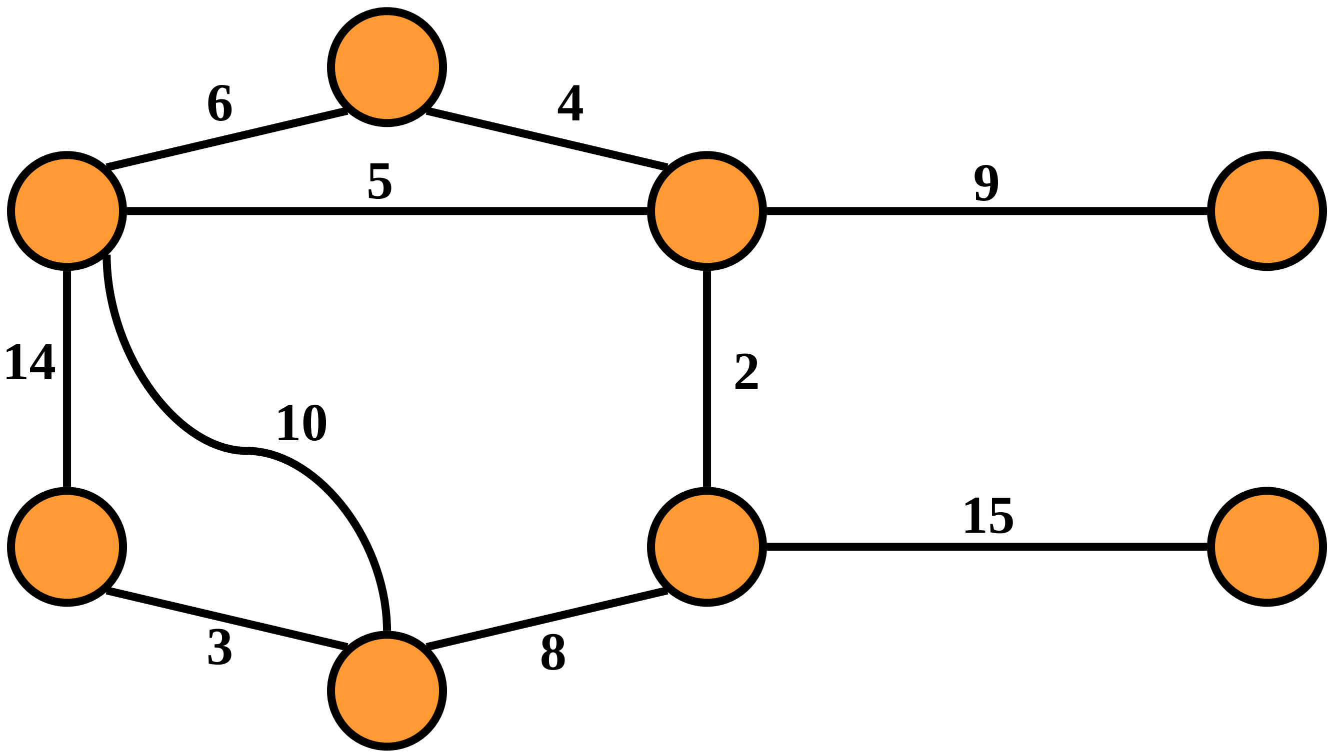
Chapter 23
Minimum Spanning Trees

Algorithm Analysis

School of CSEE

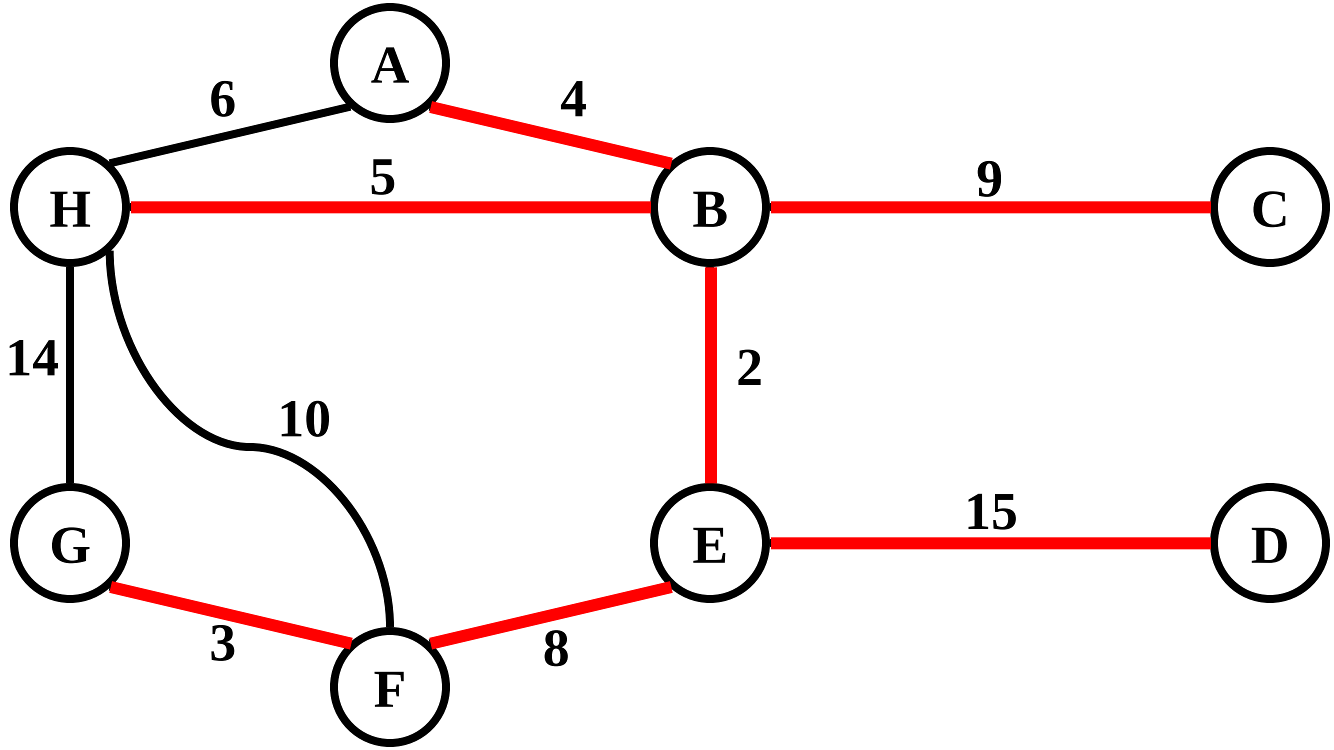
Minimum Spanning Tree

- 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 below graph?*

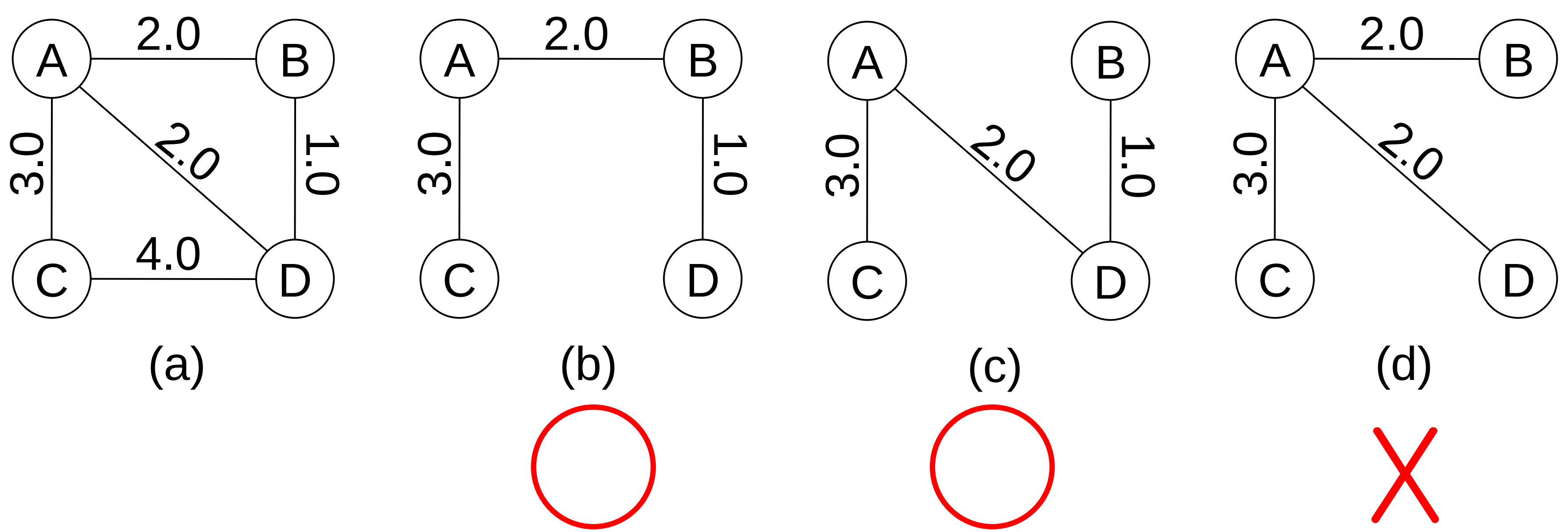


Minimum spanning tree

- Undirected graph $G = (V, E)$
- Weight $w(u,v)$ on each edge $(u,v) \in E$
- Spanning tree of G' is a minimal subgraph of G such that
  - $V(G') = V(G)$ and G' is connected.
  - Any connected graph with n vertices must have at least n-1 edges. All connected graphs with n-1 edges are trees.
- Find $T \subseteq E$ s.t.
  - T connects all vertices (T is a spanning tree), and
  - $w(T) = \sum_{(u,v) \in E} w(u,v)$ is minimized.

Minimum spanning tree

- A spanning tree whose weight is minimum over all spanning trees is called a minimum spanning tree or MST.
  – Has *n* -1 edges, no cycle, might not be unique



Minimum spanning tree

- Example)

  Interconnect *n* pins with *n*-1 wires, each connecting two pins so that we use the least amount of wire.

- We'll look at two greedy algorithms.
  - Kruskal's algorithm
  - Prim's algorithm

Building up the solution

- Build a set *A* of edges.
- Initially, *A* is empty.
- As we add edges to *A*, maintain a loop invariant : *A* is a subset of some MST.
- Add only edges that maintain the invariant.
  - If *A* is a subset of some MST, an edge (*u*,*v*) is *safe* for *A* if and only if *A* U { (*u*,*v*) } is also a subset of some MST.
  - So, we will add only safe edges.

Generic MST algorithm

GENERIC-MST(*G*, *w*)

$A = \varnothing$

while A is not a spanning tree

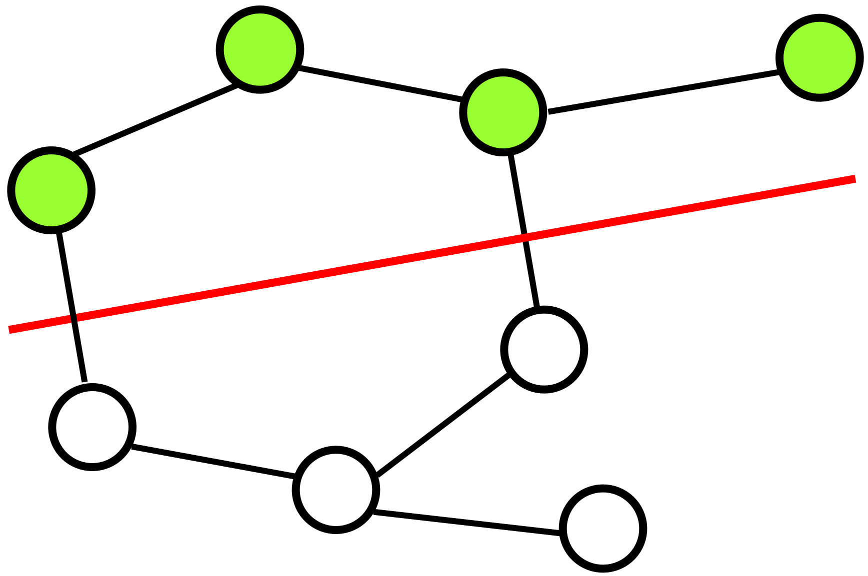
    do find an edge (u,v) that is safe for A

        $A = A \cup \{ (u,v) \}$

return A

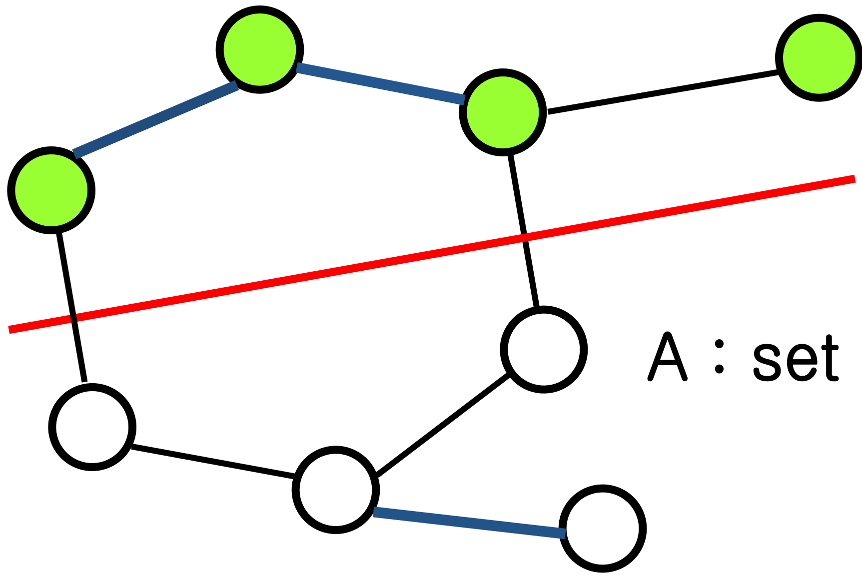
Finding a safe edge

- Let $S \subset V$ and $A \subseteq E$.
  - A *cut* $(S, V - S)$ is a partition of vertices into disjoint sets S and $V - S$.
  - Edge $(u,v) \in E$ *crosses* cut $(S, V - S)$ if one endpoint is in S and the other is in $V - S$.



