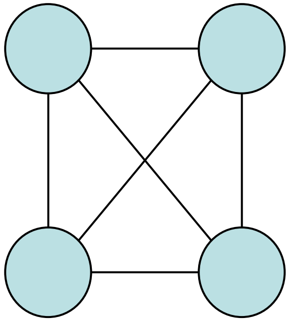
Data Structures

21. Minimum Spanning Tree (MST)

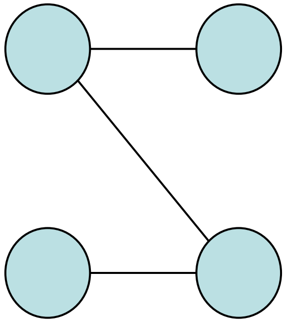
Spanning Trees

- A spanning tree of a graph is just a subgraph that contains all the vertices and is a tree

- Formal definition
  - Given a connected graph with $|V| = n$ vertices
  - A spanning tree is defined a collection of $n - 1$ edges which connect all n vertices
  - The n vertices and $n - 1$ edges define a connected sub-graph



Graph



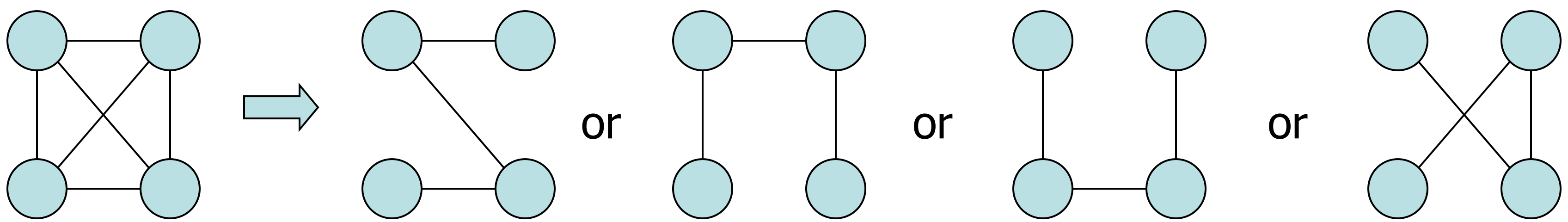
Spanning Tree

Spanning Trees

- A spanning tree is not necessarily unique

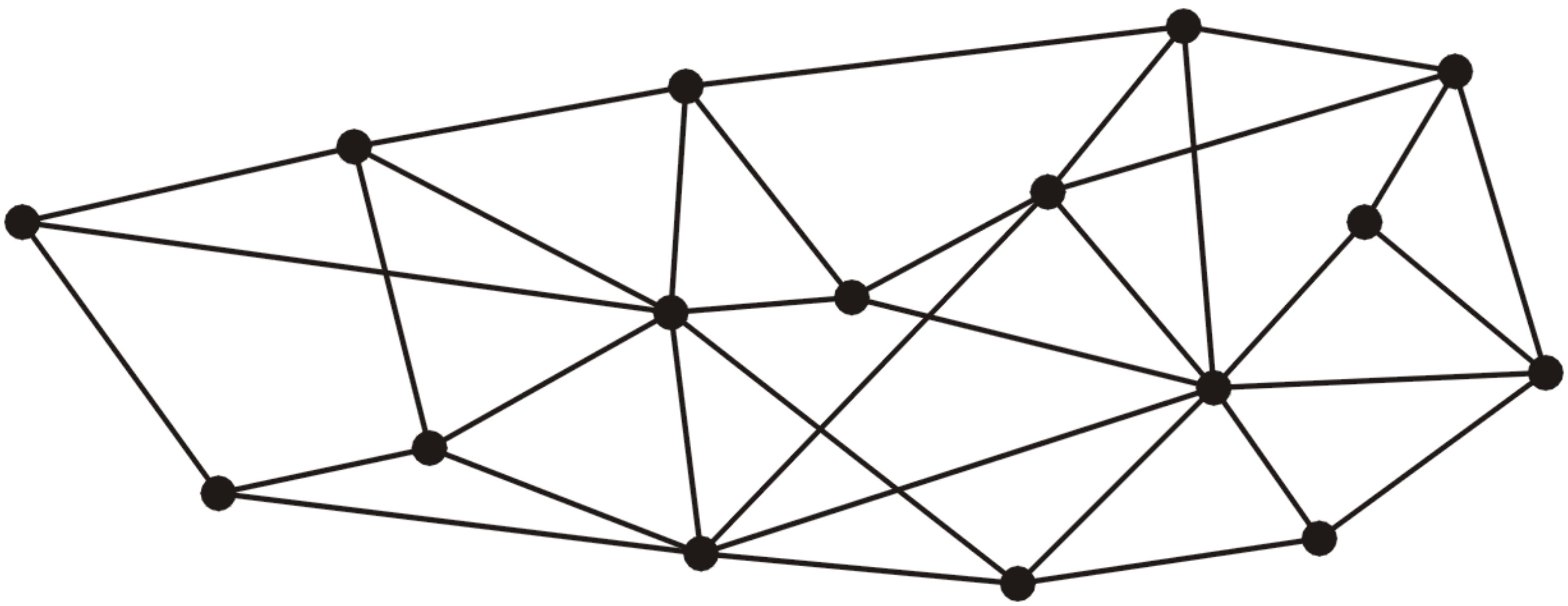
Graph A

Some Spanning Trees from Graph A



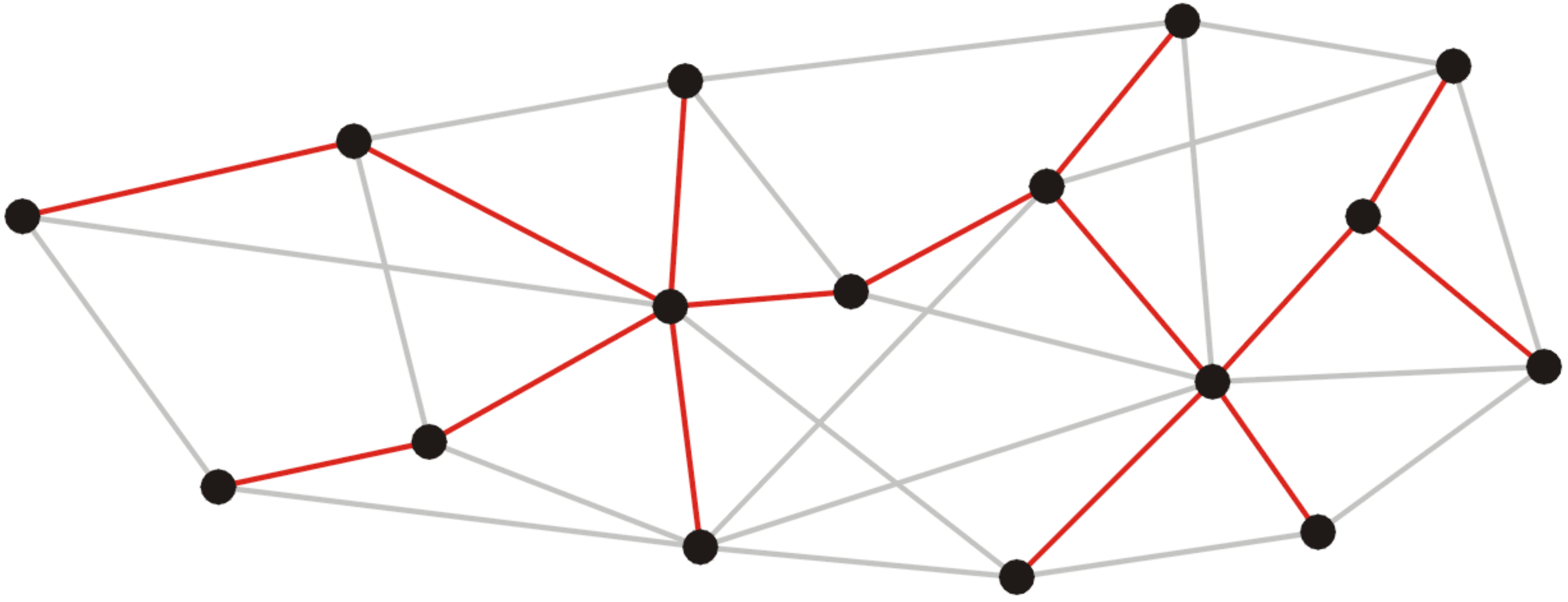
Spanning Trees – Example

- This graph has 16 vertices and 35 edges



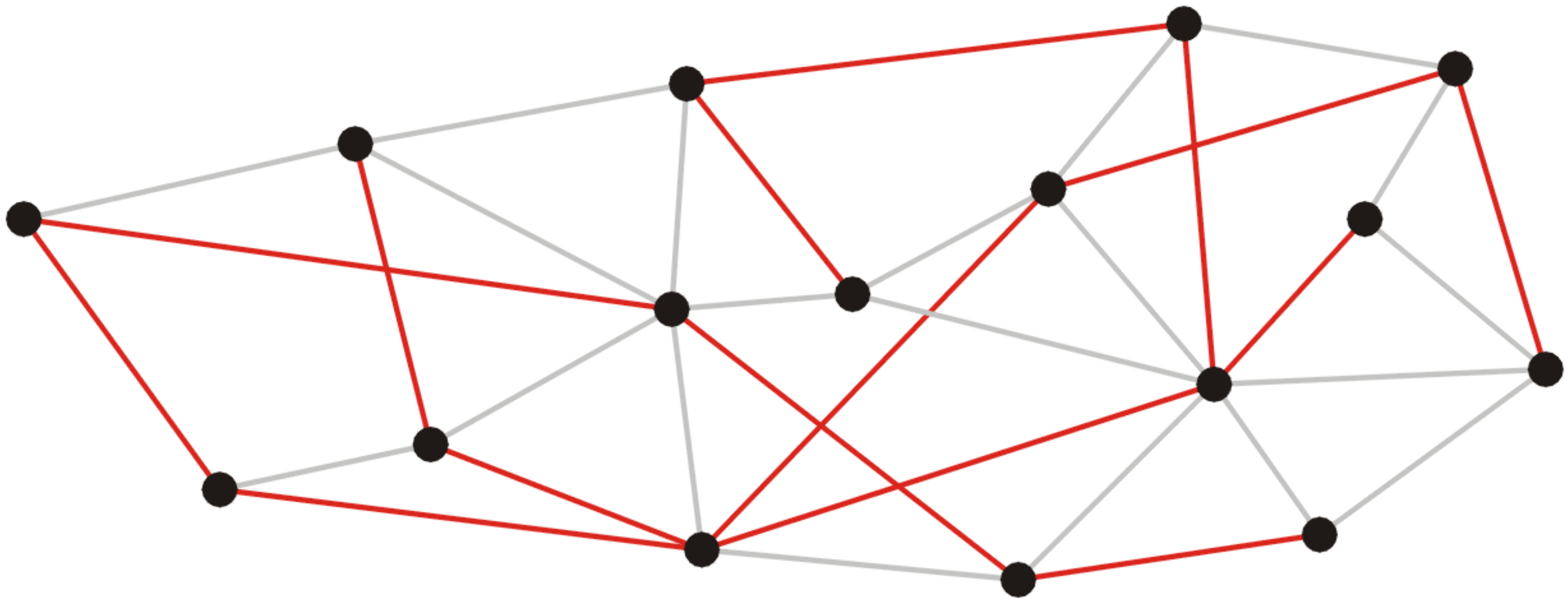
Spanning Trees – Example

- These 15 edges form a spanning tree

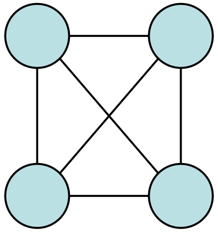


Spanning Trees – Example

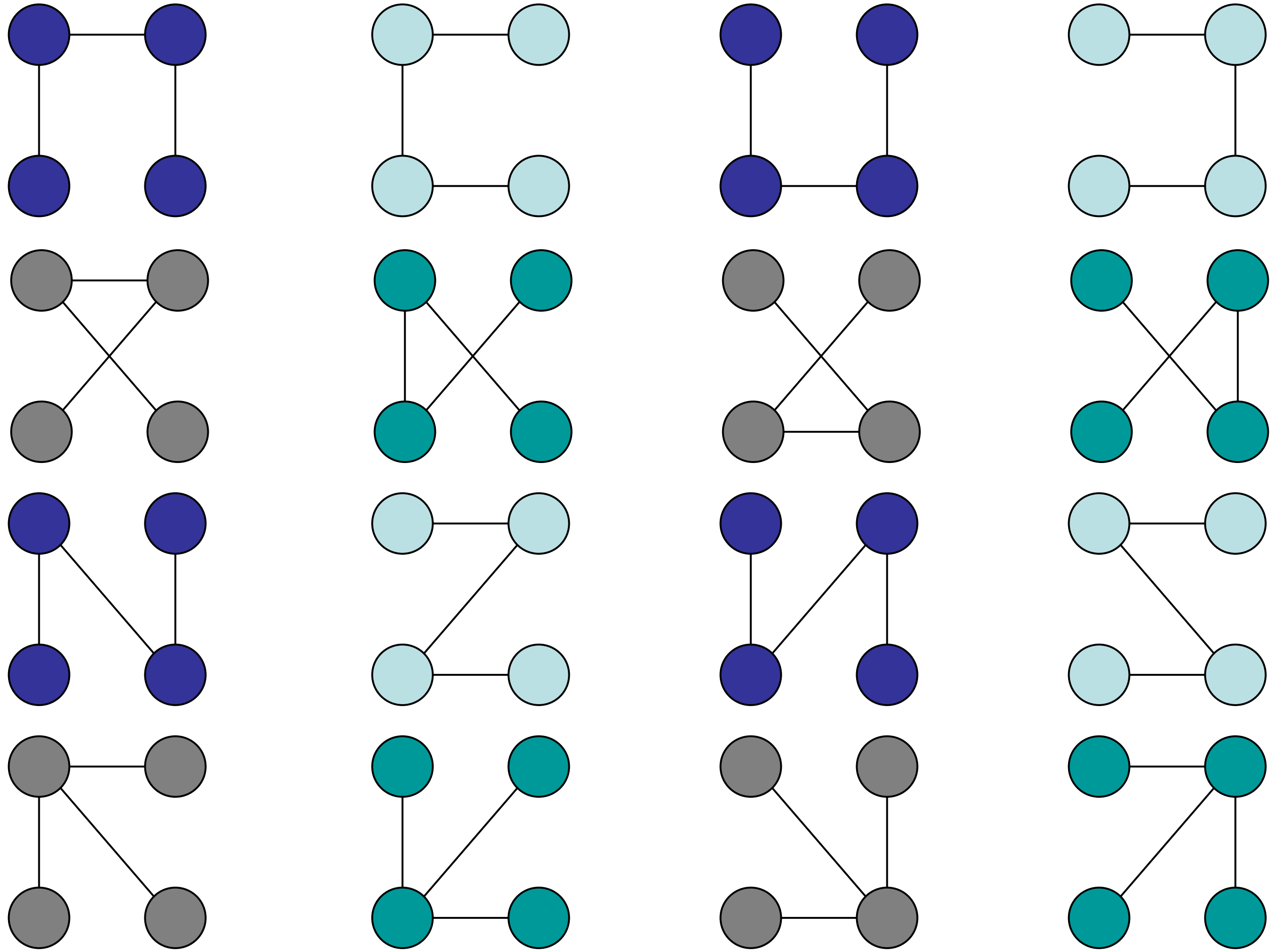
- As do these 15 edges



Spanning Trees – Example



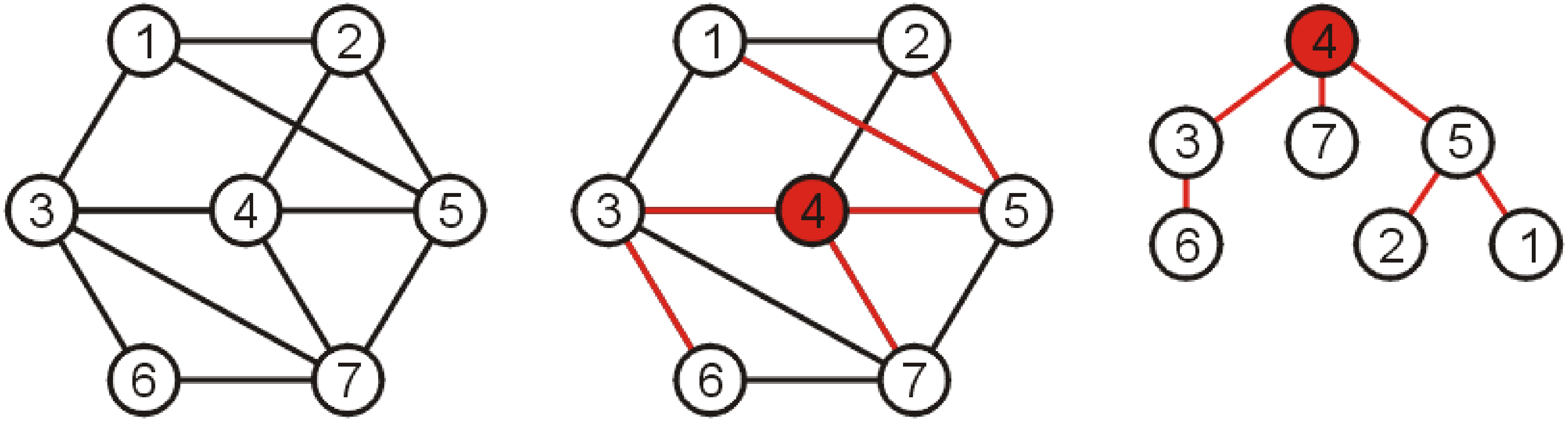
Graph



All 16 of its Spanning Trees

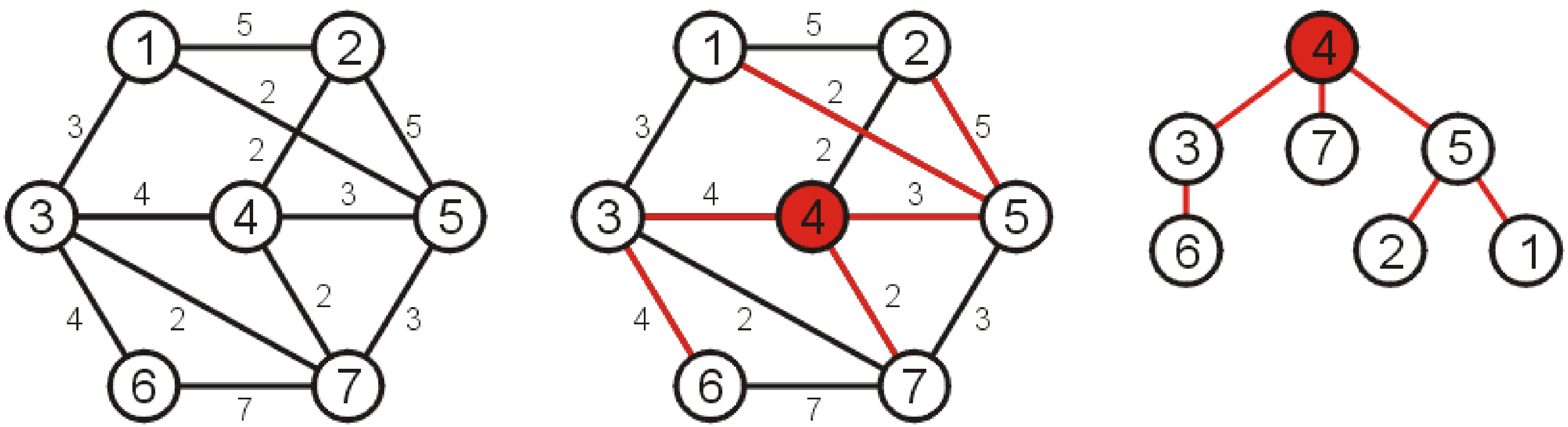
Spanning Trees

- Why such a collection of `|V|-1` edges is called a tree?
  - If any vertex is taken to be the root, we form a tree by treating the adjacent vertices as children, and so on...



Spanning Tree on Weighted Graphs

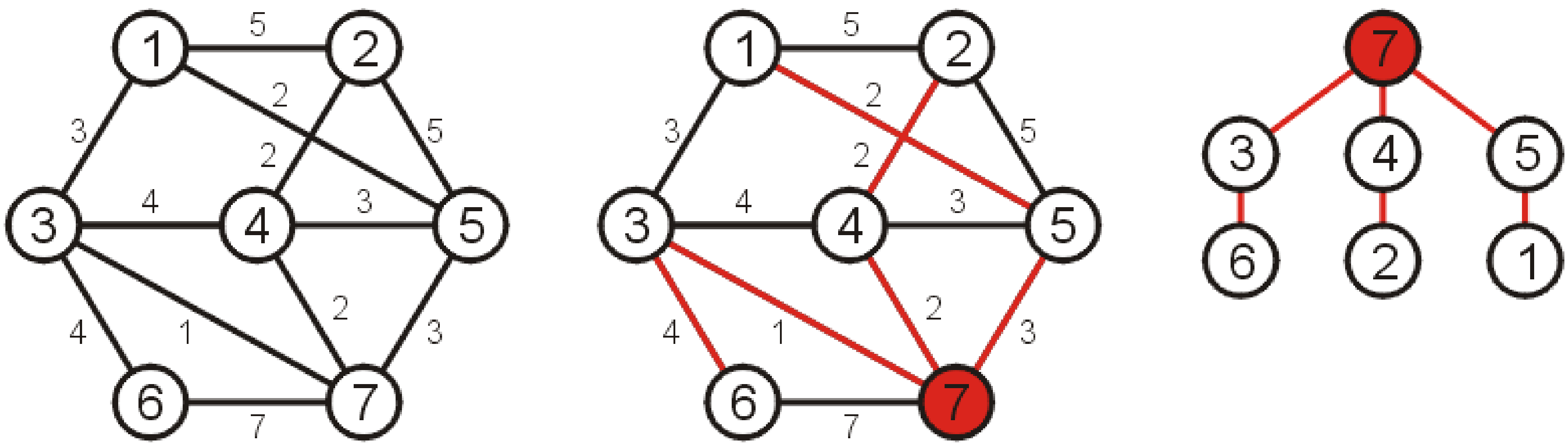
- Weight of a spanning tree
  - Sum of the weights on all the edges which comprise the spanning tree



- The weight of this spanning tree is 20

Minimum Spanning Tree (MST)

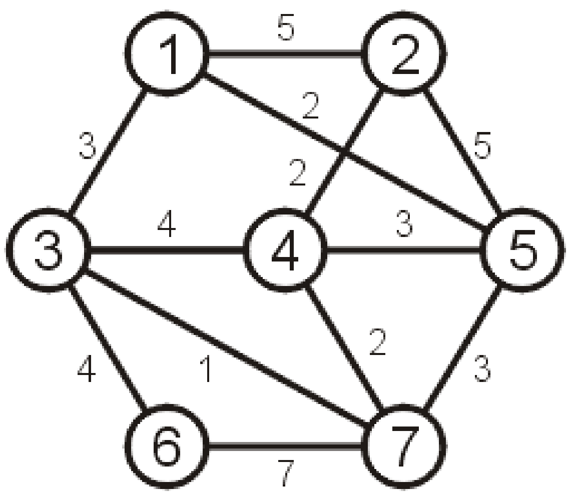
- Spanning tree that minimizes the weight
  - Such a tree is termed a minimum spanning tree

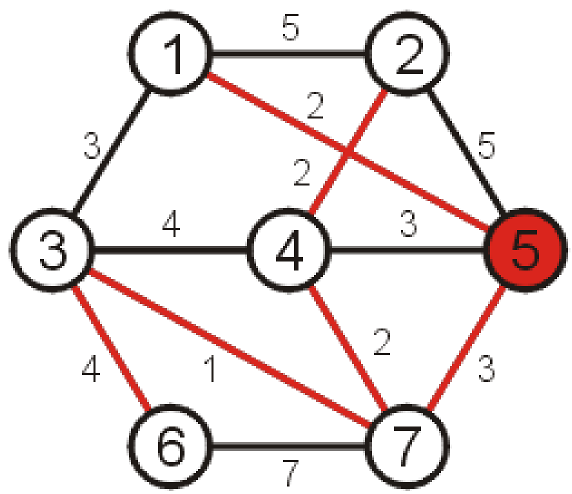


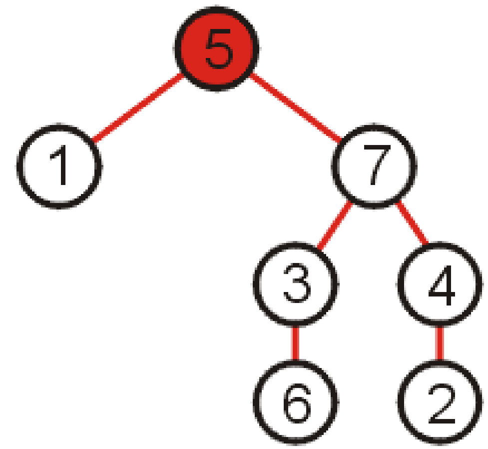
- The weight of this spanning tree is 14

Minimum Spanning Tree (MST)

- If a different vertex is used as the root
  - A different tree is obtained
  - However, this is simply the result of one or more rotations

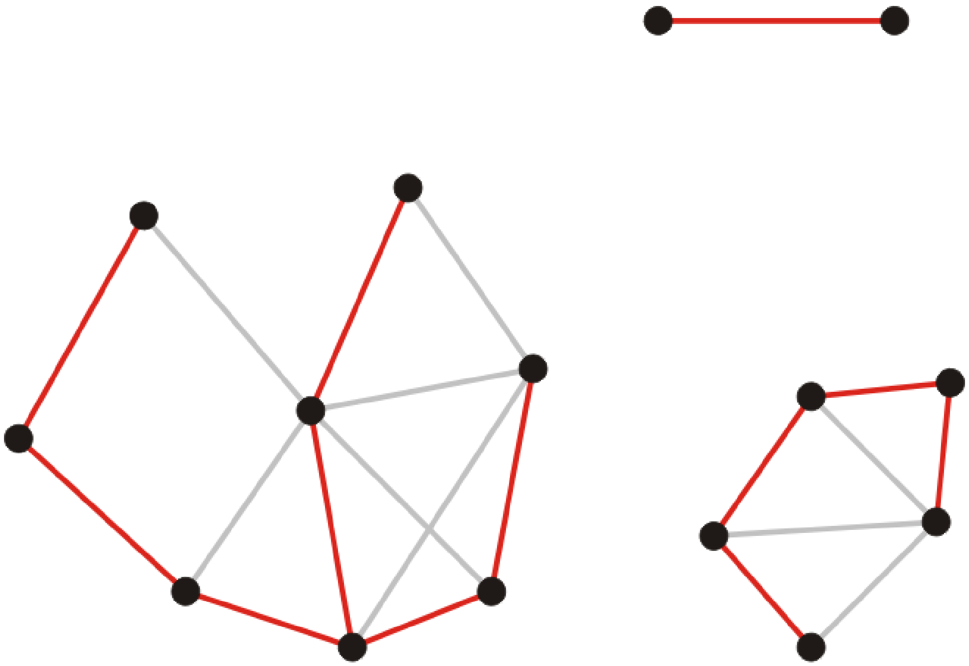






Spanning Forest

- Suppose that a graph is composed of N connected sub-graphs
- A spanning forest is a collection of N spanning trees
  - One for each connected sub-graph

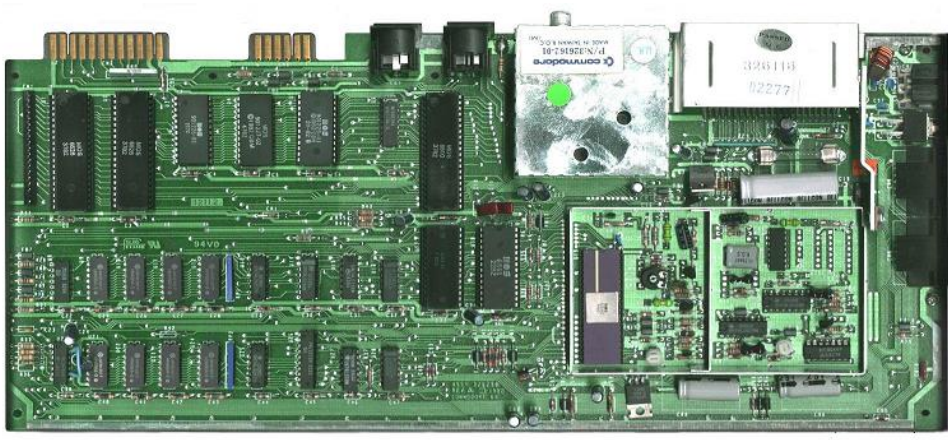


- A minimum spanning forest
  - A collection of N minimum spanning trees
  - One for each connected vertex-induced sub-graph

Applications

- Consider supplying power to
  - All circuit elements on a board
  - A number of loads within a building

