Graphs

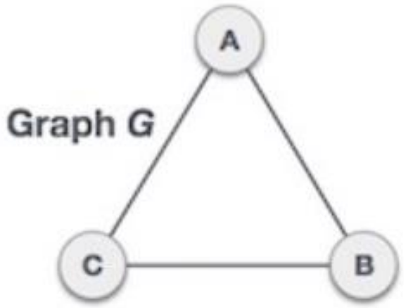
CS223: Data Structures

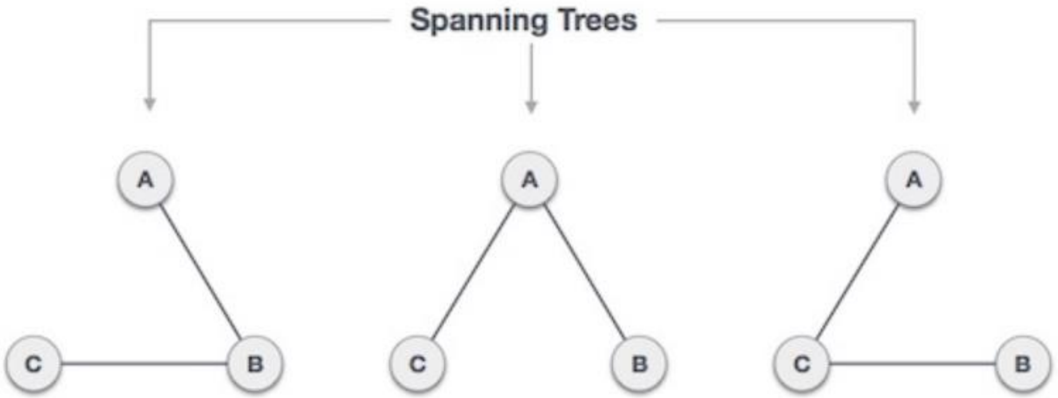
Spanning Tree

- A ***spanning tree*** is a subset of Graph G, which has all of its vertices covered with the minimum possible number of edges. Hence, a spanning tree does not have cycles and it cannot be disconnected.
- Every connected and undirected Graph G has at least one spanning tree
- A disconnected graph does not have any spanning tree, as it cannot be spanned to all its vertices.
- Removing one edge from the spanning tree will make the graph disconnected, i.e., the spanning tree is **minimally connected**.

Spanning Tree

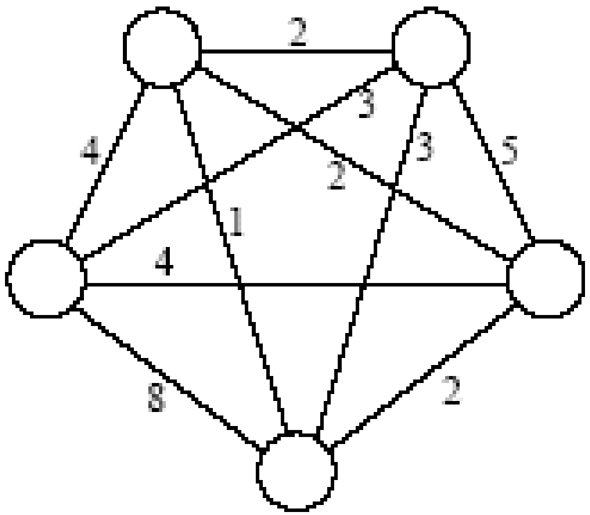
- All possible spanning trees of graph G, have the same number of edges and vertices.
- Spanning tree has n-1 edges, where n is the number of vertices.
- From a complete graph, by removing maximum e - n + 1 edges, we can construct a spanning tree.
- A complete undirected graph can have maximum n^{n-2} number of spanning trees
  - n is the number of nodes.
  - In the below example, n is 3,
    - $3^{3-2} = 3$ spanning trees

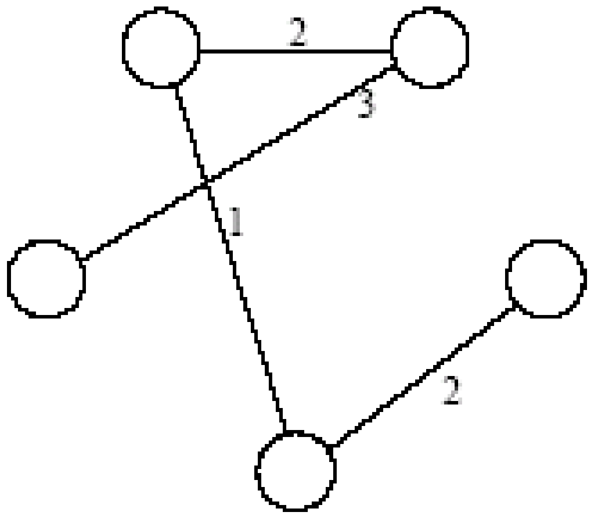




Minimum Spanning Tree

- In a weighted graph, a minimum spanning tree (MST) 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 Spanning Tree