Finding a safe edge

- Let $S \subset V$ and $A \subseteq E$.
  - A cut *respects* A if and only if no edge in A crosses the cut.
  - An edge is a *light edge* crossing a cut if and only if its weight is minimum over all edges crossing the cut. For a given cut, there can be more than one light edge crossing it.



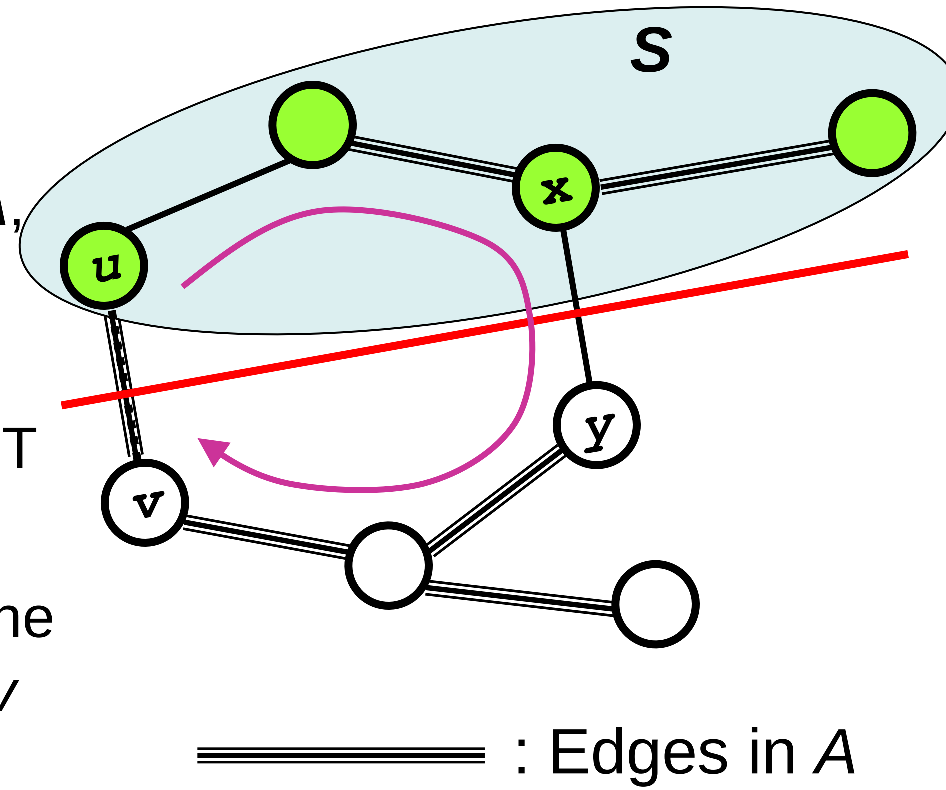
A : set of blue edges

Theorem

Let A be a subset of E in some MST, $(S, V - S)$ be a cut that respects A, and (u, v) be a light edge crossing $(S, V - S)$. Then, (u, v) is safe for A.

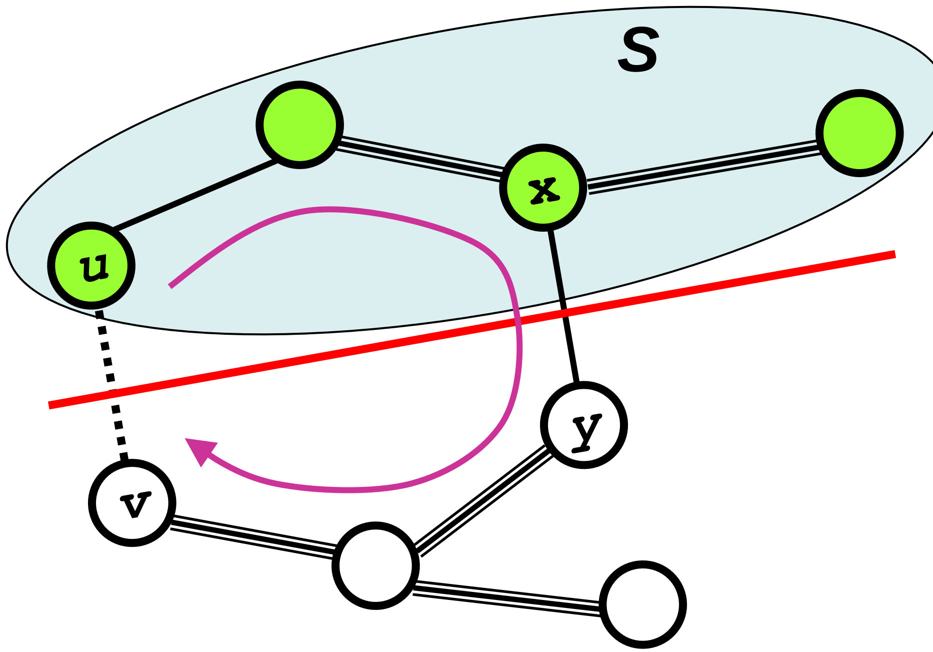
Proof)

1. Let T be a MST that includes A, and assume that T does not contain the light edge (u, v).
2. We shall construct another MST T' that includes $A \cup \{ (u,v) \}$.
3. Edge (u,v) forms a cycle with the edges on the path p from u to v in T.



Theorem

4. Since u and v are on opposite sides of the cut $(S, V - S)$ there is at least one edge in T on the path p that also crosses the cut. Let (x,y) be any such edge.



5. Removing (x,y) breaks T into two components. And adding (u,v) reconnects them to form a new spanning tree T'.

$$w(T') = w(T) - w(x,y) + w(u,v)$$

$$\leq w(T)$$

Thus, T' must be a MST.

And since $A \cup \{ (u,v) \} \subseteq T'$, (u,v) is safe for A.

Corollary

Let A be a subset of E that is included in some minimum spanning tree for G, and let $C = (V_c, E_c)$ be a connected component (tree) in the forest $G_A = (V, A)$. If (u,v) is a light edge connecting C to some other component in G_A, then (u,v) is safe for A.

Proof) The cut $(Vc, V - Vc)$ respects A, and (u,v) is a light edge for this cut. Therefore, (u,v) is safe for A.

Basic idea of Kruskal's algorithm

- Sort edges into nondecreasing order by *w.*
- The algorithm maintains *A*, a forest of trees.
- Repeatedly merges two components into one by choosing the light edge that connects them.

  i.e.,

  1. Choose the light edge crossing the cut between them.

  2. (If it forms a cycle, the edge is discarded.)

- What is the design strategy of Kruskal's algorithm?

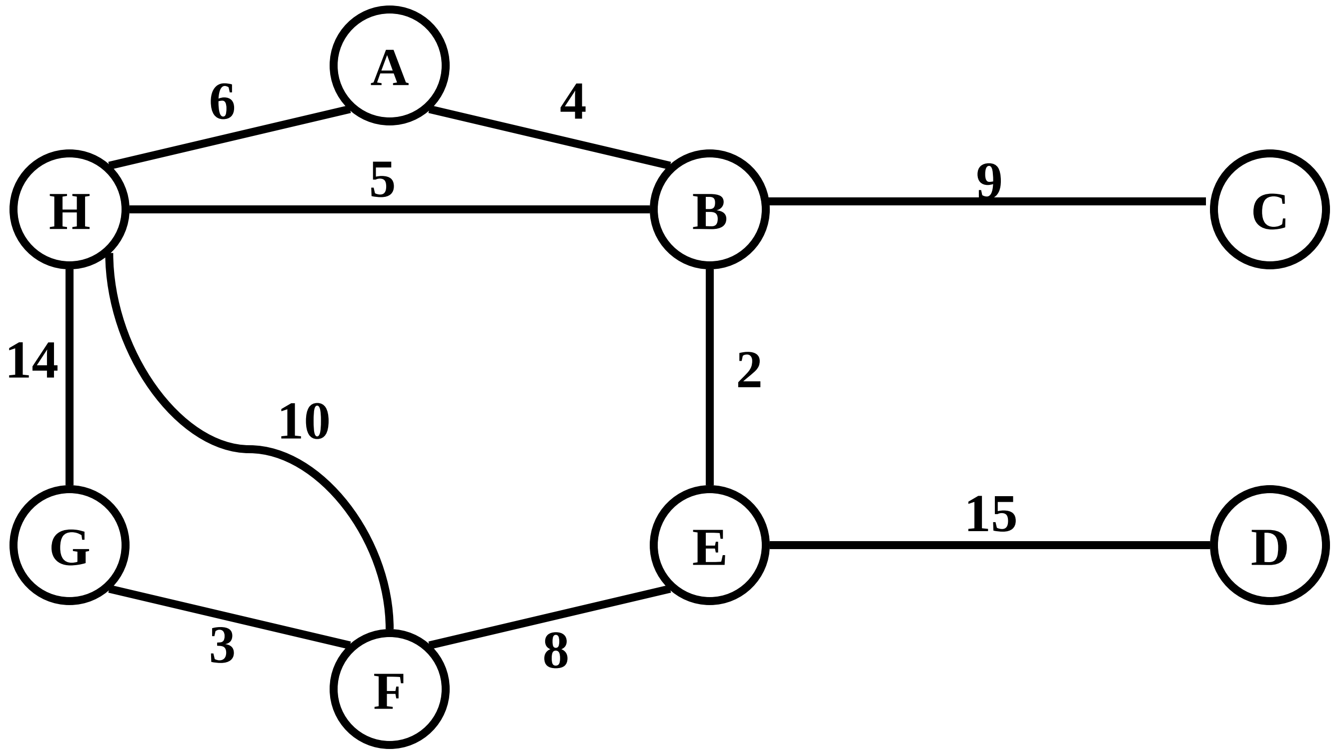
  - Greedy !!

