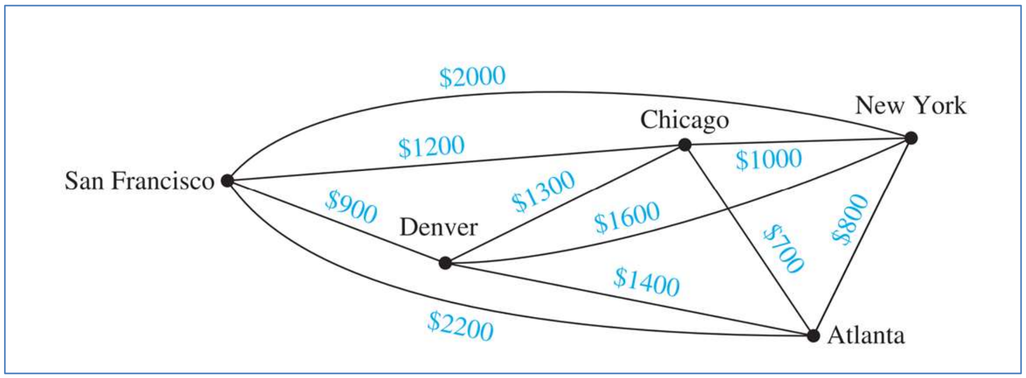
Minimum Spanning Tree

Kazi Lutful Kabir

Spring 2023

Spanning Tree

A company plans to build a communications network connecting its five computer centers. Any pair of these centers can be linked with a leased telephone line. Which links should be made to ensure that there is a path between any two computer centers so that the total cost of the network is minimized?



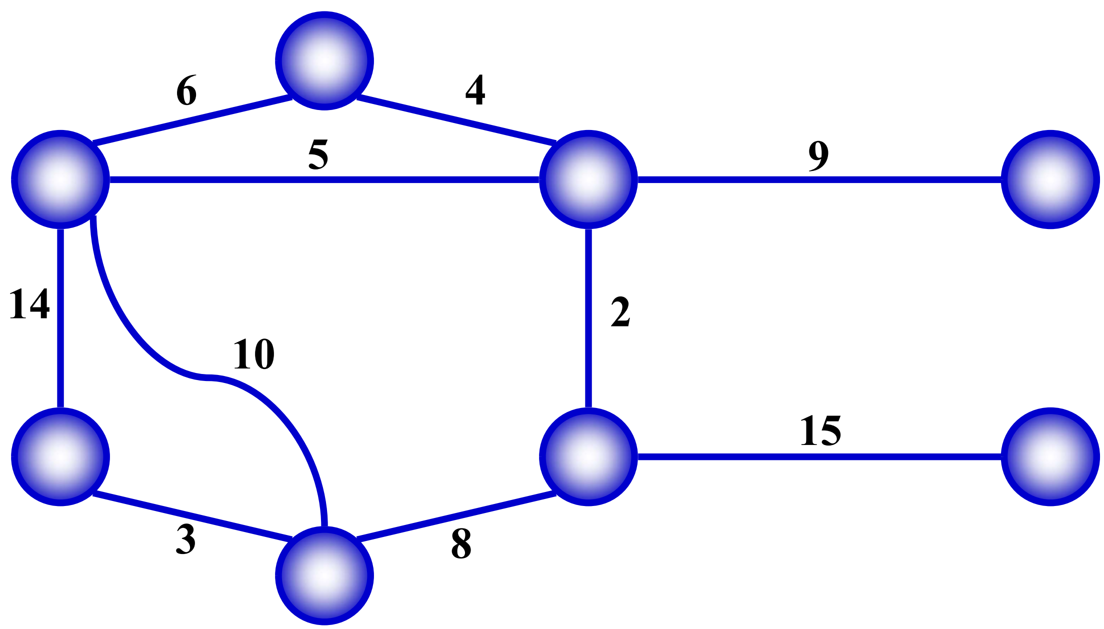
1	{Chicago, Atlanta}		\$ 700
2	{Atlanta, New York}		\$ 800
3	{Chicago, San Francisco}		\$1200
4	{San Francisco, Denver}		\$ 900
		Total:	\$3600

- A **spanning tree** T of an undirected graph G is a subgraph that is a tree which includes all the vertices of G

A *minimum spanning tree* in a connected weighted graph is a spanning tree that has the smallest possible sum of weights of its edges.

