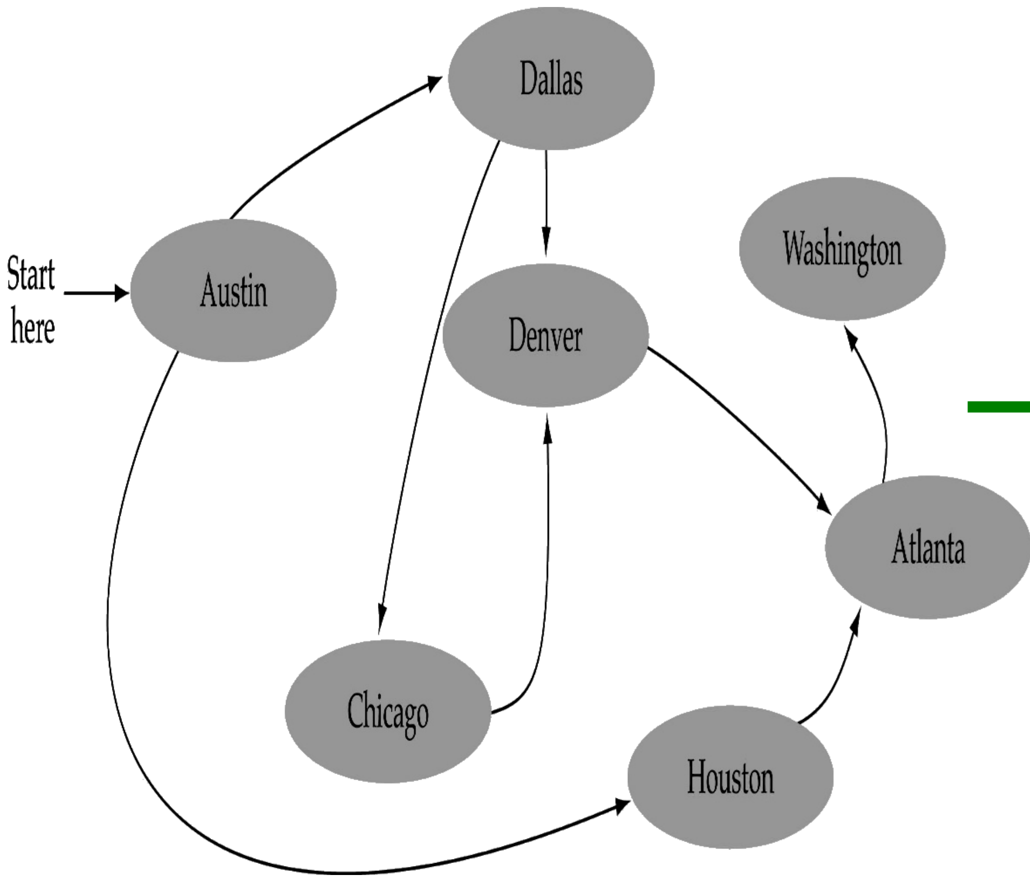
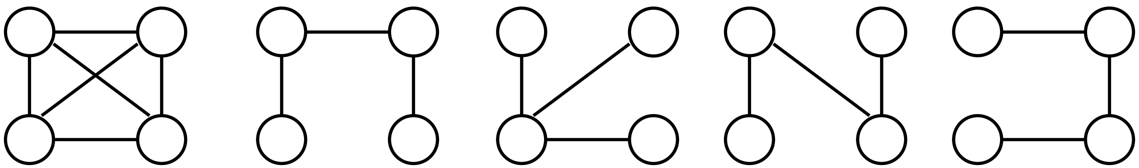


Graphs

Spanning Trees

Spanning trees

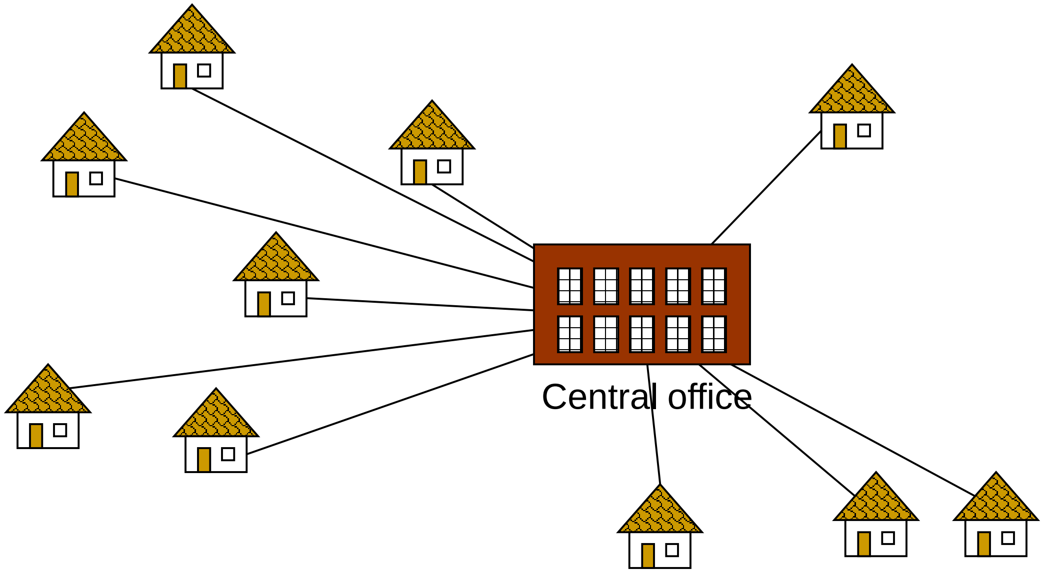
- Suppose you have a connected undirected graph
  - Connected: every node is reachable from every other node
  - Undirected: edges do not have an associated direction
- ...then a spanning tree of the graph is a connected subgraph in which there are no cycles



A connected, undirected graph

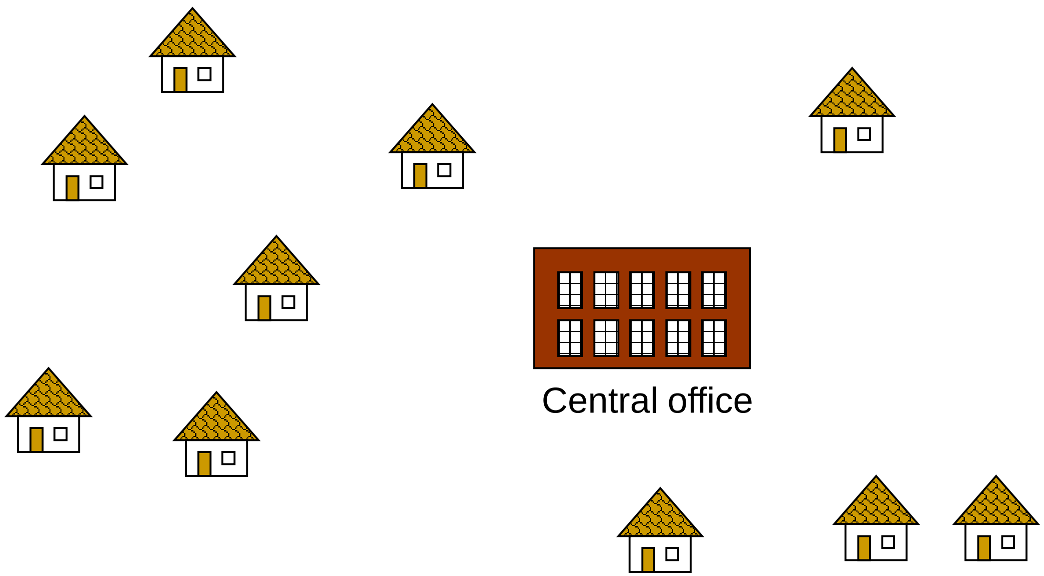
Four of the spanning trees of the graph

Wiring: Naive Approach

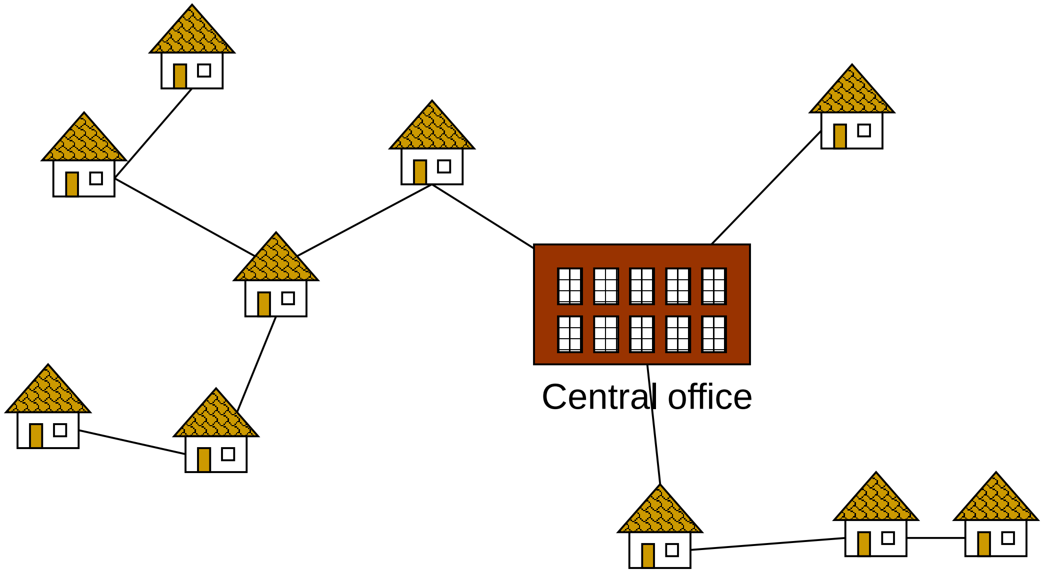


Expensive!

Problem: Laying Telephone Wire



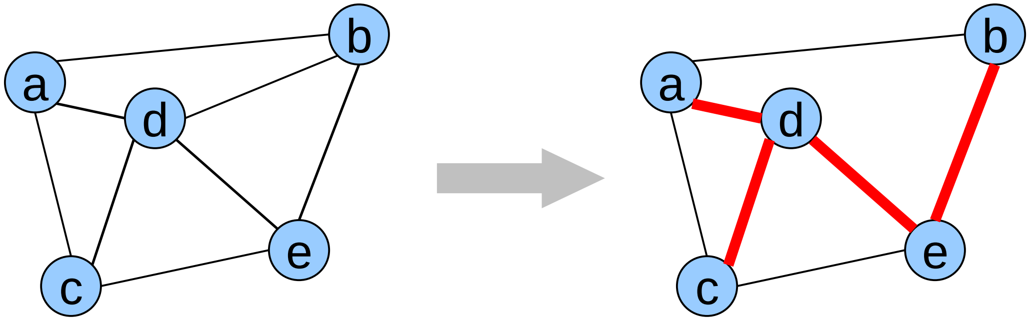
Wiring: Better Approach



Minimize the total length of wire connecting the customers

Spanning tree

- Definition: A Graph has ‘n’ vertices and ‘e’ edges G(n,e) contains (n-1) edges and there should be no loops (or) cycles where ‘n’ is the number of vertices.



Spanning Trees

- When graph G is connected, a depth first or breadth first search starting at any vertex will visit all vertices in G.
- A spanning tree is any tree that consists solely of edges in G and that includes all the vertices
- E(G): T (tree edges) + N (nontree edges)

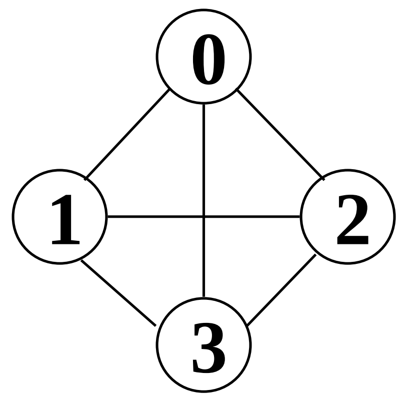
where T: set of edges used during search

N: set of remaining edges

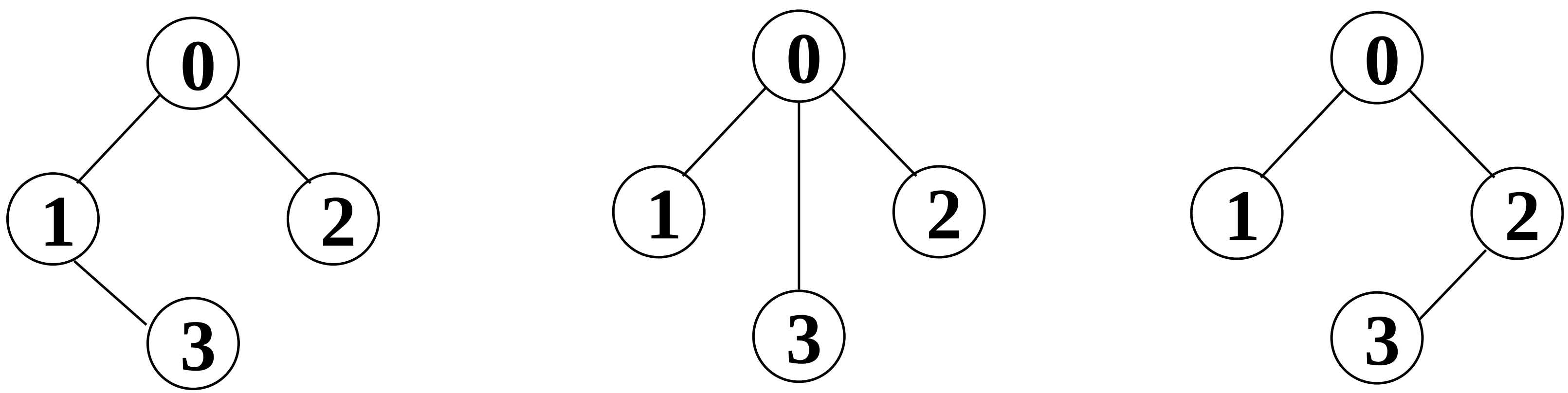
General Properties of Spanning Tree

- A connected graph G can have more than one spanning tree.
- All possible spanning trees of graph G, have the same number of edges and vertices.
- The spanning tree does not have any cycle (loops).
- Removing one edge from the spanning tree will make the graph disconnected, i.e. the spanning tree is minimally connected.
- Adding one edge to the spanning tree will create a circuit or loop, i.e. the spanning tree is maximally acyclic.

Examples of Spanning Trees



G_1



Possible spanning trees

Mathematical Properties of Spanning Tree

- Spanning tree has $n-1$ edges, where n is the number of nodes (vertices).

- From a complete graph, by removing maximum $e-n+1$ edges, we can construct a spanning tree.

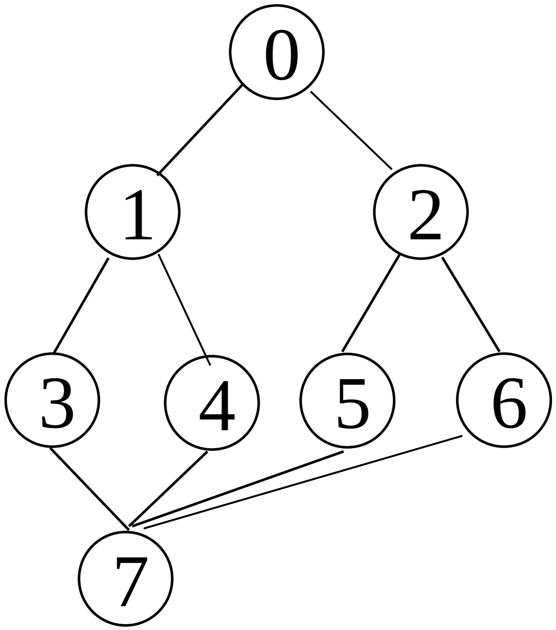
- A complete graph can have maximum n^{n-2} number of spanning trees.

