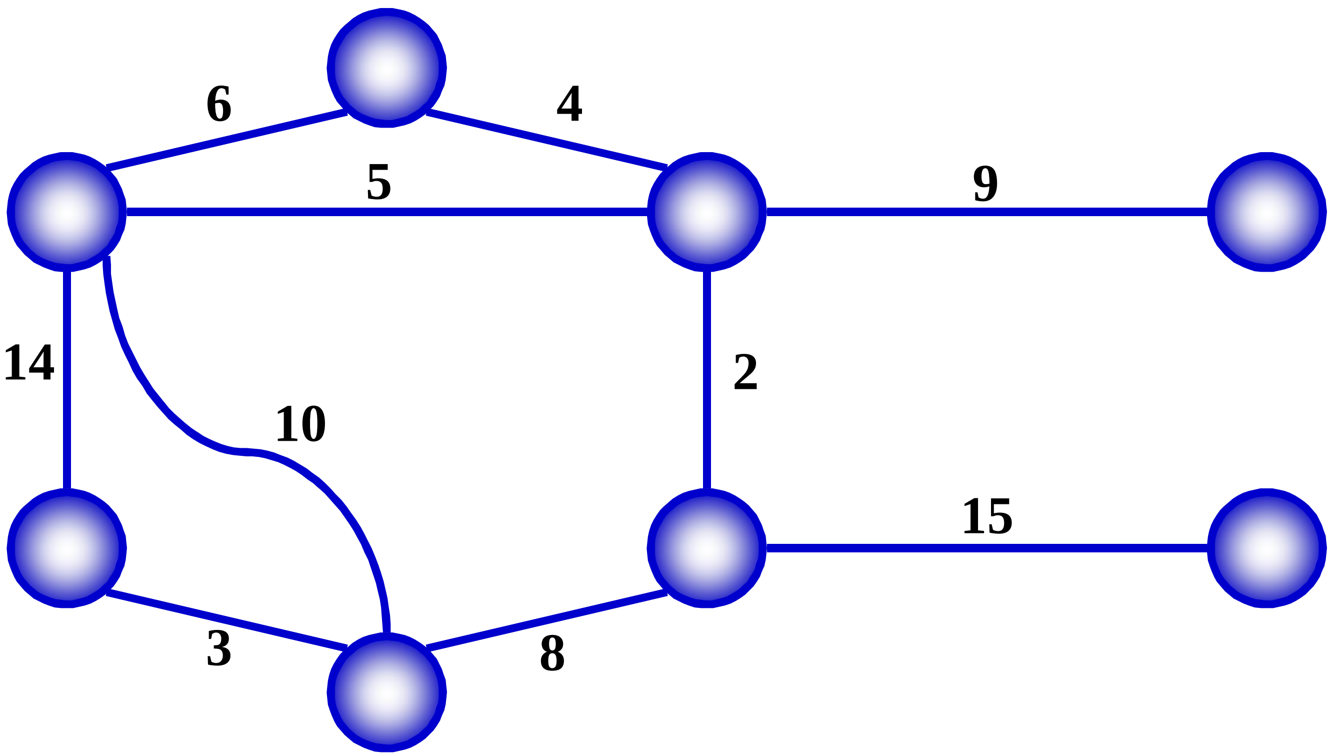
Minimum Spanning Tree

- Problem: given a connected, undirected, weighted graph:



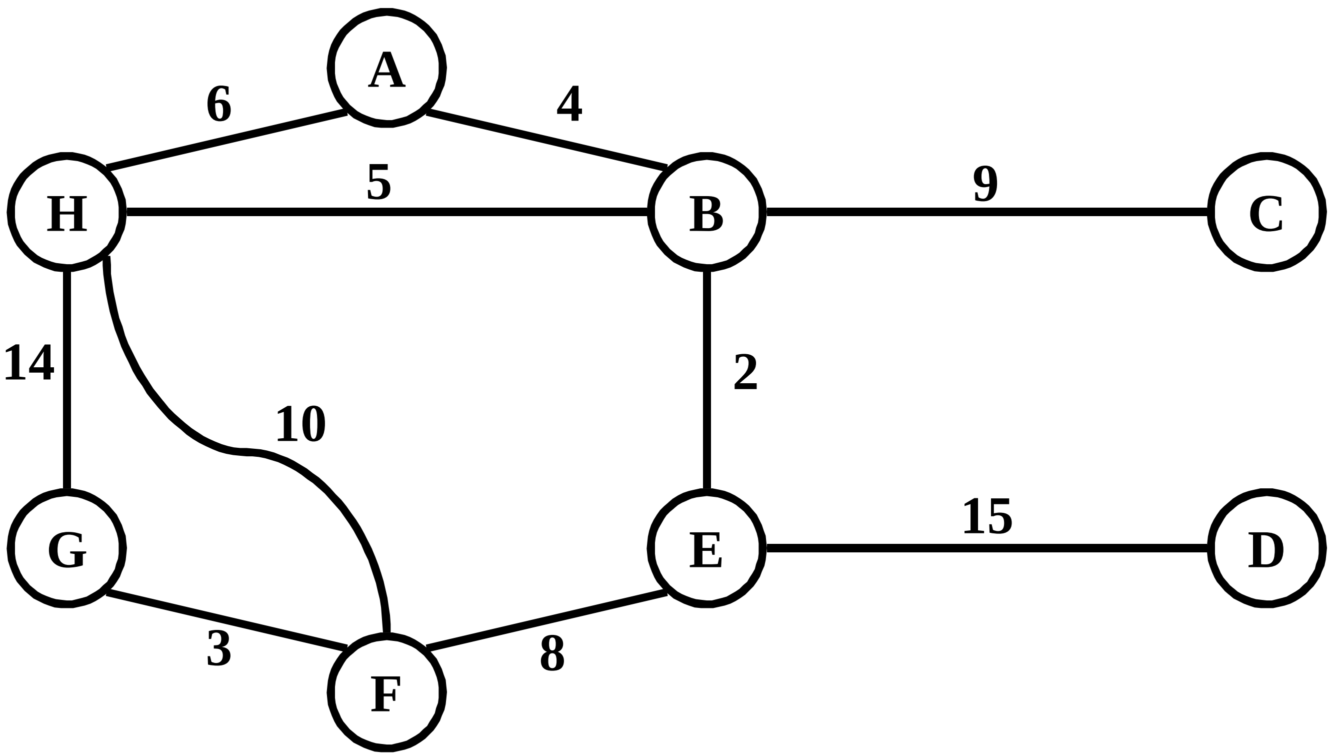
Minimum Spanning Tree

- Problem: given a connected, undirected, weighted graph, find a *spanning tree* using edges that minimize the total weight



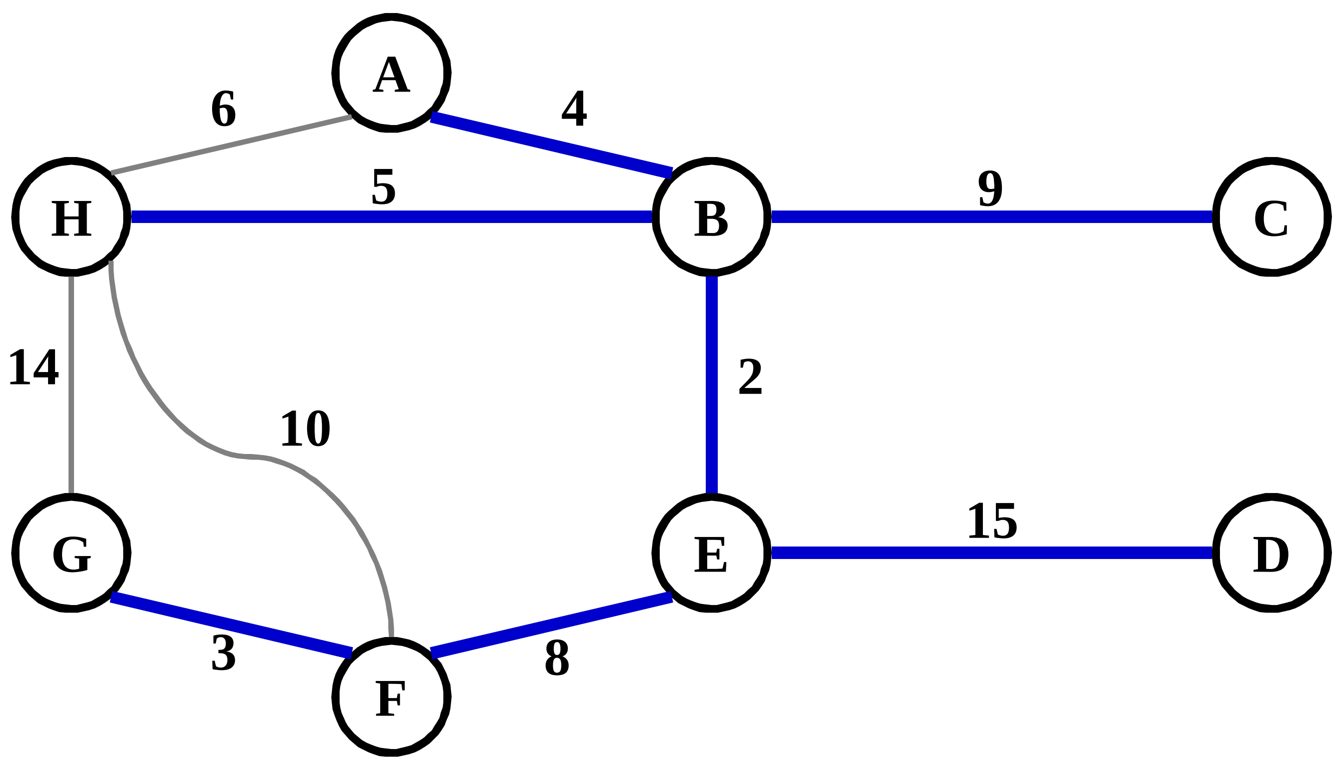
Minimum Spanning Tree

- *Which edges form the minimum spanning tree (MST) of the graph as shown below?*



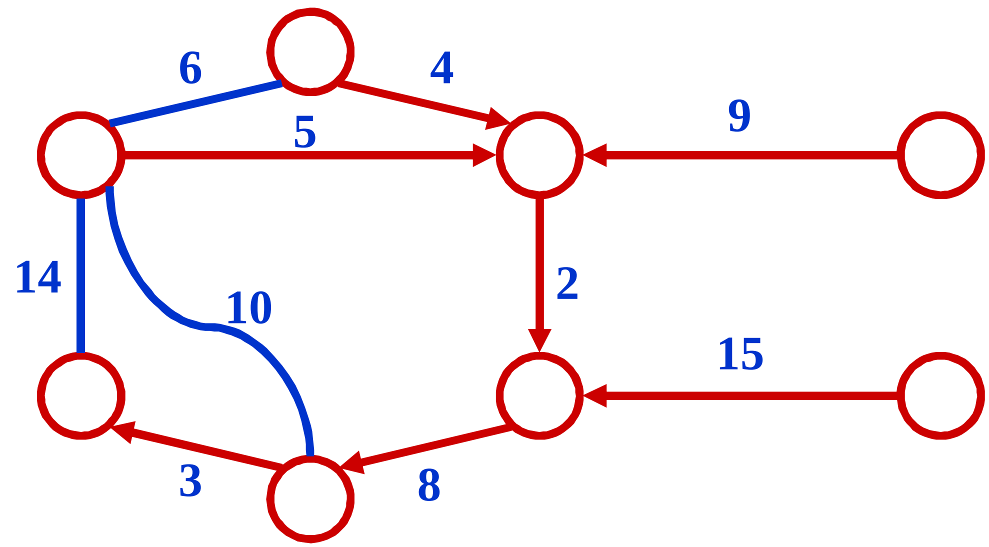
Minimum Spanning Tree

- Answer:



