Graphs Algorithms

Weighted Graphs

- Graphs that have a number assigned to each edge are called **weighted graphs**.
- Weighted graphs are used to model computer networks. Communications costs, the response times of the computers over these lines, or the distance between computers, can all be studied using weighted graphs.

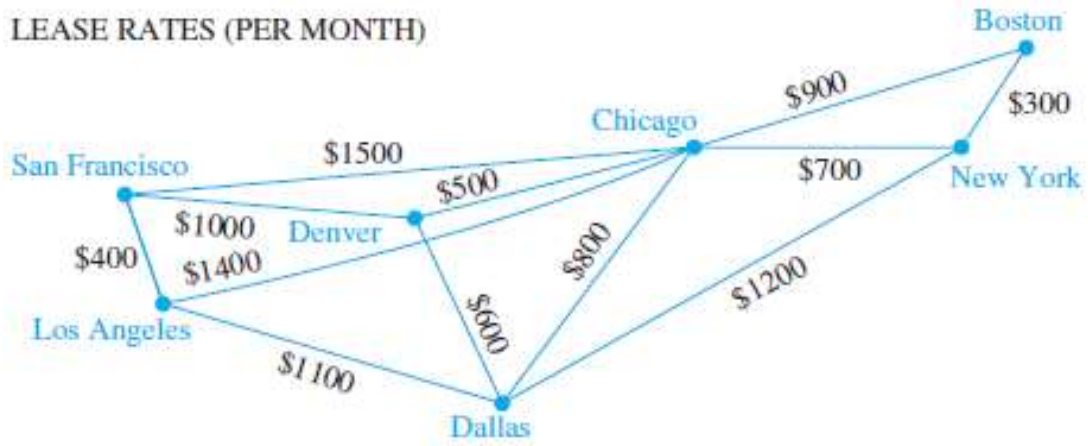
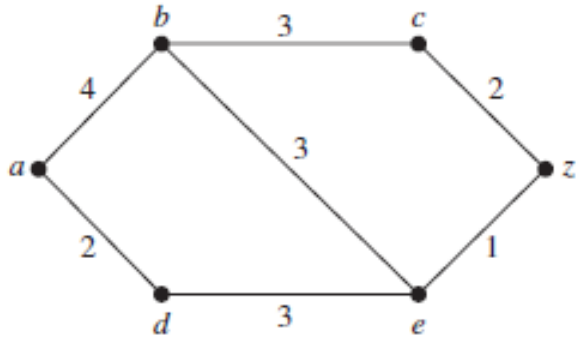


Figure: Weighted Graphs Modeling a Computer Network.

Example

- Another important problem involving weighted graphs asks for a circuit of shortest total length that visits every vertex of a complete graph exactly once.
- E.g 1: Travelling Salesman Problem
- E.g.2: What is the length of a shortest path between *a* and *z?*



Single-source shortest-path problem

- Given a graph G =(V,E), we want to find a shortest path from a given ***source*** vertex s ∈ V to each vertex v ∈ V .
- Dijkstra's algorithm solves the single-source shortest-paths problem on a weighted, connected simple graph G(V,E) for the case in which all edge weights are nonnegative.
- *Let* $L_k(v)$ is the length of a shortest path from *a* to *v*
- Updation:
- $L_k(a, v) = \min\{L_{k-1}(a, v), L_{k-1}(a, u) + w(u, v)\}$

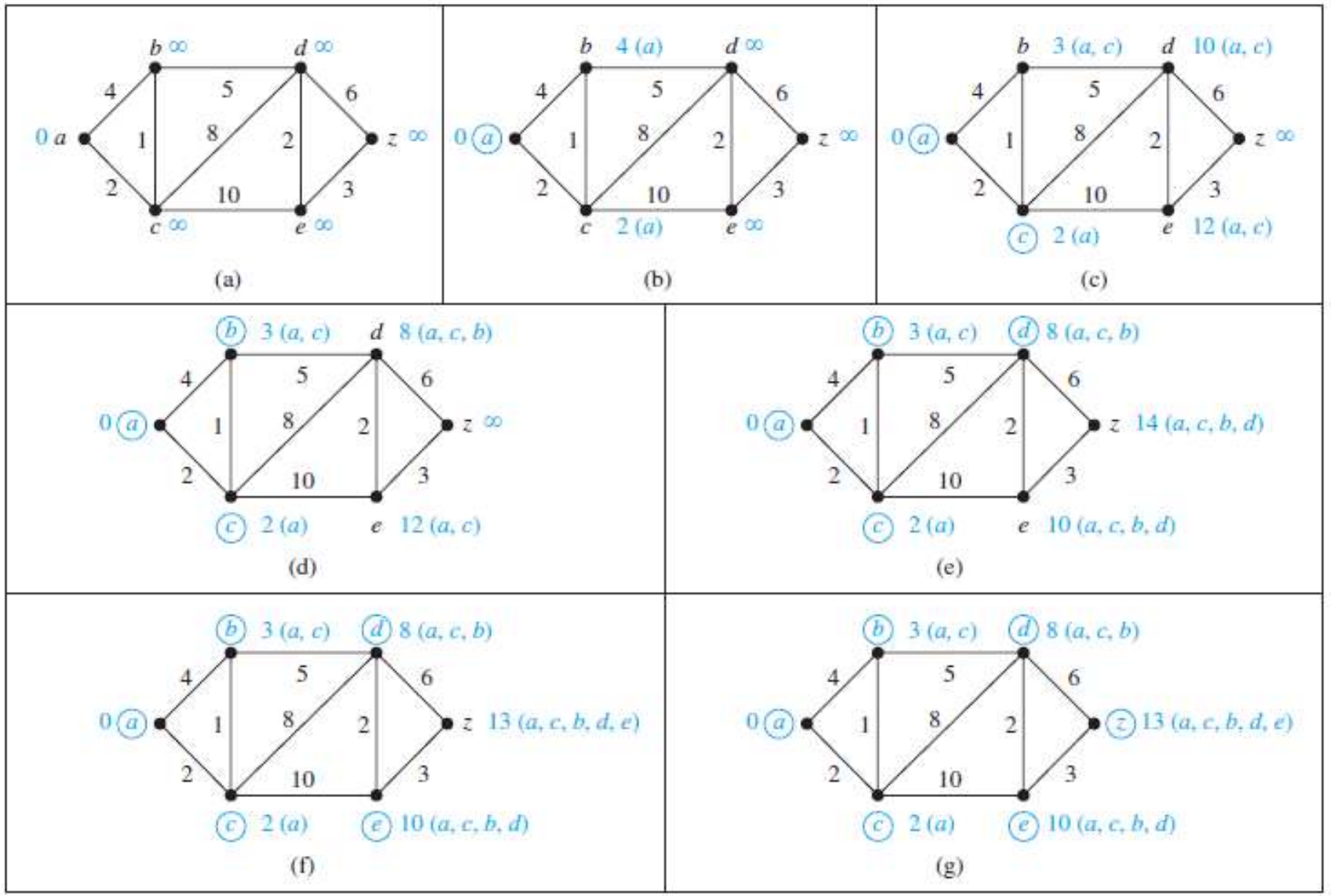
where *w(u, v)* is the length of the edge with *u* and *v* as endpoints

ALGORITHM 1 Dijkstra's Algorithm.

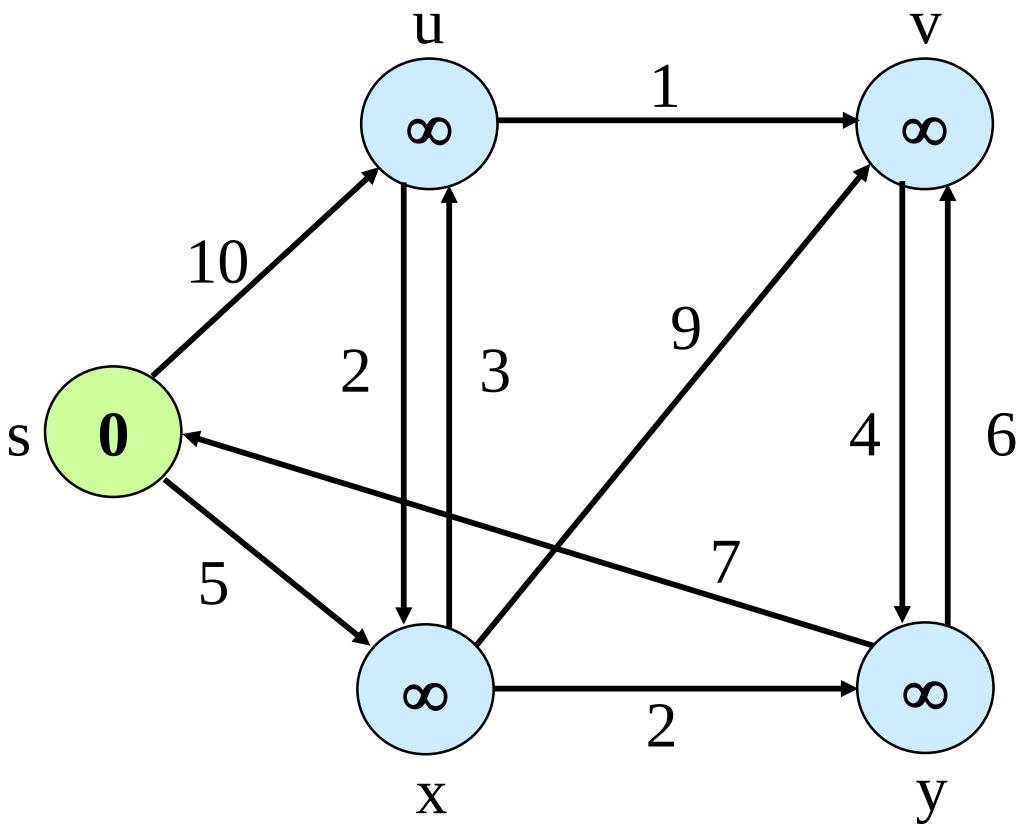
procedure *Dijkstra*(G: weighted connected simple graph, with all weights positive)
{G has vertices $a = v_0, v_1, \ldots, v_n = z$ and lengths $w(v_i, v_j)$ where $w(v_i, v_j) = \infty$ if $\{v_i, v_j\}$ is not an edge in G}
for $i := 1$ **to** n
    $L(v_i) := \infty$
$L(a) := 0$
$S := \emptyset$
{the labels are now initialized so that the label of a is 0 and all other labels are ∞, and S is the empty set}
while $z \notin S$
    $u :=$ a vertex not in S with $L(u)$ minimal
    $S := S \cup \{u\}$
    for all vertices v not in S
        if $L(u) + w(u, v) < L(v)$ **then** $L(v) := L(u) + w(u, v)$
        {this adds a vertex to S with minimal label and updates the labels of vertices not in S}
return $L(z)$ {$L(z) =$ length of a shortest path from a to z}

Example

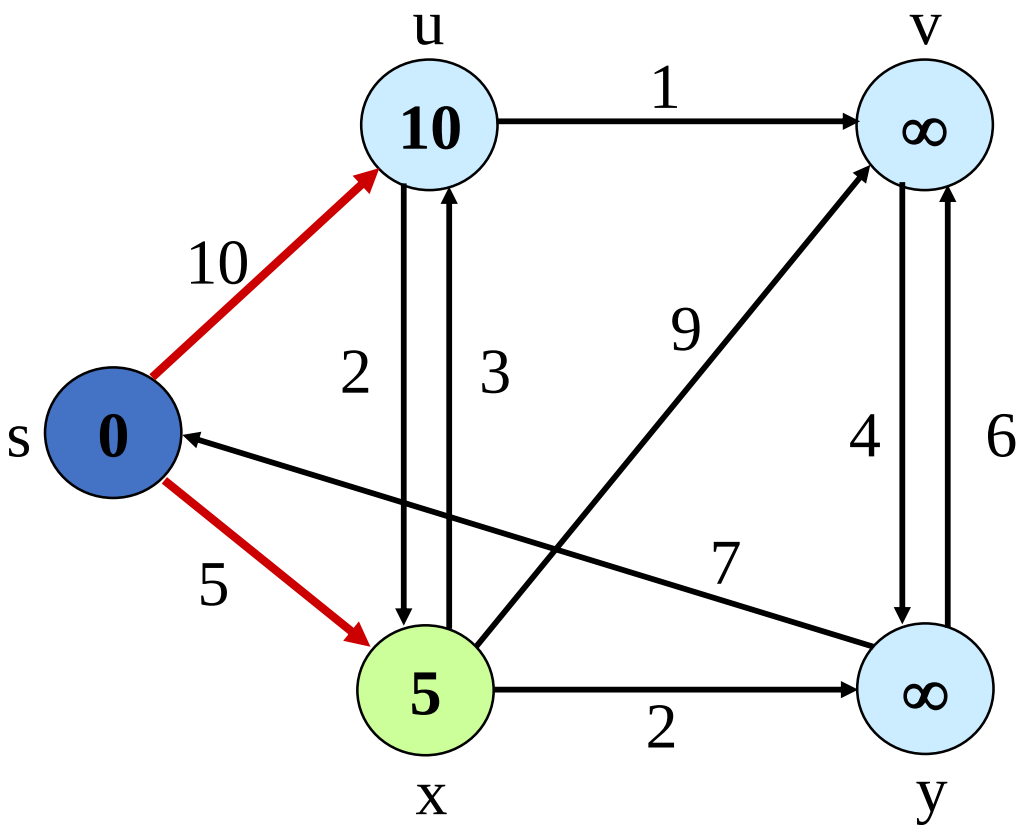
Use Dijkstra's algorithm to find the length of a shortest path between the vertices *a* and *z* in the weighted graph displayed in Figure (a).