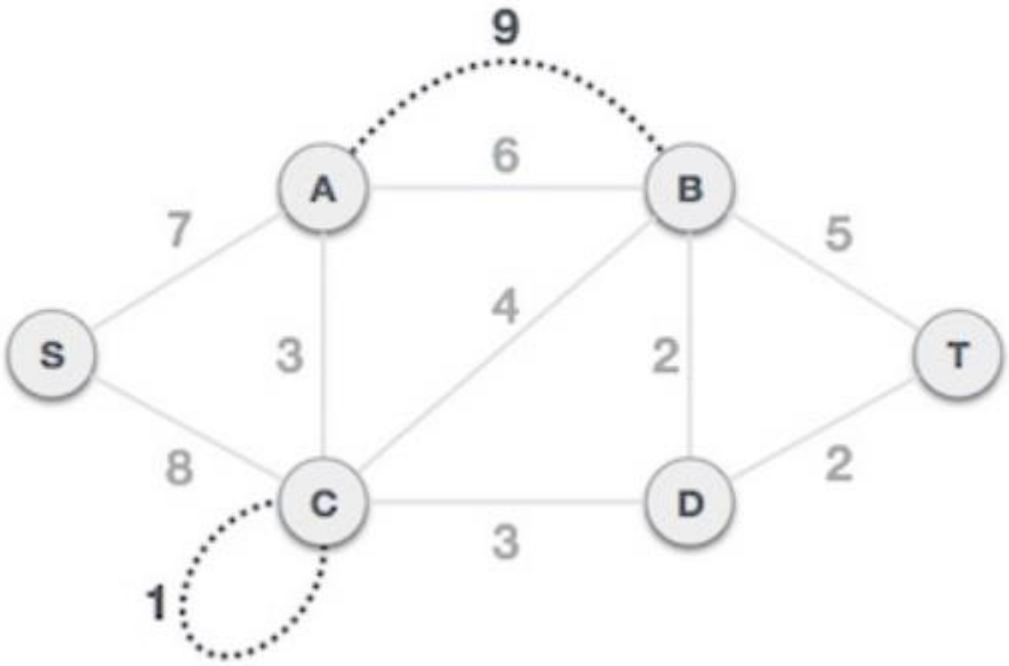
- Minimum Spanning-Tree Algorithm
  - Kruskal's Algorithm
  - Prim's Algorithm

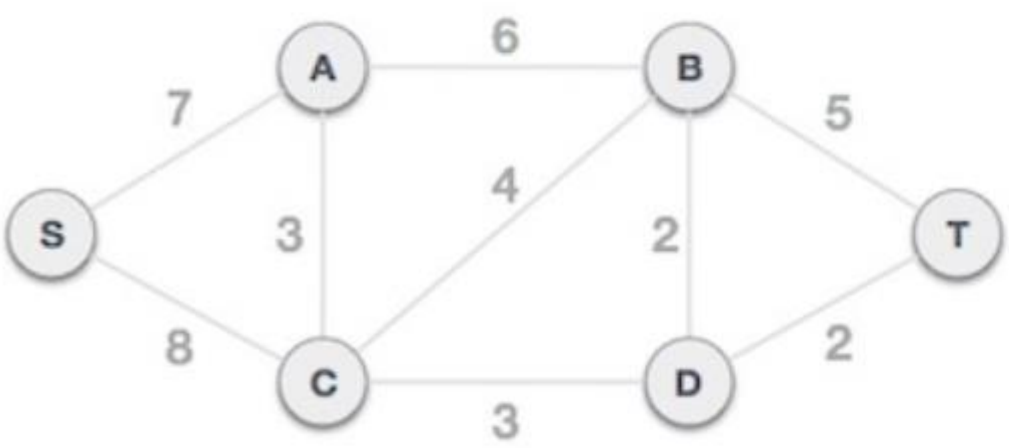
Minimum Spanning Tree: **Kruskal's Algorithm**

- **Kruskal's algorithm** for finding an MST is a **greedy** algorithm.
- Create a MST by picking edges one by one.
- The Greedy Choice is to pick the smallest weight edge that doesn’t cause a cycle in the MST constructed so far.

Minimum Spanning Tree: **Kruskal's Algorithm**

- Step 1 - Remove all loops and Parallel Edges
  - 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.

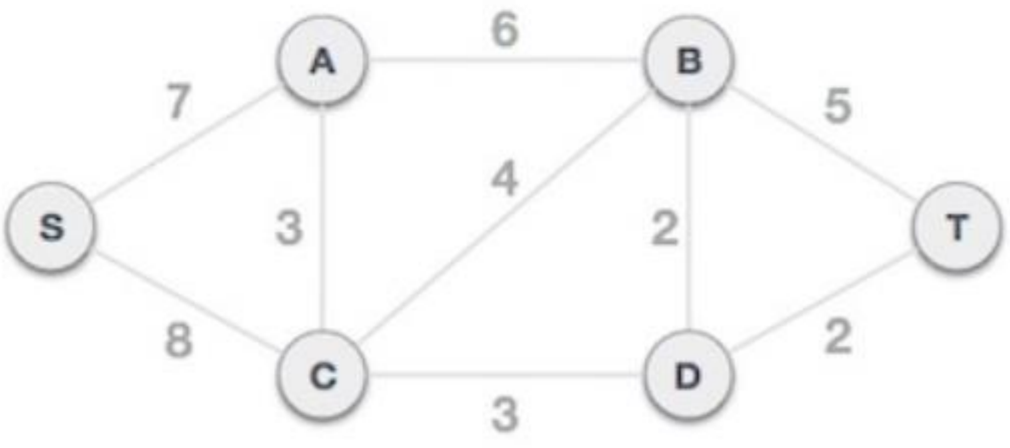




Minimum Spanning Tree: **Kruskal's Algorithm**

Step 2 - Sort all edges in their increasing order of weight

- The next step is to create a set of edges and weight, and arrange them in an ascending order of weightage (cost).