Specific pseudo-code

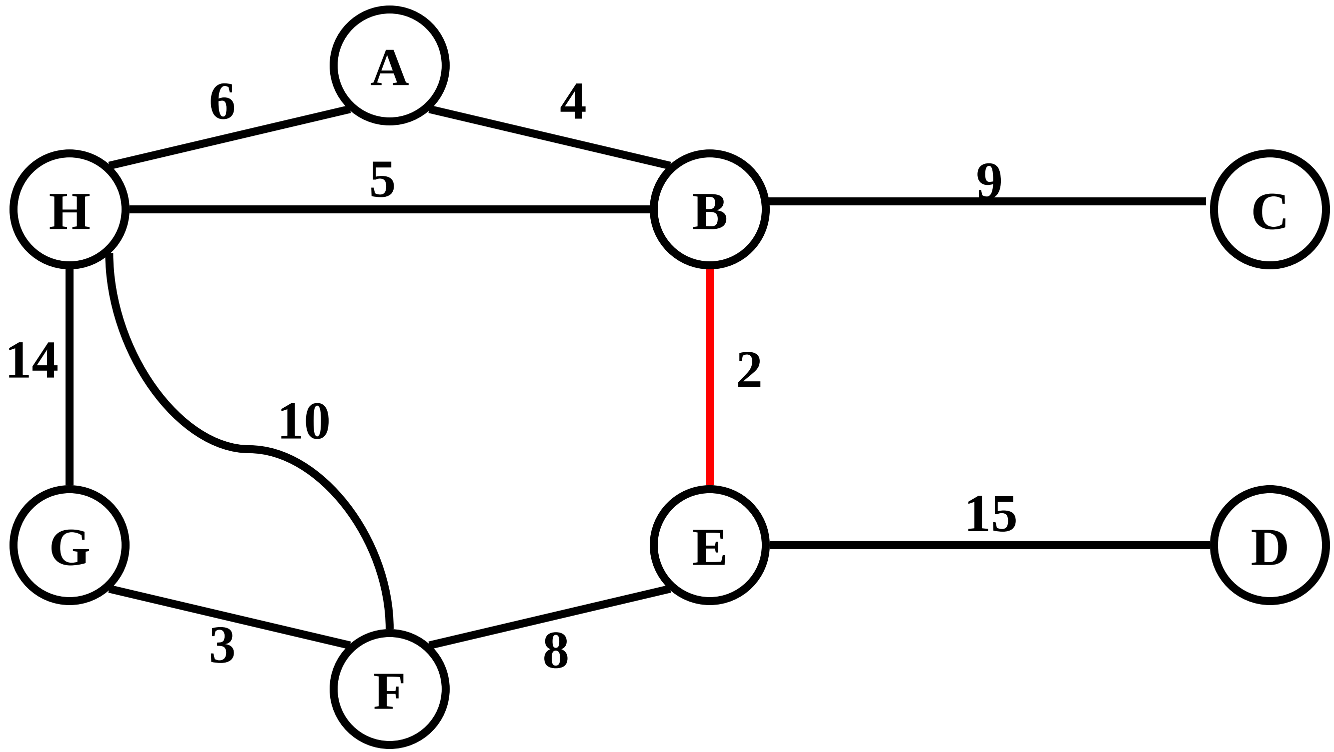
MST-Kruskal(G,w)

```
R = E;
F = 0;
While (R is not empty)
    Remove the light edge, (u,v), from R;
    if ((u,v) does not make a cycle in F)
        Add (u,v) to F;
return F;
```