- A minimum spanning tree will give the lowest-cost solution

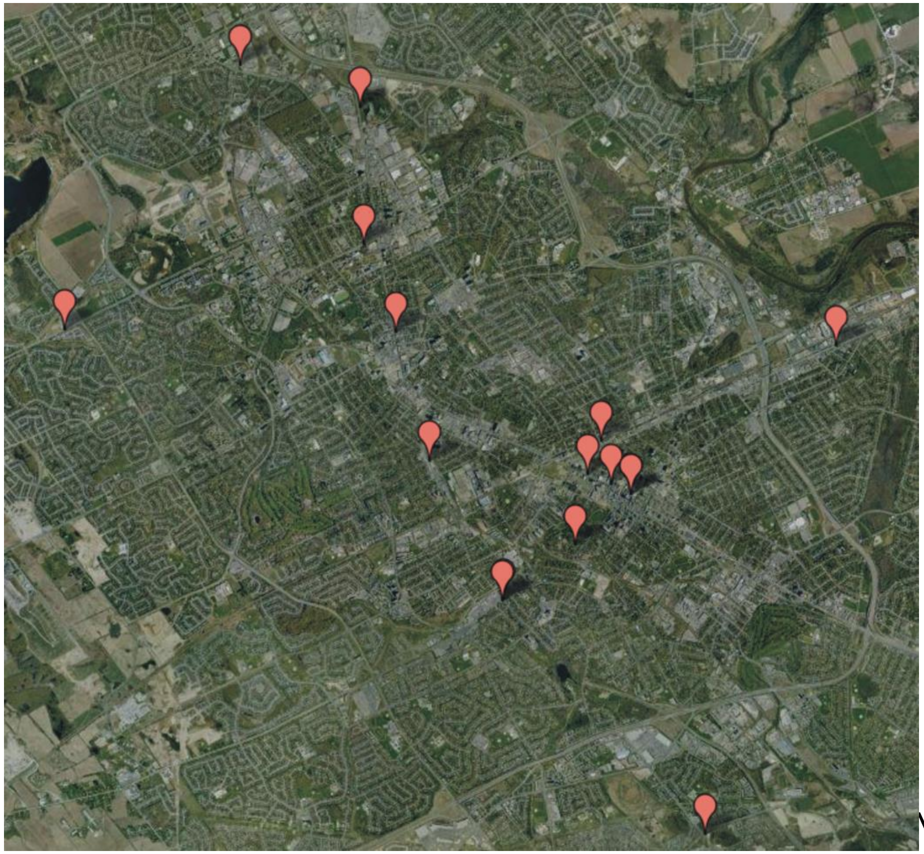


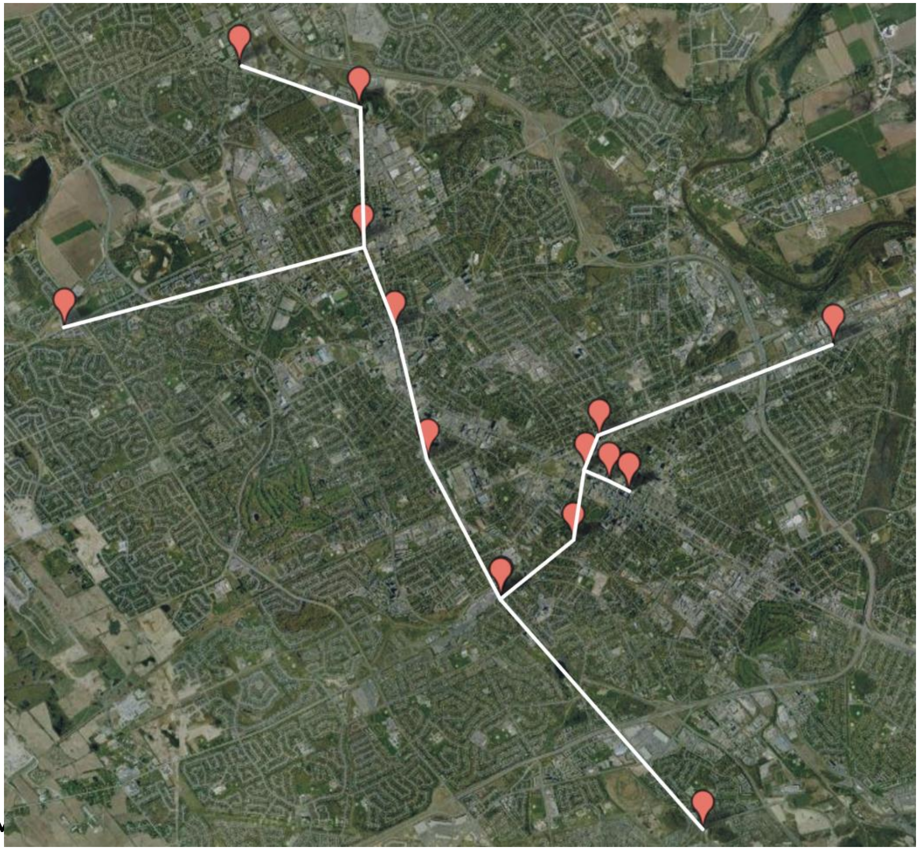
Application

- First application of a minimum spanning tree algorithm was by the Czech mathematician Otakar Borůvka
  - Designed electricity grid in Morovia in 1926

Application

- Consider attempting to find the best means of connecting a number of Local Area Networks (LANs)
  - Minimize the number of bridges
  - Costs not strictly dependent on distances





Algorithms For Obtaining MST

- Kruskal's Algorithm
- Prim's Algorithm
- Boruvka's Algorithm

Kruskal’s Algorithm

Kruskal’s Algorithm

- Kruskal’s algorithm creates a forest of trees
- Initially forest consists of N single node trees (and no edges)
- Sorts the edges by weight and goes through the edges from least weight to greatest weight
- At each step one edge (with least weight) is added so that it joins two trees together
  - As long as the addition does not create a cycle

Kruskal’s Algorithm

The halting conditions are as follows:

1. When `|V| - 1` edges have been added
   – In this case we have a minimum spanning tree

2. We have gone through all edges
   – A forest of minimum spanning trees on all connected sub-graphs

Kruskal’s Algorithm

1. The forest is constructed – with each node in a separate tree

2. The edges are placed in a priority queue

3. Until we've added `n-1` edges (assumption: connected graph)
   1. Extract the cheapest edge from the queue

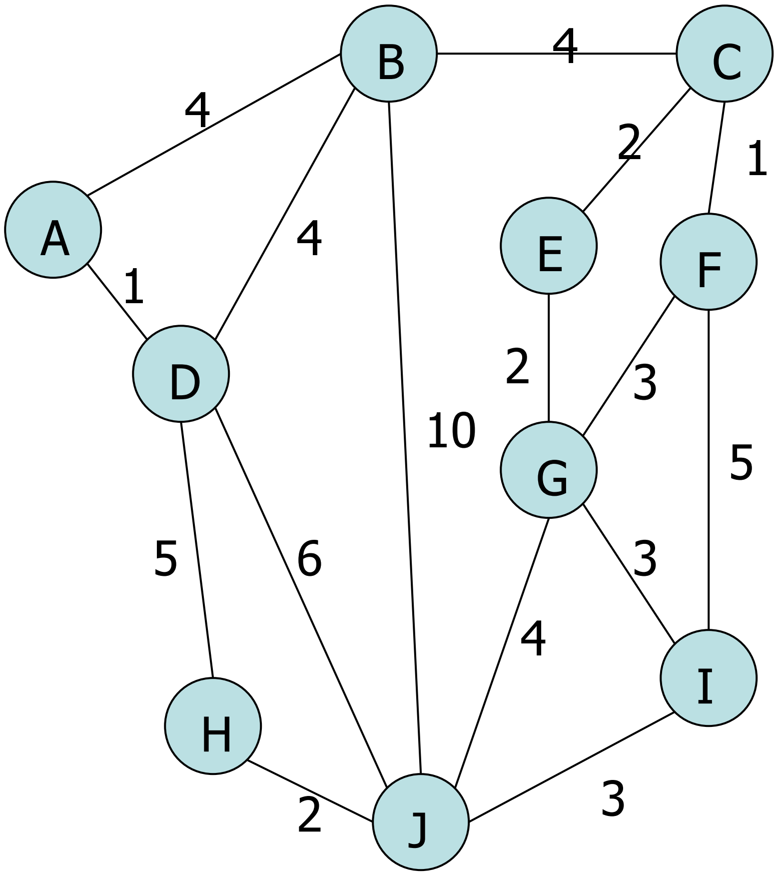
   2. If it forms a cycle, reject it

   3. Else add it to the forest. Adding it to the forest will join two trees

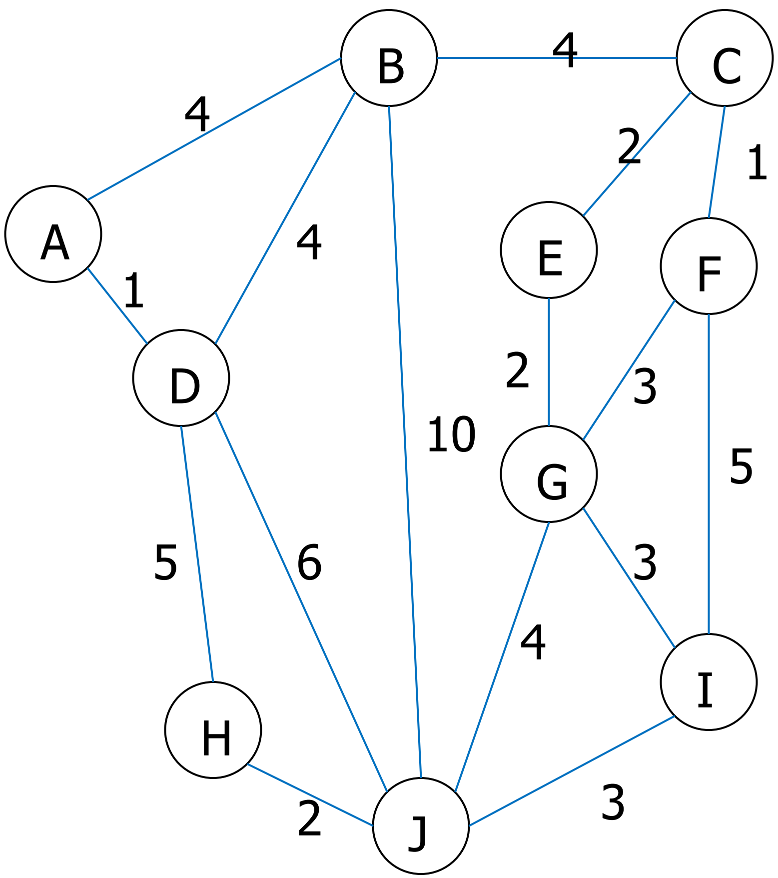
- Every step will have joined two trees in the forest together, so that at the end, there will only be one tree

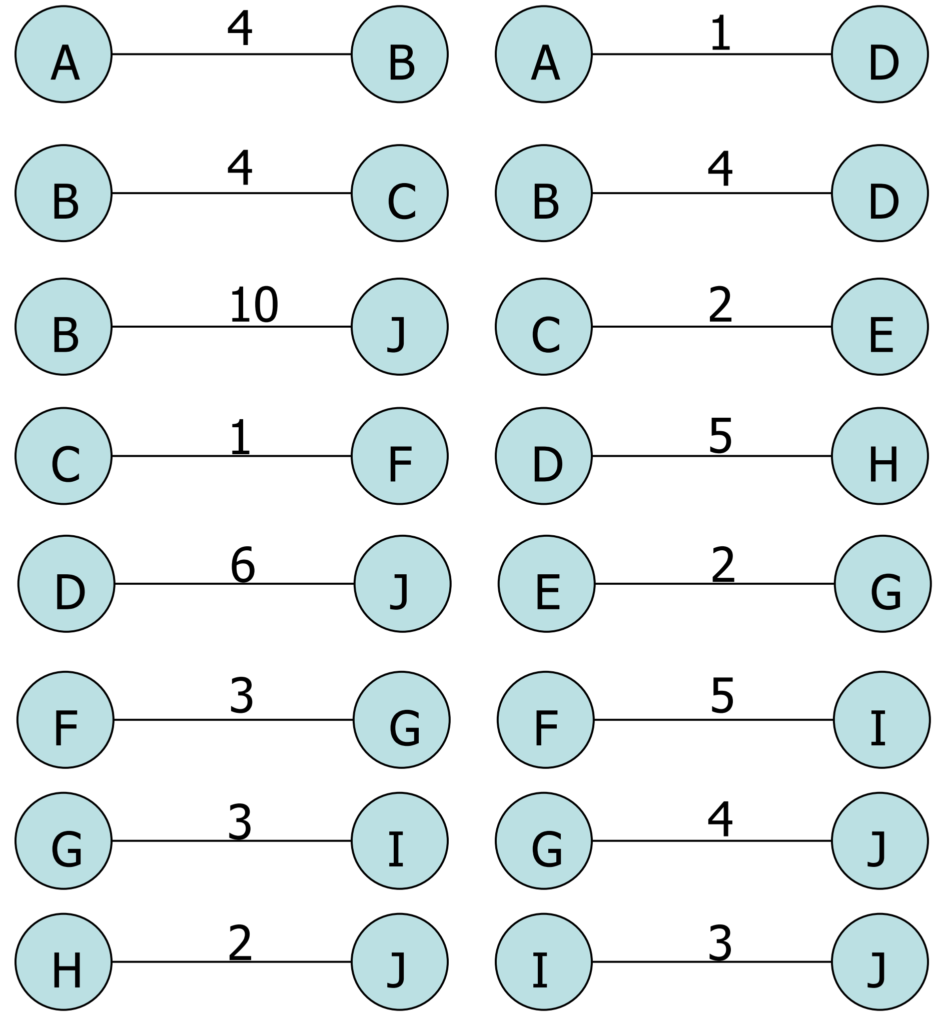
Kruskal’s Algorithm – Example

- Complete Graph



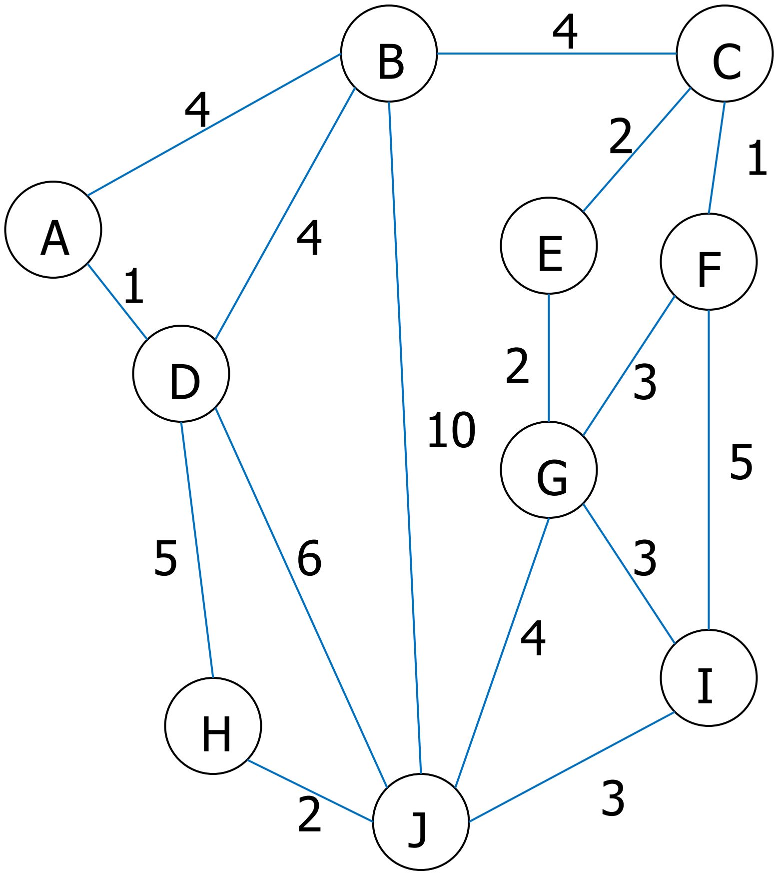
Kruskal's Algorithm – Example

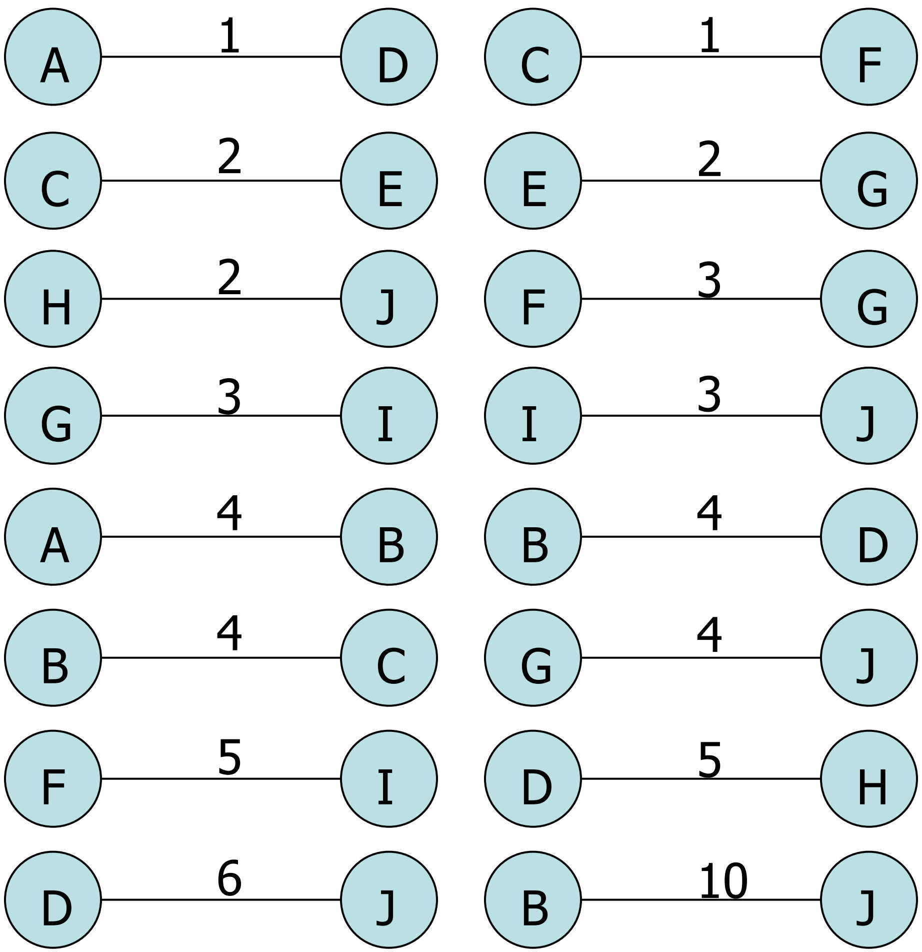




Kruskal's Algorithm – Example

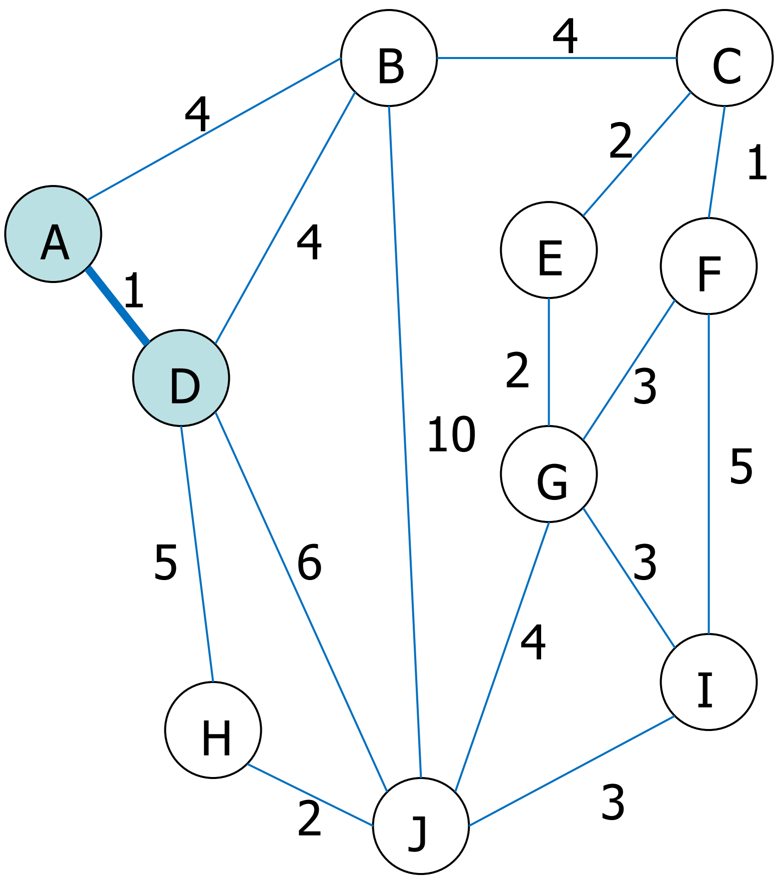
- Sort Edges: In reality priority queue is used