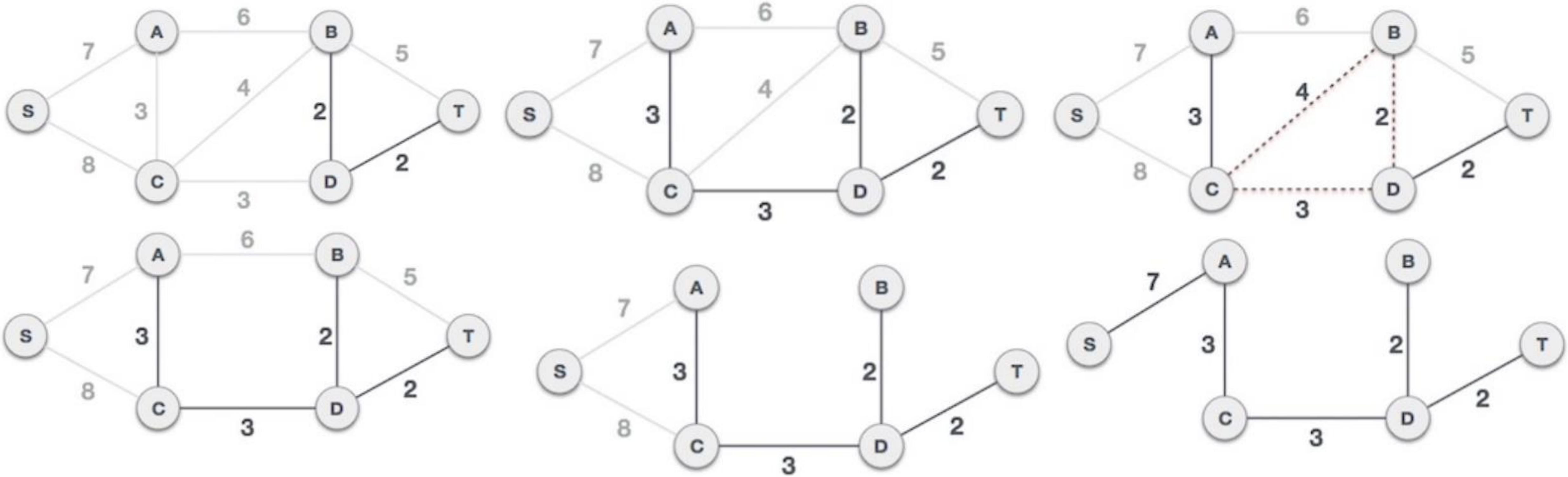


B, D	D, T	A, C	C, D	C, B	B, T	A, B	S, A	S, C
2	2	3	3	4	5	6	7	8

Minimum Spanning Tree: **Kruskal's Algorithm**

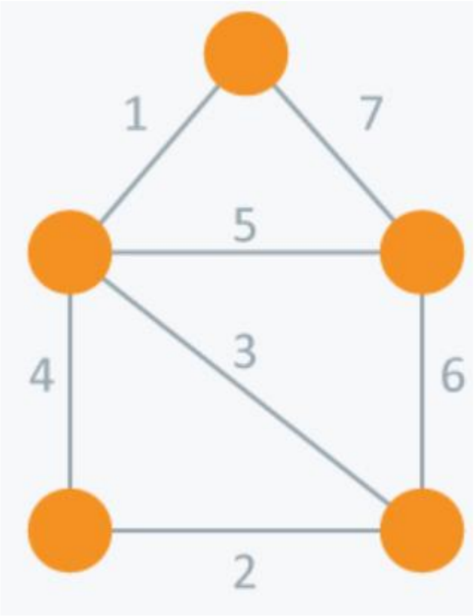
Step 3 - Add the edge which has the least weightage

- Start adding edges to the graph beginning from the one which has the least weight.
- In case, by adding one edge, the spanning tree property does not hold, then do not include the edge in the graph.

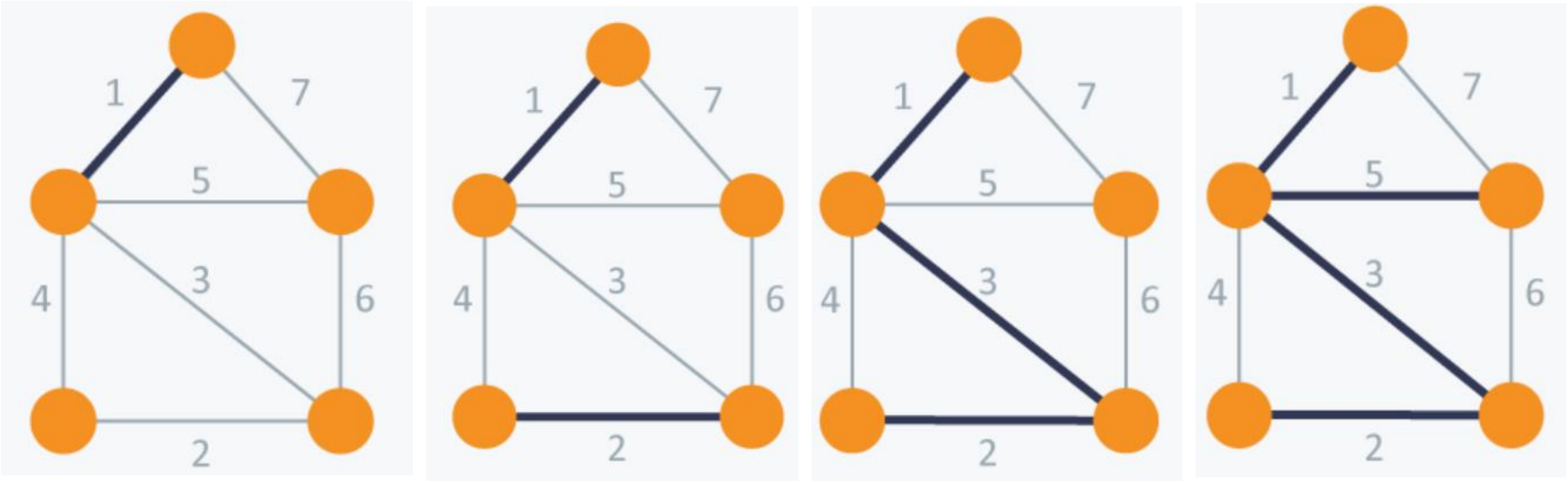


Minimum Spanning Tree: **Kruskal's Algorithm** (example 2)