Minimum Spanning Tree

- MSTs satisfy the *optimal substructure* property: an optimal minimum spanning tree is composed of optimal minimum spanning subtrees
  - Let T be an MST of G with an edge (u, v) in the middle
  - Removing (u, v) partitions T into two trees T_1 and T_2
  - Claim: T_1 is an MST of $G_1 = (V_1,E_1)$, and T_2 is an MST of $G_2 = (V_2,E_2)$
    (*Do V_1 and V_2 share vertices? Why?*)
  - Proof: $w(T) = w(u,v) + w(T_1) + w(T_2)$
    (There can't be a better tree than T_1 or T_2. Then T would be suboptimal)

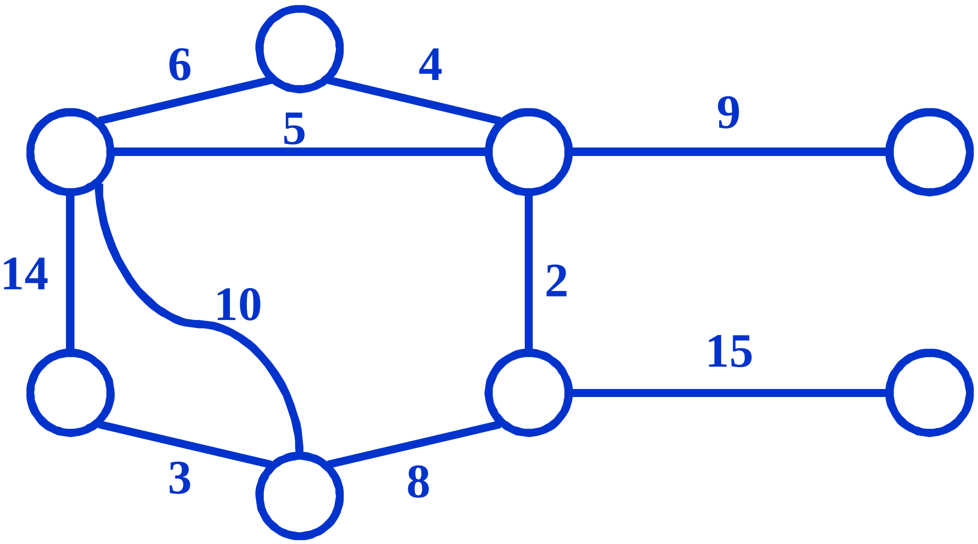


Prim’s Algorithm

```
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

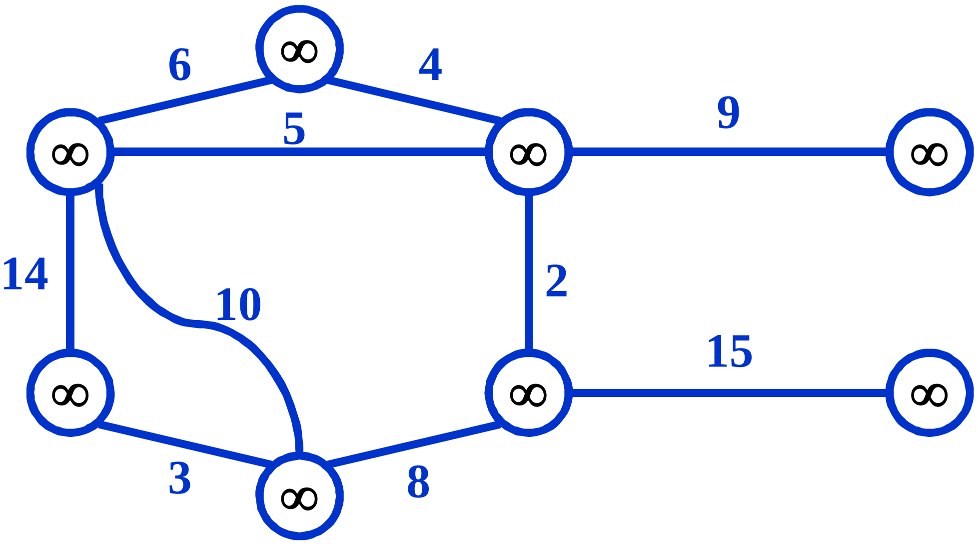
```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```



Run on example graph

Prim’s Algorithm

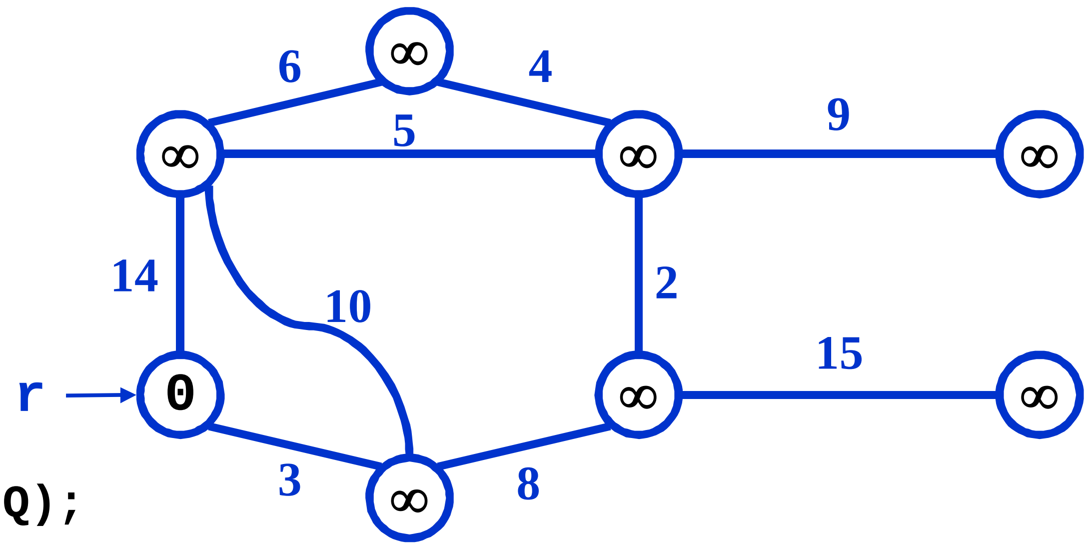
```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```



Run on example graph

Prim's Algorithm

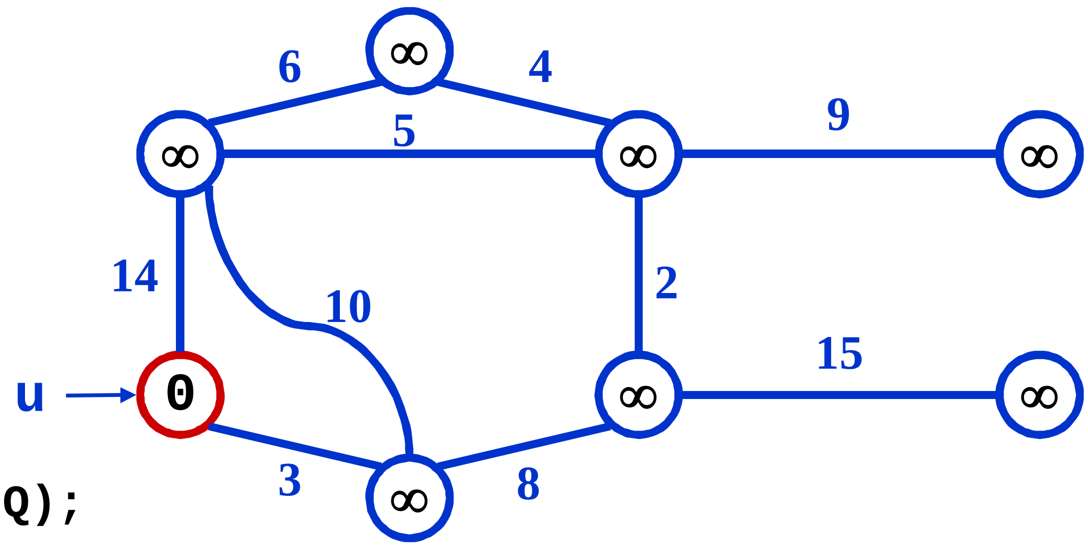
```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```



Pick a start vertex r

Prim's Algorithm

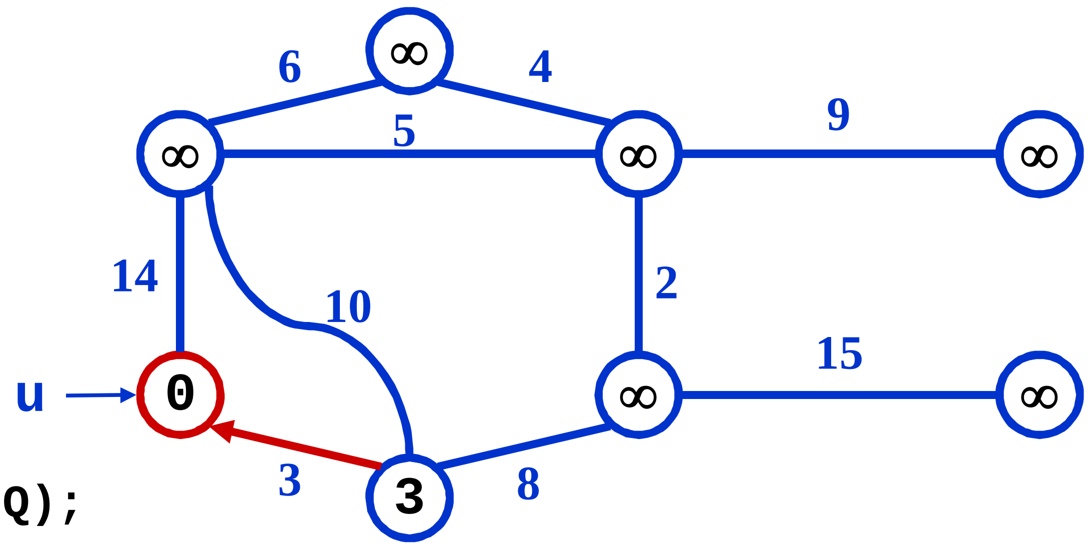
```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```



Red vertices have been removed from Q

Prim’s Algorithm

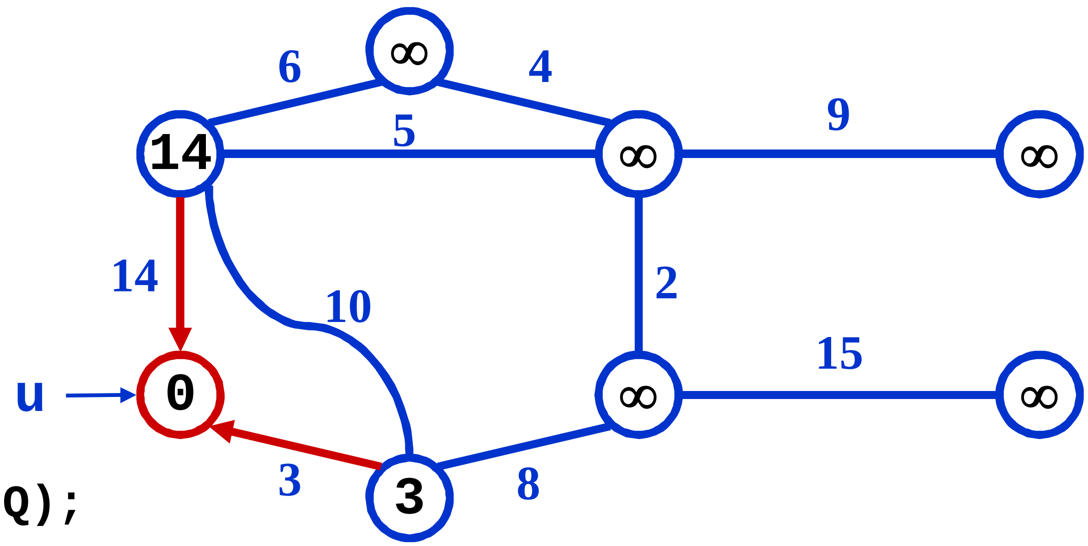
```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```