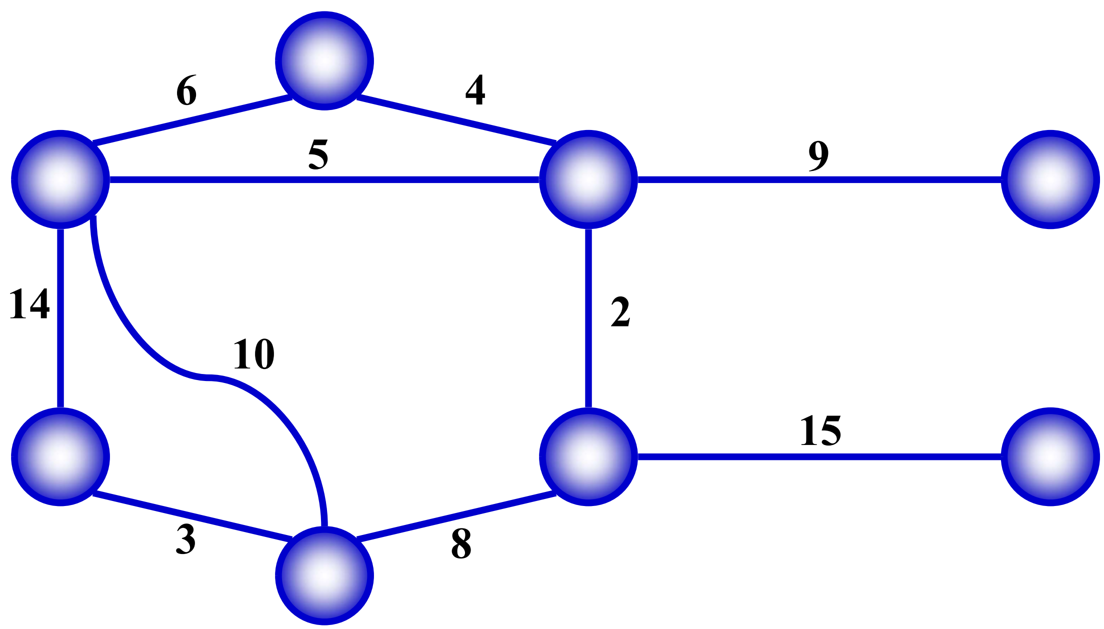
Minimum Spanning Tree

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



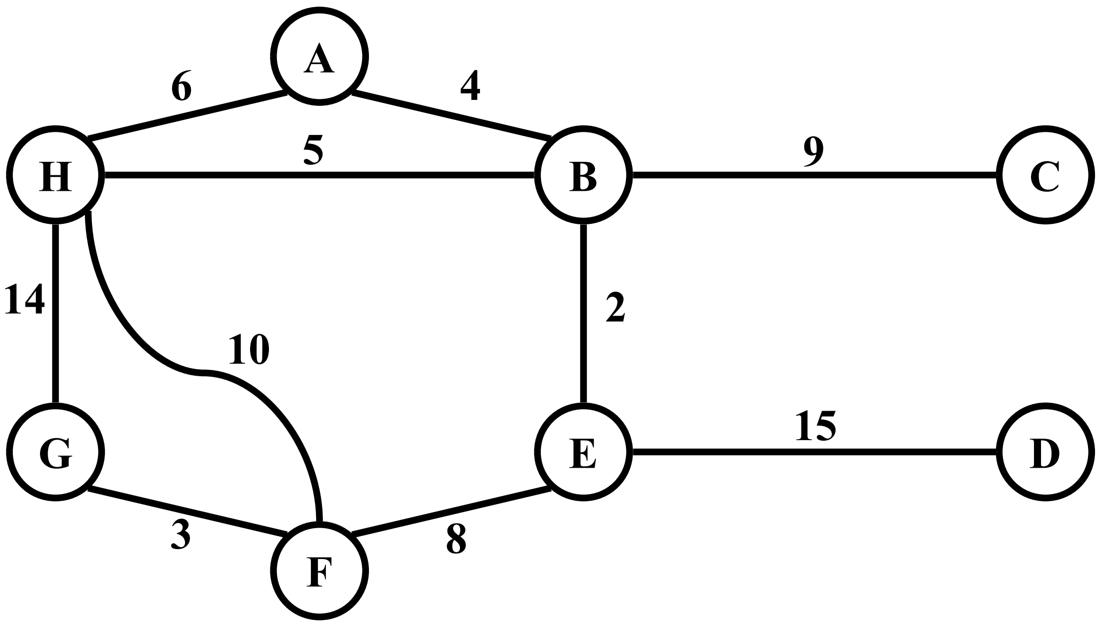
Minimum Spanning Tree

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



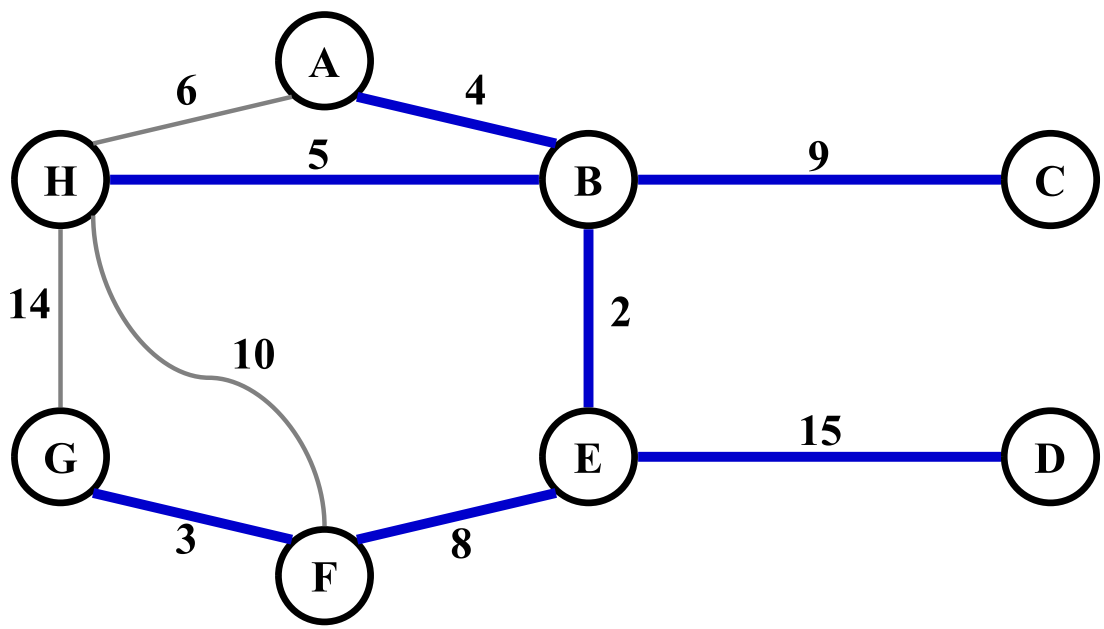
Minimum Spanning Tree

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



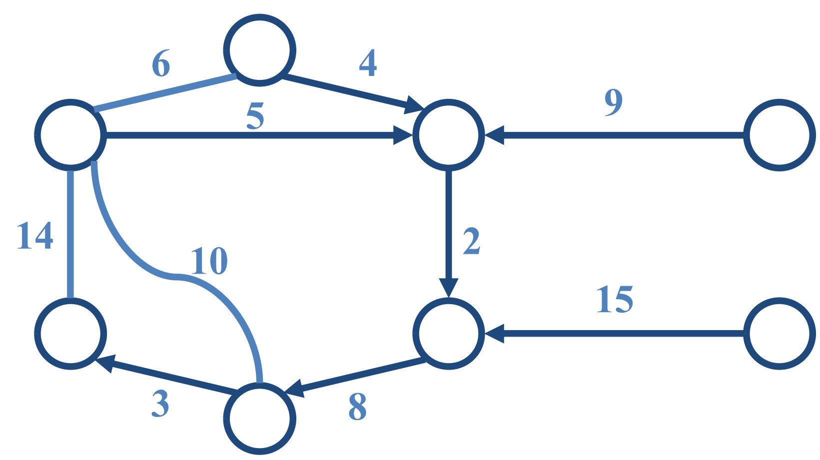
Minimum Spanning Tree

- Answer:



Minimum Spanning Tree

- *Optimal substructure* property: an optimal minimum spanning tree is composed of optimal minimum spanning subtrees



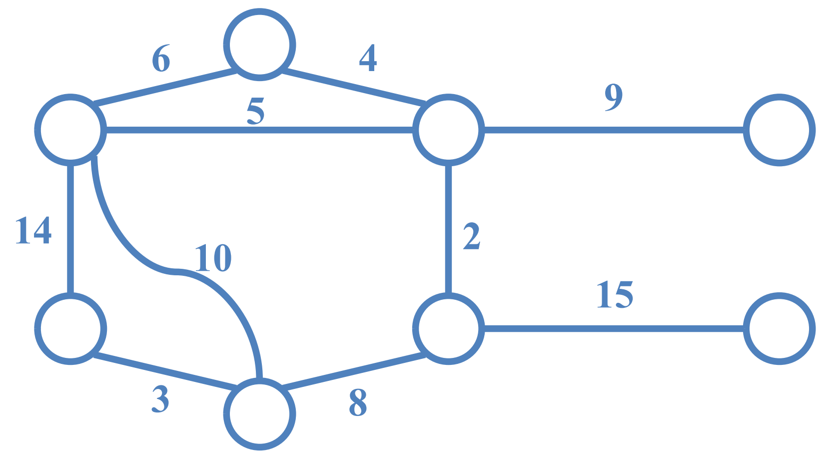
Prim's Algorithm

MST-PRIM(G, w, r)

```
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

Prim’s Algorithm

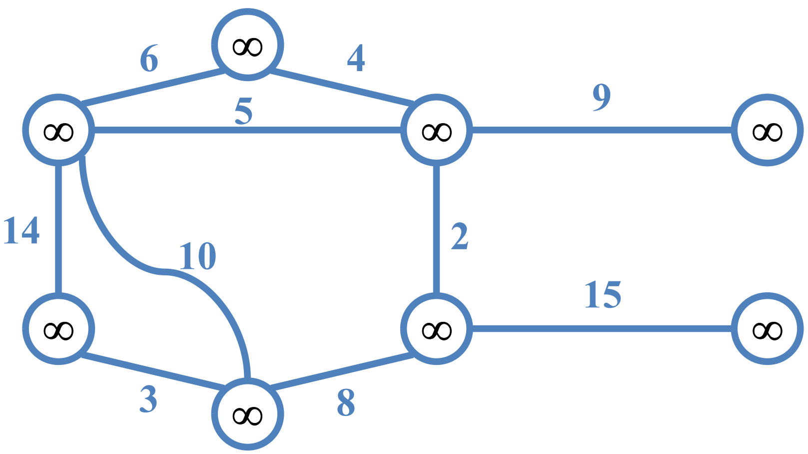
```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```