A given graph



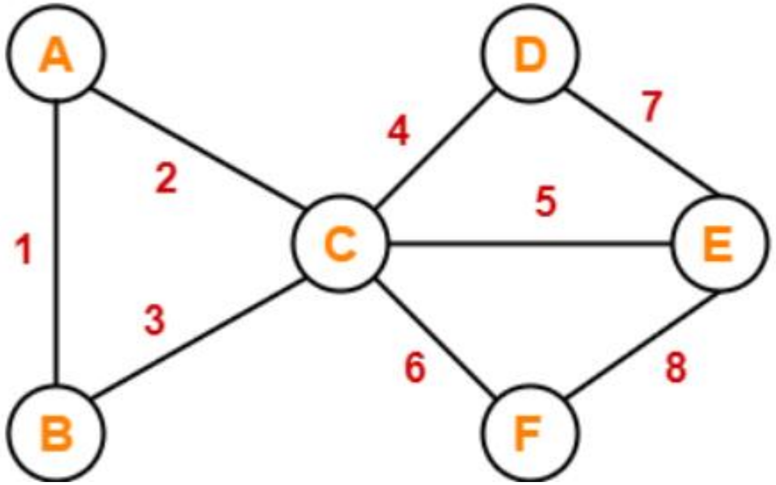
Kruskal's Algorithm Steps:



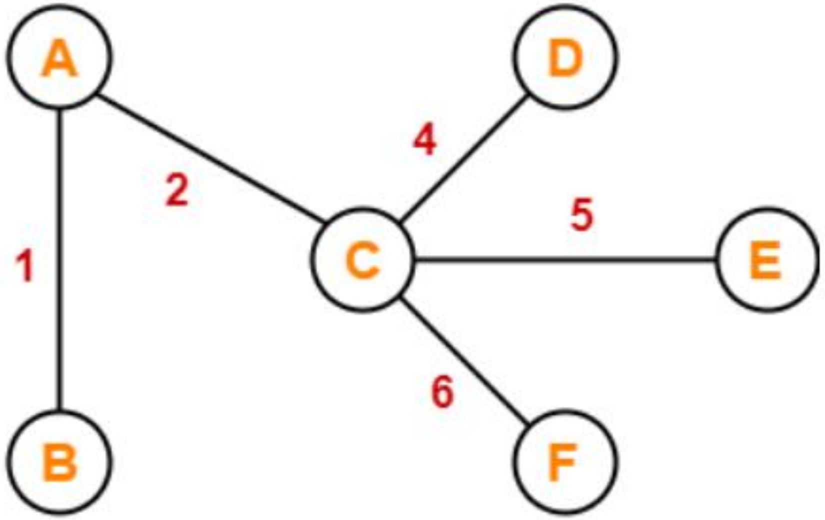
The cost of the minimum spanning tree = 11 (1 + 2 + 3 + 5).

Minimum Spanning Tree: **Kruskal's Algorithm**
(examples 3 & 4)

A given graph

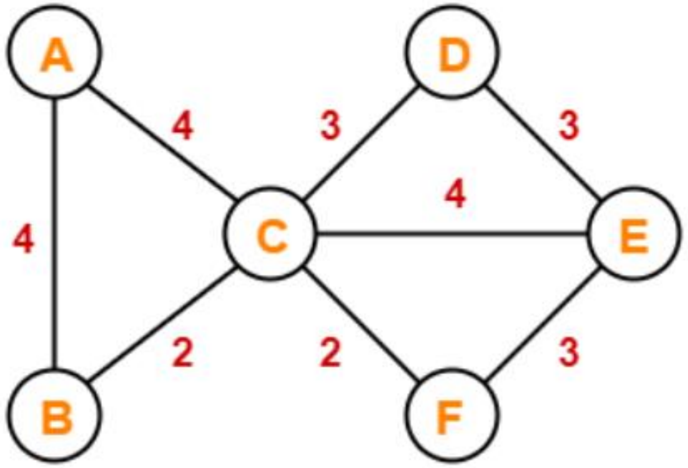


MST

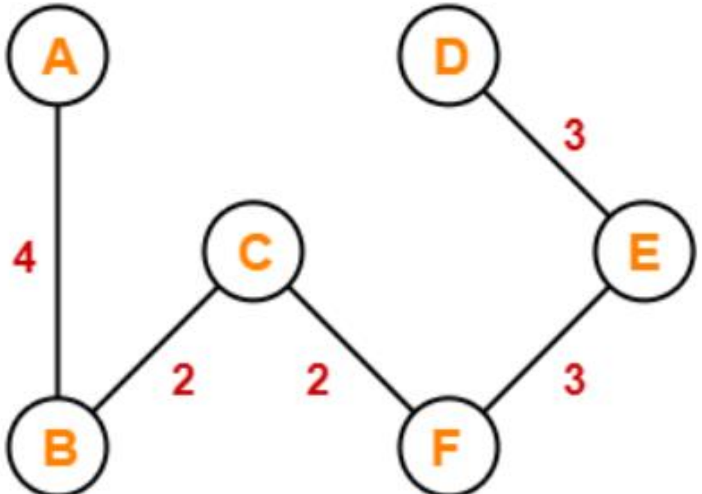


The cost of the minimum spanning tree = 18 (1 + 2 + 4 + 5 + 6).