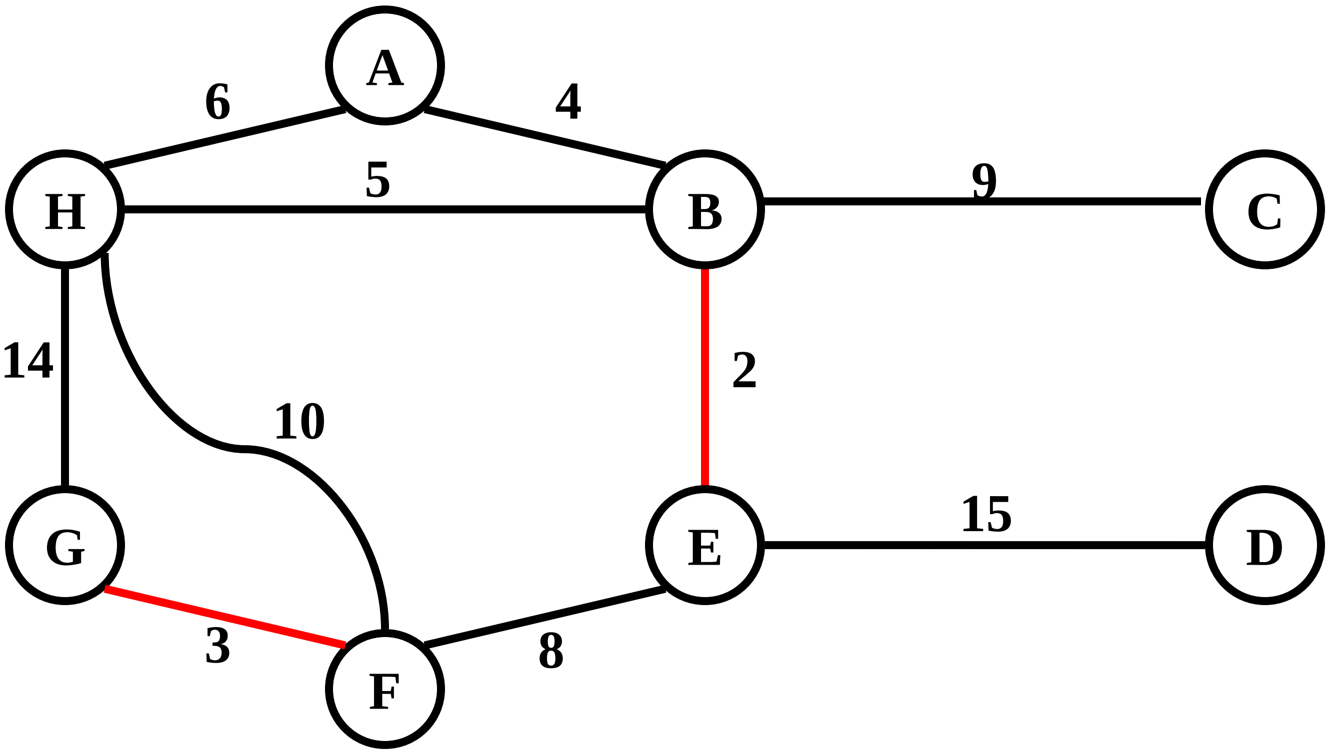
Example of Kruskal



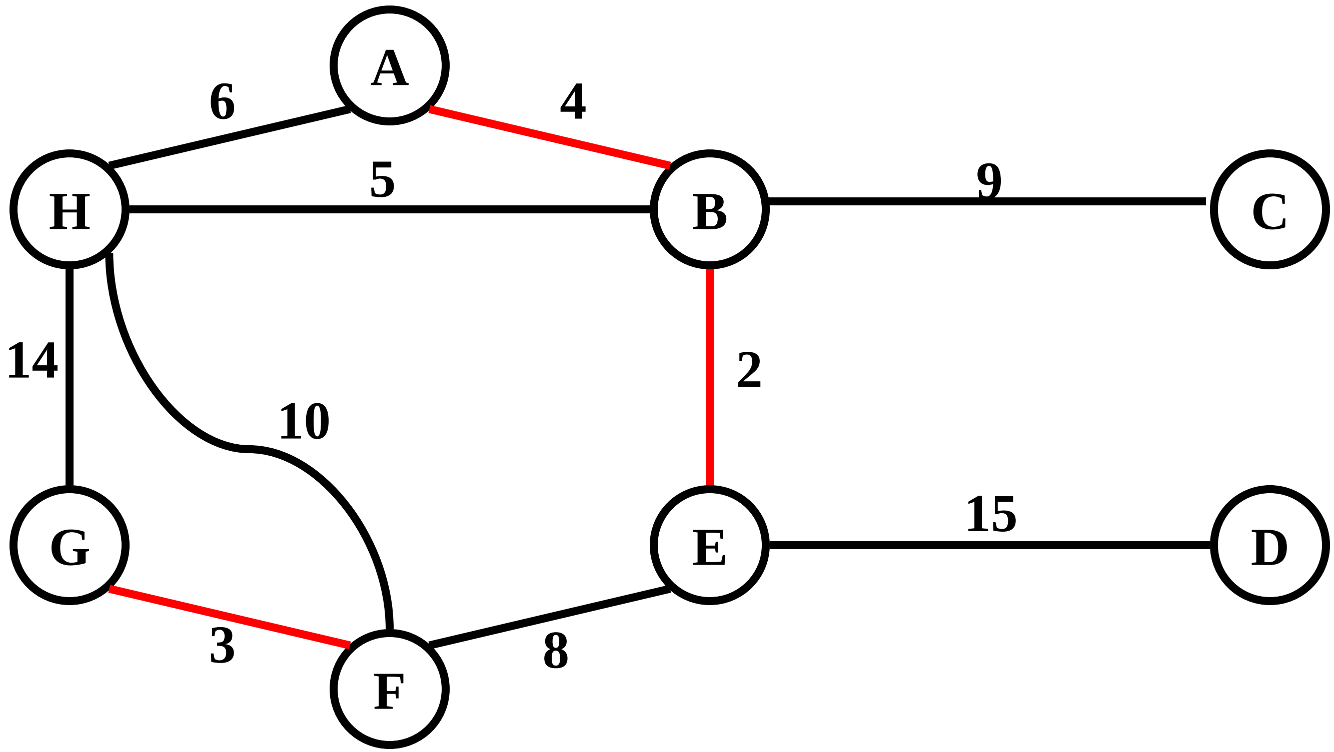
Example of Kruskal



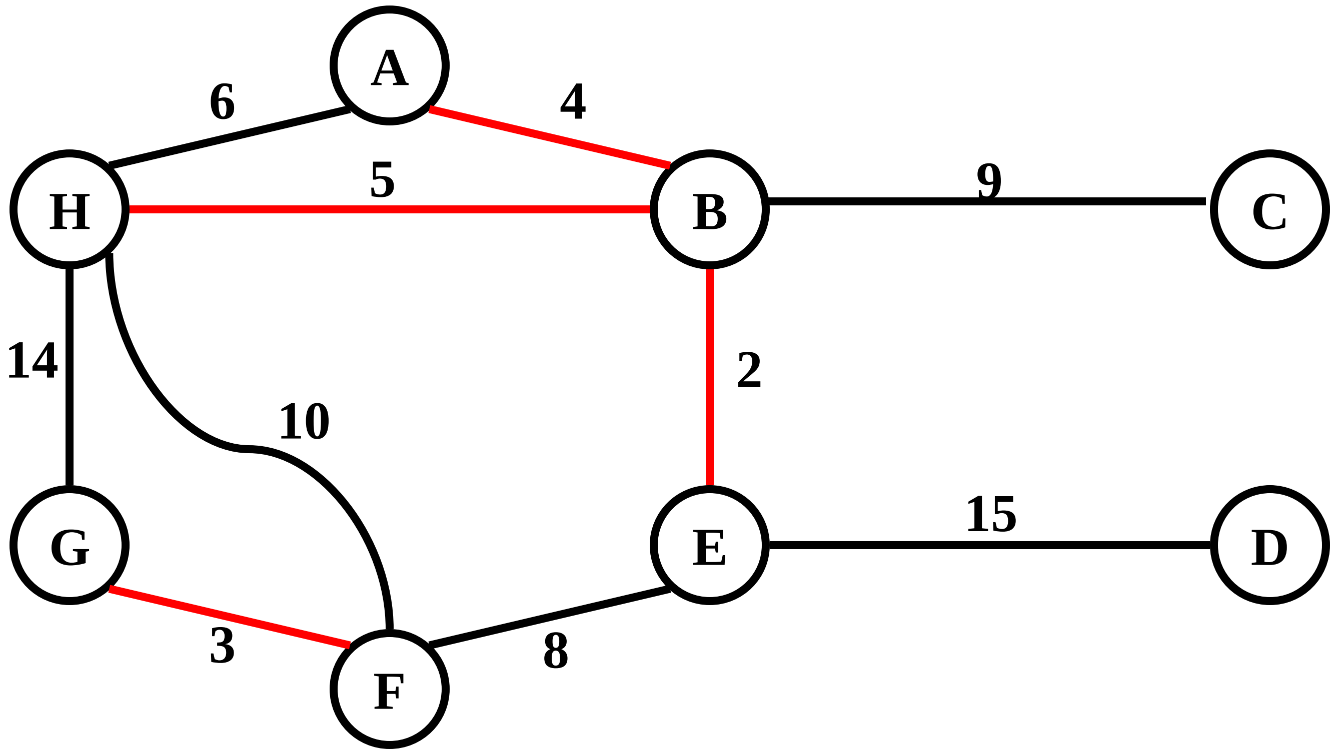
Example of Kruskal



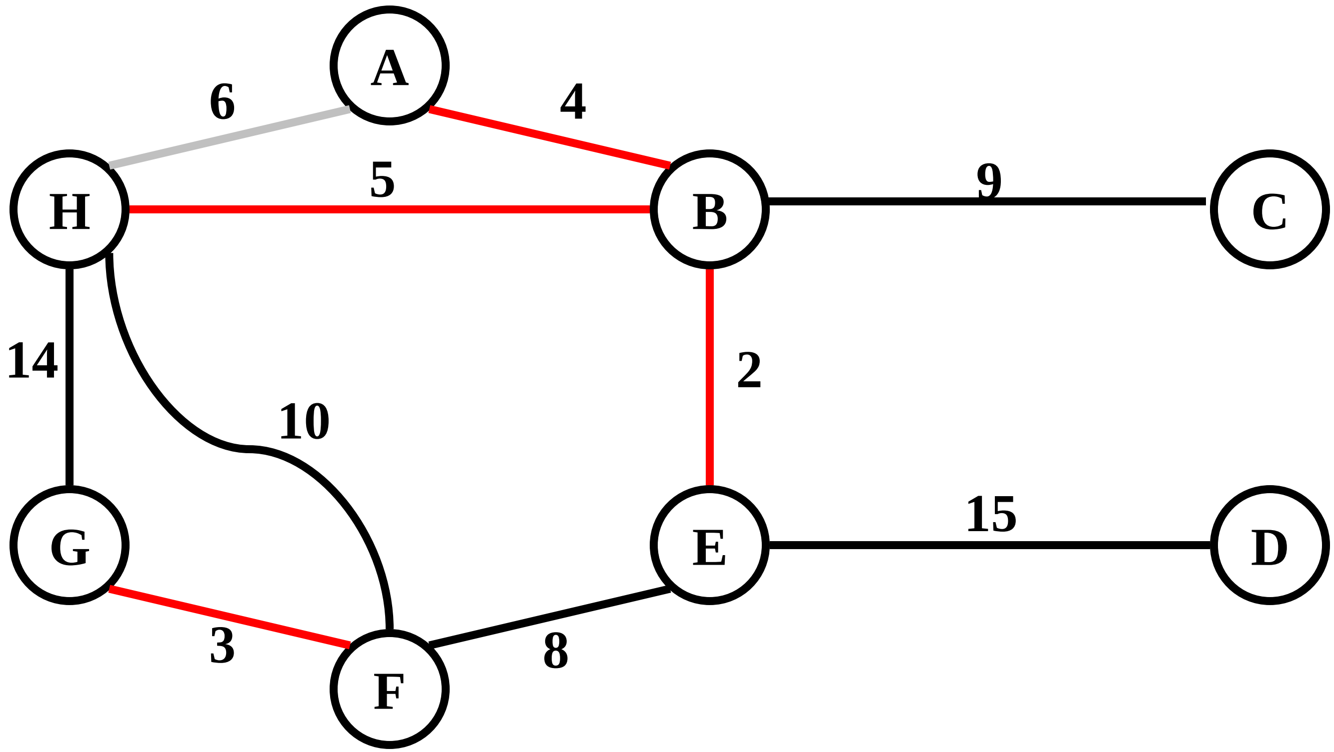
Example of Kruskal



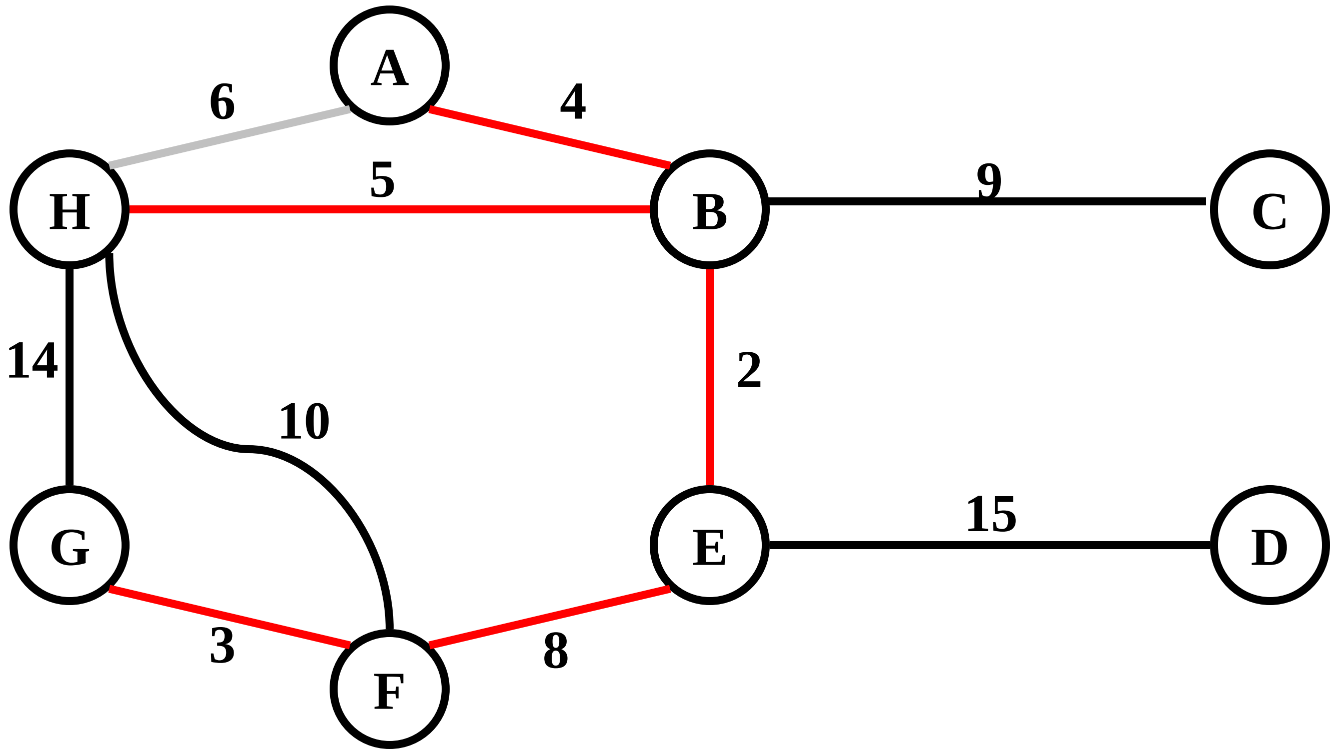
Example of Kruskal



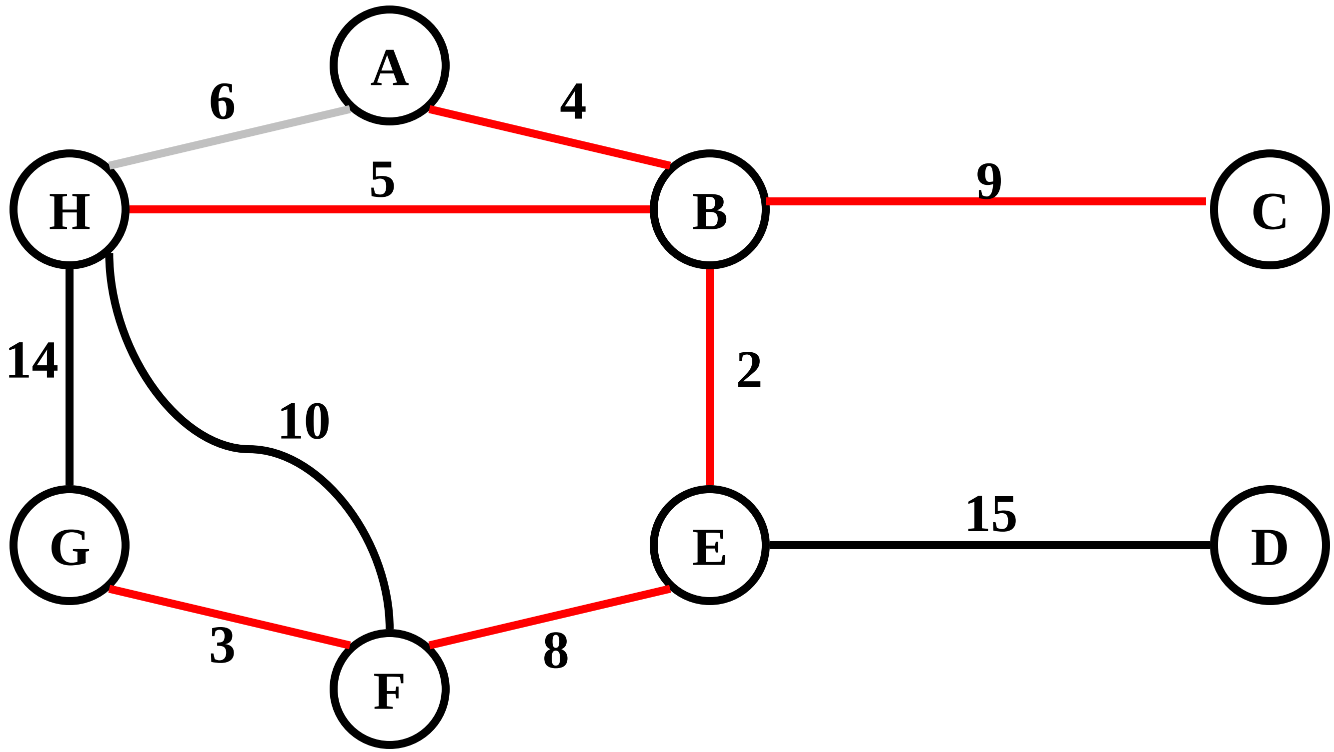
Example of Kruskal



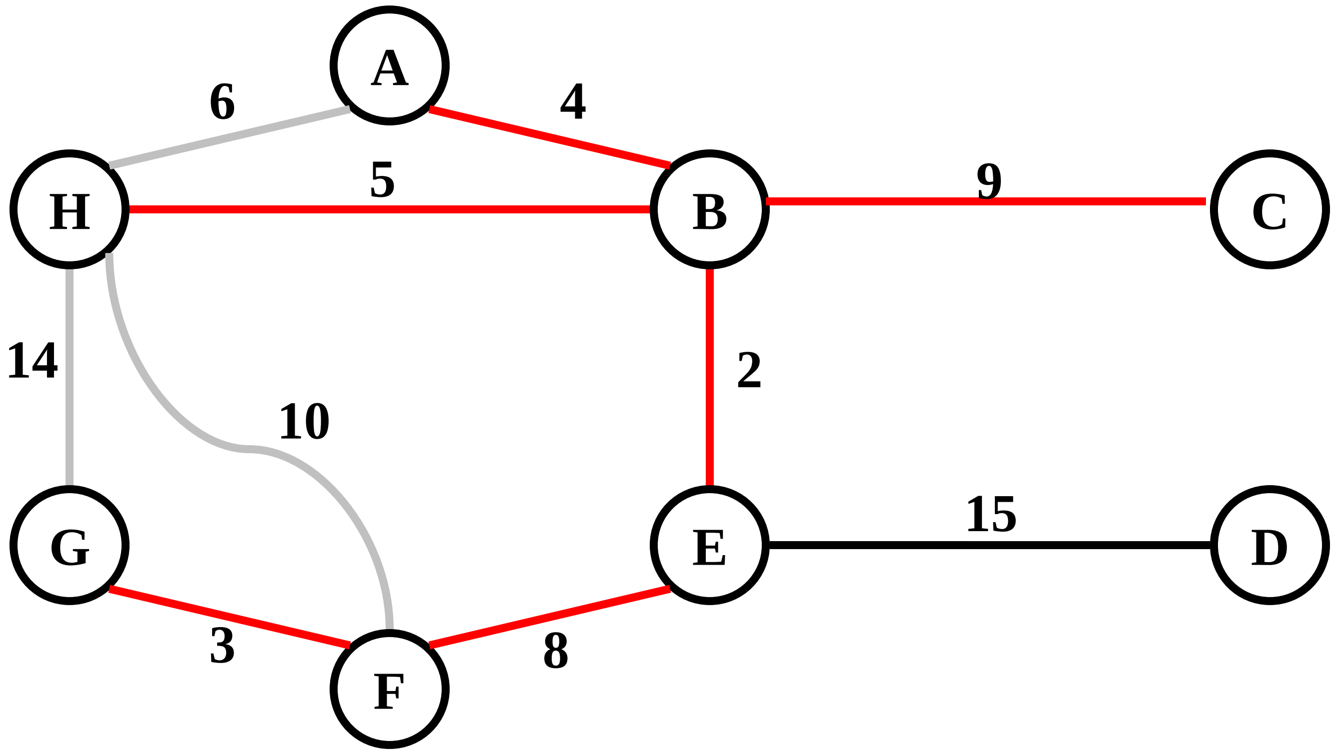
Example of Kruskal



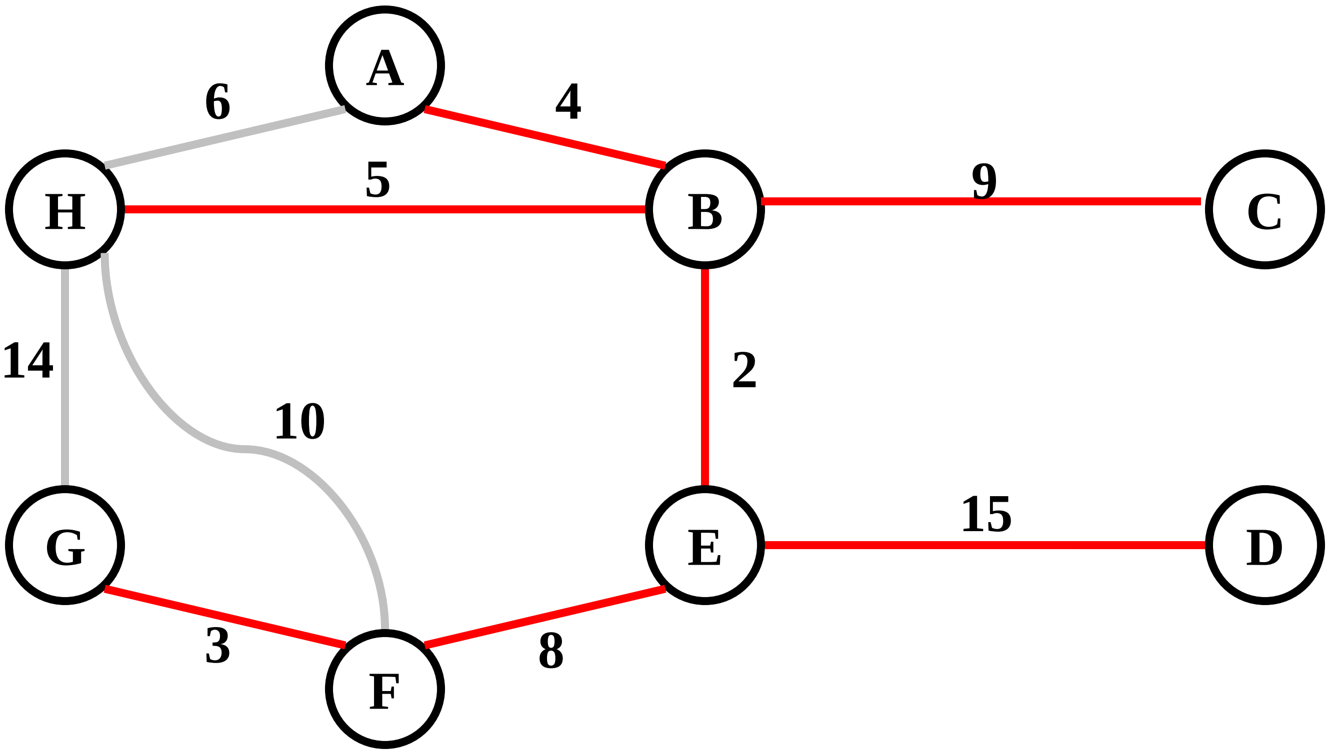
Example of Kruskal



Example of Kruskal



Example of Kruskal



Prim's algorithm

- Builds one tree, so A is always a tree.
- Start from an arbitrary "root" r.
- At each step, find a light edge crossing cut $(V_A, V - V_A)$, where V_A = vertices that A is incident on.
- $\pi[v]$ = parent of v, NIL if it has no parent or $v = r$.
- To find a light edge quickly
  - use a priority queue Q.

Prim's MST: Outline

PrimMST(G,n)

Initialize all vertices as *unseen*.

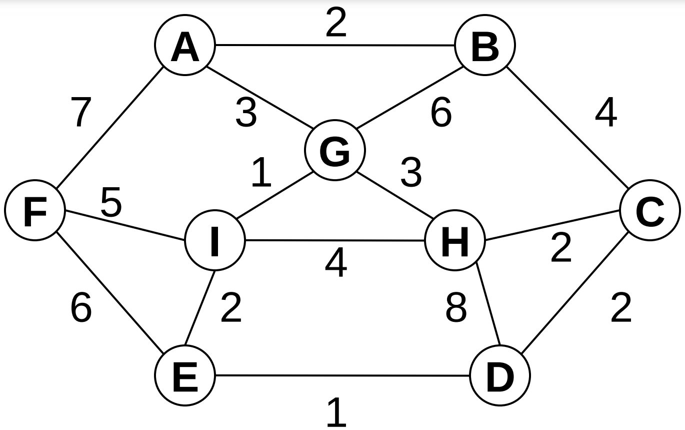
Select an arbitrary vertex *r* to start the tree; reclassify it as *tree*

Reclassify all vertices adjacent to *r* as *fringe*.

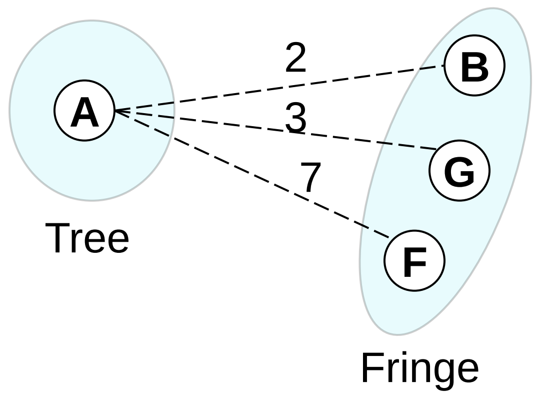
While there are fringe vertices;

- Select an edge of minimum weight between a tree vertex *t* and a fringe vertex *v*;
- Reclassify *v* as *tree*; add edge (*t,v*) to the tree;