Run on example graph

Prim’s Algorithm

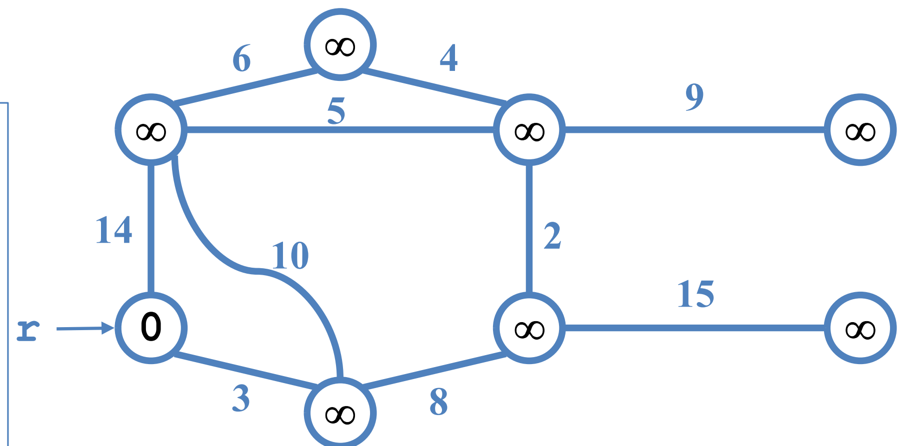
```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```



Run on example graph

Prim's Algorithm

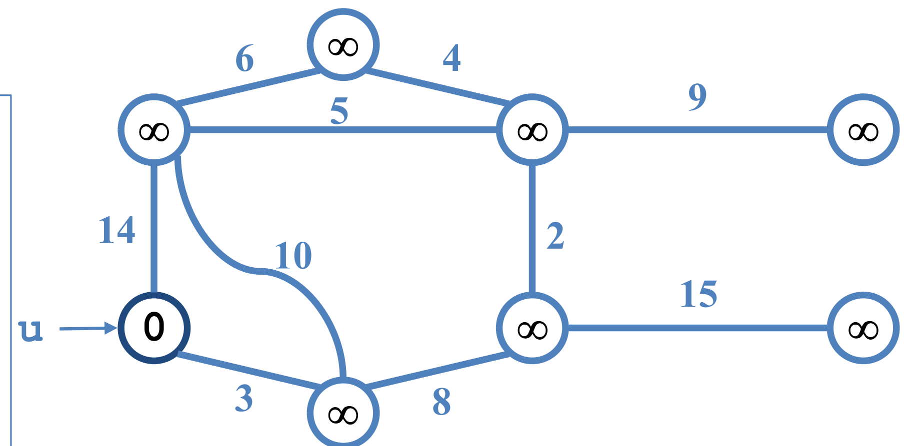
```
MST-PRIM(G, w, r)
    for each u ∈ G.V
        u.key = ∞
        u.π = NIL
    r.key = 0
    Q = G.V
    while Q ≠ ∅
        u = EXTRACT-MIN(Q)
        for each v ∈ G.Adj[u]
            if v ∈ Q and w(u, v) < v.key
                v.π = u
                v.key = w(u, v)
```



Pick a start vertex r

Prim's Algorithm

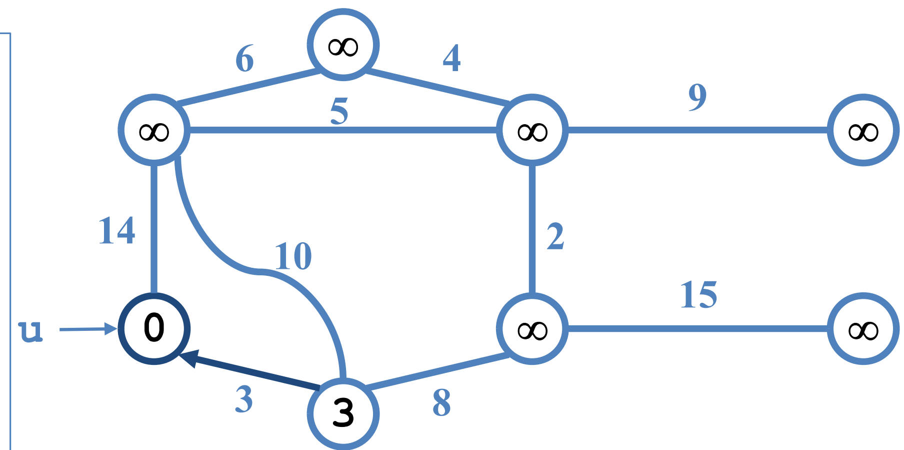
```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```



Dark Blue vertices have been removed from Q

Prim's Algorithm

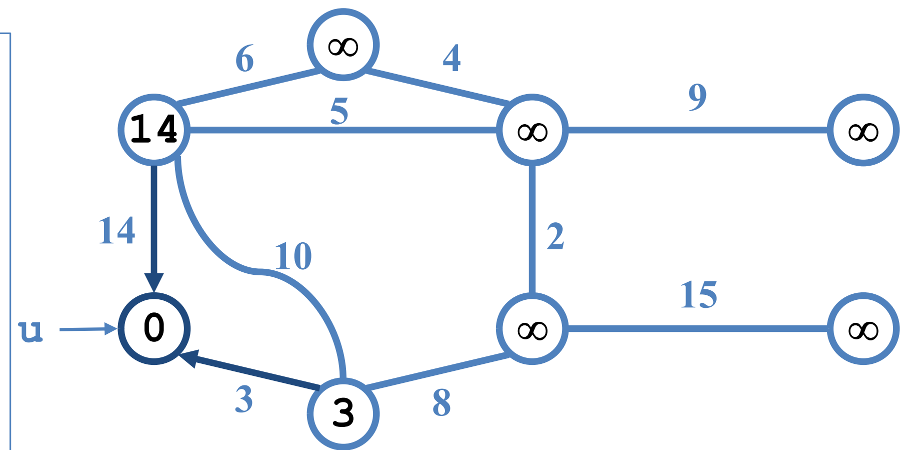
```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```



Dark Blue arrows indicate parent pointers

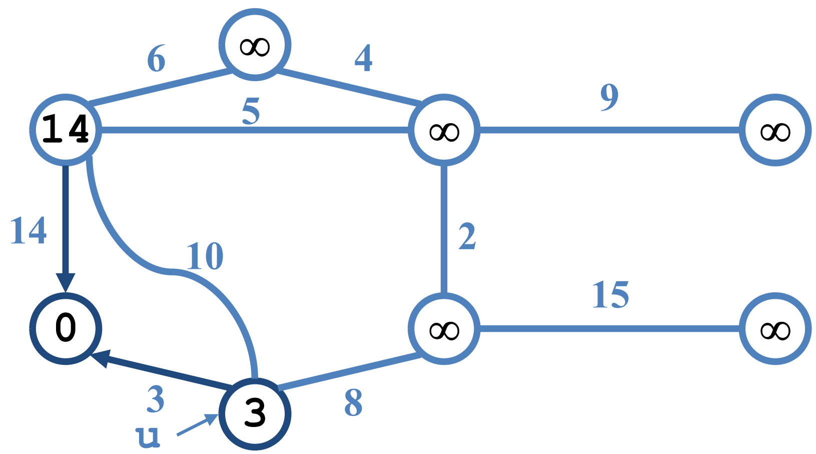
Prim's Algorithm

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```



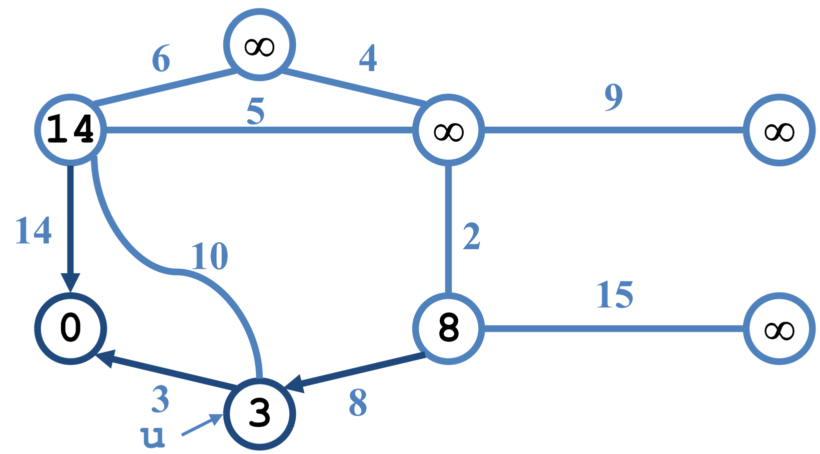
Prim's Algorithm

