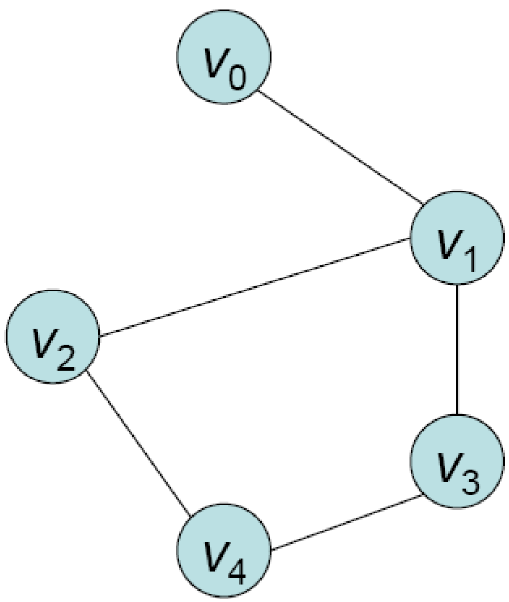
Minimum Spanning Tree Algoritm

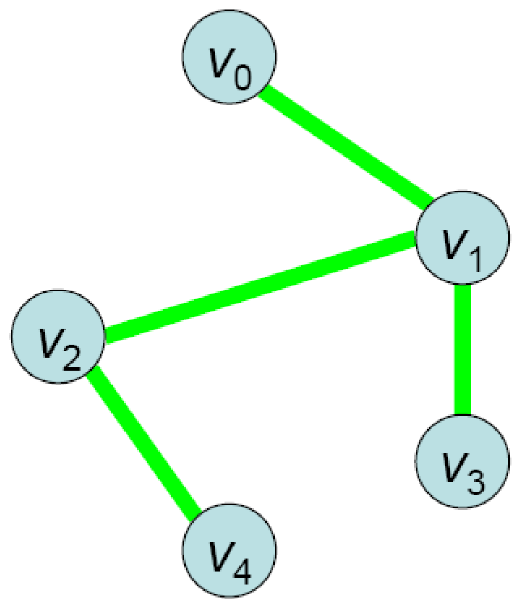
- Spanning Trees
- Minimum Spanning Tree
- Minimum Spanning Tree Algorithms
    - Kruskal Algorithms
    - Prim’s Algoritms

Spanning Tree

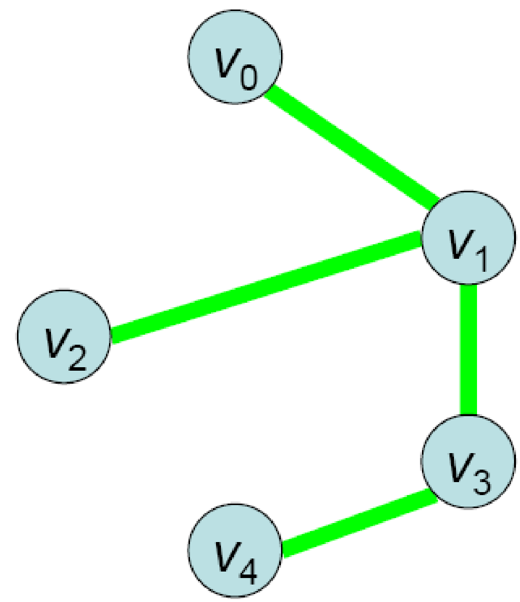
- A ***spanning tree*** (ST) of an *undirected* graph is a *tree* which contains *all vertices* and *some edges* of the graph.



