CS 311: Algorithm Design and Analysis

Lecture 7

Last Lecture we have

- Heap Sort Example
- Linear Sorting Algorithms
- Multiplication of large integers
- Tower of Hanoi

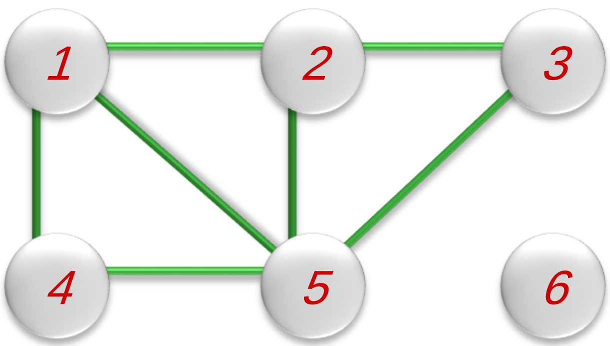
This Lecture we have

- Graph
- MST

Graph

Graph G = (V, E)
V = V(G) = ***vertex set*** *of G*
E = E(G) = ***edge set*** *of G (a set of pairs of vertices of G)*

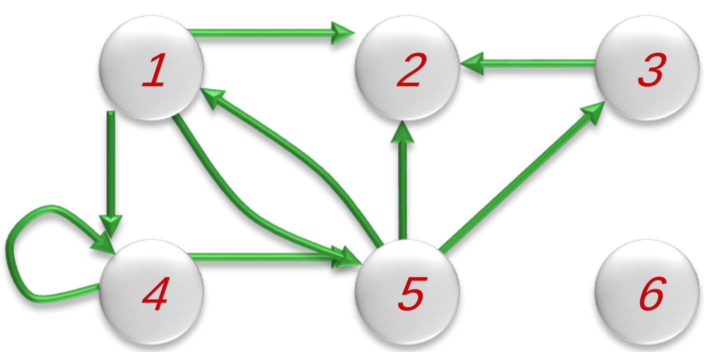
Undirected graph: *edges are unordered pairs of vertices:*



V = { 1, 2, 3, 4, 5, 6 }

E = {(1,2), (1,4), (1,5), (2,3), (2,5), (3,5), (4,5)}

Directed graph *(or digraph): edges are ordered pairs of vertices:*



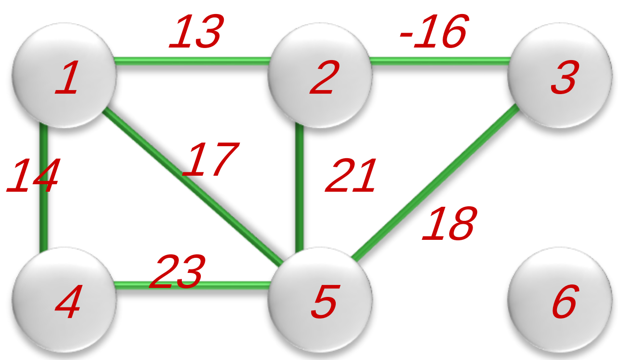
V = { 1, 2, 3, 4, 5, 6 }

E = { (1,2), (1,4), (1,5), (3,2),
(4,4), (4,5), (5,1), (5,2), (5,3)}

Edge Weighted Graph

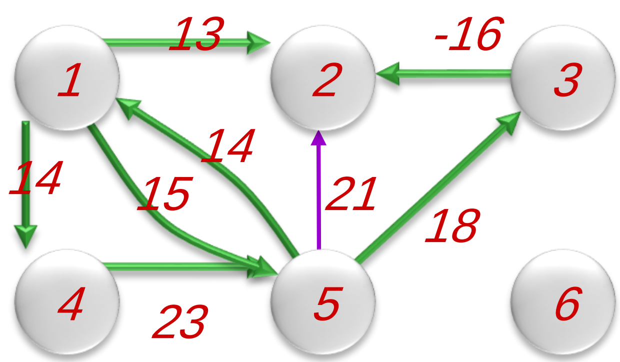
$G = (V, E, w)$

$w: E \longmapsto \Re$



$E = \{(1,2,13),\ (1,4,14),\ (1,5,17),\ (2,3,-16),\ (2,5,21),\ (3,5,18),\ (4,5,23)\}$

e.g., $w(1,5) = w(5,1) = 17$.

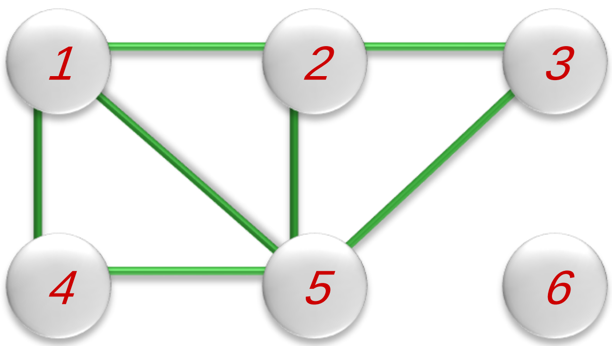


$E = \{\ (1,2,13),\ (1,4,14),\ (1,5,15),\ (3,2,-16)\ (4,5,23),\ (5,1,14),\ (5,2,21),\ (5,3,18)\}$

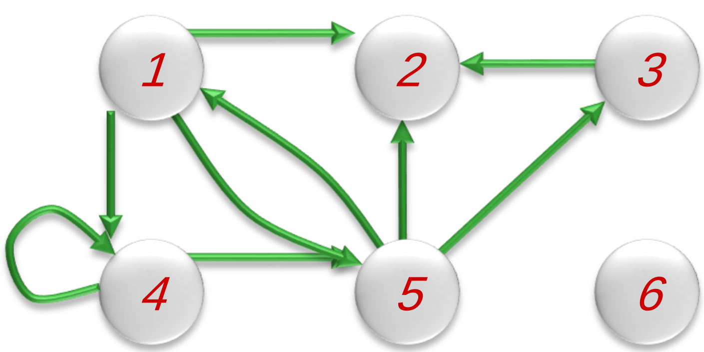
e.g., $w(1,5) = 15,\ \ w(5,1) = 14$.

Adjacency Matrix

$$A[i, j] = \begin{cases} 1 & \text{if } (i, j) \in E(G) \\ 0 & \text{otherwise} \end{cases}, \quad \text{for } i, j \in V(G).$$



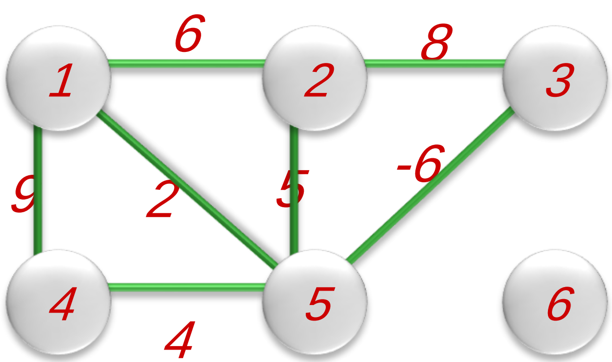
$$A = \begin{array}{cc} & \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{matrix} & \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$



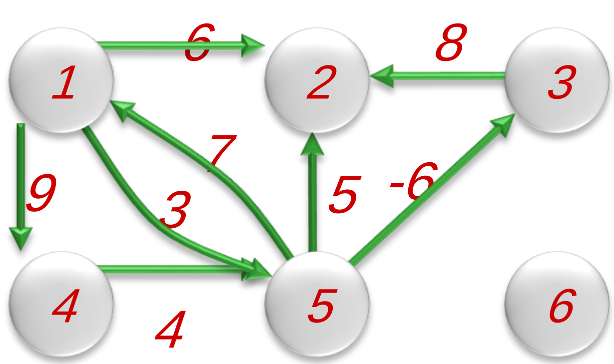
$$A = \begin{array}{cc} & \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{matrix} & \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Weighted Adjacency Matrix

$$A[i, j] = \begin{cases} w(i, j) & \text{if } (i, j) \in E(G) \\ 0 & \text{if } i = j, (i, j) \notin E(G) \\ \infty & \text{otherwise} \end{cases}, \quad \text{for } i, j \in V(G).$$