```
MST-PRIM(G, w, r)
    for each u ∈ G.V
        u.key = ∞
        u.π = NIL
    r.key = 0
    Q = G.V
    while Q ≠ ∅
        u = EXTRACT-MIN(Q)
        for each v ∈ G.Adj[u]
            if v ∈ Q and w(u, v) < v.key
                v.π = u
                v.key = w(u, v)
```



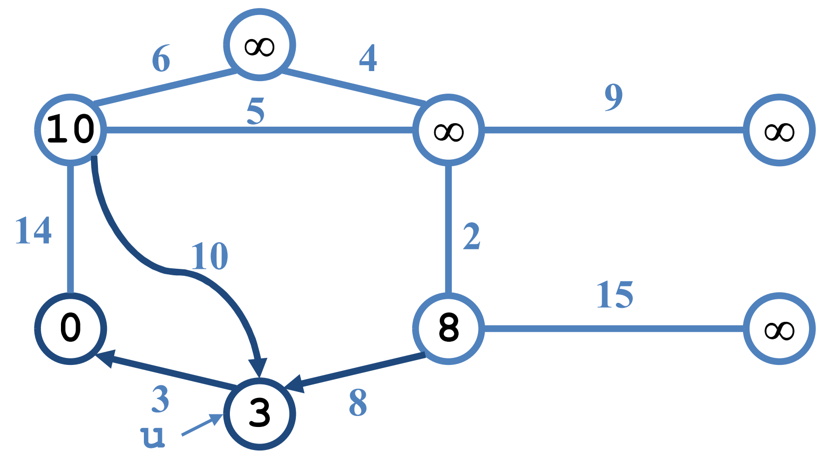
Prim's Algorithm

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```



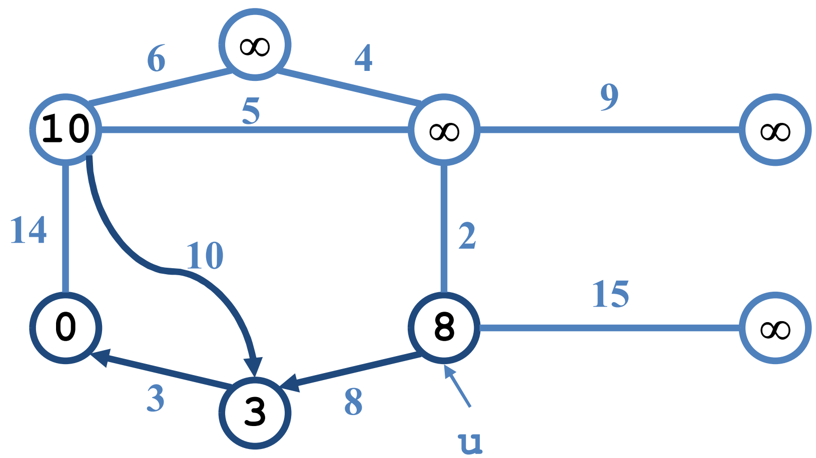
Prim's Algorithm

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```



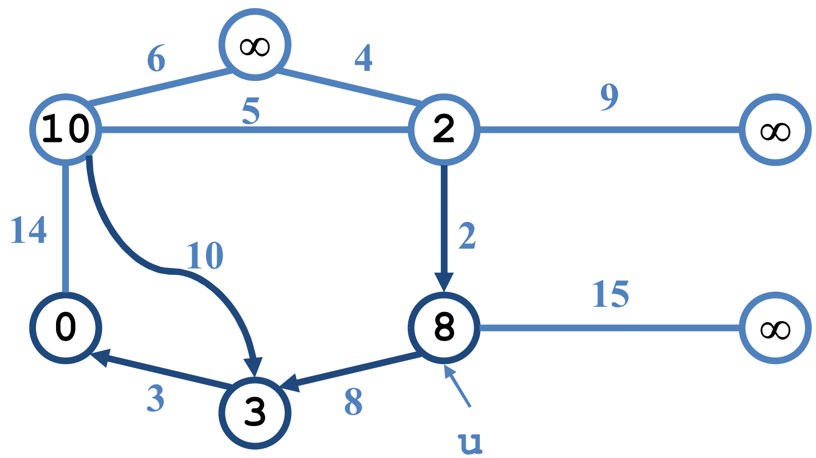
Prim’s Algorithm

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```



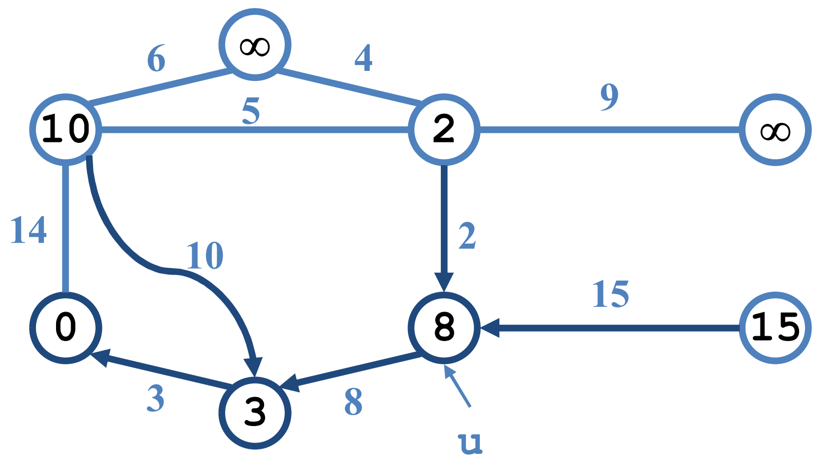
Prim's Algorithm

```
MST-PRIM(G, w, r)
  for each u ∈ G.V
      u.key = ∞
      u.π = NIL
  r.key = 0
  Q = G.V
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      for each v ∈ G.Adj[u]
          if v ∈ Q and w(u, v) < v.key
              v.π = u
              v.key = w(u, v)
```



Prim's Algorithm

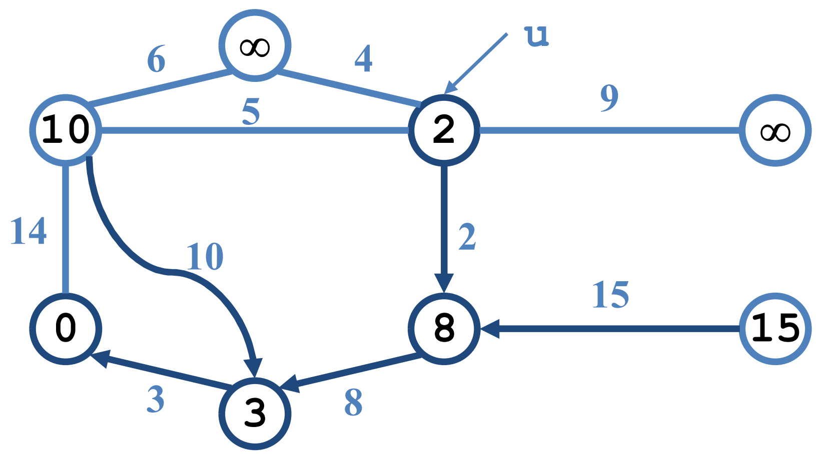
```
MST-PRIM(G, w, r)
    for each u ∈ G.V
        u.key = ∞
        u.π = NIL
    r.key = 0
    Q = G.V
    while Q ≠ ∅
        u = EXTRACT-MIN(Q)
        for each v ∈ G.Adj[u]
            if v ∈ Q and w(u, v) < v.key
                v.π = u
                v.key = w(u, v)
```



Prim's Algorithm

MST-PRIM(G, w, r)

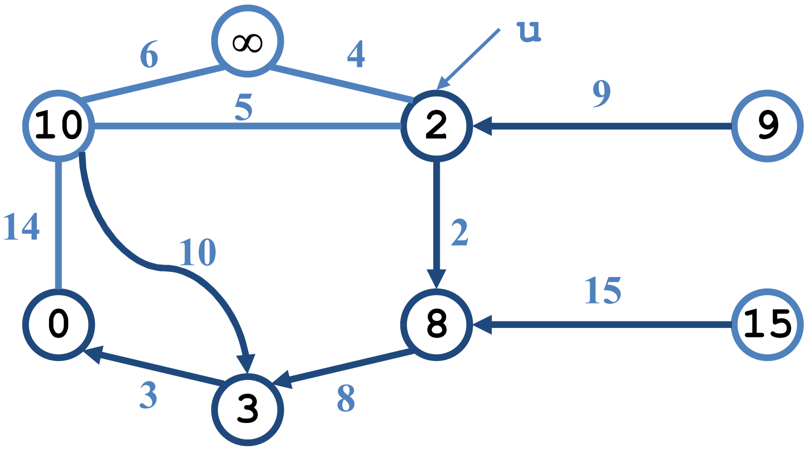
```
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```