A given graph

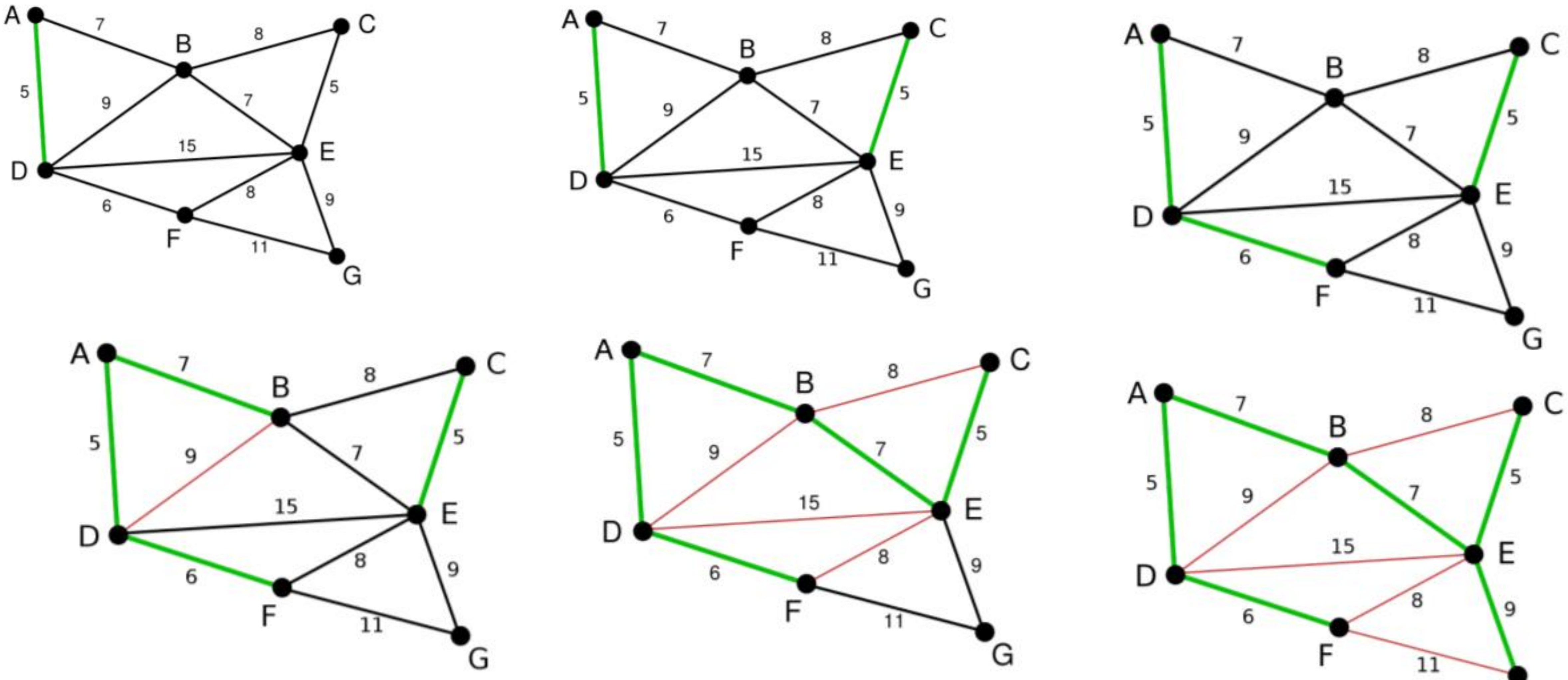


MST



The cost of the minimum spanning tree = 14 (2 + 2 + 3 + 3 + 4).

Minimum Spanning Tree: **Kruskal's Algorithm** (example 5)



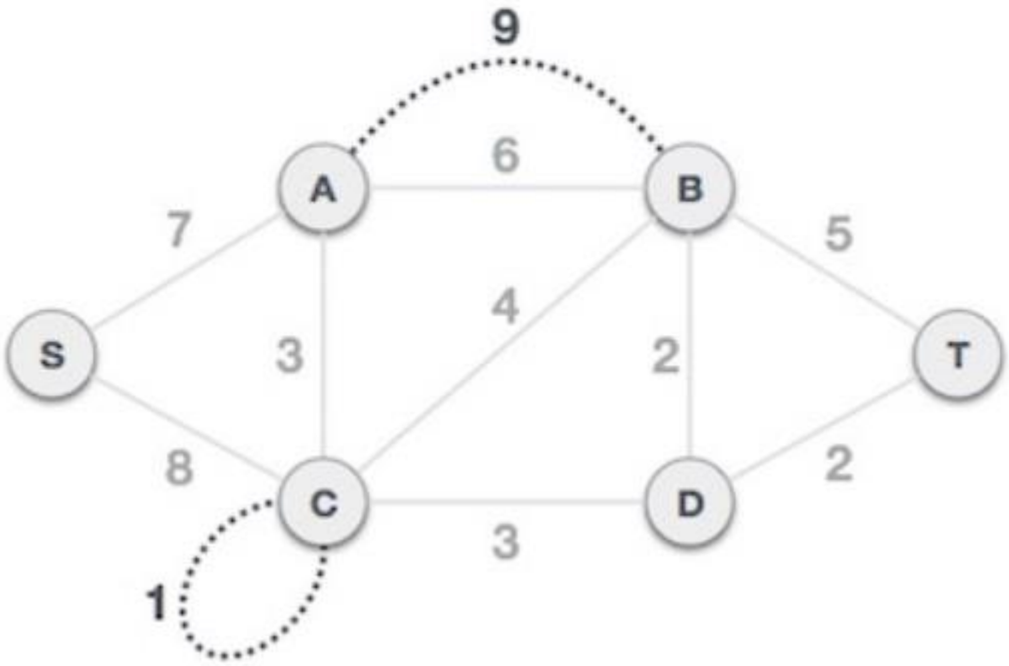
The cost of the minimum spanning tree = 39 (5 + 5 + 6 + 7 + 7 + 9).