- Reclassify all *unseen* vertices adjacent to *v* as *fringe*

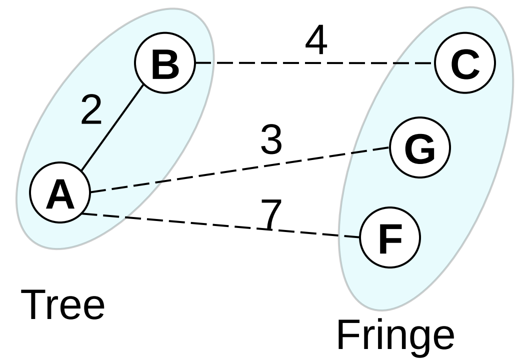
The Algorithm in action, e.g.



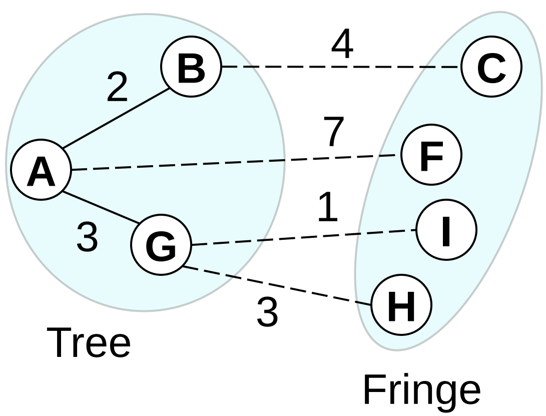
(a) A weighted graph



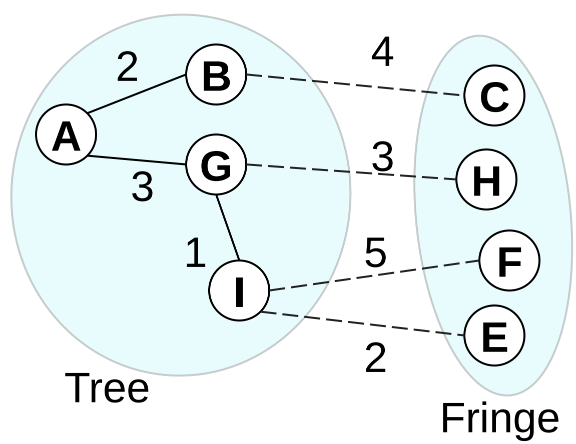
(b) After selection of the starting vertex



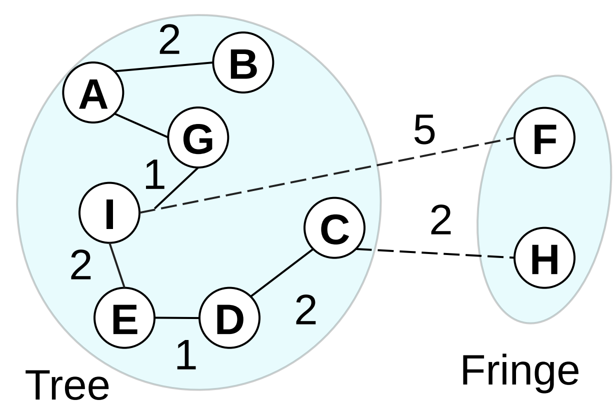
(c) *BG* was considered but did not replace *AG* as a candidate.



(d) After *A G* is selected and fringe and candidates are updated



(e) I F has replaced A F as a candidate.



(f) After several more passes: The two candidate edges will be put in the tree

Prim's Algorithm

MST-Prim(G, w, r)

$Q = V[G]$;

for each $u \in Q$

  $key[u] = \infty$; $\pi[u]$ = NIL;

$key[r] = 0$;

$\pi[r]$ = NULL;

while (Q not empty)

  u = ExtractMin(Q);

  for each $v \in$ Adj[u]

    if ($v \in Q$ and $w(u,v) < key[v]$)

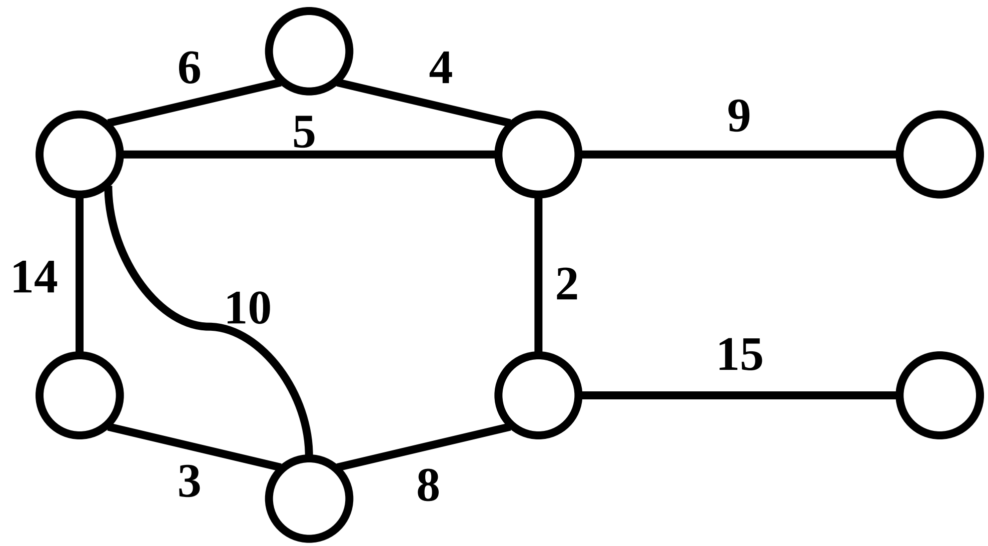
      $\pi[v] = u$;

      $key[v] = w(u,v)$;

Prim's Algorithm

MST-Prim(*G*, *w*, *r*)

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

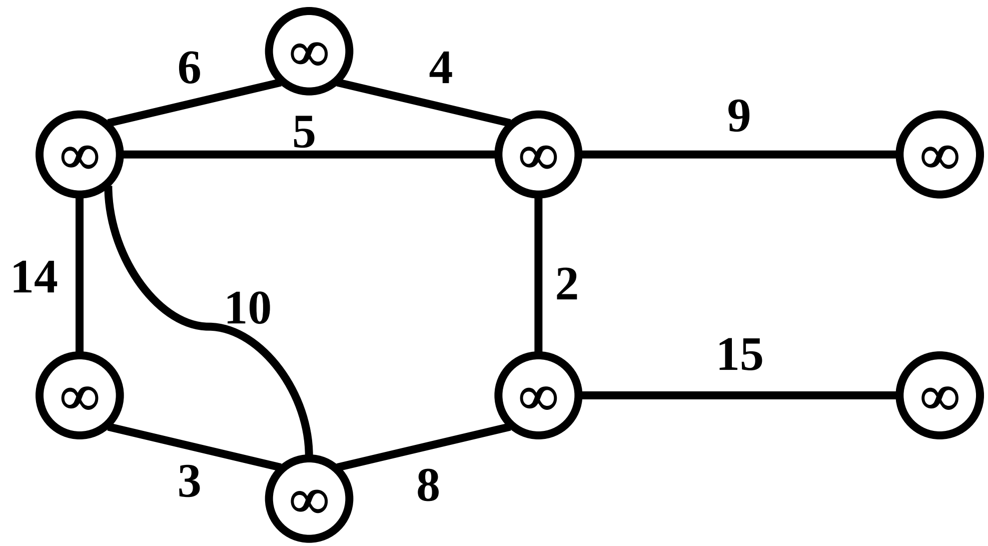


Run on example graph

Prim's Algorithm

MST-Prim(*G*, *w*, *r*)

$Q = V[G]$;
for each $u \in Q$
  $key[u] = \infty$; $\pi[u] =$ NIL;
$key[r] = 0$;
$\pi[r] =$ NULL;
while (Q not empty)
  $u =$ ExtractMin(Q);
  for each $v \in$ Adj[u]
    if ($v \in Q$ and $w(u,v) < key[v]$)
      $\pi[v] = u$;
      $key[v] = w(u,v)$;