Prim's Algorithm

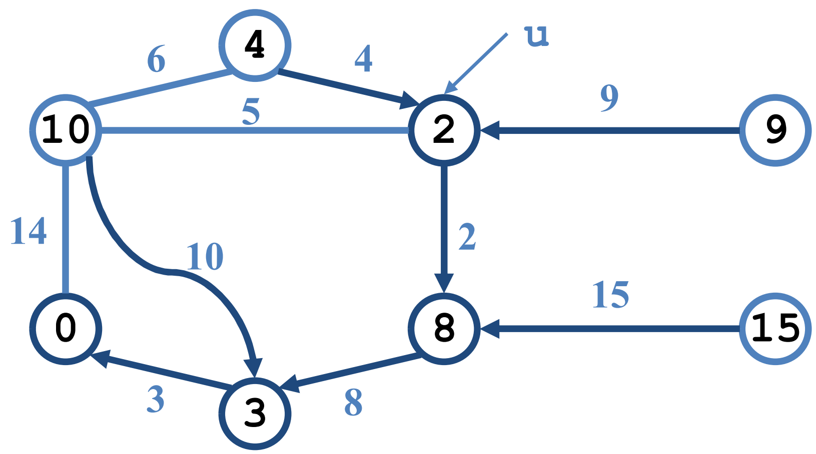
MST-PRIM(G, w, r)

```
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```



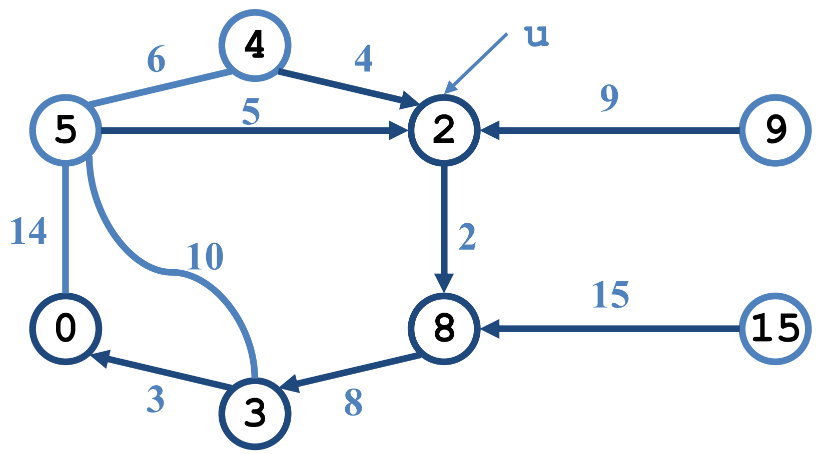
Prim's Algorithm

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```



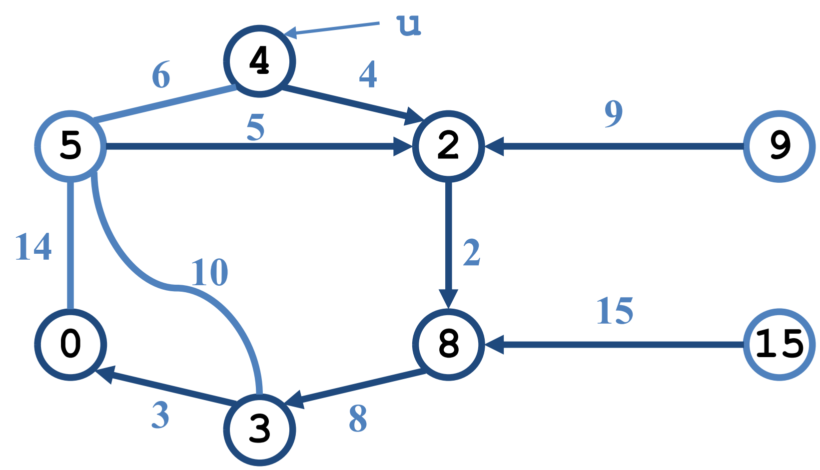
Prim's Algorithm

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```



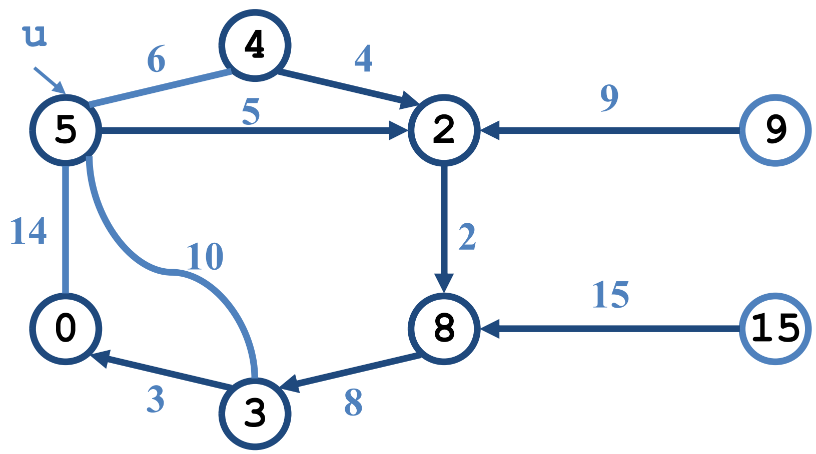
Prim's Algorithm

```
MST-PRIM(G, w, r)
    for each u ∈ G.V
        u.key = ∞
        u.π = NIL
    r.key = 0
    Q = G.V
    while Q ≠ ∅
        u = EXTRACT-MIN(Q)
        for each v ∈ G.Adj[u]
            if v ∈ Q and w(u, v) < v.key
                v.π = u
                v.key = w(u, v)
```



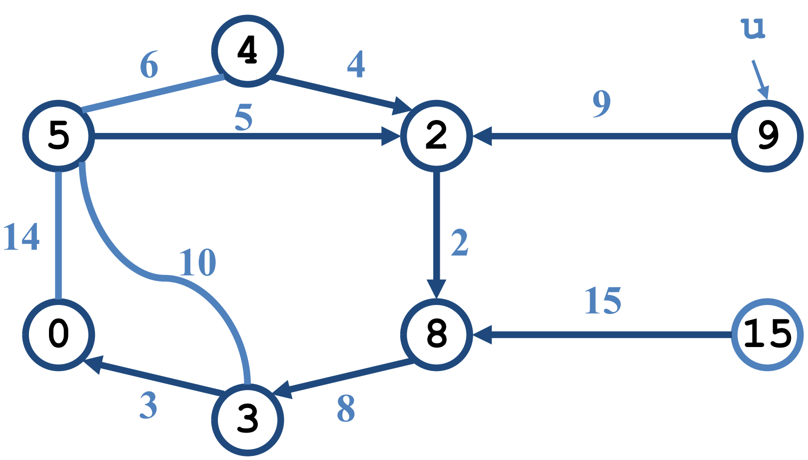
Prim's Algorithm

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```



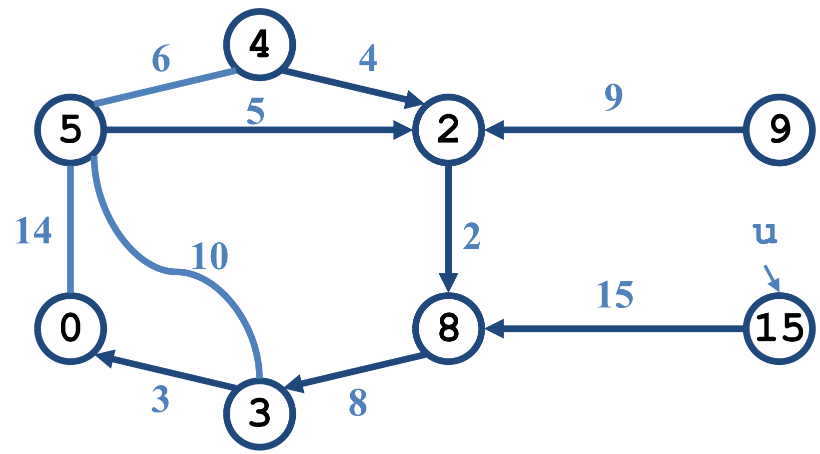
Prim's Algorithm

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```



Prim's Algorithm

```
MST-PRIM(G, w, r)
    for each u ∈ G.V
        u.key = ∞
        u.π = NIL
    r.key = 0
    Q = G.V
    while Q ≠ ∅
        u = EXTRACT-MIN(Q)
        for each v ∈ G.Adj[u]
            if v ∈ Q and w(u, v) < v.key
                v.π = u
                v.key = w(u, v)
```



Prim’s Running Time

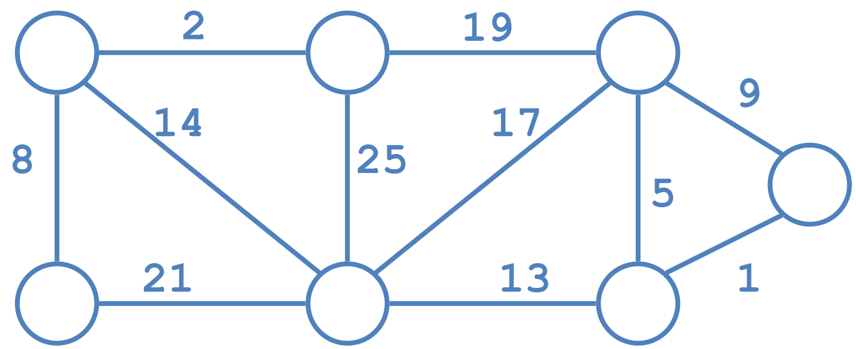
```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

A simple implementation of Prim's, using an adjacency matrix or an adjacency list graph representation and linearly searching an array of weights to find the minimum weight edge to add, requires **$O(|V|^2)$** running time.