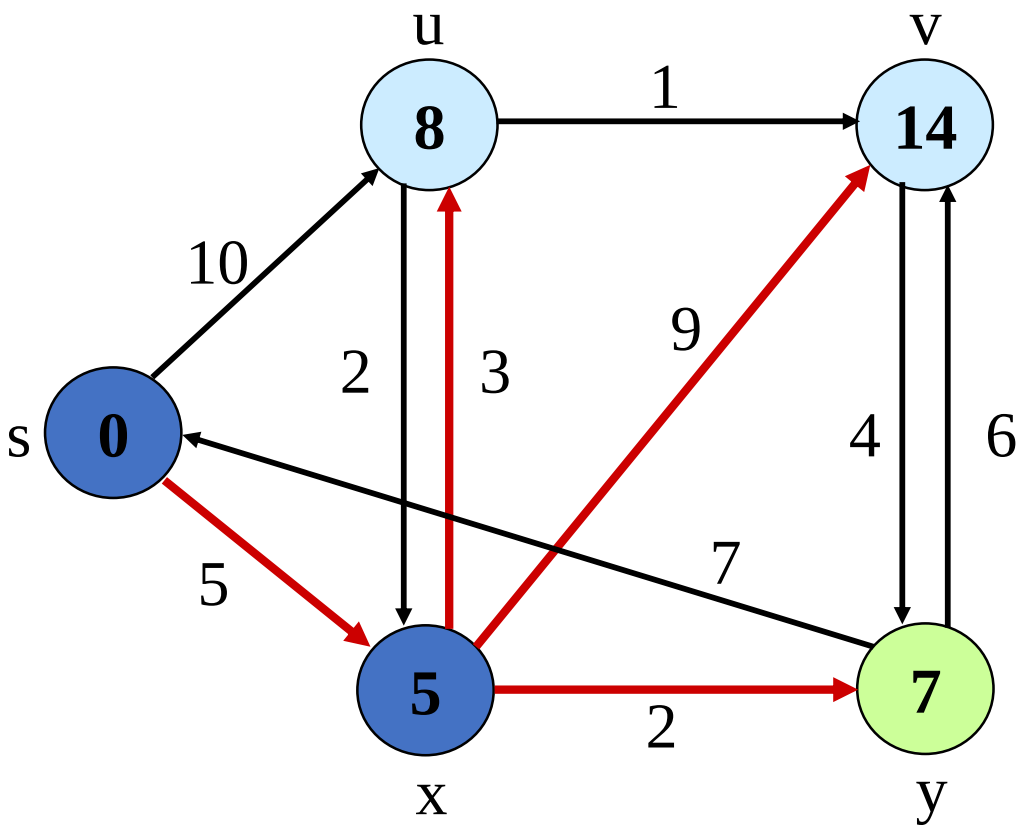
Example 2



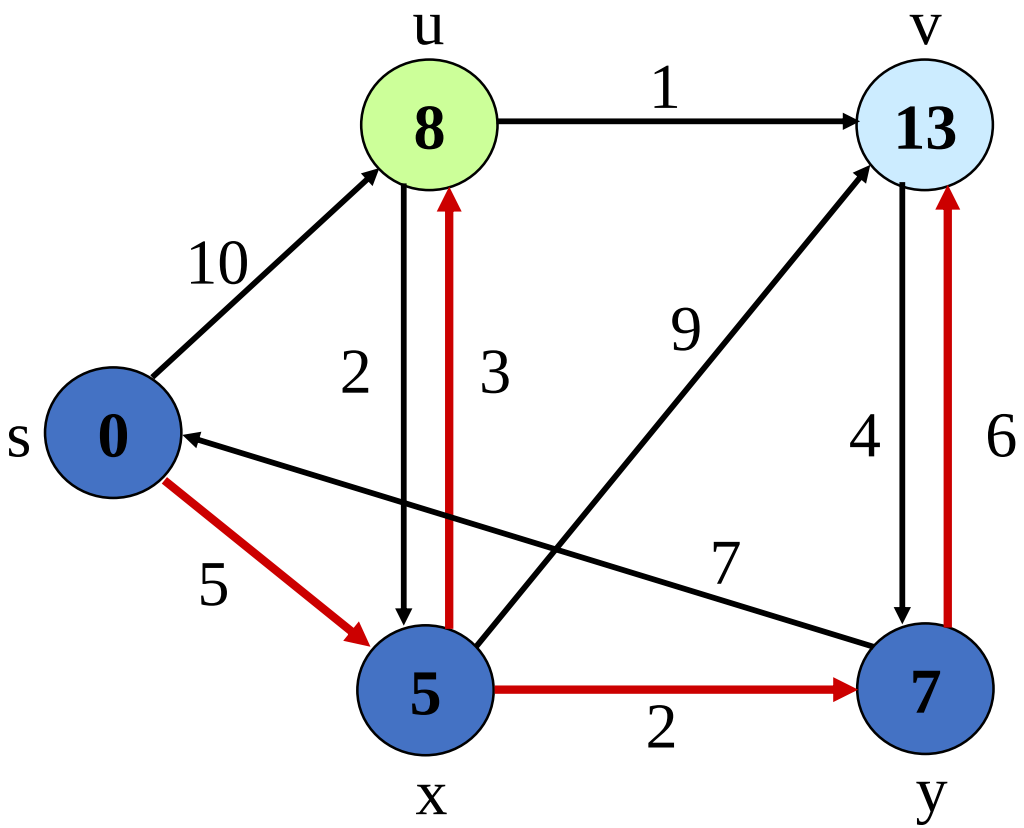
Example 2



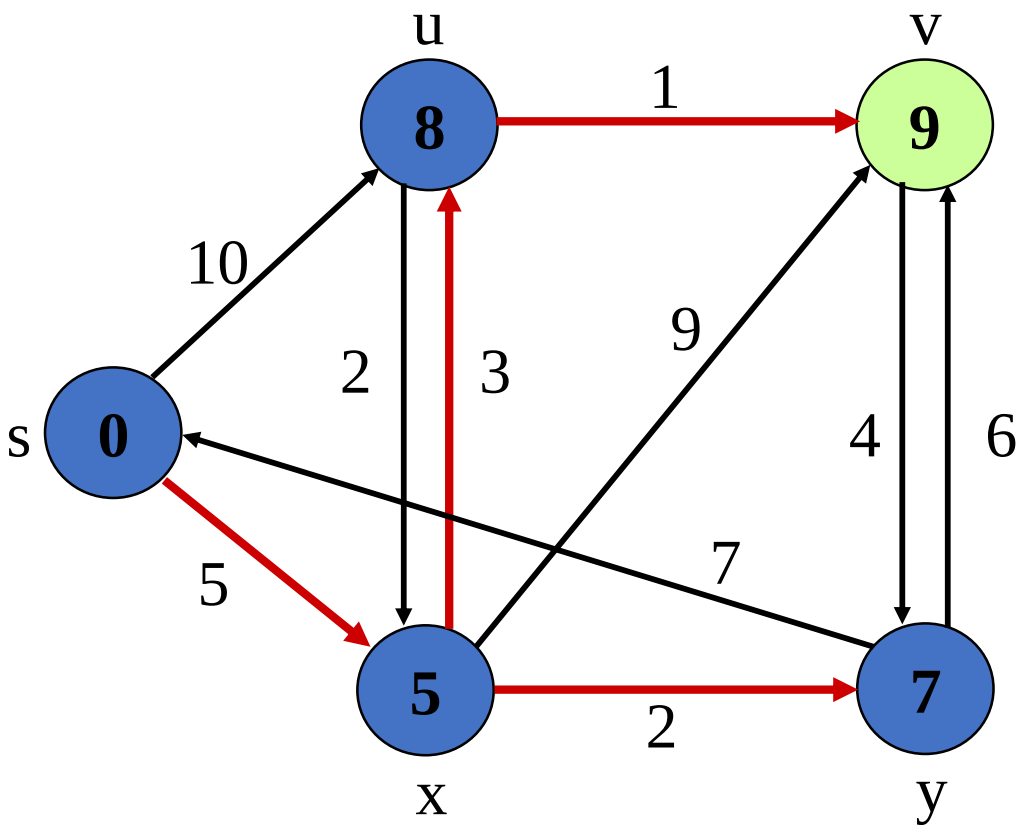
Example 2



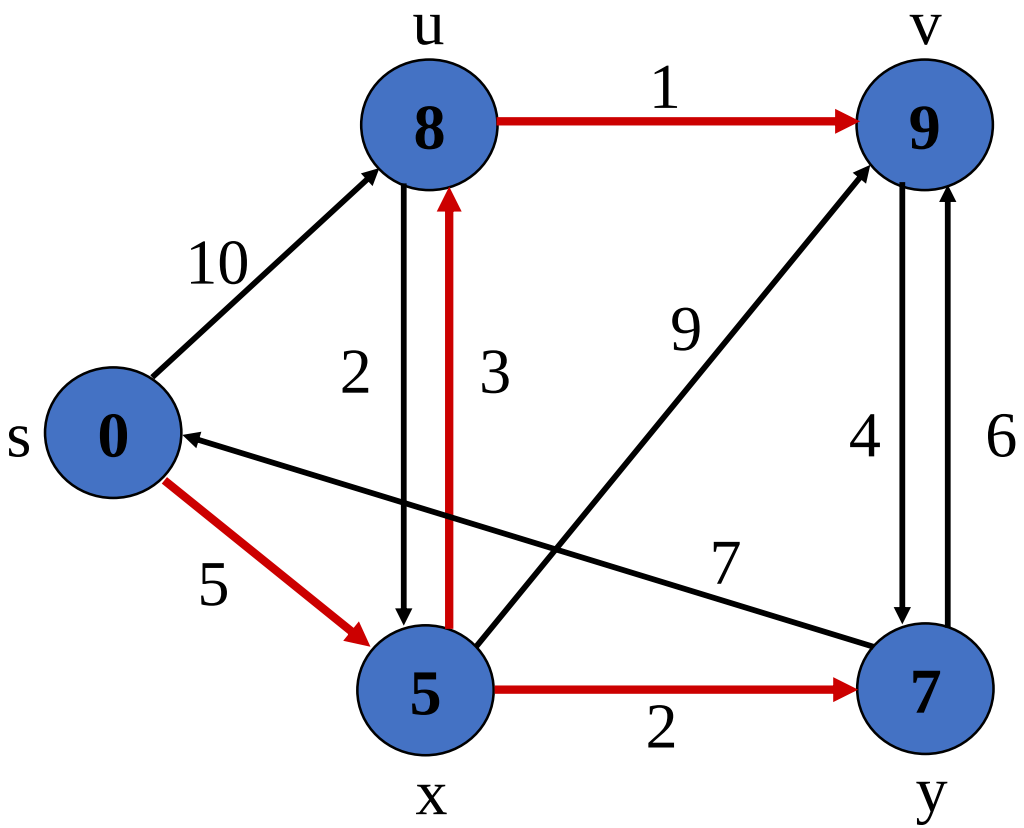
Example 2



Example 2



Example 2



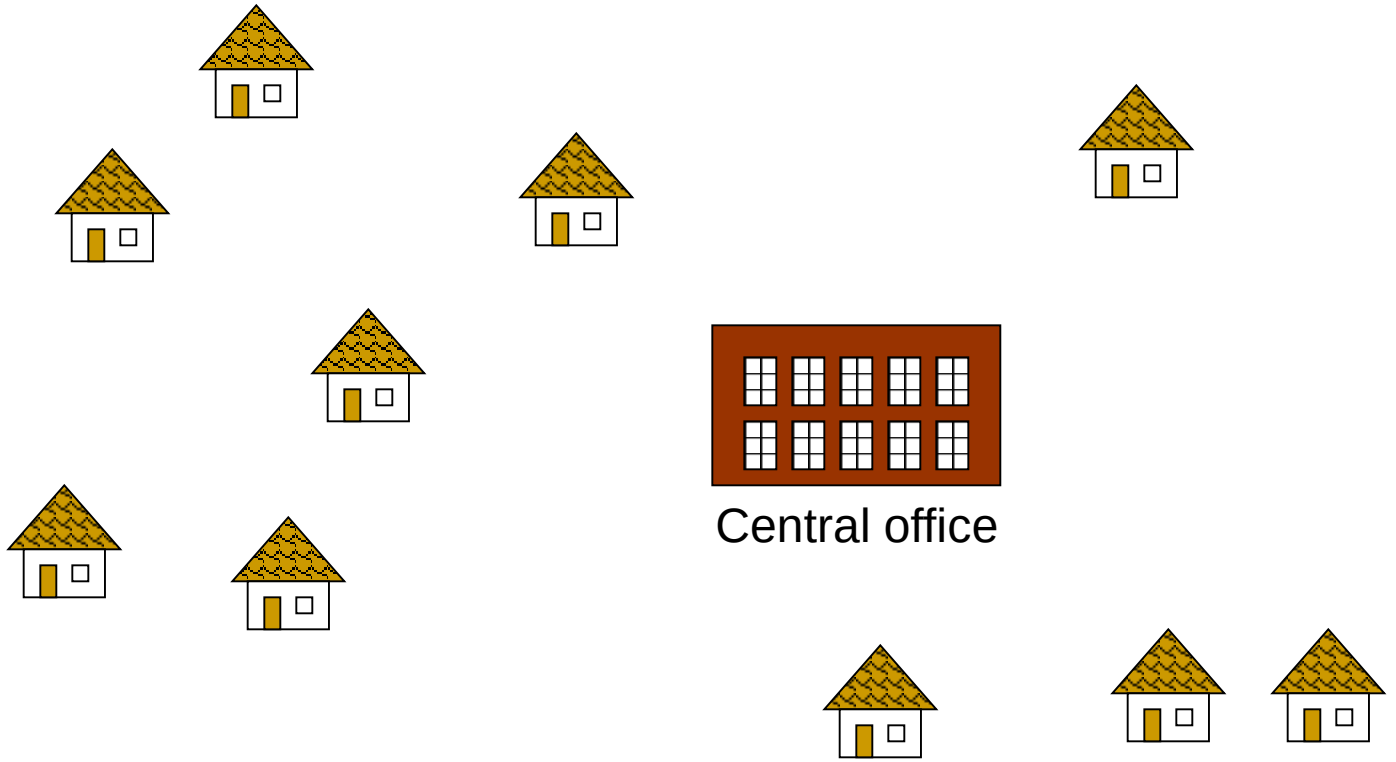
Minimum Spanning Tree(MST)

- 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?

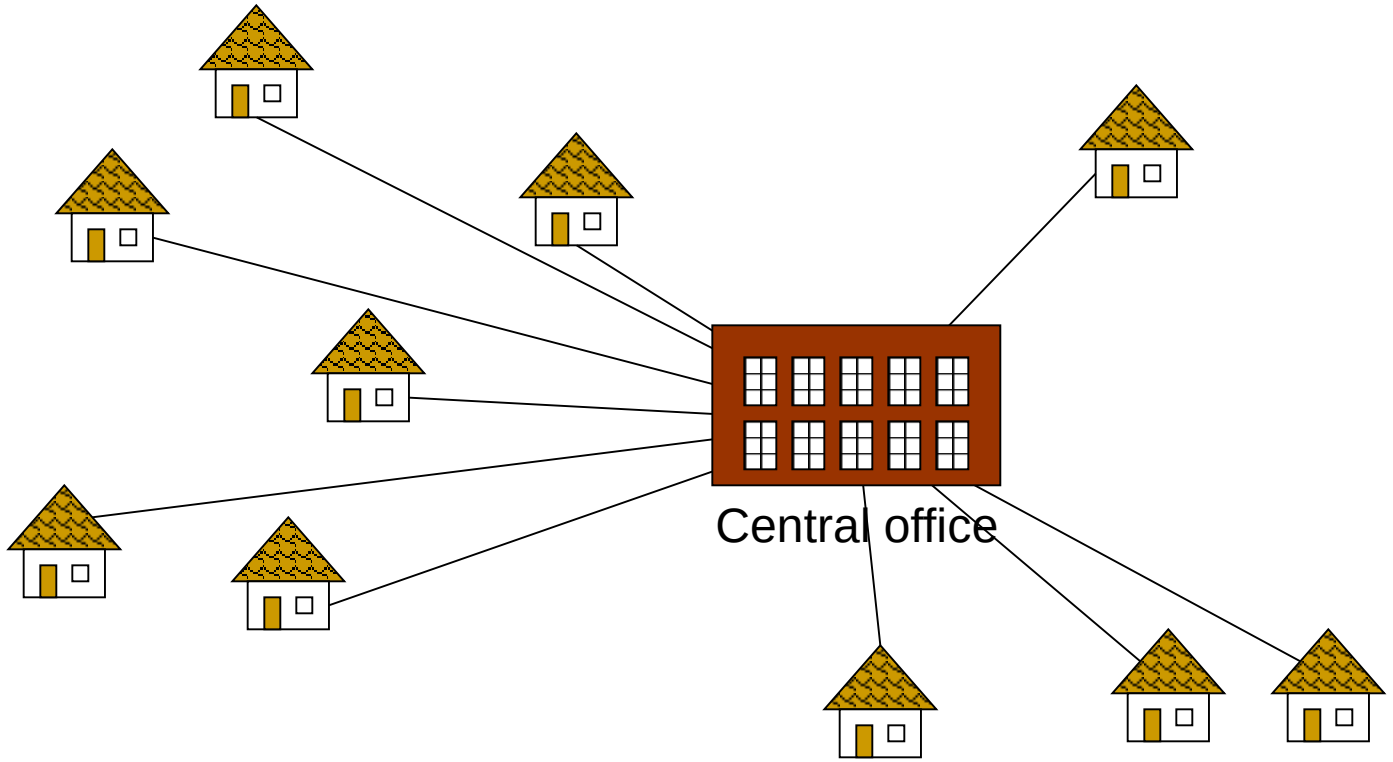
- We can solve this problem by finding a spanning tree so that the sum of the weights of the edges of the tree is minimized. Such a spanning tree is called a minimum spanning tree.