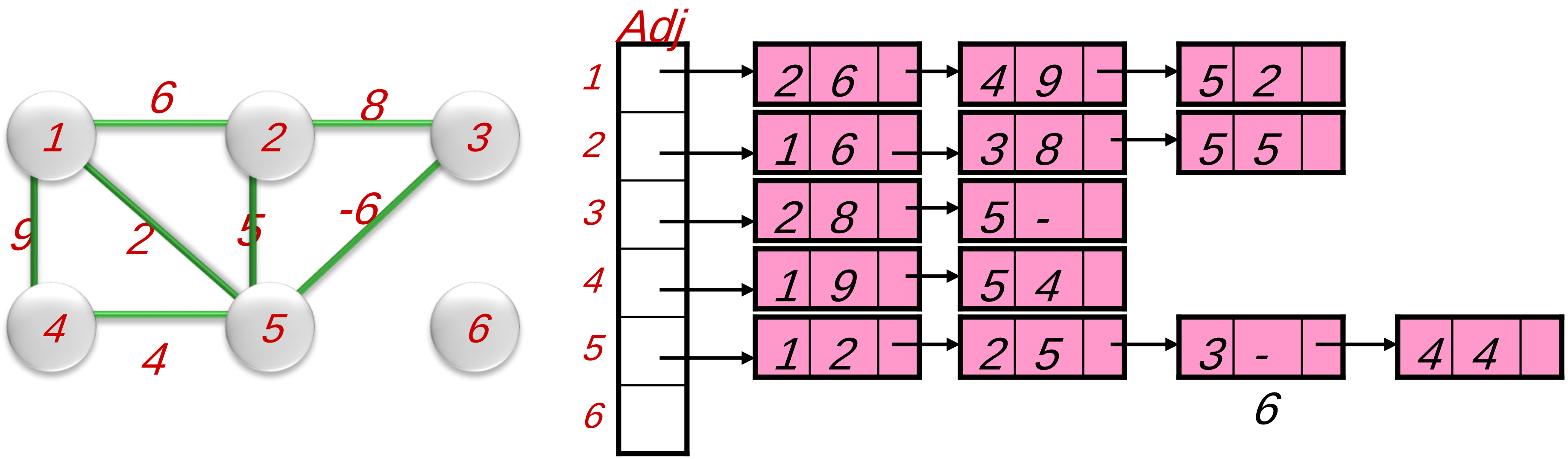
$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \left[\begin{array}{cccccc} 0 & 6 & \infty & 9 & 2 & \infty \\ 6 & 0 & 8 & \infty & 5 & \infty \\ \infty & 8 & 0 & \infty & -6 & \infty \\ 9 & \infty & \infty & 0 & 4 & \infty \\ 2 & 5 & -6 & 4 & 0 & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 \end{array}\right] \end{array}$$

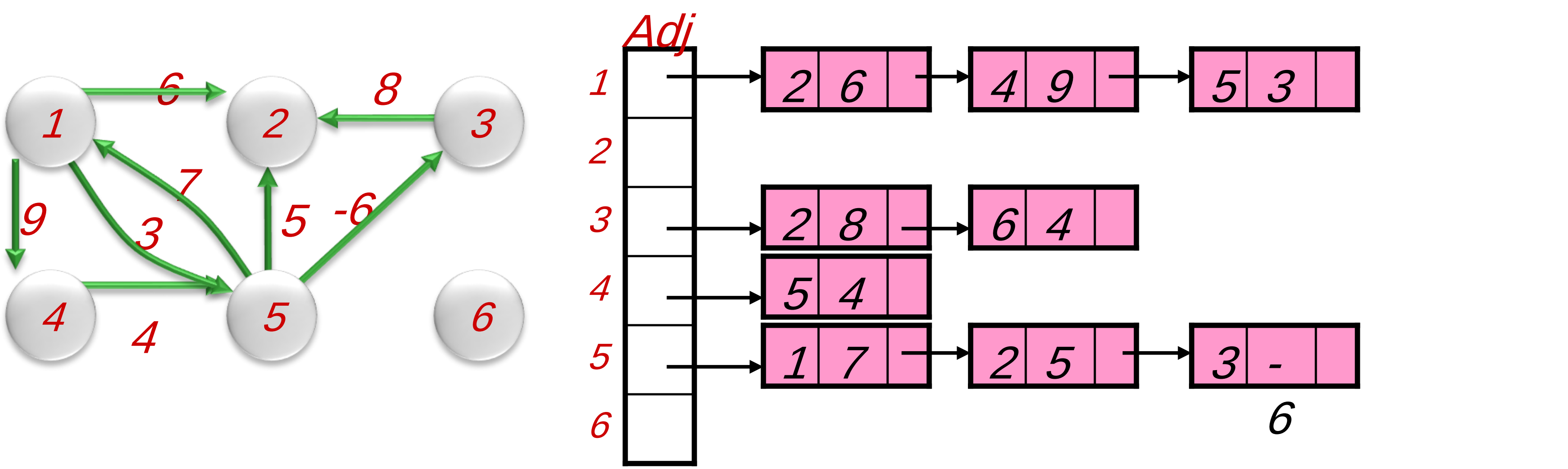


$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \left[\begin{array}{cccccc} 0 & 6 & \infty & 9 & 3 & \infty \\ \infty & 0 & \infty & \infty & \infty & \infty \\ \infty & 8 & 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & 0 & 4 & \infty \\ 7 & 5 & -6 & \infty & 0 & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 \end{array}\right] \end{array}$$

(Weighted) Adjacency List Structure

$$\mathrm{Adj}[i] = \{\, \langle j, w(i,j) \rangle \ \mid \ (i,j) \in E(G) \,\}, \quad \text{for } i \in V(G).$$

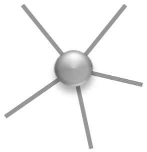




The Hand-Shaking Lemma

Vertex $v \in V(G)$: degree (or valance) , in-degree, out-degree

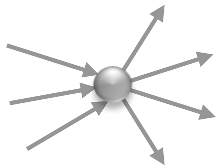
Undirected G: $deg(v) = |\{ u \mid (v,u) \in E(G) \}| = |Adj[v]|$



Digraph G: $outdeg(v) = |\{ u \mid (v,u) \in E(G) \}| = |Adj[v]|$

$indeg(v) = |\{ u \mid (u,v) \in E(G) \}|$

$deg(v) = outdeg(v) + indeg(v)$



The Hand-Shaking Lemma:

For any graph (directed or undirected) we have:

$$\sum_{v \in V(G)} \deg(v) = 2\,|E|\,.$$

For any directed graph we also have:

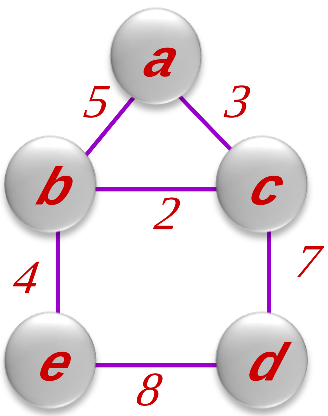
$$\sum_{v \in V(G)} \mathrm{indeg}(v) = \sum_{v \in V(G)} \mathrm{outdeg}(v) = |E|\,.$$

Adjacency Matrix vs Adjacency List Structure

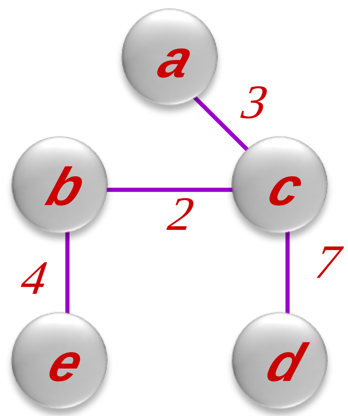
complexity		Adjacency Matrix	Adjacency List
# memory cells	Space	$O(V^2)$	$O(V + E)$
Initialize structure	Time	$O(V^2)$	$O(V + E)$
Scan (incident edges of) all vertices		$O(V^2)$	$O(V + E)$
List vertices adjacent to $u \in V(G)$		$O(V)$	$O(\|Adj[u]\|)$
Is $(u,v) \in E(G)$?		$O(1)$	$O(\|Adj[u]\|)$

Minimum Spanning Tree (MST)

- A *spanning tree* for a connected, undirected graph, $G=(V, E)$ is
  - a connected subgraph of G that forms an undirected tree incident with each vertex.
- In a weighted graph $G=(V, E, W)$,
  - the weight of a subgraph is the sum of the weights of the edges in the subgraph.
- A *minimum spanning tree* (MST) for a weighted graph is
  - a spanning tree with the minimum weight.