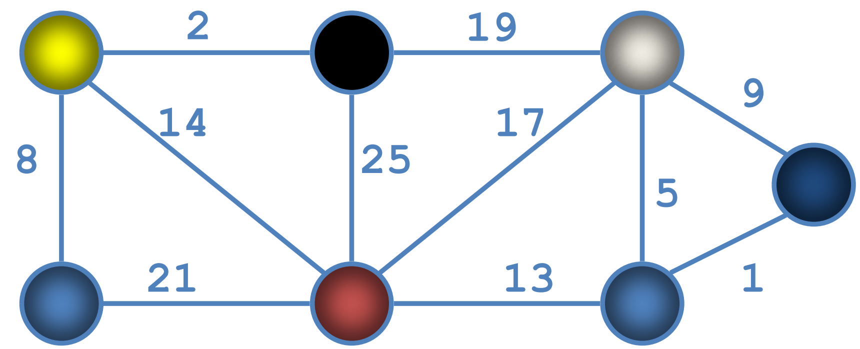
Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

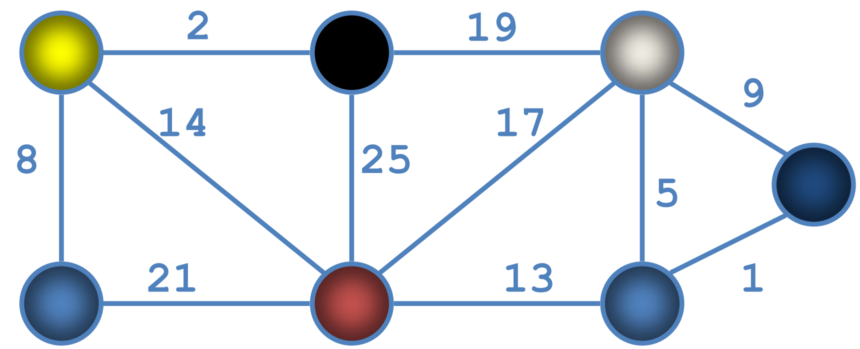
Kruskal's Algorithm



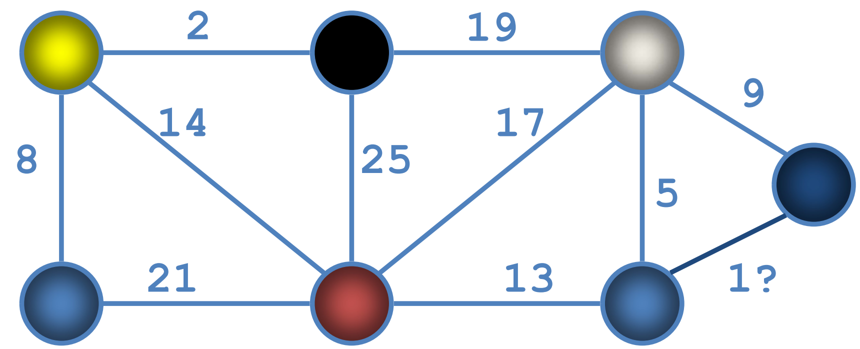
Kruskal’s Algorithm



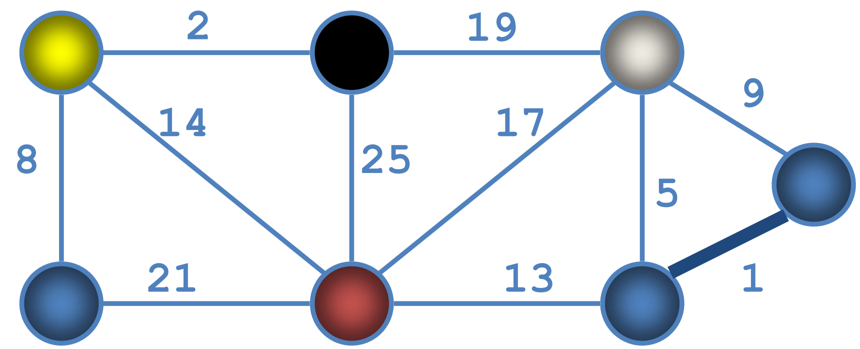
Kruskal's Algorithm



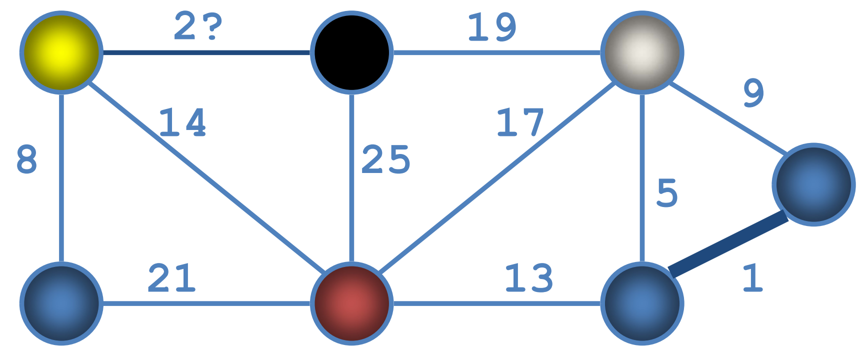
Kruskal's Algorithm



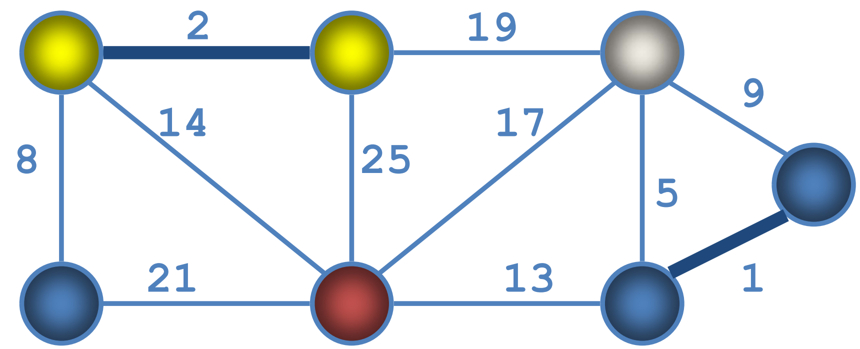
Kruskal’s Algorithm

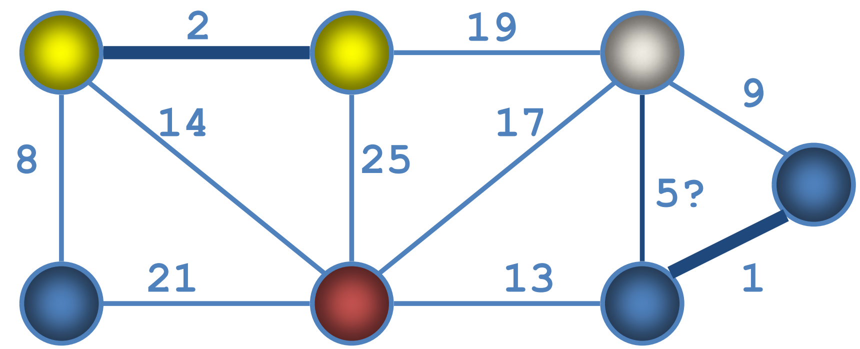


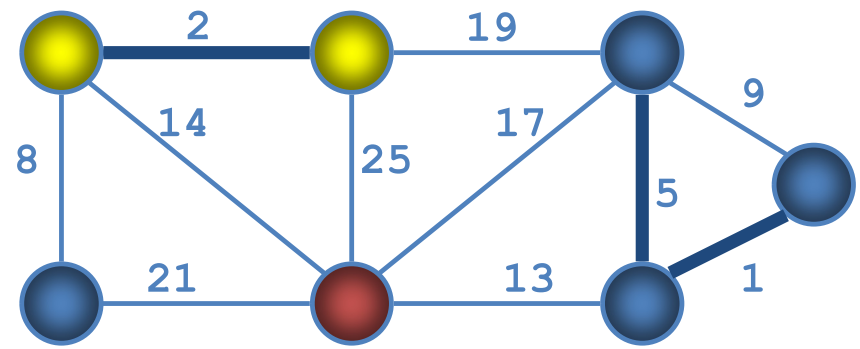
Kruskal's Algorithm

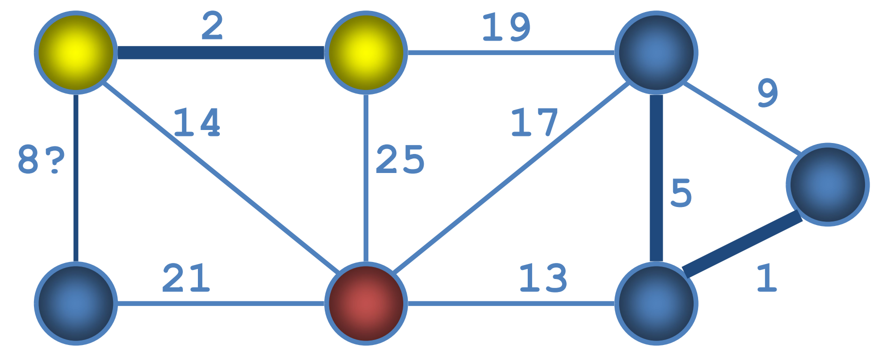


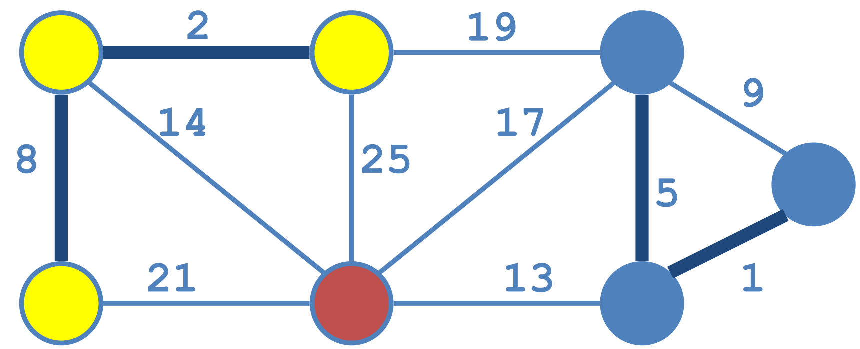
Kruskal's Algorithm

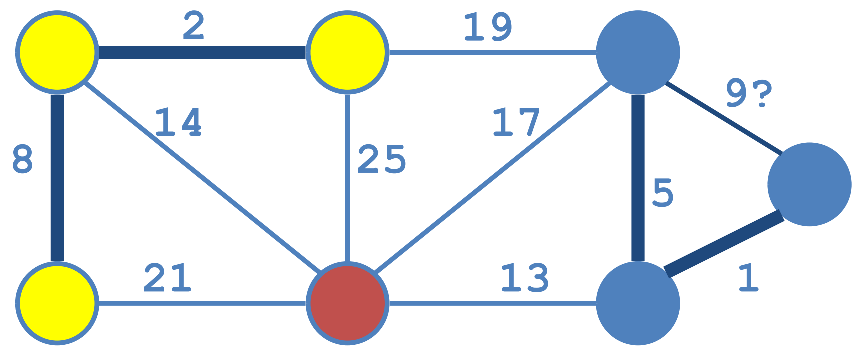



2
19
8
14
25