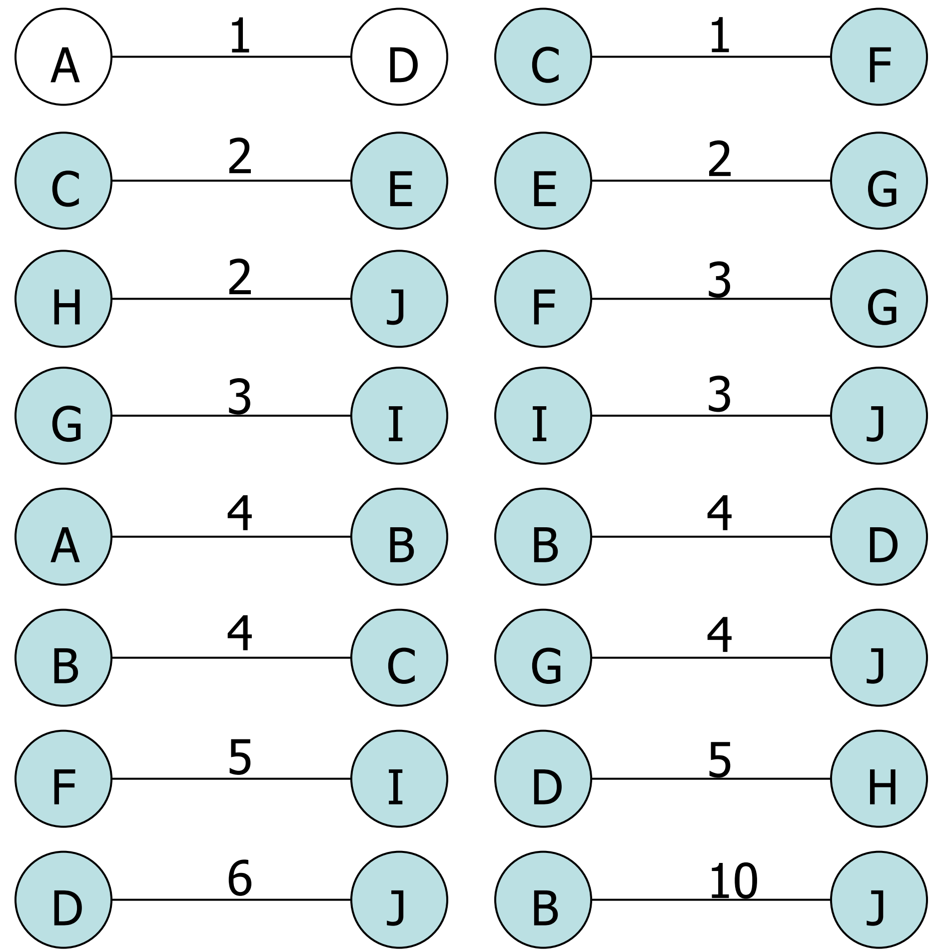




Kruskal’s Algorithm – Example

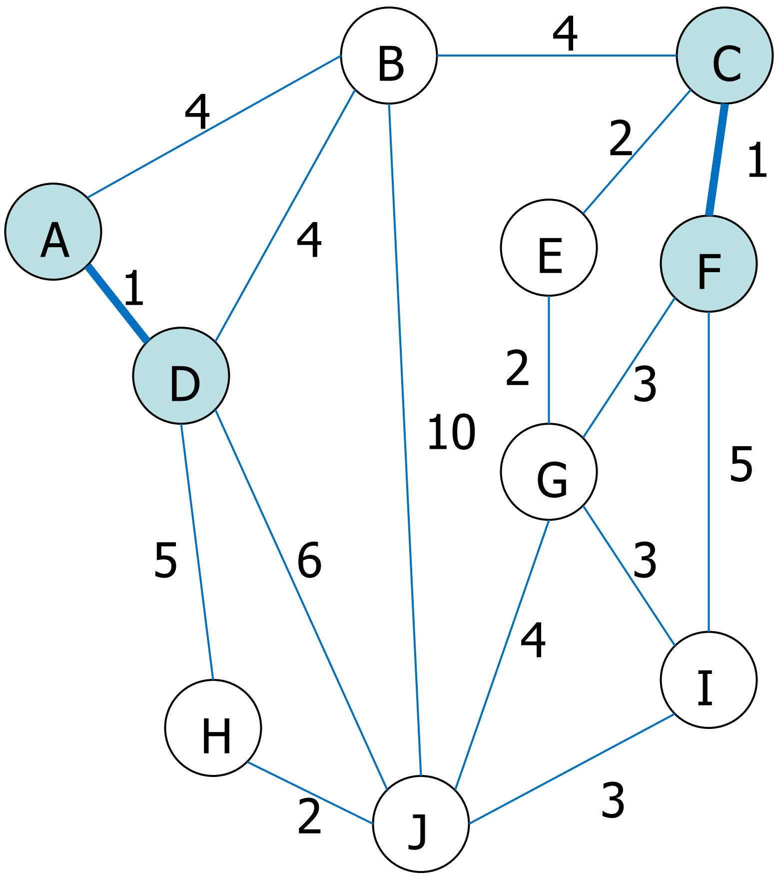
- Add Edge

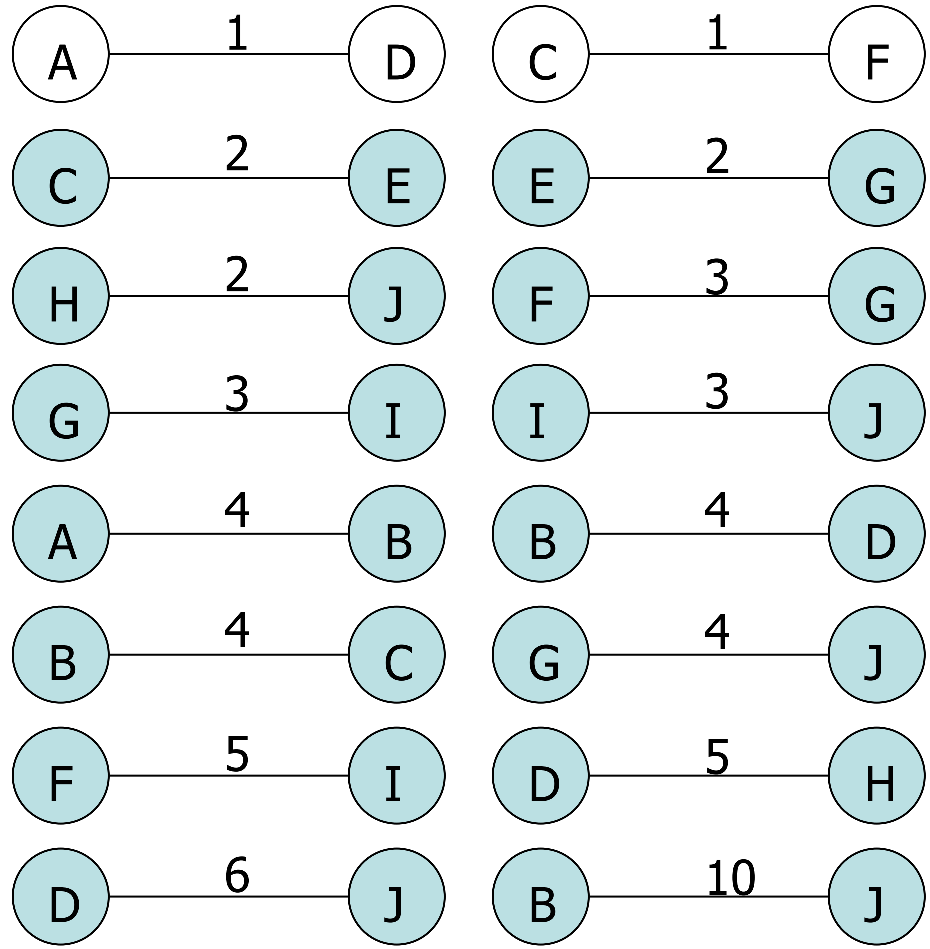




Kruskal’s Algorithm – Example

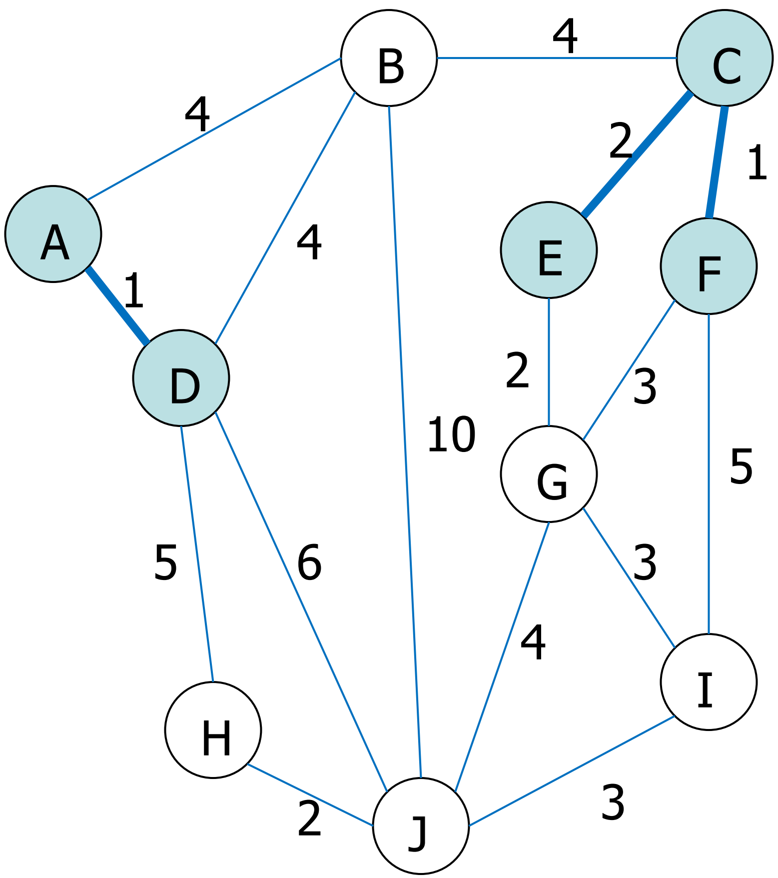
- Add Edge

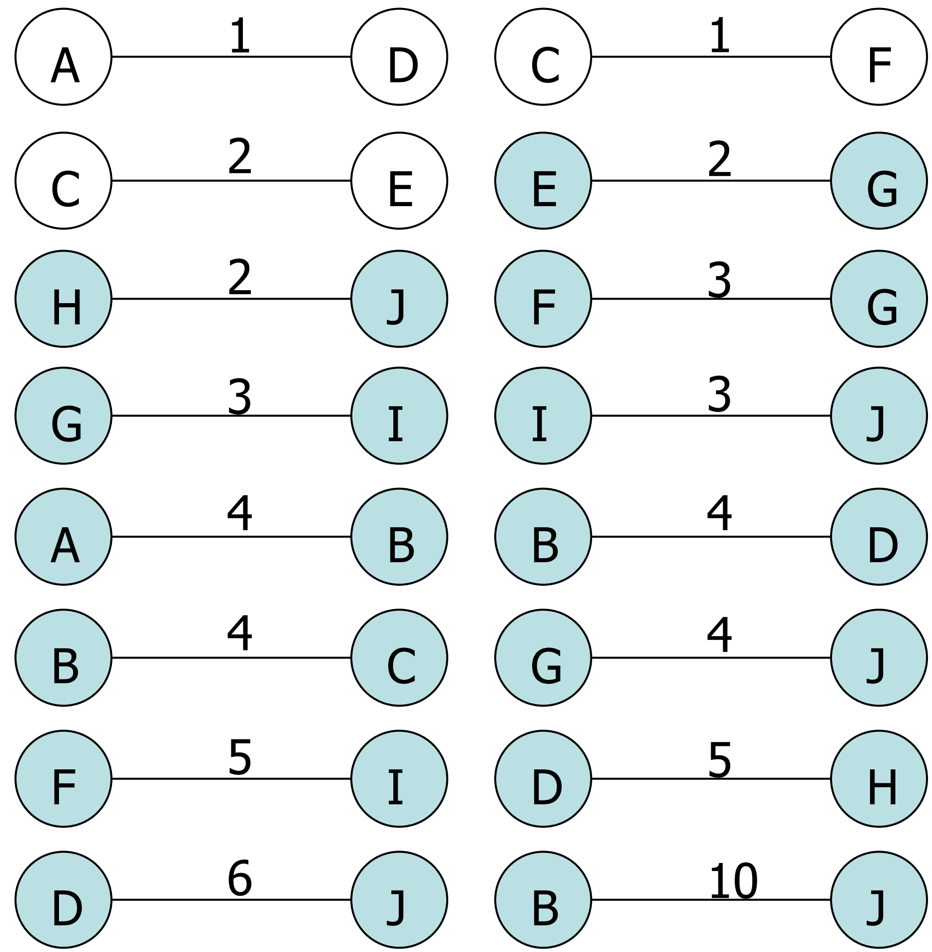




Kruskal’s Algorithm – Example

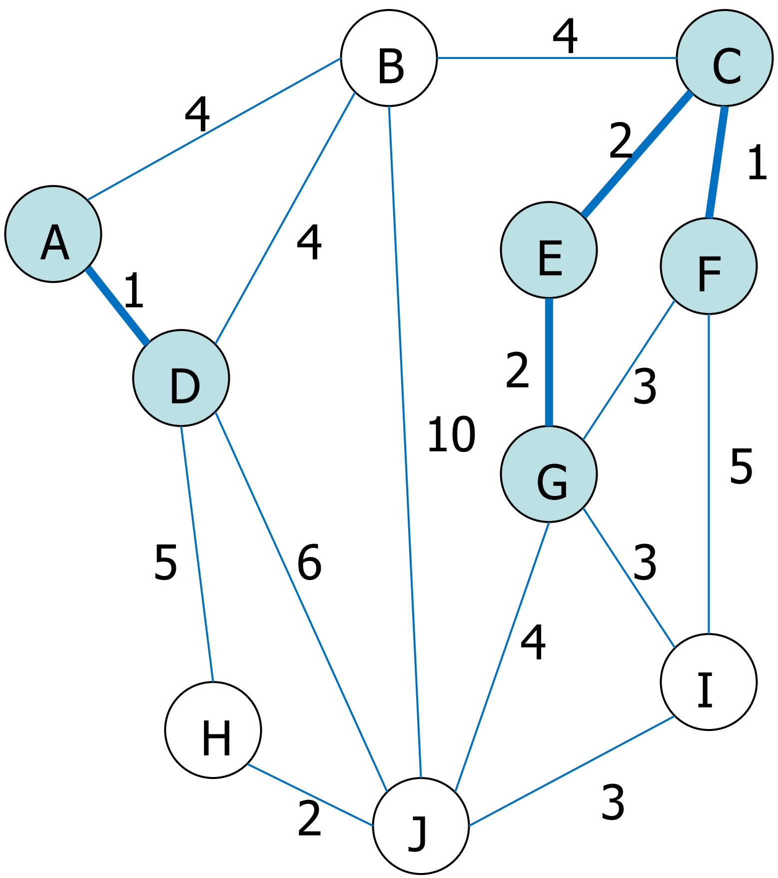
- Add Edge

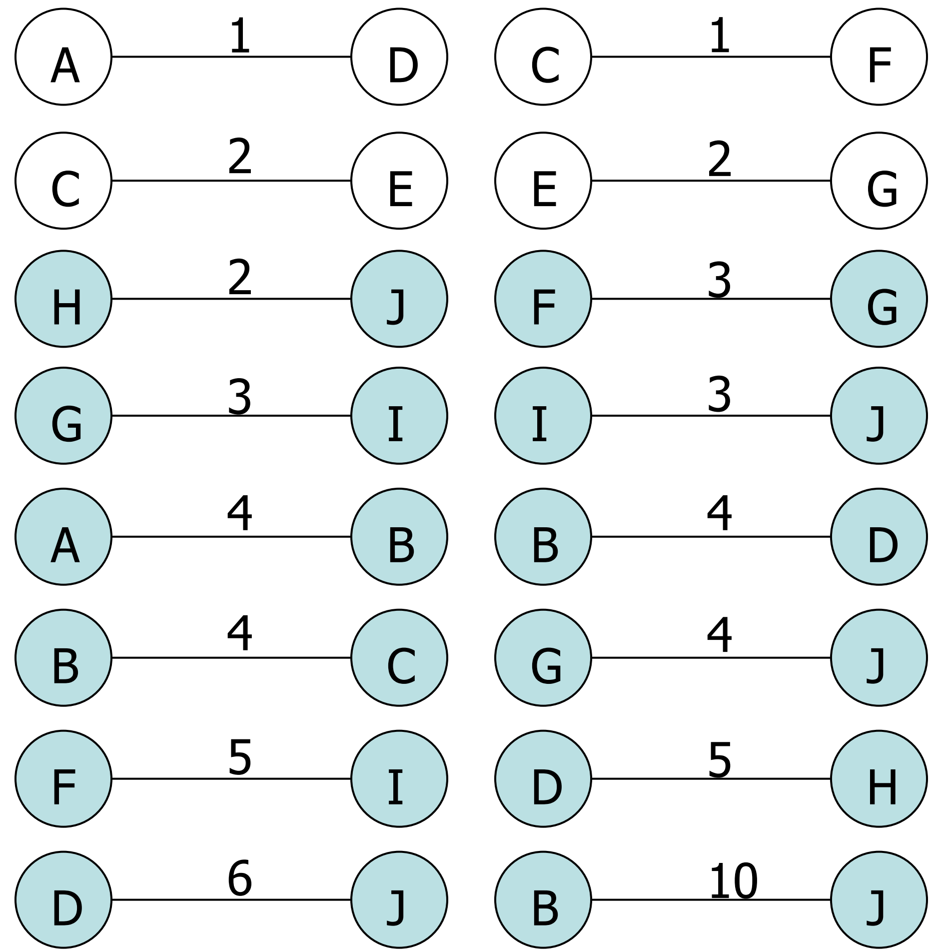




Kruskal's Algorithm – Example

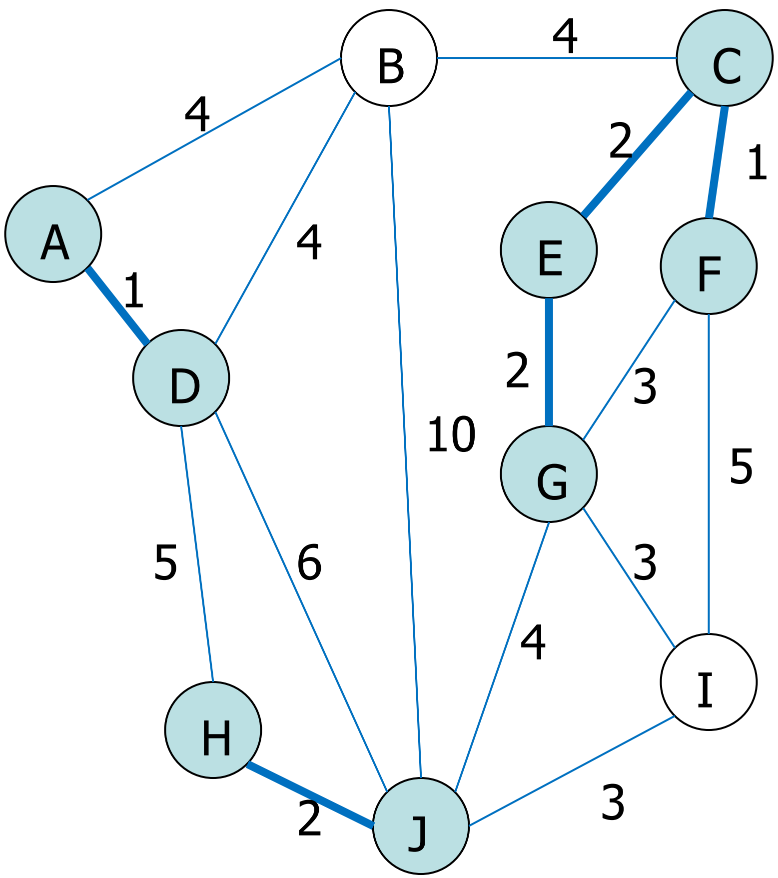
- Add Edge

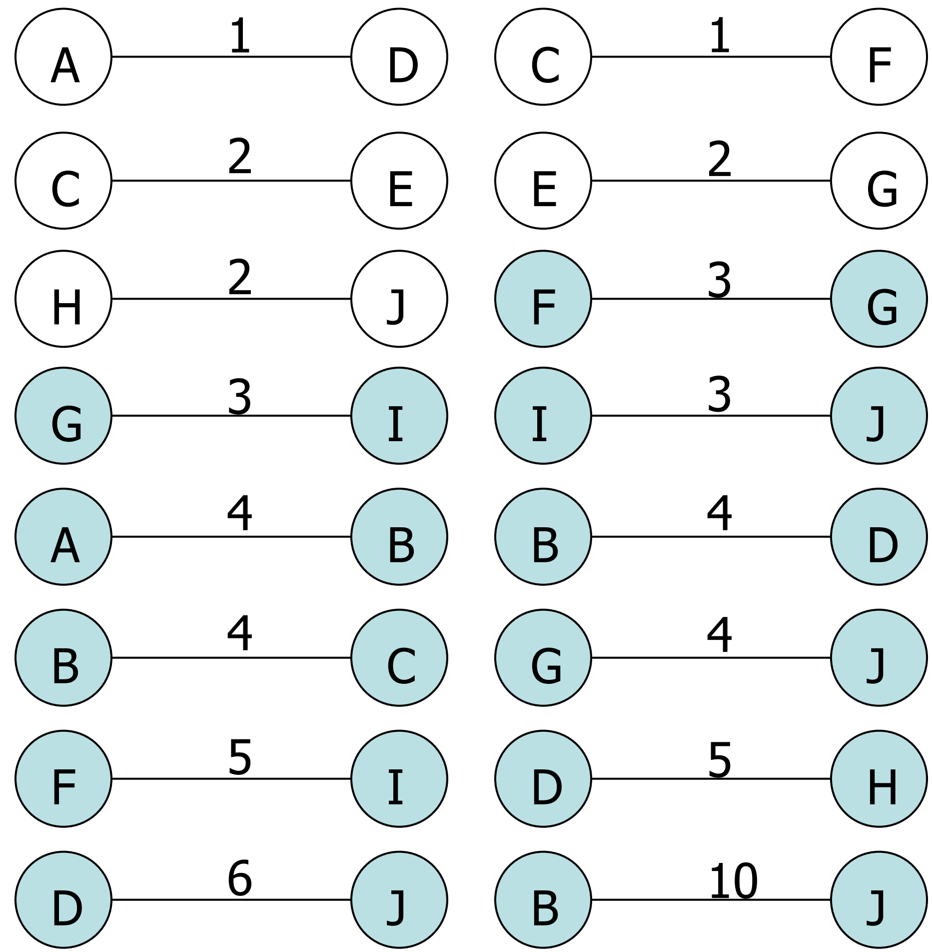




Kruskal’s Algorithm – Example

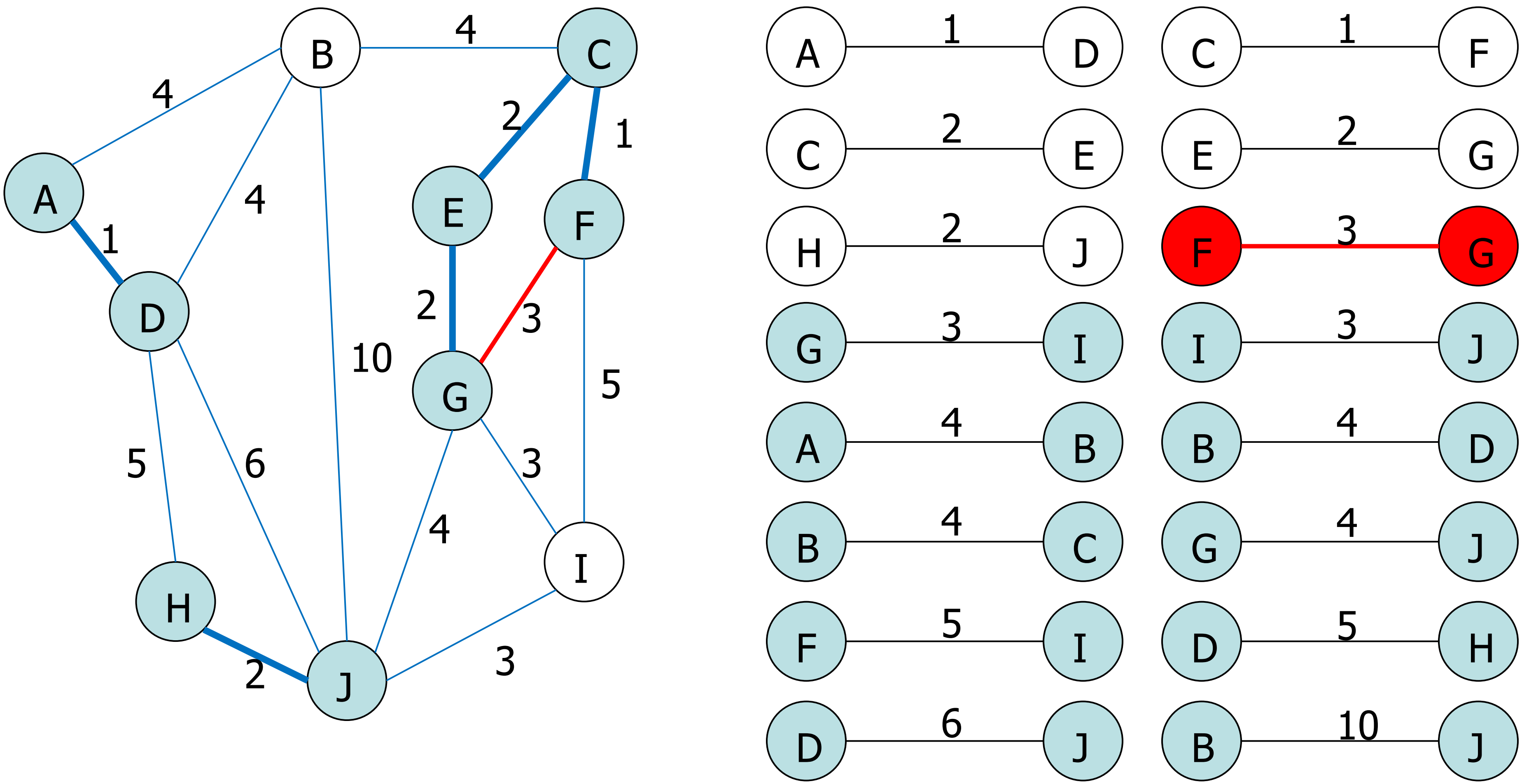
- Add Edge





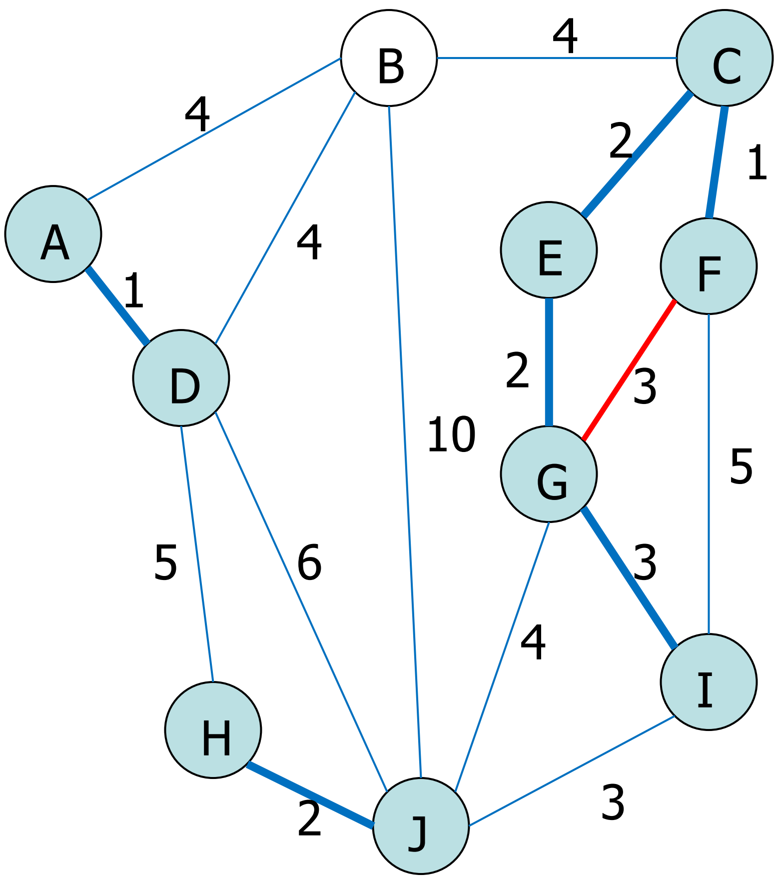
Kruskal’s Algorithm – Example

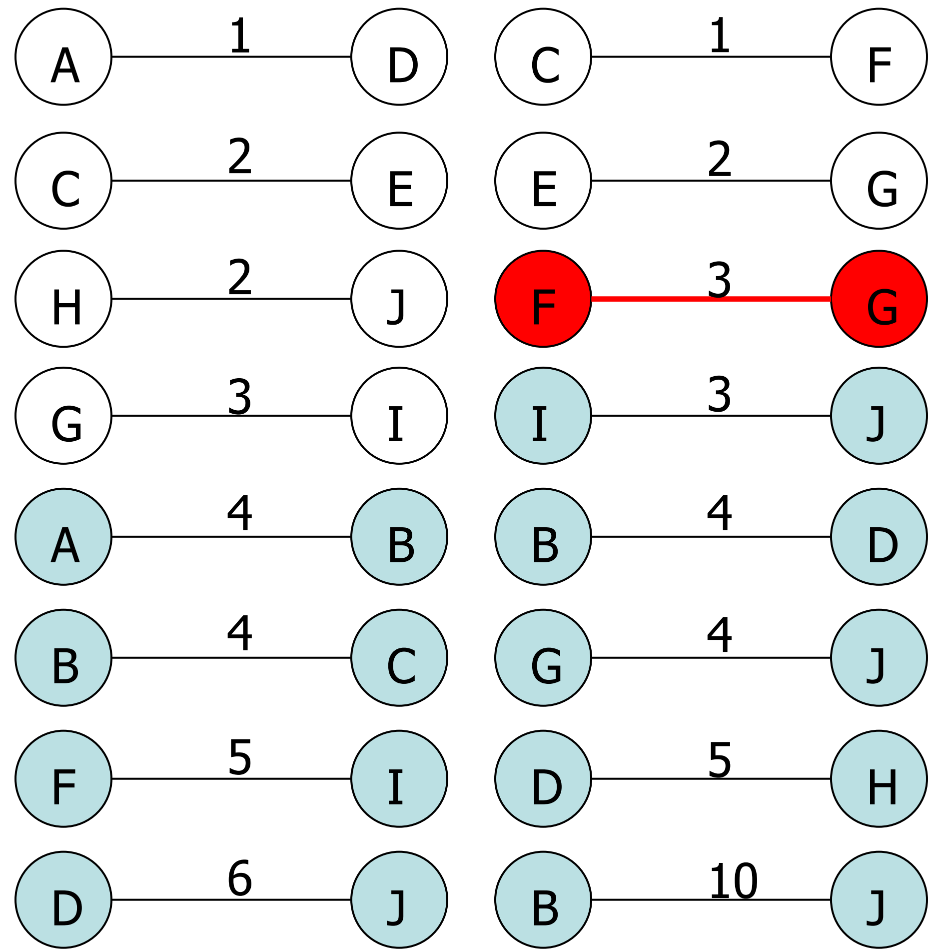
- Add Edge



Kruskal's Algorithm – Example

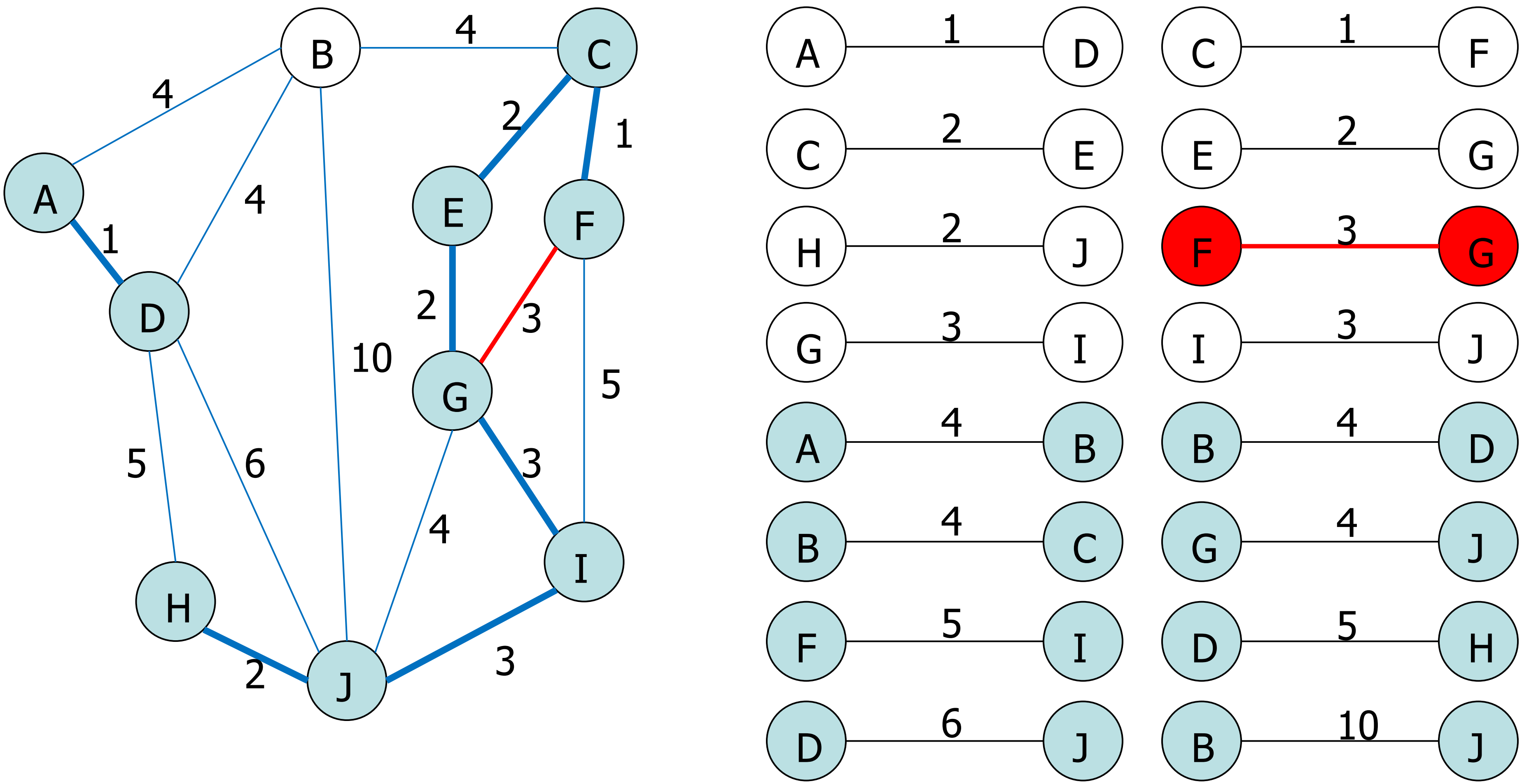
- Add Edge





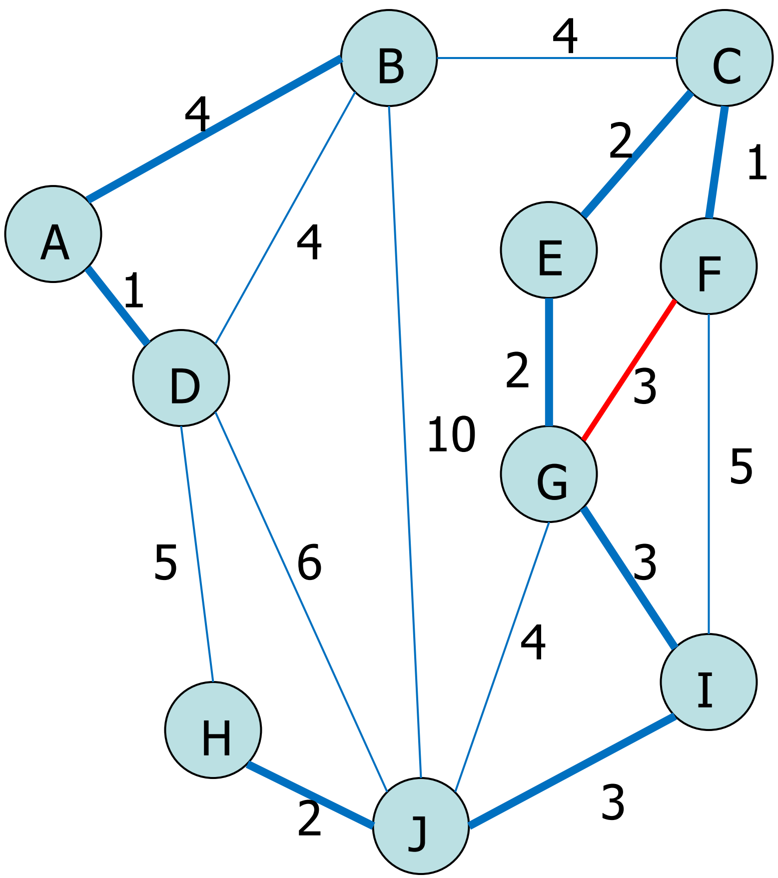
Kruskal's Algorithm – Example

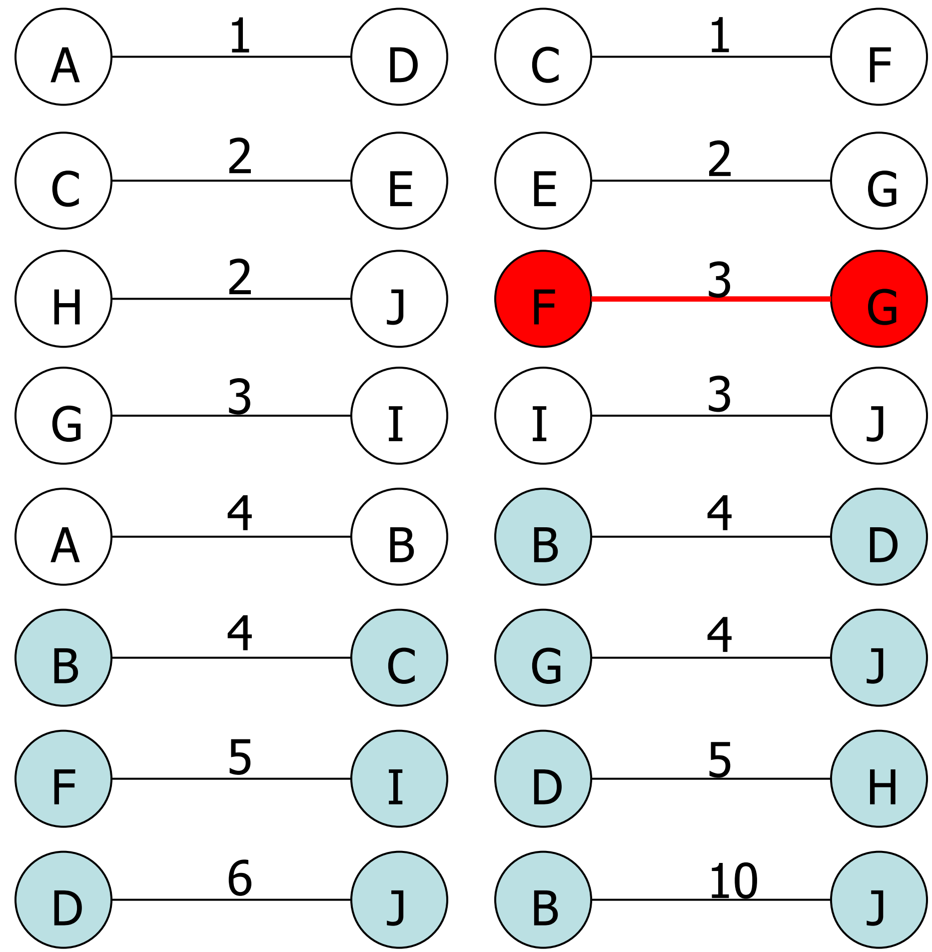
- Add Edge



Kruskal's Algorithm – Example

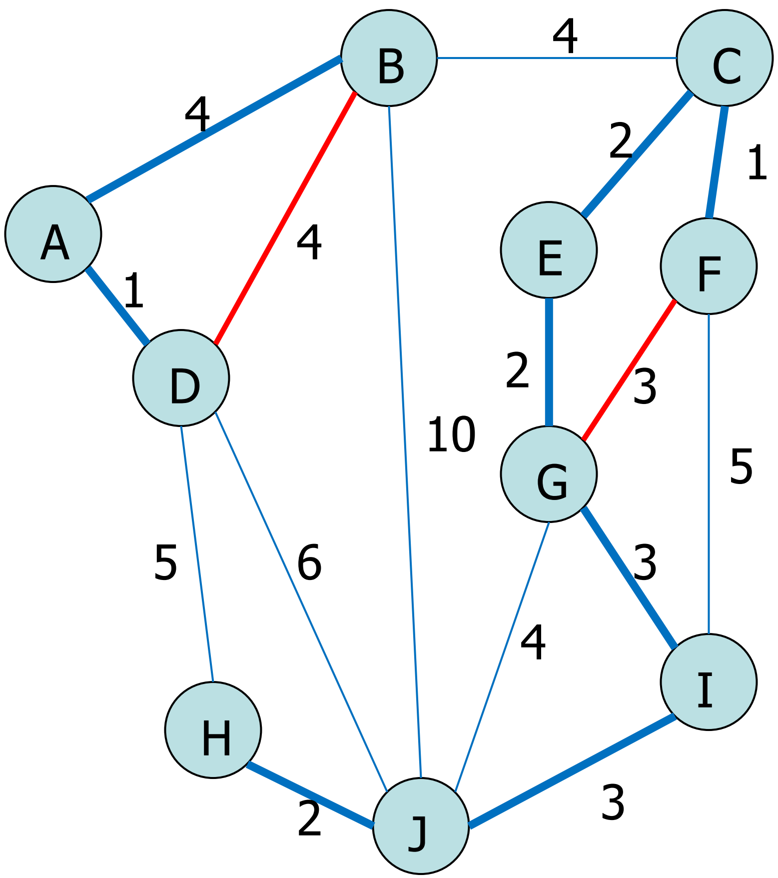
- Add Edge