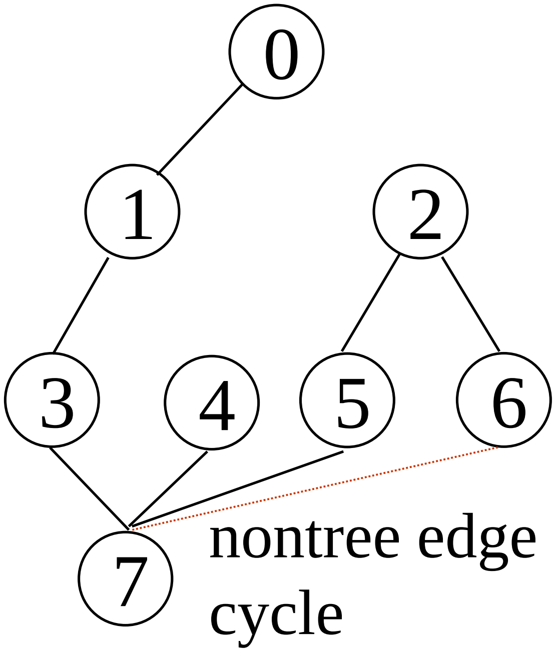
- Thus, we can conclude that spanning trees are a subset of connected Graph G and disconnected graphs do not have spanning tree.

Creation of Spanning Trees

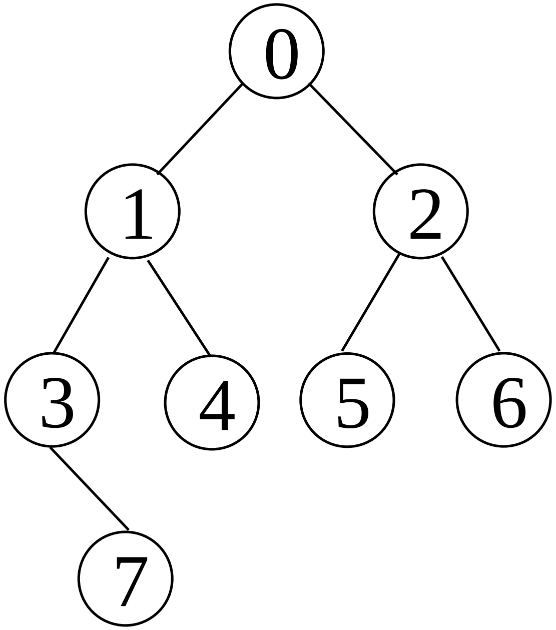
- Either DFS or BFS can be used to create a spanning tree
  - When DFS is used, the resulting spanning tree is known as a depth first spanning tree
  - When BFS is used, the resulting spanning tree is known as a breadth first spanning tree
- While adding a nontree edge into any spanning tree, this will create a cycle

DFS vs. BFS Spanning Tree





DF Spanning Tree



BF Spanning Tree

Applications of Spanning Trees

- Spanning tree is basically used to find a minimum path to connect all nodes in a graph. Common application of spanning trees are –
  - **Civil Network Planning**
  - **Computer Network Routing Protocol**

- **Example:**
  - Consider, city network as a huge graph and now plans to deploy telephone lines in such a way that in minimum lines we can connect to all city nodes. This is where the spanning tree comes into picture.

Minimum (Cost) Spanning Tree

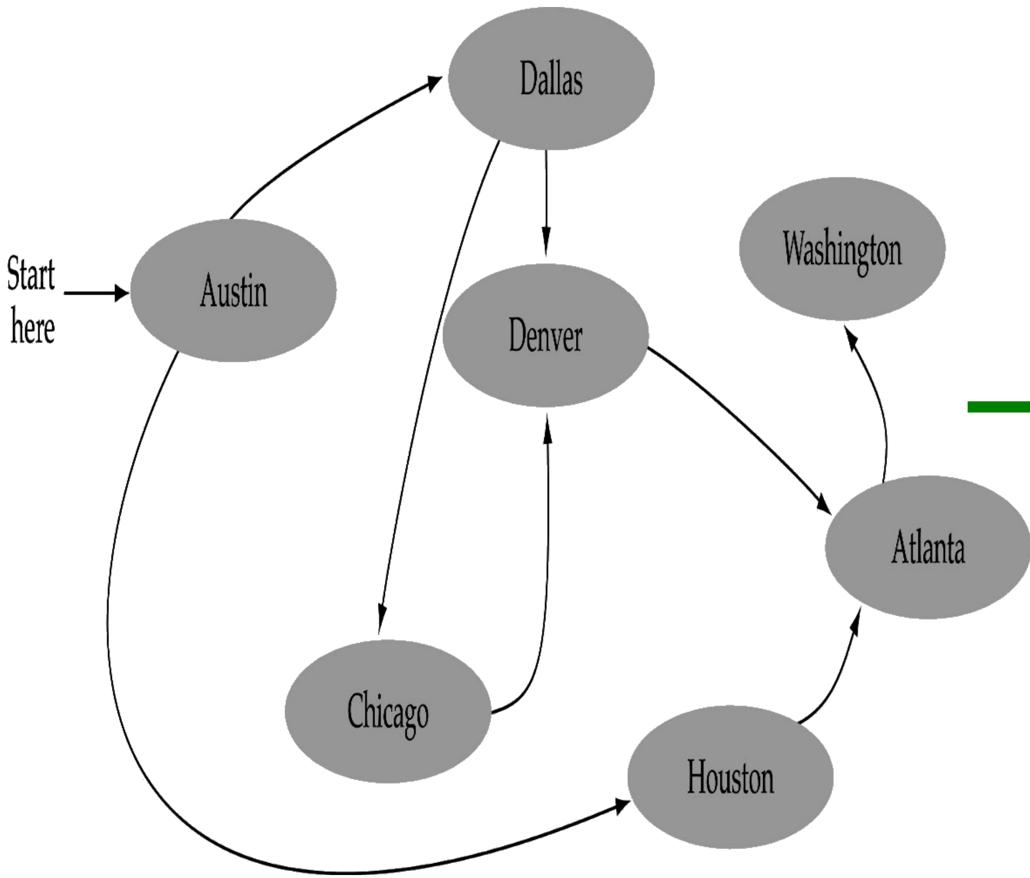
- The cost of the spanning tree is the sum of the weights of all the edges in the tree.
- In a weighted graph, a minimum spanning tree is a spanning tree that has minimum weight than all other spanning trees of the same graph.
- In real-world situations, this weight can be measured as distance, congestion, traffic load or any arbitrary value denoted to the edges.

Minimum (Cost) Spanning Tree

- Given an undirected, weighted graph G, we are interested in finding a tree T that contains all the vertices in G and minimizes the sum

$$w(T) = \sum_{(u,v) \text{ in } T} w(u,v)$$

- Two most important minimum spanning tree (MST) algorithms are:
  - Kruskal's Algorithm
  - Prim's Algorithm



Graphs

Minimum Spanning Trees

Kruskal's Algorithm

Kruskal's MST Algorithm

- This algorithm treats the graph as a forest and every node it has as an individual tree.
- A tree connects to another only and only if, it has the least cost among all available options and does not violate MST properties.
- Kruskal's Algorithm builds the spanning tree by adding edges one by one into a growing spanning tree.
- A minimum spanning tree has $(V - 1)$ edges where V is the number of vertices in the given graph.

Kruskal's MST Algorithm (cont.)

```
Kruskal's algorithm:
sort the edges of G in increasing order by length
keep a subgraph S of G, initially empty
for each edge e in sorted order
    if the endpoints of e are disconnected in S
    add e to S
return S
```

- Note that, whenever you add an edge (u,v), it is always the smallest connecting part of S reachable from u with the rest of G, so it must be part of the MST.

Kruskal's MST Algorithm (cont.)

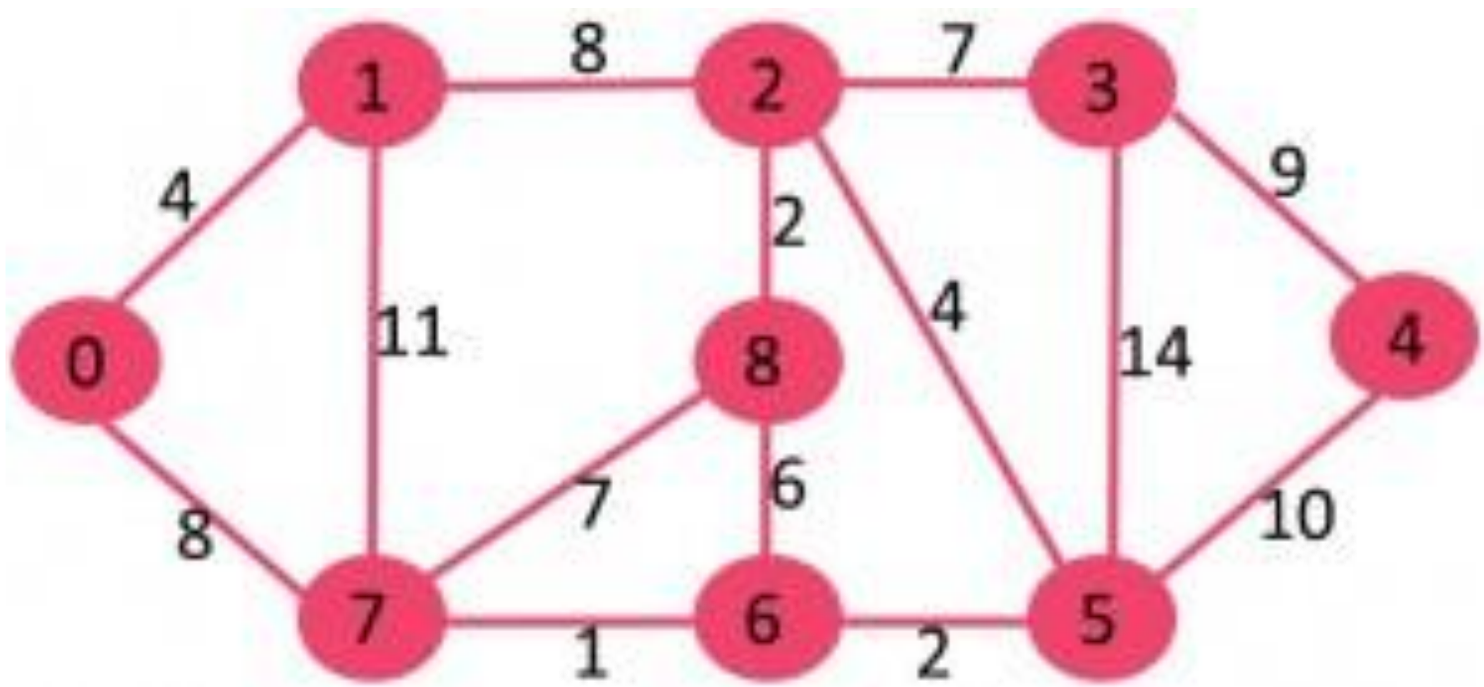
- <u>Steps of Kruskal's MST Algorithm</u>

Input: Weighted Graph with V vertices

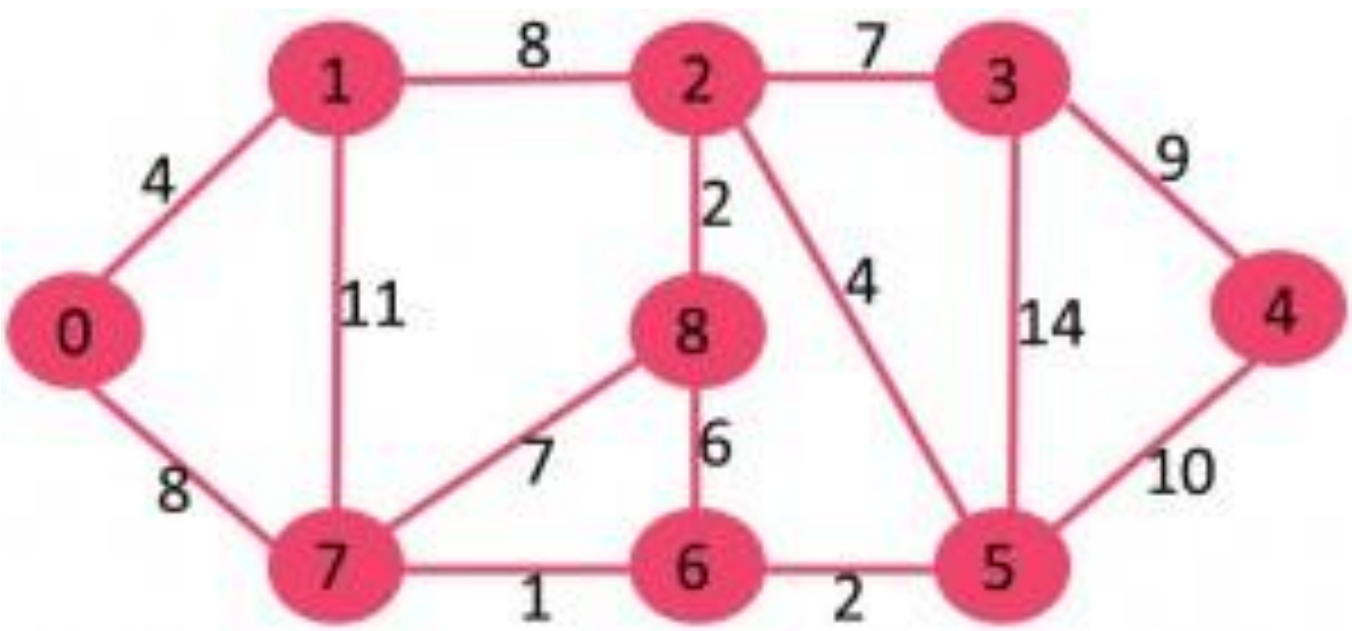
1. Sort all the edges in non-decreasing order of their weight.
2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far.
   - If cycle is not formed, include this edge.
   - Else, discard it.
3. Repeat step#2 until there are (V-1) edges in the spanning tree.

Kruskal’s MST Example

- Consider the below input graph which contains 9 vertices and 14 edges.
- So, the minimum spanning tree formed will be having (9 – 1) = 8 edges.



Kruskal's MST Example (cont.)



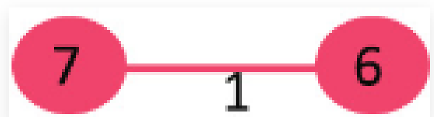
After sorting:

Wght	Src	Dest
1	7	6
2	8	2
2	6	5
4	0	1
4	2	5
6	8	6
7	2	3
7	7	8
8	0	7
8	1	2
9	3	4
10	5	4
11	1	7
14	3	5

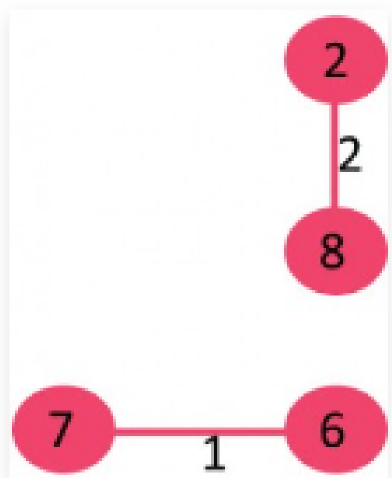
Kruskal’s MST Example (cont.)

Now pick all edges one by one from sorted list of edges

1. *Pick edge 7-6:* No cycle is formed, include it.

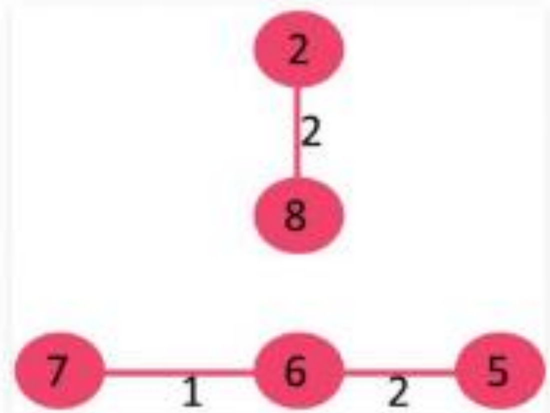


2. *Pick edge 8-2:* No cycle is formed, include it.