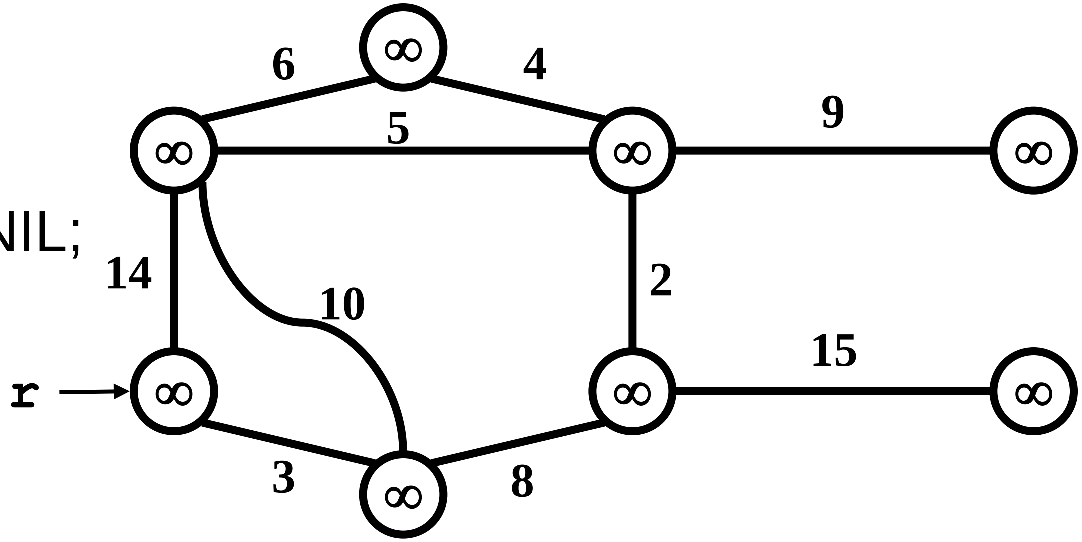


Run on example graph

Prim's Algorithm

MST-Prim(G, w, r)

$Q = V[G]$;
for each $u \in Q$
  $key[u] = \infty$; $\pi[u] =$ NIL;
$key[r] = 0$;
$\pi[r] =$ NULL;
while (Q not empty)
  $u =$ ExtractMin(Q);
  for each $v \in$ Adj[u]
    if ($v \in Q$ and $w(u,v) < key[v]$)
      $\pi[v] = u$;
      $key[v] = w(u,v)$;

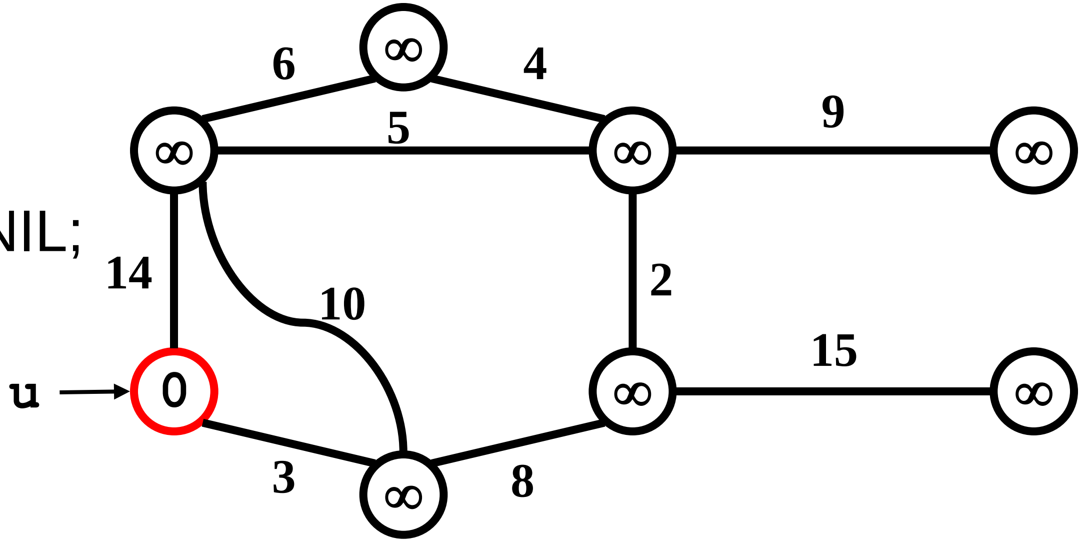


Pick a start vertex r

Prim's Algorithm

MST-Prim(*G*, *w*, *r*)

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

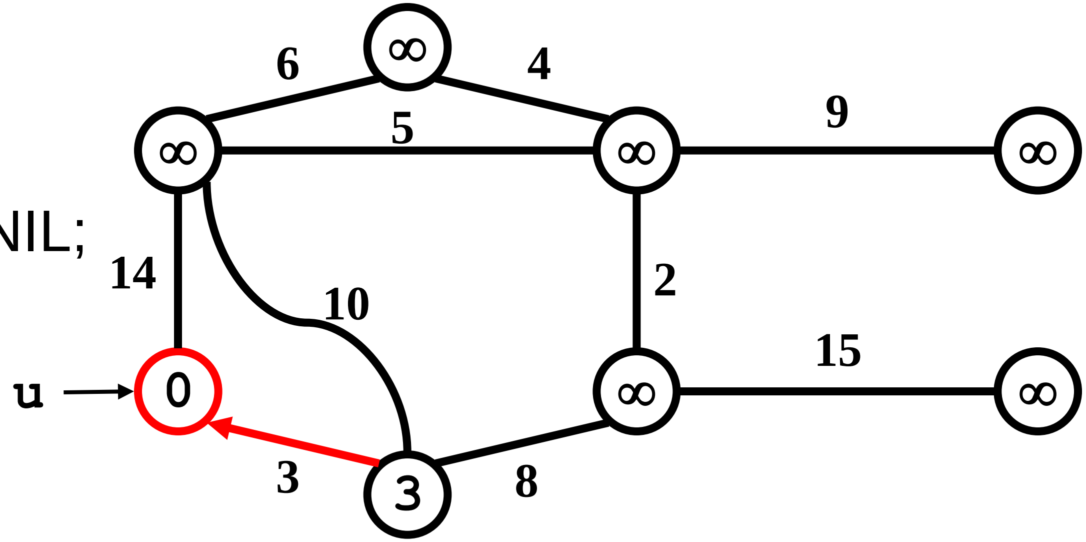


Red vertices have been removed from Q

Prim's Algorithm

MST-Prim(*G*, *w*, *r*)

$Q = V[G]$;
for each $u \in Q$
  $key[u] = \infty$; $\pi[u] = \text{NIL}$;
$key[r] = 0$;
$\pi[r] = \text{NULL}$;
while (Q not empty)
  $u = \text{ExtractMin}(Q)$;
  for each $v \in \text{Adj}[u]$
    if ($v \in Q$ and $w(u,v) < key[v]$)
      $\pi[v] = u$;
      $key[v] = w(u,v)$;

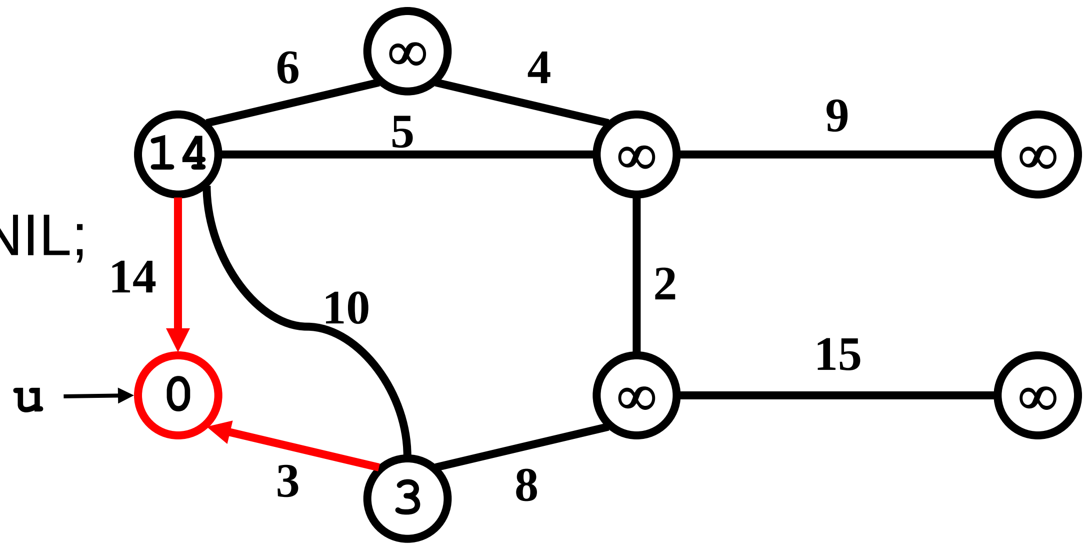


Red arrows indicate parent pointers

Prim's Algorithm

MST-Prim(*G*, *w*, *r*)

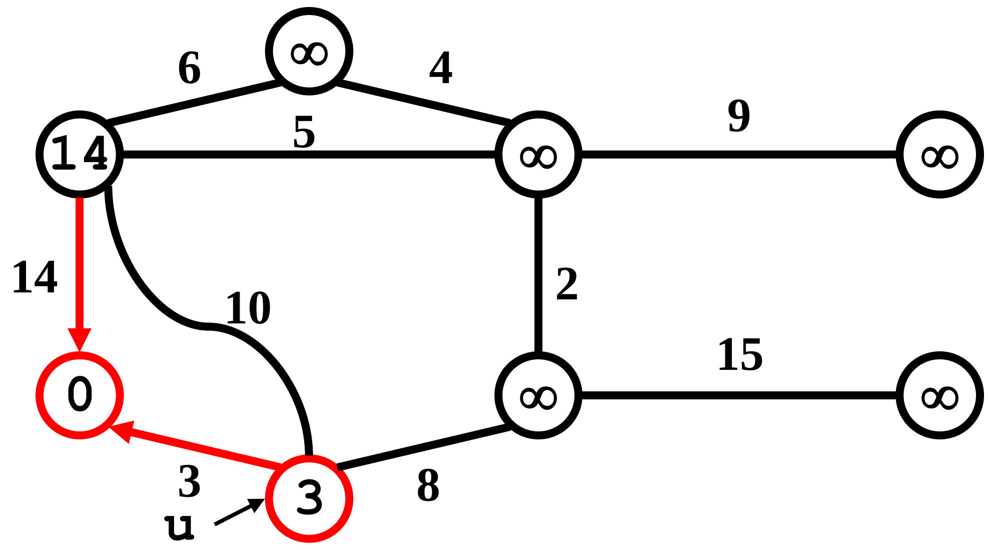
$Q = V[G]$;
for each $u \in Q$
  $key[u] = \infty$; $\pi[u] =$ NIL;
$key[r] = 0$;
$\pi[r] =$ NULL;
while (Q not empty)
  $u =$ ExtractMin(Q);
  for each $v \in$ Adj[u]
    if ($v \in Q$ and $w(u,v) < key[v]$)
      $\pi[v] = u$;
      $key[v] = w(u,v)$;



Prim's Algorithm

MST-Prim(*G*, *w*, *r*)

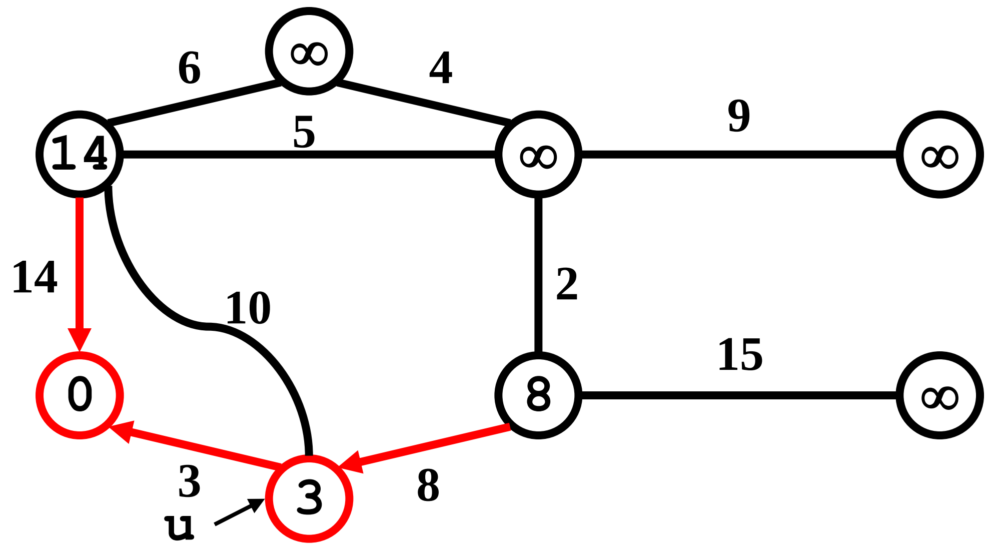
$Q = V[G]$;
for each $u \in Q$
  $key[u] = \infty$; $\pi[u]$ = NIL;
$key[r] = 0$;
$\pi[r]$ = NULL;
while (Q not empty)
  u = ExtractMin(Q);
  for each $v \in$ Adj[u]
    if ($v \in Q$ and $w(u,v) < key[v]$)
      $\pi[v] = u$;
      $key[v] = w(u,v)$;