Red arrows indicate parent pointers

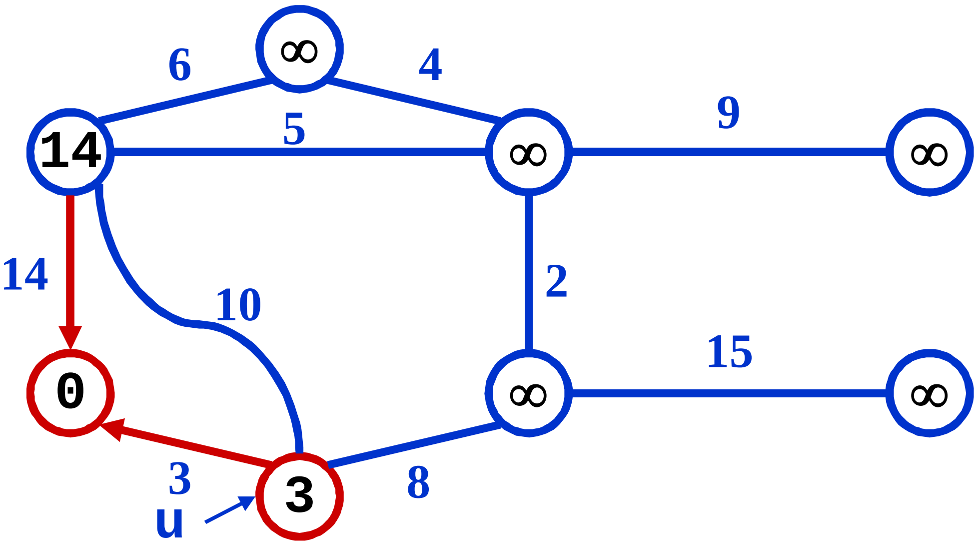
Prim’s Algorithm

```
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);
```



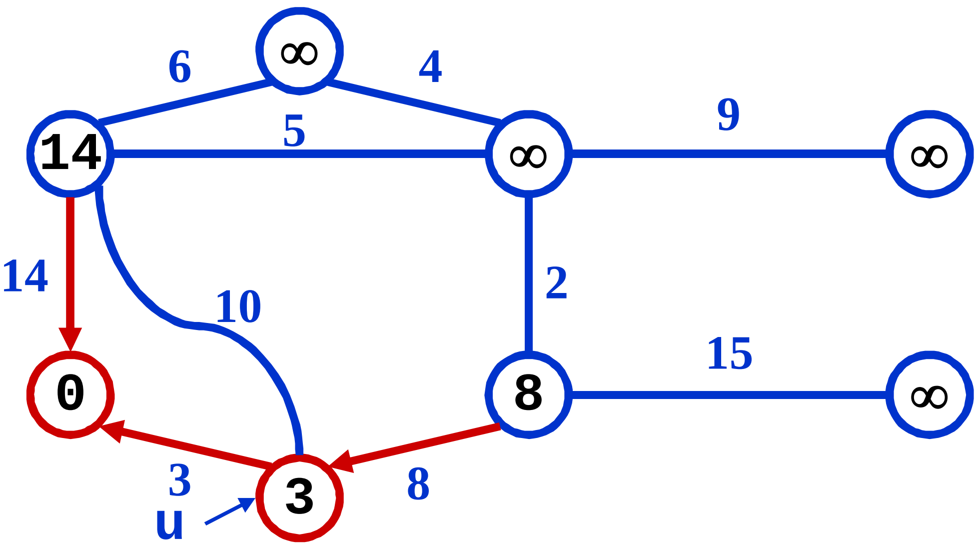
Prim's Algorithm

```
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);
```



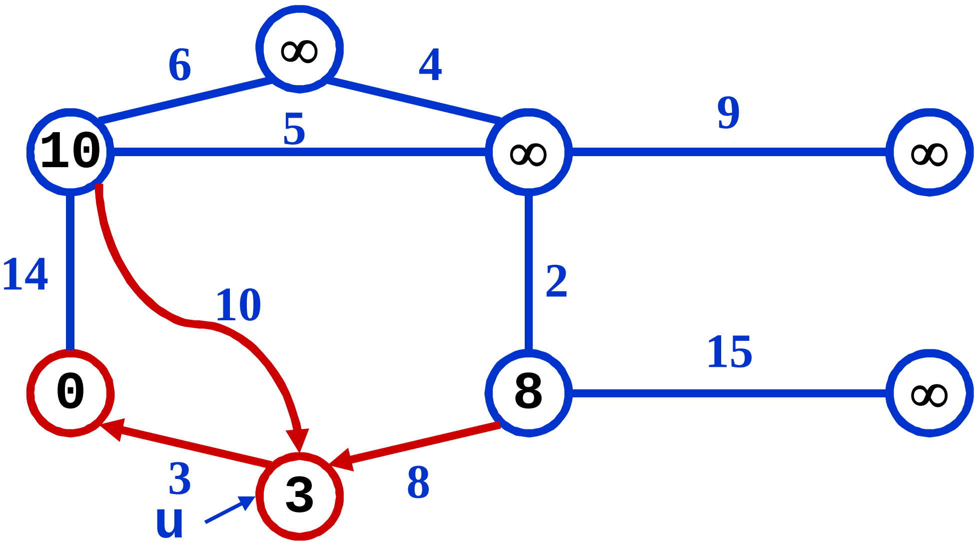
Prim's Algorithm

```
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);
```



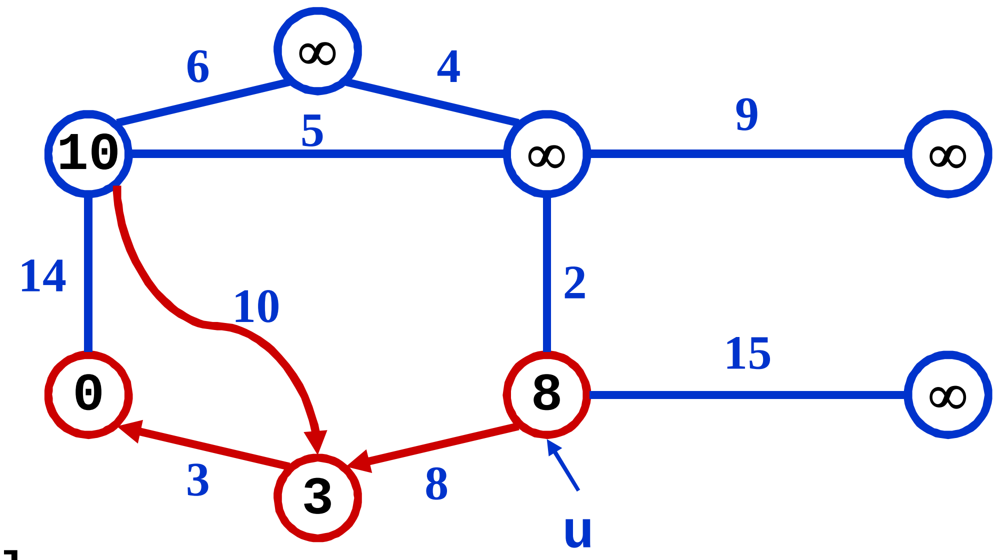
Prim's Algorithm

```
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);
```



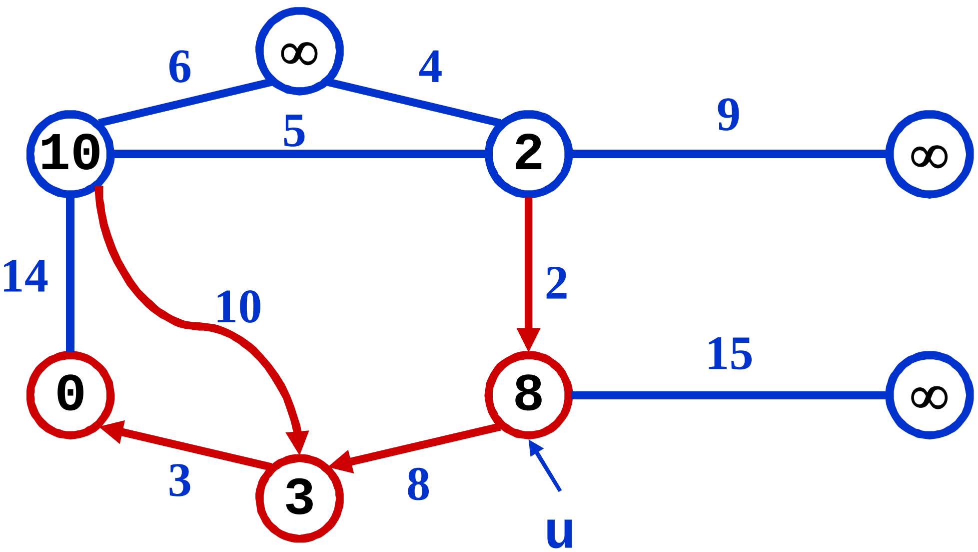
Prim's Algorithm

```
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);
```



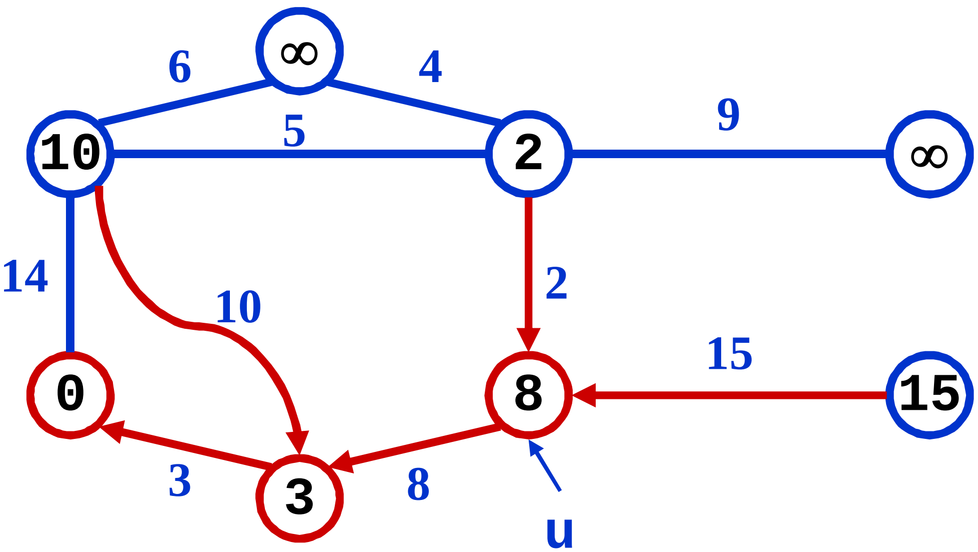
Prim's Algorithm

```
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);
```