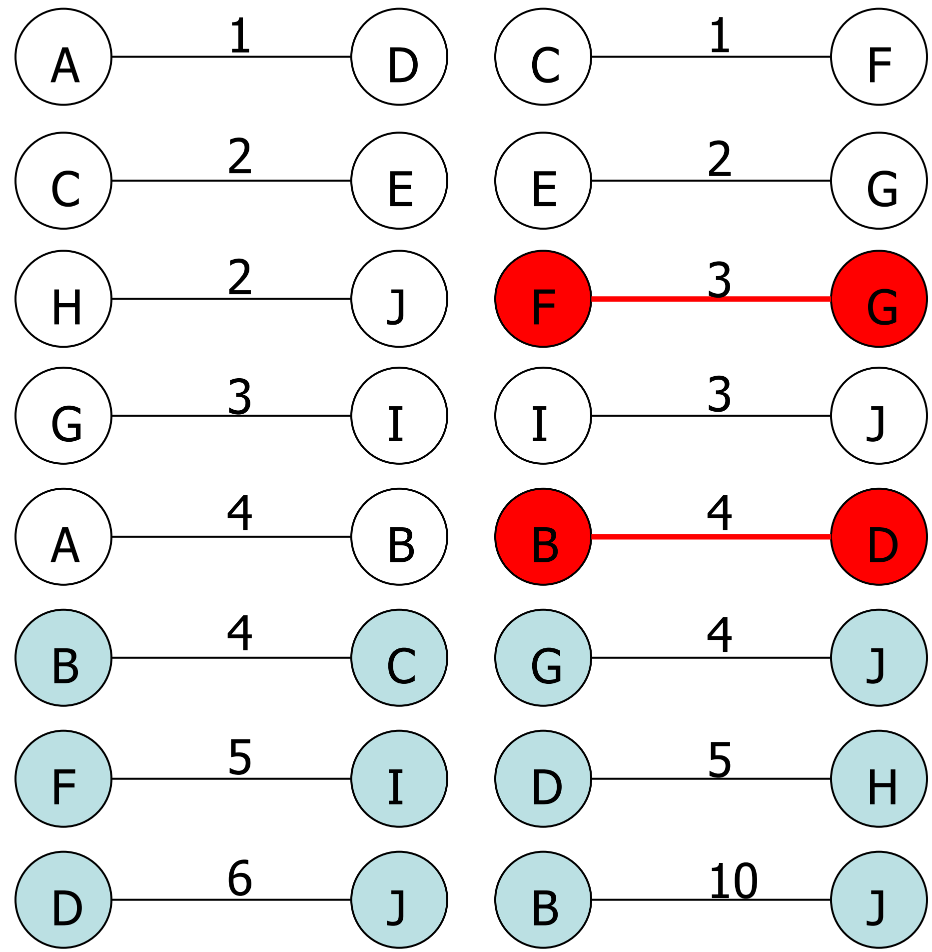




Kruskal's Algorithm – Example

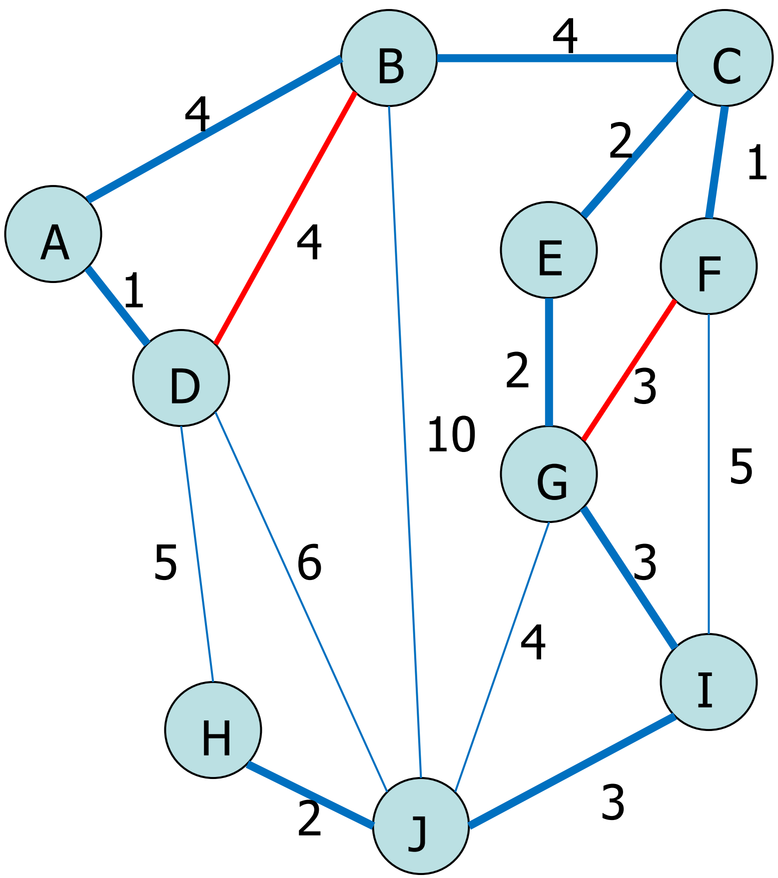
- Add Edge

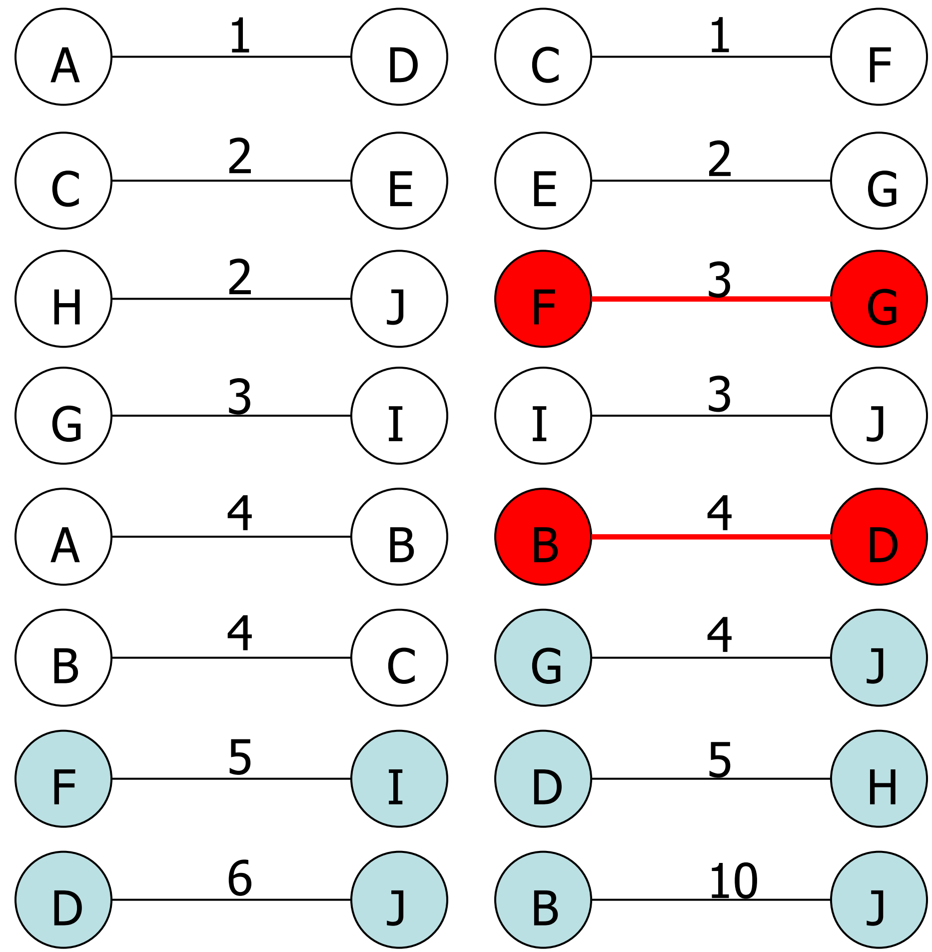




Kruskal's Algorithm – Example

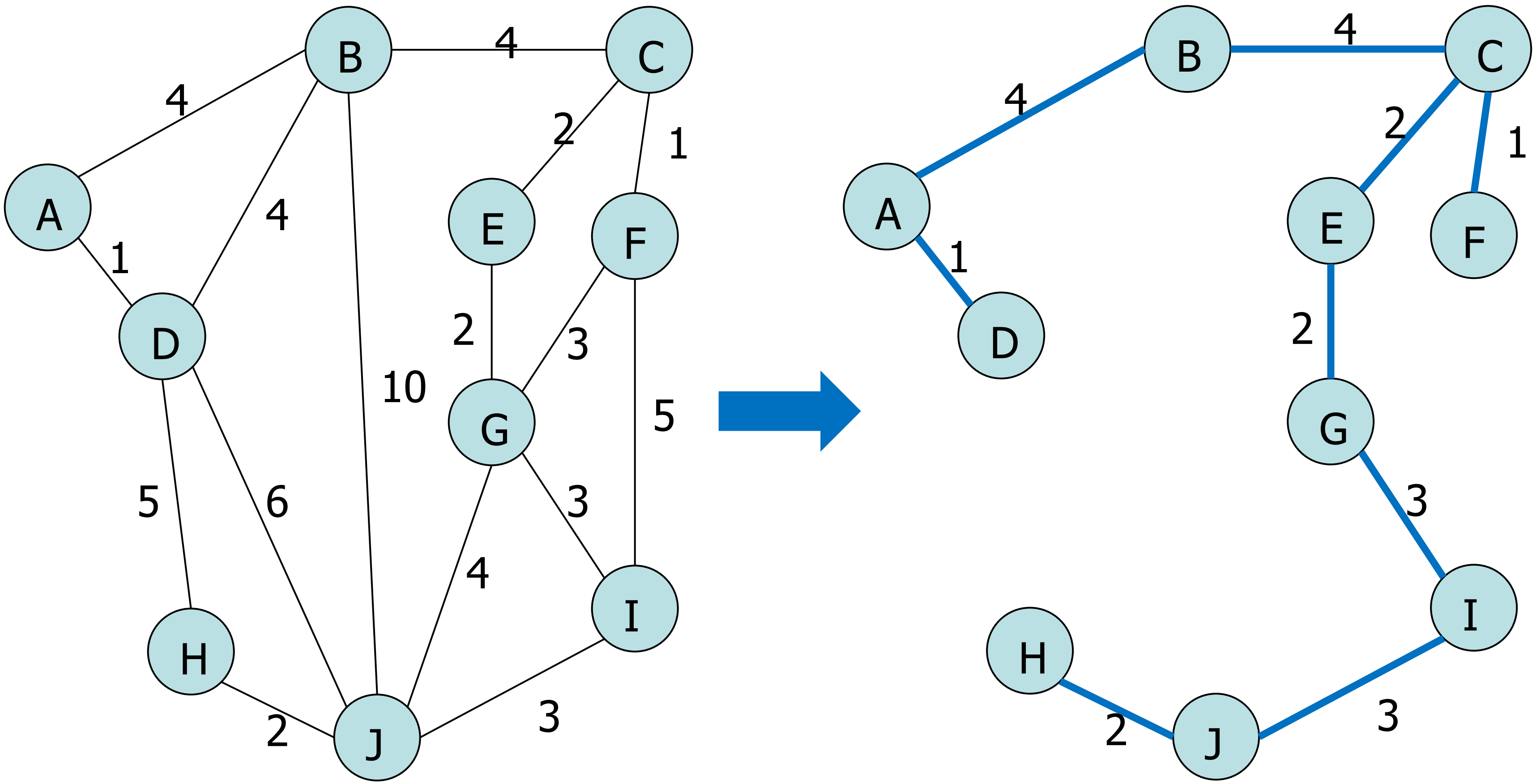
- Add Edge





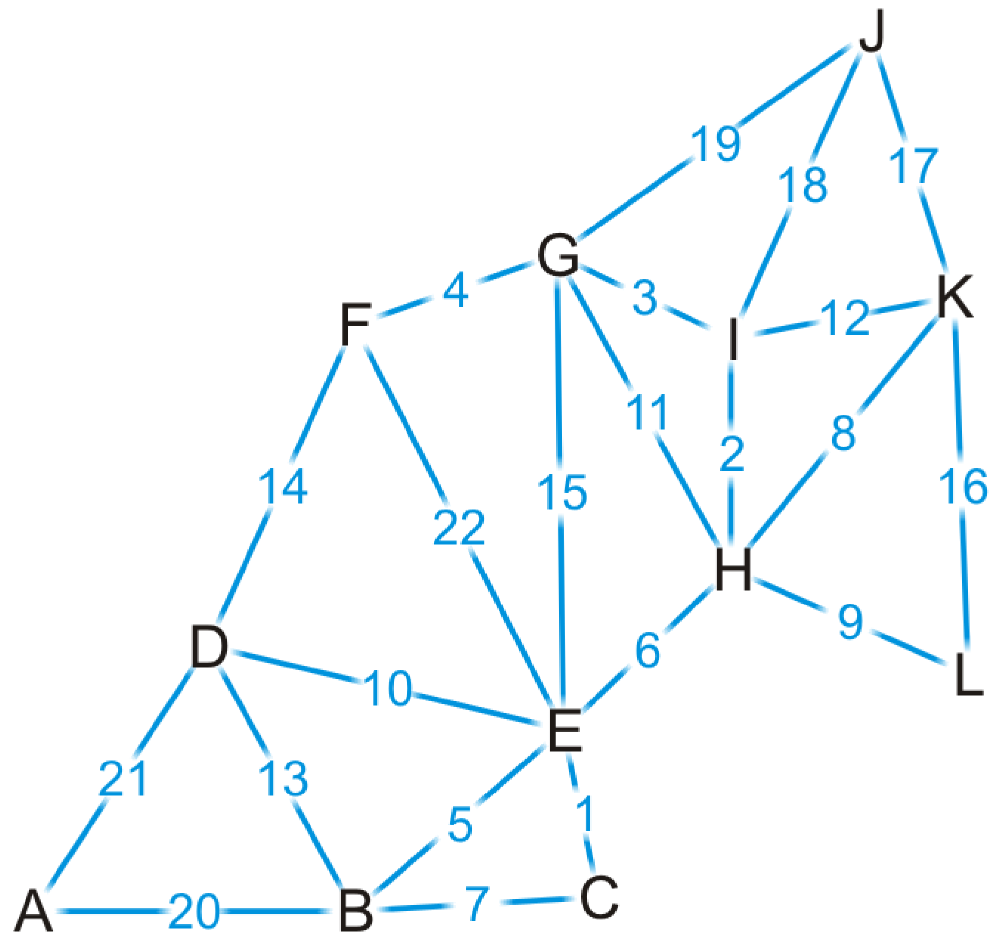
Kruskal's Algorithm – Example

- Minimum spanning tree



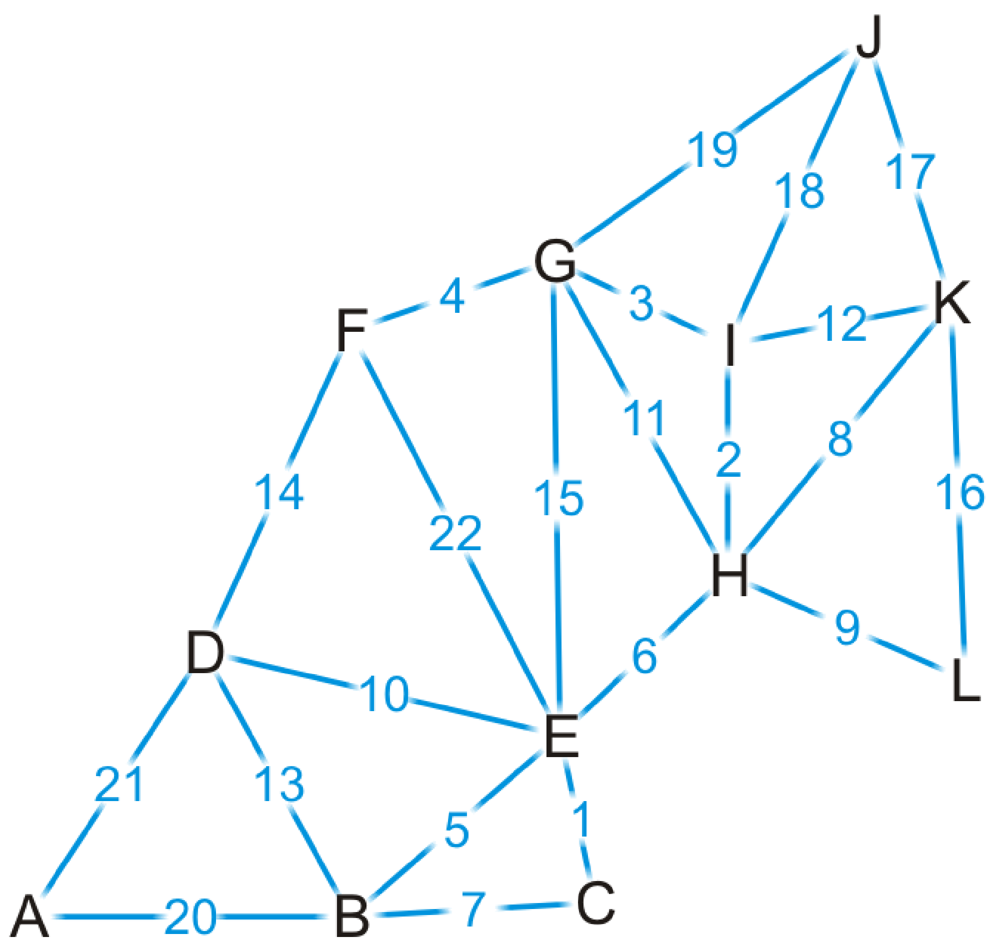
Kruskal's Algorithm – Example

- Complete graph



Kruskal’s Algorithm – Example

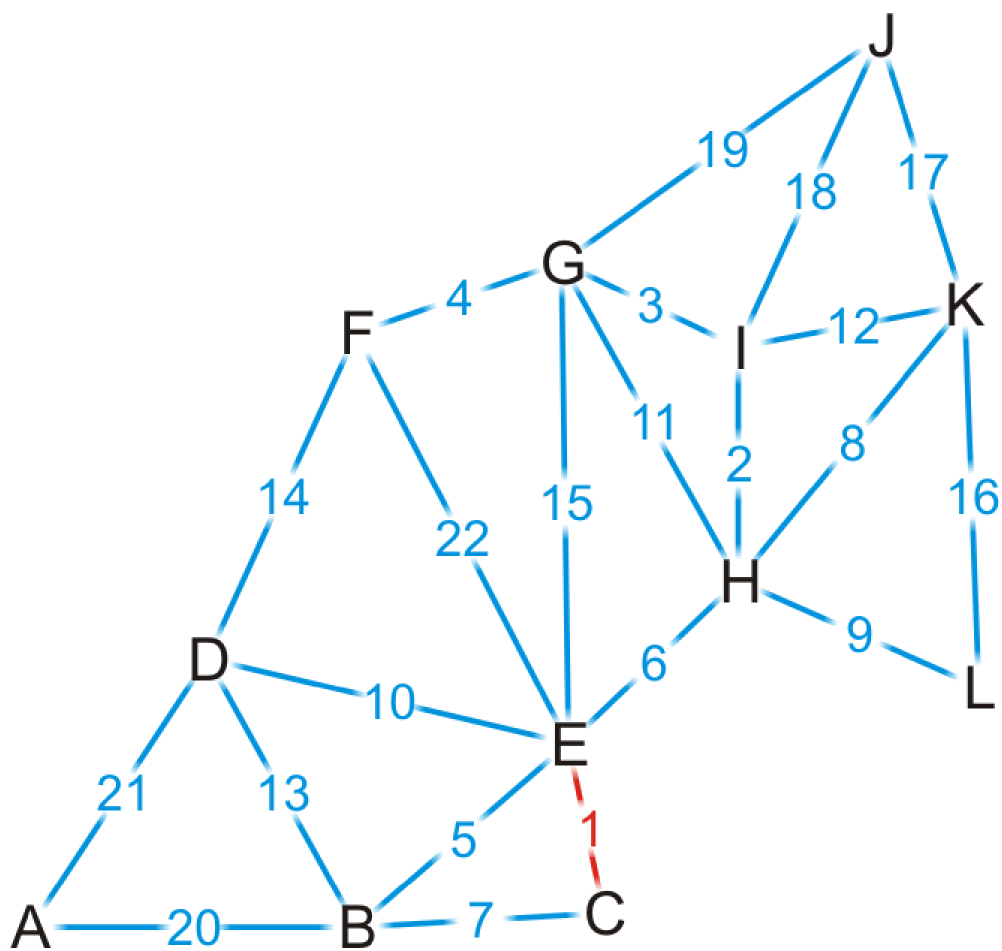
- Sort edges based on weight



{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Kruskal's Algorithm – Example

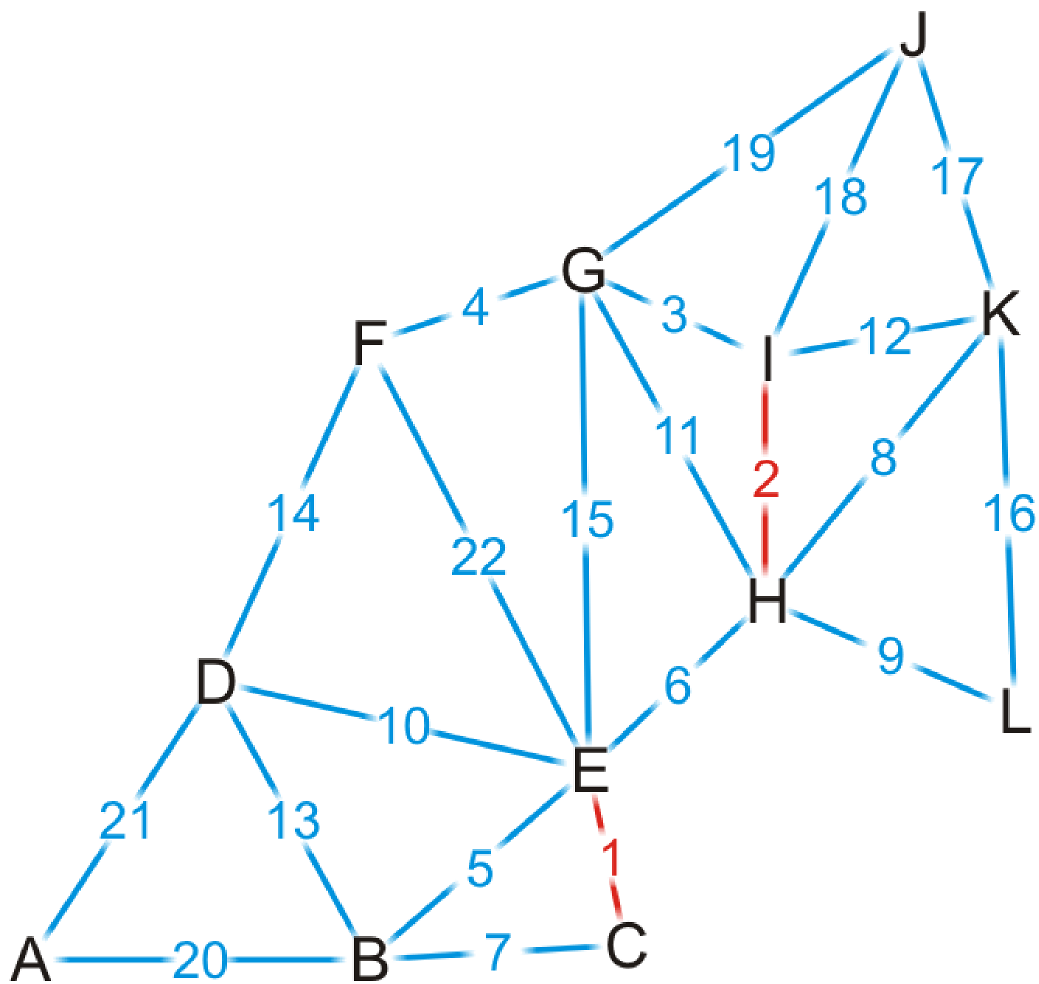
- We start by adding edge {C, E}



{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Kruskal's Algorithm – Example

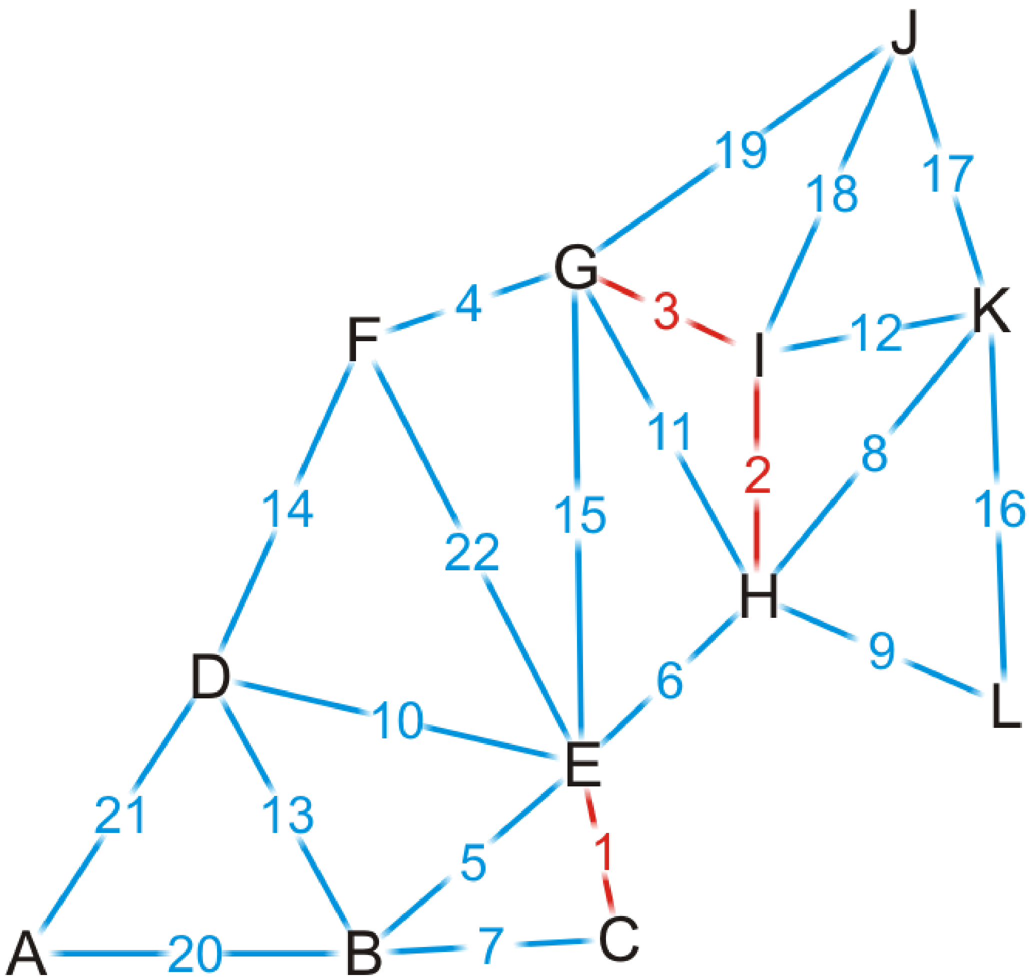
- We add edge {H, I}



{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Kruskal’s Algorithm – Example

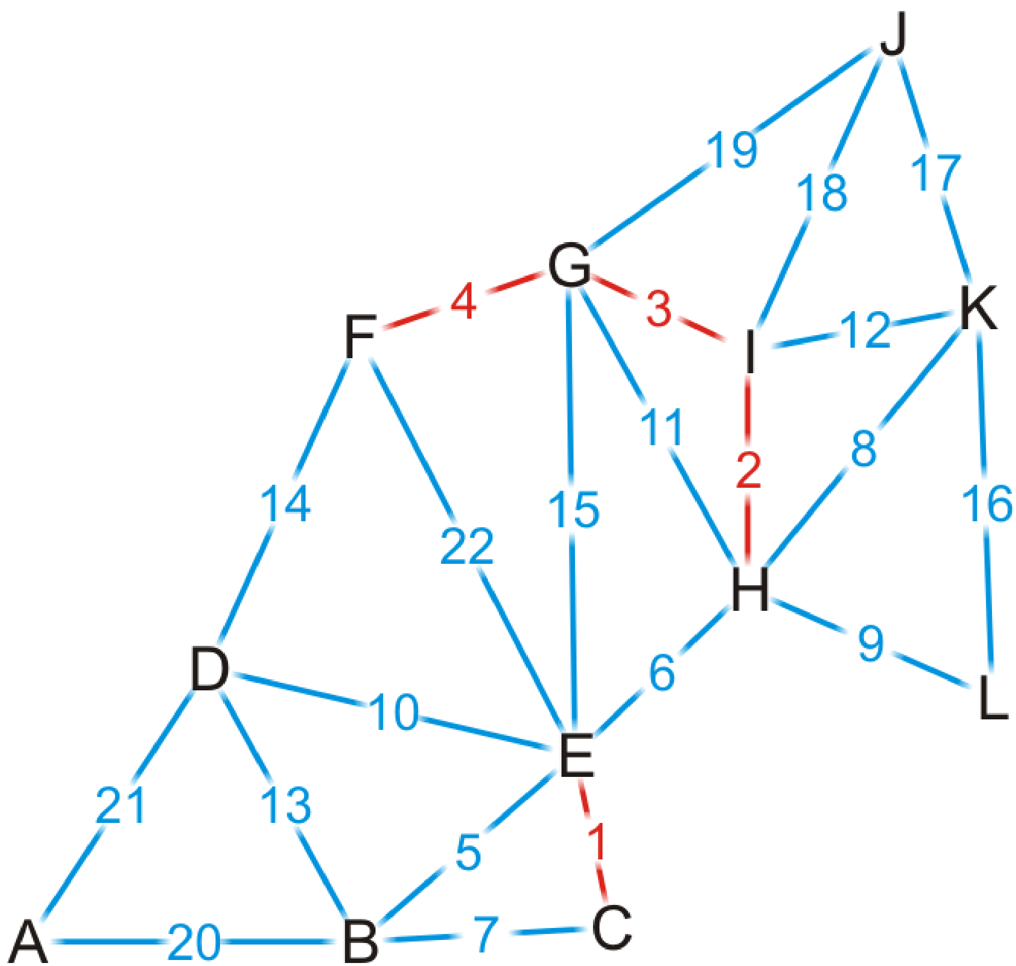
- We add edge {G, I}



{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Kruskal's Algorithm – Example