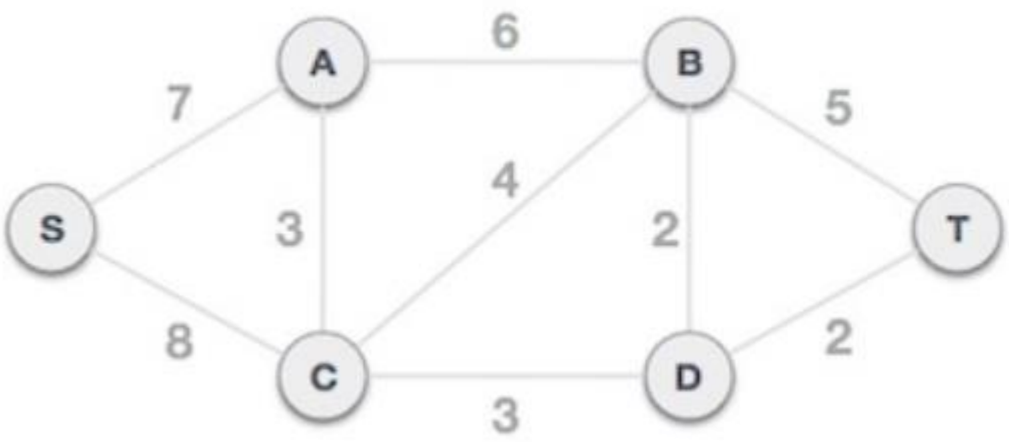
Minimum Spanning Tree: **Prim's Algorithm**

- Prim's algorithm for finding an MST is a **greedy** algorithm.
- Start by selecting an arbitrary vertex, include it into the current MST.
- Grow the current MST by inserting into it the vertex closest to one of the vertices already in current MST.

Minimum Spanning Tree: Prim's Algorithm

- Step 1 - Remove all loops and Parallel Edges
  - 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.





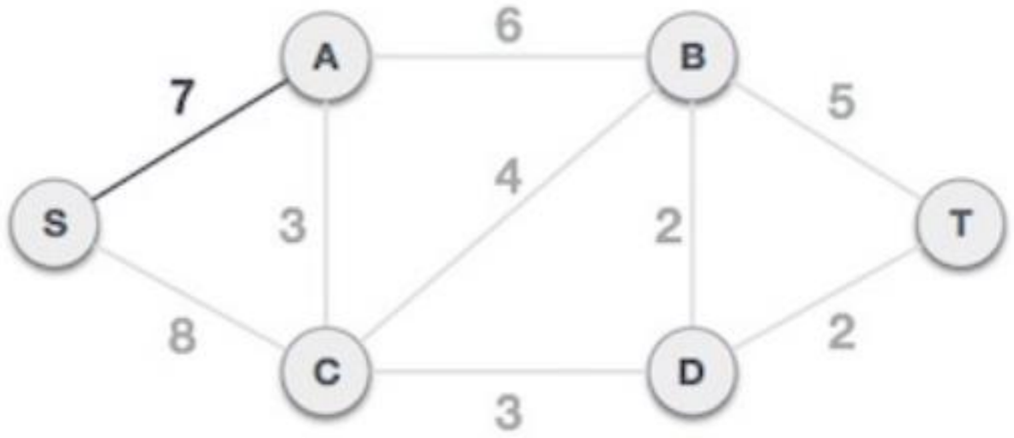
Minimum Spanning Tree: Prim's Algorithm

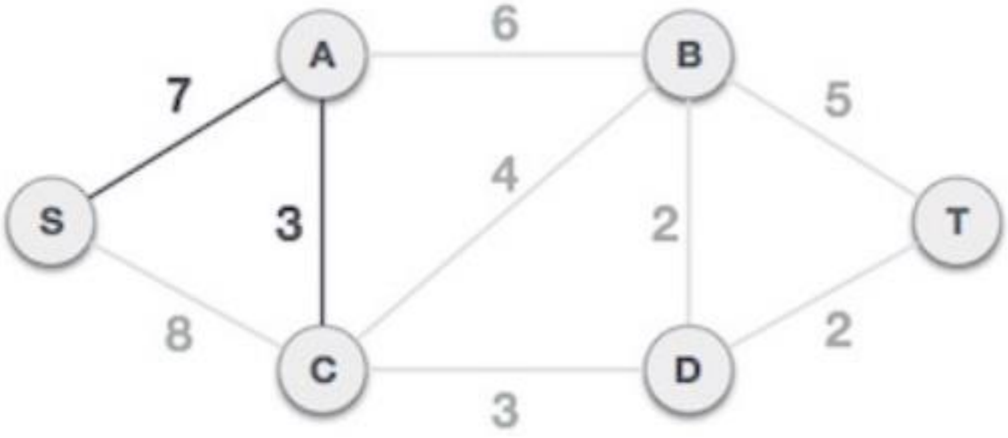
Step 2 - Choose any arbitrary node as root node