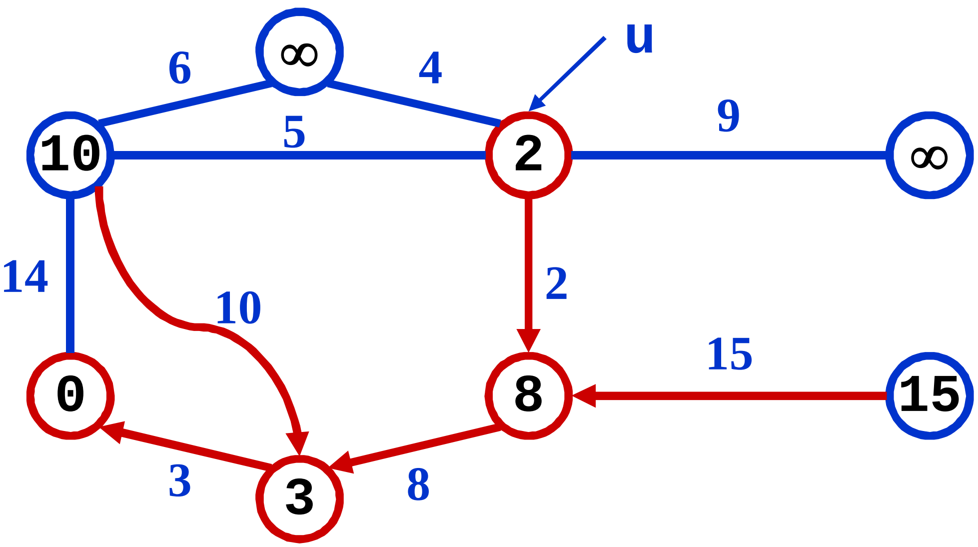
Prim's Algorithm

```
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);
```



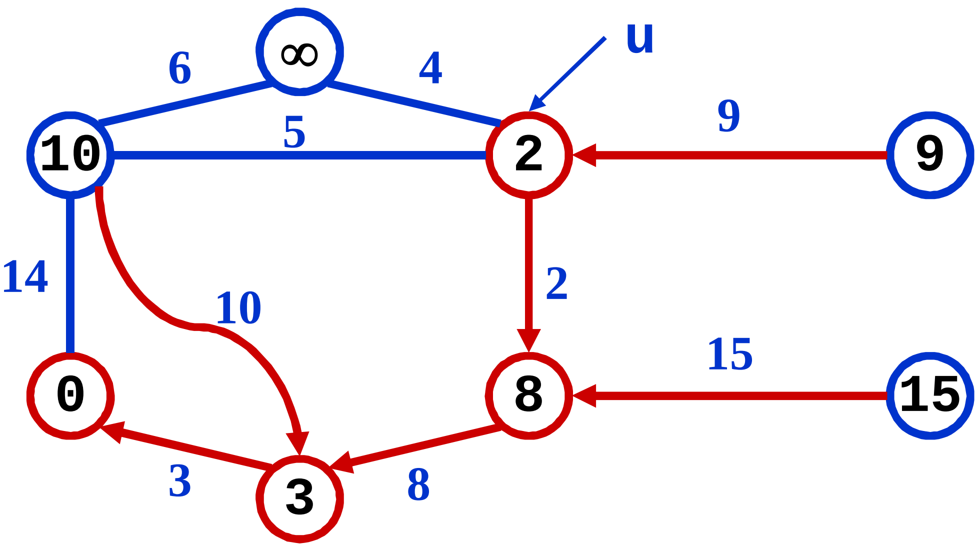
Prim’s Algorithm

```
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);
```



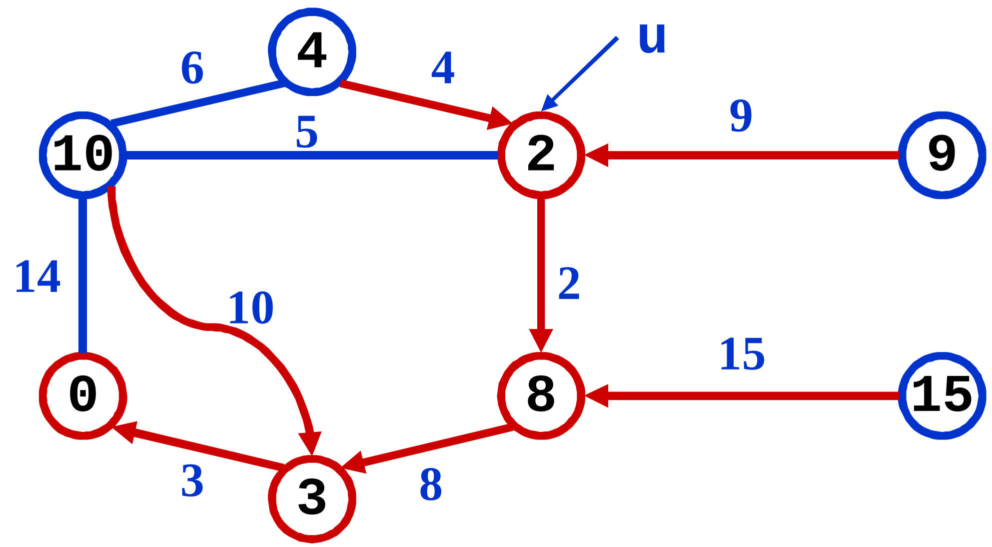
Prim's Algorithm

```
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);
```



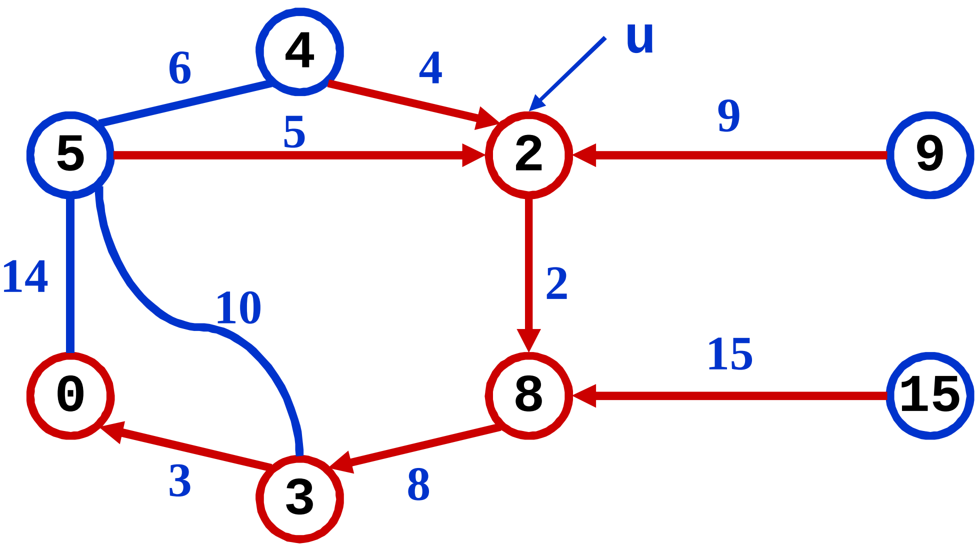
Prim's Algorithm

```
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);
```



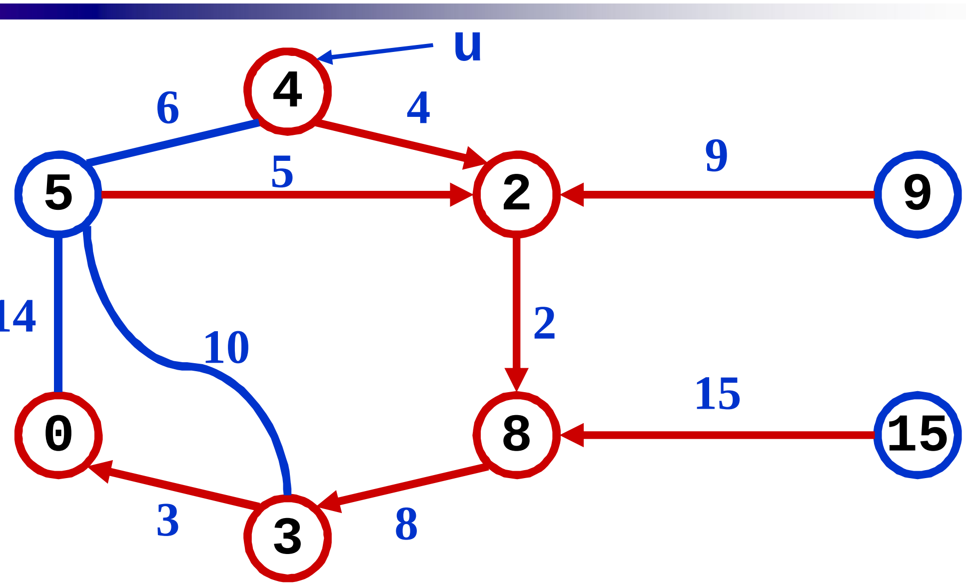
Prim's Algorithm

```
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);
```



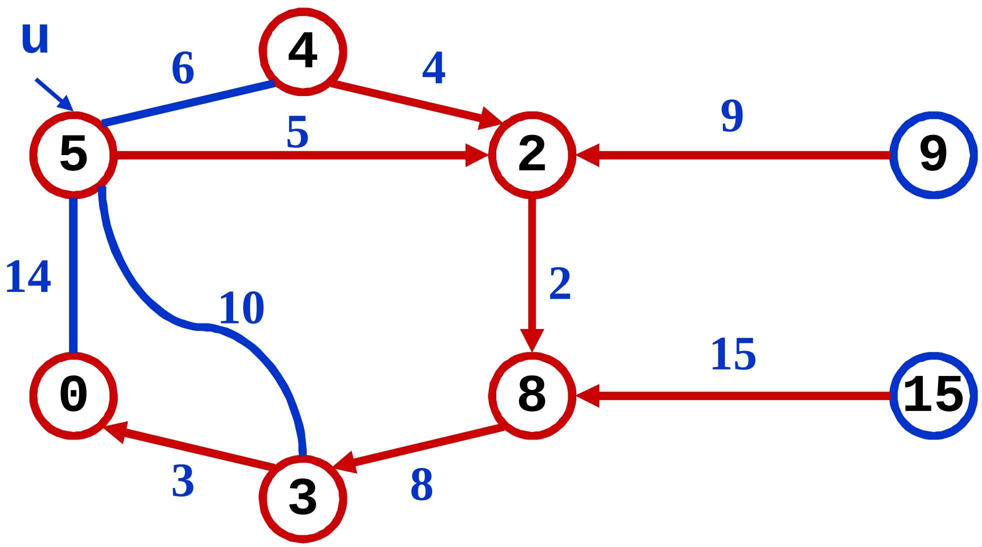
Prim's Algorithm

```
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);
```