Graph *G*



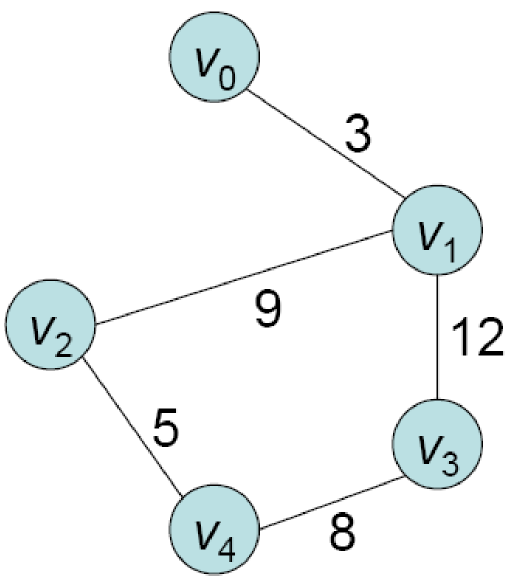
A spanning tree of *G*



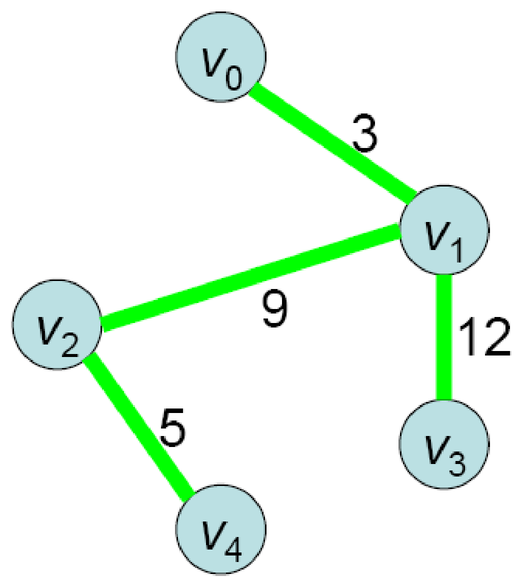
Another spanning tree of *G*

Minimum Spanning Tree

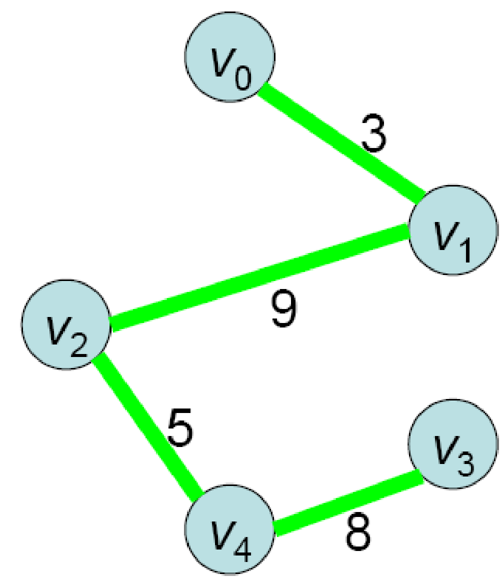
- A ***minimum spanning tree*** (MST) of an *undirected weighted* graph is a spanning tree whose sum of all weights is minimum.