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

Problem: Laying Telephone Wire

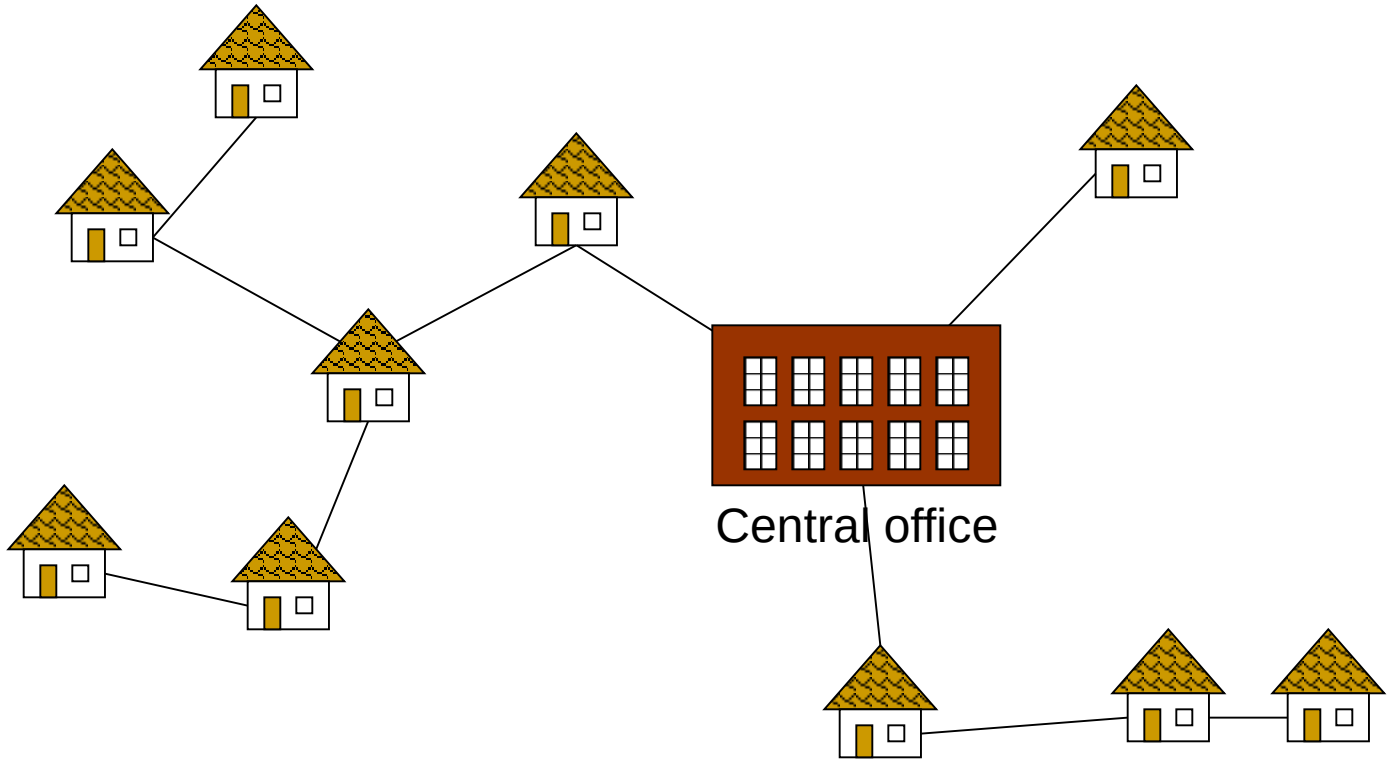


Wiring: Naïve Approach



Expensive!

Wiring: Better Approach



Minimize the total length of wire connecting the customers

MST Algorithms

- Two greedy approach algorithms for constructing MST :
- Prim’s Algorithm
- Kruskal Algorithm

Kruskal's algorithm -

- Select the edges in order of smallest weight and accept an edge if it does not cause a cycle

MST-KRUSKAL*(G,w)*

1 $A \leftarrow \emptyset$

2 **for each vertex *v* ∈ *V[G]***

3 **do MAKE-SET*(v)***

4 sort the edges of *E into non decreasing order by weight w*

5 **for each edge *(u, v) ∈ E, taken in non decreasing order by weight***

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

7 **then *A* ← *A* ∪ *{(u, v)}***

8 UNION*(u, v)*

9 **return *A***

Algorithm Explained

MAKE-SET*(x)*

p[x] ← *x*

rank[x] ← *0*

FIND-SET*(x)*

if *x != p[x]*

then *p[x]* ← *FIND-SET(p[x])*

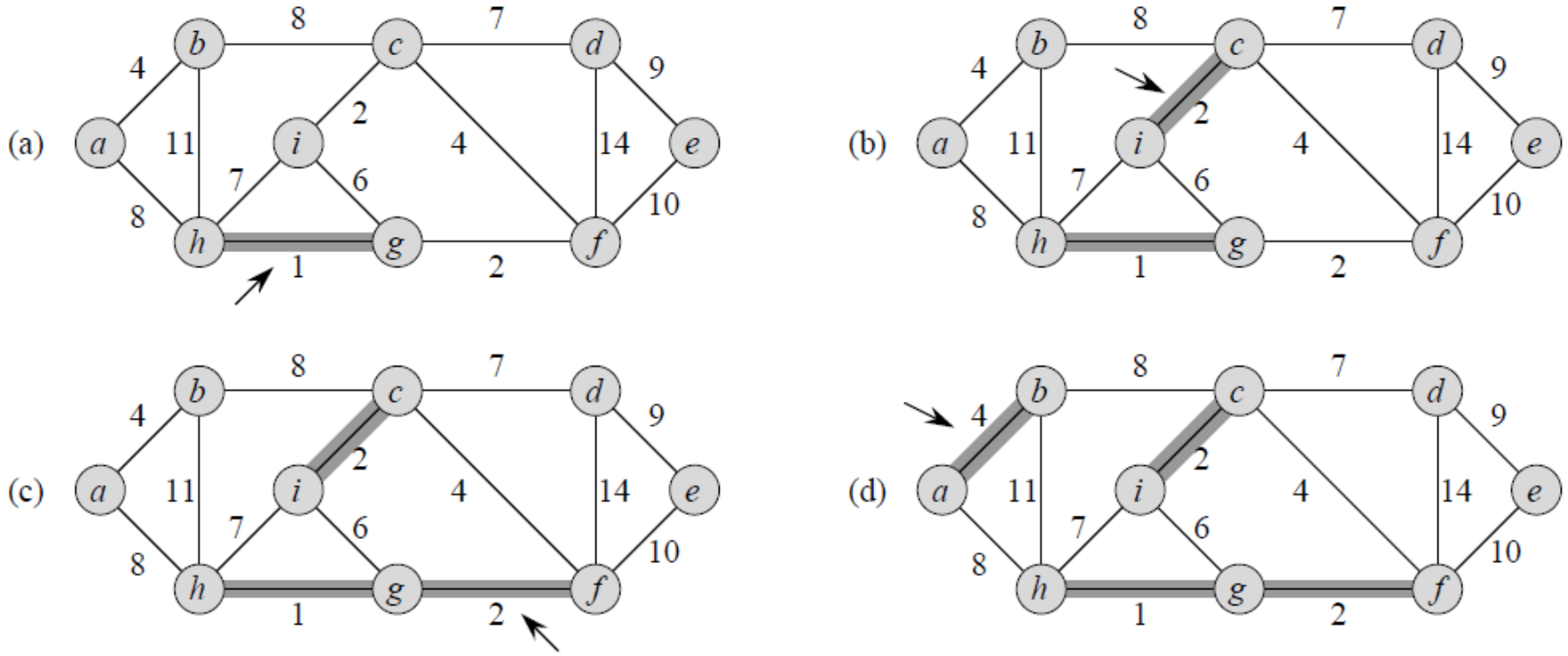
return *p[x]*

Algorithm Explained

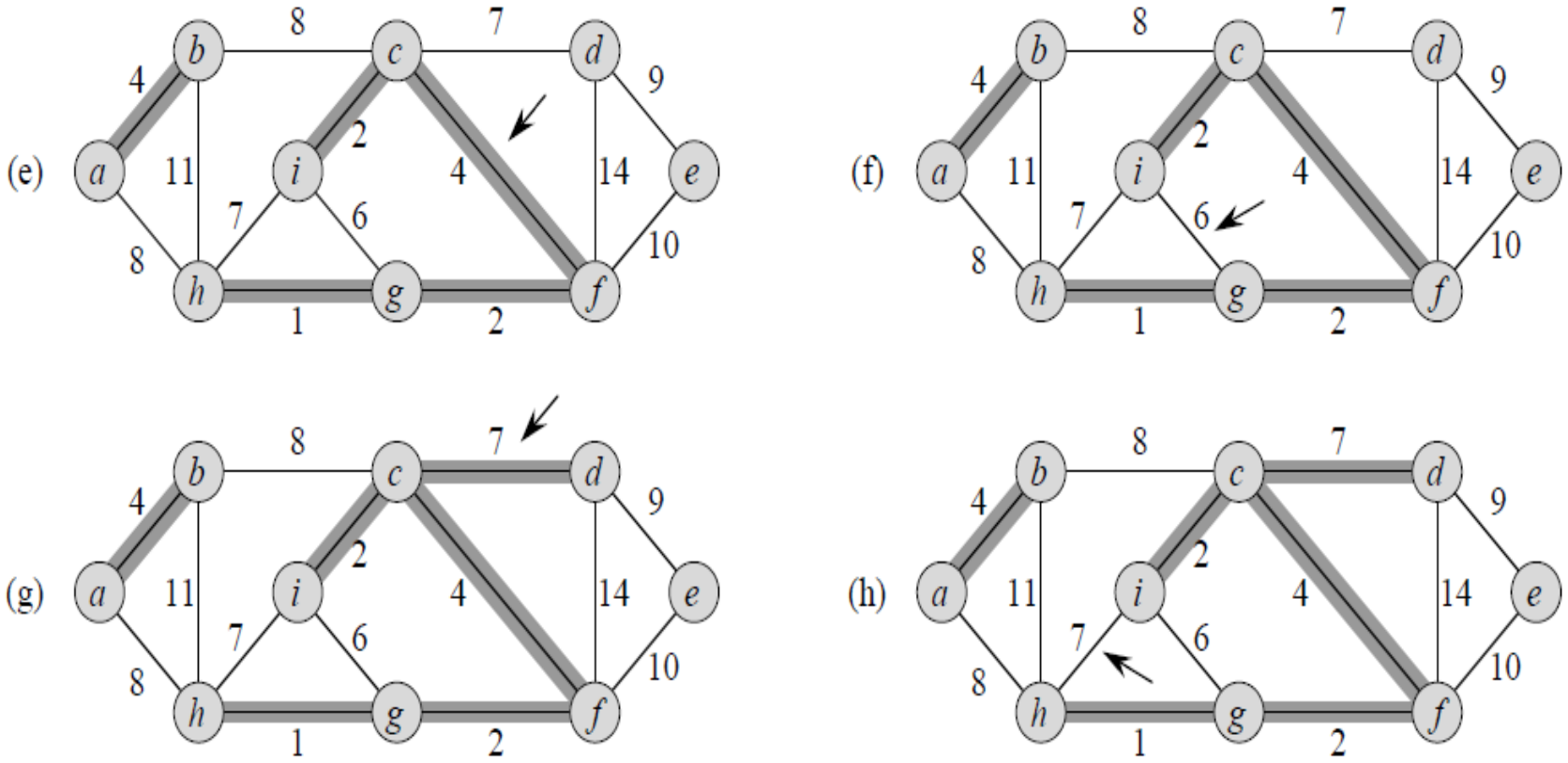
```
UNION(x, y)
1   LINK(FIND-SET(x), FIND-SET(y))

LINK(x, y)
1   if rank[x] > rank[y]
2       then p[y] ← x
3       else p[x] ← y
4            if rank[x] = rank[y]
5               then rank[y] ← rank[y] + 1
```

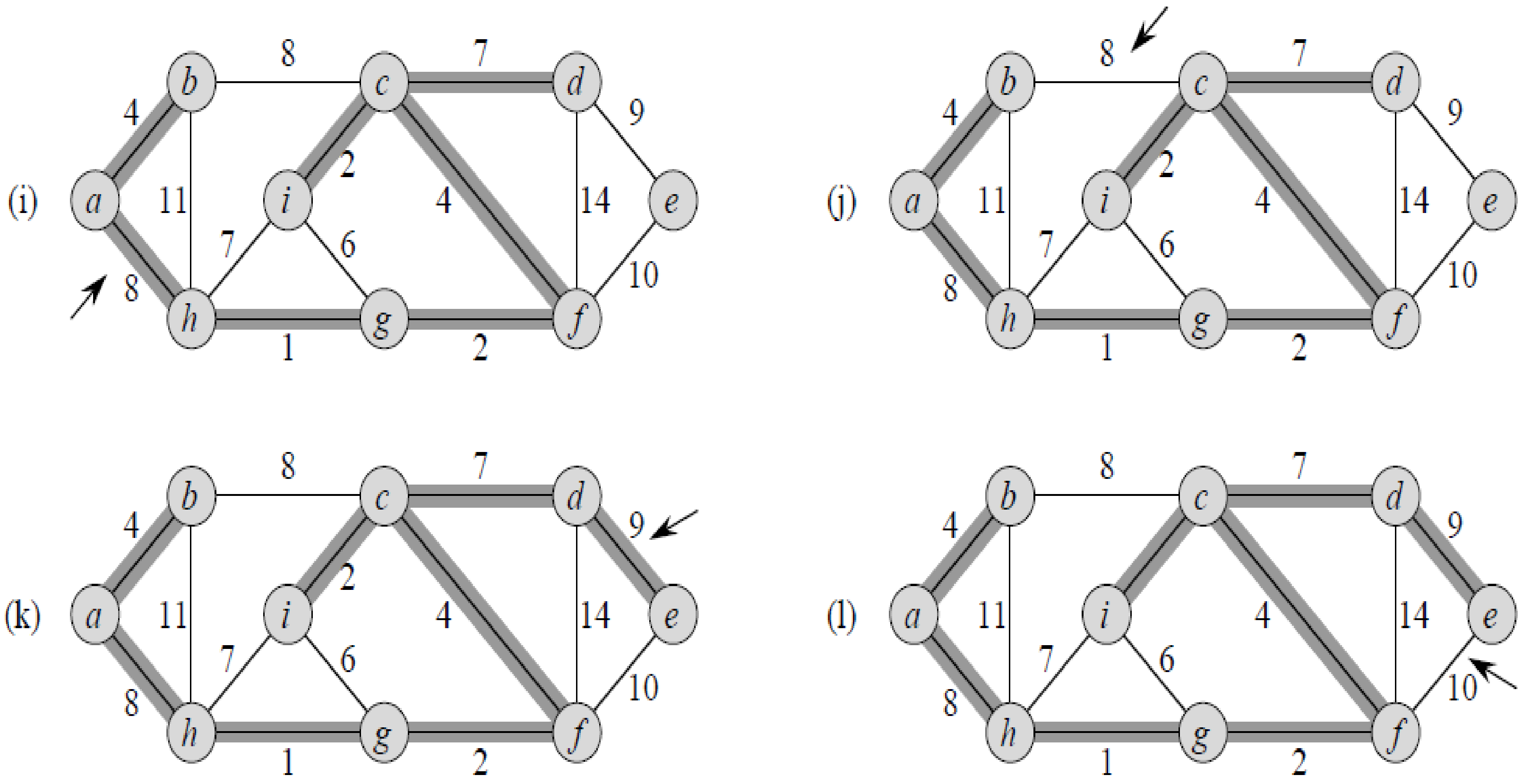
Example.



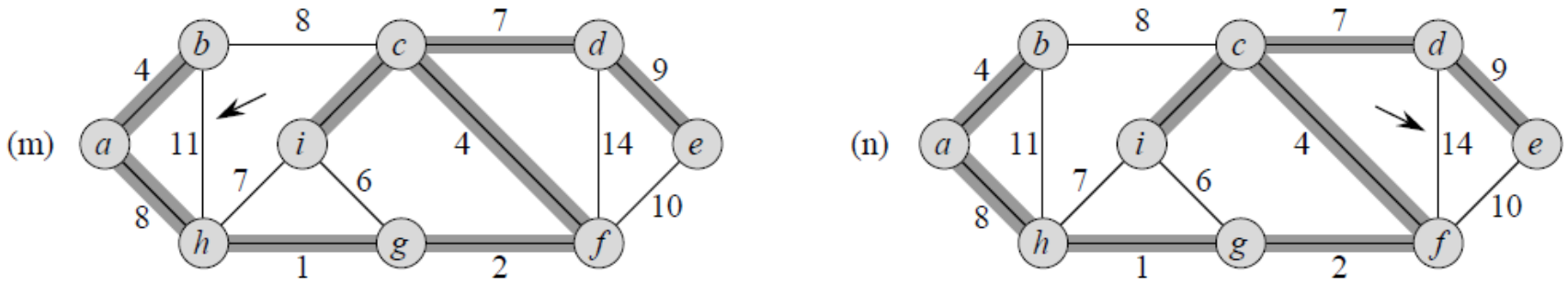
Example.