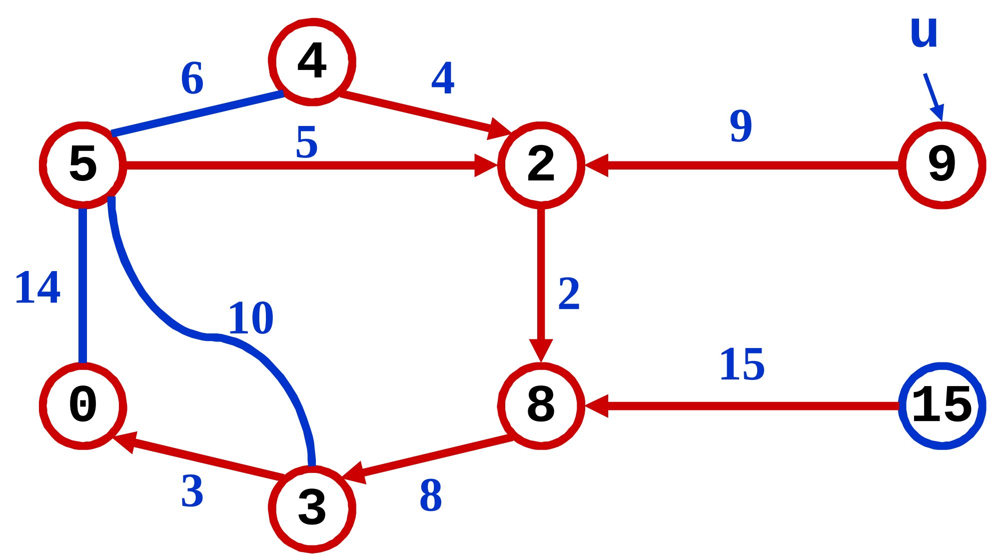
Prim's Algorithm

```
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);
```



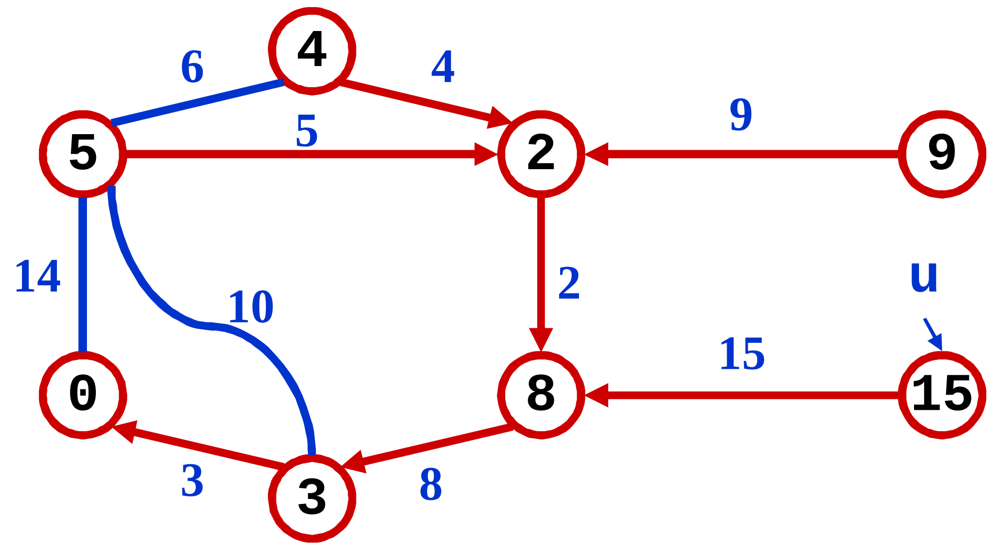
Prim’s Algorithm

```
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);
```



Review: 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);
```

What is the hidden cost in this code?

Review: 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));
```

Review: 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));
```

How often is ExtractMin() called?

How often is DecreaseKey() called?

Review: 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);
```

What will be the running time?

A: Depends on queue

binary heap: O(E lg V)

Fibonacci heap: O(V lg V + E)

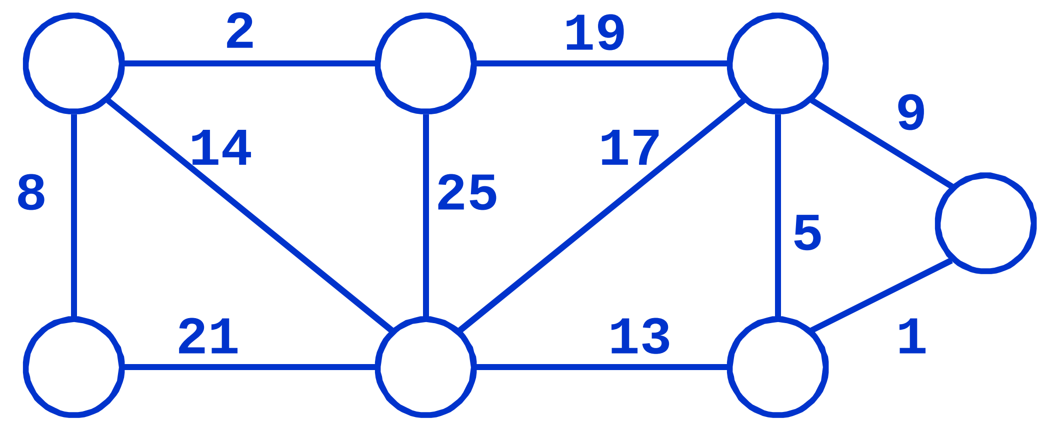
Disjoint-Set Union Problem

- Want a data structure to support disjoint sets
  - Collection of disjoint sets $S = \mathbf{U}_i \{S_i\}, \quad S_i \cap S_j = \varnothing$
- Need to support following operations:
  - MakeSet(x): $S = S \, \mathbf{U} \, \{\{x\}\}$
  - Union(S_i, S_j): $S = S - \{S_i, S_j\} \, \mathbf{U} \, \{S_i \, \mathbf{U} \, S_j\}$
  - FindSet(x): return $S_i \in S$ such that $x \in S_i$
- Before discussing implementation details, we look at example application: MSTs

Kruskal's Algorithm

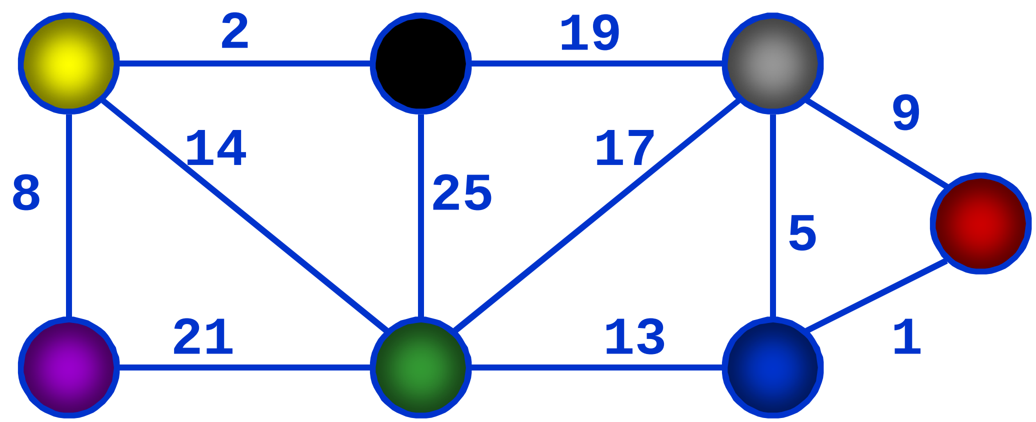
```
Kruskal()
{
   T = ∅;
   for each v ∈ V
      MakeSet(v);
   sort E into nondecreasing order by weight w
   for each (u,v) ∈ E (in sorted order)
      if FindSet(u) ≠ FindSet(v)
         T = T U {{u,v}};
         Union(FindSet(u), FindSet(v));
}
```

Kruskal's Algorithm



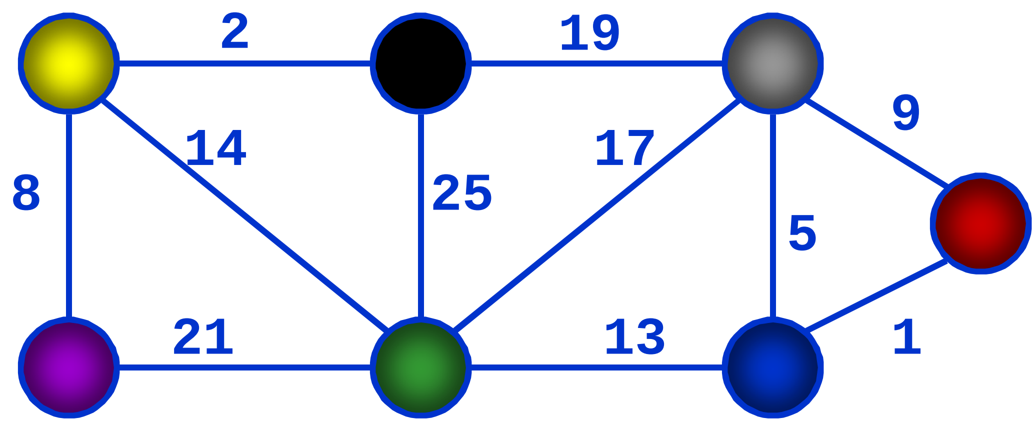

```
Kruskal()
{
   T = ∅;
   for each v ∈ V
      MakeSet(v);
   sort E into nondecreasing order by weight w
   for each (u,v) ∈ E (in sorted order)
      if FindSet(u) ≠ FindSet(v)
         T = T U {{u,v}};
         Union(FindSet(u), FindSet(v));
}
```

Kruskal's Algorithm



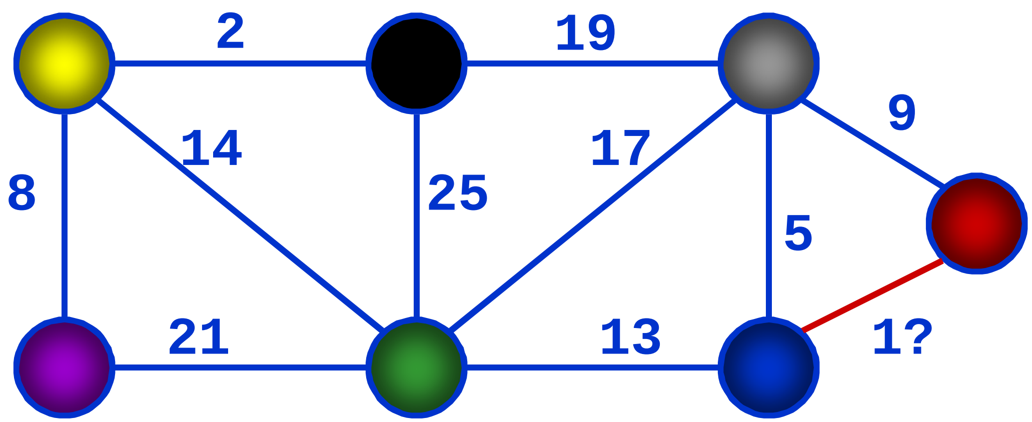

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E into nondecreasing order by weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