Graph G

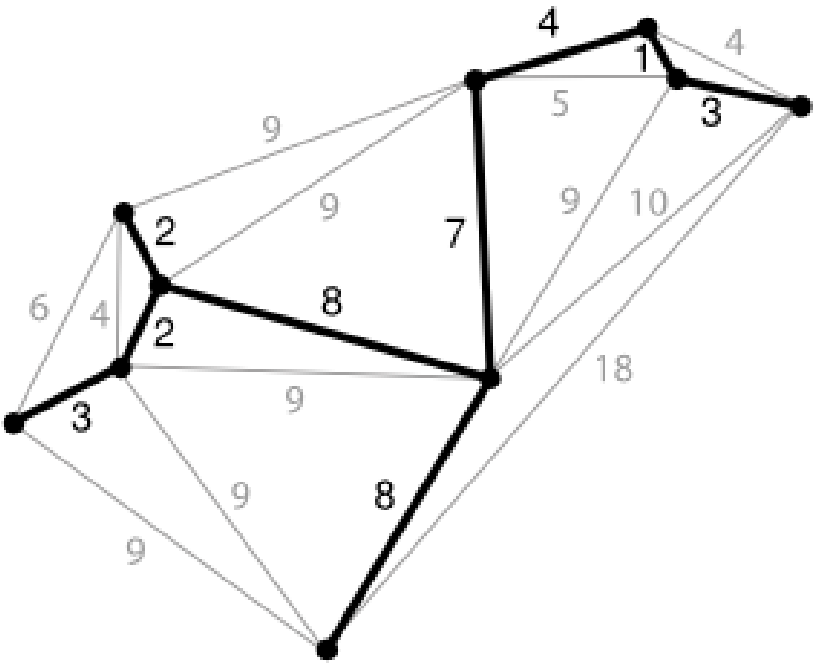


A spanning tree of G
(total weight = 3+9+12+5 = 29)



A MST of G
(total weight = 3+9+5+8 = 25)

Application



- ISP company laying LAN cable
- graph represents which houses are connected by those LAN cables
- A *spanning tree* for this graph would be a subset of those paths that has no cycles but still connects to every house
- might be several spanning trees possible.
- *minimum spanning tree* would be one with the lowest total cost

MST Algorithm

- Two common algorithms for finding MSTs.

  - ***Kruskal's algorithm***
    - From "forest" to tree

  - ***Prim's algorithm***
    - Build tree to span all vertices

Kruskal’s Algorithm

KRUSKAL(G(V, E), w)

A ← { } → *Set A will finally contains the edges of the MST*

for *each vertex v in V[G]*

do *MAKE-SET(v)*

sort edges of E into nondecreasing order by weight w

for *each edge (u, v) in E taken in nondecreasing order by weight from the sorted list*

do if *FIND-SET(u) != FIND-SET(v)*

then *A* ← *A* ∪ *{(u, v)}*

UNION(u, v)