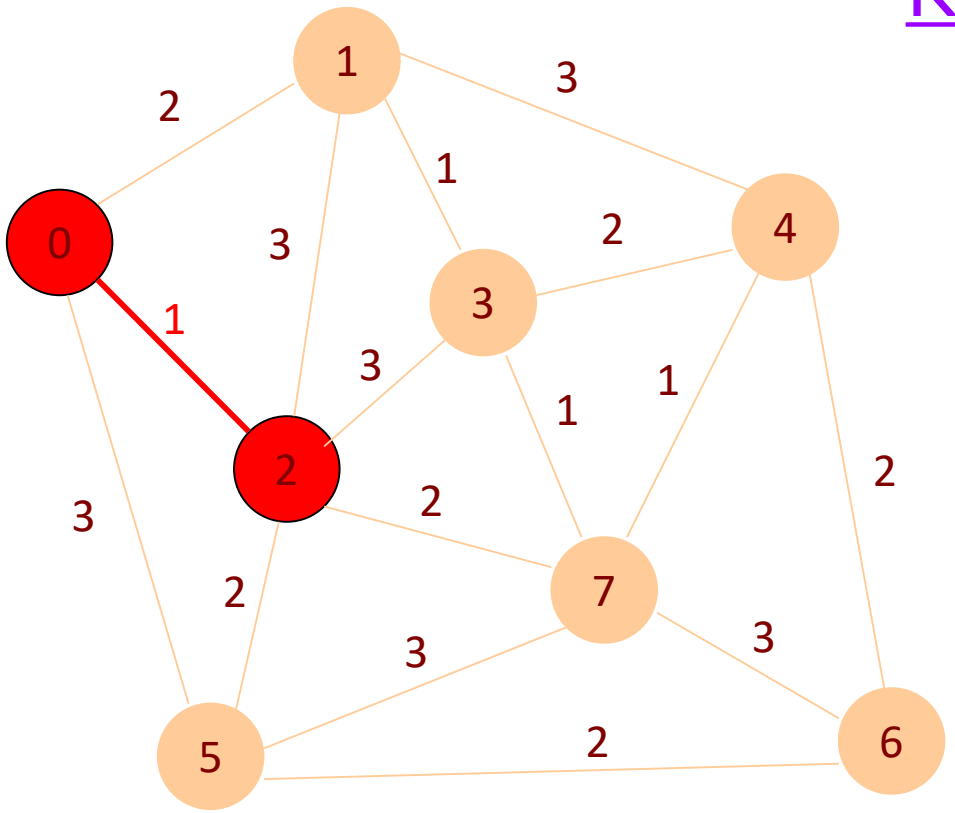
Example..



Example.


(m)
a
b
c
d
e
f
g
h
i
4
8
7
9
11
14
4
7
6
10
8
1
2
(n)
a
b
c
d
e
f
g
h
i
4
8
7
9
11
14
4
7
6
10
8
1
2

Minimum Spanning Tree(Example 2)

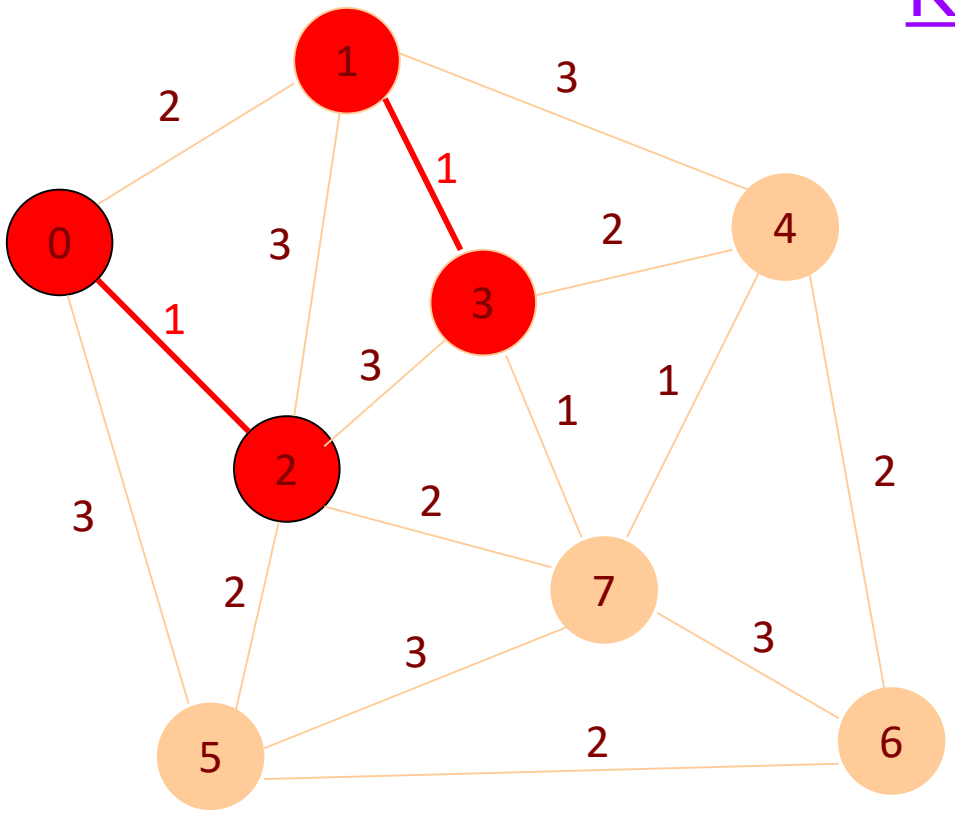


Kruskal's Algorithm

F = {0}, {1}, {2}, {3}, {4}, {5}, {6}, {7}

S = { 1: (0 - 2),
1: (1 – 3),
1: (3 – 7),
1: (4 – 7),
2: (0 – 1),
2: (3 – 4),
2: (2 – 7),
2: (4 – 6),
2: (5 – 6),
2: (2 – 5),
3: (1 – 2),
3: (1 – 4),
3: (2 – 3),
3: (6 – 7),
3: (0 – 5),
3: (5 – 7) }

Minimum Spanning Tree

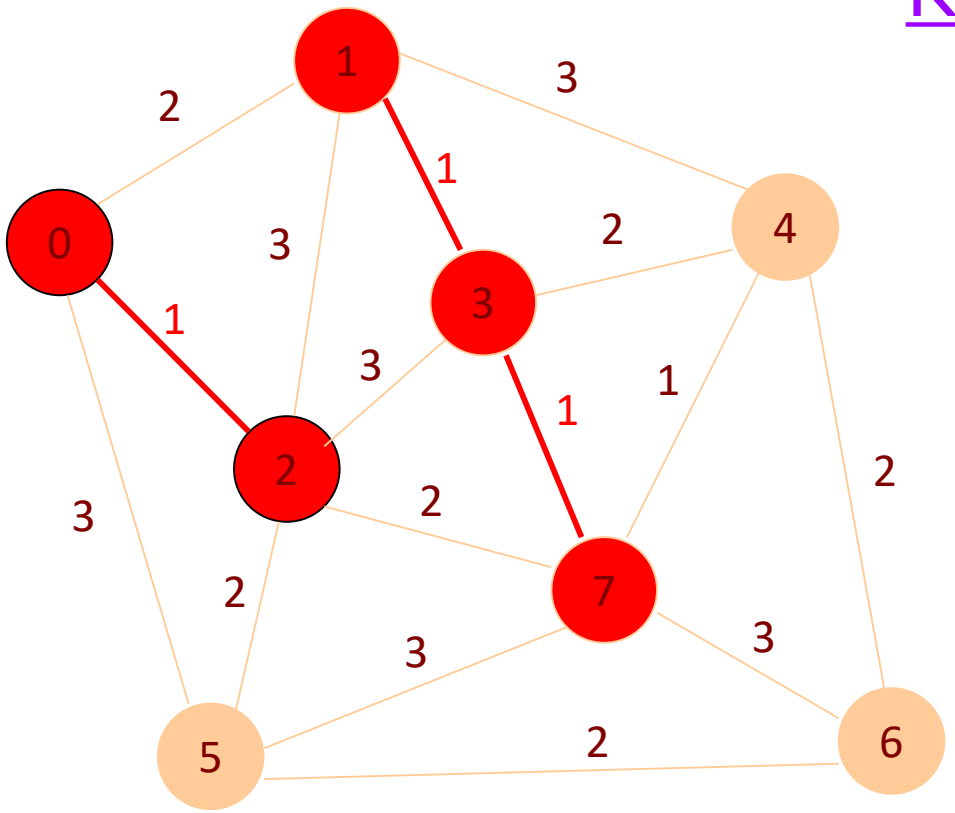


Kruskal's Algorithm

F = {0, 2}, {1}, {3}, {4}, {5}, {6}, {7}

S = { 1: (0 - 2),
1: (1 – 3),
1: (3 – 7),
1: (4 – 7),
2: (0 – 1),
2: (3 – 4),
2: (2 – 7),
2: (4 – 6),
2: (5 – 6),
2: (2 – 5),
3: (1 – 2),
3: (1 – 4),
3: (2 – 3),
3: (6 – 7),
3: (0 – 5),
3: (5 – 7) }

Minimum Spanning Tree

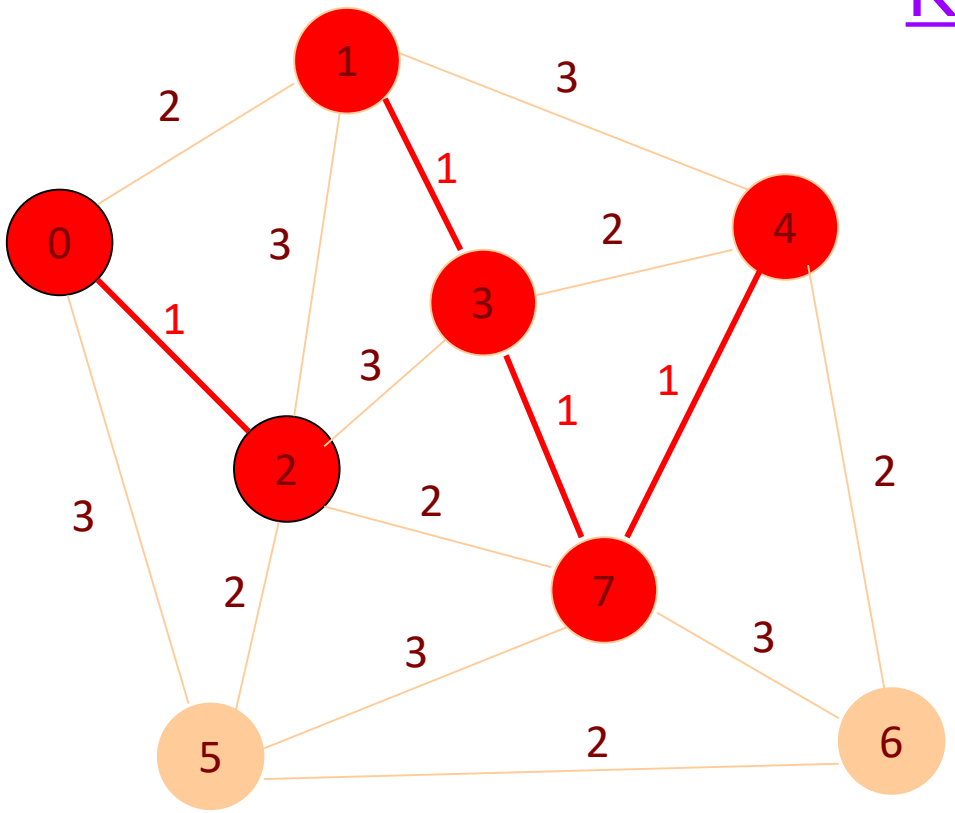


Kruskal's Algorithm

F = {0, 2} {1, 3}, {4}, {5}, {6}, {7}

S = { 1: (0 - 2),
1: (1 – 3),
1: (3 – 7),
1: (4 – 7),
2: (0 – 1),
2: (3 – 4),
2: (2 – 7),
2: (4 – 6),
2: (5 – 6),
2: (2 – 5),
3: (1 – 2),
3: (1 – 4),
3: (2 – 3),
3: (6 – 7),
3: (0 – 5),
3: (5 – 7) }

Minimum Spanning Tree

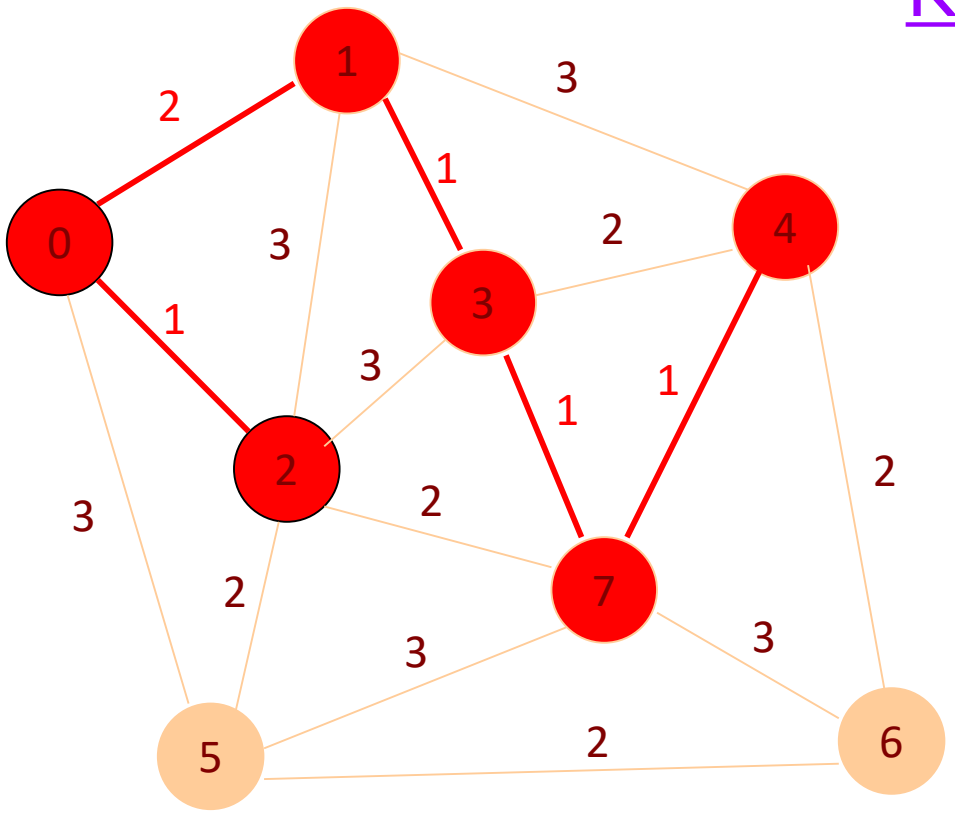


Kruskal's Algorithm

F = {0, 2} {1, 3, 7}, {4}, {5}, {6}

S = { 1: (0 - 2),
1: (1 – 3),
1: (3 – 7),
1: (4 – 7),
2: (0 – 1),
2: (3 – 4),
2: (2 – 7),
2: (4 – 6),
2: (5 – 6),
2: (2 – 5),
3: (1 – 2),
3: (1 – 4),
3: (2 – 3),
3: (6 – 7),
3: (0 – 5),
3: (5 – 7) }

Minimum Spanning Tree

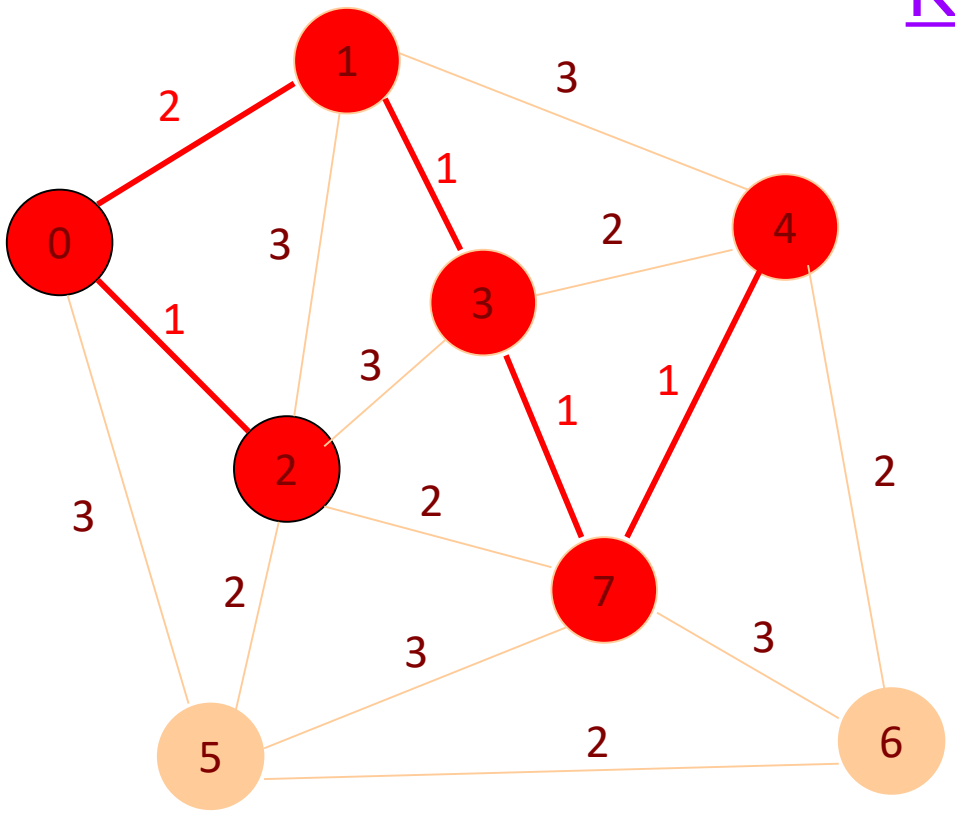


Kruskal's Algorithm

F = {0, 2} {1, 3, 4, 7}, {5}, {6}

S = { 1: (0 - 2),
1: (1 – 3),
1: (3 – 7),
1: (4 – 7),
2: (0 – 1),
2: (3 – 4),
2: (2 – 7),
2: (4 – 6),
2: (5 – 6),
2: (2 – 5),
3: (1 – 2),
3: (1 – 4),
3: (2 – 3),
3: (6 – 7),
3: (0 – 5),
3: (5 – 7) }