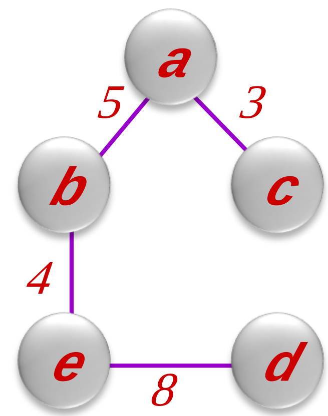


G



Minimum spanning tree T

Cost(T) = 3+2+4+7 = 16

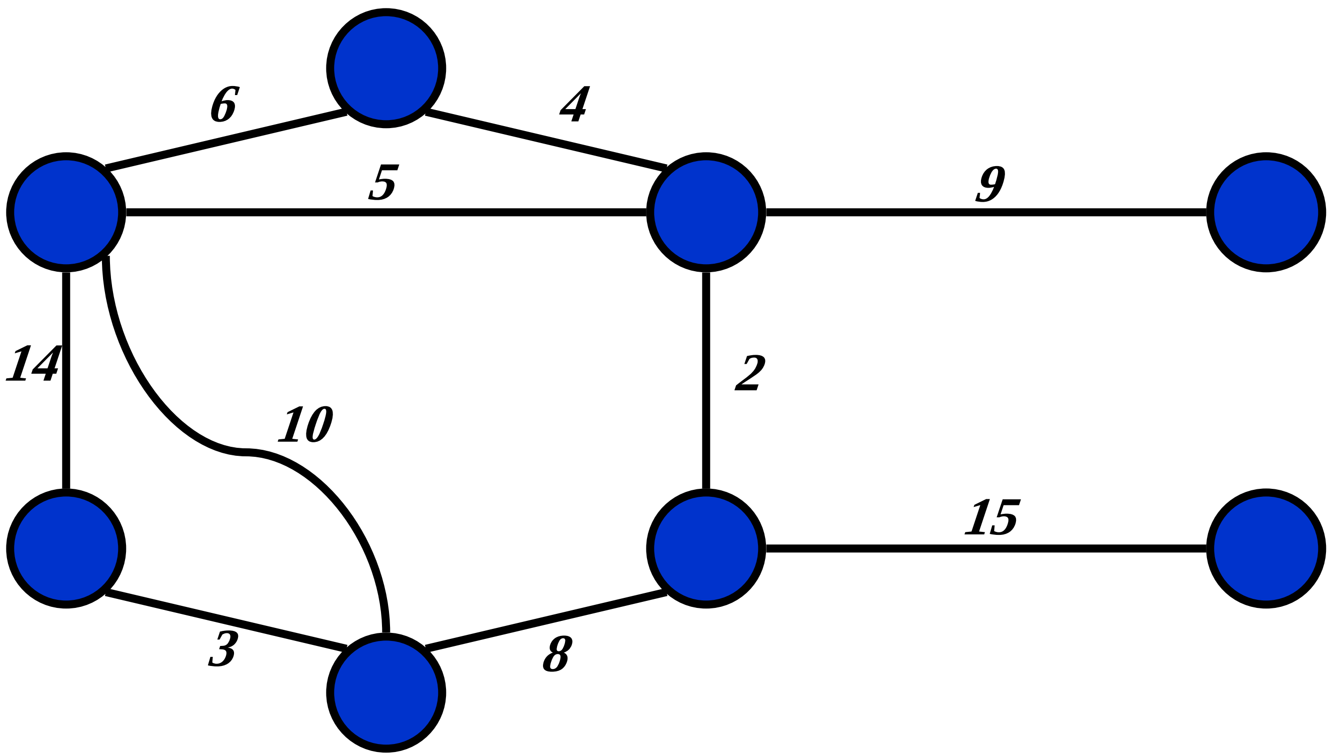


Another spanning tree T'