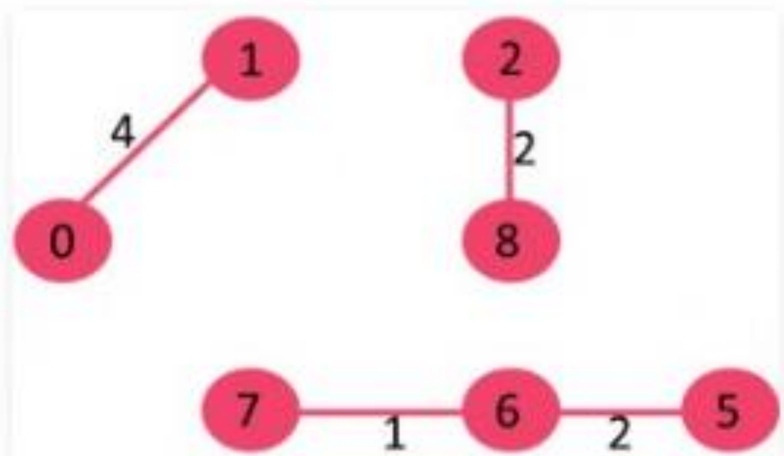


Kruskal’s MST Example (cont.)

3. *Pick edge 6-5:* No cycle is formed, include it.

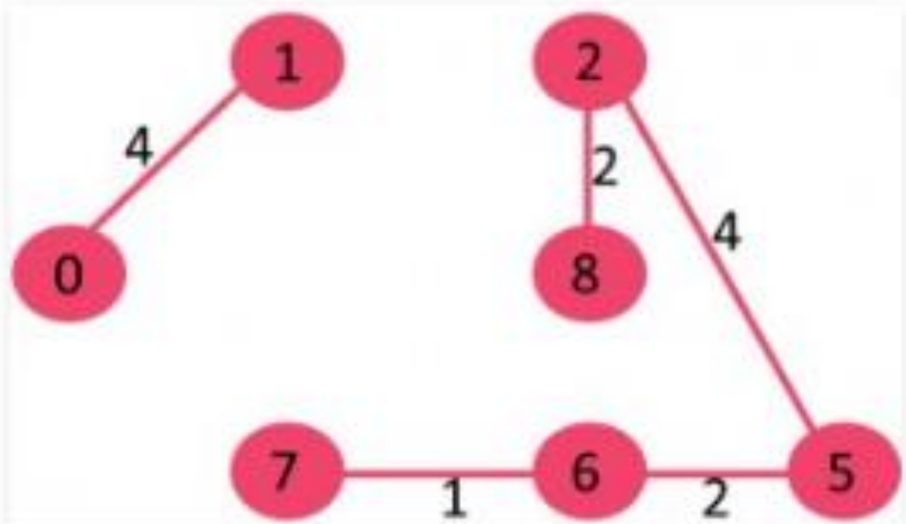


4. *Pick edge 0-1:* No cycle is formed, include it.



Kruskal's MST Example (cont.)

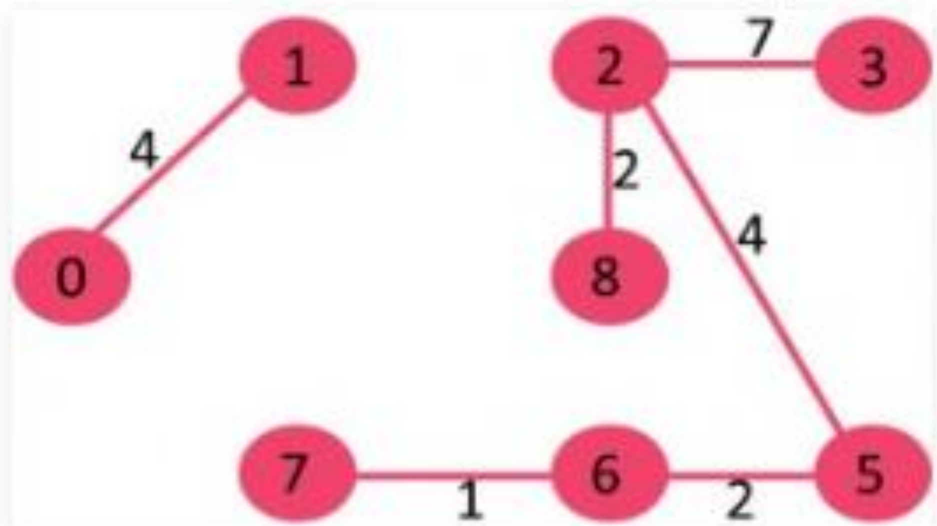
5. *Pick edge 2-5:* No cycle is formed, include it.



6. *Pick edge 8-6:* Since including this edge results in cycle, discard it.

Kruskal’s MST Example (cont.)

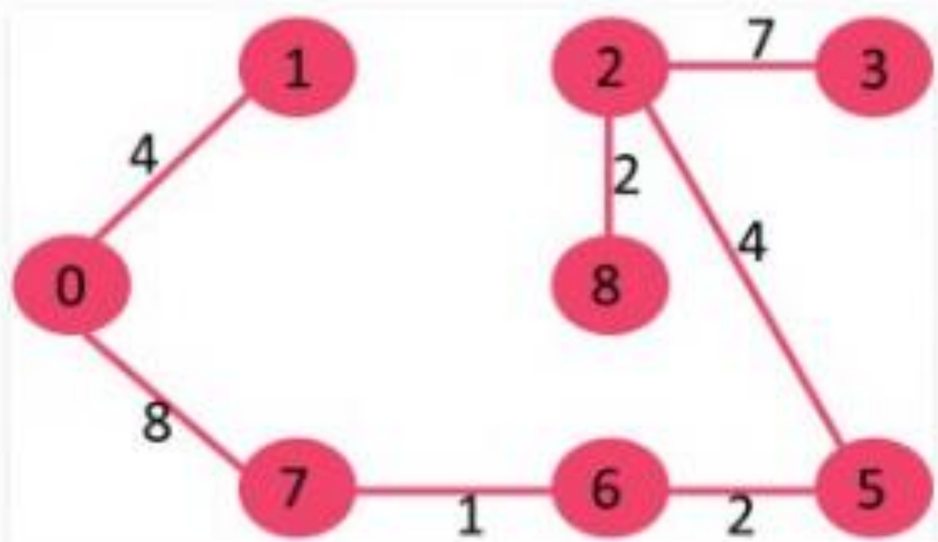
7. *Pick edge 2-3:* No cycle is formed, include it.



8. *Pick edge 7-8:* Since including this edge results in cycle, discard it.

Kruskal's MST Example (cont.)

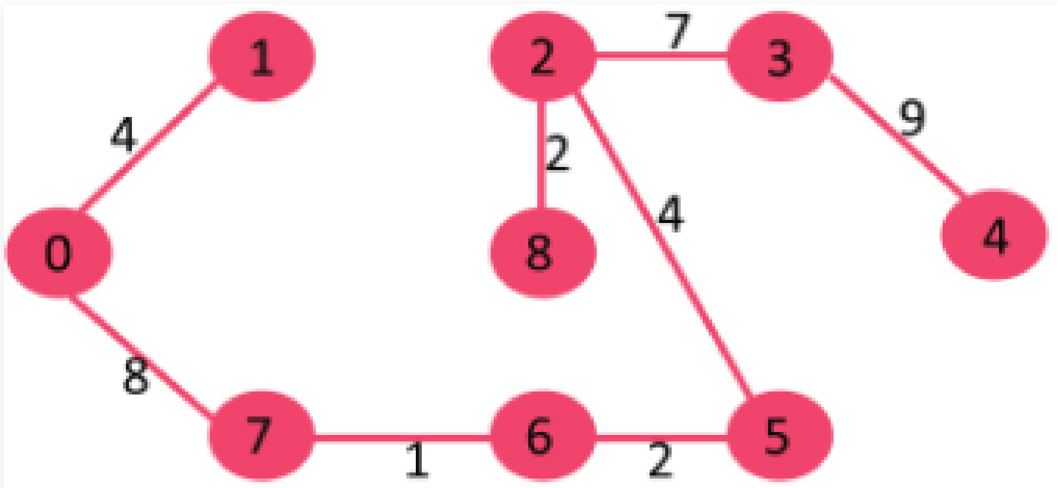
9. *Pick edge 0-7:* No cycle is formed, include it.



10. *Pick edge 1-2:* Since including this edge results in cycle, discard it.

Kruskal's MST Example (cont.)

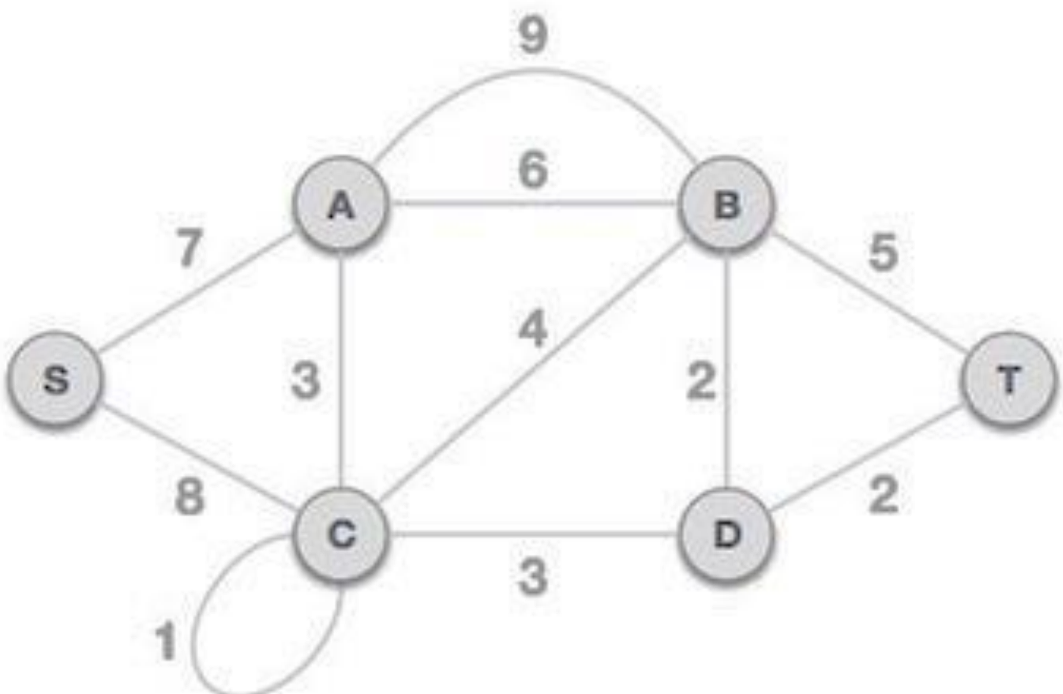
11. *Pick edge 3-4:* No cycle is formed, include it.



Since the number of edges included equals (V – 1), the algorithm stops here.

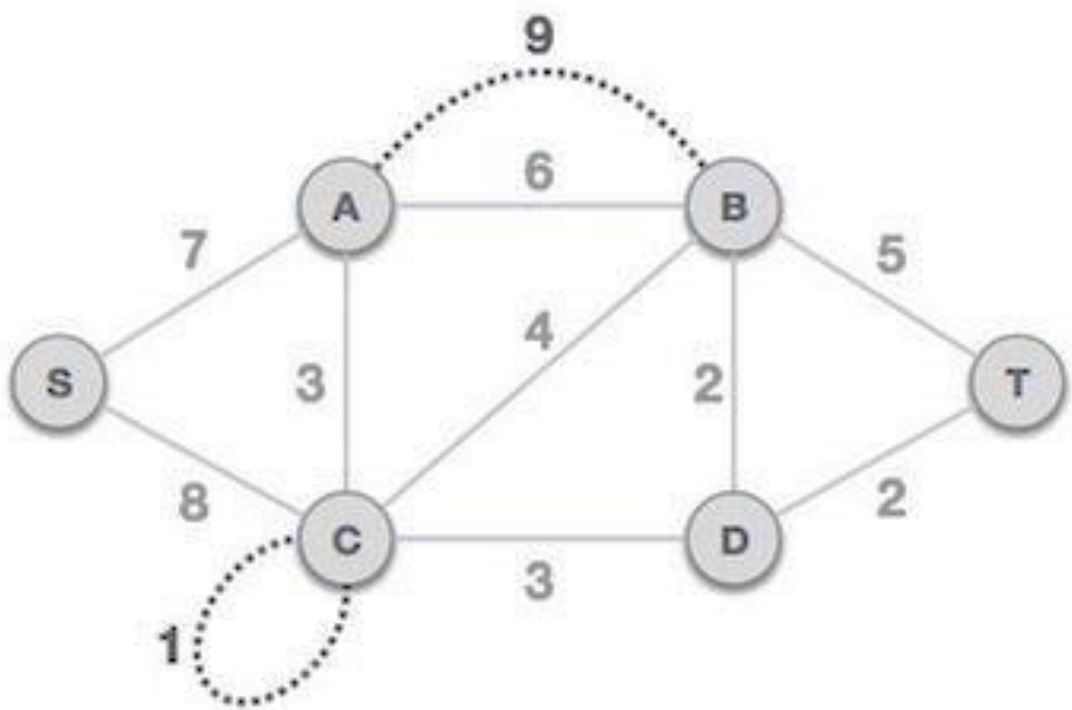
Kruskal's MST Example 2

- Consider the below input graph which contains 6 vertices and 11 edges.
- So, the minimum spanning tree formed will be having (6 – 1) = 5 edges.



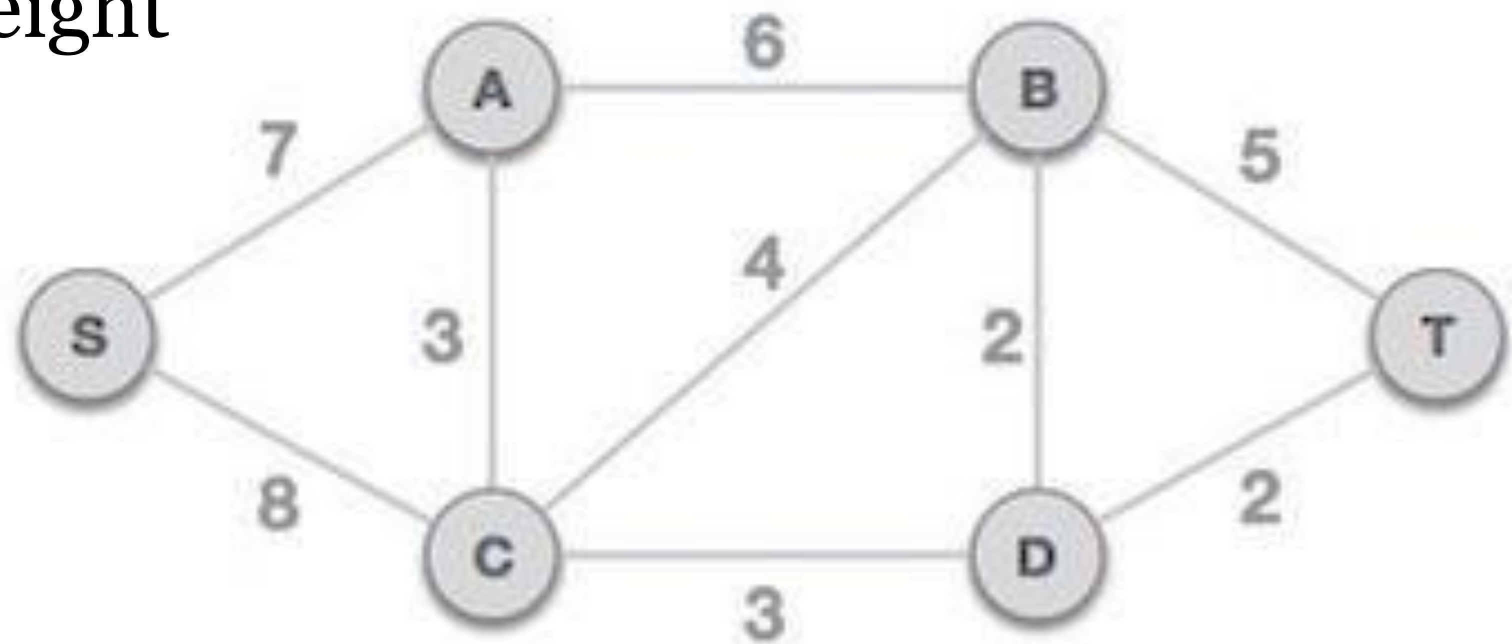
Kruskal's MST Example 2

- Remove all loops and parallel edges from the given graph.
- In case of parallel edges, keep the one which has the least cost associated and remove all others.