Minimum Spanning Tree

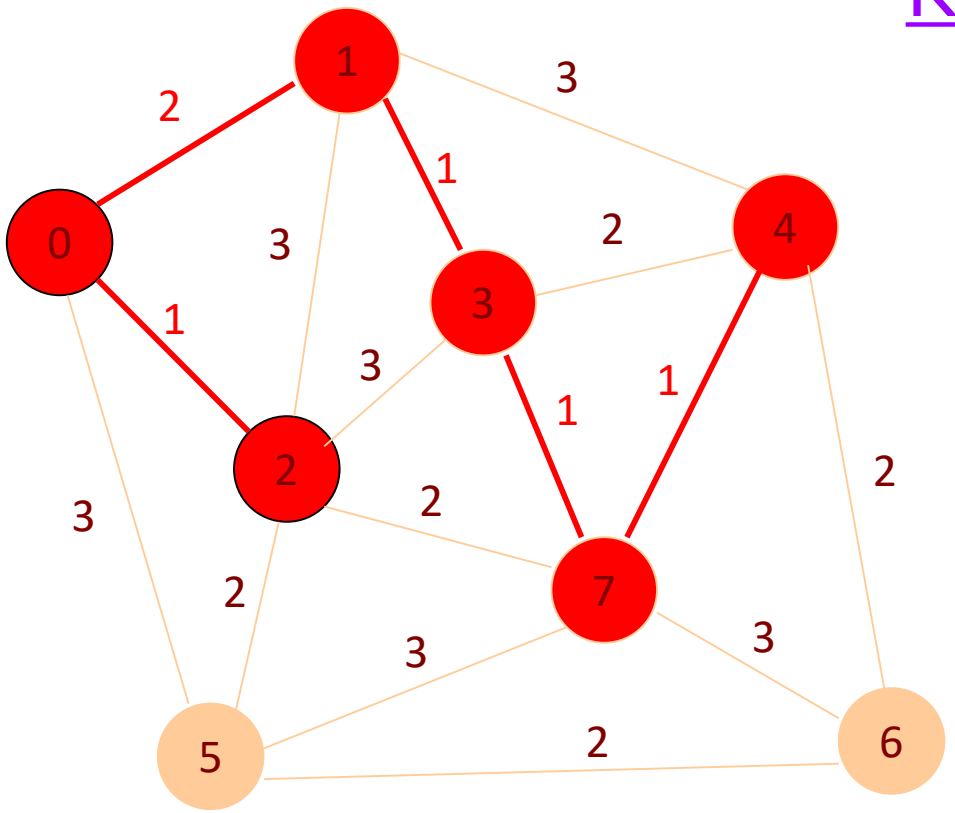


Kruskal's Algorithm

F = {0,1, 2, 3, 4, 7}, {5}, {6}

S = { 1: (0 - 2),
1: (1 – 3),
1: (3 – 7),
1: (4 – 7),
2: (0 – 1),
2: (3 – 4), Rejected
2: (2 – 7),
2: (4 – 6),
2: (5 – 6),
2: (2 – 5),
3: (1 – 2),
3: (1 – 4),
3: (2 – 3),
3: (6 – 7),
3: (0 – 5),
3: (5 – 7) }

Minimum Spanning Tree

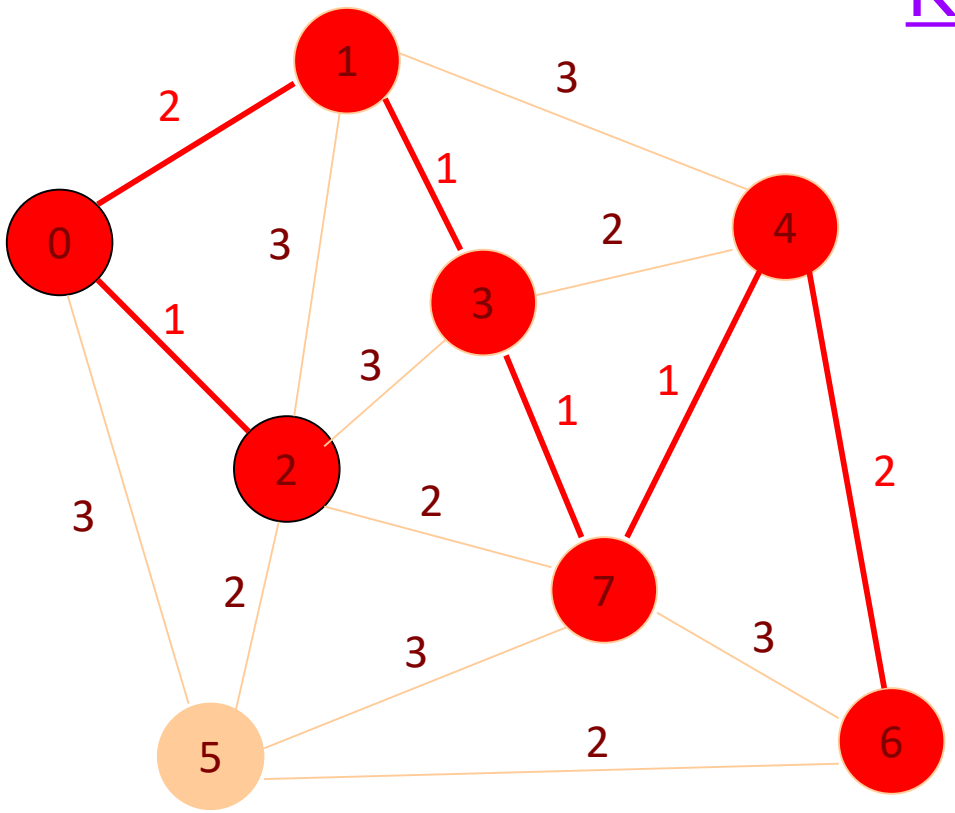


Kruskal's Algorithm

F = {0,1, 2, 3, 4, 7}, {5}, {6}

S = { 1: (0 - 2),
1: (1 – 3),
1: (3 – 7),
1: (4 – 7),
2: (0 – 1),
2: (3 – 4), Rejected
2: (2 – 7), Rejected
2: (4 – 6),
2: (5 – 6),
2: (2 – 5),
3: (1 – 2),
3: (1 – 4),
3: (2 – 3),
3: (6 – 7),
3: (0 – 5),
3: (5 – 7) }

Minimum Spanning Tree

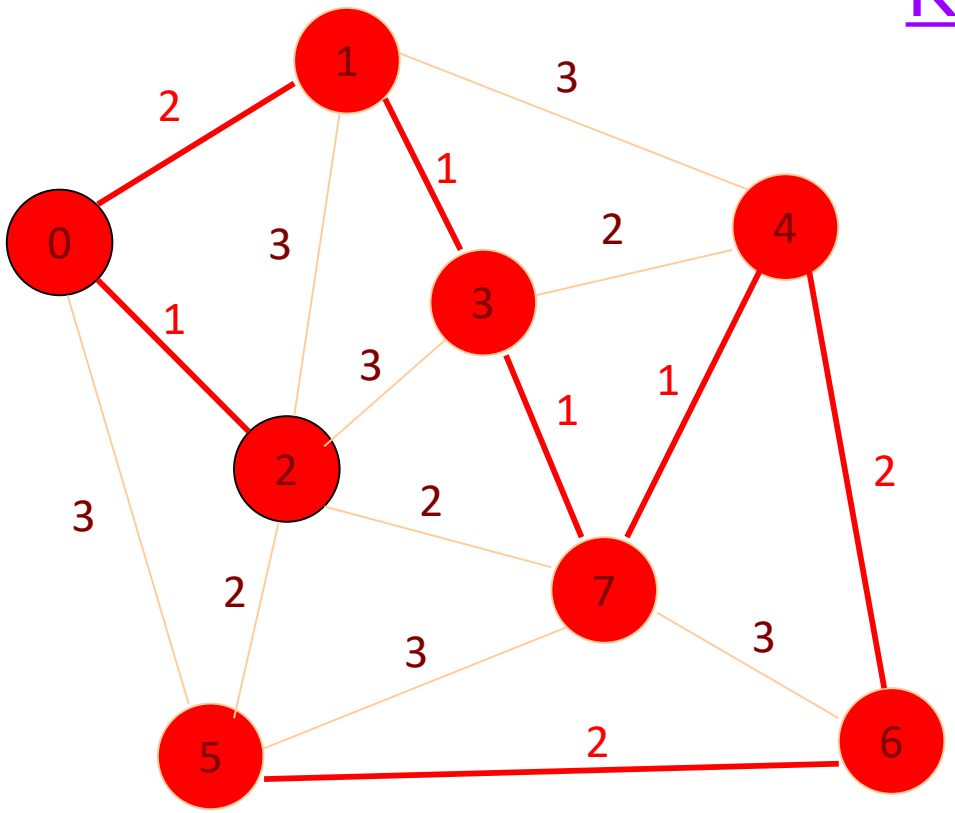


Kruskal's Algorithm

F = {0,1, 2, 3, 4, 7}, {5}, {6}

S = { 1: (0 - 2),
1: (1 – 3),
1: (3 – 7),
1: (4 – 7),
2: (0 – 1),
2: (3 – 4), Rejected
2: (2 – 7), Rejected
2: (4 – 6),
2: (5 – 6),
2: (2 – 5),
3: (1 – 2),
3: (1 – 4),
3: (2 – 3),
3: (6 – 7),
3: (0 – 5),
3: (5 – 7) }

Minimum Spanning Tree

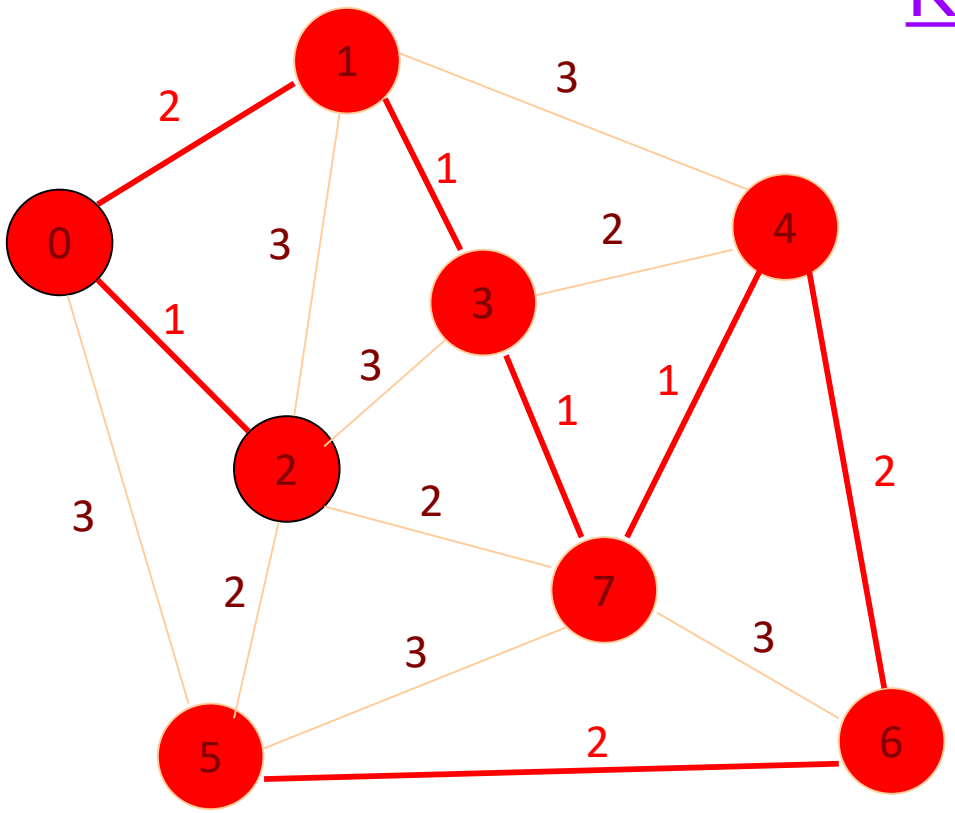


Kruskal's Algorithm

F = {0,1, 2, 3, 4, 6, 7}, {5}

S = { 1: (0 - 2),
1: (1 – 3),
1: (3 – 7),
1: (4 – 7),
2: (0 – 1),
2: (3 – 4), Rejected
2: (2 – 7), Rejected
2: (4 – 6),
2: (5 – 6),
2: (2 – 5),
3: (1 – 2),
3: (1 – 4),
3: (2 – 3),
3: (6 – 7),
3: (0 – 5),
3: (5 – 7) }

Minimum Spanning Tree

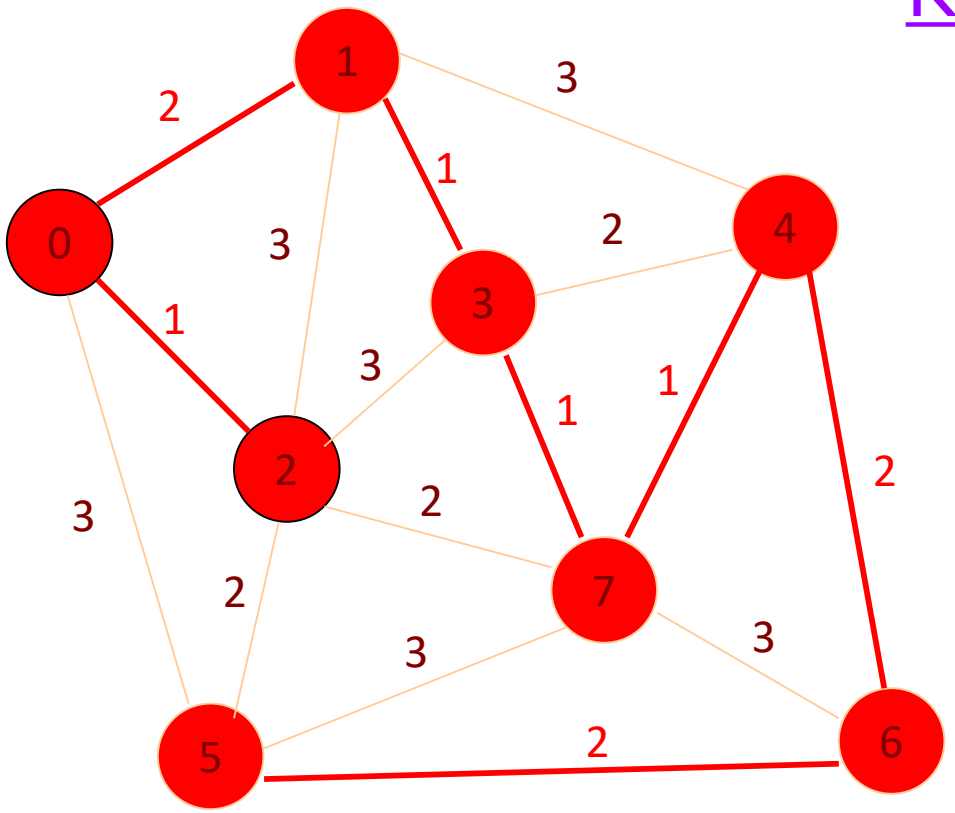


Kruskal's Algorithm

F = {0,1, 2, 3, 4, 5, 6, 7}

S = { 1: (0 - 2),
1: (1 – 3),
1: (3 – 7),
1: (4 – 7),
2: (0 – 1),
2: (3 – 4), Rejected
2: (2 – 7), Rejected
2: (4 – 6),
2: (5 – 6),
2: (2 – 5), Rejected
3: (1 – 2),
3: (1 – 4),
3: (2 – 3),
3: (6 – 7),
3: (0 – 5),
3: (5 – 7) }

