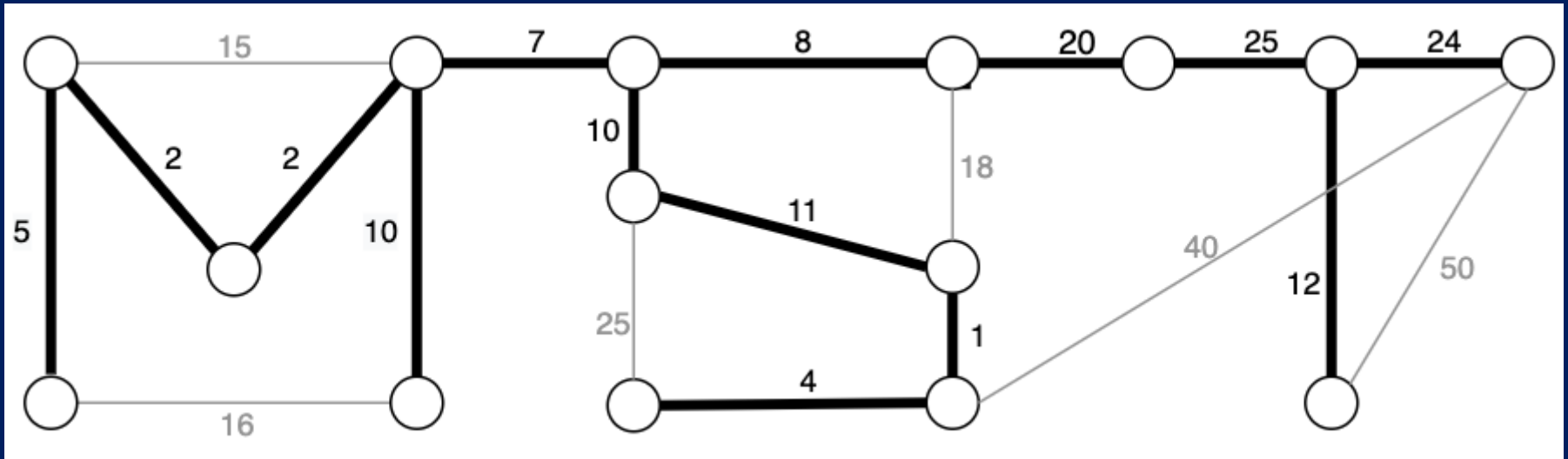
GLA UNIVERSITY MATHURA
Recognised by UGC Under Section 2(f)
Accredited with A Grade by NAAC

DESIGN & ANALYSIS OF ALGORITHM (BCSC0012)