Kruskal's MST Example 2

- Arrange all edges in their increasing order of weight

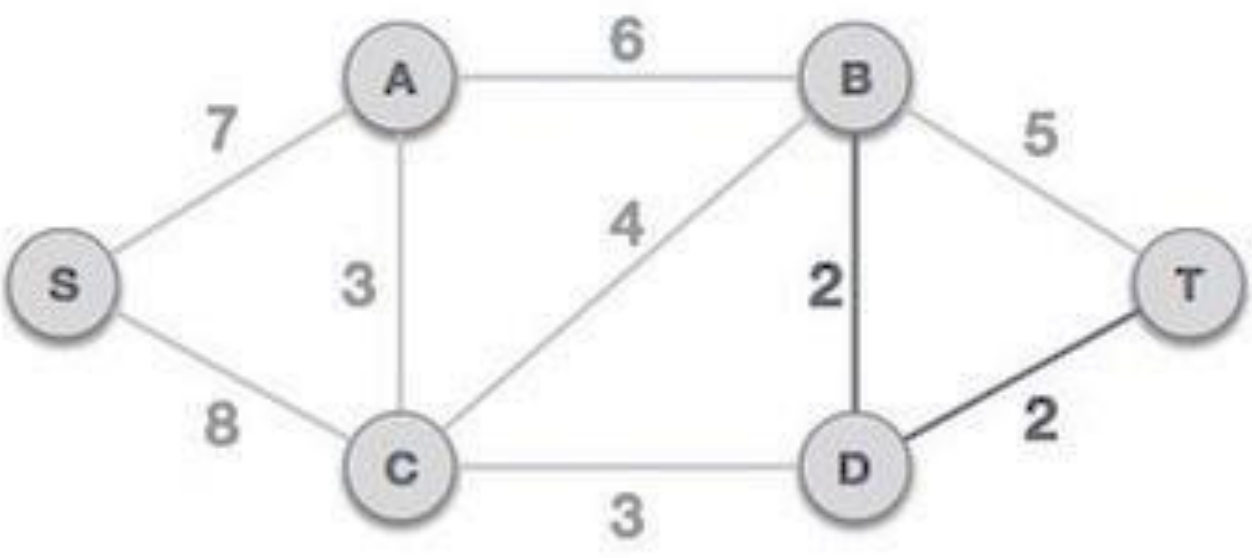


B, D	D, T	A, C	C, D	C, B	B, T	A, B	S, A	S, C
2	2	3	3	4	5	6	7	8

Kruskal's MST Example 2

- Now we start adding edges to the graph beginning from the one which has the least weight.

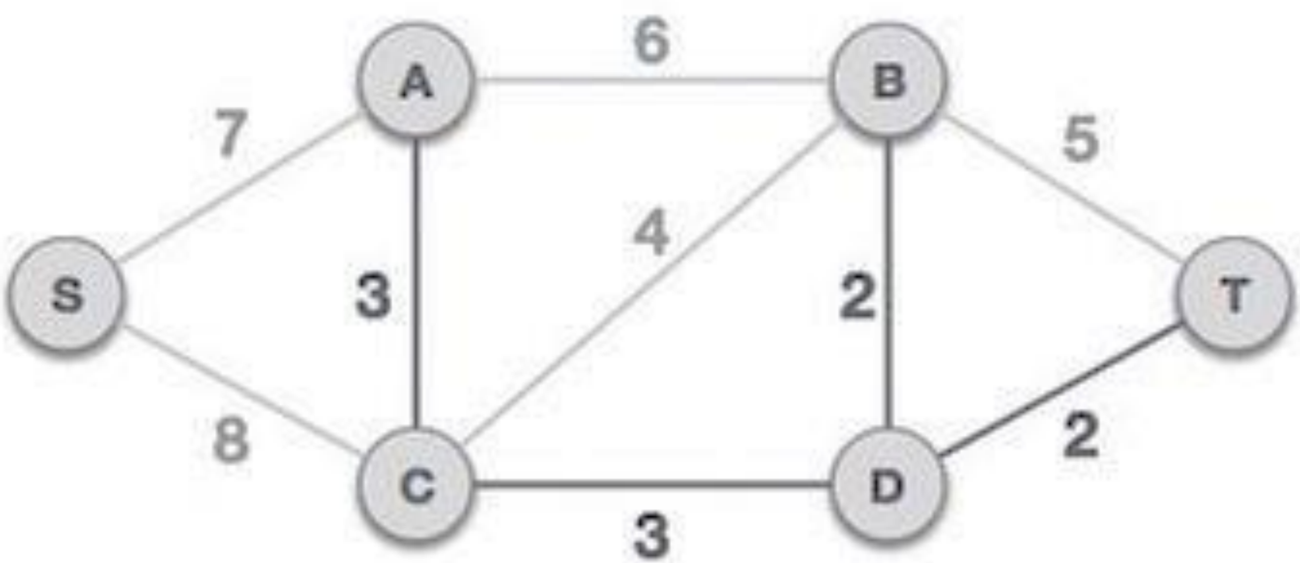
B, D	D, T	A, C	C, D	C, B	B, T	A, B	S, A	S, C
2	2	3	3	4	5	6	7	8



Kruskal's MST Example 2

- Now we start adding edges to the graph beginning from the one which has the least weight.

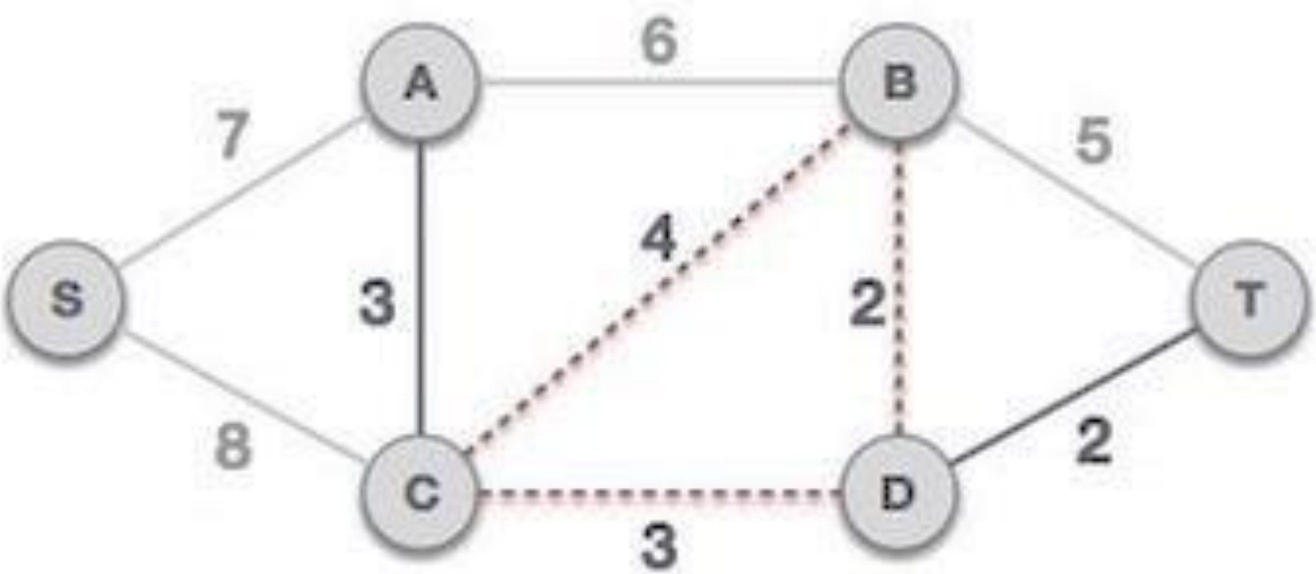
B, D	D, T	A, C	C, D	C, B	B, T	A, B	S, A	S, C
2	2	3	3	4	5	6	7	8



Kruskal's MST Example 2

- Now we start adding edges to the graph beginning from the one which has the least weight.

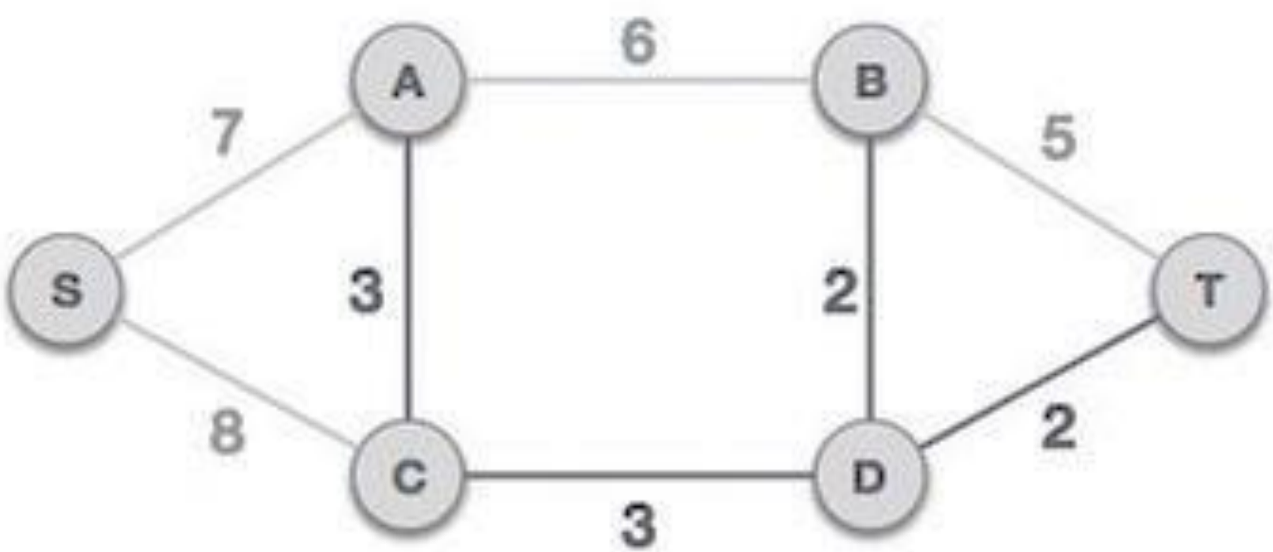
B, D	D, T	A, C	C, D	C, B	B, T	A, B	S, A	S, C
2	2	3	3	4	5	6	7	8



Kruskal's MST Example 2

- Now we start adding edges to the graph beginning from the one which has the least weight.

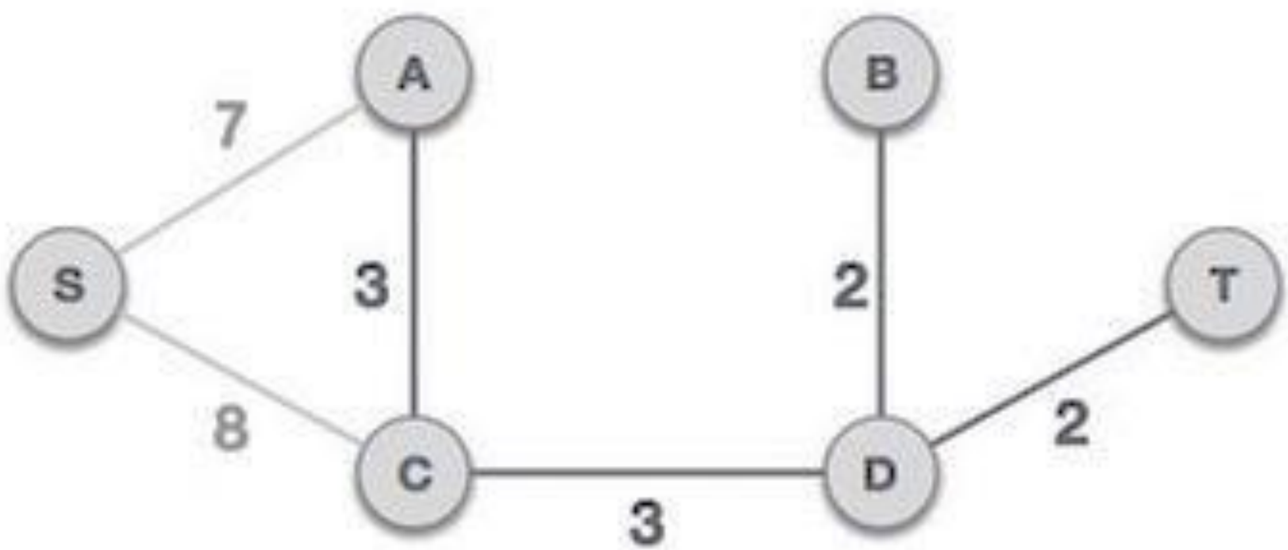
B, D	D, T	A, C	C, D	C, B	B, T	A, B	S, A	S, C
2	2	3	3	4	5	6	7	8



Kruskal's MST Example 2

- Now we start adding edges to the graph beginning from the one which has the least weight.