Minimum Spanning Tree

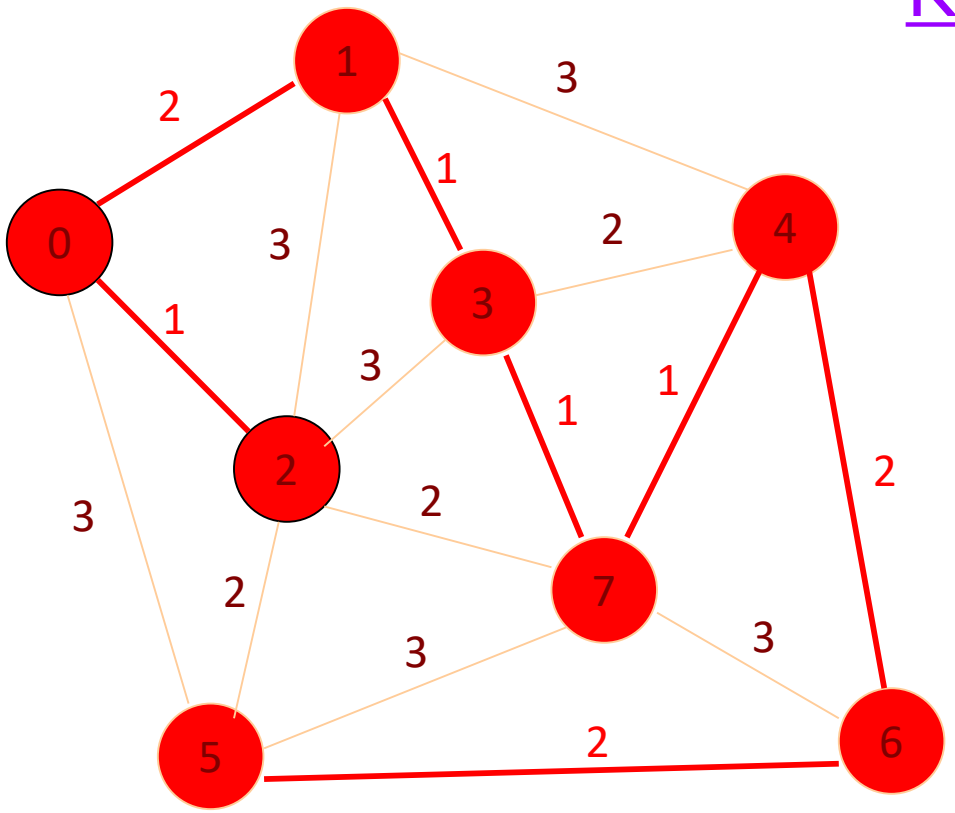


Kruskal's Algorithm

F = {0,1, 2, 3, 4, 5, 6, 7}

S = { 1: (0 - 2),
1: (1 – 3),
1: (3 – 7),
1: (4 – 7),
2: (0 – 1),
2: (3 – 4), Rejected
2: (2 – 7), Rejected
2: (4 – 6),
2: (5 – 6),
2: (2 – 5), Rejected
3: (1 – 2), Rejected
3: (1 – 4),
3: (2 – 3),
3: (6 – 7),
3: (0 – 5),
3: (5 – 7) }

Minimum Spanning Tree

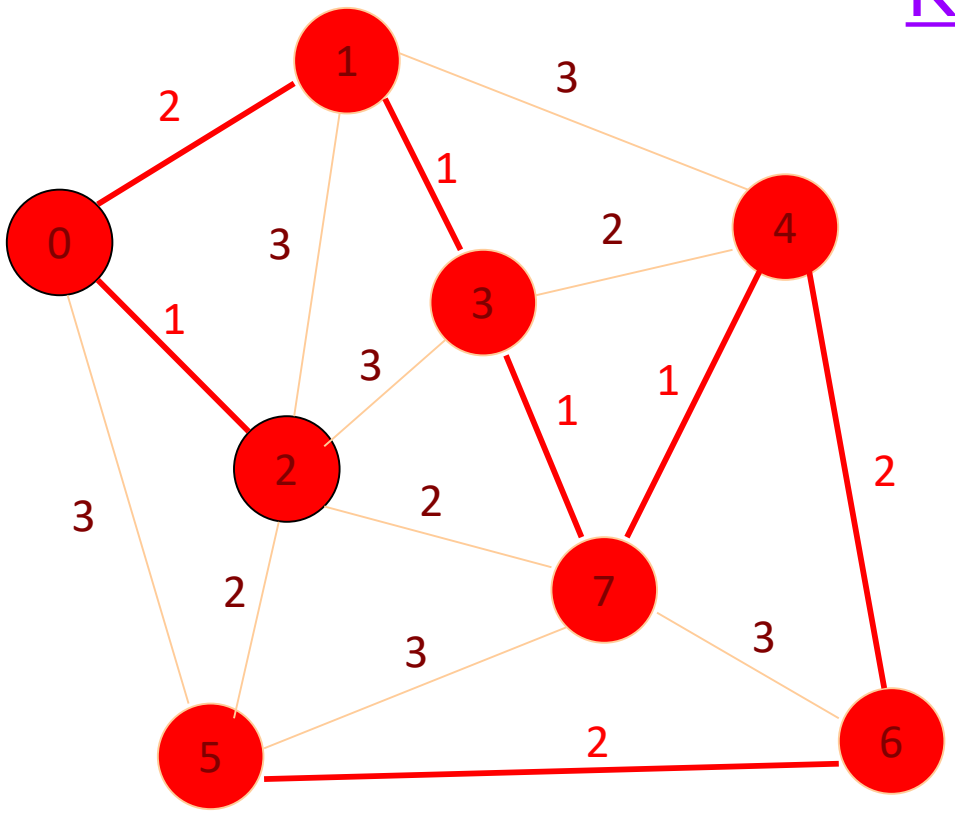


Kruskal's Algorithm

F = {0,1, 2, 3, 4, 5, 6, 7}

S = { 1: (0 - 2),
1: (1 – 3),
1: (3 – 7),
1: (4 – 7),
2: (0 – 1),
2: (3 – 4), Rejected
2: (2 – 7), Rejected
2: (4 – 6),
2: (5 – 6),
2: (2 – 5), Rejected
3: (1 – 2), Rejected
3: (1 – 4), Rejected
3: (2 – 3),
3: (6 – 7),
3: (0 – 5),
3: (5 – 7) }

Minimum Spanning Tree

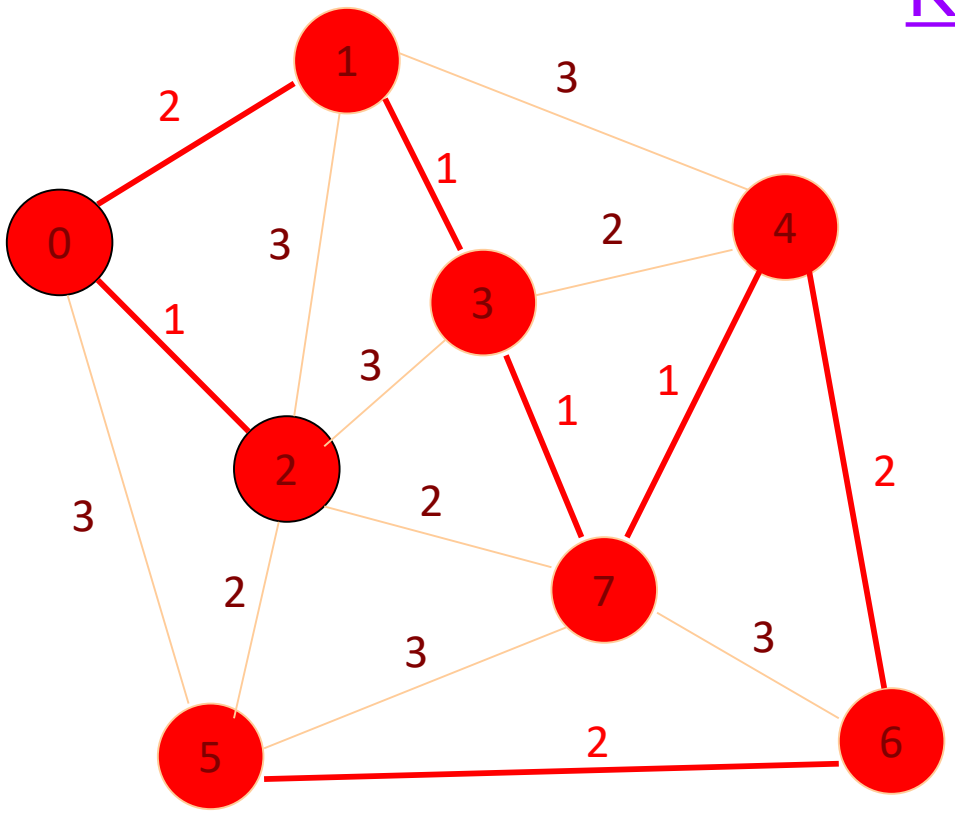


Kruskal's Algorithm

F = {0,1, 2, 3, 4, 5, 6, 7}

S = { 1: (0 - 2),
1: (1 – 3),
1: (3 – 7),
1: (4 – 7),
2: (0 – 1),
2: (3 – 4), Rejected
2: (2 – 7), Rejected
2: (4 – 6),
2: (5 – 6),
2: (2 – 5), Rejected
3: (1 – 2), Rejected
3: (1 – 4), Rejected
3: (2 – 3), Rejected
3: (6 – 7),
3: (0 – 5),
3: (5 – 7) }

Minimum Spanning Tree

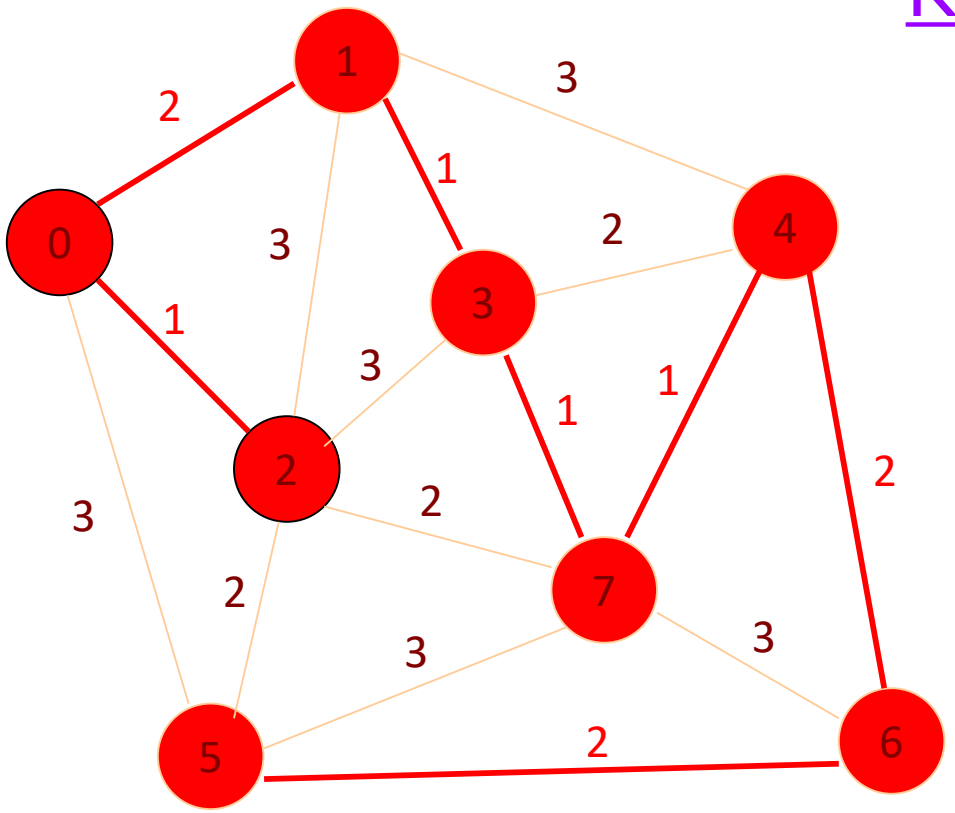


Kruskal's Algorithm

F = {0,1, 2, 3, 4, 5, 6, 7}

S = { 1: (0 - 2),
1: (1 – 3),
1: (3 – 7),
1: (4 – 7),
2: (0 – 1),
2: (3 – 4), Rejected
2: (2 – 7), Rejected
2: (4 – 6),
2: (5 – 6),
2: (2 – 5), Rejected
3: (1 – 2), Rejected
3: (1 – 4), Rejected
3: (2 – 3), Rejected
3: (6 – 7), Rejected
3: (0 – 5),
3: (5 – 7) }

Minimum Spanning Tree

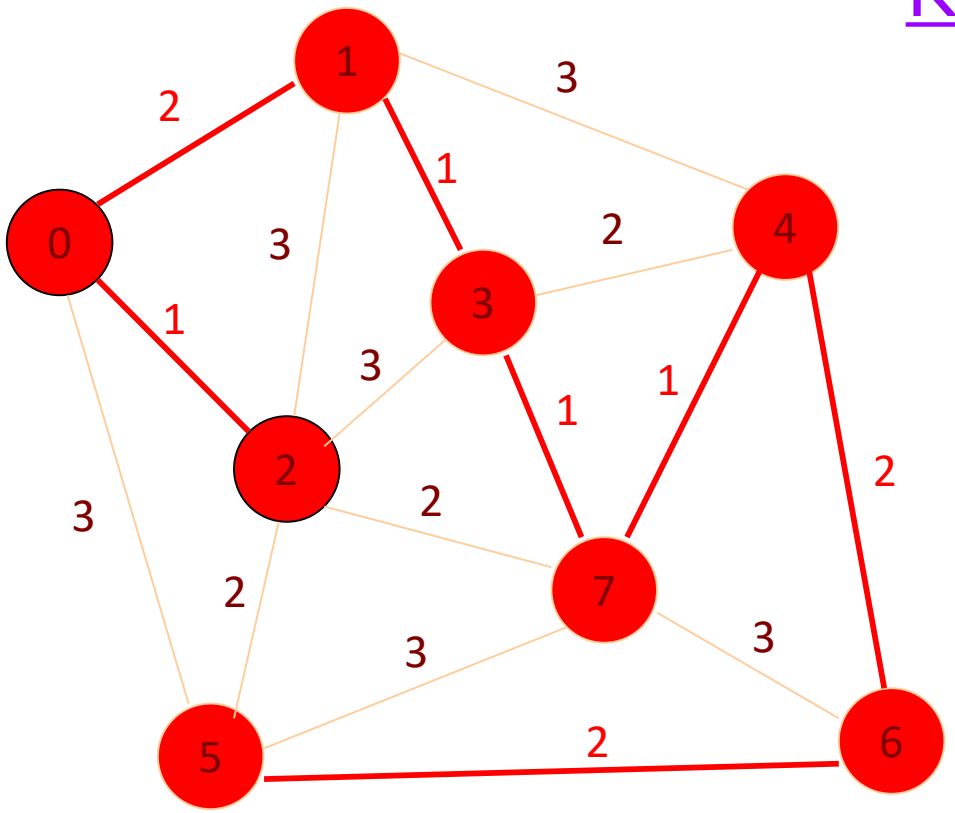


Kruskal's Algorithm

F = {0,1, 2, 3, 4, 5, 6, 7}

S = { 1: (0 - 2),
1: (1 – 3),
1: (3 – 7),
1: (4 – 7),
2: (0 – 1),
2: (3 – 4), Rejected
2: (2 – 7), Rejected
2: (4 – 6),
2: (5 – 6),
2: (2 – 5), Rejected
3: (1 – 2), Rejected
3: (1 – 4), Rejected
3: (2 – 3), Rejected
3: (6 – 7), Rejected
3: (0 – 5), Rejected
3: (5 – 7) }

Minimum Spanning Tree



Kruskal's Algorithm

F = {0,1, 2, 3, 4, 5, 6, 7}

S = { 1: (0 - 2),
1: (1 – 3),
1: (3 – 7),
1: (4 – 7),
2: (0 – 1),
2: (3 – 4), Rejected
2: (2 – 7), Rejected
2: (4 – 6),
2: (5 – 6),
2: (2 – 5), Rejected
3: (1 – 2), Rejected
3: (1 – 4), Rejected
3: (2 – 3), Rejected
3: (6 – 7), Rejected
3: (0 – 5), Rejected
3: (5 – 7) Rejected }

Prim's algorithm
(minimum spanning tree)

- Grow the tree in successive stages
- At each stage, a new vertex is added to the tree by choosing the edge (u, v) such that the cost of (u, v) is the smallest among all edges where u is in the tree and v is not.