17
9
5?
21
13
1

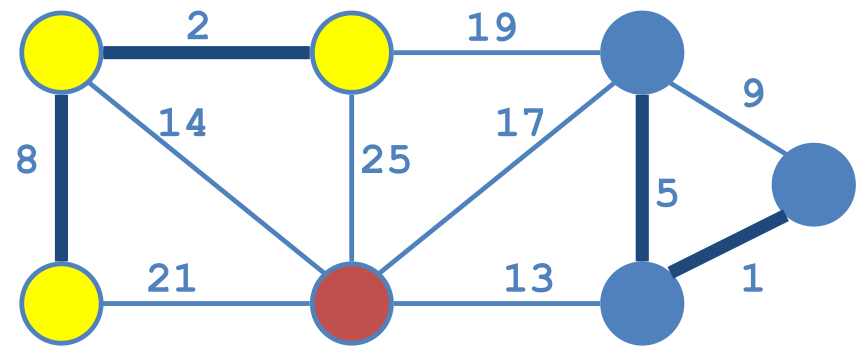

2
19
9
14
17
8
25
5
21
13
1

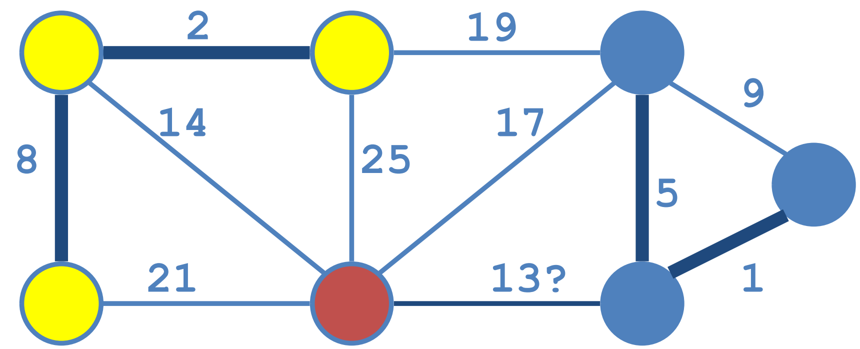

2
19
9
14
8?
25
17
5
21
13
1


2
19
9
14
17
8
25
5
21
13
1

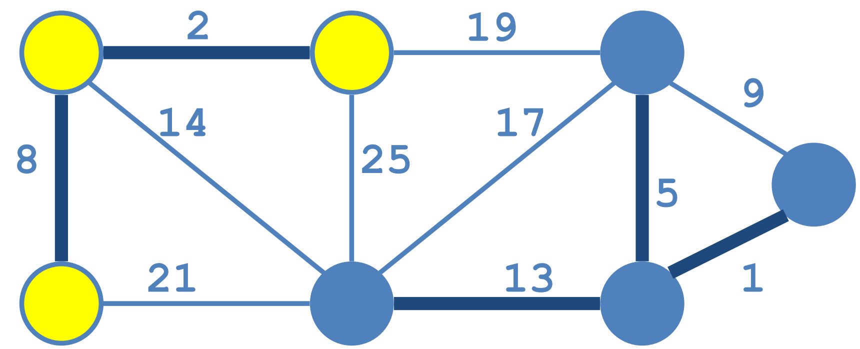


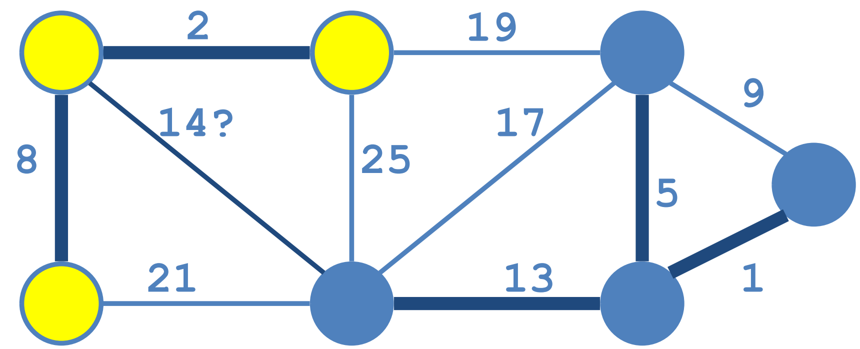
2
19
9?
14
17
8
25
5
21
13
1

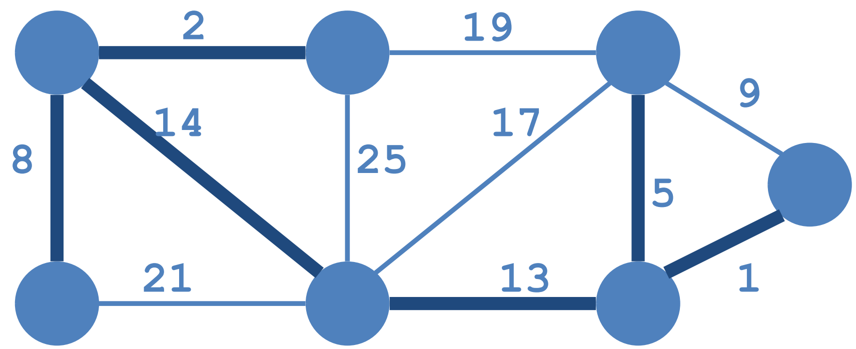

2
19
9
8
14
25
17
5
21
13
1



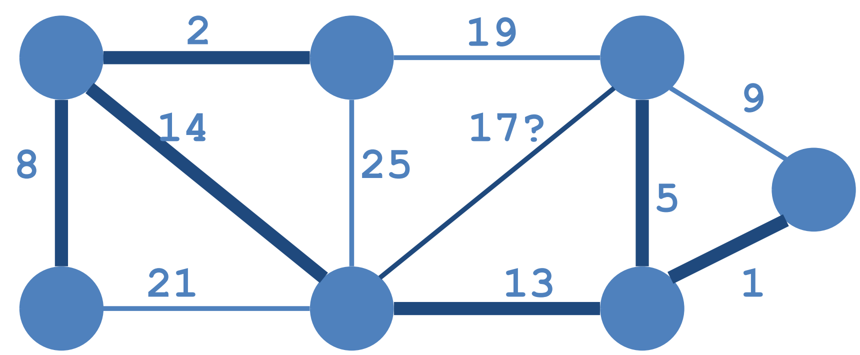
2
19
9
14
17
8
25
5
21
13?
1

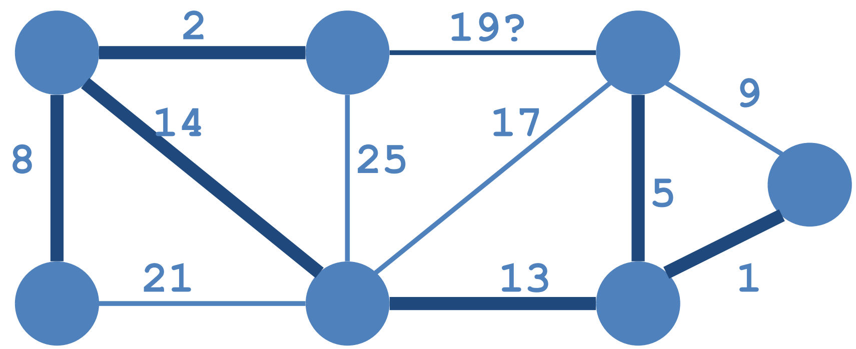

2
19
9
8
14
25
17
5
21
13
1

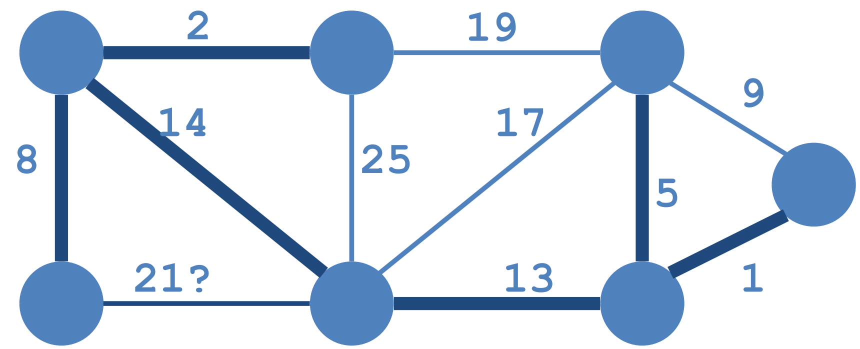

2
19
9
8
14?
25
17
5
21
13
1

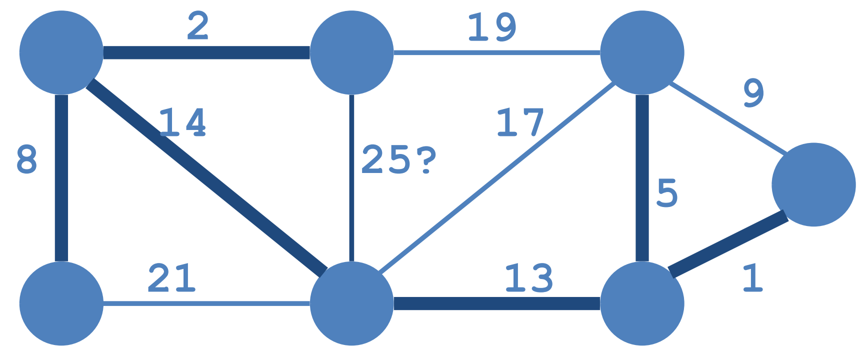


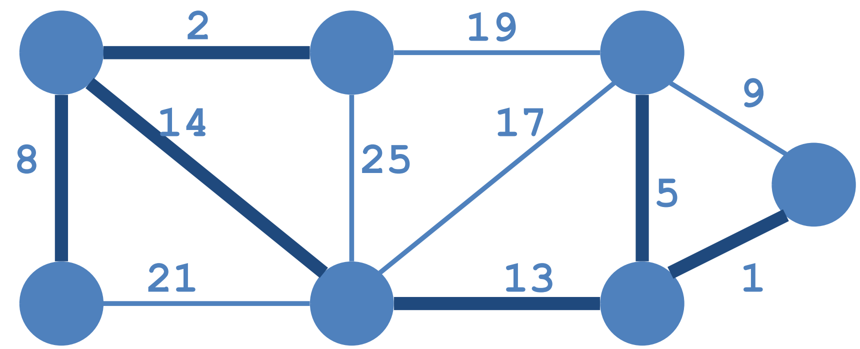
2
19
9