return *A*

--G is a connected graph

--w is edge weights

Running Time: ***O(E lg V)***

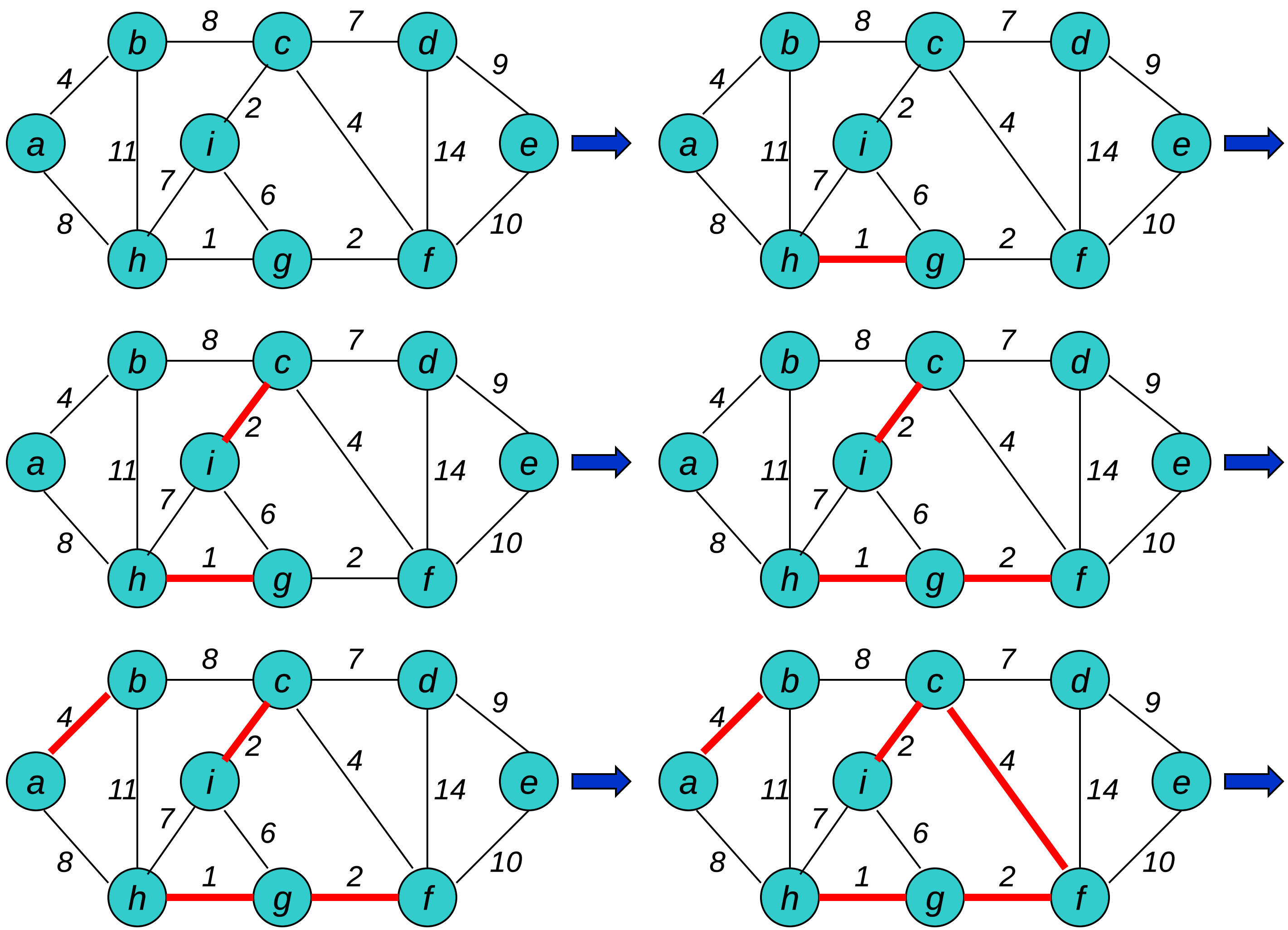
Make_SET(v): Create a new set whose only member is pointed to by v.

FIND_SET(v): Returns a representative element to the set containing v.

UNION(u, v): combining of trees i.e. Combines the dynamic sets that contain u and v into a new set that is union of these two sets