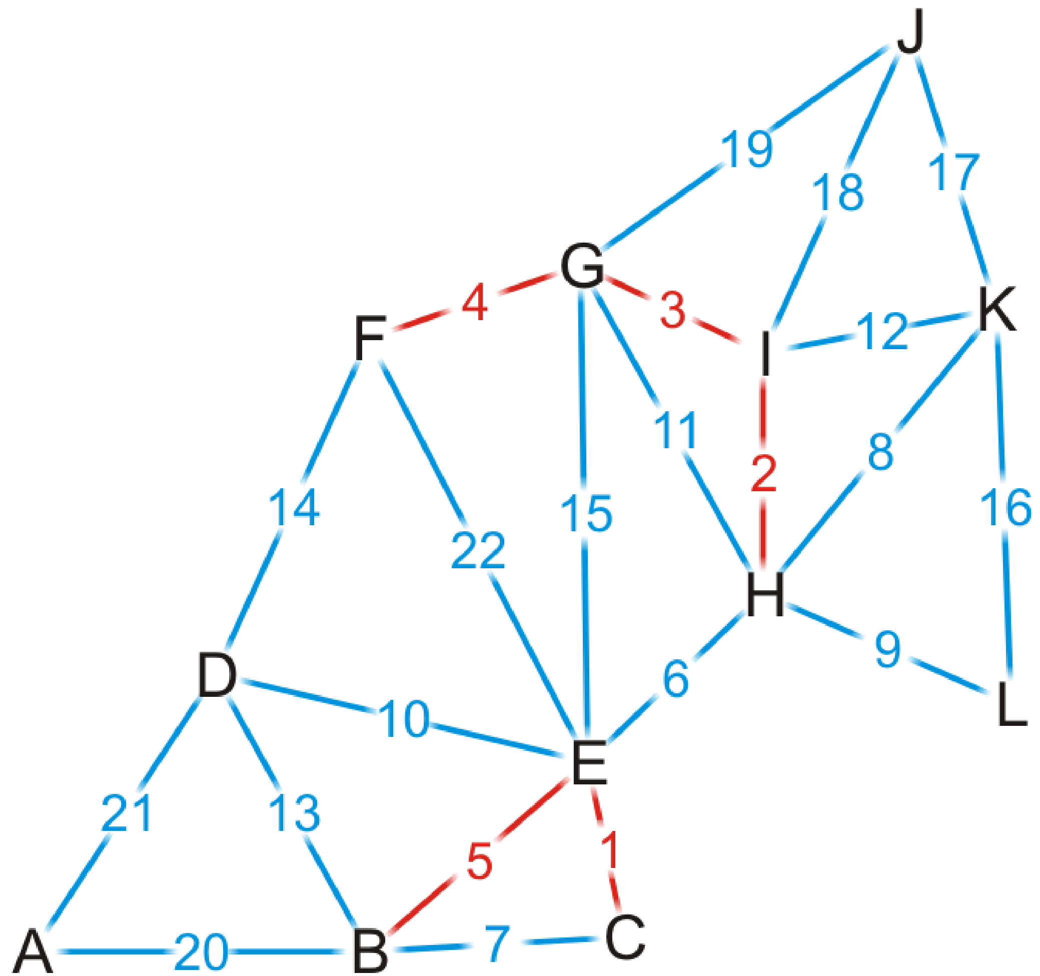
- We add edge {F, G}



{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Kruskal's Algorithm – Example

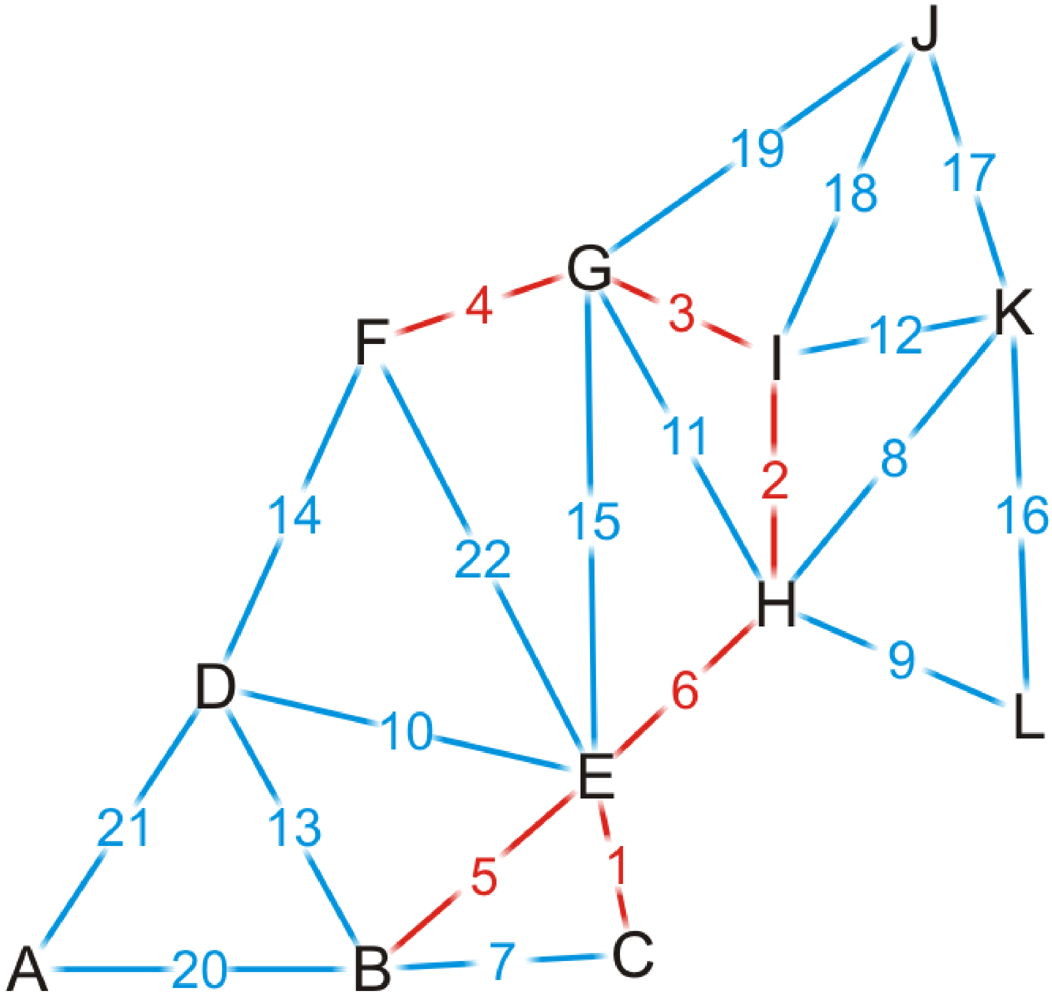
- We add edge {B, E}



{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Kruskal's Algorithm – Example

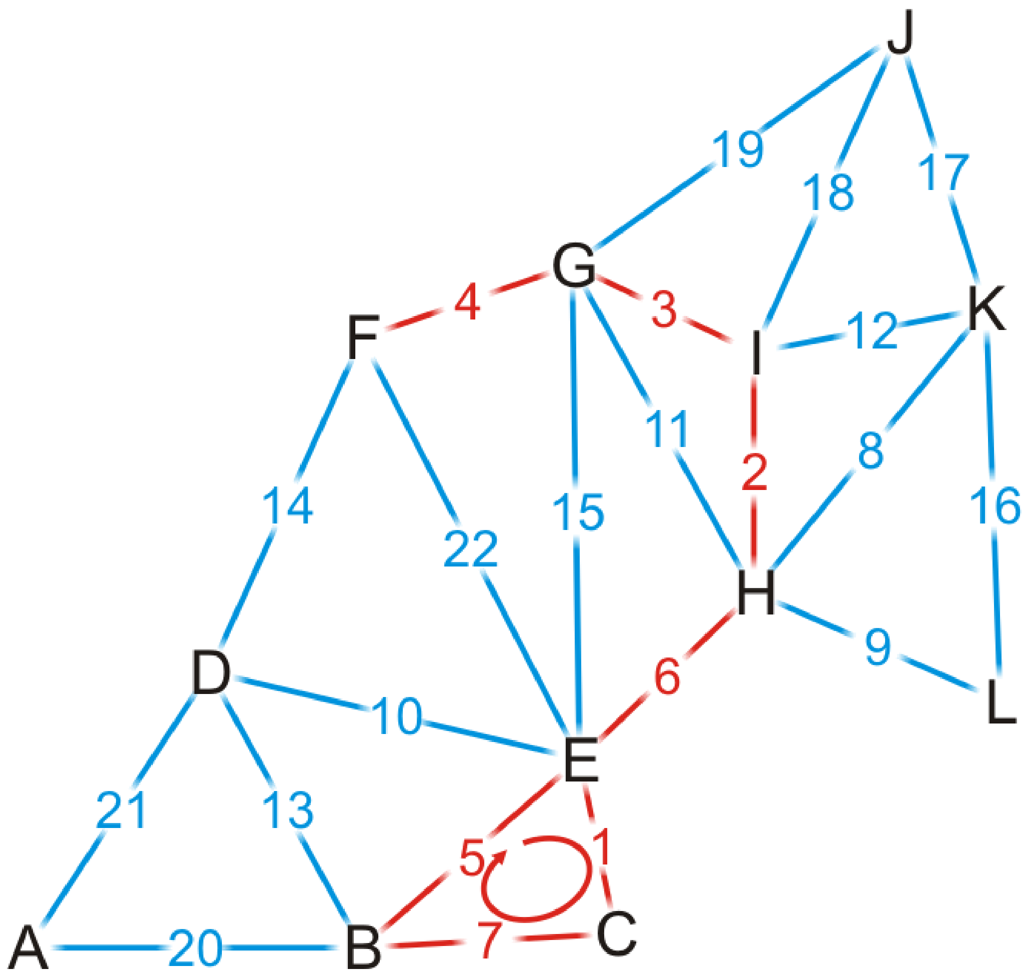
- We add edge {E, H}
  - This coalesces the two spanning sub-trees into one



{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Kruskal's Algorithm – Example

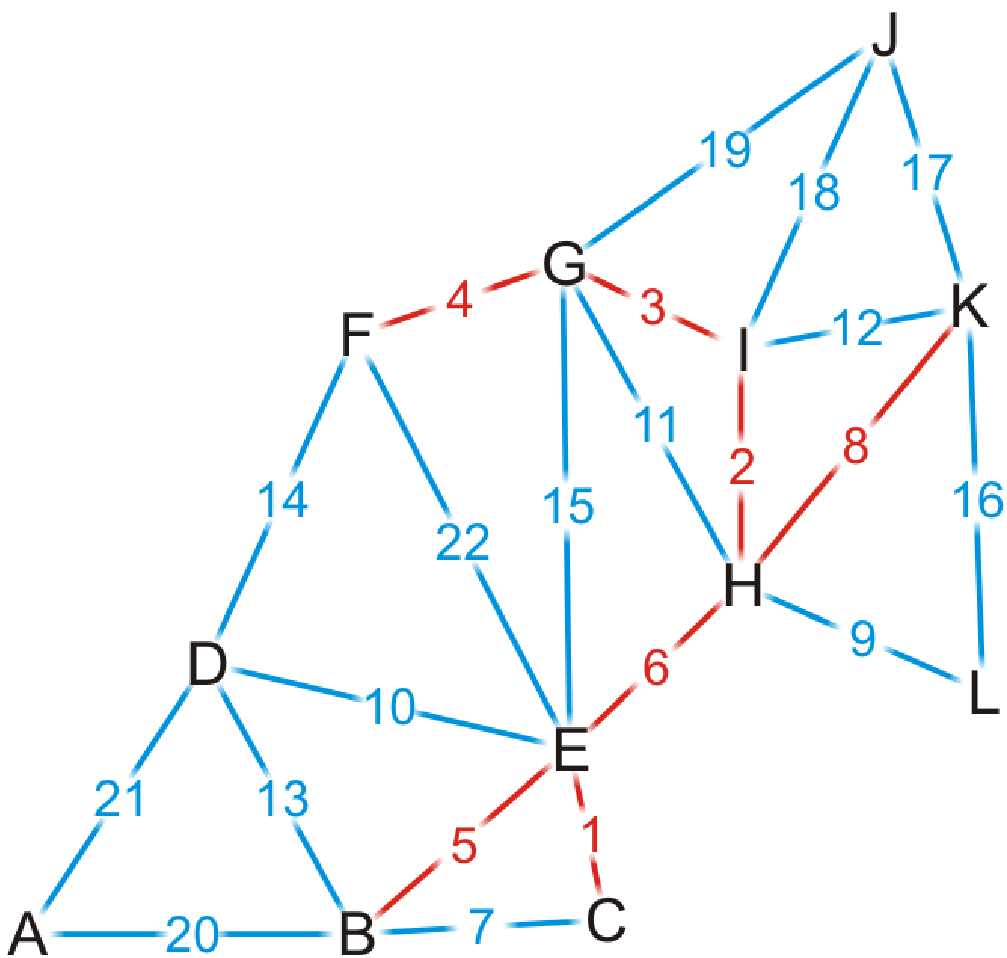
- We try adding {B, C}, but it creates a cycle



{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Kruskal's Algorithm – Example

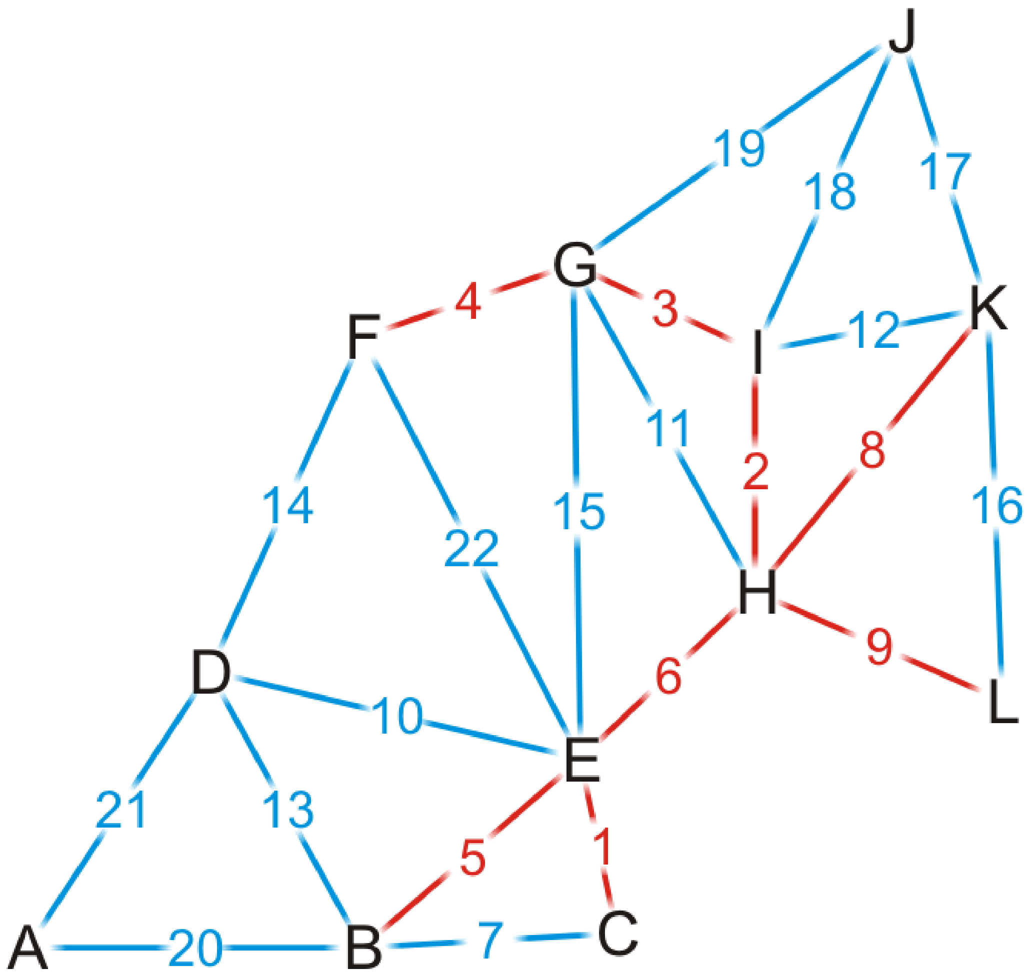
- We add edge {H, K}



{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
→ {H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Kruskal's Algorithm – Example

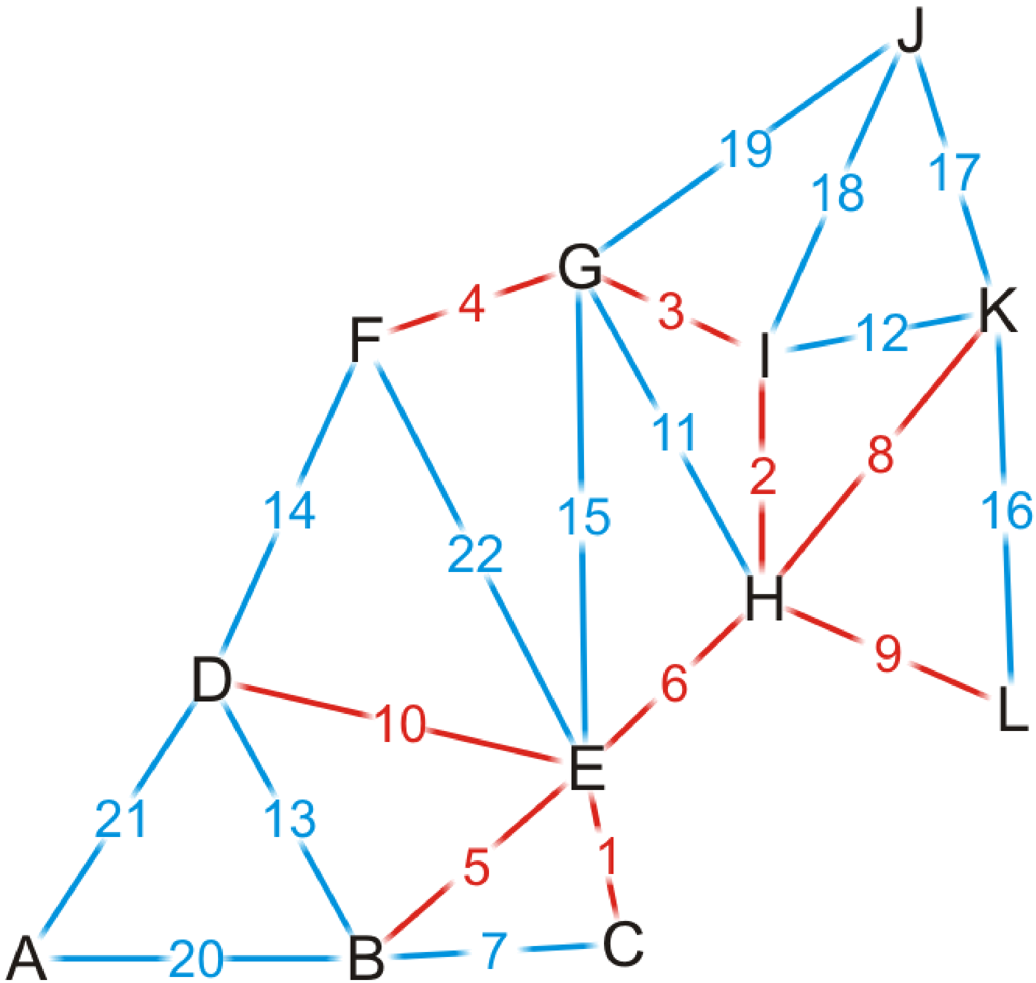
- We add edge {H, L}



{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Kruskal's Algorithm – Example

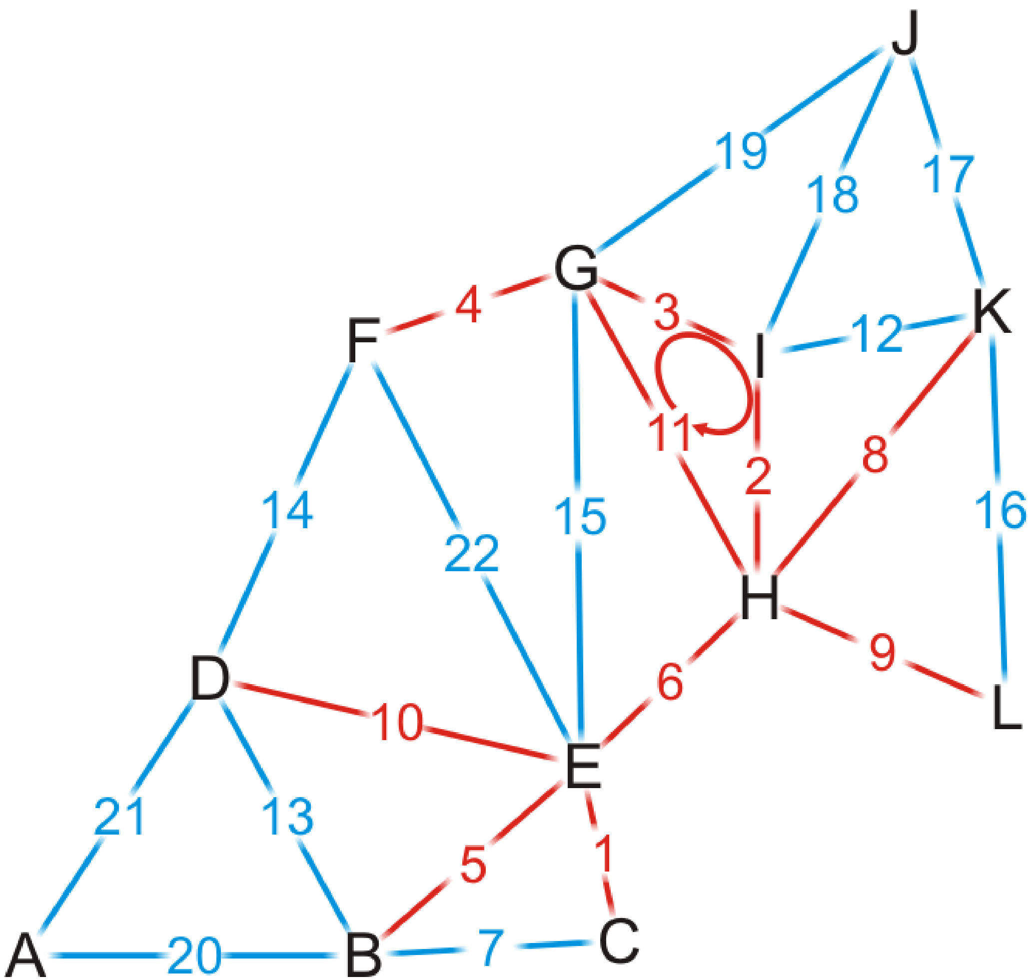
- We add edge {D, E}



{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Kruskal’s Algorithm – Example

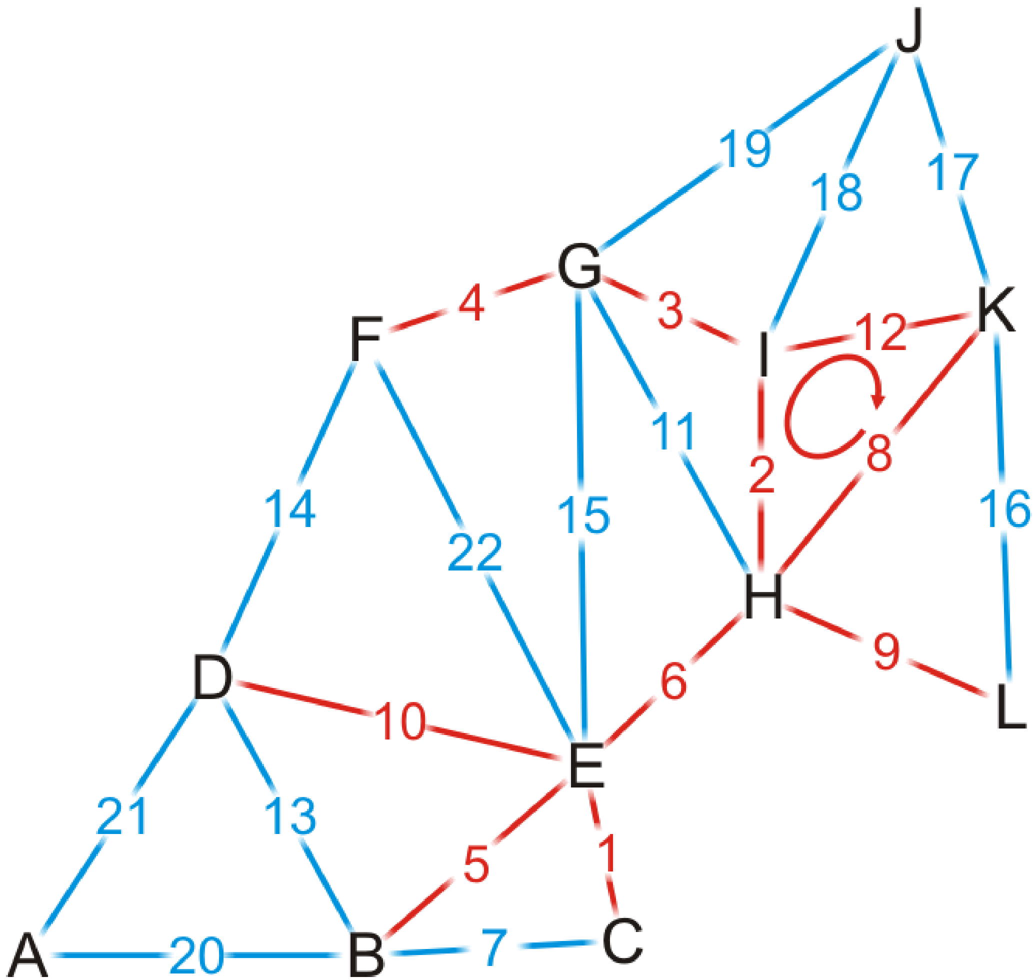
- We try adding {G, H}, but it creates a cycle



{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Kruskal’s Algorithm – Example

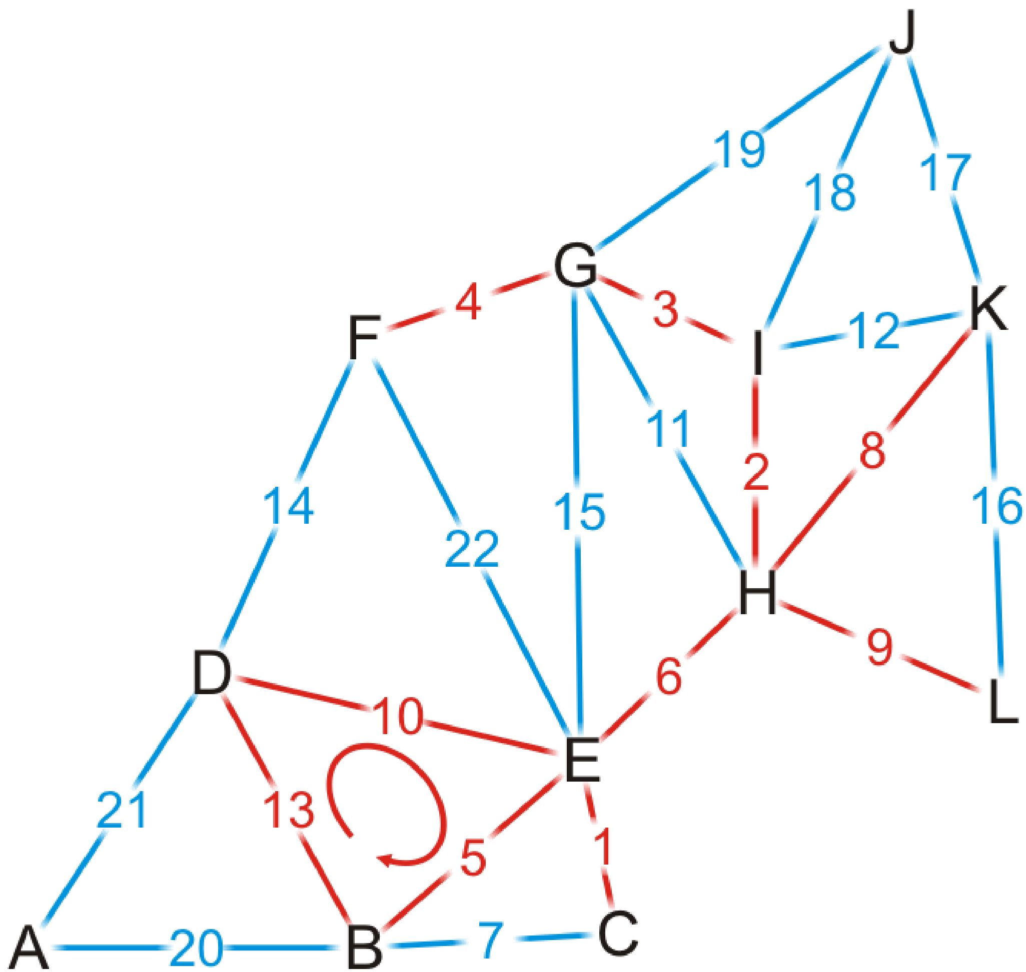
- We try adding {I, K}, but it creates a cycle



{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Kruskal's Algorithm – Example