Cost(T') = 3+5+4+8 = 20

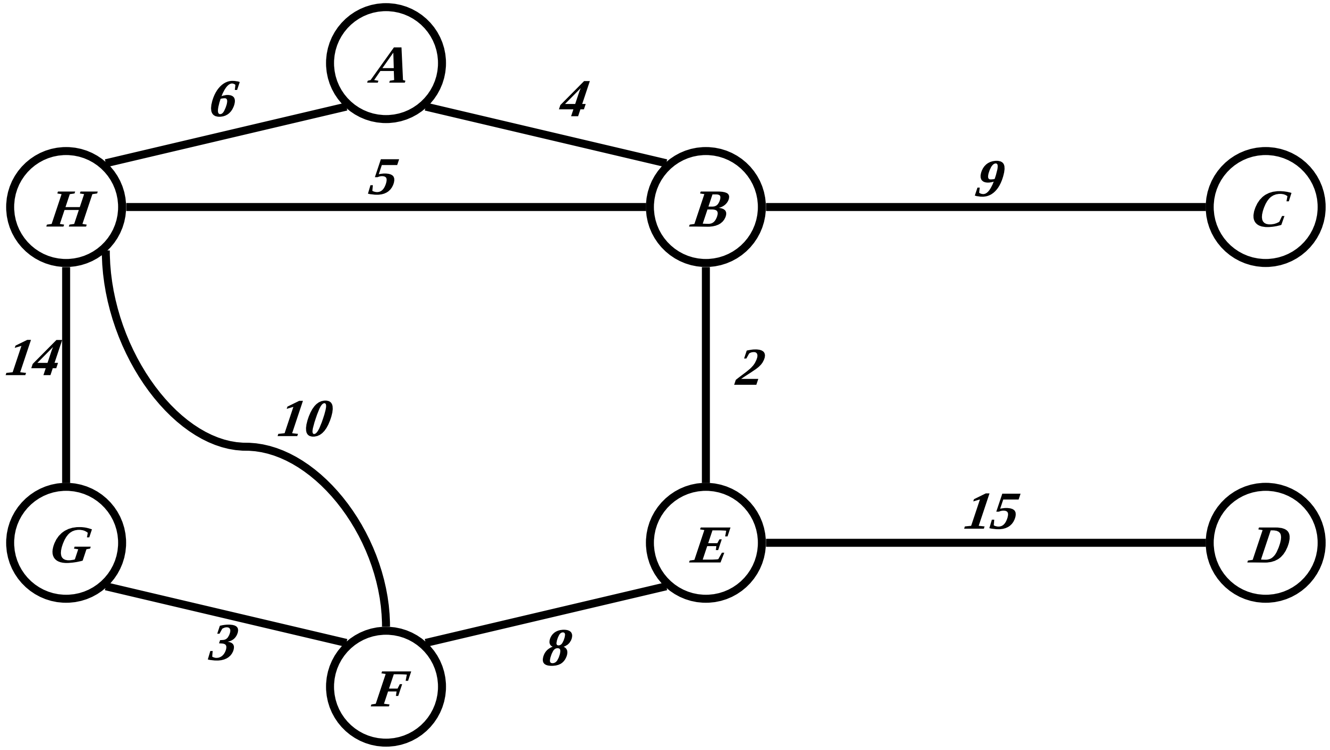
Minimum Spanning Tree

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



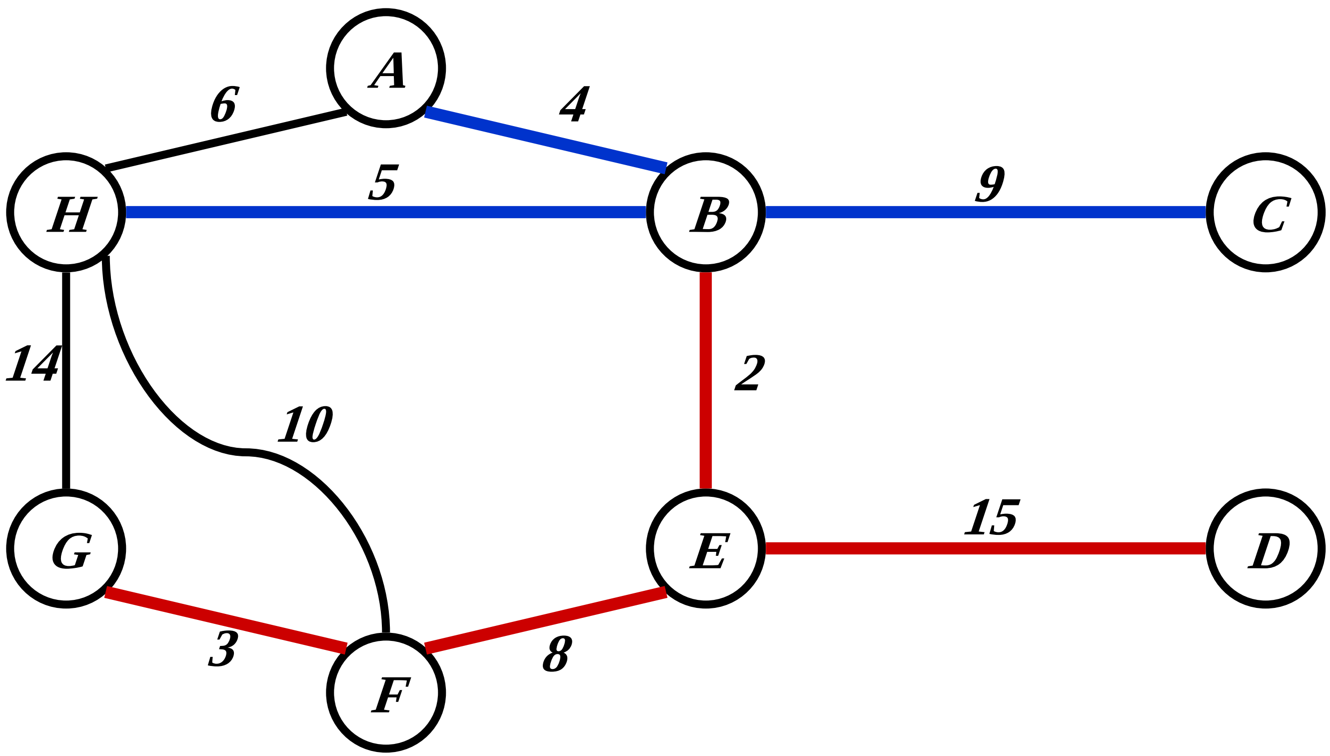
Minimum Spanning Tree

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



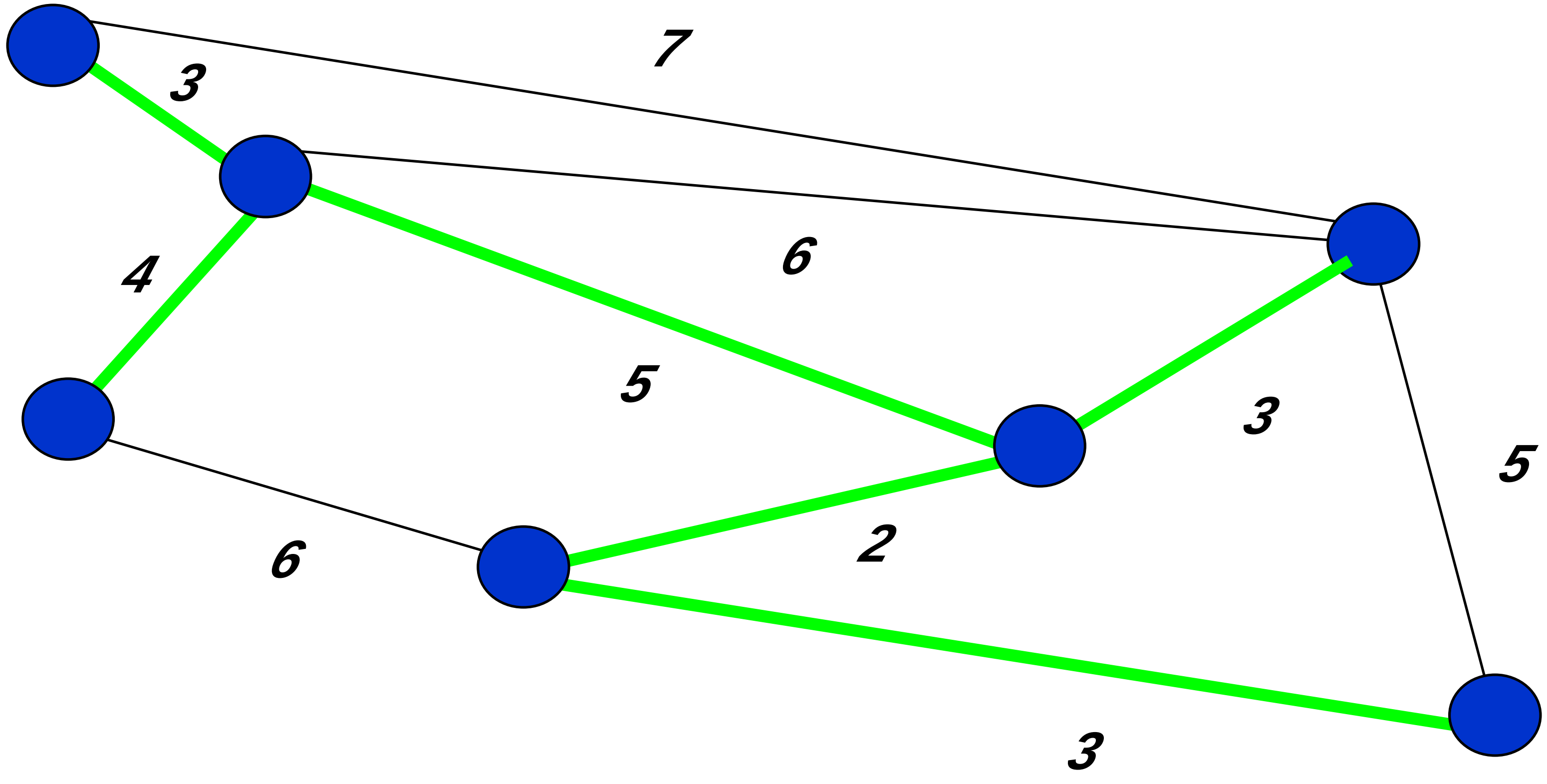
Minimum Spanning Tree

- Answer:



Another Example

- Given a weighted graph $G=(V, E, W)$, find a MST of G



Finding a MST

- MSTs satisfy the *optimal substructure* property: an optimal tree is composed of optimal subtrees
- Principal greedy methods: algorithms by Prim and Kruskal
- Prim
  - Grow a single tree by repeatedly adding the least cost edge that connects a vertex in the existing tree to a vertex not in the existing tree
    - Intermediary solution is a subtree
- Kruskal
  - Grow a tree by repeatedly adding the least cost edge that does not introduce a cycle among the edges included so far
    - Intermediary solution is a spanning forest

Prime’s Algorithm (High-Level Pseudocode)

- Prime(G)

  //Input: A weighted connected graph G = <V, E>

  //Output: E_T --- the set of edges composing MST of G

  $V_T = \{v_0\}$

  $E_T = \varnothing$

  for i = 1 **to** |V| - 1 **do**

  find a minimum-weight edge e* = (u*, v*) among all the edges (u, v) such that u is in V_T and v is in $V-V_T$

  $V_T = V_T \cup \{v^*\}$

  $E_T = E_T \cup \{e^*\}$

  return E_T

Prime's Algorithm (High-Level Pseudocode)

- Prime(G)

  //Input: A weighted connected graph G = <V, E>

  //Output: E_T --- the set of edges composing MST of G

  $V_T = \{v_0\}$

  $E_T = \varnothing$

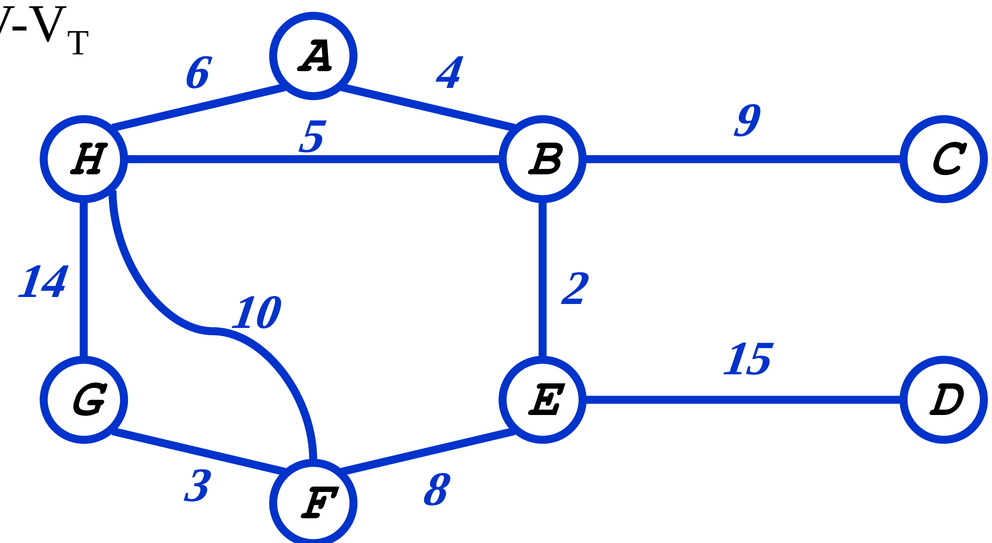
  for i = 1 **to** |V| - 1 **do**

  find a minimum-weight edge e* = (u*, v*) among all the edges (u, v) such that u is in V_T and v is in $V\text{-}V_T$

  $V_T = V_T \cup \{v^*\}$

  $E_T = E_T \cup \{e^*\}$

  return E_T



Prim's Algorithm

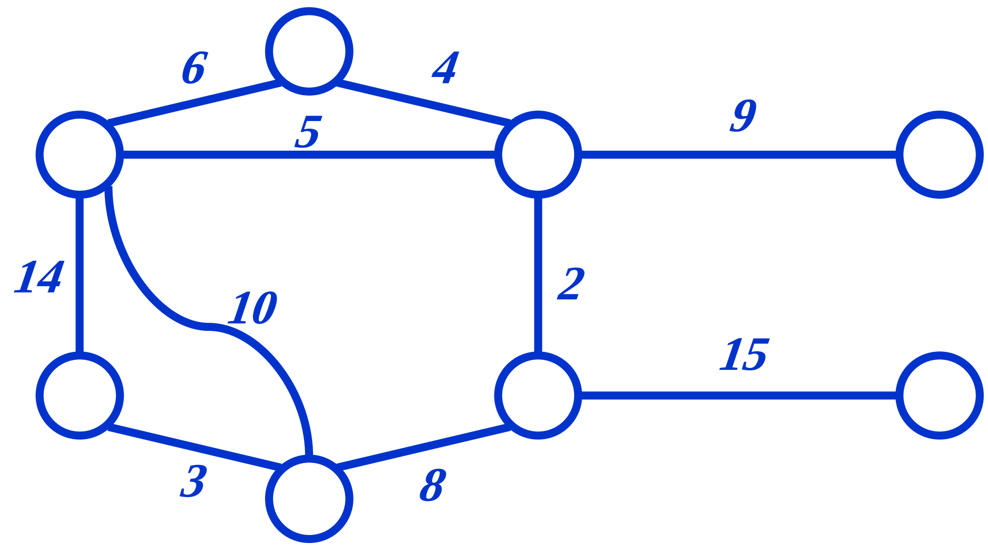
```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```

Grow a single tree by repeatedly adding the least cost edge that connects a vertex in the existing tree to a vertex not in the existing tree

Intermediary solution is a subtree

Prim's Algorithm

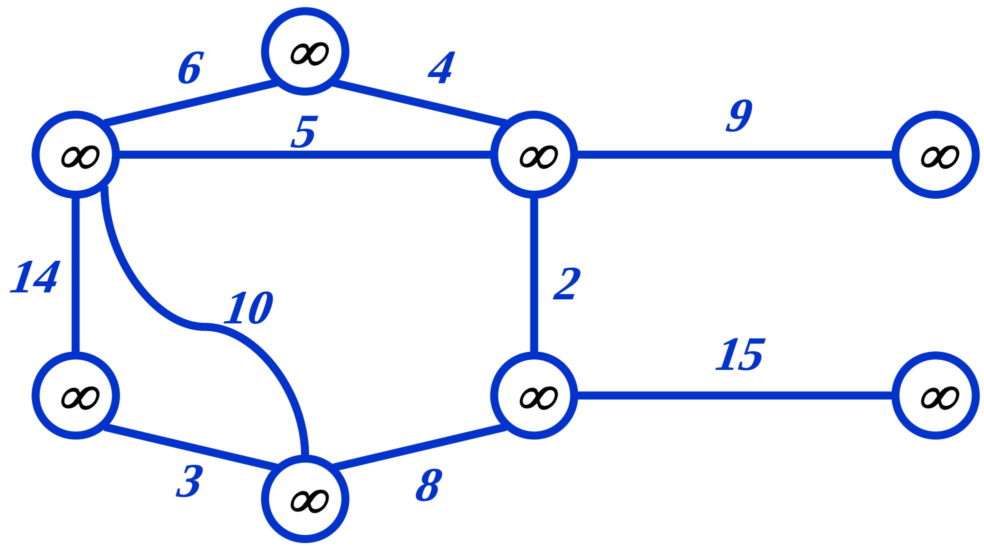
```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```