Prim's Algorithm

MST-Prim(*G*, *w*, *r*)

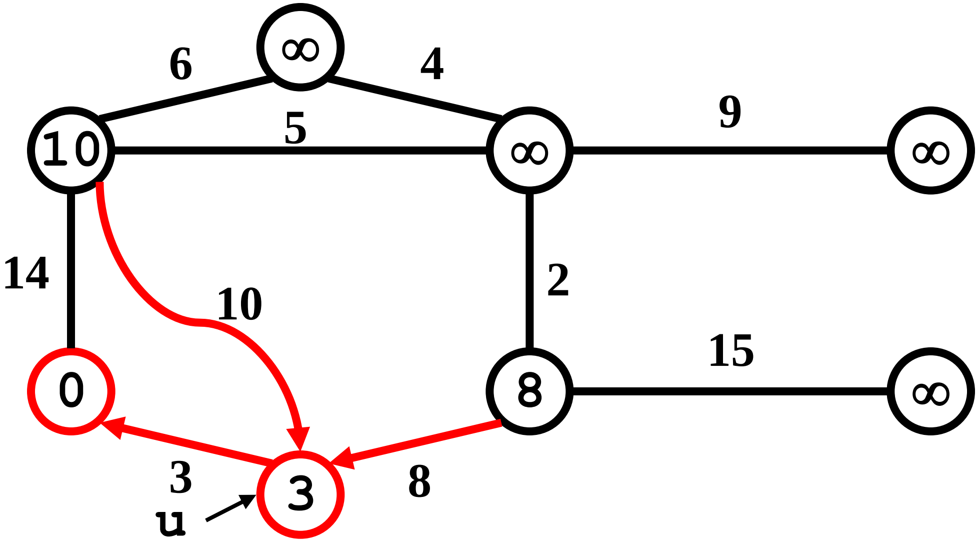
- $Q = V[G]$;
- for each $u \in Q$
  - $key[u] = \infty$; $\pi[u] = \text{NIL}$;
- $key[r] = 0$;
- $\pi[r] = \text{NULL}$;
- while (Q not empty)
  - $u = \text{ExtractMin}(Q)$;
  - for each $v \in \text{Adj}[u]$
    - if ($v \in Q$ and $w(u,v) < key[v]$)
      - $\pi[v] = u$;
      - $key[v] = w(u,v)$;



Prim's Algorithm

MST-Prim(*G*, *w*, *r*)

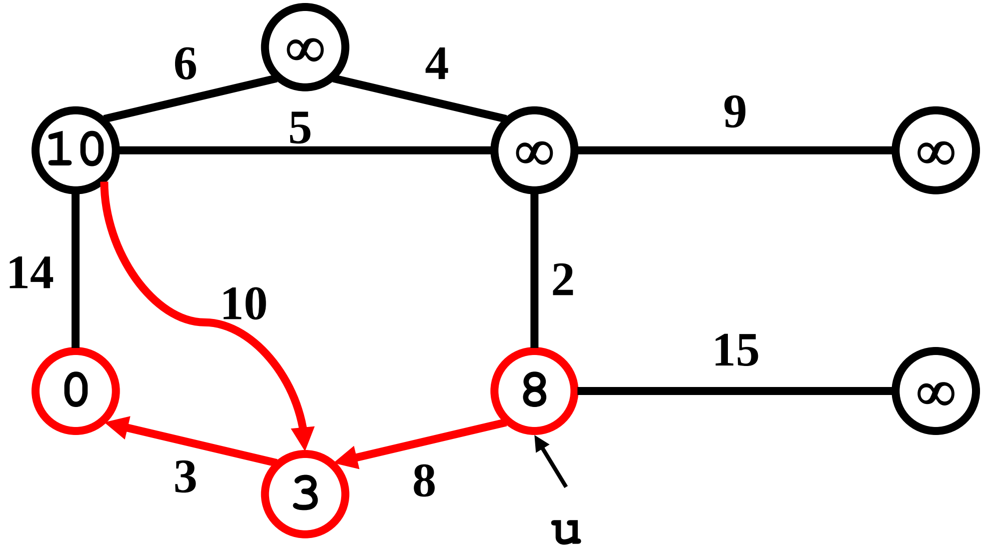
$Q = V[G]$;
for each $u \in Q$
  $key[u] = \infty$; $\pi[u]$ = NIL;
$key[r] = 0$;
$\pi[r]$ = NULL;
while (Q not empty)
  u = ExtractMin(Q);
  for each $v \in \mathrm{Adj}[u]$
    if ($v \in Q$ and $w(u,v) < key[v]$)
      $\pi[v] = u$;
      $key[v] = w(u,v)$;



Prim's Algorithm

MST-Prim(G, *w*, *r*)

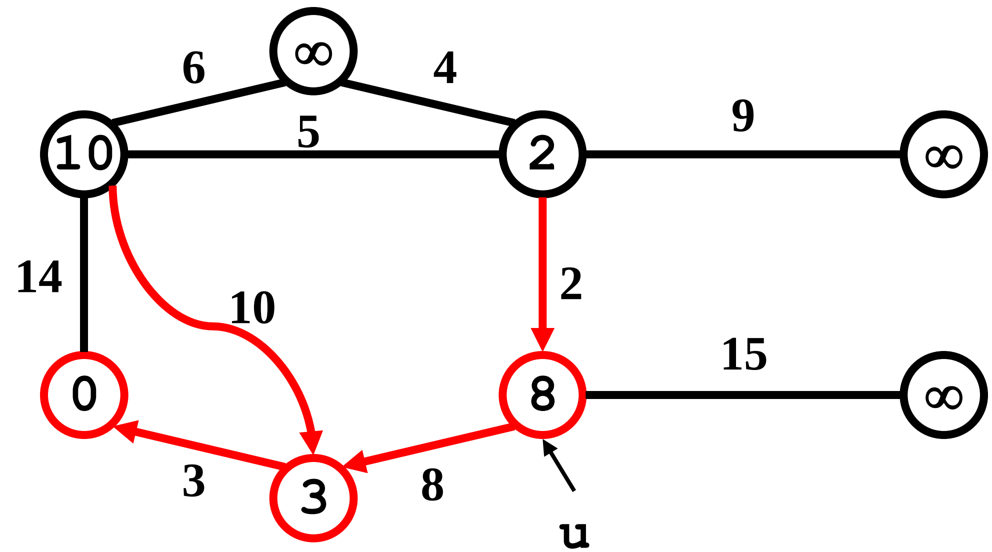
Q = *V*[*G*];
for each *u* ∈ *Q*
  key[*u*] = ∞; π[*u*] = NIL;
key[*r*] = 0;
π[*r*] = NULL;
while (*Q* not empty)
  u = ExtractMin(*Q*);
  for each *v* ∈ Adj[*u*]
    if (*v* ∈ *Q* and *w*(*u*,*v*) < *key*[*v*])
      π[*v*] = *u*;
      key[*v*] = *w*(*u*,*v*);



Prim's Algorithm

MST-Prim(*G*, *w*, *r*)

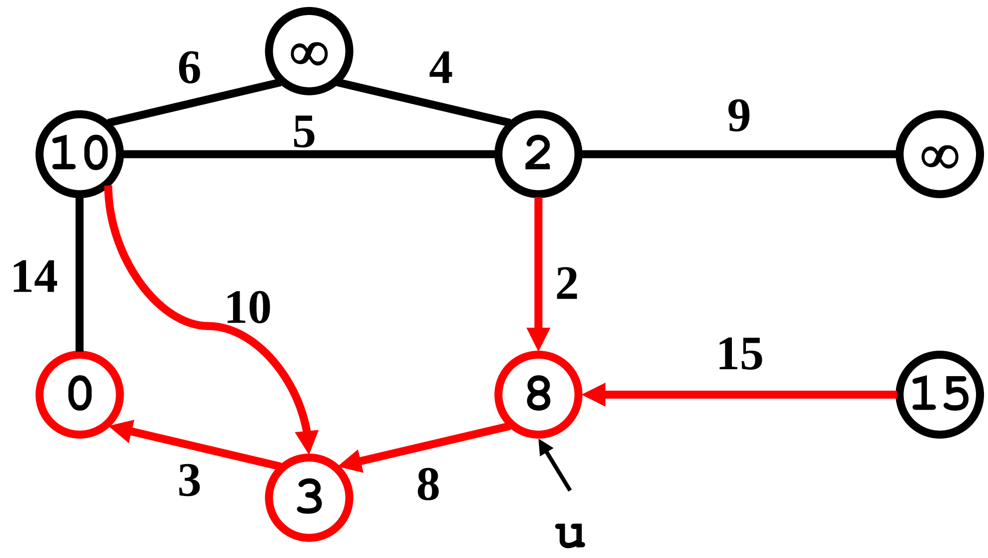
```
Q = V[G];
for each u ∈ Q
    key[u] = ∞; π[u] = NIL;
key[r] = 0;
π[r] = NULL;
while (Q not empty)
    u = ExtractMin(Q);
    for each v ∈ Adj[u]
        if (v ∈ Q and w(u,v) < key[v])
            π[v] = u;
            key[v] = w(u,v);
```



Prim's Algorithm

MST-Prim(*G*, *w*, *r*)

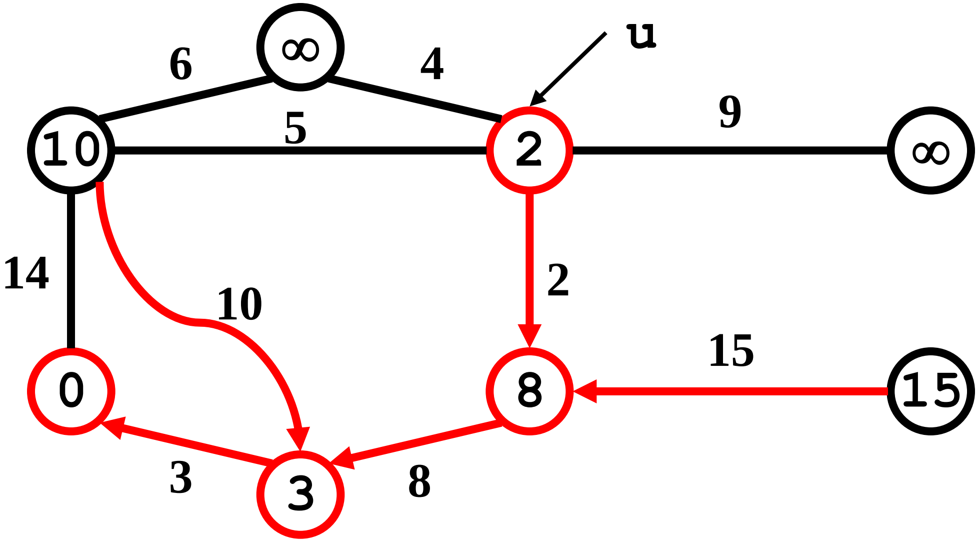
$Q = V[G]$;
for each $u \in Q$
  $key[u] = \infty$; $\pi[u] = \text{NIL}$;
$key[r] = 0$;
$\pi[r] = \text{NULL}$;
while (Q not empty)
  $u = \text{ExtractMin}(Q)$;
  for each $v \in \text{Adj}[u]$
    if ($v \in Q$ and $w(u,v) < key[v]$)
      $\pi[v] = u$;
      $key[v] = w(u,v)$;



Prim's Algorithm

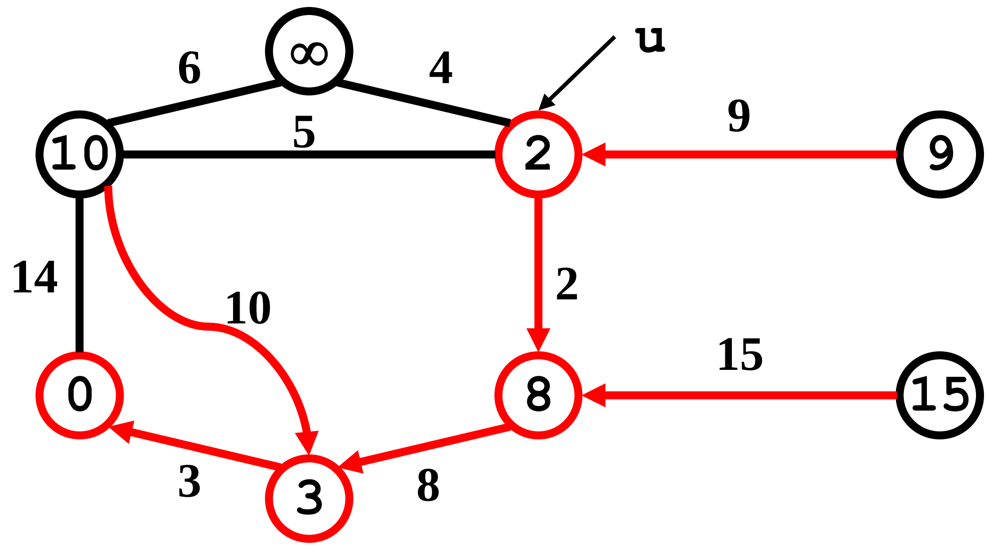
MST-Prim(*G*, *w*, *r*)