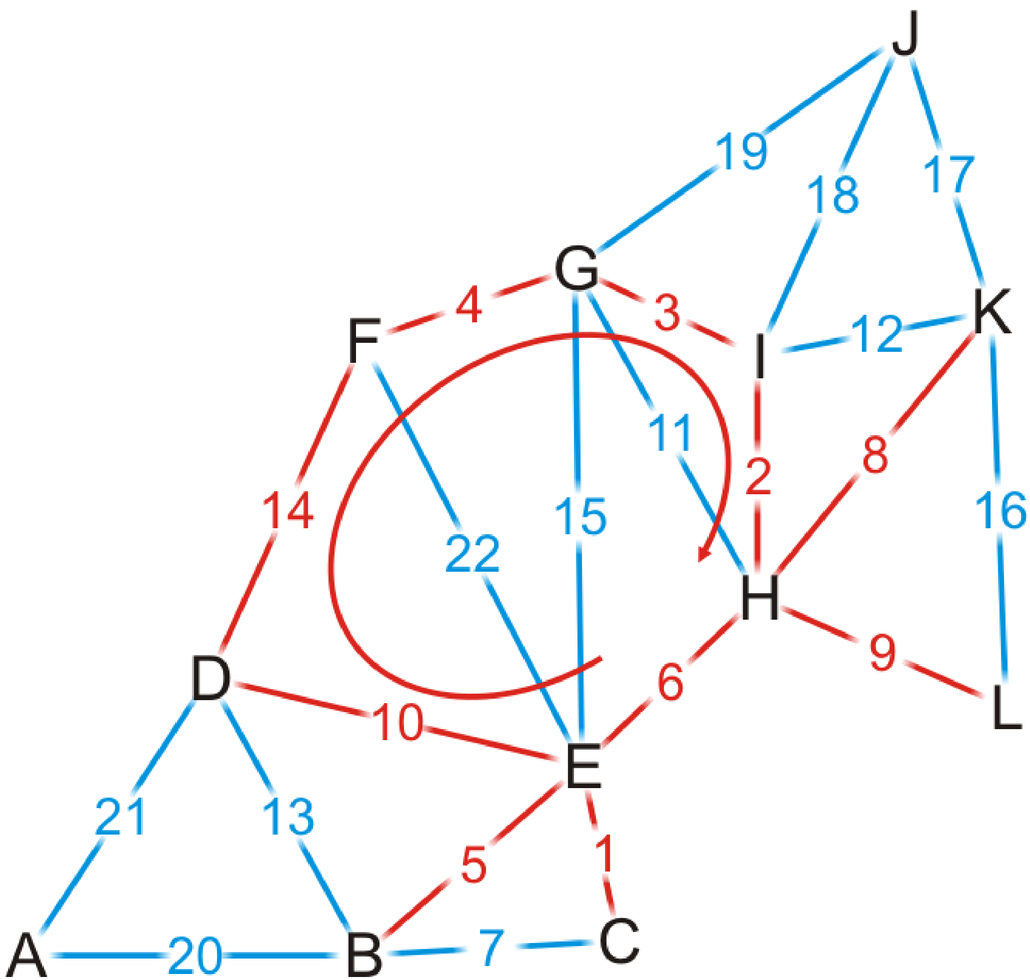
- We try adding {B, D}, but it creates a cycle



{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Kruskal's Algorithm – Example

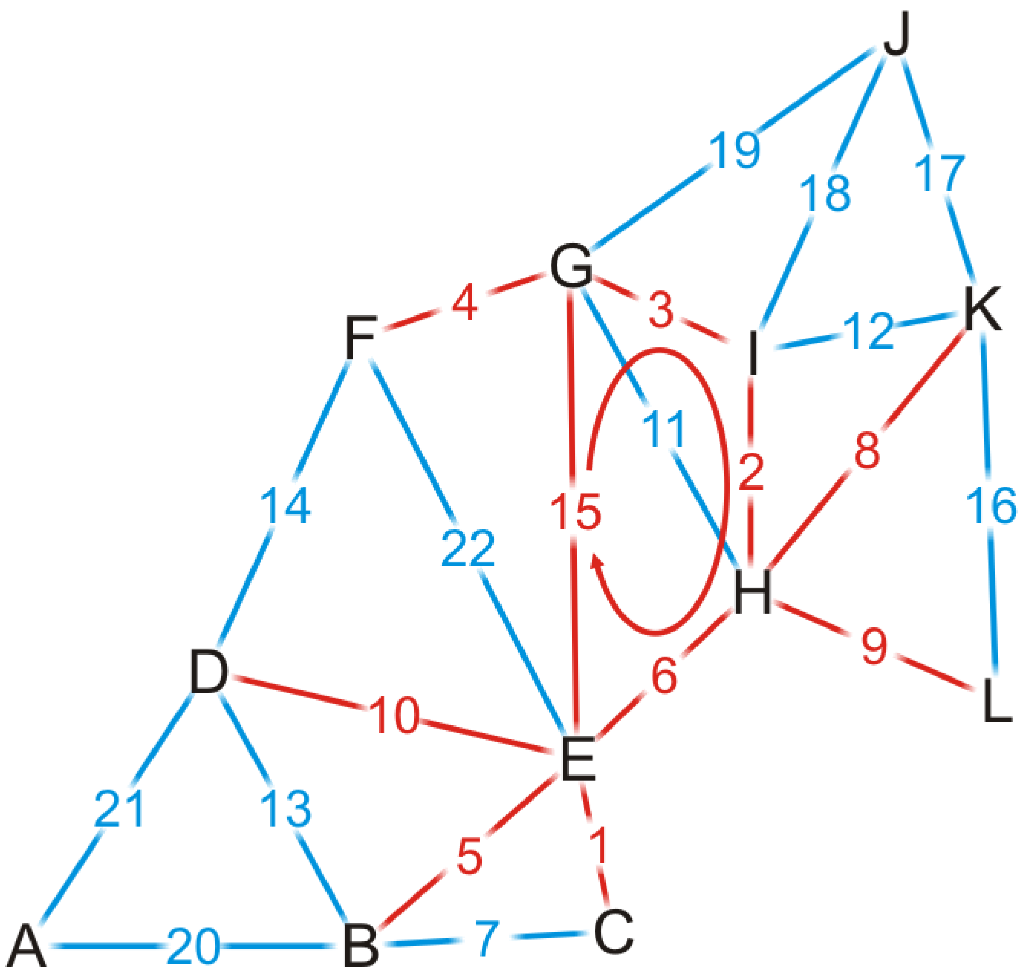
- We try adding {D, F}, but it creates a cycle



{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
→ {D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Kruskal's Algorithm – Example

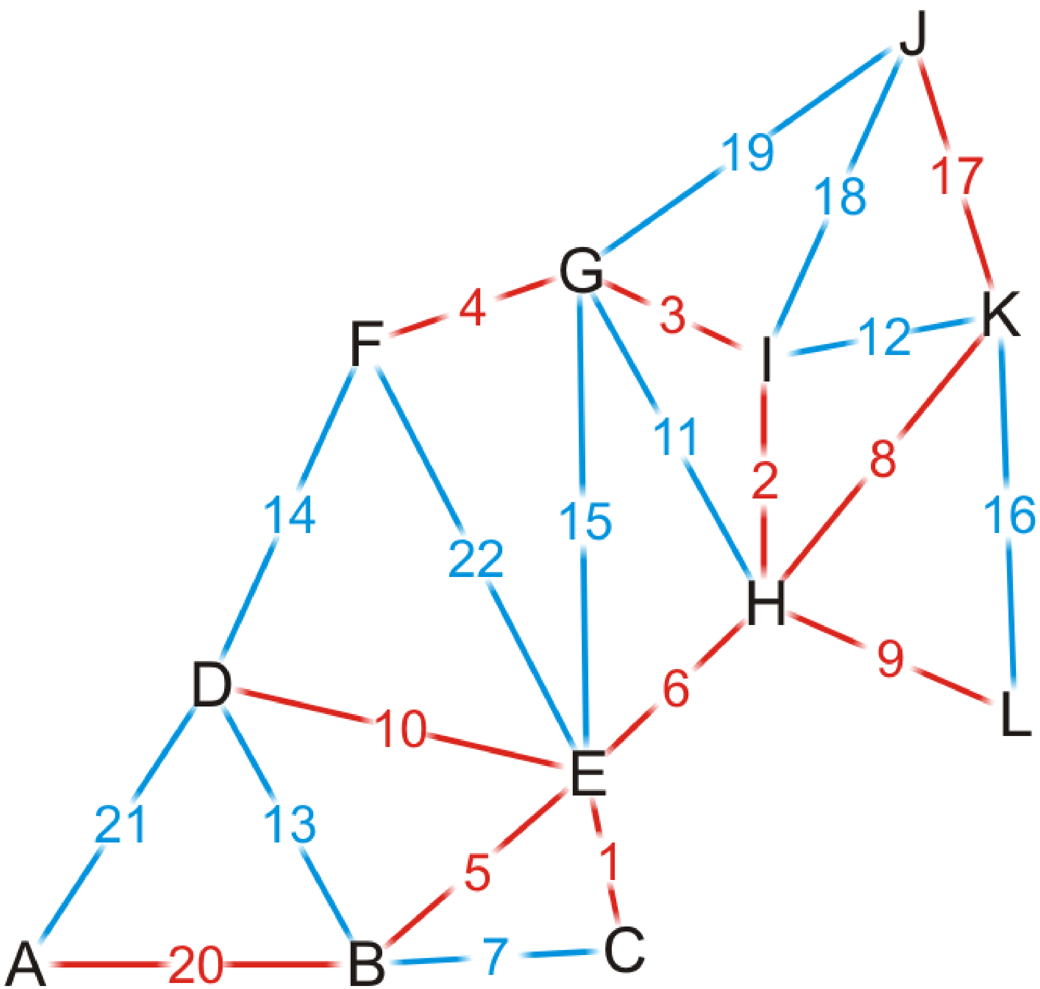
- We try adding {E, G}, but it creates a cycle



{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Kruskal's Algorithm – Example

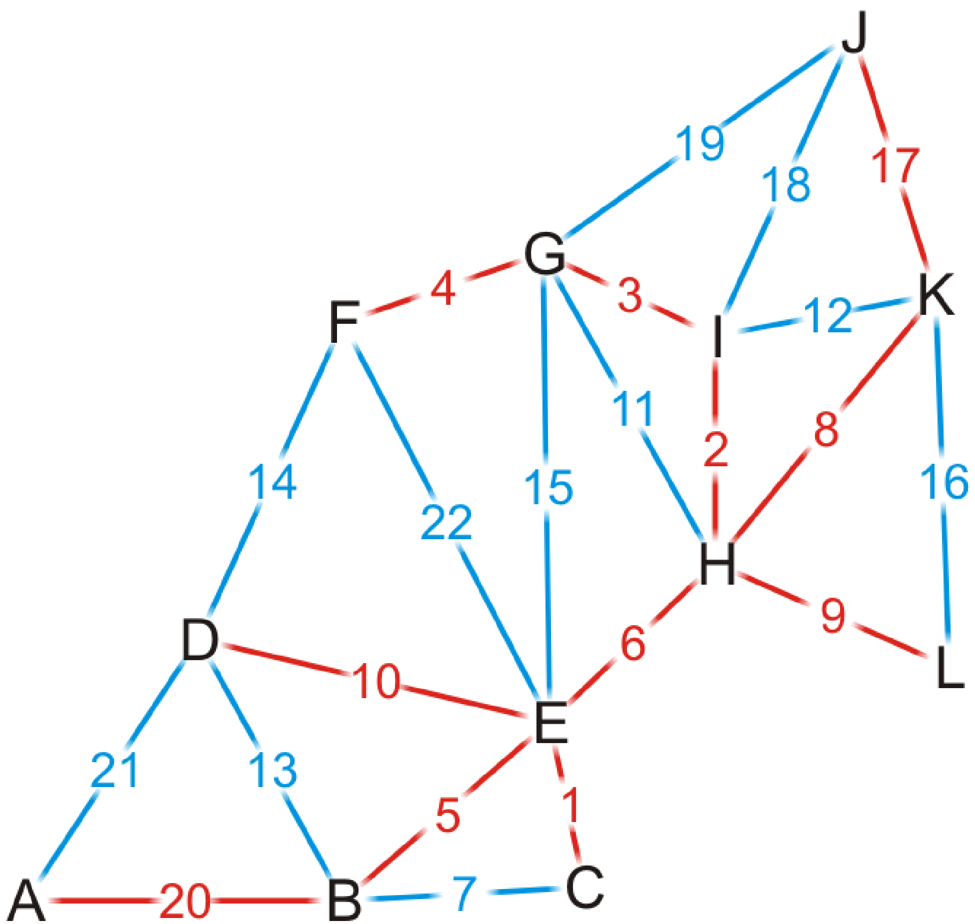
- By observation, we can still add edges {J, K} and {A, B}



{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Kruskal's Algorithm – Example

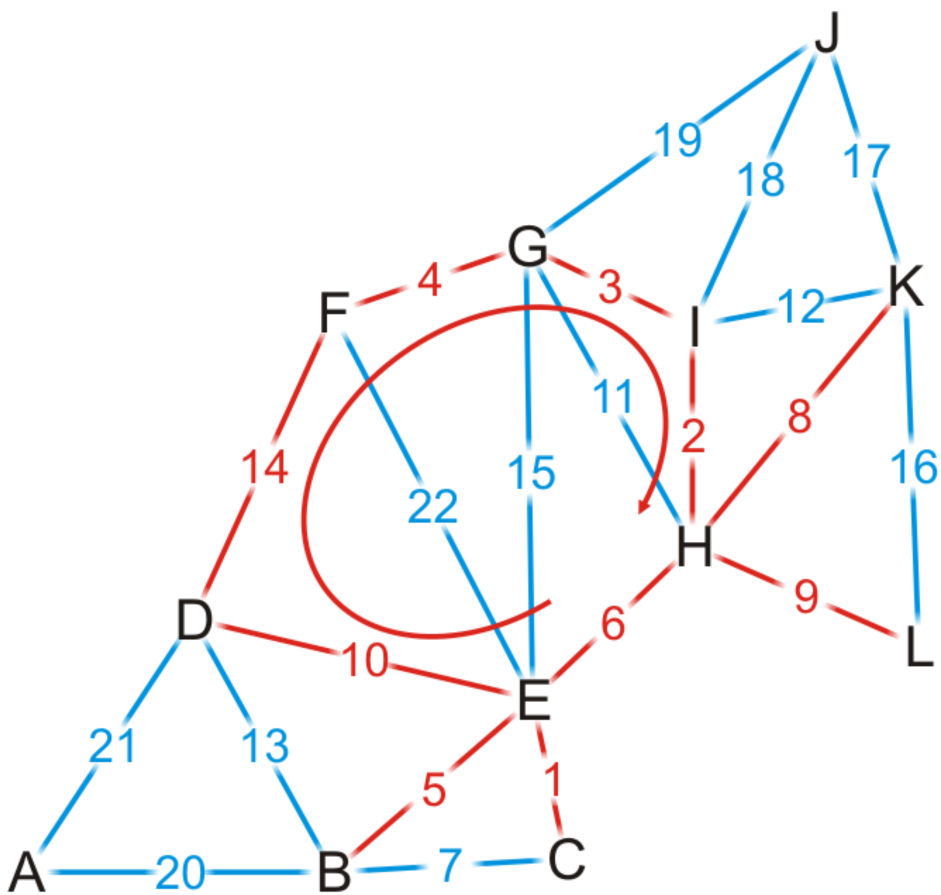
- Having added {A, B}, we now have 11 edges
  - We terminate the loop
  - We have our minimum spanning tree



{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Detecting a Cycle

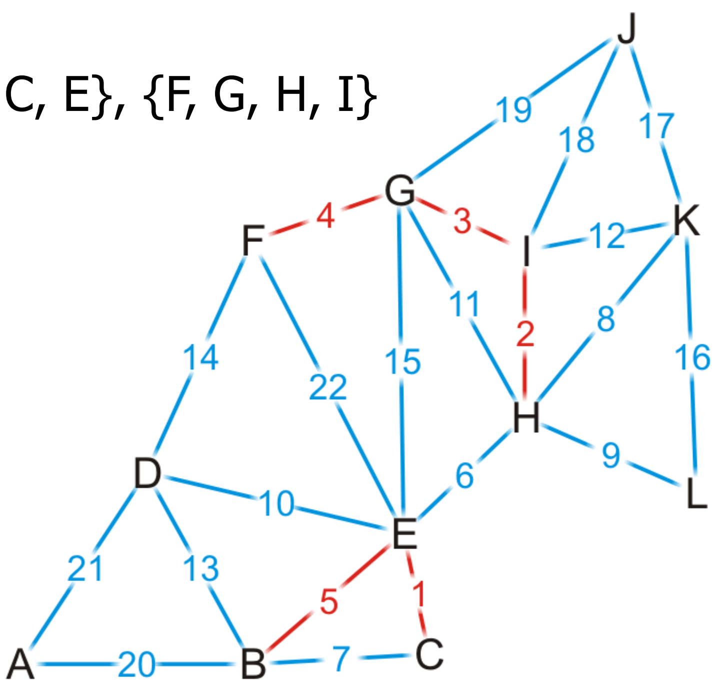
- To determine if a cycle is created, we could perform a traversal
  - A run-time of O(|V|)



Detecting a Cycle – Disjoint Sets

- Consider edges in the same connected sub-graph as forming a set

{B, C, E}, {F, G, H, I}

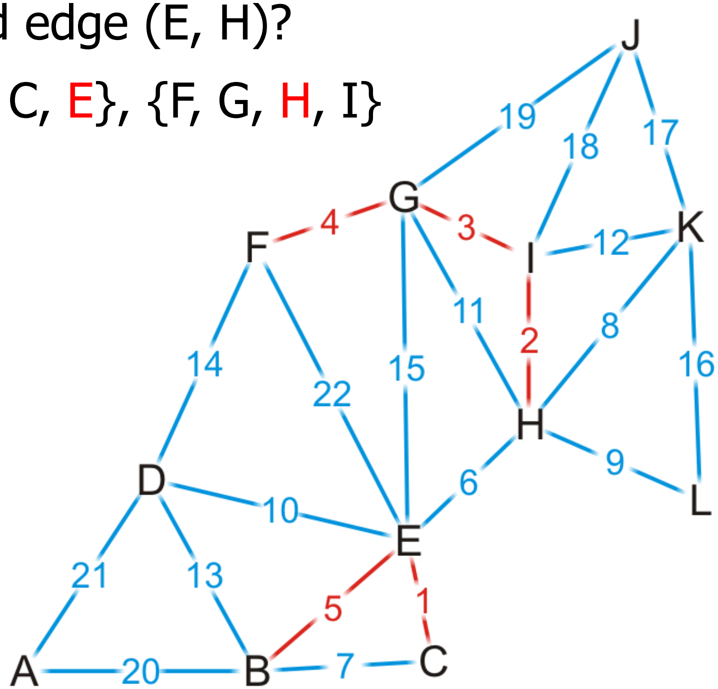


Detecting a Cycle – Disjoint Sets

- Consider edges in the same connected sub-graph as forming a set
- If the vertices of the next edge are in different sets
  - Take the union of the two sets

Add edge (E, H)?

{B, C, E}, {F, G, H, I}

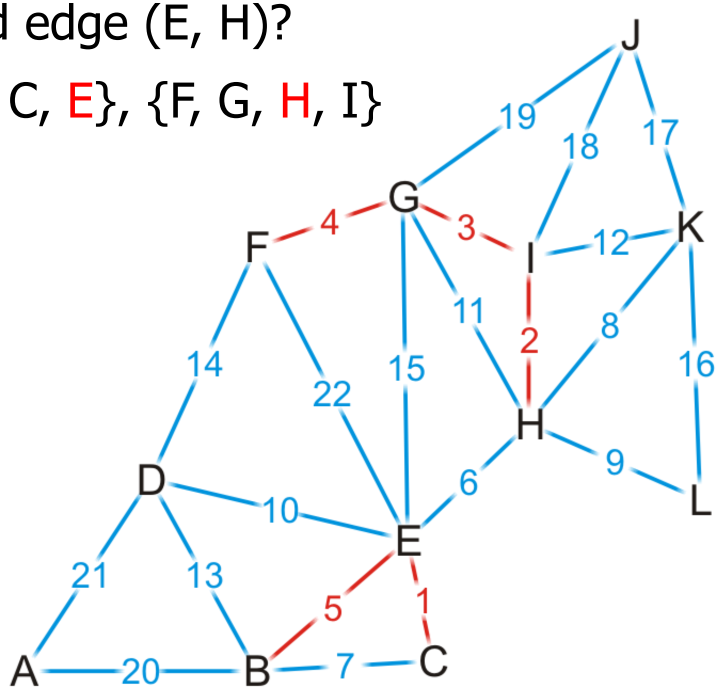


Detecting a Cycle – Disjoint Sets

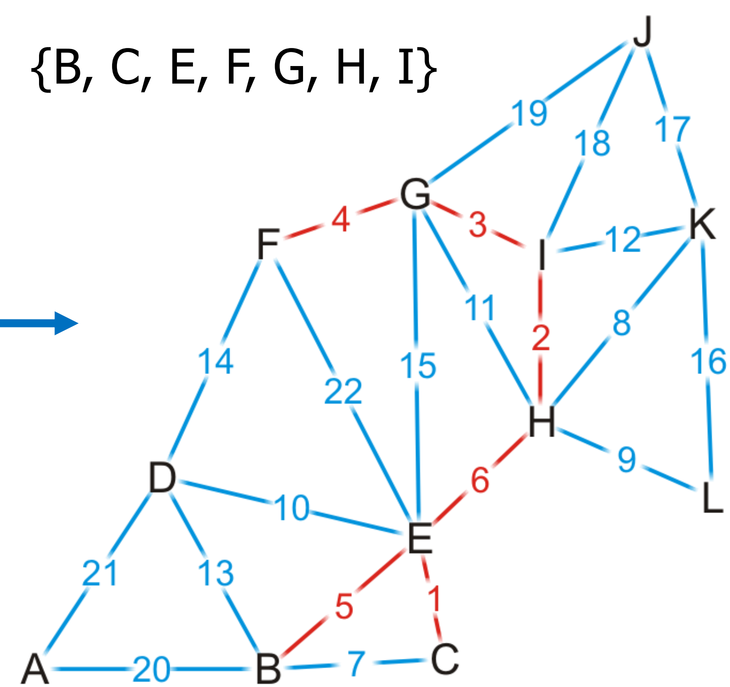
- Consider edges in the same connected sub-graph as forming a set
- If the vertices of the next edge are in different sets
  - Take the union of the two sets

Add edge (E, H)?

{B, C, E}, {F, G, H, I}



{B, C, E, F, G, H, I}

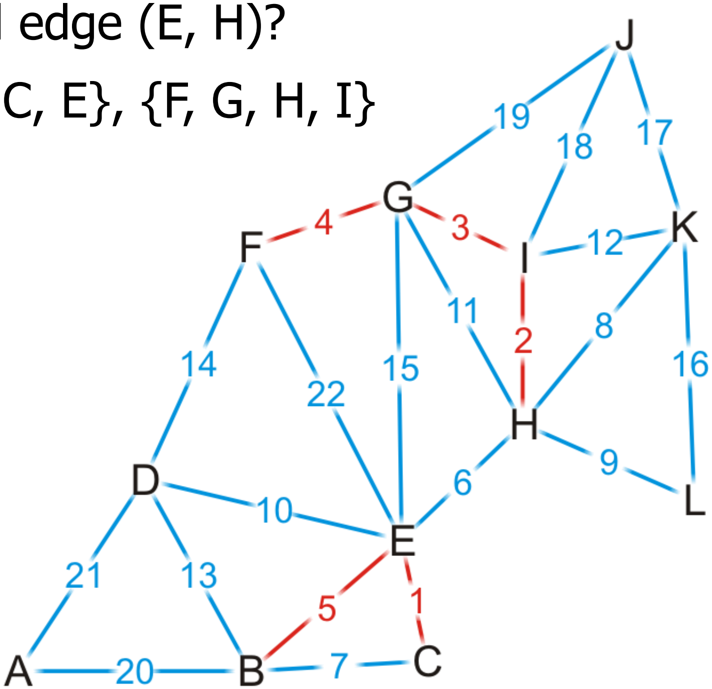


Detecting a Cycle – Disjoint Sets

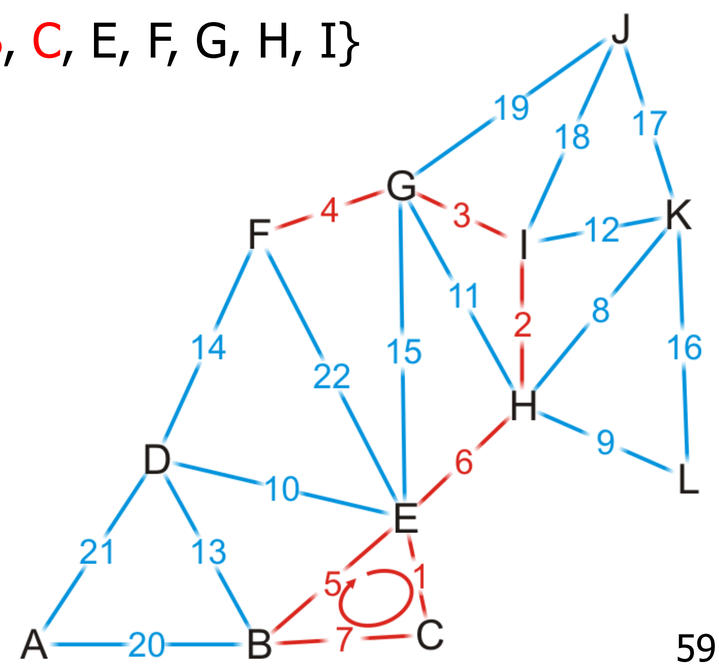
- Consider edges in the same connected sub-graph as forming a set
- If the vertices of the next edge are in different sets
  - Take the union of the two sets
- Do not add an edge if both vertices are in the same set

Add edge (E, H)?

{B, C, E}, {F, G, H, I}



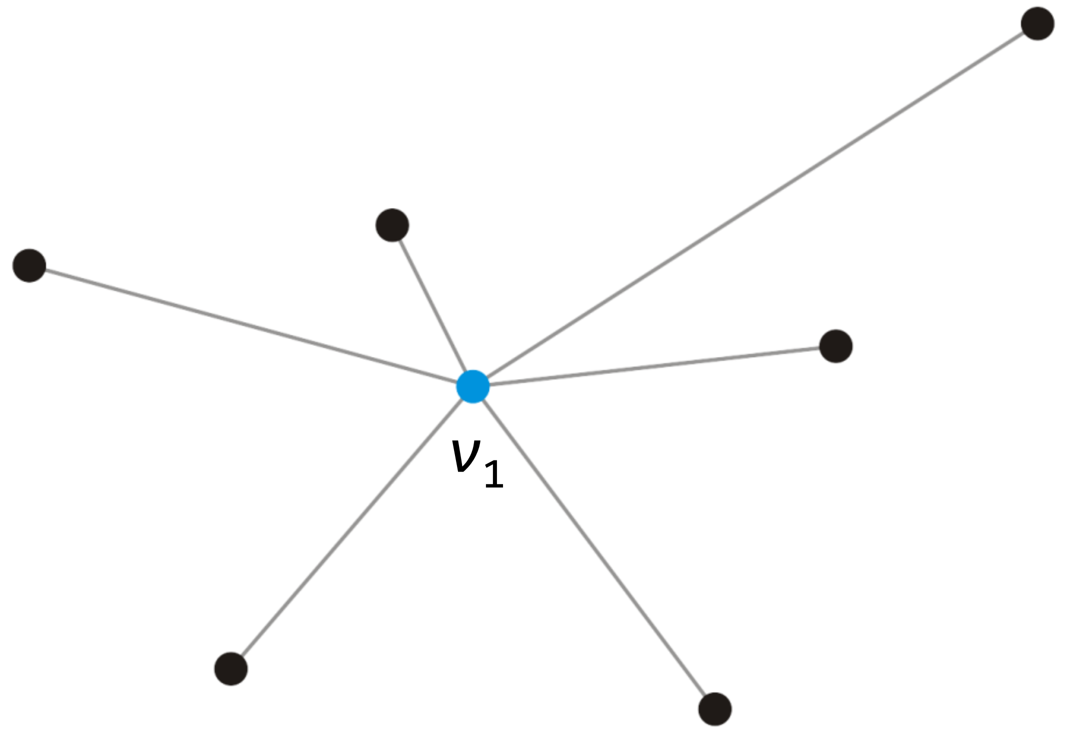
{B, C, E, F, G, H, I}



Prim's Algorithm

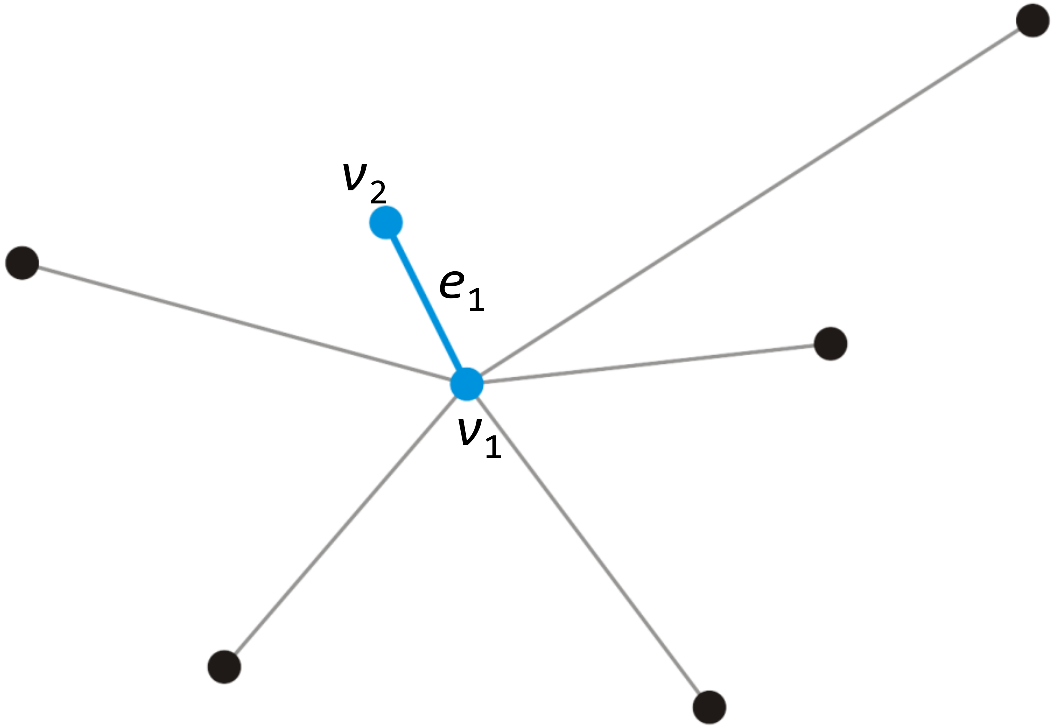
Idea

- Suppose we take a vertex v_1
  - It forms a minimum spanning tree on one vertex



Idea

- Add that adjacent vertex v_2 that has a connecting edge e_1 of minimum weight
  - This forms a minimum spanning tree on our two vertices
  - e_1 must be in any minimum spanning tree containing the vertices v_1 and v_2



Prim’s Algorithm

- Start with an arbitrary vertex to form a minimum spanning tree on one vertex
- At each step, add a vertex v not yet in the minimum spanning tree
  - Though an edge with least weight that connects v to the existing minimum spanning sub-tree
- Continue until we have $n - 1$ edges and n vertices