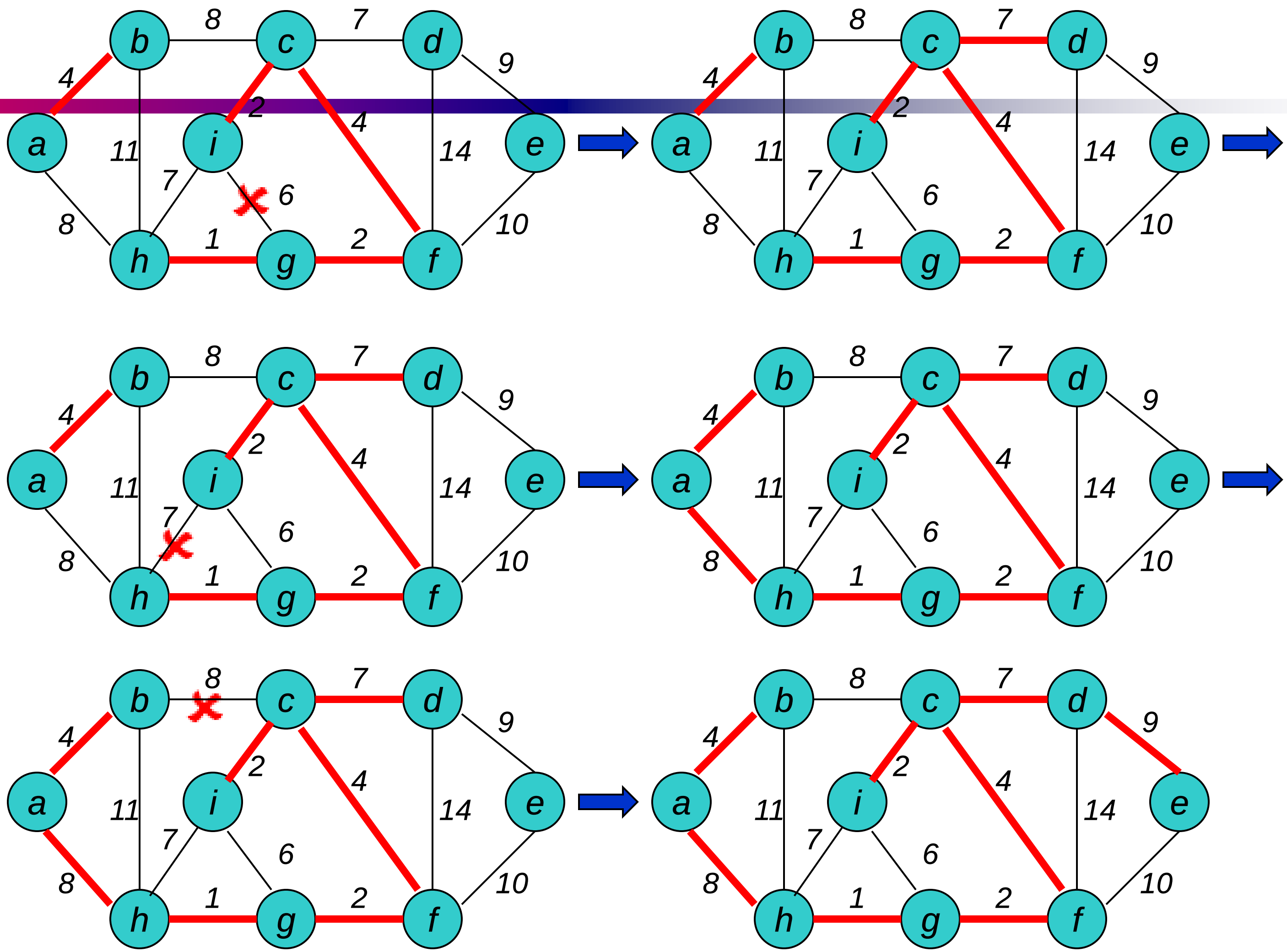
Kruskal’s Algorithm

Edge	*Weight*
<h, g>	*1*
<c, i>	*2*
<g, f>	*2*
<a, b>	*4*
<c, f>	*4*
<g, i>	*6*
<c, d>	*7*
<h, i>	*7*
<a, h>	*8*
<b, c>	*8*
<d, e>	*9*
<e, f>	*10*
<b, h>	*11*
<d, f>	*14*



Kruskal’s Algorithm

Edge	***Weight***	
<h, g>	*1*	✓
<c, i>	*2*	✓
<g, f>	*2*	✓
<a, b>	*4*	✓
<c, f>	*4*	✓
<g, i>	*6*	✗
<c, d>	*7*	✓
<h, i>	*7*	✗
<a, h>	*8*	✓
<b, c>	*8*	✗
<d, e>	*9*	✓
<e, f>	*10*	
<b, h>	*11*	
<d, f>	*14*	