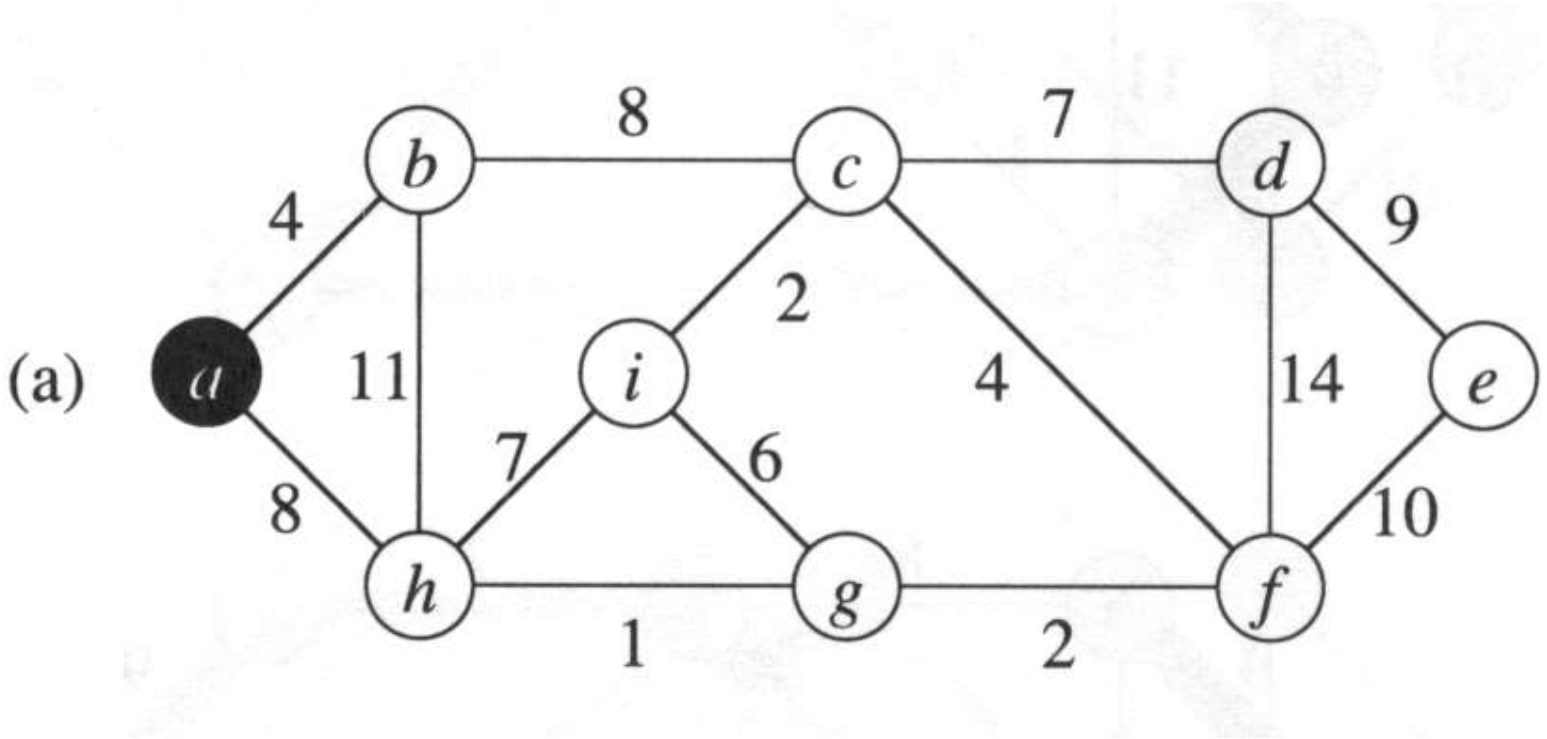
- EXTRACT-MIN(Q) deletes the node from array/queue Q whose key is minimum, returning a pointer to the node.

- MST-PRIM*(G,w, r)*
- 1 **for each *u ∈ V[G]***
- 2 **do *key[u]←∞***
- 3 *π[u]← NIL*
- 4 *key[r] ← 0*
- 5 *Q ← V[G]*
- 6 **while *Q! = Ø***
- 7 **do *u ← EXTRACT-MIN(Q)***
- 8 **for each *v ∈ Adj[u]***
- 9 **do if *v ∈ Q and w(u, v) < key[v]***
- 10 **then *π[v]← u***
- 11 *key[v] ← w(u, v)*

MST

Minimum spanning tree: (Prim's algorithm)

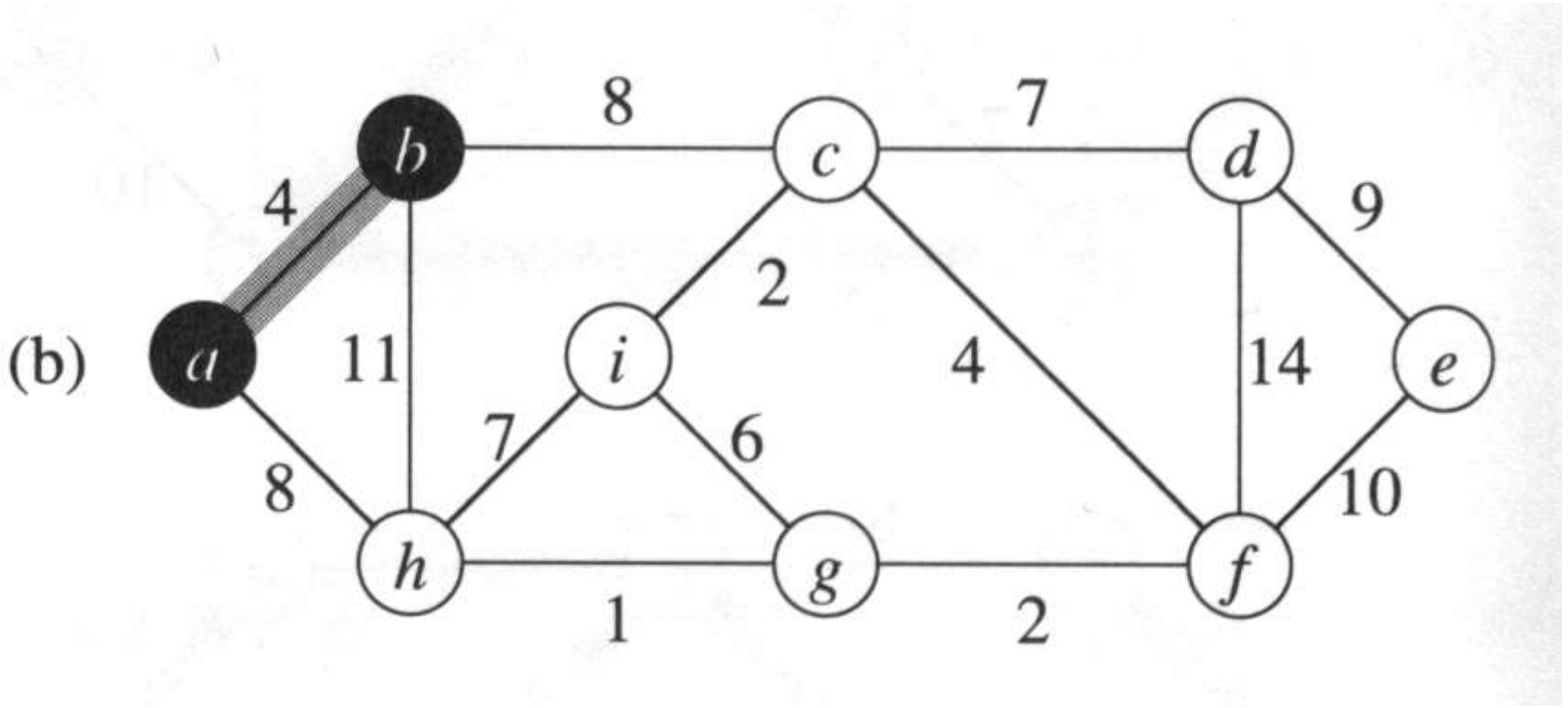
Start anywhere and repeatedly choose the next-smallest edge out from your current tree.



MST

Minimum spanning tree: (Prim's algorithm)

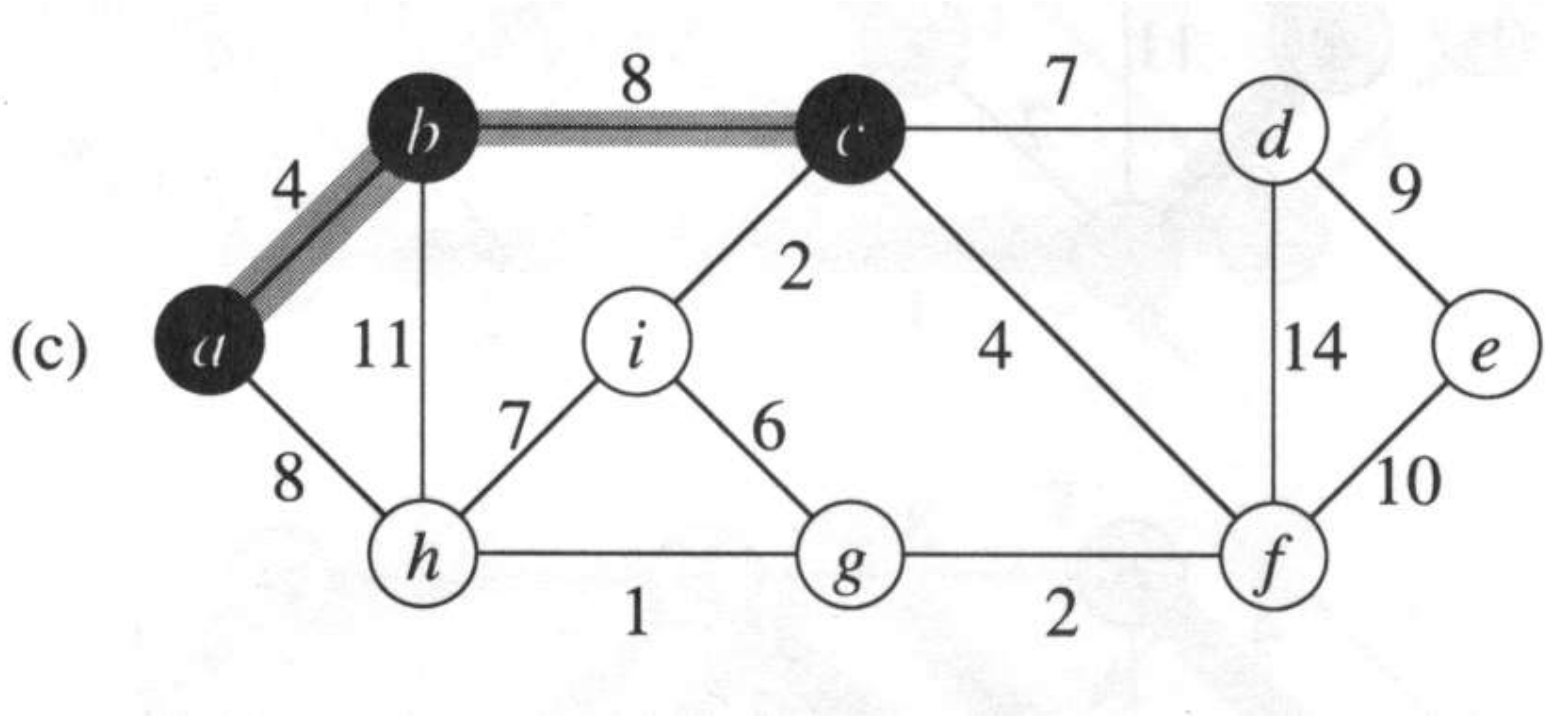
Start anywhere and repeatedly choose the next-smallest edge out from your current tree.



MST

Minimum spanning tree: (Prim's algorithm)

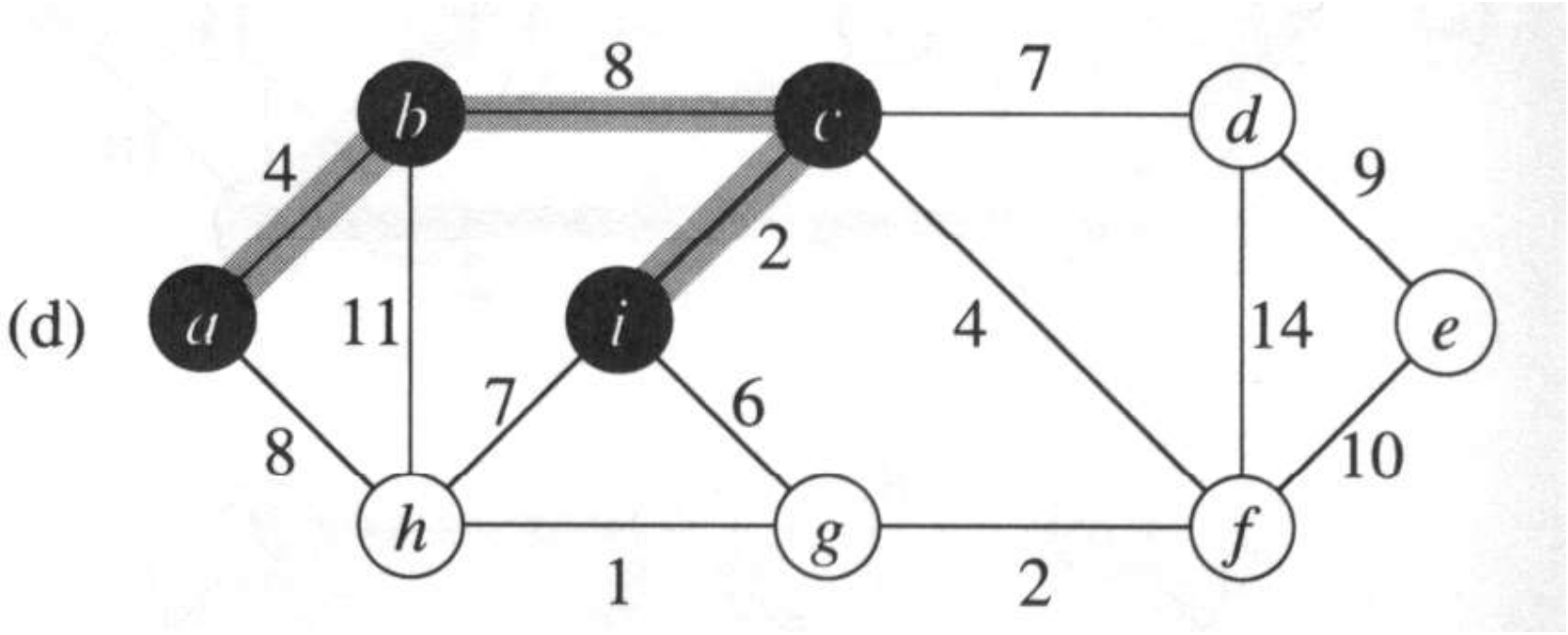
Start anywhere and repeatedly choose the next-smallest edge out from your current tree.