Prim’s Algorithm – Pseudocode

```
MST-Prim(G, w, r) { // w is the weight matrix of edges, r is root
    Q = V[G];  // Insert graph vertices to a Queue
    for each u ∈ Q  // Set distance of all vertices as ∞
        key[u] = ∞;
    key[r] = 0;  // Distance of root is set to 0
    p[r] = NULL; // Parent of root is NULL
    while (Q not empty) {
        u = ExtractMin(Q); // Get the vertex u with min key[u]
        for each v ∈ Adj[u] {  // Adj is the adjacency list
            if (v ∈ Q and w(u,v) < key[v]) {
                p[v] = u;
                key[v] = w(u,v); // weight of an edge (u,v)
            }
        }
    }
}
```

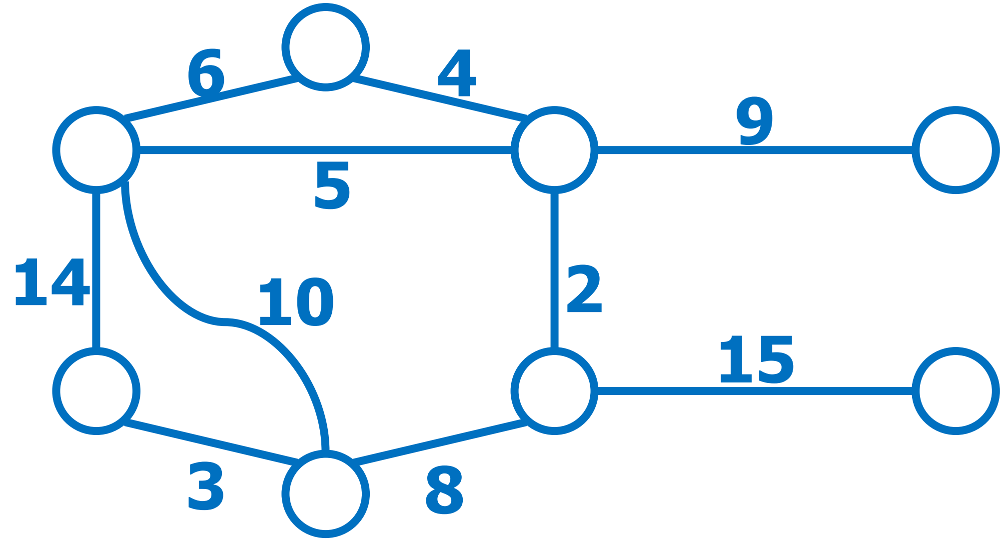
Prim's Algorithm – Data Structure

- Associate with each vertex two items of data
  - The minimum distance to the partially constructed tree
    - For a given vertex `v`, `key[v]` represent minimum distance
  - Pointer to the vertex that will form the parent node in resulting tree
    - For a given vertex v, `p[v]` represent parent node

- Initialization
  - Set the distance of all vertices as ∞, e.g., for all $u \in G$, `key[u]=` ∞
  - Set all vertices to being unvisited
    - Add vertices to the Queue
  - Select a root node and set its distance as 0, i.e., `key[r] = 0`
  - Set the parent pointer of root to NULL, i.e., `p[r] = NULL`

Prim's Algorithm – Example

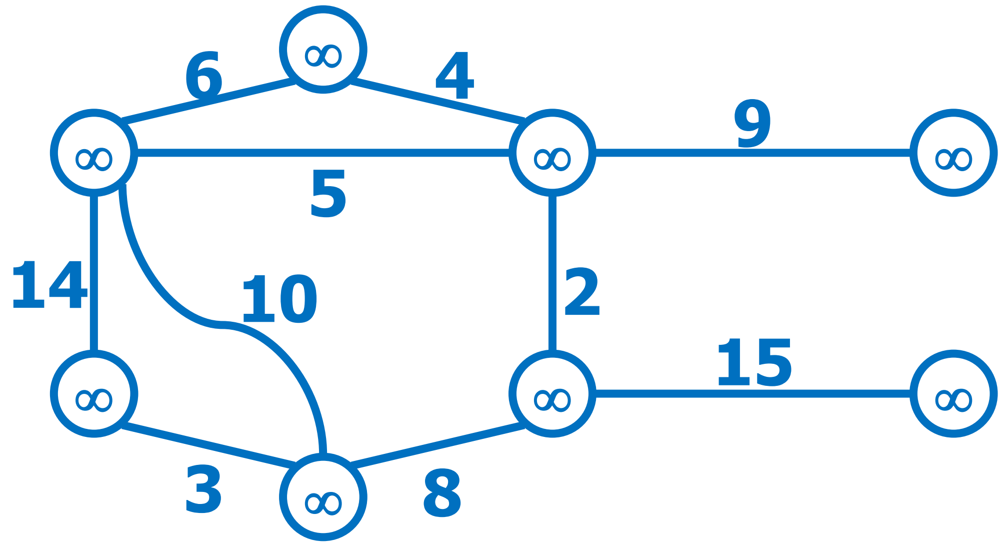
```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```



Run on example graph

Prim's Algorithm – Example

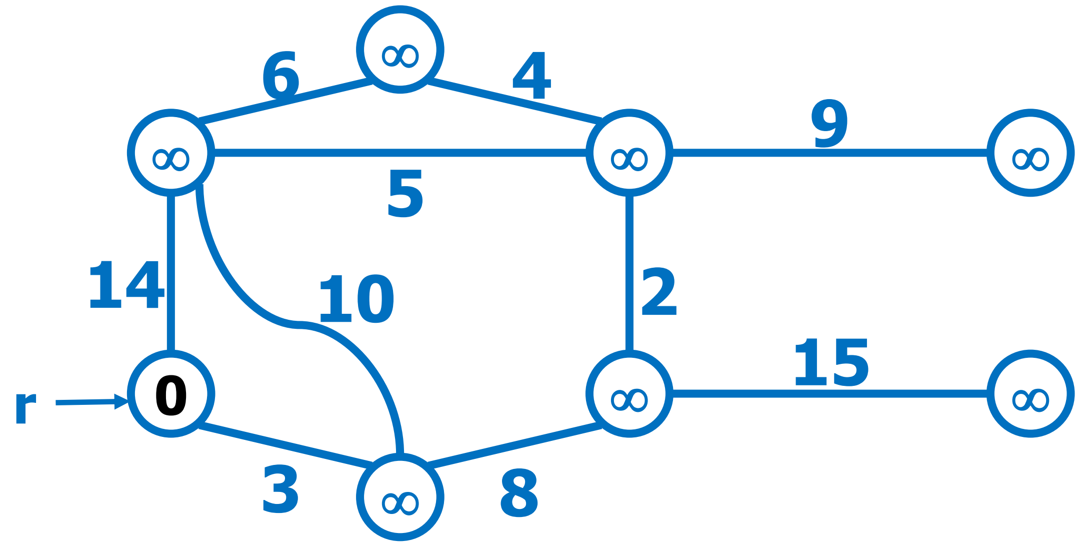
```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```



Run on example graph

Prim's Algorithm – Example

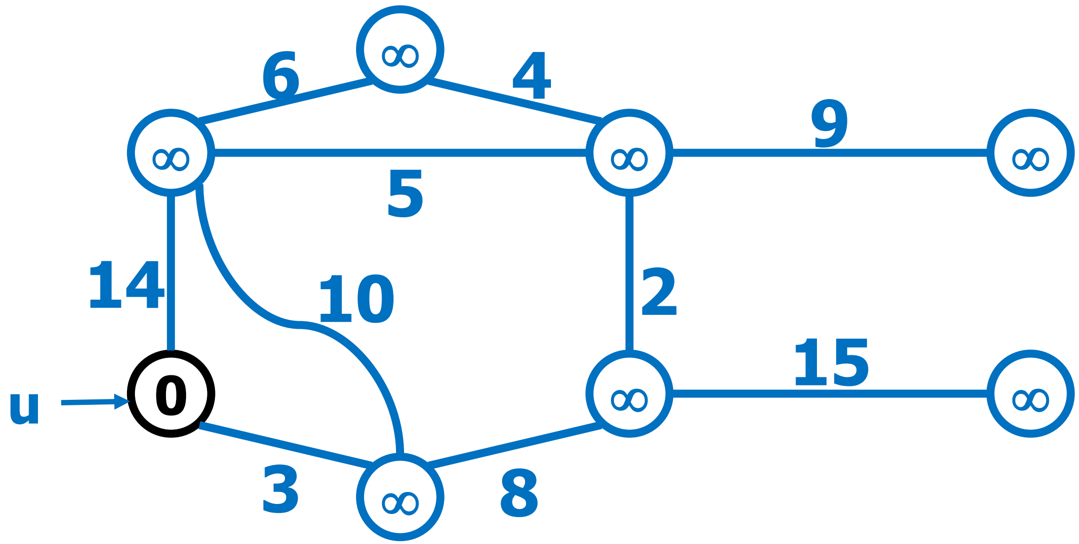
```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```



Pick a start vertex r

Prim's Algorithm – Example

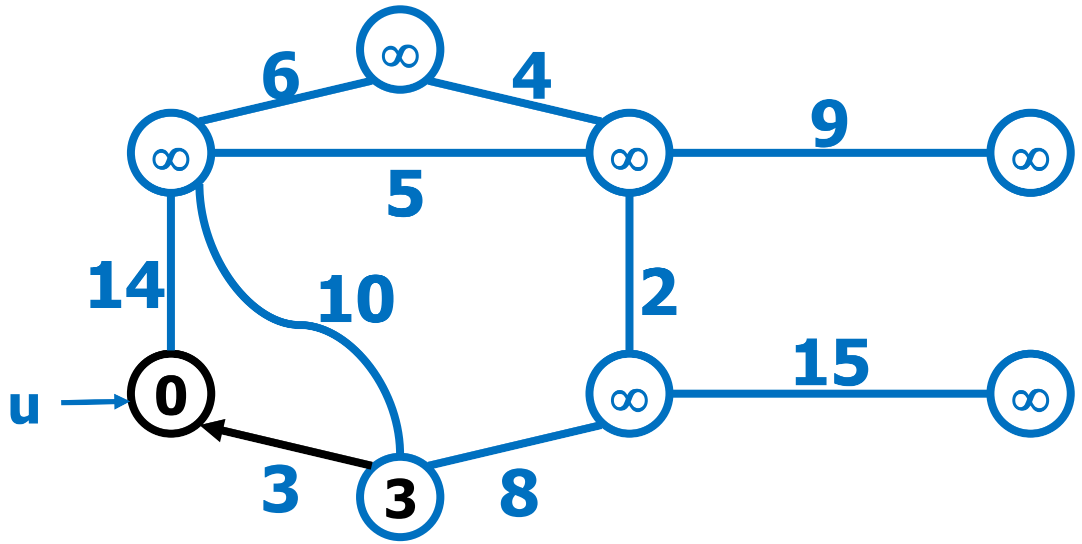
```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```



Black vertices have been removed from Q

Prim's Algorithm – Example

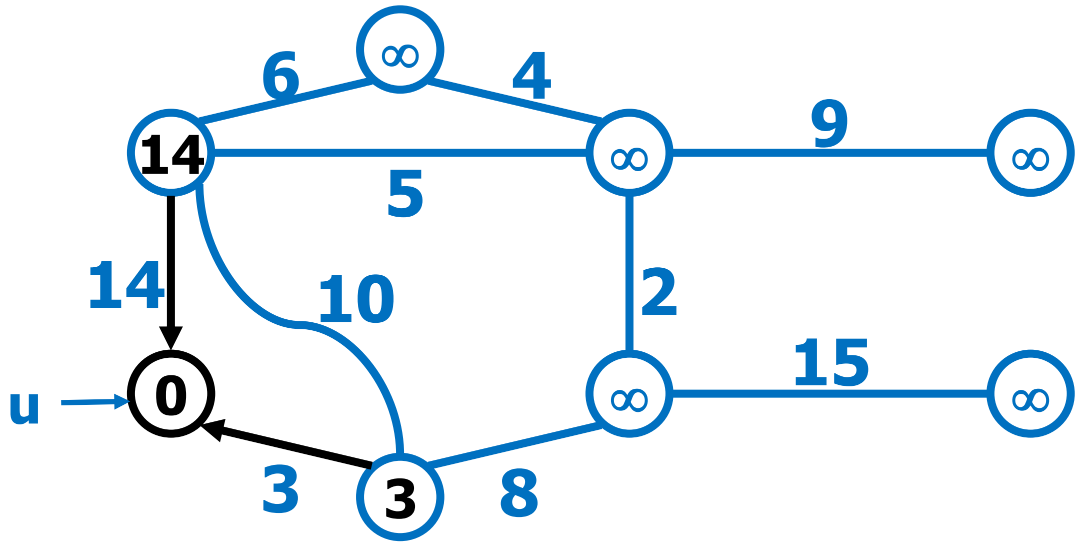
```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```



Black arrows indicate parent pointers

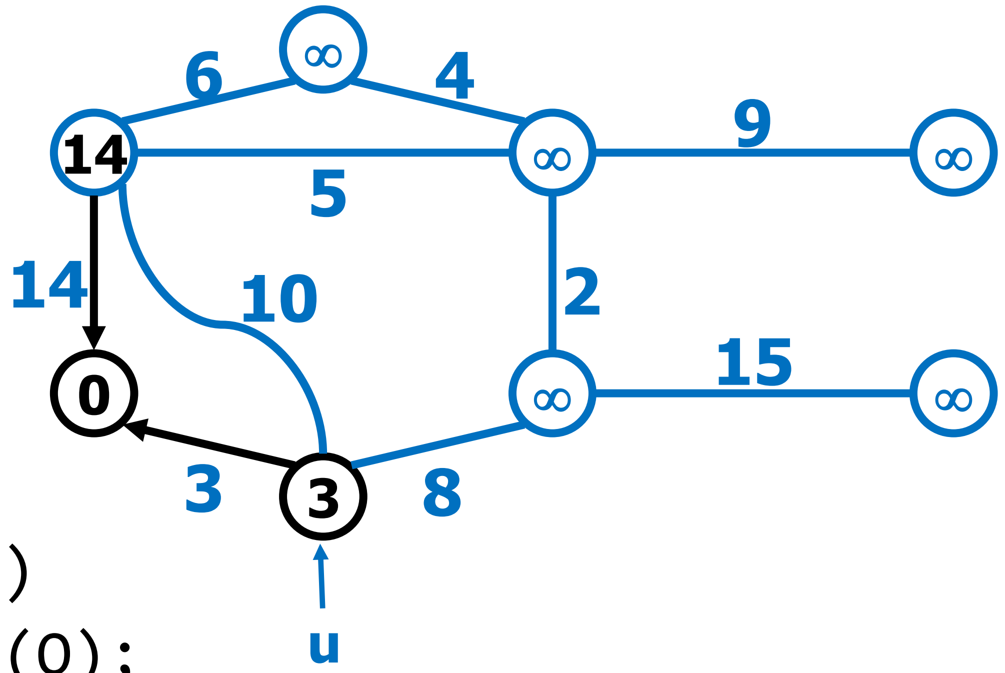
Prim’s Algorithm – Example

```
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);
```



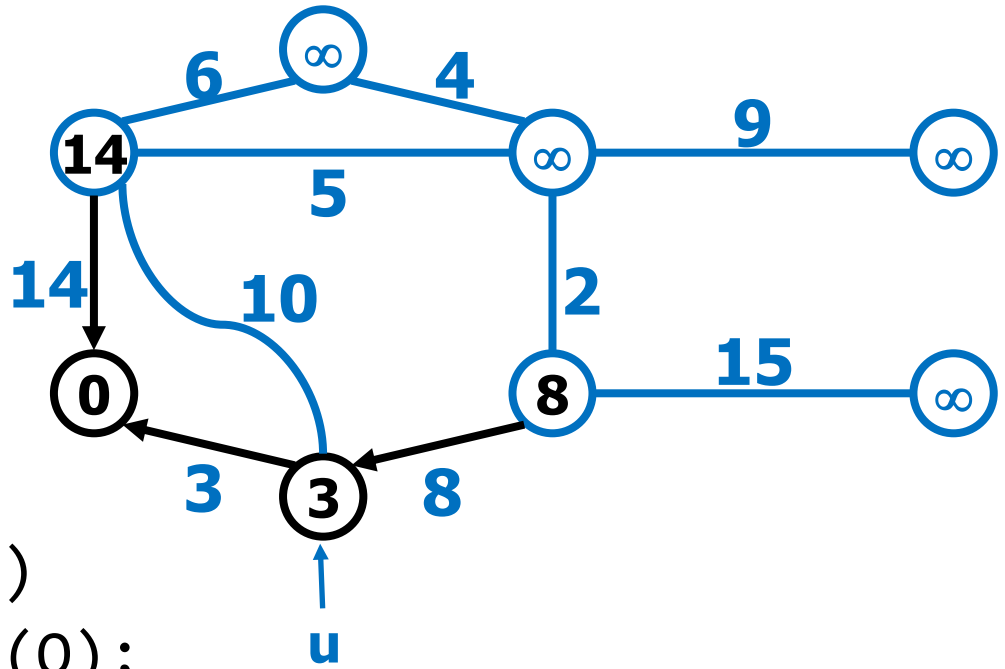
Prim’s Algorithm – Example

```
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);
```



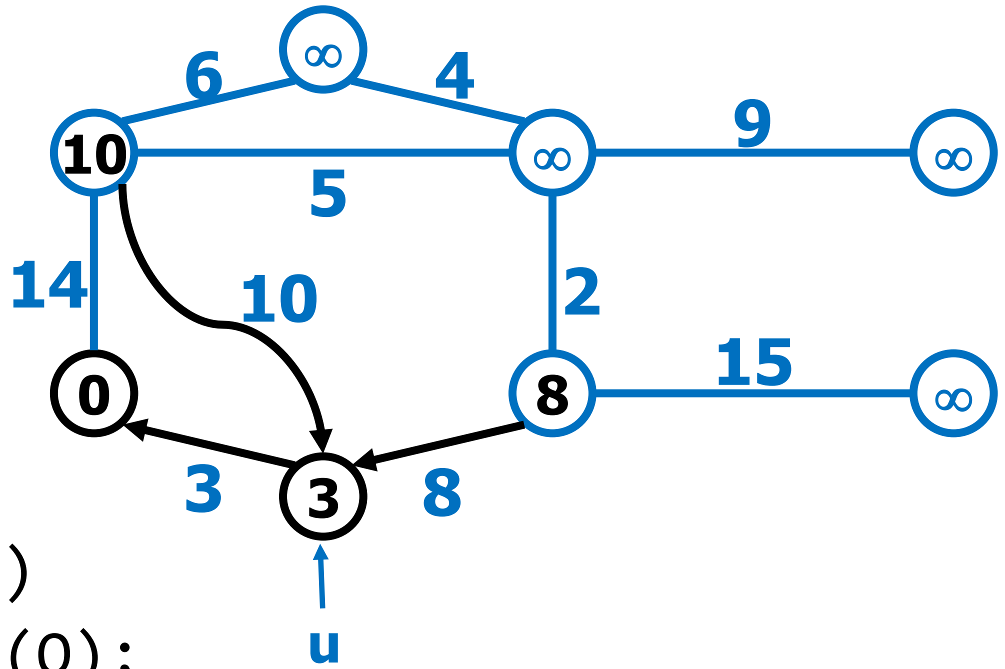
Prim's Algorithm – Example

```
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);
```



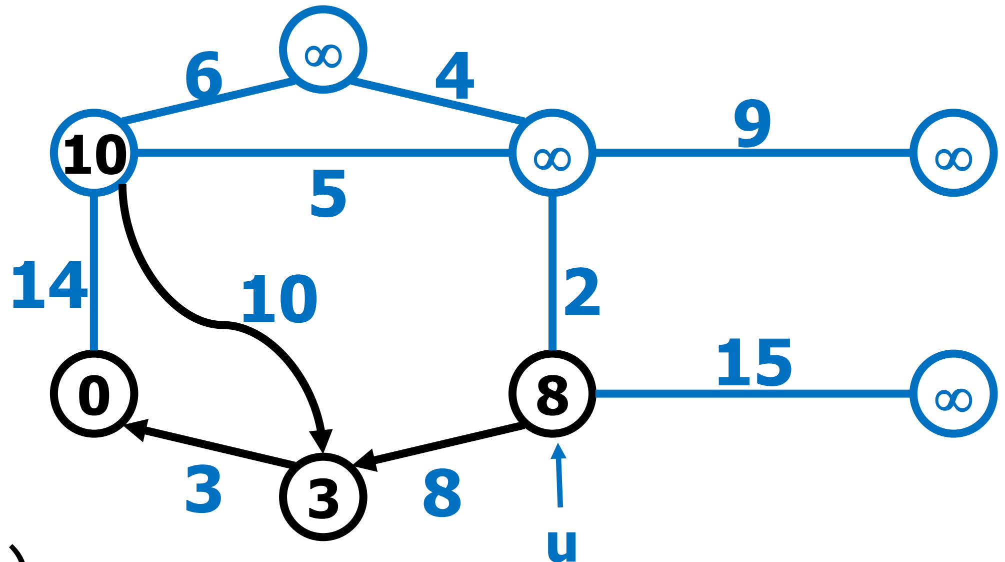
Prim's Algorithm – Example

```
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);
```



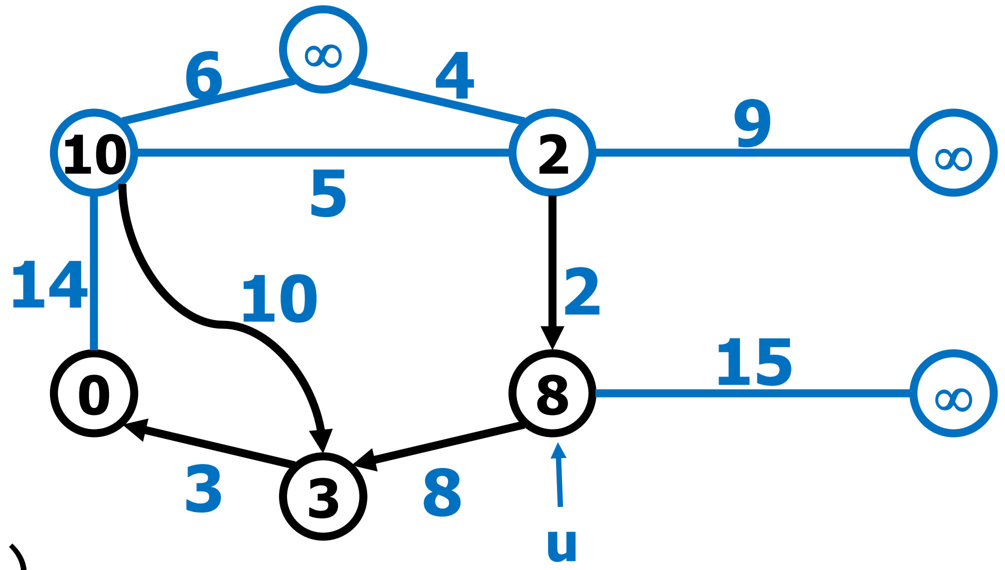
Prim’s Algorithm – Example

```
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);
```



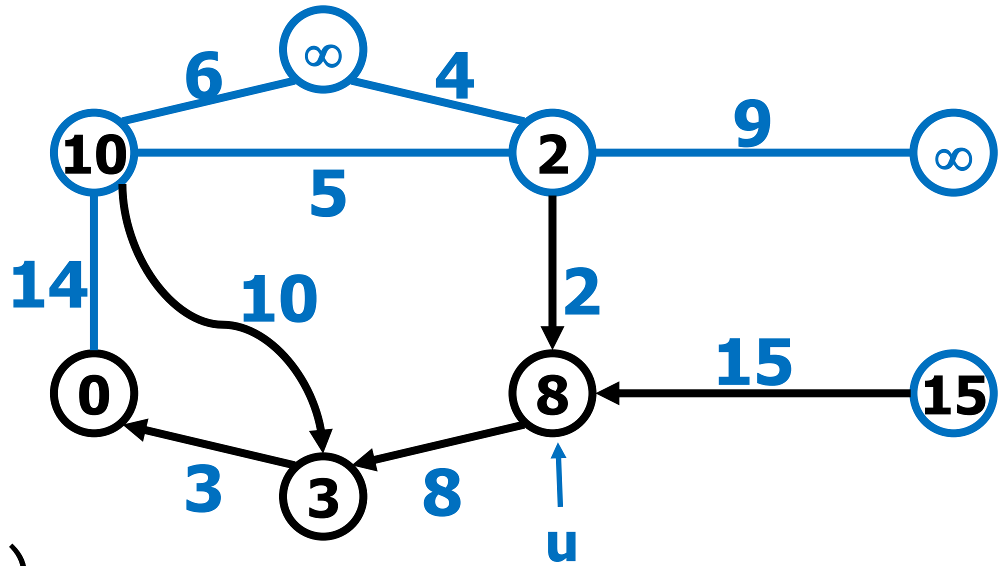
Prim's Algorithm – Example

```
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);
```



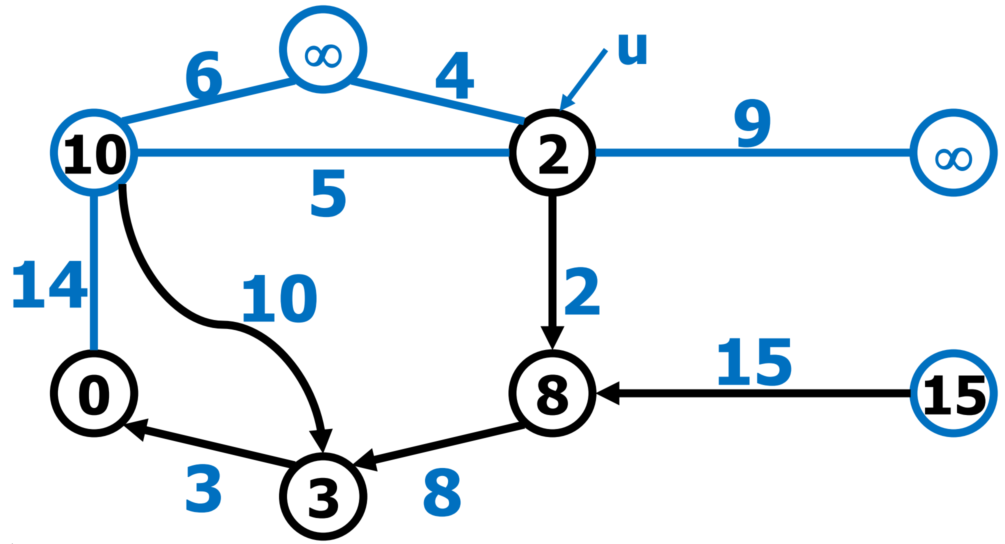
Prim's Algorithm – Example

```
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);
```



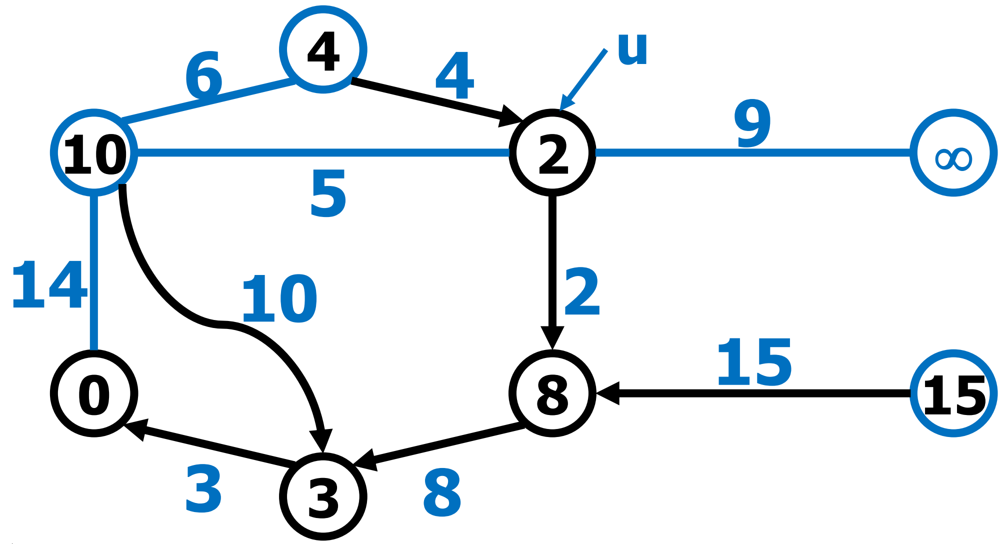
Prim's Algorithm – Example

```
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);
```



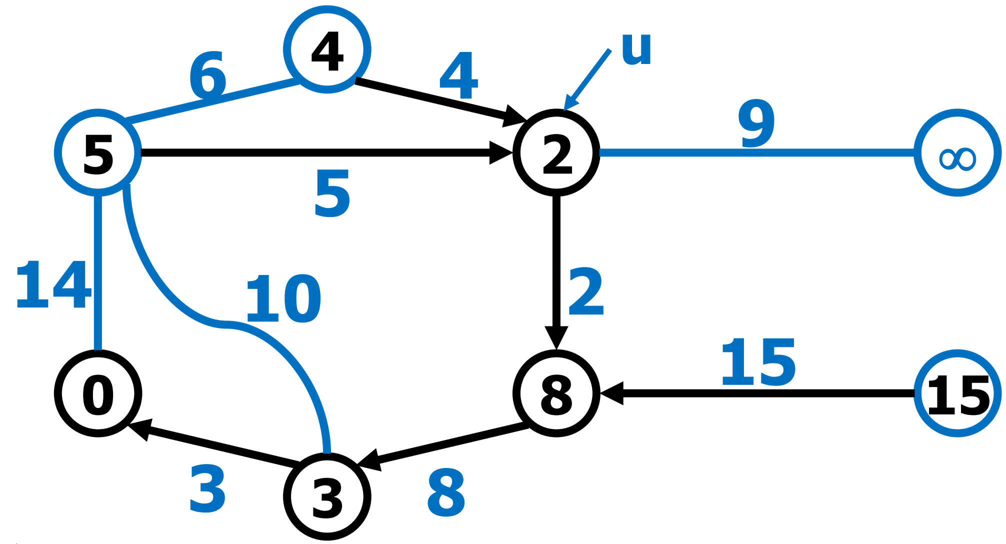
Prim's Algorithm – Example