8
14
25
17
5
21
13
1

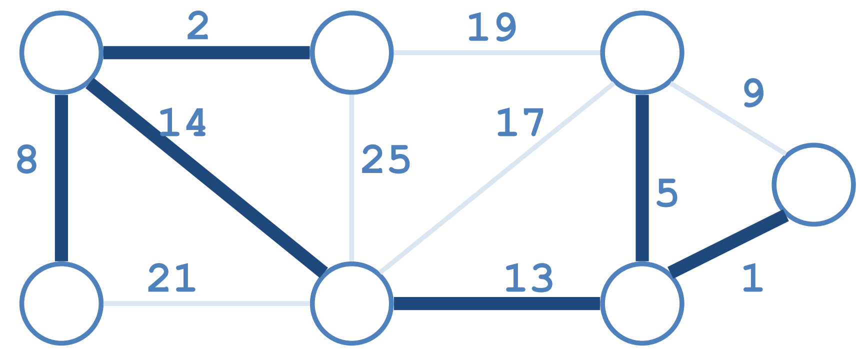

2
19
9
14
17?
8
25
5
21
13
1


2
19?
14
8
25
17
9
5
21
13
1


2
19
9
14
17
8
25
5
21?
13
1


2
19
9
8
14
25?
17
5
21
13
1


2
19
9
8
14
25
17
5
21
13
1


2
19
9
14
17
8
25
5
21
13
1

Kruskal's Algorithm: Running Time

- To summarize:
  - Sort edges: O(E lg E)
  - O(V) for Make-Set()
  - O(E) for Find-Set()
  - O(E) for Union()
  - overall time: **O(E lg E)**

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

Reference

Chap-23, Introduction to Algorithms (3rd Ed.) by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein

STEVENS
INSTITUTE OF TECHNOLOGY
1870

THANK **YOU**

Stevens Institute of Technology