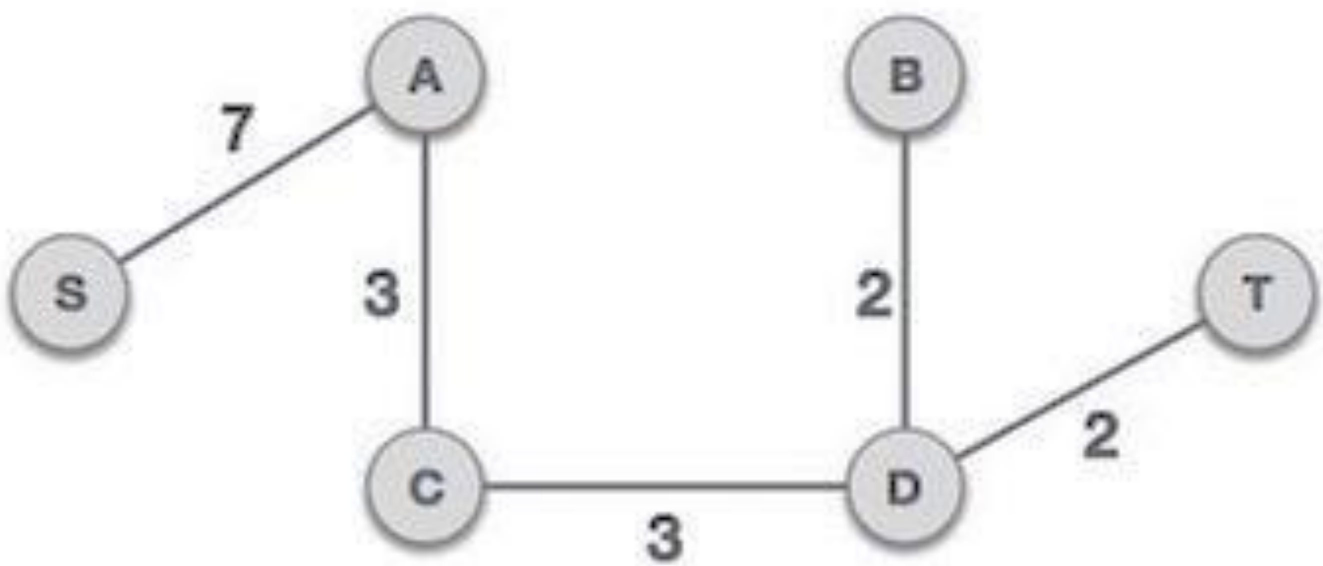
B, D	D, T	A, C	C, D	C, B	B, T	A, B	S, A	S, C
2	2	3	3	4	5	6	7	8

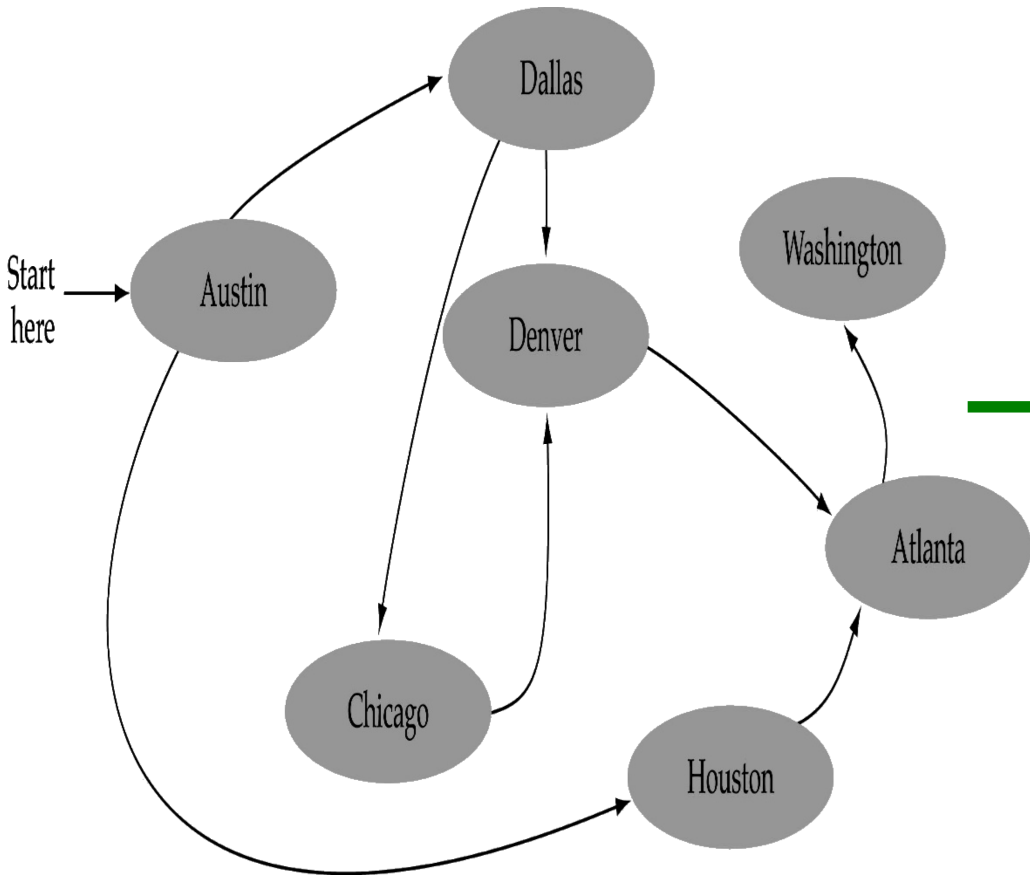


Kruskal's MST Example 2

- Now we start adding edges to the graph beginning from the one which has the least weight.

B, D	D, T	A, C	C, D	C, B	B, T	A, B	S, A	S, C
2	2	3	3	4	5	6	7	8





Graphs

Minimum Spanning Trees

Prim's Algorithm

Prim's MST Algorithm

- It starts with an empty spanning tree.
- The idea is to maintain two sets of vertices.
- The first set contains the vertices already included in the MST, the other set contains the vertices not yet included.
- At every step, it considers all the edges that connect the two sets, and picks the minimum weight edge from these edges.
- After picking the edge, it moves the other endpoint of the edge to the set containing MST.

Prim’s MST Algorithm (cont.)

- Prim's algorithm shares a similarity with the shortest path first algorithms.
- A group of edges that connects two set of vertices in a graph is called cut in graph theory.
- So, at every step of Prim’s algorithm, we find a cut (of two sets, one contains the vertices already included in MST and other contains rest of the vertices), pick the minimum weight edge from the cut and include this vertex to MST Set (the set that contains already included vertices).

Prim's MST Algorithm (cont.)

```
Prim's algorithm:
let T be a single vertex x
while (T has fewer than n vertices)
{
    find the smallest edge connecting T to G-T
    add it to T
}
```

- It looks like the loop has a slow step in it.
- But again, some data structures can be used to speed this up.
- The idea is to use a **heap** to remember, for each vertex, the smallest edge connecting T with that vertex.

Prim's MST Algorithm (cont.)

```
Prim with heaps:
make a heap of values (vertex,edge,weight(edge))
    initially (v,-,infinity) for each vertex
    let tree T be empty
while (T has fewer than n vertices)
{
    let (v,e,weight(e)) have the smallest weight in the heap
    remove (v,e,weight(e)) from the heap
    add v and e to T
    for each edge f=(u,v)
    if u is not already in T
        find value (u,g,weight(g)) in heap
        if weight(f) < weight(g)
        replace (u,g,weight(g)) with (u,f,weight(f))
}
```

Prim's MST Algorithm (cont.)

1) Create a set mstS that keeps track of vertices already included in MST.

2) Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the **first vertex** so that **it is picked first**.

3) While mstS doesn't include all vertices

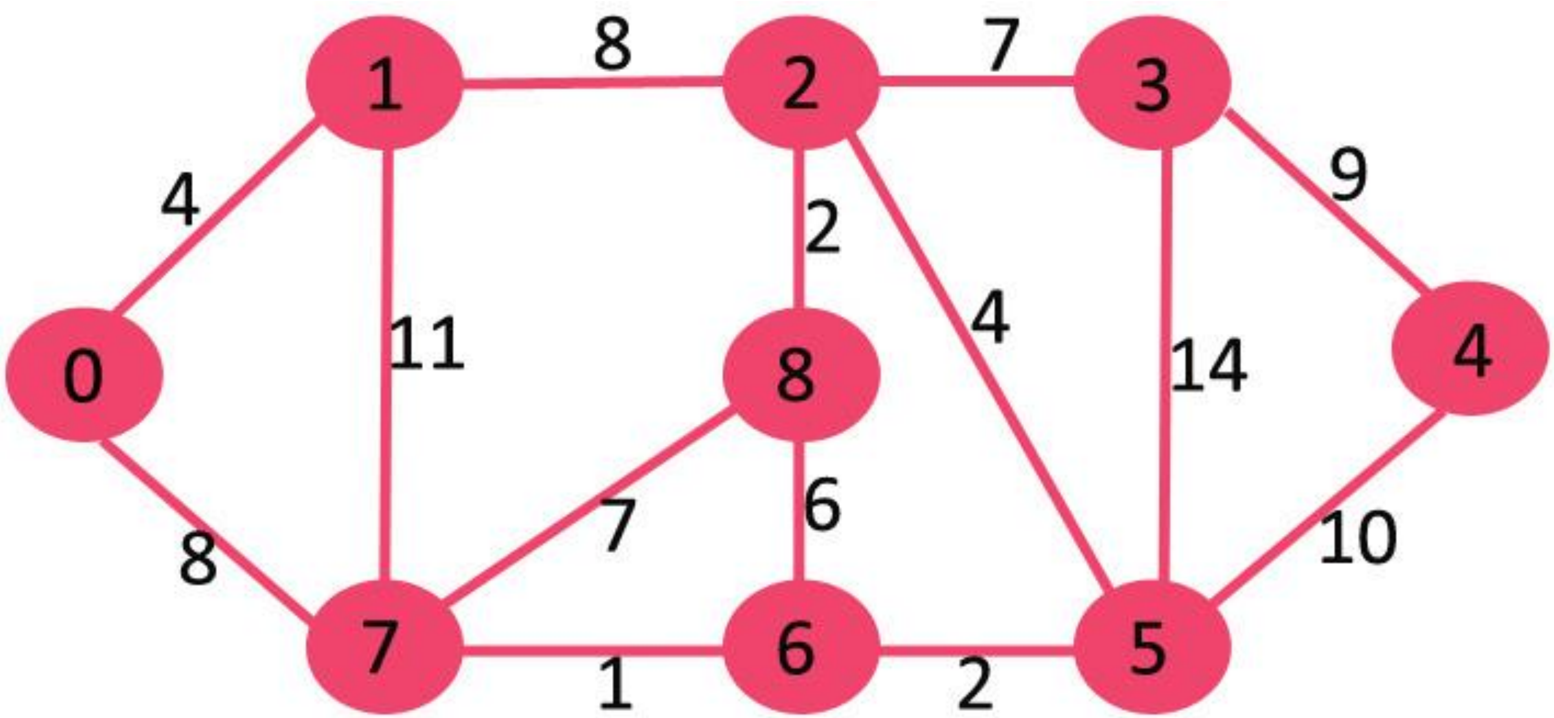
a) Pick a vertex u which is not there in mstS and has minimum key value.

b) Include u to mstS.

c) Update key value of all adjacent vertices of u. To update the key values, iterate through all adjacent vertices. For every adjacent vertex v, if weight of edge (u,v) is less than the previous key value of v, update the key value as weight of (u,v).

Prim's MST Example

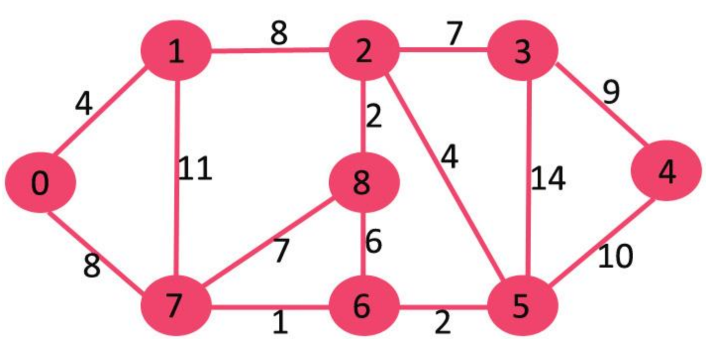
- Consider the below input graph which contains 9 vertices and 14 edges.
- So, the minimum spanning tree formed will be having (9 – 1) = 8 edges.

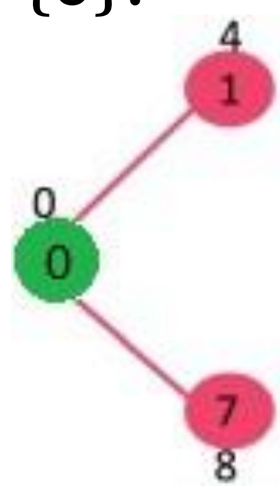


Prim’s MST Example (cont.)

- Let vertex 0 be the starting point.
- The set mstS is initially empty and keys assigned to vertices are {0, INF, INF, INF, INF, INF, INF, INF, INF}.
- The vertex 0 is picked, include it in mstS. So mstS becomes {0}.

V	Key
0	0
1	∞
2	∞
3	∞
4	∞
5	∞
6	∞
7	∞
8	∞



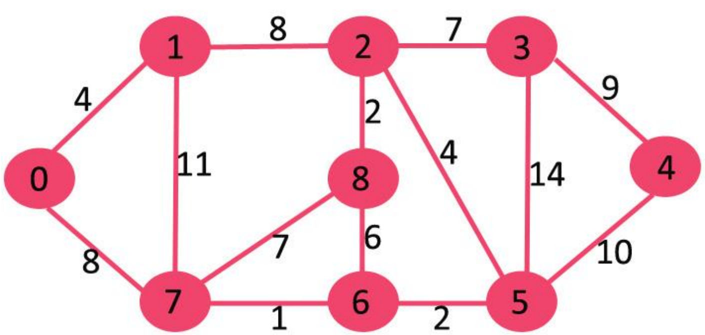


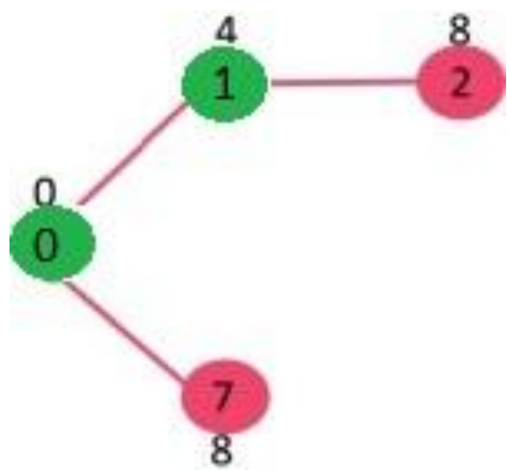
mstS={o}

Prim's MST Example (cont.)

- Pick the vertex with minimum key value and not already included in mstS.
- The vertex 1 is picked and added to mstS. So mstS now becomes {0, 1}.
- Update the key values of adjacent vertices of 1. The key value of vertex 2 becomes 8.

V	Key
0	0
1	4
2	8
3	∞
4	∞
5	∞
6	∞
7	8
8	∞



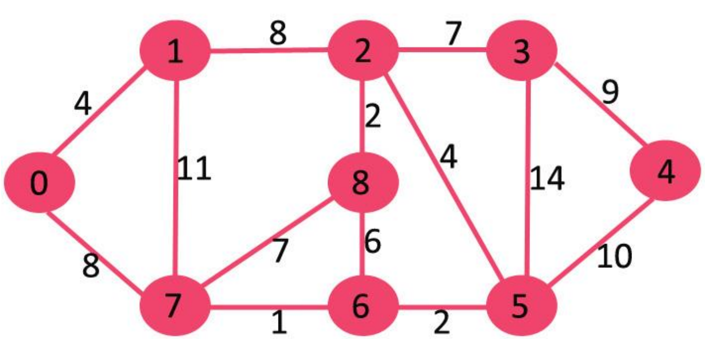


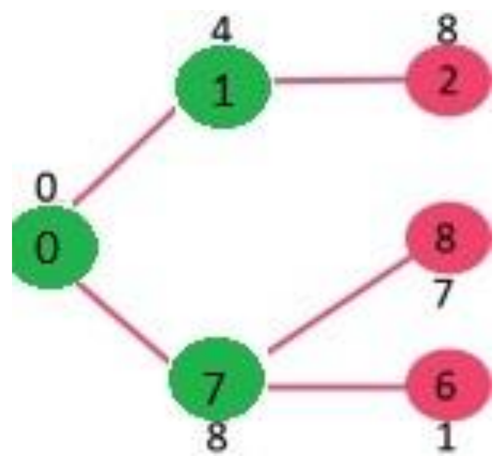
mstS={0,1}

Prim's MST Example (cont.)

- Pick the vertex with minimum key value and not already included in mstS.
- We can either pick vertex 7 or vertex 2, let vertex 7 is picked. So mstS now becomes {0,1,7}.
- Update the key values of adjacent vertices of 7. The key value of vertex 6 and 8 becomes finite (1 and 7 respectively).