```
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);
```



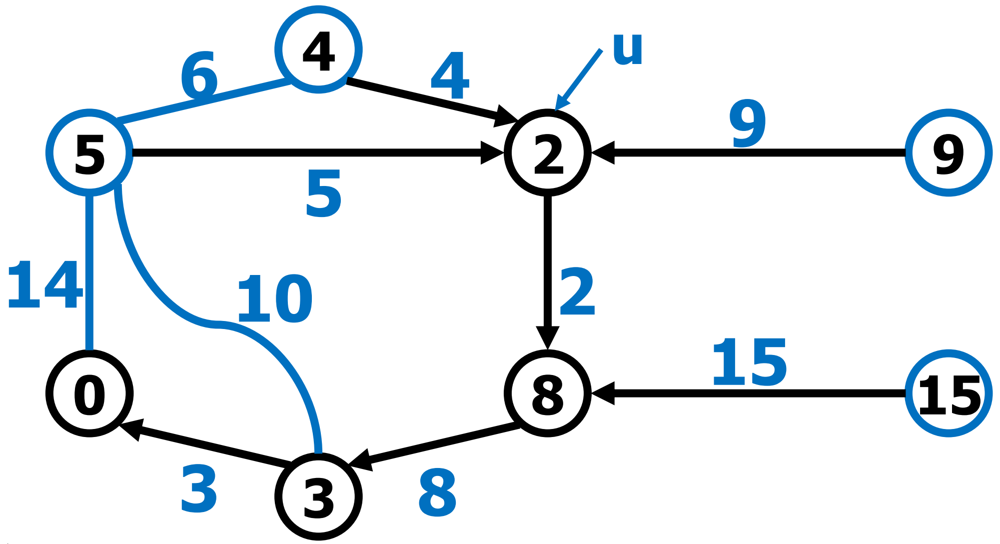
Prim's Algorithm – Example

```
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);
```



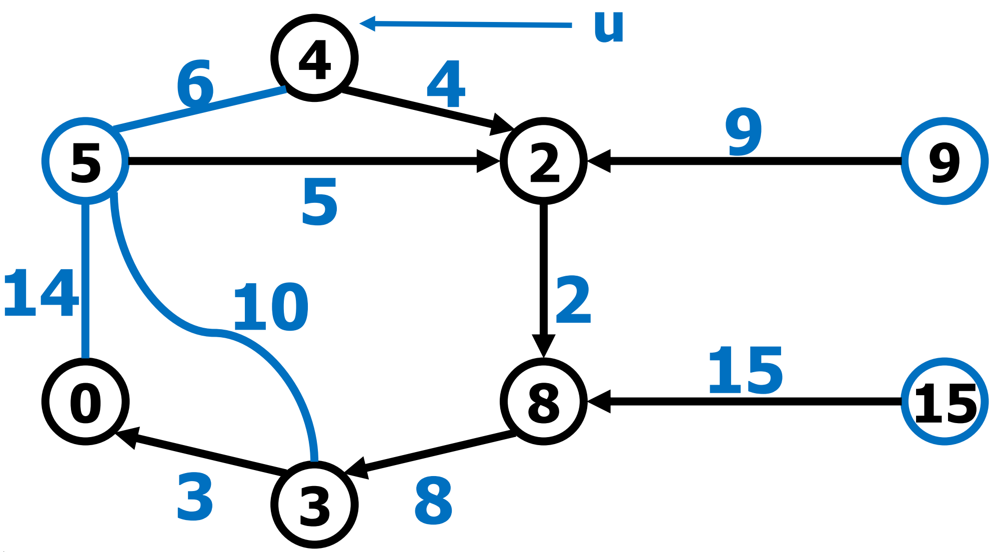
Prim's Algorithm – Example

```
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);
```



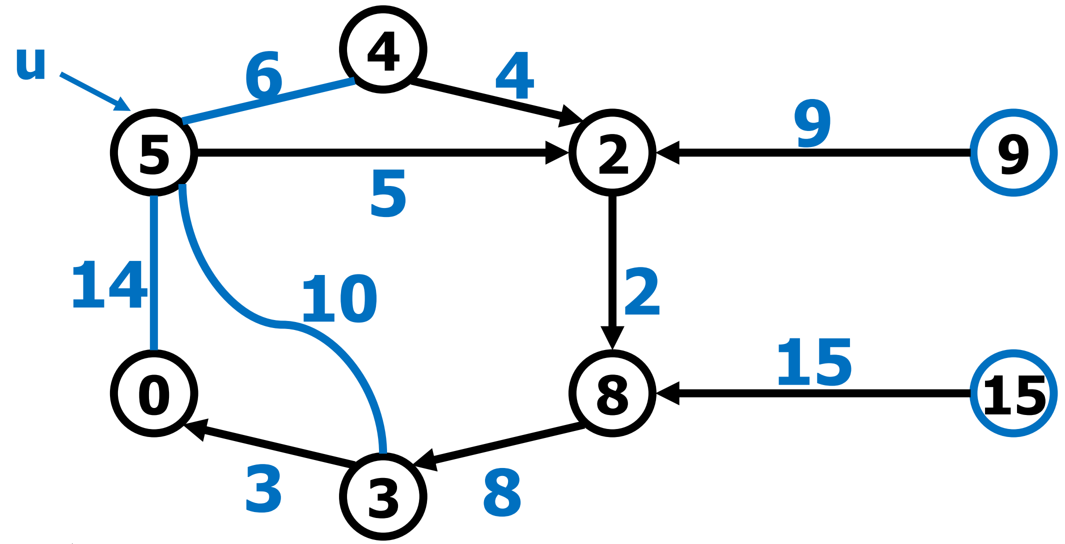
Prim's Algorithm – Example

```
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);
```



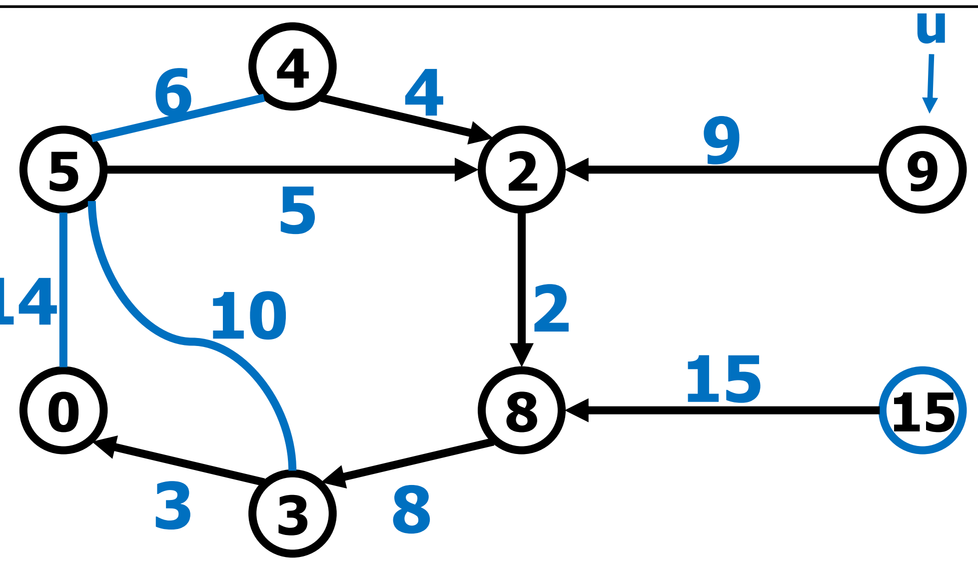
Prim's Algorithm – Example

```
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);
```



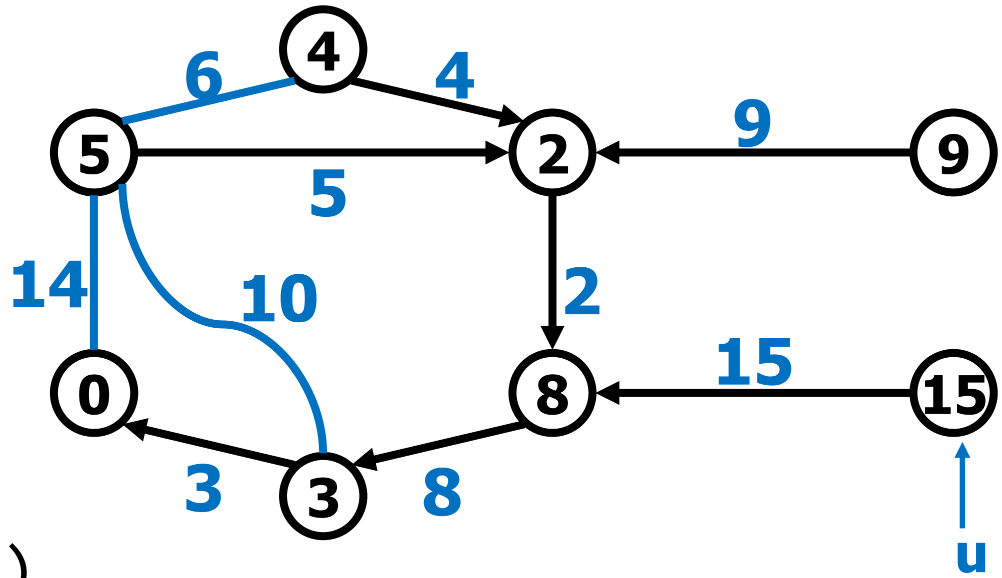
Prim's Algorithm – Example

```
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);
```

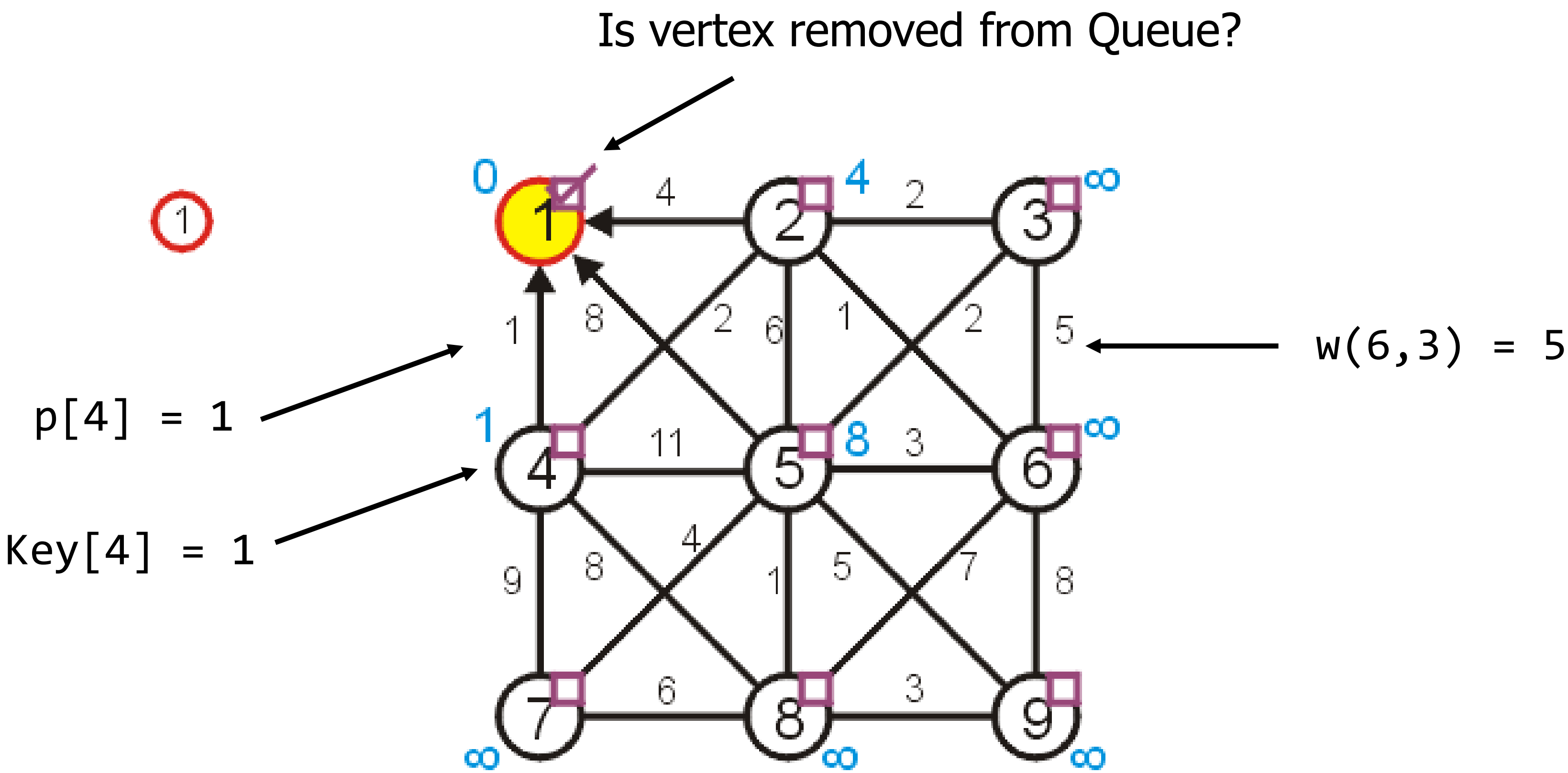


Prim's Algorithm – Example

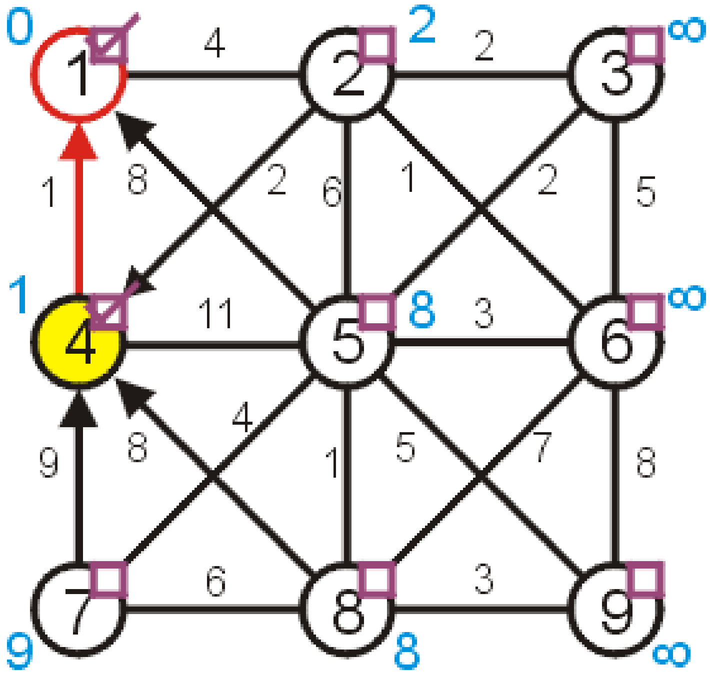
```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```



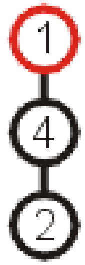
Prim’s Algorithm – Example

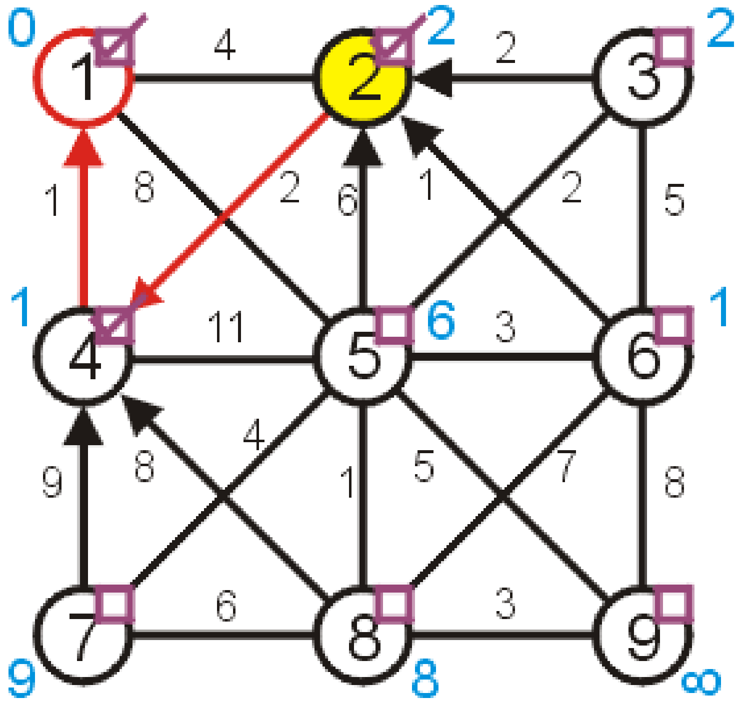


Prim's Algorithm – Example

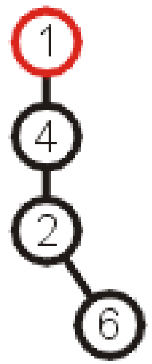


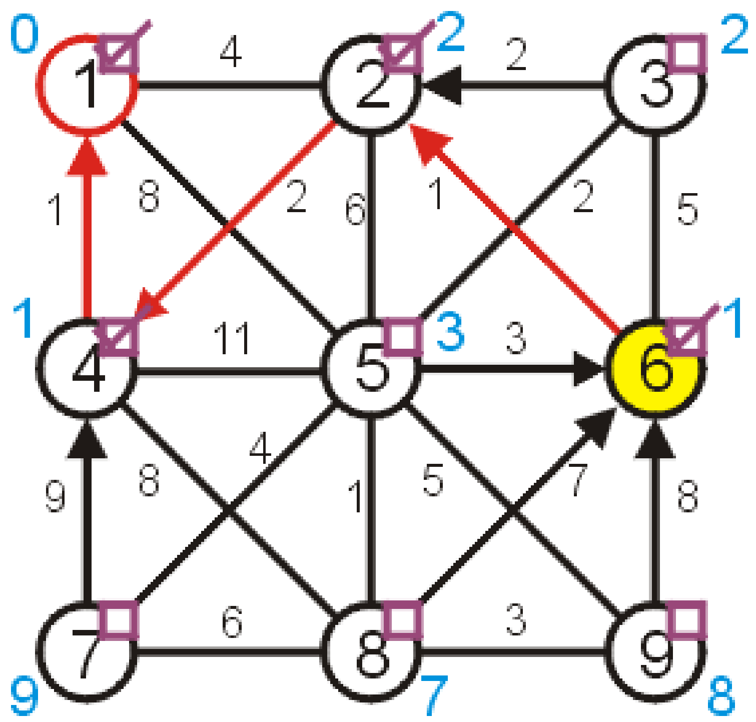
Prim's Algorithm – Example



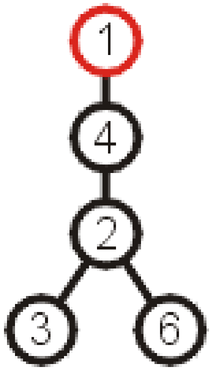


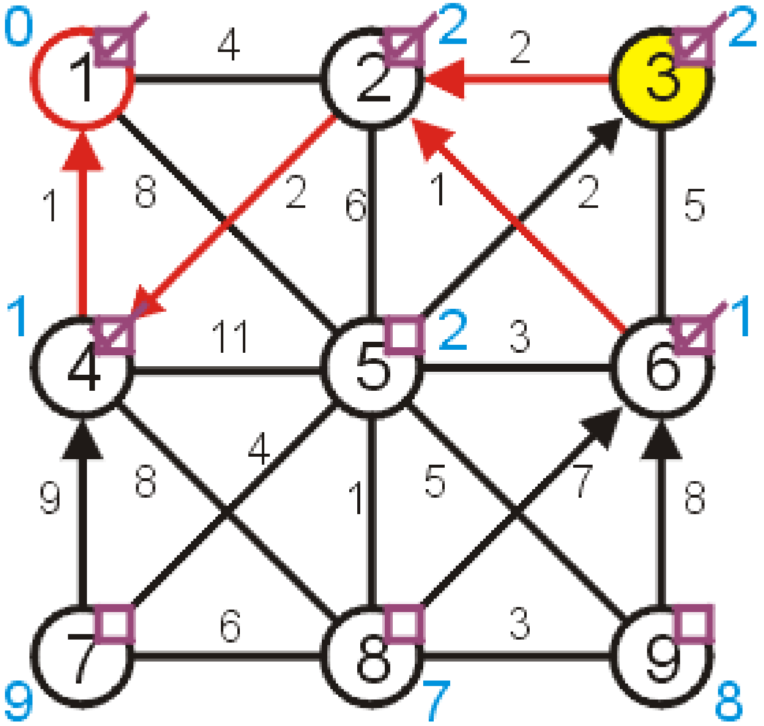
Prim's Algorithm – Example



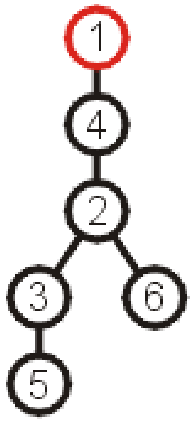


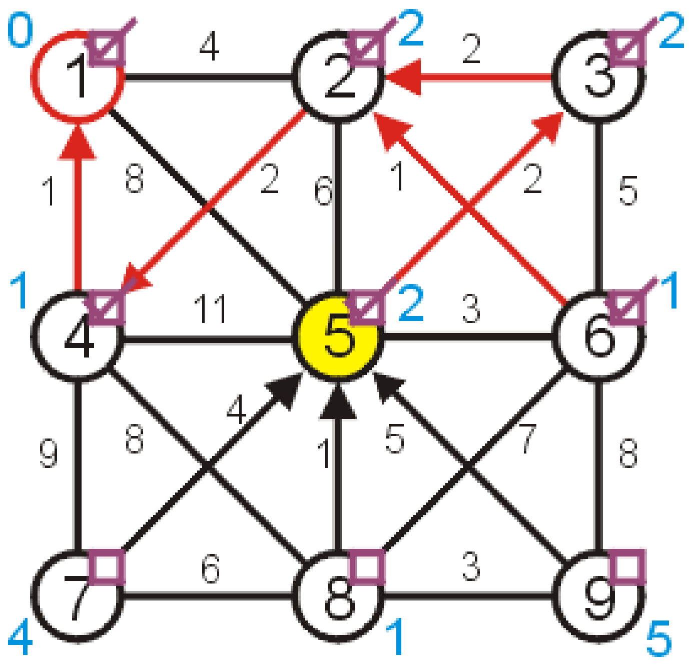
Prim’s Algorithm – Example



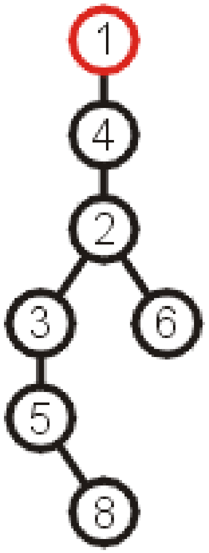


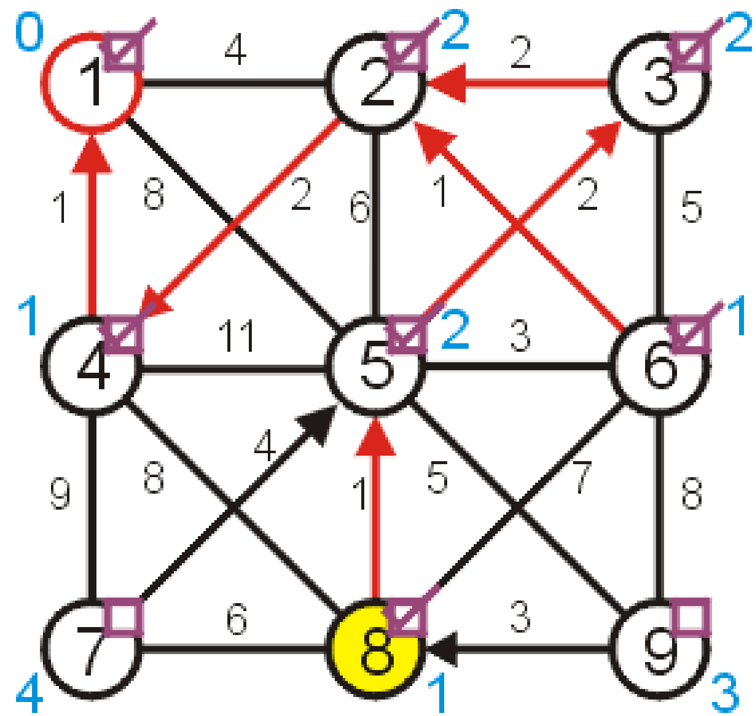
Prim's Algorithm – Example



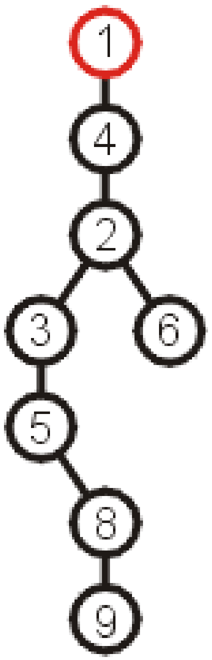


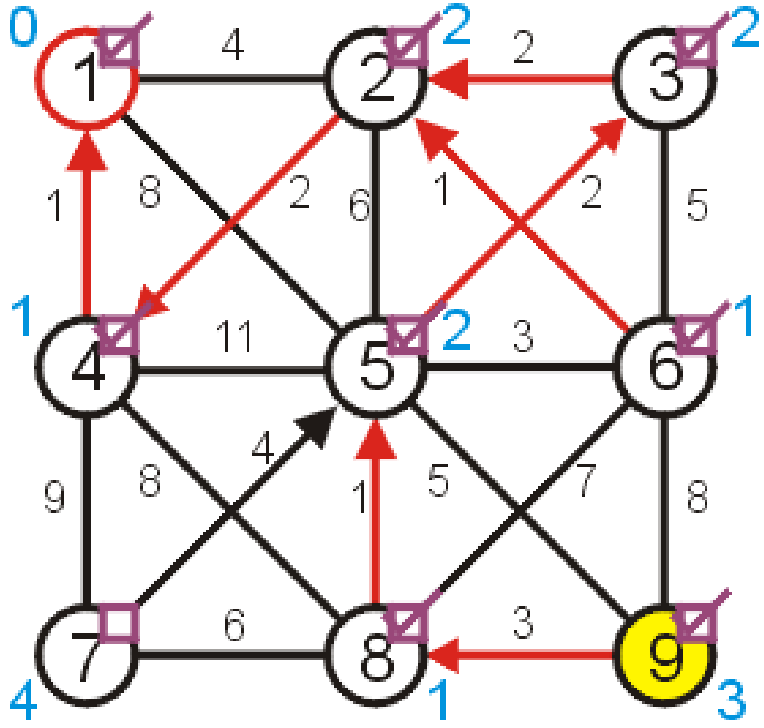
Prim's Algorithm – Example



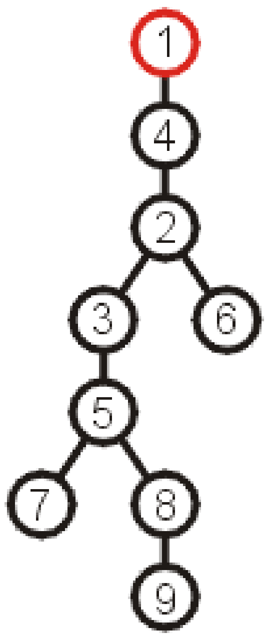


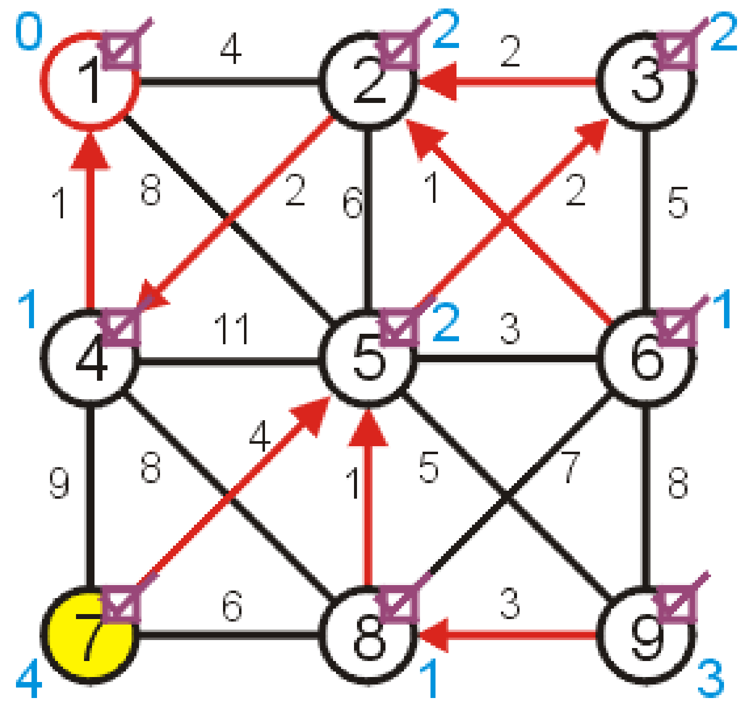
Prim's Algorithm – Example



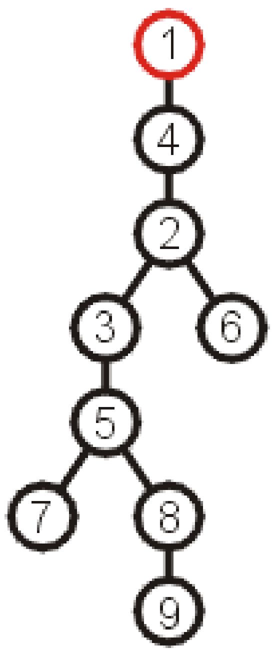


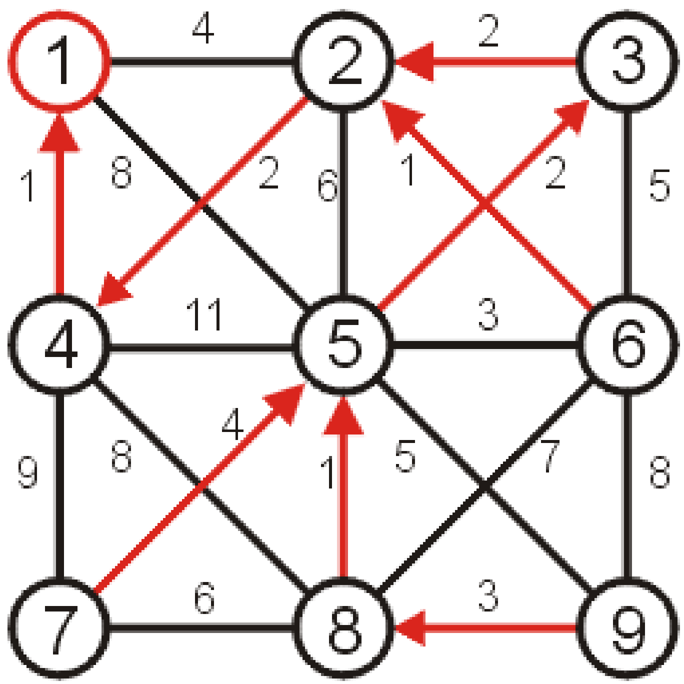
Prim’s Algorithm – Example





Prim’s Algorithm – Example





Kruskal’s Algorithm vs Prim’s Algorithm

- In prim’s algorithm, graph must be a connected
- Kruskal’s algorithm can function on disconnected graphs too
- Prim's algorithm is significantly faster for dense graphs with more number of edges than vertices
- Kruskal's algorithm runs faster in the case of sparse graphs

Any Question So Far?