Kruskal's Algorithm



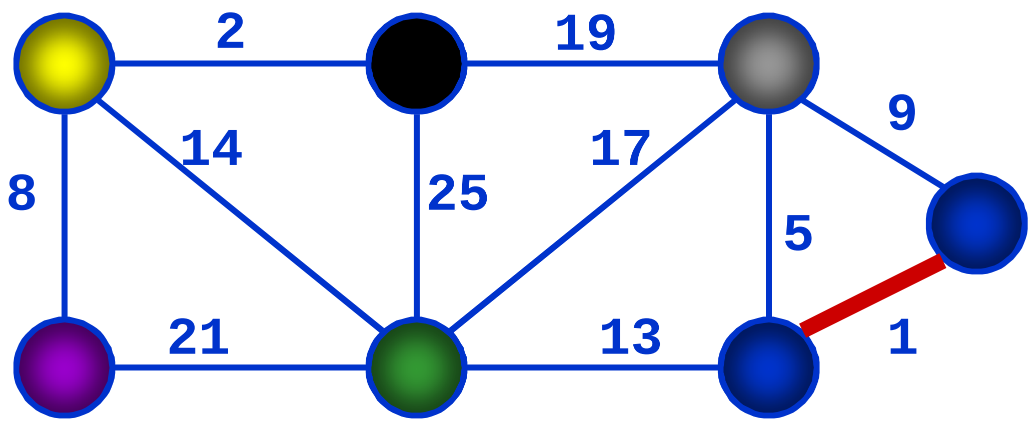

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E into nondecreasing order by weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

Kruskal's Algorithm



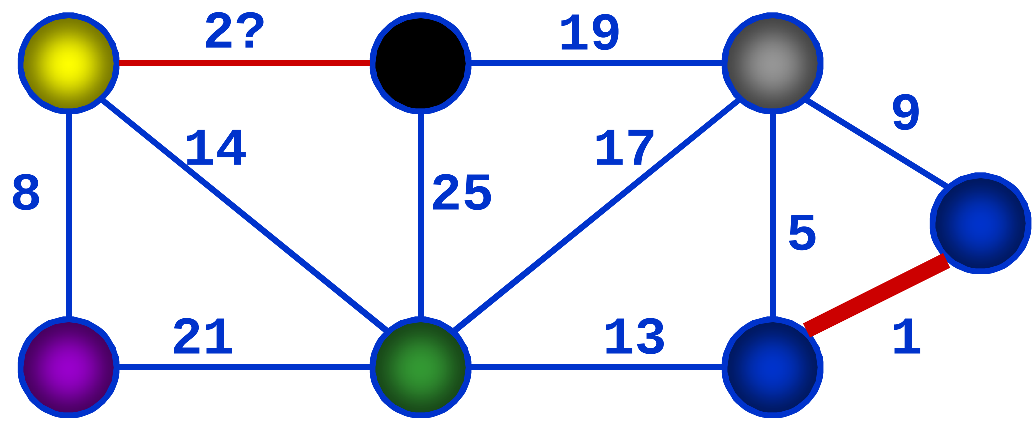

```
Kruskal()
{
   T = ∅;
   for each v ∈ V
      MakeSet(v);
   sort E into nondecreasing order by weight w
   for each (u,v) ∈ E (in sorted order)
      if FindSet(u) ≠ FindSet(v)
         T = T U {{u,v}};
         Union(FindSet(u), FindSet(v));
}
```

Kruskal's Algorithm



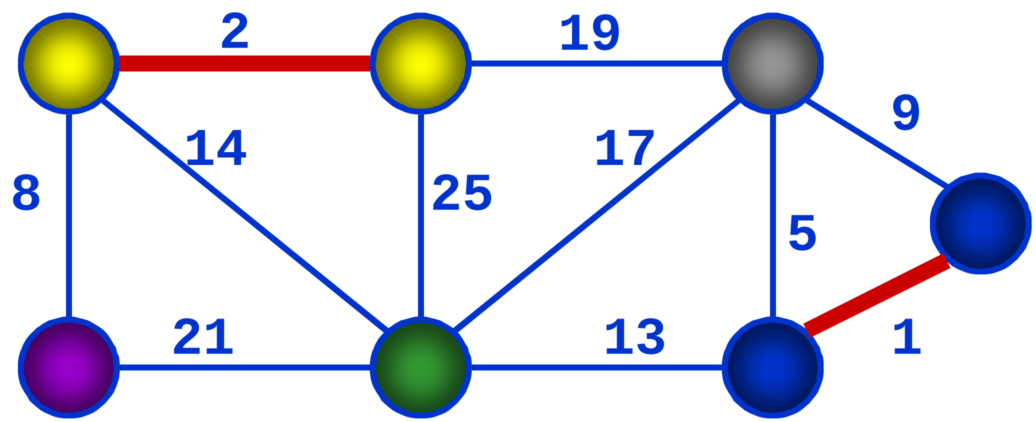

```
Kruskal()
{
   T = ∅;
   for each v ∈ V
      MakeSet(v);
   sort E into nondecreasing order by weight w
   for each (u,v) ∈ E (in sorted order)
      if FindSet(u) ≠ FindSet(v)
         T = T U {{u,v}};
         Union(FindSet(u), FindSet(v));
}
```

Kruskal's Algorithm




```
Kruskal()
{
   T = ∅;
   for each v ∈ V
      MakeSet(v);
   sort E into nondecreasing order by 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));
}
```

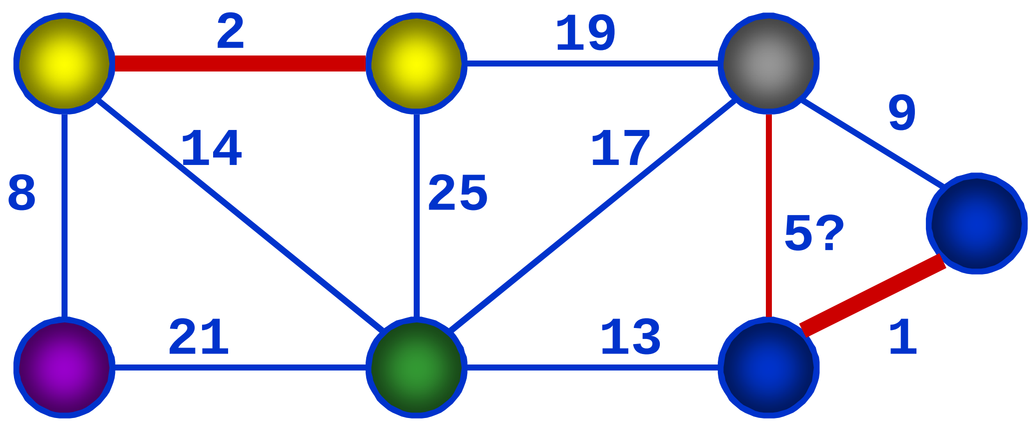
Kruskal's Algorithm



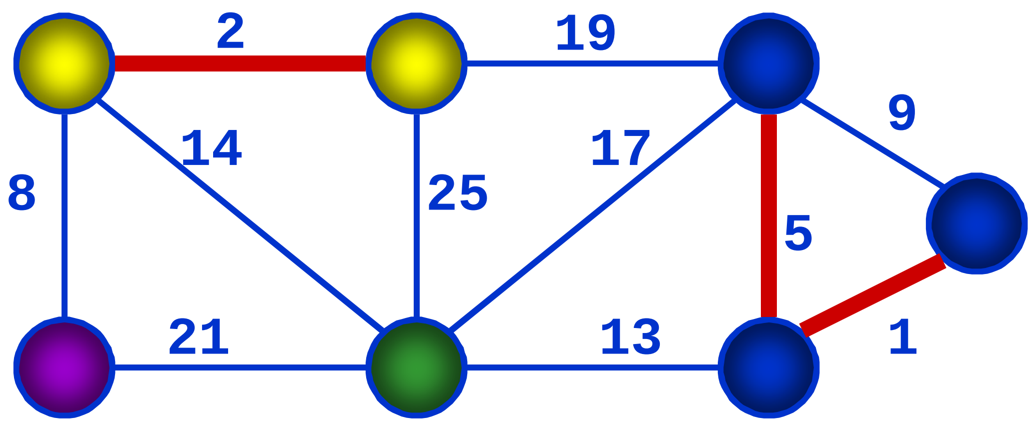

```
Kruskal()
{
   T = ∅;
   for each v ∈ V
      MakeSet(v);
   sort E into nondecreasing order by weight w
   for each (u,v) ∈ E (in sorted order)
      if FindSet(u) ≠ FindSet(v)
         T = T U {{u,v}};
         Union(FindSet(u), FindSet(v));
}
```

Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E into nondecreasing order by 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));
}
```

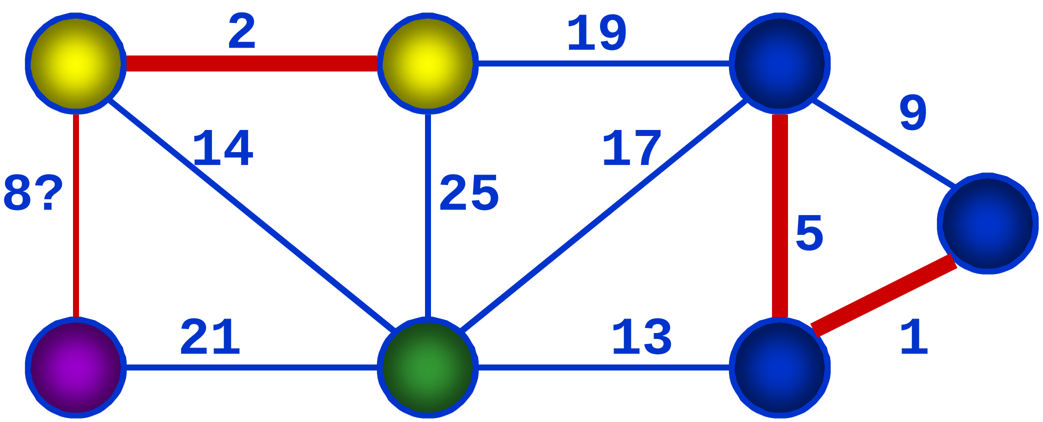


Kruskal’s Algorithm



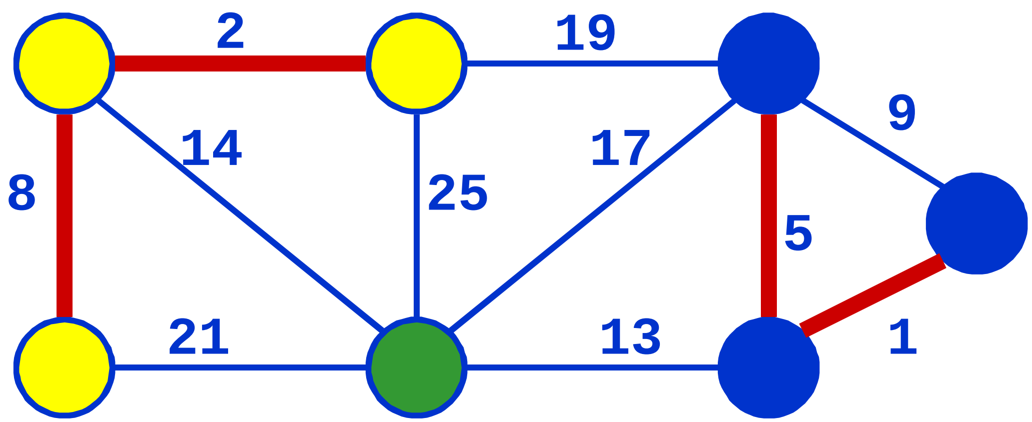

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E into nondecreasing order by weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

Kruskal’s Algorithm



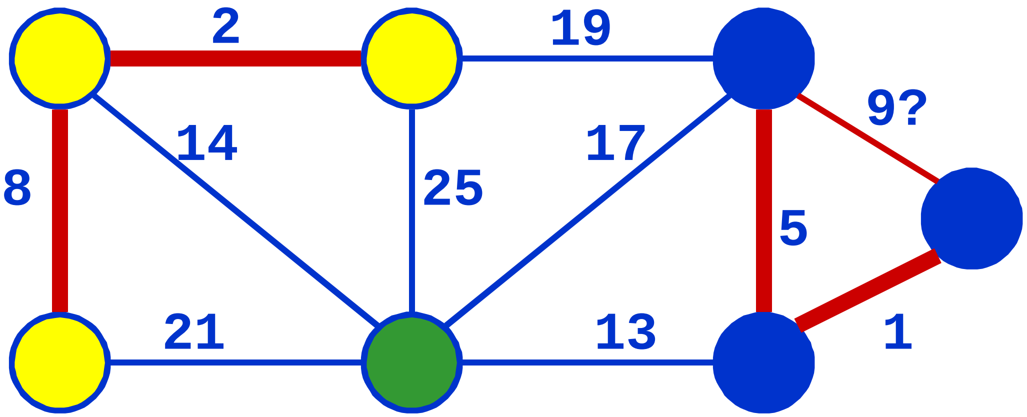