$Q = V[G]$;
for each $u \in Q$
  $key[u] = \infty$; $\pi[u]$ = NIL;
$key[r] = 0$;
$\pi[r]$ = NULL;
while (Q not empty)
  u = ExtractMin(Q);
  for each $v \in$ Adj[u]
    if ($v \in Q$ and $w(u,v) < key[v]$)
      $\pi[v] = u$;
      $key[v] = w(u,v)$;



Prim’s Algorithm

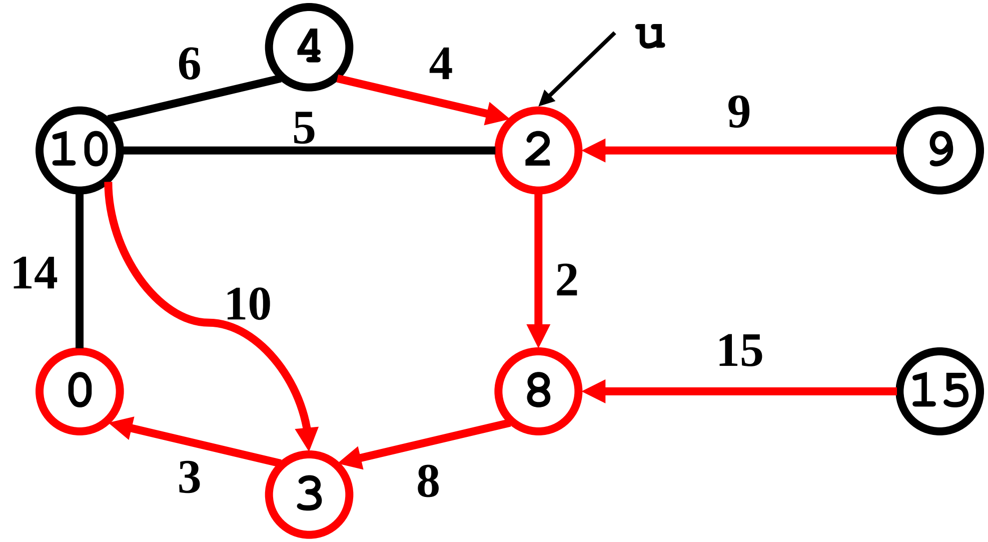
```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞; π[u] = NIL;
    key[r] = 0;
    π[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                π[v] = u;
                key[v] = w(u,v);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



Prim's Algorithm

MST-Prim(*G*, *w*, *r*)

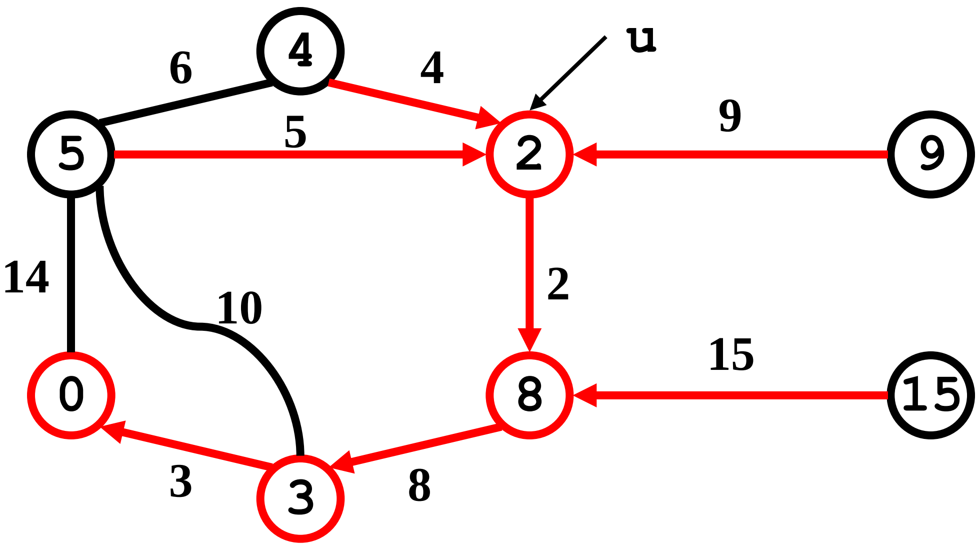
Q = *V*[*G*];
for each *u* ∈ *Q*
    key[*u*] = ∞; π[*u*] = NIL;
key[*r*] = 0;
π[*r*] = NULL;
while (*Q* not empty)
    u = ExtractMin(*Q*);
    for each *v* ∈ Adj[*u*]
        if (*v* ∈ *Q* and *w*(*u*,*v*) < *key*[*v*])
            π[*v*] = *u*;
            key[*v*] = *w*(*u*,*v*);



Prim's Algorithm

MST-Prim(*G*, *w*, *r*)

$Q = V[G]$;
for each $u \in Q$
  $key[u] = \infty$; $\pi[u]$ = NIL;
$key[r] = 0$;
$\pi[r]$ = NULL;
while (Q not empty)
  u = ExtractMin(Q);
  for each $v \in$ Adj[u]
    if ($v \in Q$ and $w(u,v) < key[v]$)
      $\pi[v] = u$;
      $key[v] = w(u,v)$;



Prim's Algorithm

MST-Prim(*G*, *w*, *r*)

$Q = V[G]$;

for each $u \in Q$

$key[u] = \infty$; $\pi[u]$ = NIL;

$key[r] = 0$;

$\pi[r]$ = NULL;

while (Q not empty)

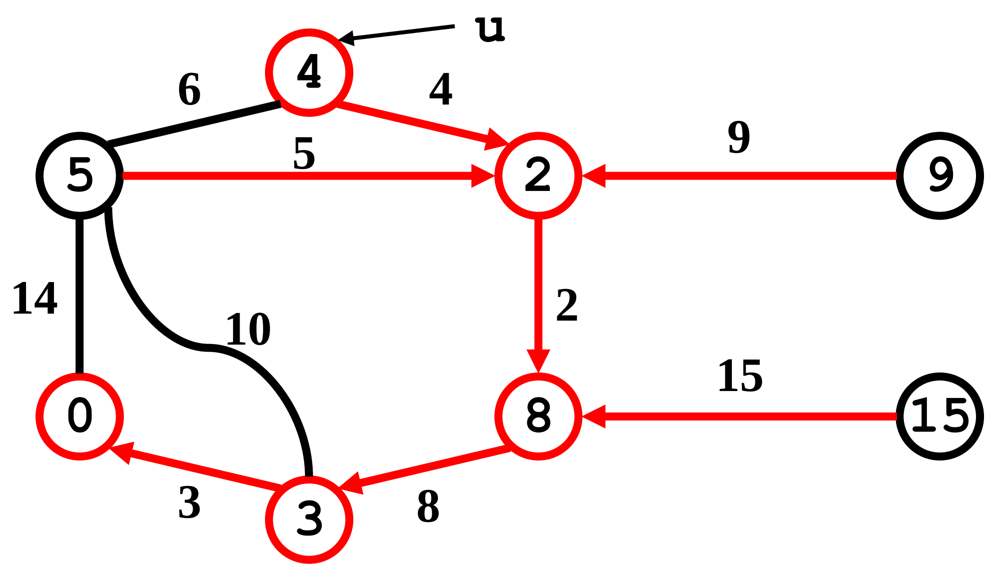
u = ExtractMin(Q);

for each $v \in$ Adj[u]

if ($v \in Q$ and $w(u,v) < key[v]$)

$\pi[v] = u$;

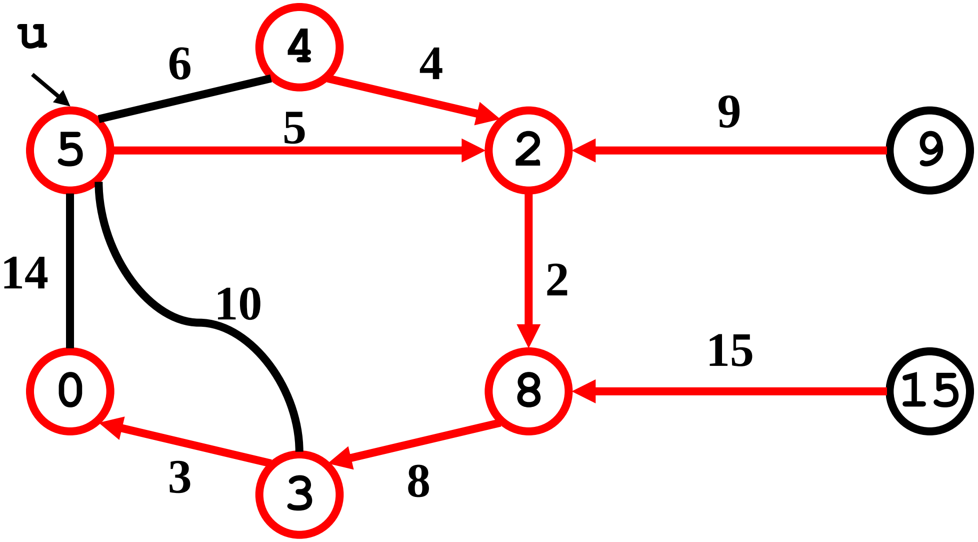
$key[v] = w(u,v)$;



Prim's Algorithm

MST-Prim(*G*, *w*, *r*)

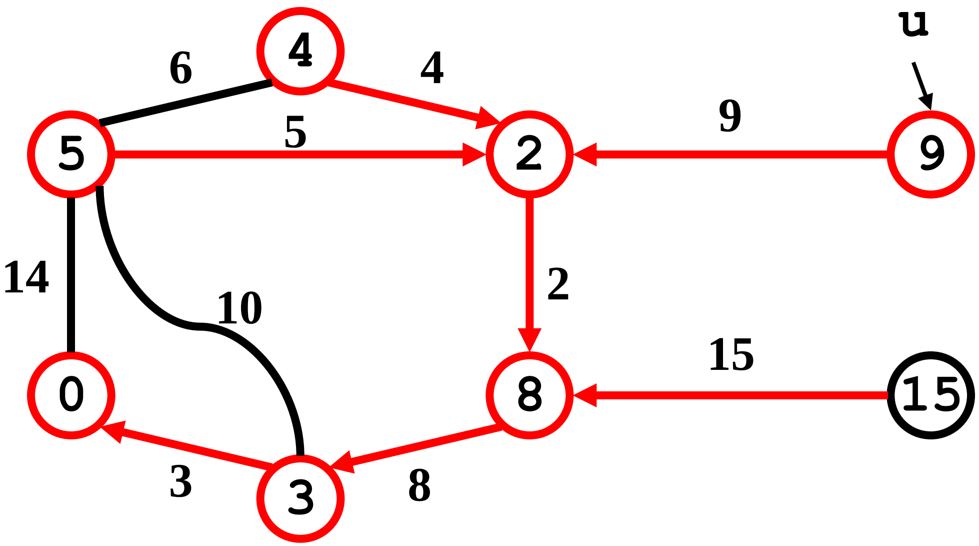
$Q = V[G]$;
for each $u \in Q$
  $key[u] = \infty$; $\pi[u] =$ NIL;
$key[r] = 0$;
$\pi[r] =$ NULL;
while (Q not empty)
  $u =$ ExtractMin(Q);
  for each $v \in$ Adj[u]
    if ($v \in Q$ and $w(u,v) < key[v]$)
      $\pi[v] = u$;
      $key[v] = w(u,v)$;



Prim's Algorithm

MST-Prim(*G*, *w*, *r*)

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



Prim's Algorithm

MST-Prim(*G*, *w*, *r*)

- $Q = V[G]$;
- for each $u \in Q$
  - $key[u] = \infty$; $\pi[u]$ = NIL;
- $key[r] = 0$;
- $\pi[r]$ = NULL;
- while (Q not empty)
  - u = ExtractMin(Q);
  - for each $v \in$ Adj[u]
    - if ($v \in Q$ and $w(u,v) < key[v]$)
      - $\pi[v] = u$;
      - $key[v] = w(u,v)$;