V	Key
0	0
1	4
2	8
3	∞
4	∞
5	∞
6	1
7	8
8	7



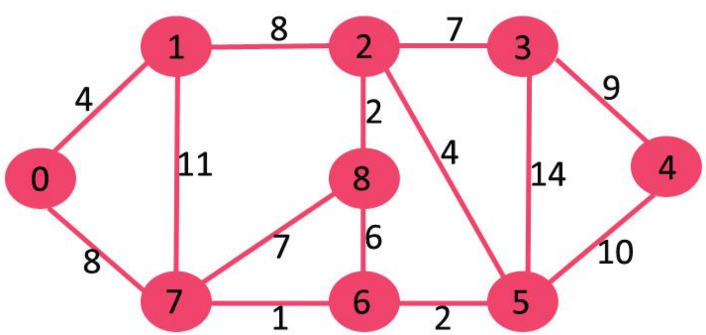


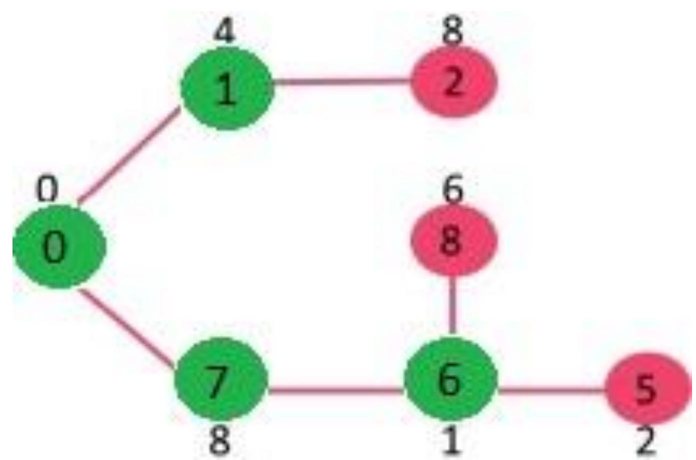
mstS={0,1,7}

Prim's MST Example (cont.)

- Pick the vertex with minimum key value and not already included in mstS.
- Vertex 6 is picked. So mstS now becomes {0,1,7,6}.
- Update the key values of adjacent vertices of 6. The key value of vertex 5 and 8 are updated.

V	Key
0	0
1	4
2	8
3	∞
4	∞
5	2
6	1
7	8
8	6

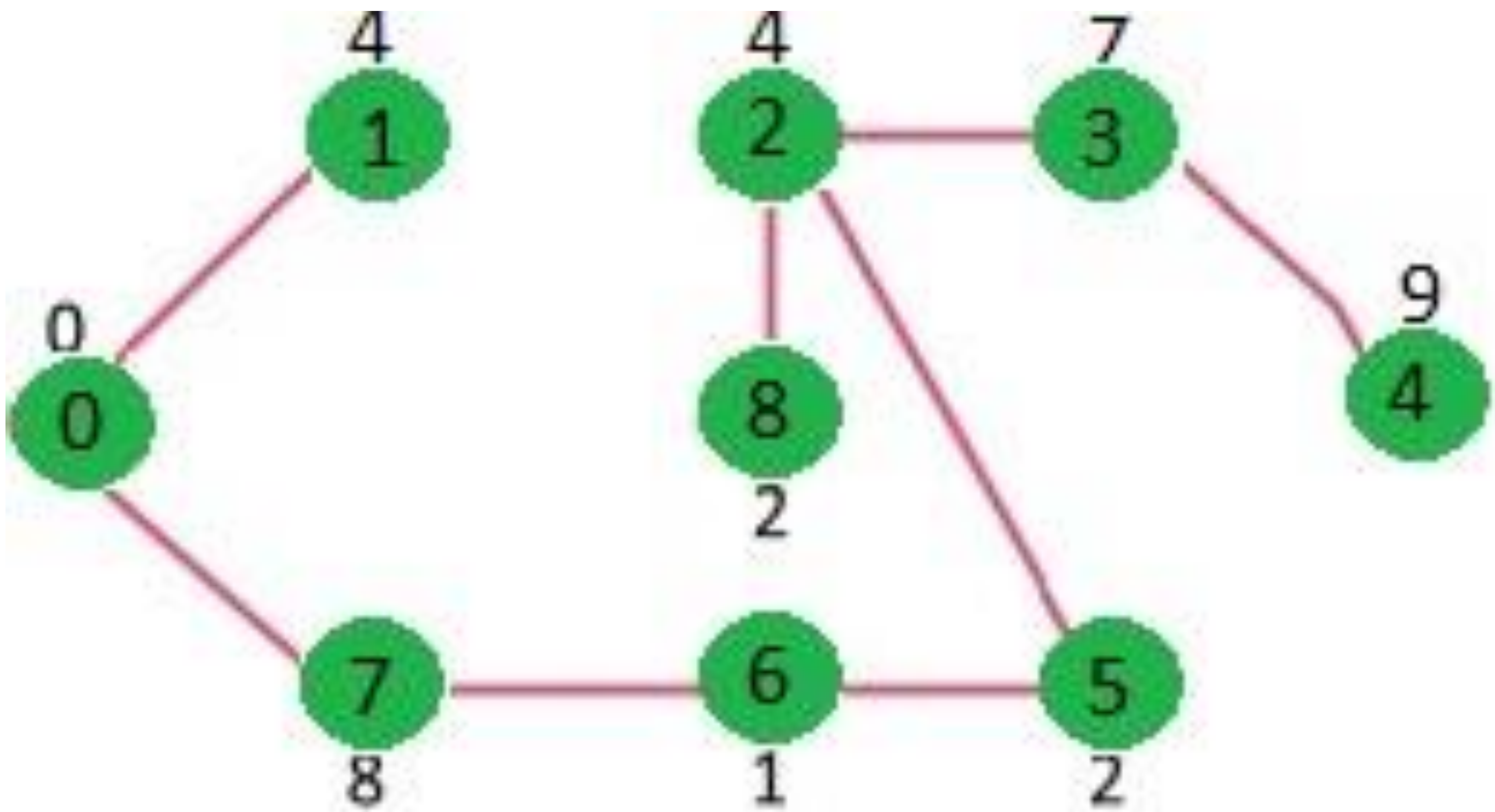




mstS={0,1,7,6}

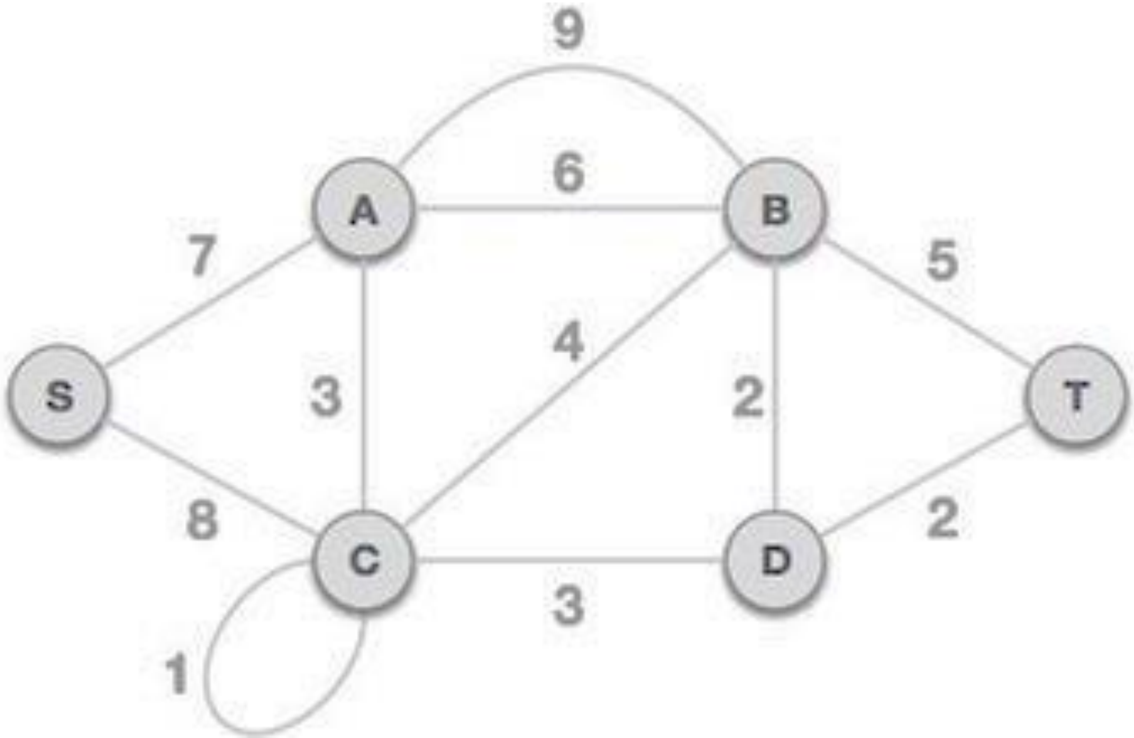
Prim's MST Example (cont.)

- We repeat the above steps until mstS includes all vertices of given graph.
- Finally, we get the following graph (MST).



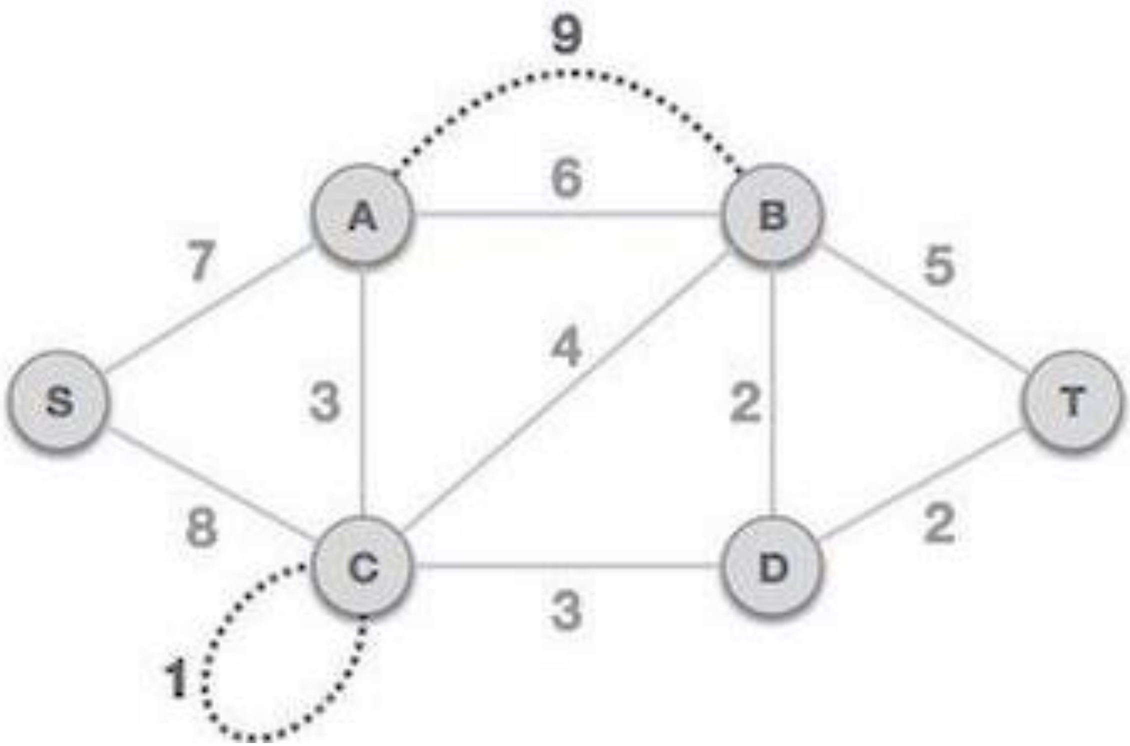
Prim's MST Example 2

- Consider the below input graph which contains 6 vertices and 11 edges.
- So, the minimum spanning tree formed will be having (6 – 1) = 5 edges.



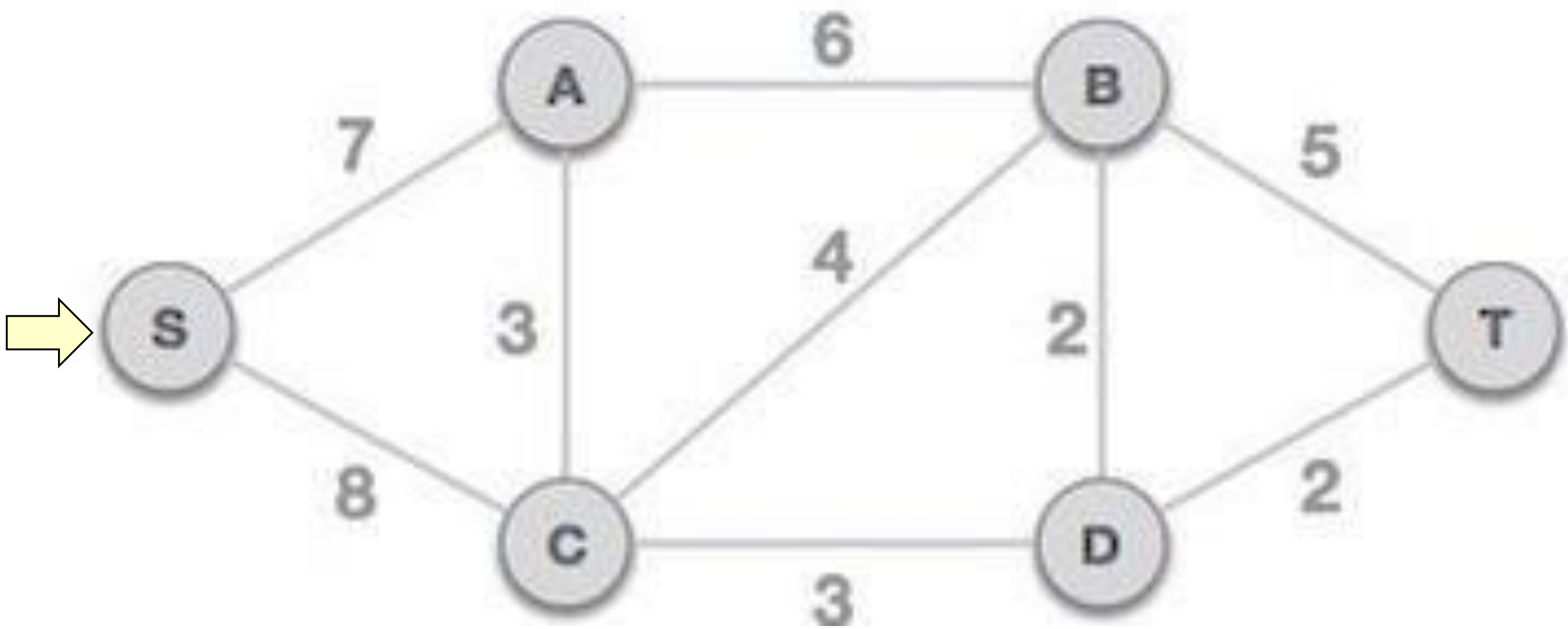
Prim's MST Example 2 (cont.)

- Remove all loops and parallel edges from the given graph.
- In case of parallel edges, keep the one which has the least cost associated and remove all others.