```
Kruskal()
{
   T = ∅;
   for each v ∈ V
      MakeSet(v);
   sort E into nondecreasing order by weight w
   for each (u,v) ∈ E (in sorted order)
      if FindSet(u) ≠ FindSet(v)
         T = T U {{u,v}};
         Union(FindSet(u), FindSet(v));
}
```

Kruskal's Algorithm



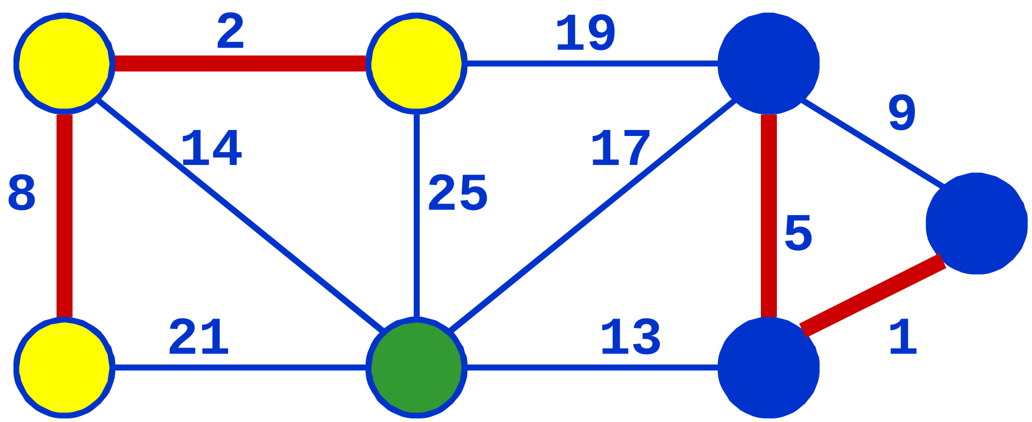

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E into nondecreasing order by weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

Kruskal's Algorithm



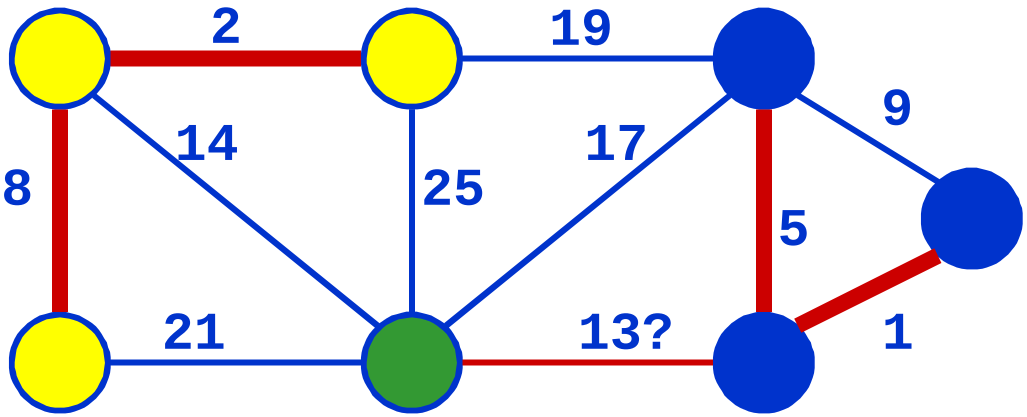

```
Kruskal()
{
   T = ∅;
   for each v ∈ V
      MakeSet(v);
   sort E into nondecreasing order by weight w
   for each (u,v) ∈ E (in sorted order)
      if FindSet(u) ≠ FindSet(v)
         T = T U {{u,v}};
         Union(FindSet(u), FindSet(v));
}
```

Kruskal's Algorithm



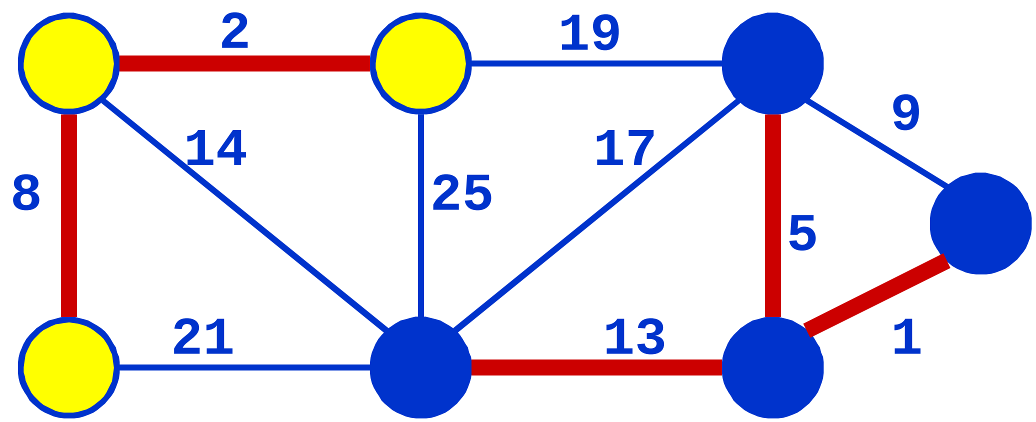

```
Kruskal()
{
   T = ∅;
   for each v ∈ V
      MakeSet(v);
   sort E into nondecreasing order by weight w
   for each (u,v) ∈ E (in sorted order)
      if FindSet(u) ≠ FindSet(v)
         T = T U {{u,v}};
         Union(FindSet(u), FindSet(v));
}
```

Kruskal’s Algorithm



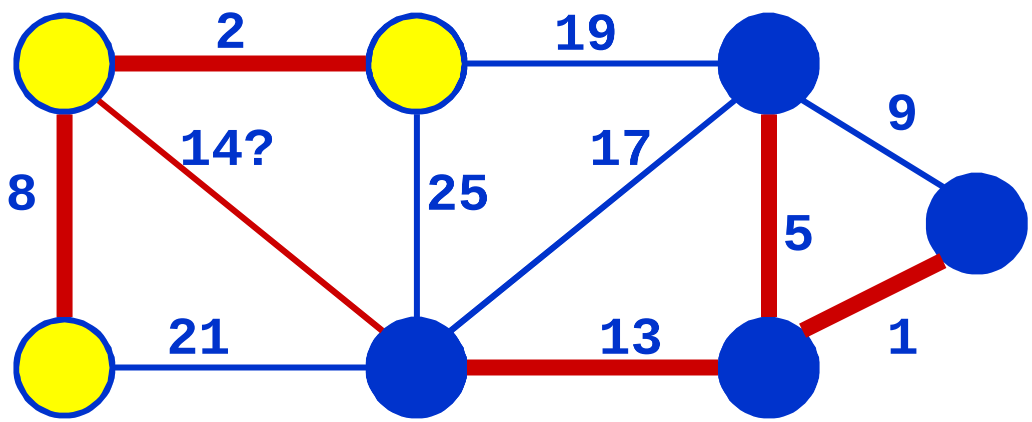

```
Kruskal()
{
   T = ∅;
   for each v ∈ V
      MakeSet(v);
   sort E into nondecreasing order by weight w
   for each (u,v) ∈ E (in sorted order)
      if FindSet(u) ≠ FindSet(v)
         T = T U {{u,v}};
         Union(FindSet(u), FindSet(v));
}
```

Kruskal's Algorithm



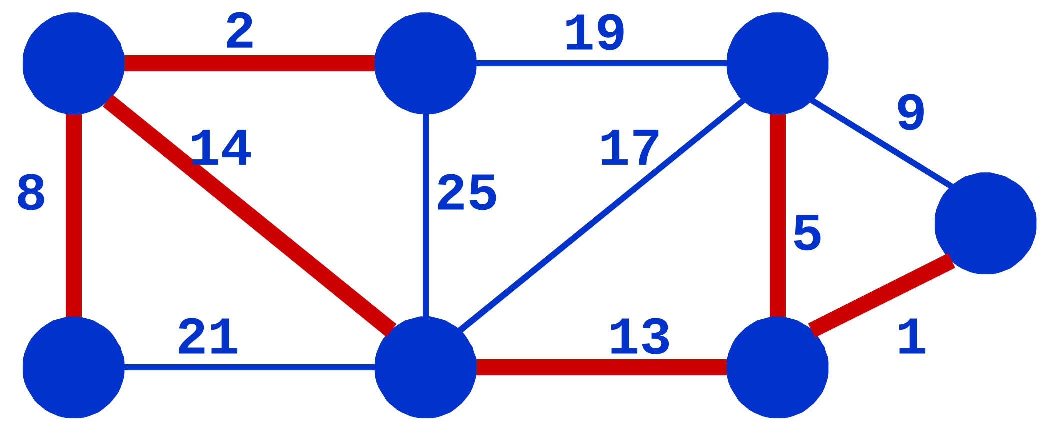