Prim’s Algorithm

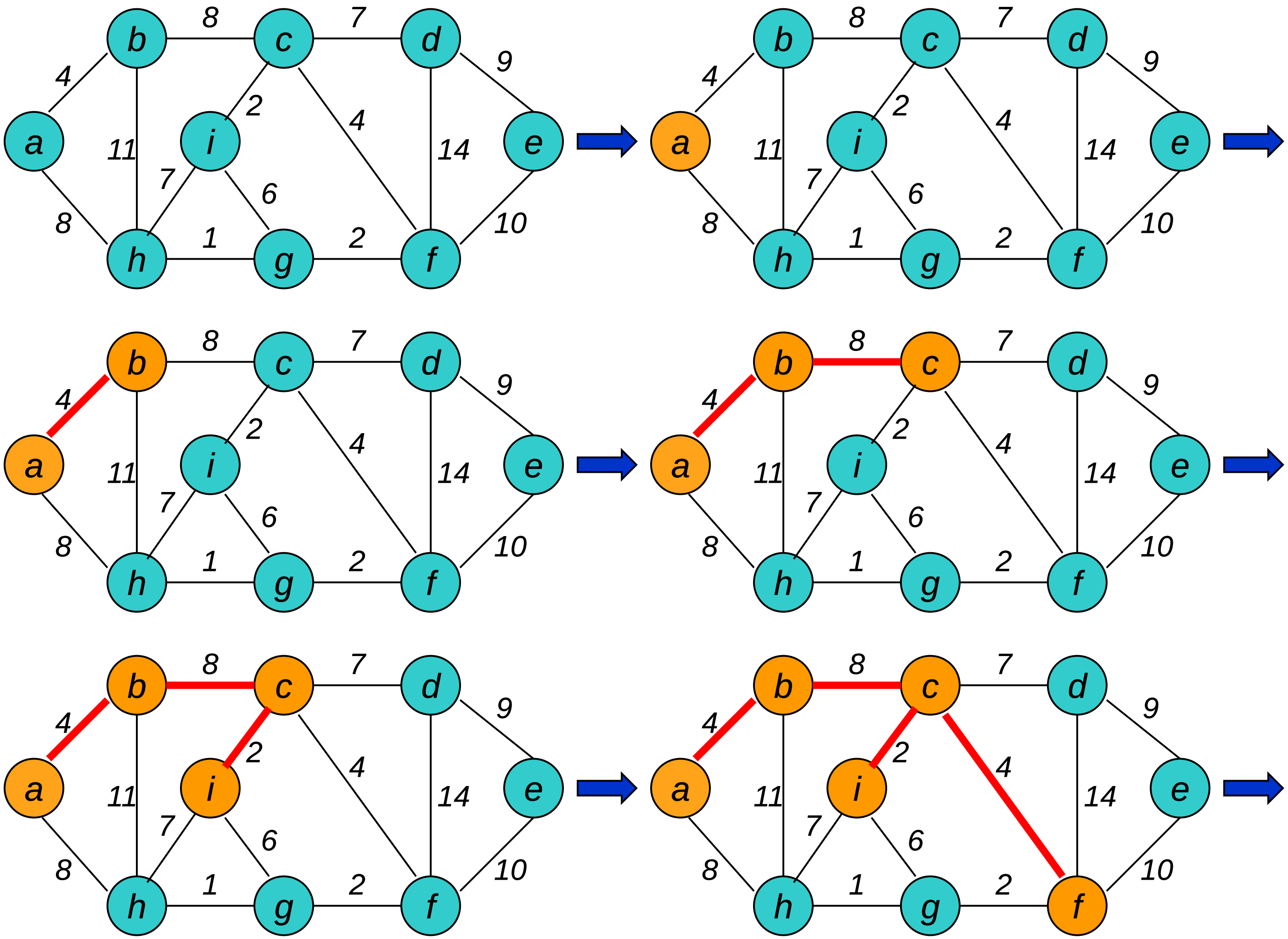
```
MST-Prim(G, w, r)
    for each u ∈ V[G]
       do key[u] = ∞;
       π[r] = NULL;
 key[r] = 0;
 Q = V[G];
while (Q ≠ Ø)
        u = Extract_Min(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                π[v] = u;
               key[v] = w(u,v);
```

--G is a connected graph
--w is edge weights
--r is root

Running Time: ***O(E + V lg V)***

[using a Fibonacci heap for the priority queue]

Prim's Algorithm



Prim’s Algorithm