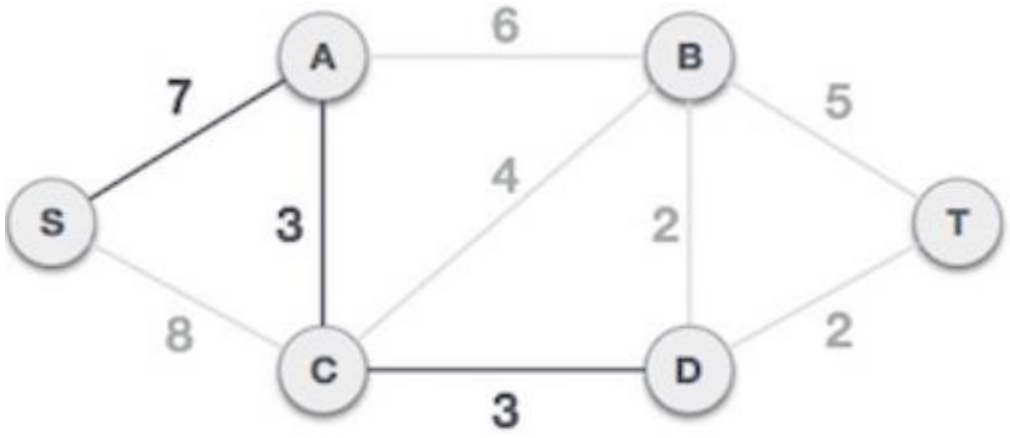
- In this case, we choose **S** node as the root node of Prim's spanning tree.
  - This node is arbitrarily chosen, so any node can be the root node.

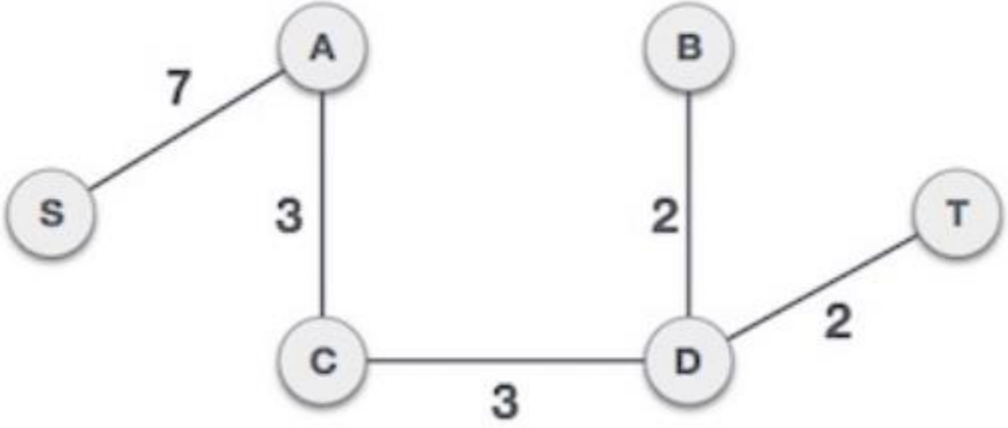
Minimum Spanning Tree: **Prim's Algorithm**