```
Kruskal()
{
   T = ∅;
   for each v ∈ V
      MakeSet(v);
   sort E into nondecreasing order by weight w
   for each (u,v) ∈ E (in sorted order)
      if FindSet(u) ≠ FindSet(v)
         T = T U {{u,v}};
         Union(FindSet(u), FindSet(v));
}
```

Kruskal's Algorithm



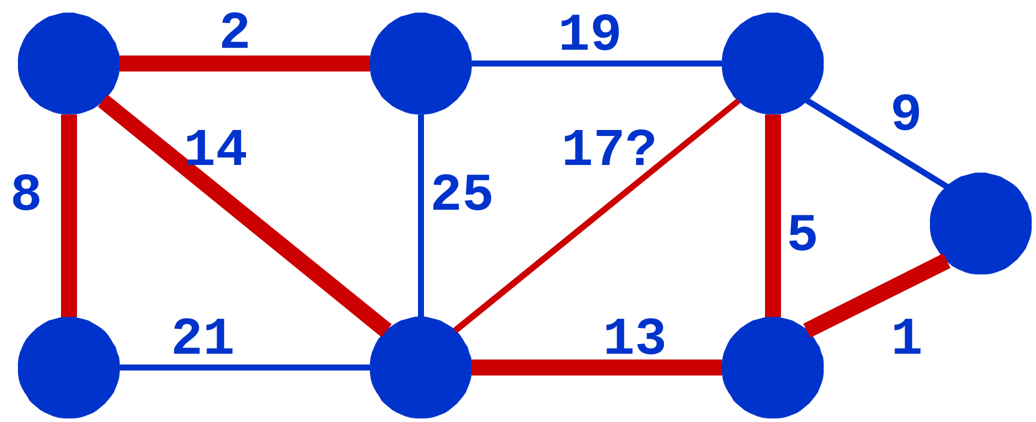

```
Kruskal()
{
   T = ∅;
   for each v ∈ V
      MakeSet(v);
   sort E into nondecreasing order by weight w
   for each (u,v) ∈ E (in sorted order)
      if FindSet(u) ≠ FindSet(v)
         T = T U {{u,v}};
         Union(FindSet(u), FindSet(v));
}
```

Kruskal's Algorithm



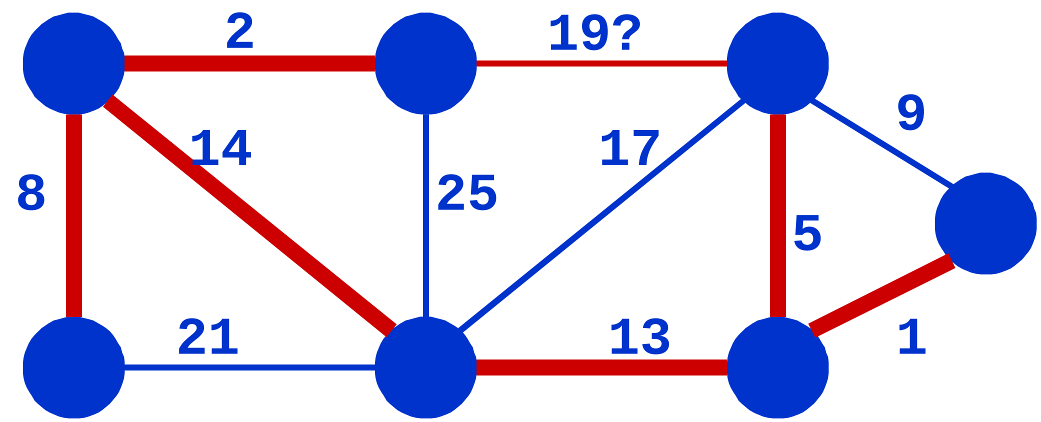

```
Kruskal()
{
   T = ∅;
   for each v ∈ V
      MakeSet(v);
   sort E into nondecreasing order by weight w
   for each (u,v) ∈ E (in sorted order)
      if FindSet(u) ≠ FindSet(v)
         T = T U {{u,v}};
         Union(FindSet(u), FindSet(v));
}
```

Kruskal's Algorithm



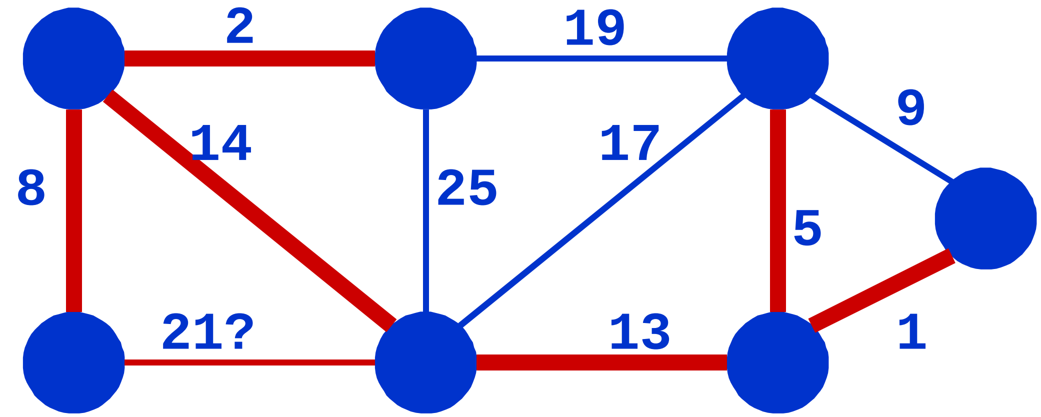

```
Kruskal()
{
   T = ∅;
   for each v ∈ V
      MakeSet(v);
   sort E into nondecreasing order by weight w
   for each (u,v) ∈ E (in sorted order)
      if FindSet(u) ≠ FindSet(v)
         T = T U {{u,v}};
         Union(FindSet(u), FindSet(v));
}
```

Kruskal's Algorithm



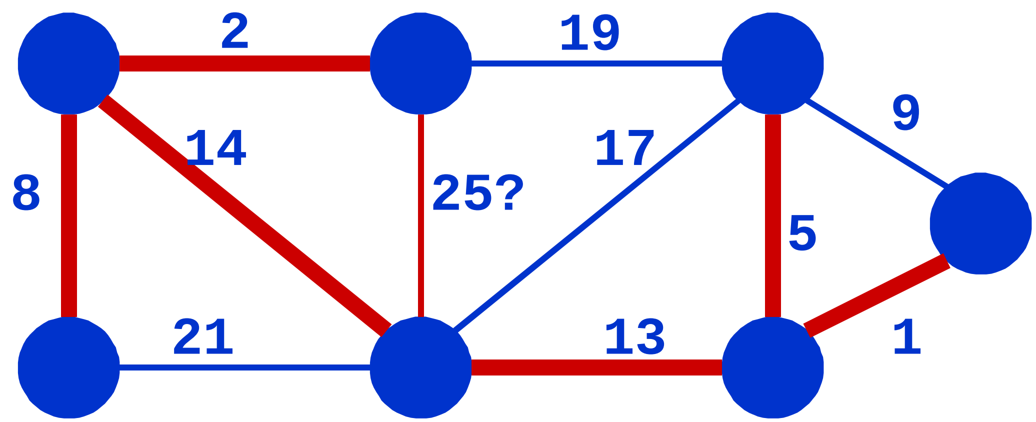

```
Kruskal()
{
   T = ∅;
   for each v ∈ V
      MakeSet(v);
   sort E into nondecreasing order by weight w
   for each (u,v) ∈ E (in sorted order)
      if FindSet(u) ≠ FindSet(v)
         T = T U {{u,v}};
         Union(FindSet(u), FindSet(v));
}
```

Kruskal's Algorithm



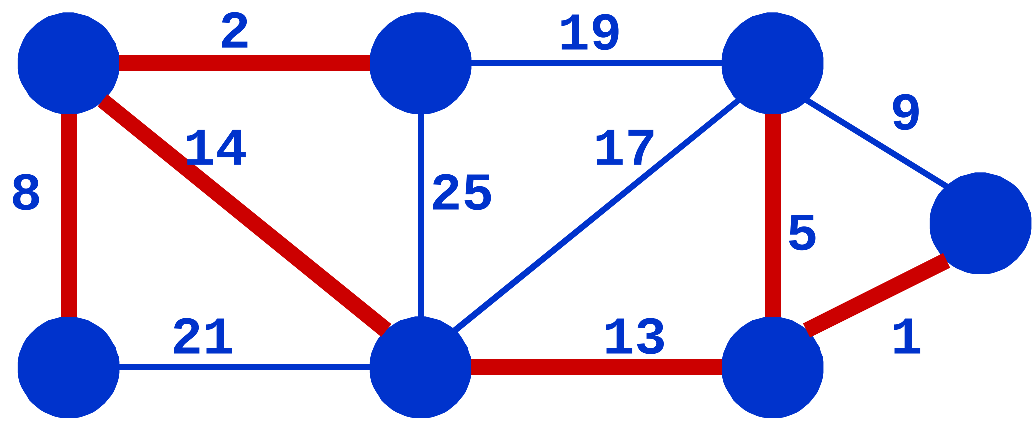

```
Kruskal()
{
   T = ∅;
   for each v ∈ V
      MakeSet(v);
   sort E into nondecreasing order by weight w
   for each (u,v) ∈ E (in sorted order)
      if FindSet(u) ≠ FindSet(v)
         T = T U {{u,v}};
         Union(FindSet(u), FindSet(v));
}
```

Kruskal's Algorithm



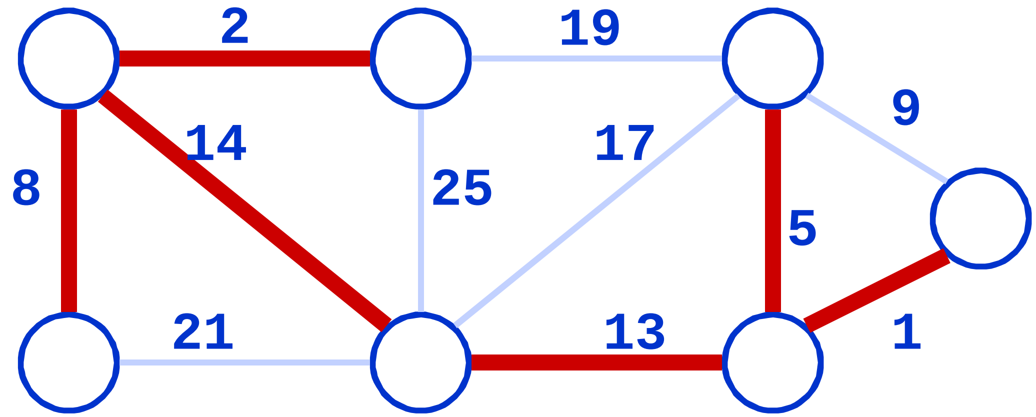

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E into nondecreasing order by weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

Kruskal's Algorithm




```
Kruskal()
{
   T = ∅;
   for each v ∈ V
      MakeSet(v);
   sort E into nondecreasing order by 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));
}
```

Kruskal's Algorithm




```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E into nondecreasing order by 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));
}
```

Kruskal's Algorithm: Running Time

What will affect the running time?

```
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));
}
```

Kruskal's Algorithm: Running Time

```
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));
}
```

What will affect the running time?

1 Sort

O(V) MakeSet() calls

O(E) FindSet() calls

O(V) Union() calls

(*Exactly how many Union()s?*)

Kruskal's Algorithm: Running Time

- To summarize:
  - Sort edges: O(E lg E)
  - O(V) MakeSet()'s
  - O(E) FindSet()'s
  - O(V) Union()'s
- Upshot:
  - Best disjoint-set operation algorithm makes above three operations to take O(E lg E) time.
  - Thus overall time is O(E lg E) = O(E lg V), since $|E| < |V|^2$