Run on example graph

Prim's Algorithm

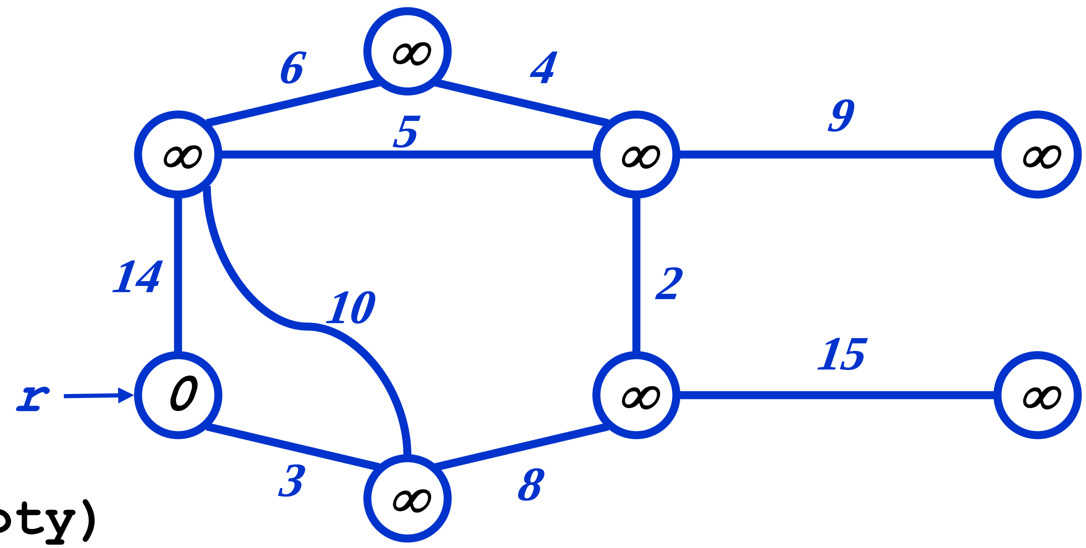
```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```



Run on example graph

Prim's Algorithm

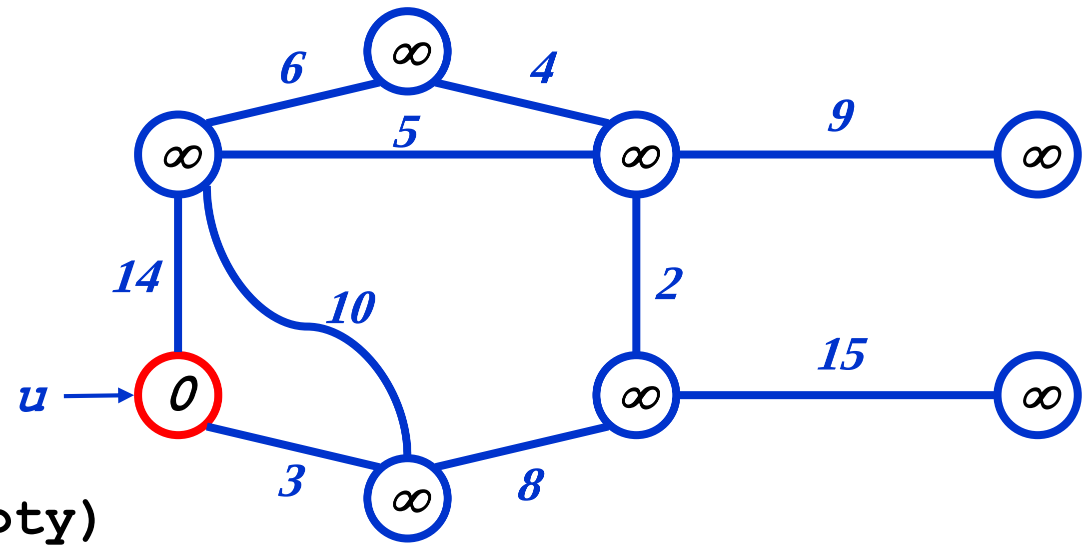
```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```



Pick a start vertex r

Prim's Algorithm

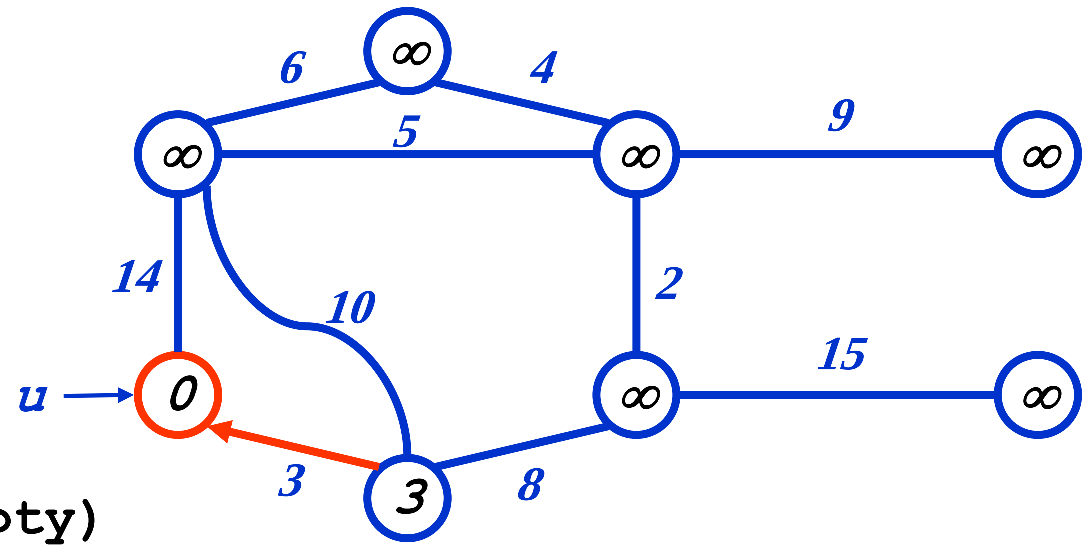
```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```



Red vertices have been removed from Q

Prim's Algorithm

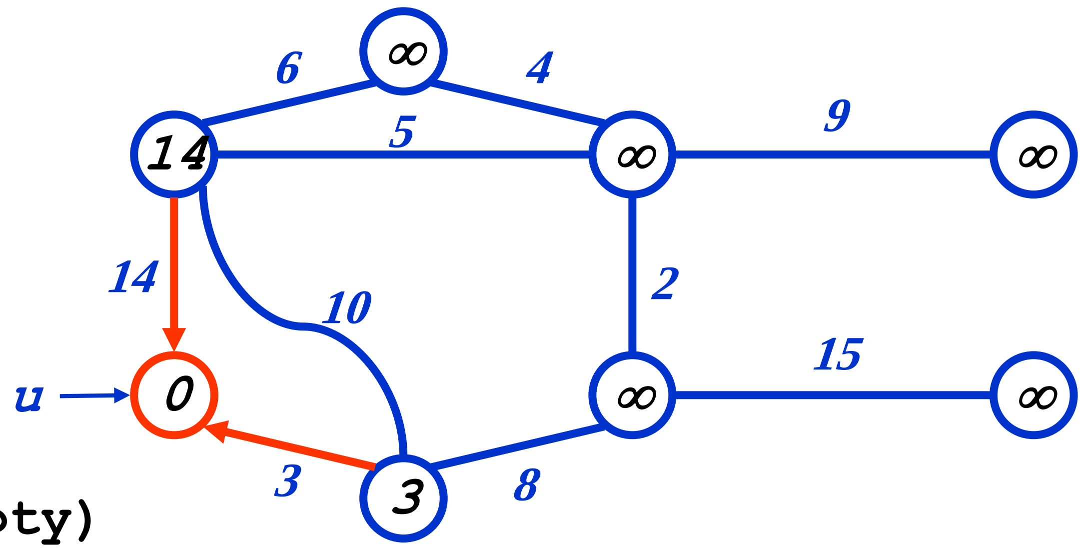
```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```



Red arrows indicate parent pointers

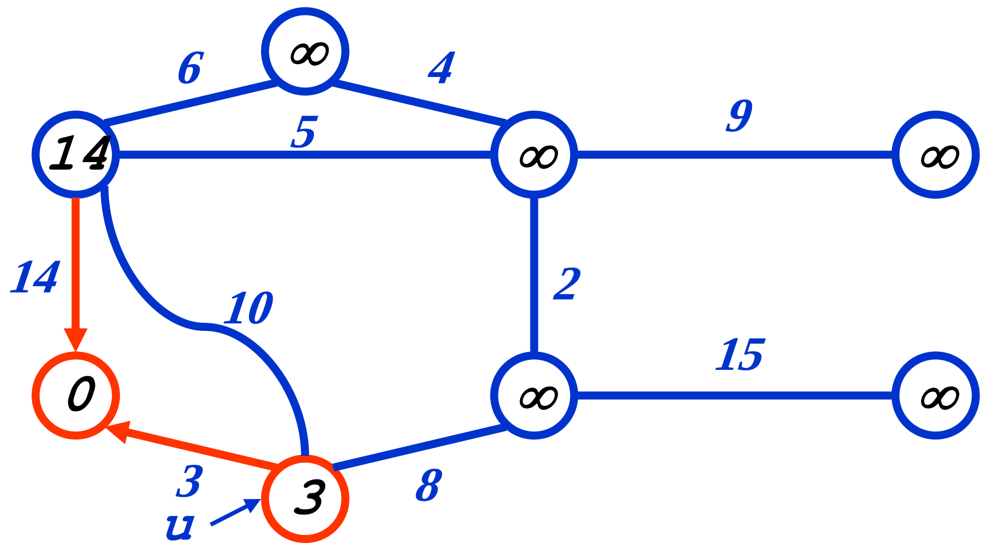
Prim's Algorithm

```
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);
```