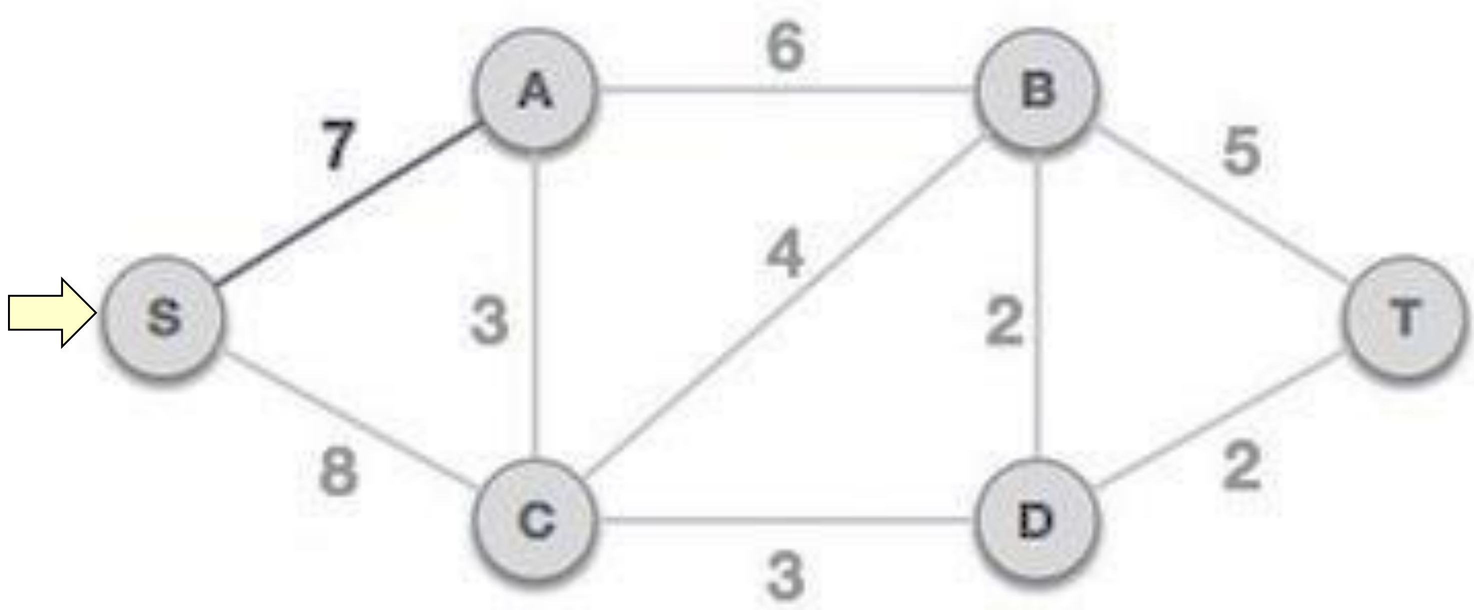
Prim’s MST Example 2 (cont.)

- Choose the given source node or any arbitrary node as source node if no input about source node.
- In this case, we choose S node as the root node of Prim's minimum spanning tree. any node can be arbitrarily chosen as the root node if no input.



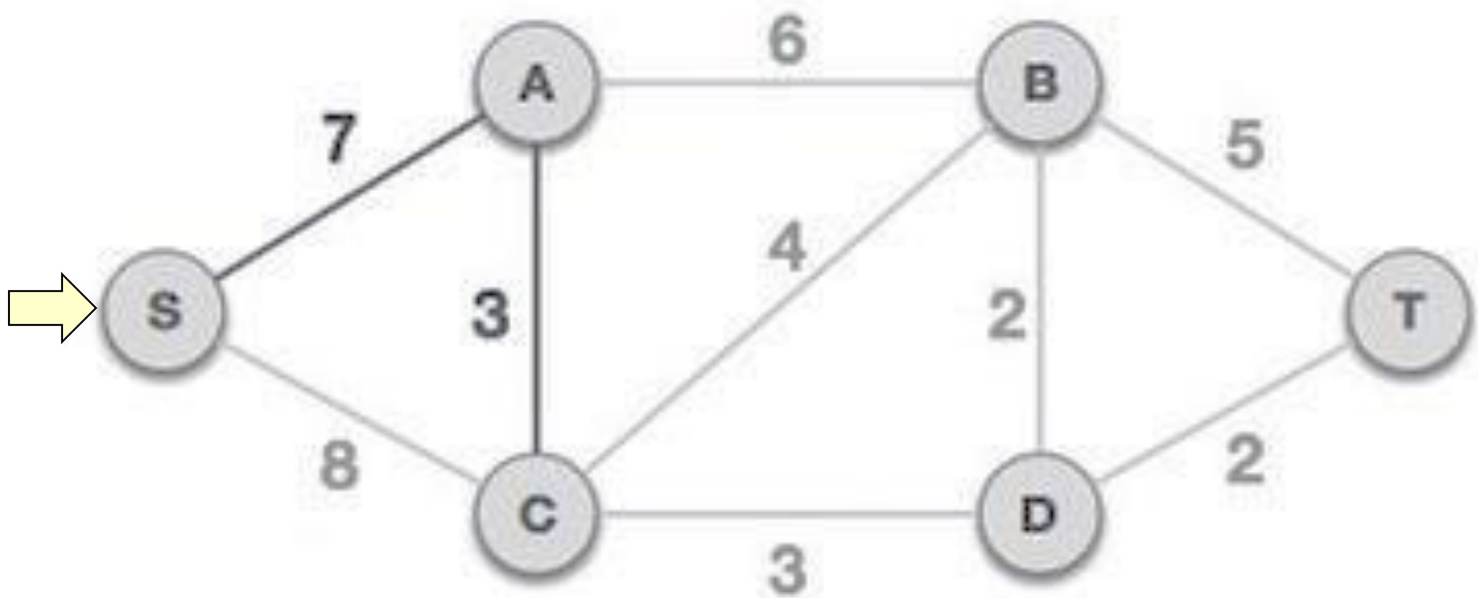
Prim's MST Example 2 (cont.)

- Check outgoing edges and select the one with less cost
  - After choosing the root node S, we see that (S,A) and (S,C) are two edges with weight 7 and 8, respectively.
  - We choose the edge (S,A) as it is lesser than the other.



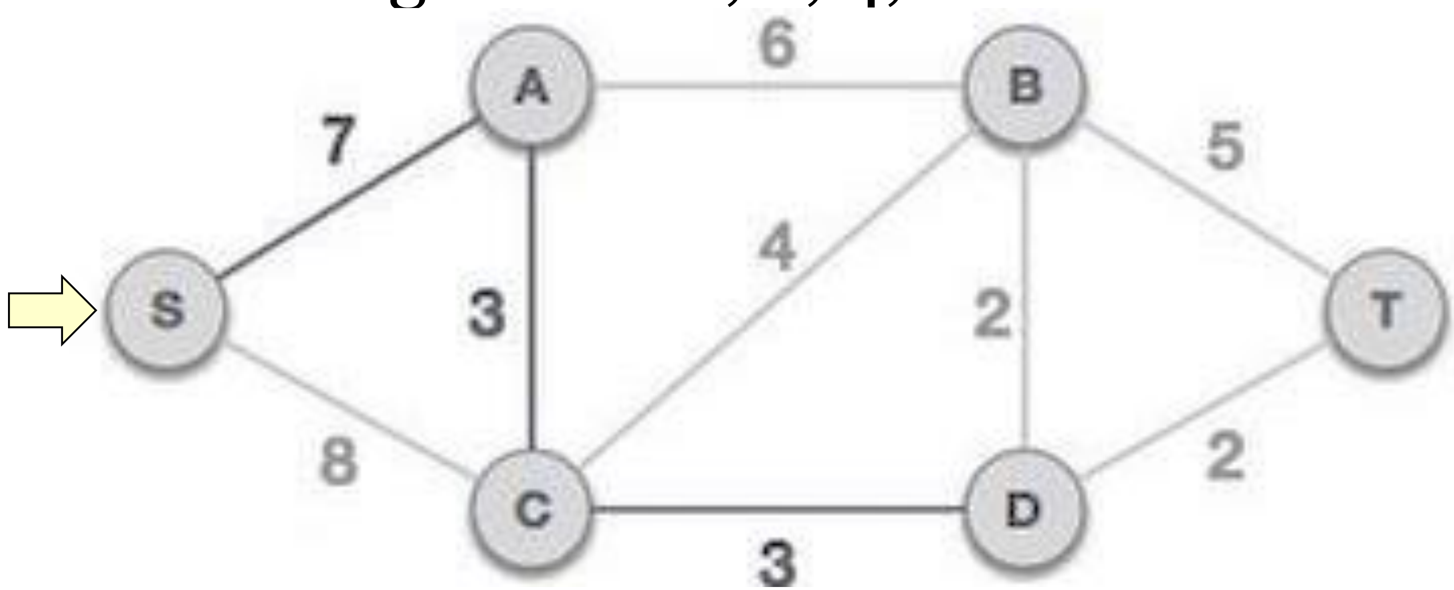
Prim’s MST Example 2 (cont.)

- Now, the tree S-7-A is treated as one node and we check for all edges going out from it.
- We select the one which has the lowest cost and include it in the tree.



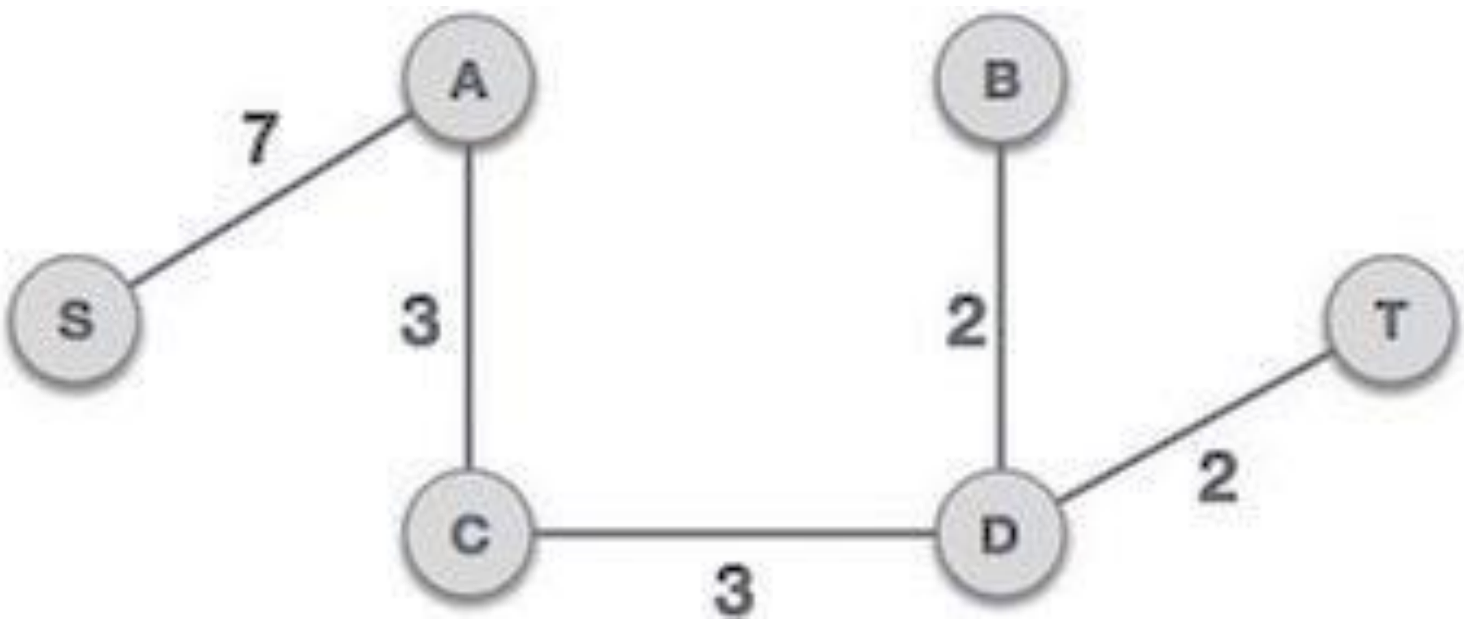
Prim's MST Example 2 (cont.)

- After this step, S-7-A-3-C tree is formed.
- Now we'll again treat it as a node and will check all the edges again.
- However, we will choose only the least cost edge.
- In this case, C-3-D is the new edge, which is less than other edges' cost 8, 6, 4, etc.



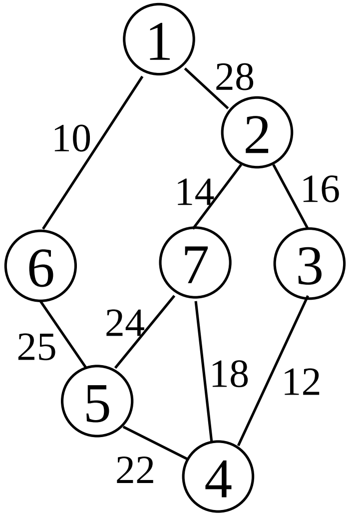
Prim's MST Example 2 (cont.)

- After adding node D to the spanning tree, we now have two edges going out of it having the same cost, i.e. D-2-T and D-2-B.
- Thus, we can add either one. But the next step will again yield edge 2 as the least cost.
- Hence, the final spanning tree is as shown below.



```
Algorithm Kruskal(E, cost, n, t)
// E is the set of edges in G. G has n vertices. cost[u, v] is the
// cost of edge (u, v). t is the set of edges in the minimum-cost
// spanning tree. The final cost is returned.
{
    Construct a heap out of the edge costs using Heapify;
    for i := 1 to n do parent[i] := -1;
    // Each vertex is in a different set.
    i := 0; mincost := 0.0;
    while ((i < n - 1) and (heap not empty)) do
    {
        Delete a minimum cost edge (u, v) from the heap
        and reheapify using Adjust;
        j := Find(u); k := Find(v);
        if (j ≠ k) then
        {
            i := i + 1;
            t[i, 1] := u; t[i, 2] := v;
            mincost := mincost + cost[u, v];
            Union(j, k);
        }
    }
    if (i ≠ n - 1) then write ("No spanning tree");
    else return mincost;
}
```