Step 3 - Check outgoing edges and select the one with less cost

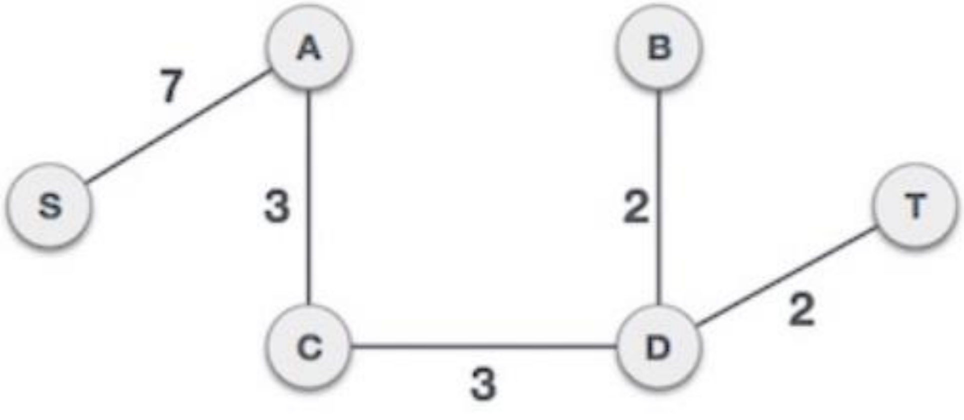




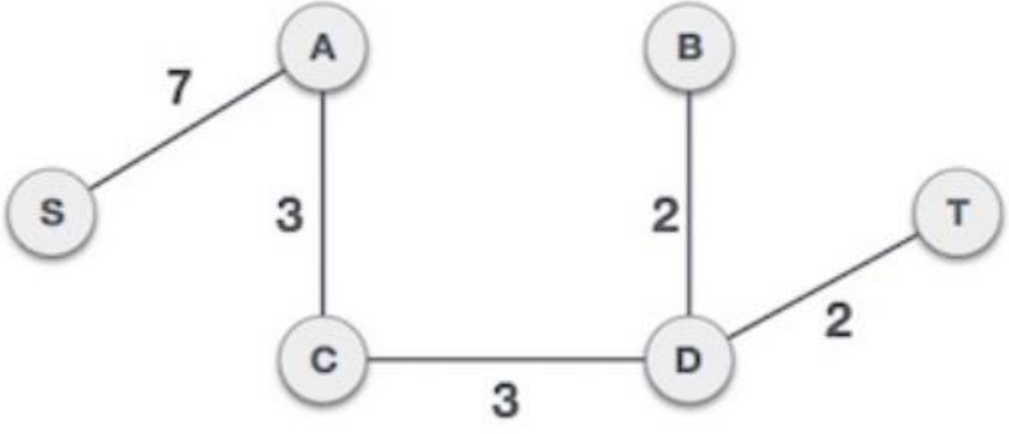




Minimum Spanning Tree: Kruskal vs. Prim



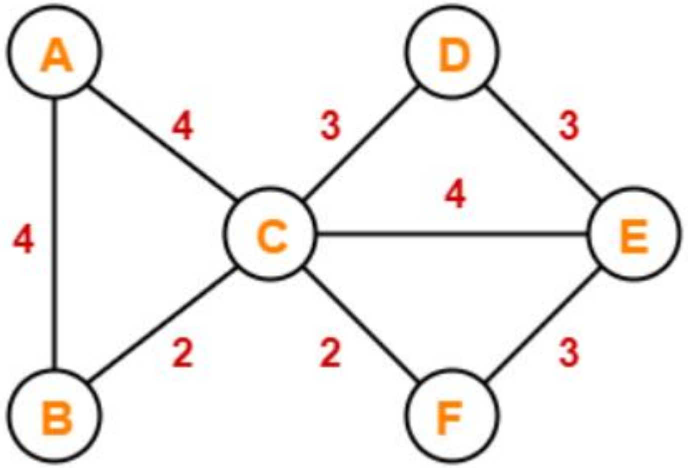
Kruskal

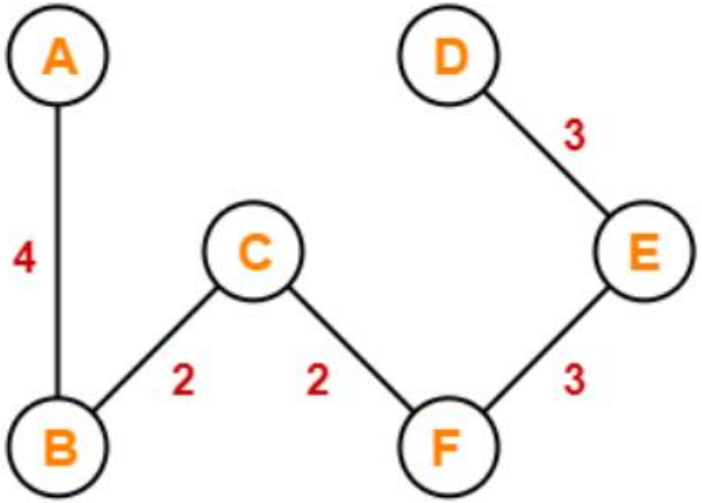


Prim

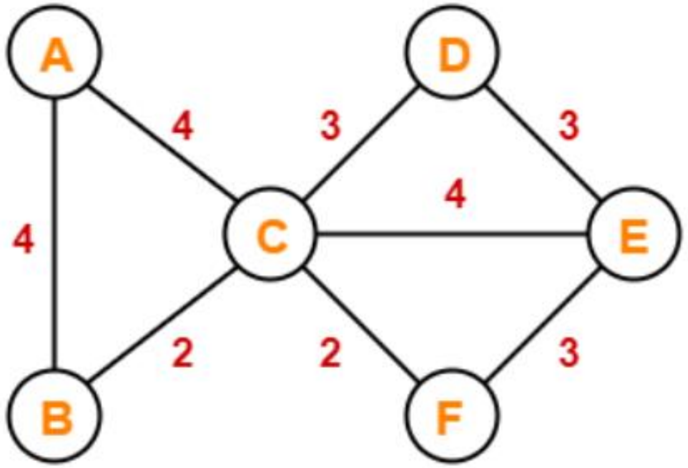
The output spanning tree of the same graph using two different algorithms might be the same.

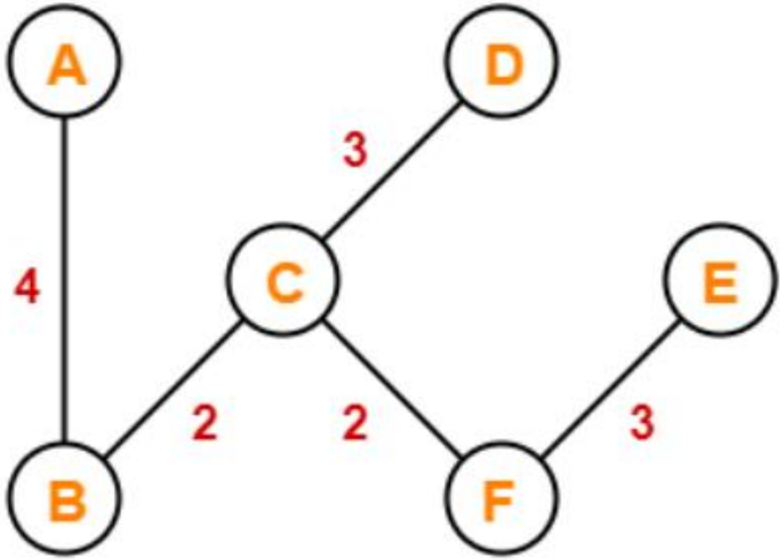
Minimum Spanning Tree: **Kruskal** vs. **Prim**





Kruskal:
The cost of the minimum spanning tree = 14 (2 + 2 + 3 + 3 + 4).





Prim:
The cost of the minimum spanning tree = 14 (2 + 2 + 3 + 3 + 4).

Minimum Spanning Tree: **Kruskal** vs. **Prim**

Prim's Algorithm	Kruskal's Algorithm
The tree that we are making or growing always remains connected.	The tree that we are making or growing usually remains disconnected.
Prim's Algorithm grows a solution from a random vertex by adding the next cheapest vertex to the existing tree.	Kruskal's Algorithm grows a solution from the cheapest edge by adding the next cheapest edge to the existing tree / forest.
Prim's Algorithm is faster for dense graphs.	Kruskal's Algorithm is faster for sparse graphs.