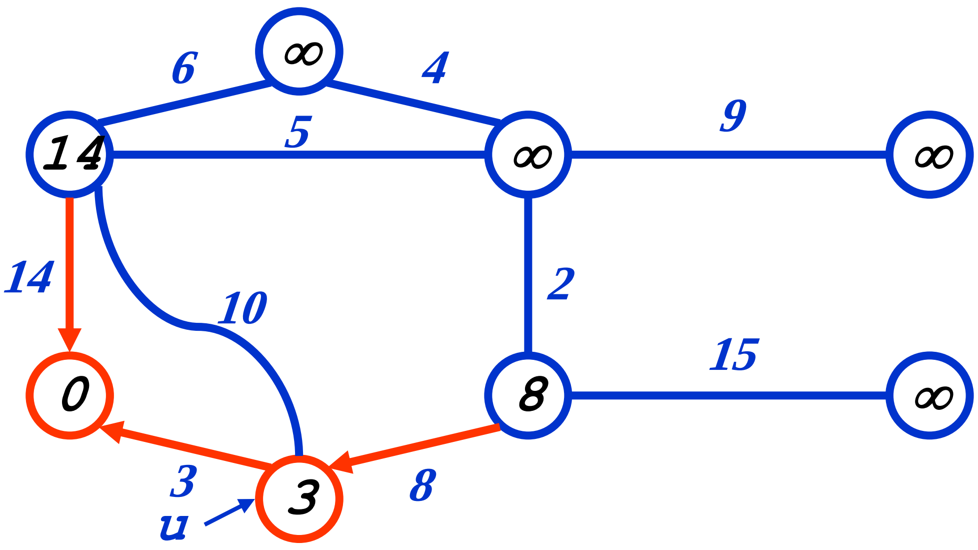
Prim's Algorithm

```
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);
```



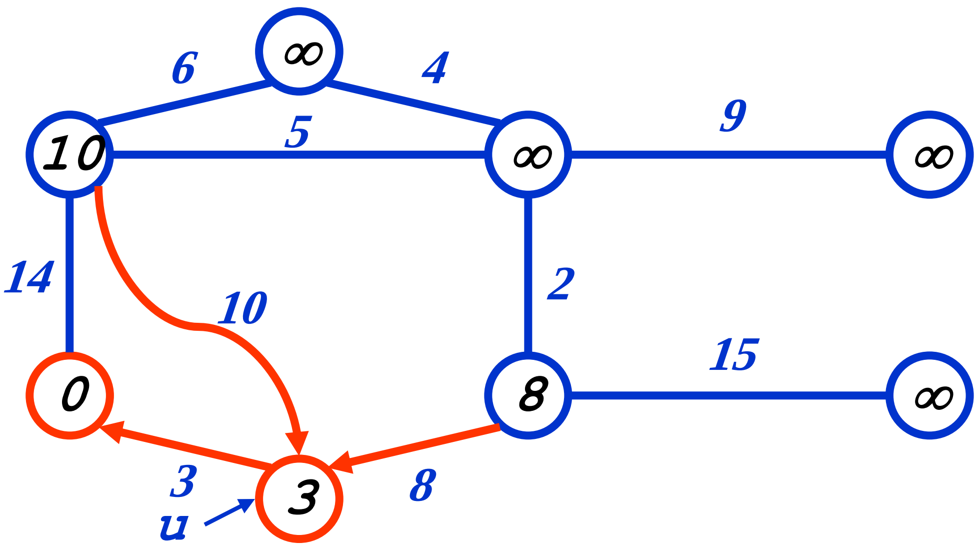
Prim's Algorithm

```
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);
```



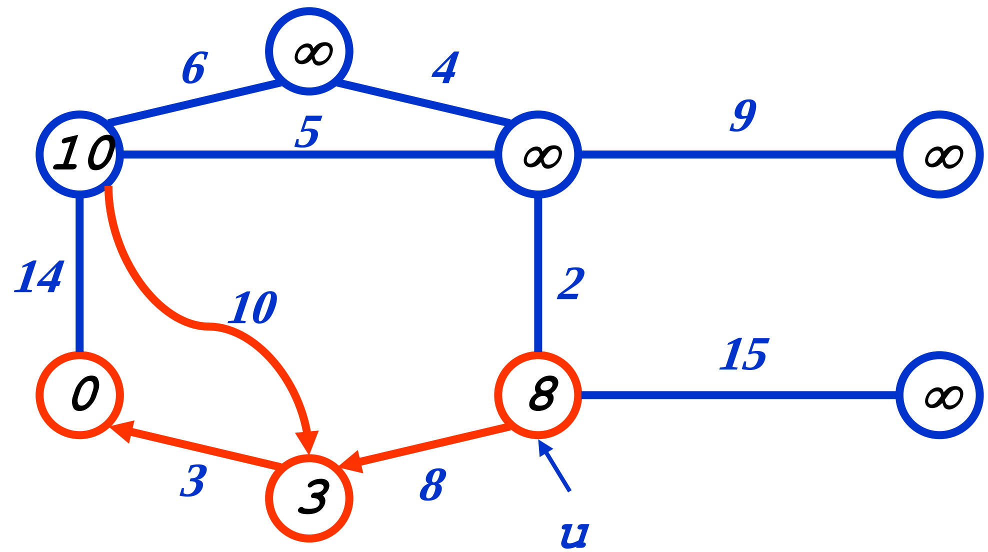
Prim's Algorithm

```
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);
```



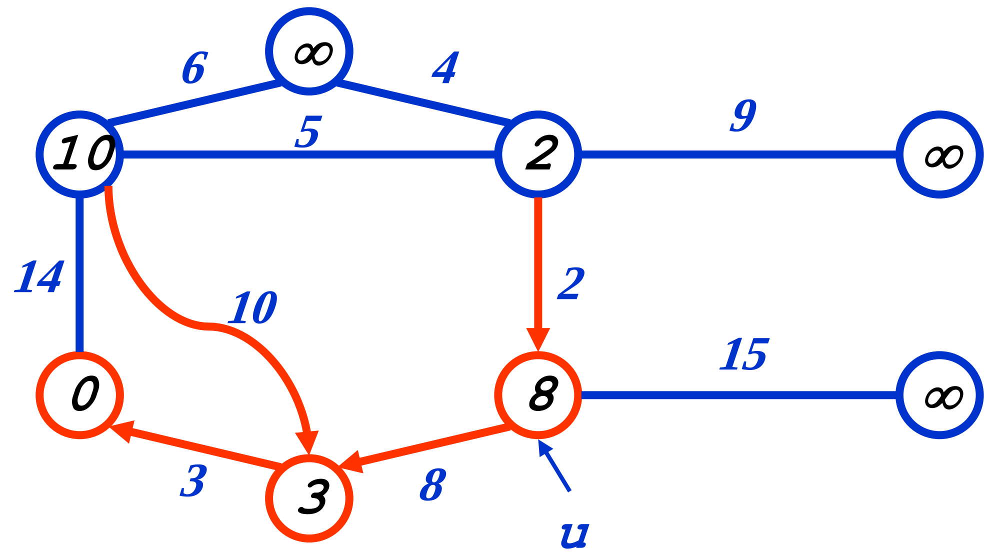
Prim's Algorithm

```
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);
```



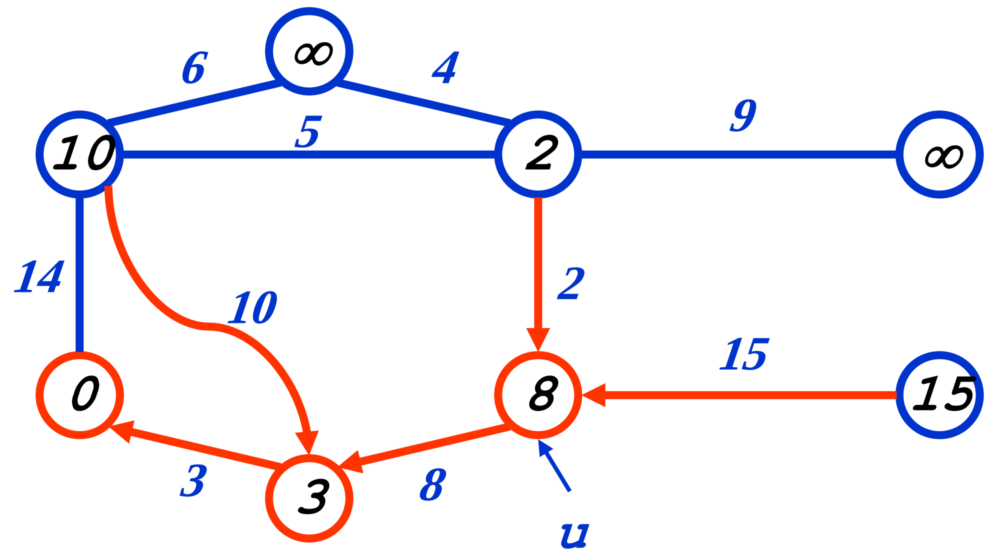
Prim's Algorithm

```
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);
```