Examples for Kruskal's Algorithm



6/9

1
6 10

3 12 4

2 14
7

16
2 3

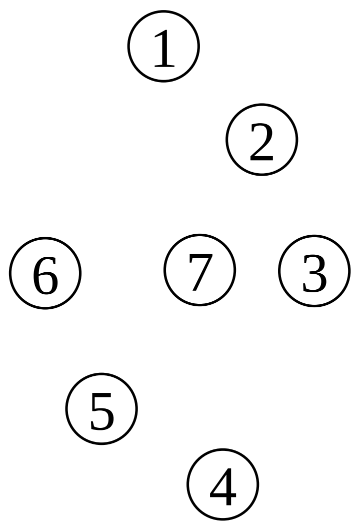
18
4
7 22

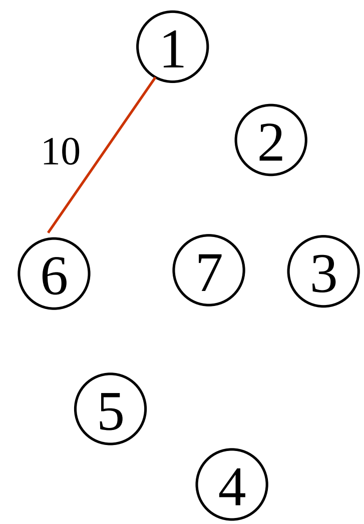
4 5
24

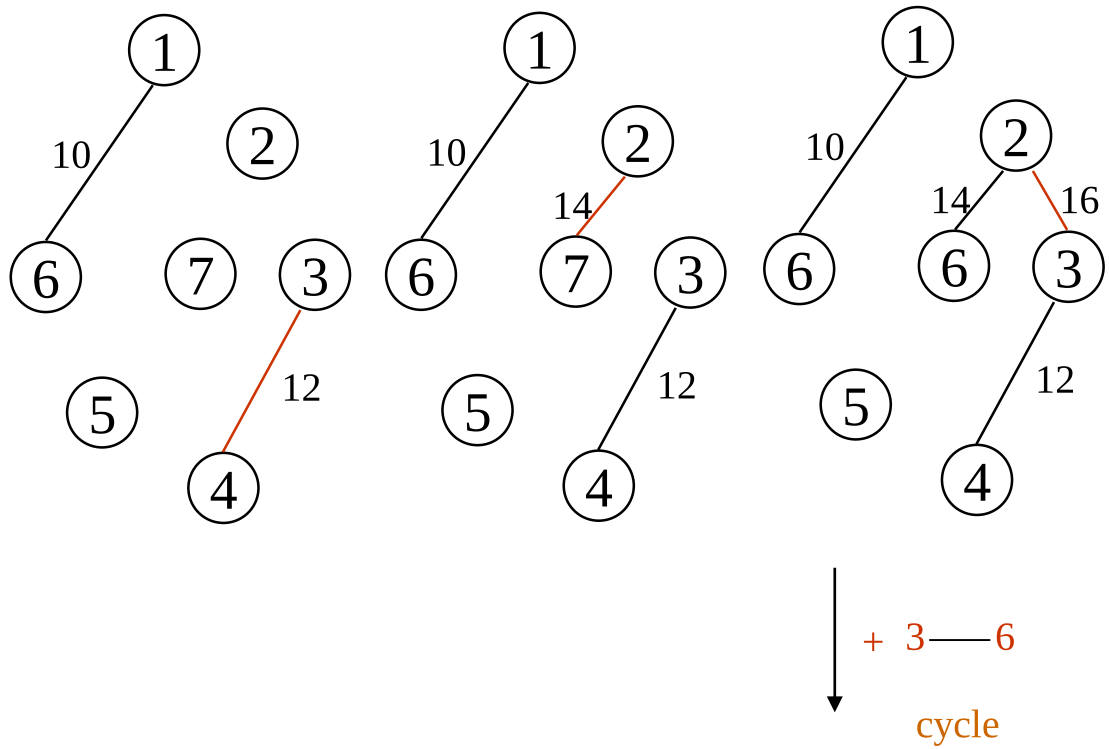
5 7
25

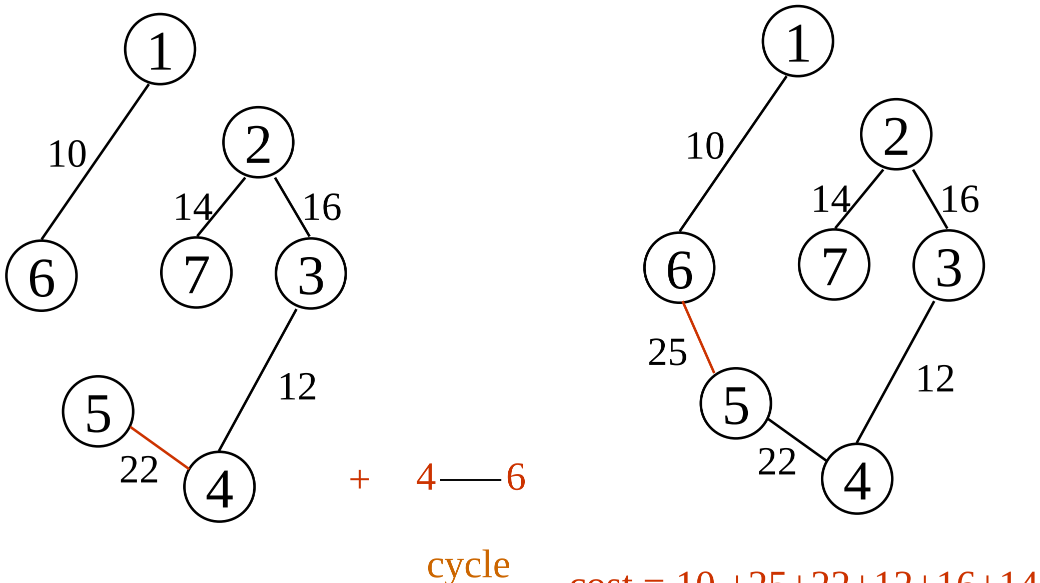
5 28 6

1 2



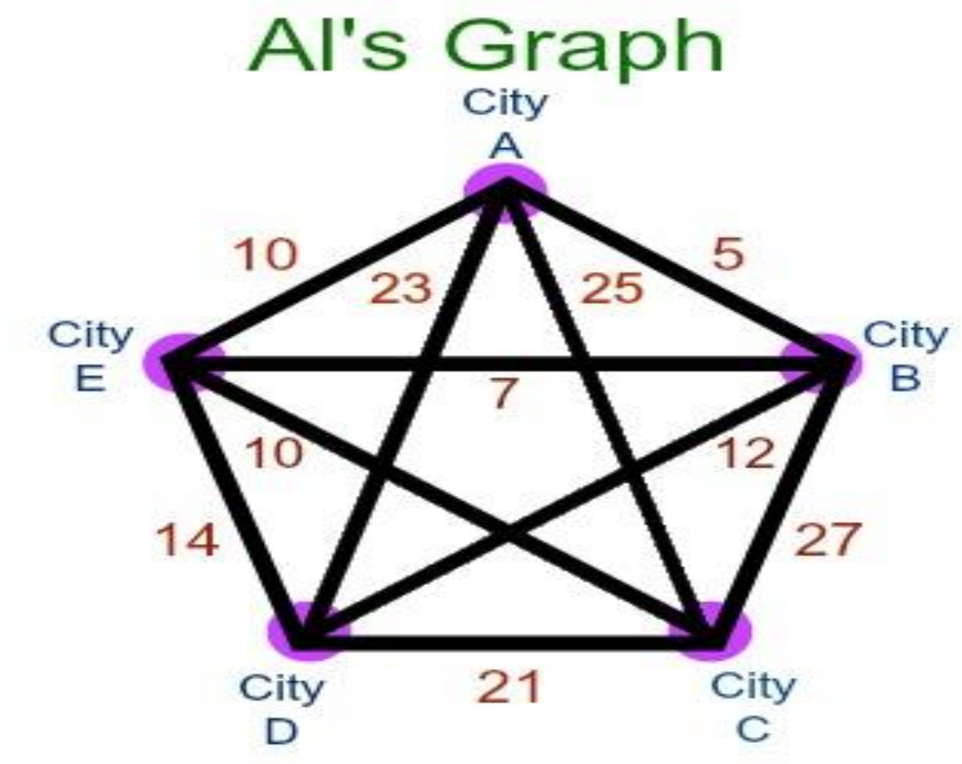



1
2
6
7
3
5
4
10
12
14
16
+ 3——6
cycle



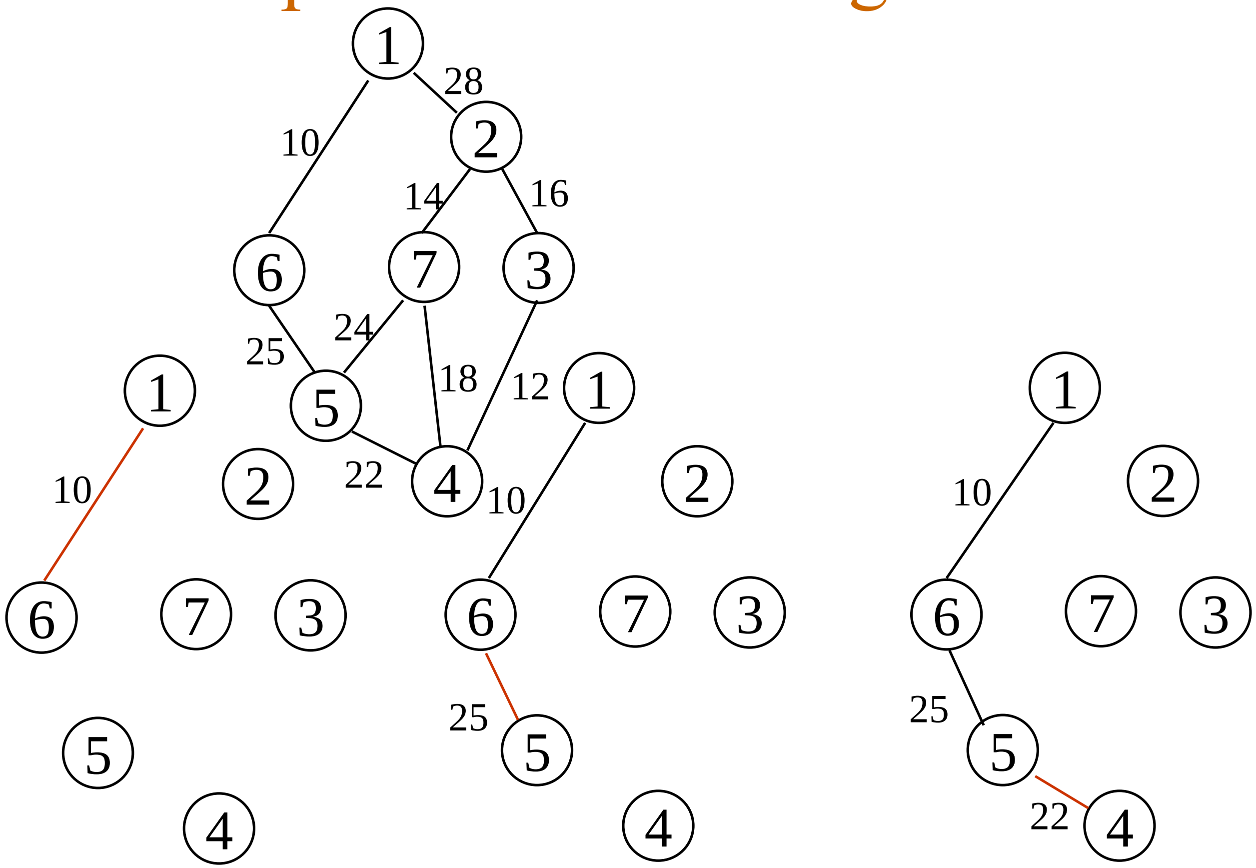
cost = 10 +25+22+12+16+14

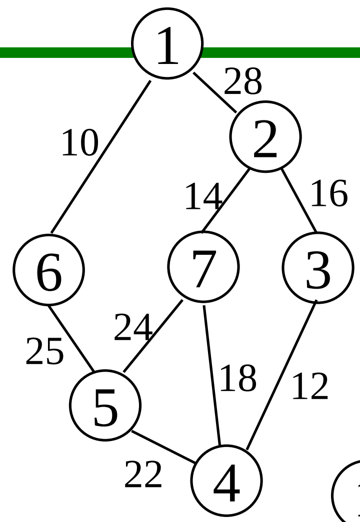
Find the Minimum spanning tree using kruskal’s algorithm

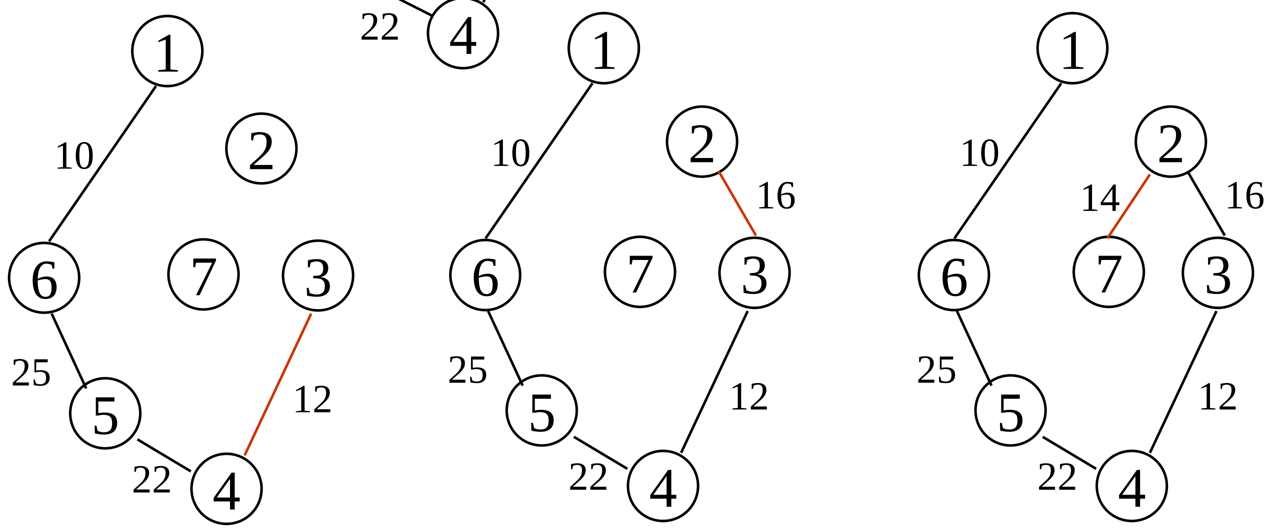


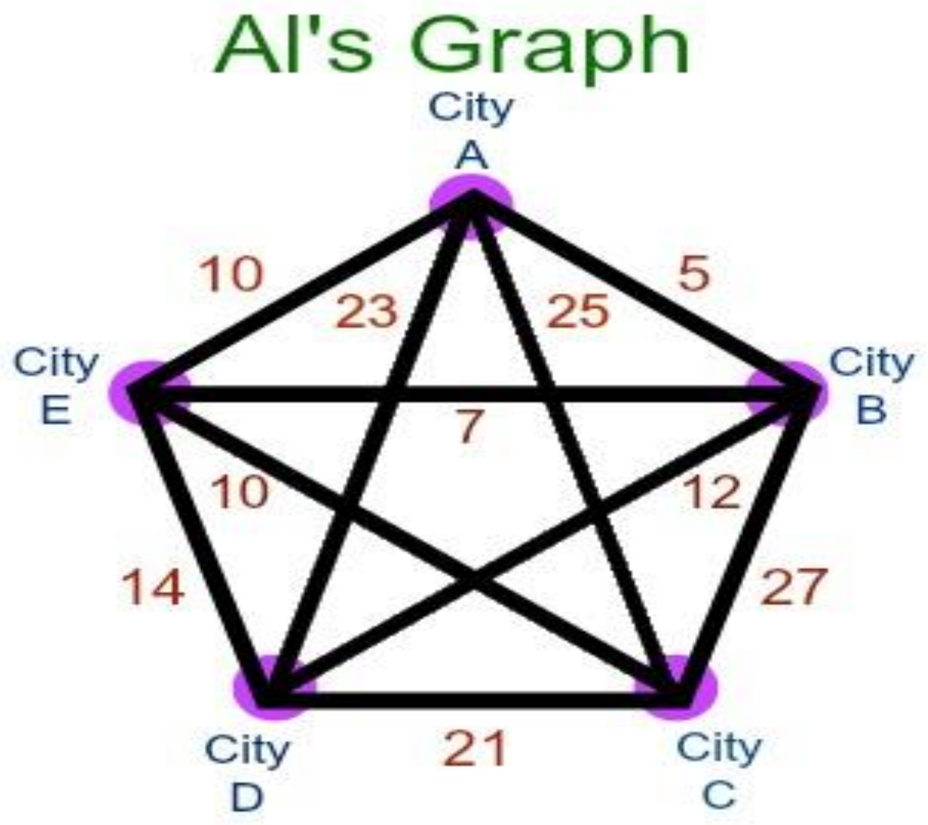
```
Algorithm Prim(E, cost, n, t)
// E is the set of edges in G. cost[1 : n, 1 : n] is the cost
// adjacency matrix of an n vertex graph such that cost[i, j] is
// either a positive real number or ∞ if no edge (i, j) exists.
// A minimum spanning tree is computed and stored as a set of
// edges in the array t[1 : n − 1, 1 : 2]. (t[i, 1], t[i, 2]) is an edge in
// the minimum-cost spanning tree. The final cost is returned.
{
    Let (k, l) be an edge of minimum cost in E;
    mincost := cost[k, l];
    t[1, 1] := k; t[1, 2] := l;
    for i := 1 to n do // Initialize near.
        if (cost[i, l] < cost[i, k]) then near[i] := l;
        else near[i] := k;
    near[k] := near[l] := 0;
    for i := 2 to n − 1 do
    { // Find n − 2 additional edges for t.
        Let j be an index such that near[j] ≠ 0 and
        cost[j, near[j]] is minimum;
        t[i, 1] := j; t[i, 2] := near[j];
        mincost := mincost + cost[j, near[j]];
        near[j] := 0;
        for k := 1 to n do // Update near[ ].
            if ((near[k] ≠ 0) and (cost[k, near[k]] > cost[k, j]))
                then near[k] := j;
    }
    return mincost;
}
```

Examples for Prim's Algorithm




1
28
10
2
14
16
6
7
3
24
25
18
12
5
22
4


1
10
2
6
7
3
25
12
5
22
4
1
10
2
16
6
7
3
25
12
5
22
4
1
10
2
14
16
6
7
3
25
12
5
22
4


Al's Graph
City
A
City
B
City
C
City
D
City
E
10
5
23
25
7
10
12
14
27
21