MST

Minimum spanning tree: (Prim's algorithm)

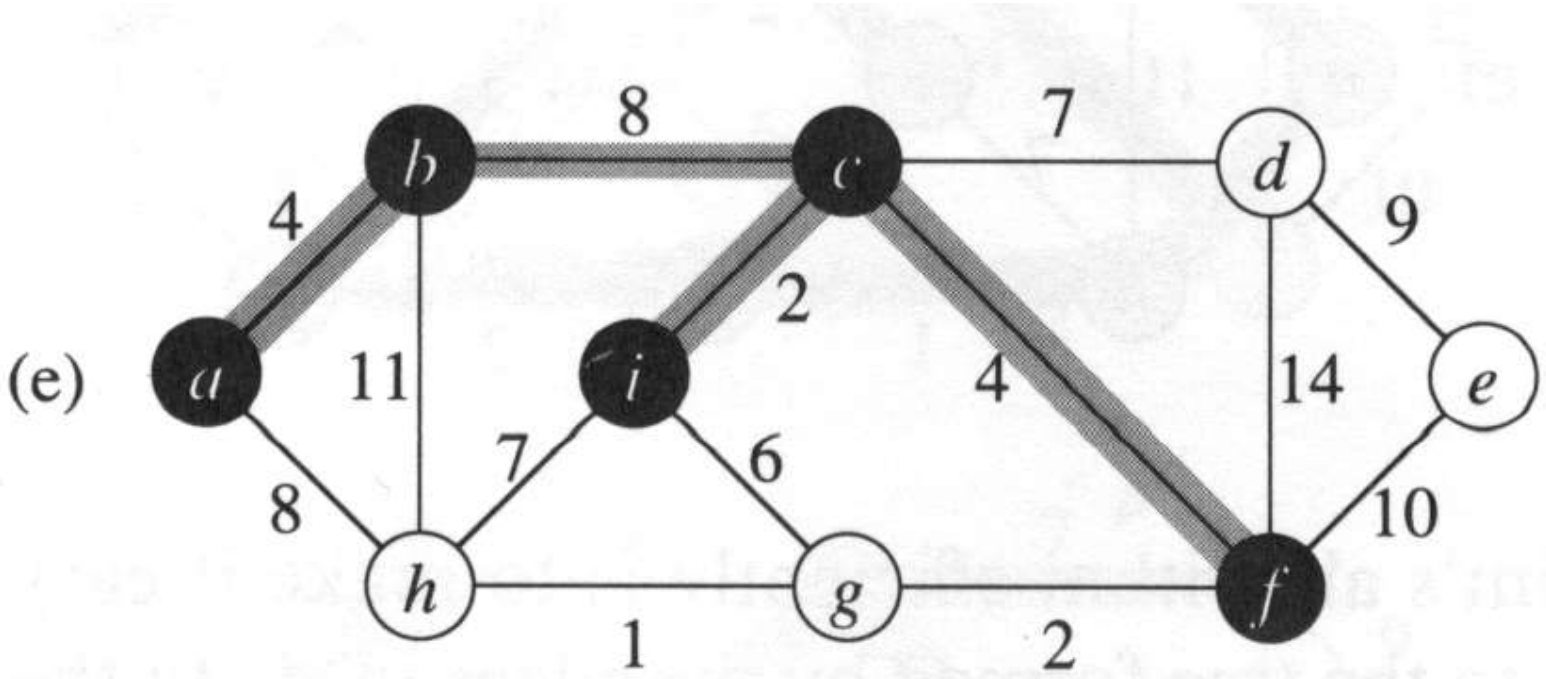
Start anywhere and repeatedly choose the next-smallest edge out from your current tree.



MST

Minimum spanning tree: (Prim's algorithm)

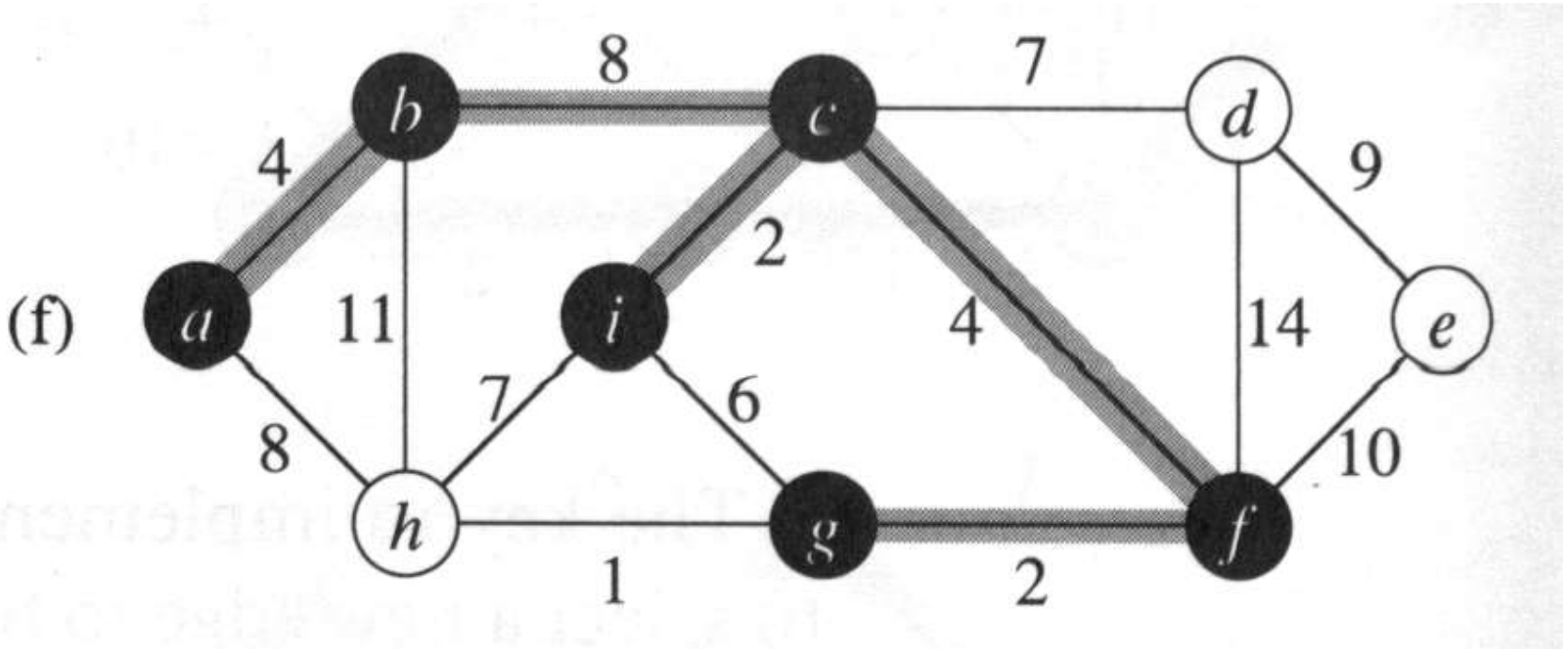
Start anywhere and repeatedly choose the next-smallest edge out from your current tree.



MST

Minimum spanning tree: (Prim's algorithm)

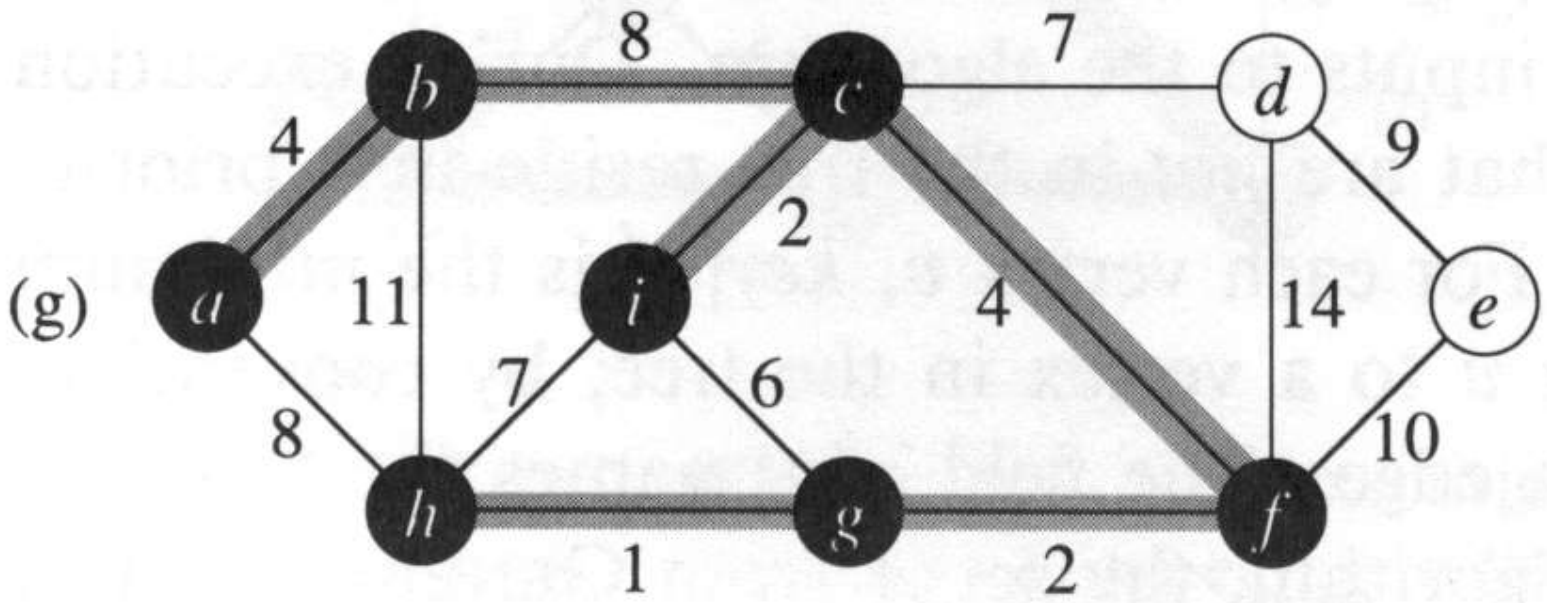
Start anywhere and repeatedly choose the next-smallest edge out from your current tree.



MST

Minimum spanning tree: (Prim's algorithm)

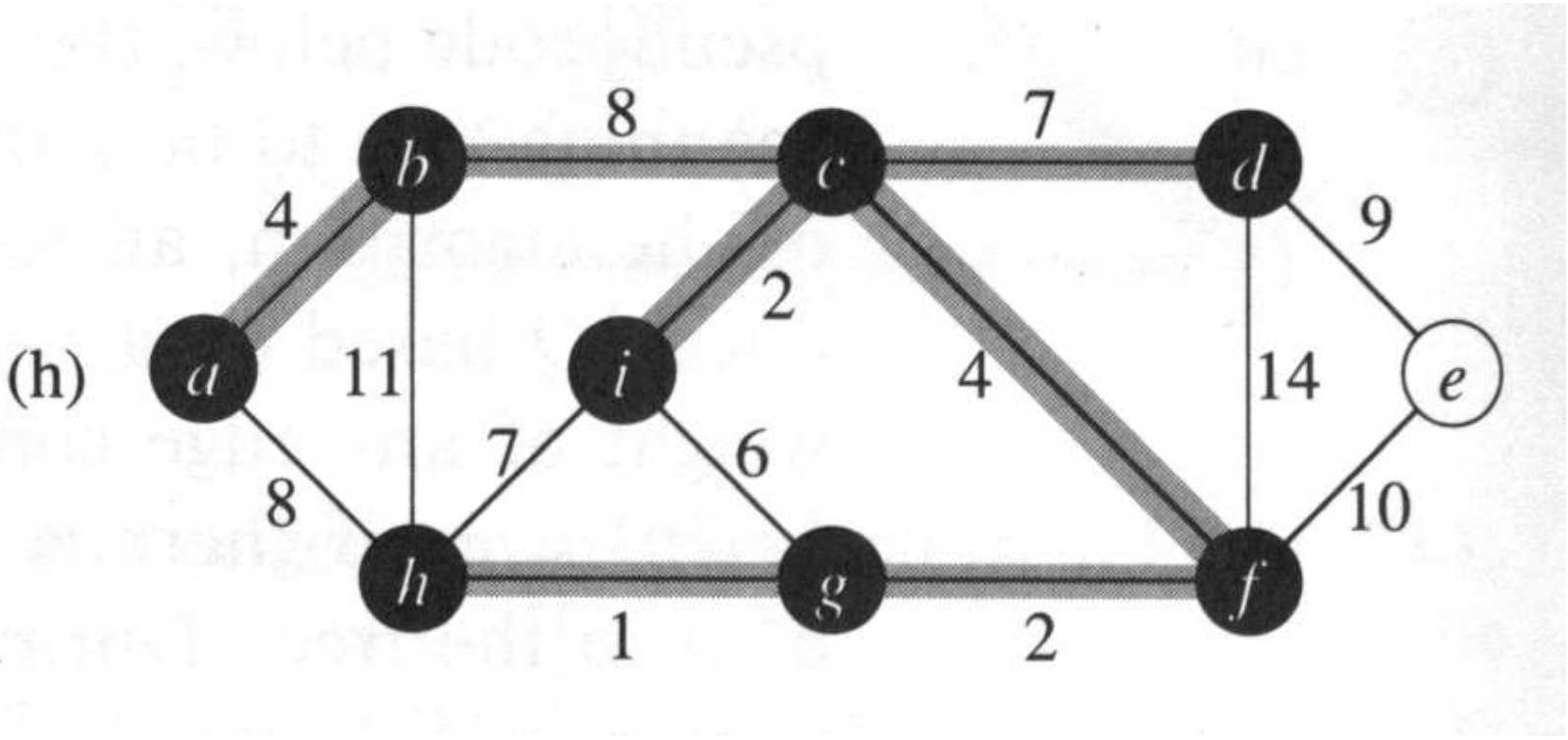
Start anywhere and repeatedly choose the next-smallest edge out from your current tree.



MST

Minimum spanning tree: (Prim's algorithm)

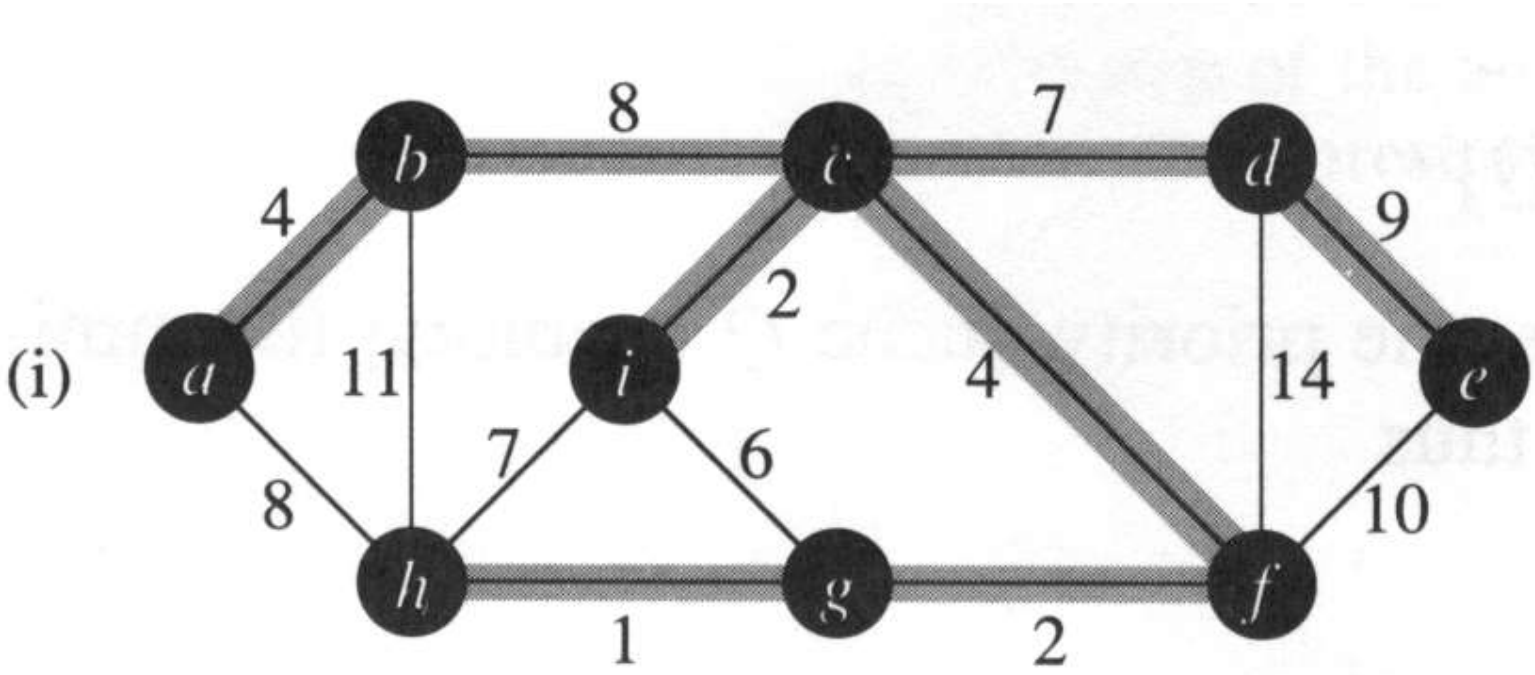
Start anywhere and repeatedly choose the next-smallest edge out from your current tree.



MST

Minimum spanning tree: (Prim's algorithm)

Start anywhere and repeatedly choose the next-smallest edge out from your current tree.



Done!