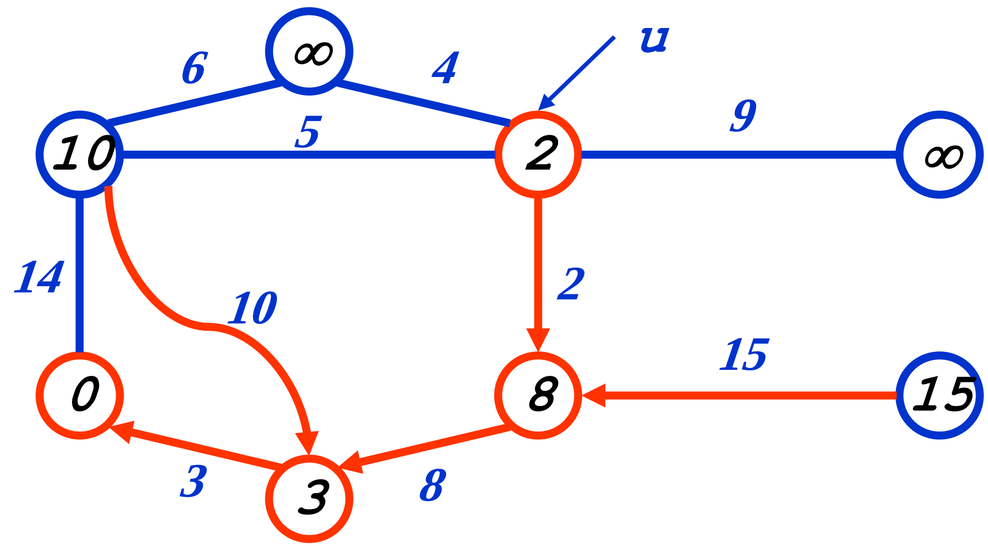
Prim's Algorithm

```
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);
```



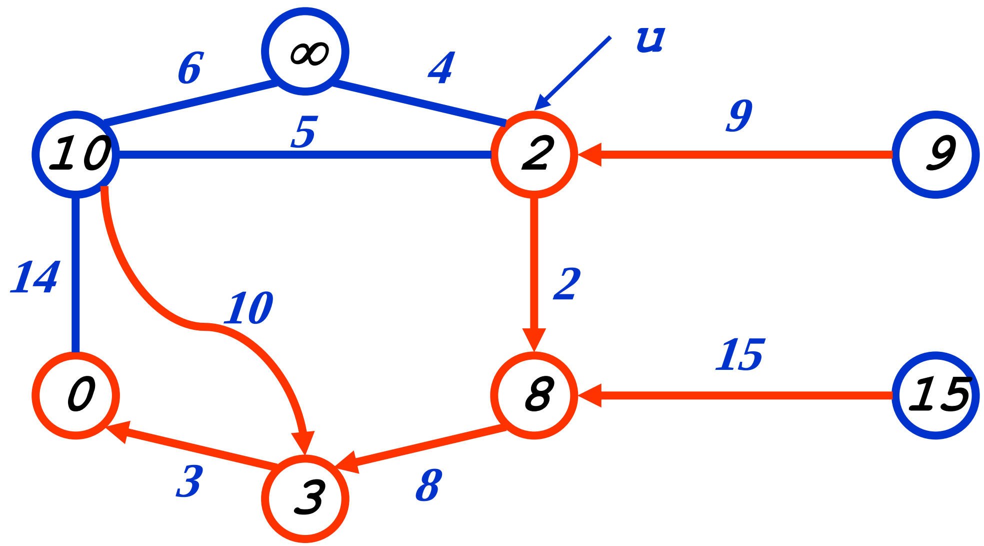
Prim's Algorithm

```
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);
```



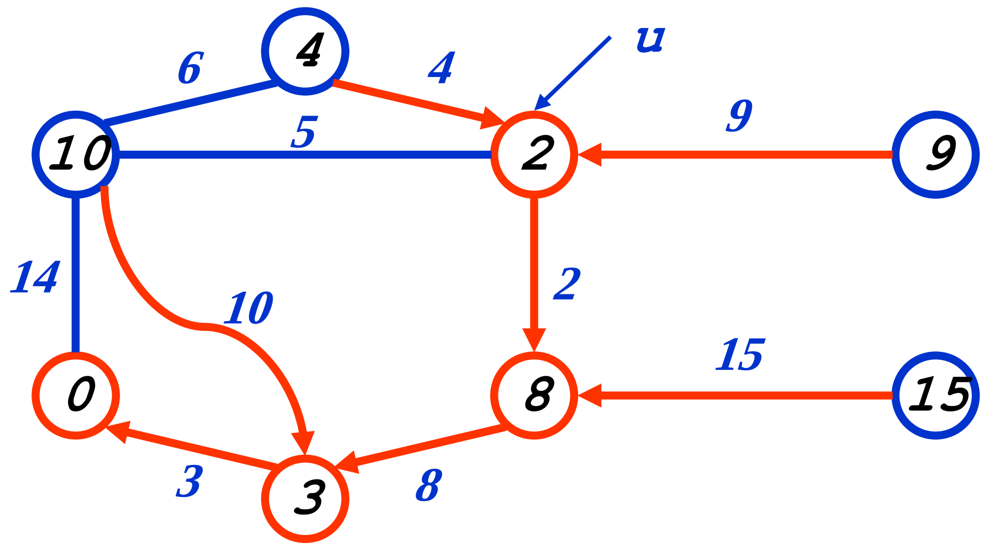
Prim's Algorithm

```
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);
```



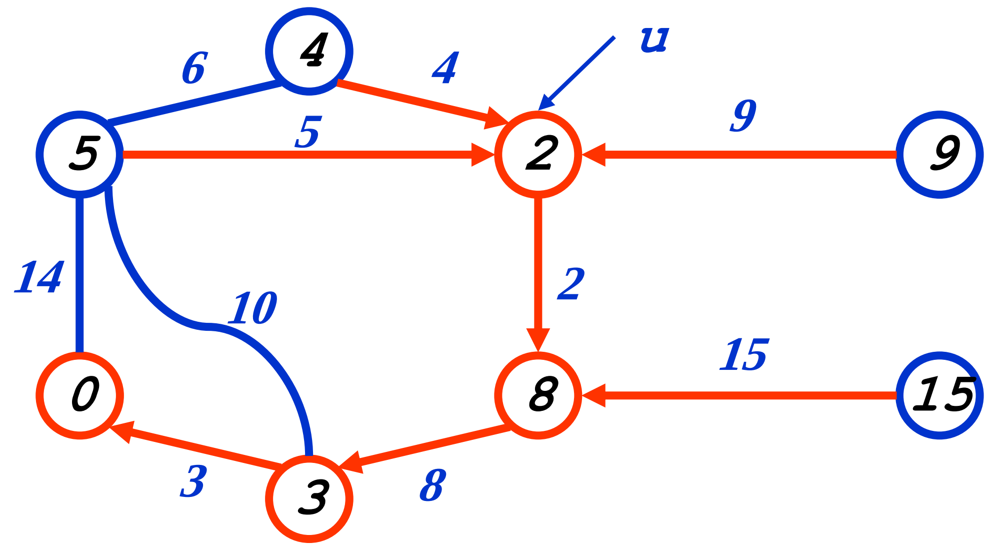
Prim's Algorithm

```
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);
```



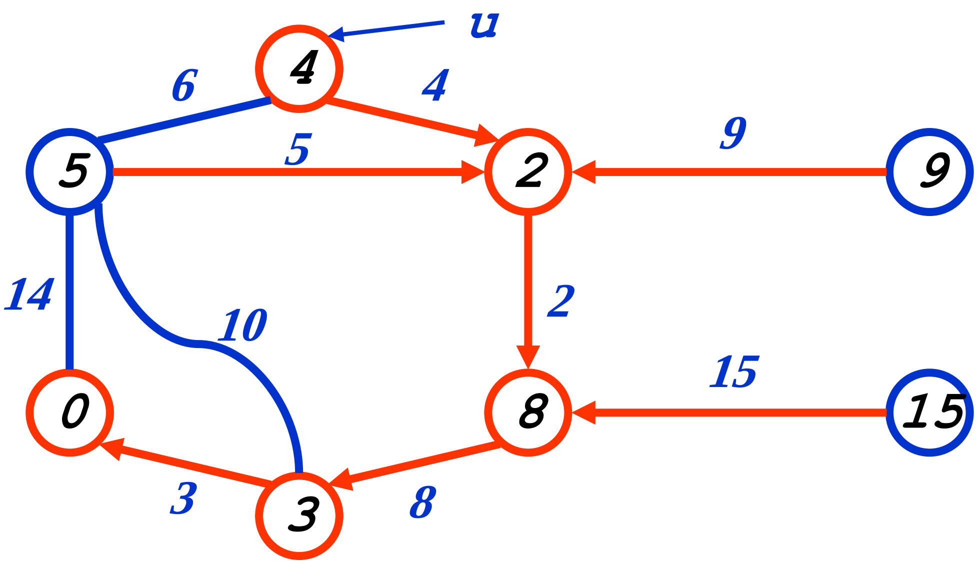
Prim's Algorithm

```
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);
```



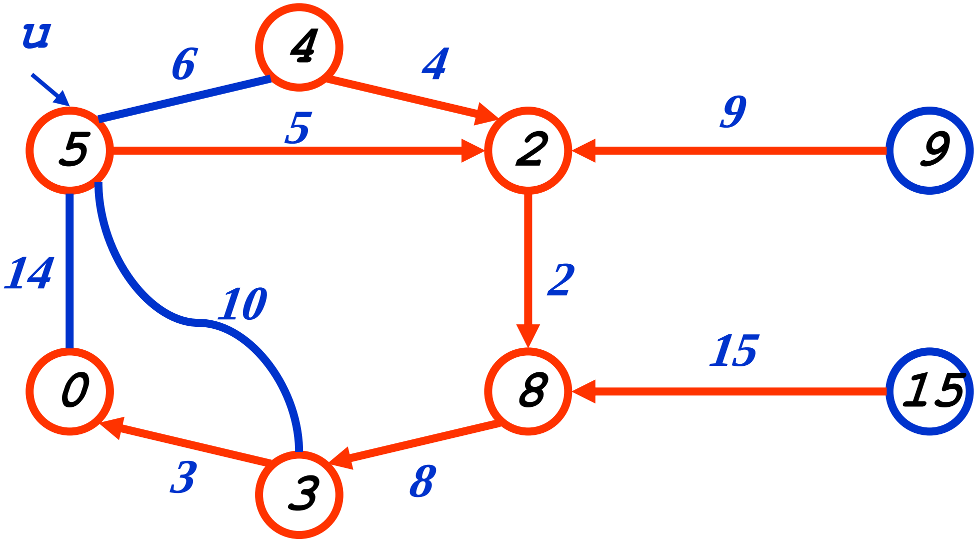
Prim's Algorithm

```
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);
```



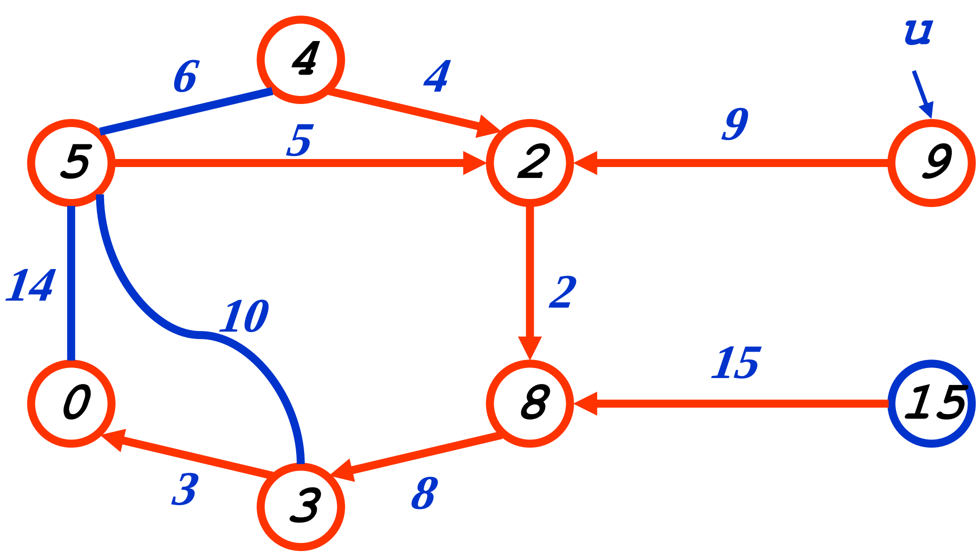
Prim's Algorithm

```
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);
```



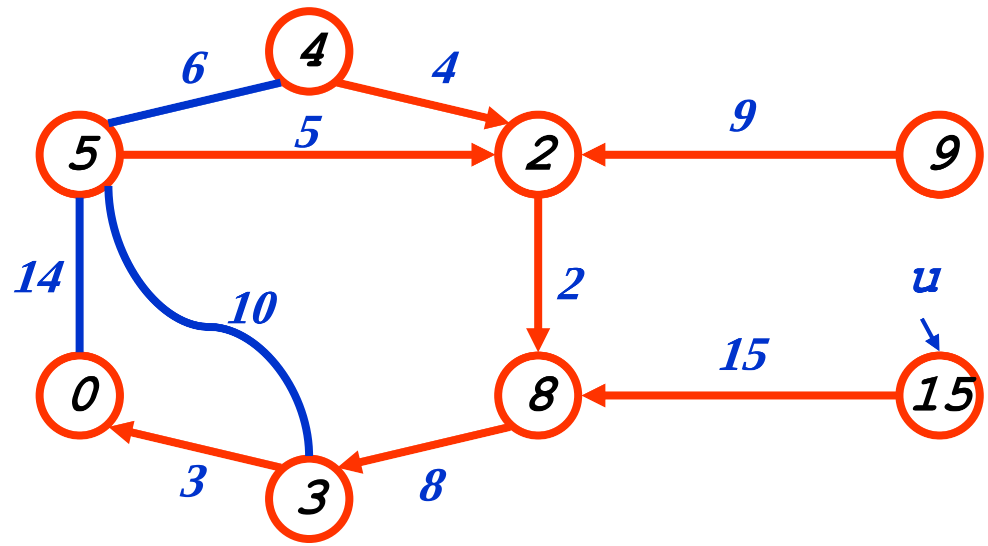
Prim's Algorithm

```
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));
```

delete the smallest element from the min-heap

decrease an element's value in the min-heap

(outline an efficient algorithm for it)

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?

There are $n=|V|$ ExtractMin calls and $m=|E|$ DecreaseKey calls.

The priority Q implementation has a large impact on performance.

E.g., $O((n+m)\lg n) = O(m \lg n)$ using min-heap for Q (for connected graph, $n - 1 \leq m$)

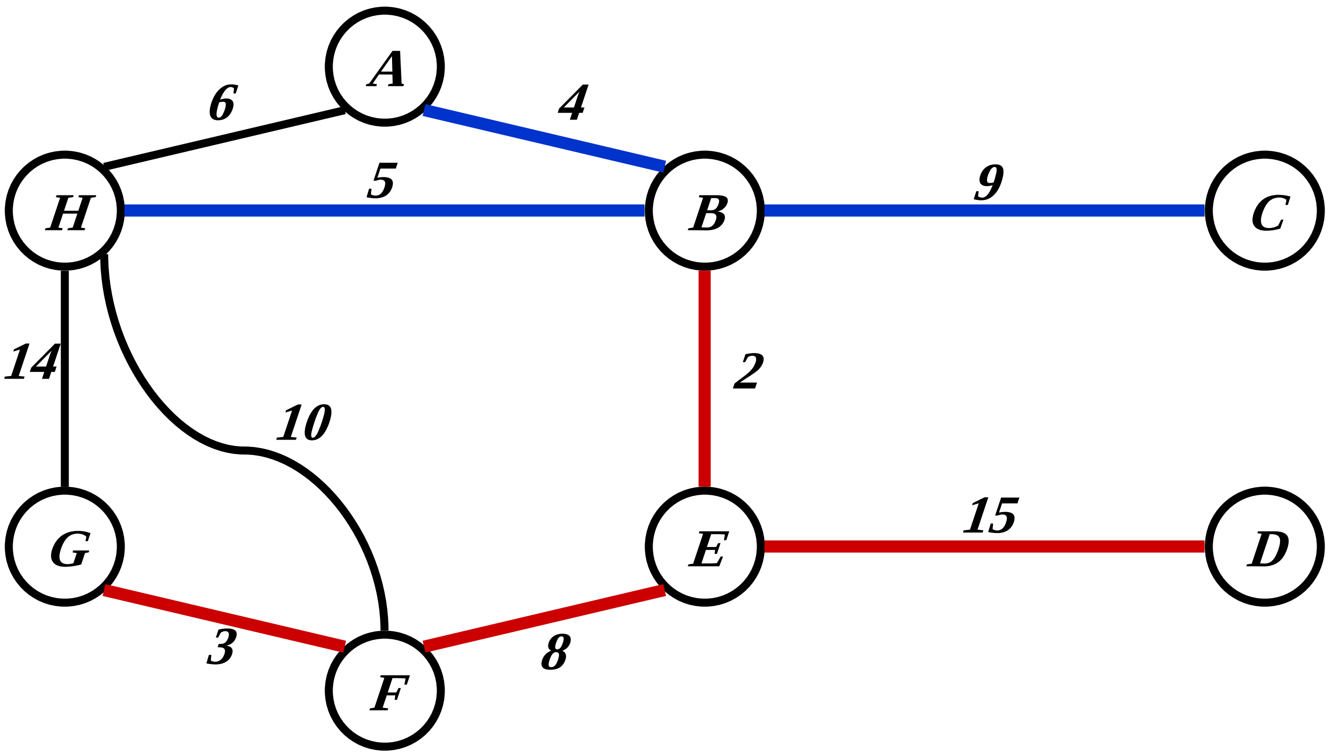
Finding a MST

- Principal greedy methods: algorithms by Prim and Kruskal

- Prim
  - Grow a single tree by repeatedly adding the least cost edge that connects a vertex in the existing tree to a vertex not in the existing tree
    - Intermediary solution is a subtree

- Kruskal
  - Grow a tree by repeatedly adding the least cost edge that does not introduce a cycle among the edges included so far
    - Intermediary solution is a spanning forest

MST Applications?



MST Applications

- Network design
  - telephone, electrical power, hydraulic, TV cable, computer, road
- MST gives a minimum-cost network connecting all sites
- MST: the most economical construction of a network

The End