Chapter 11: Minimum Spanning Tree

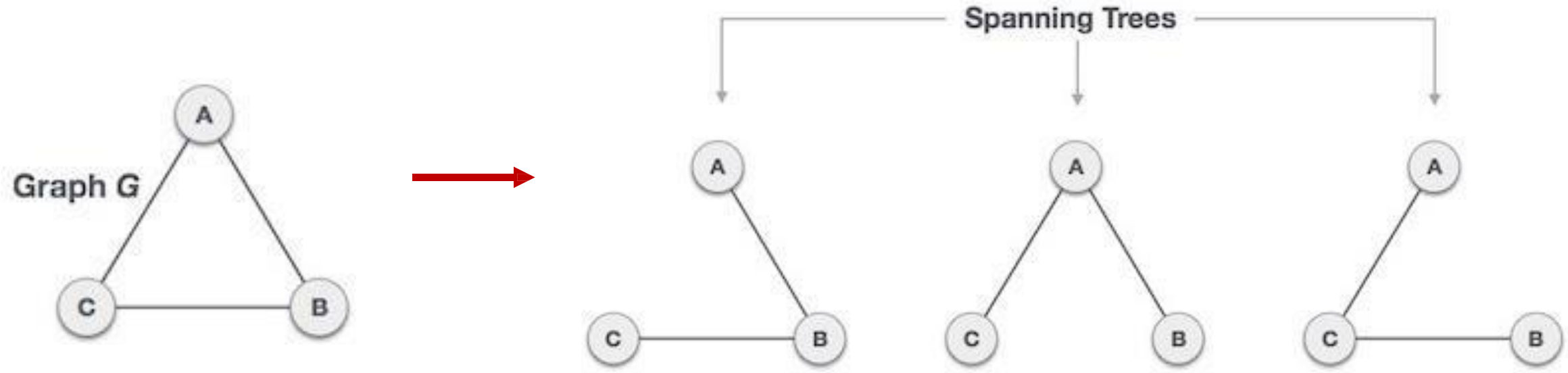


Prof. Anand Singh Jalal

Department of Computer Engineering & Applications

Spanning Tree

- A spanning tree is a subset of Graph G, which has all the vertices covered with 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 complete undirected graph can have maximum n^{n-2} number of spanning trees, where n is the number of nodes.

General Properties of Spanning Tree

Following are few properties of the spanning tree connected to graph G –

- A connected graph G can have more than one spanning tree.
- All possible spanning trees of graph G, have the same number of 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.
- 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.

Application of Spanning Tree

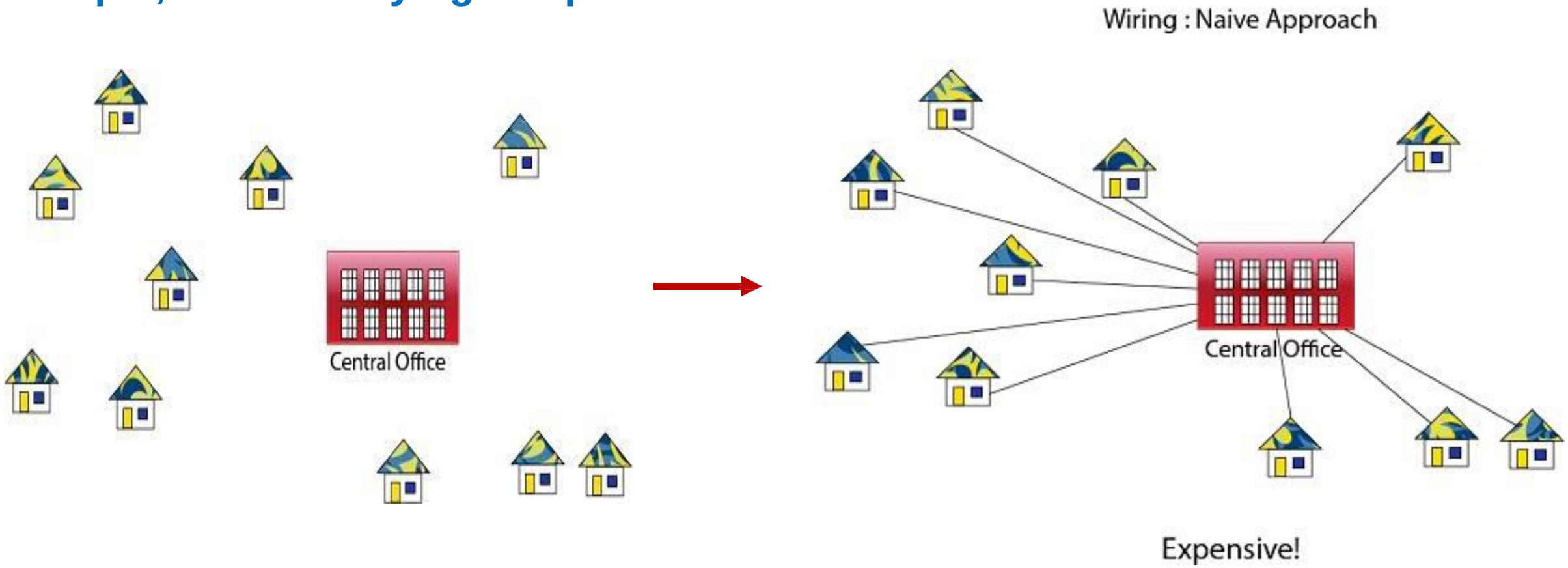
Spanning tree is basically used to find a minimum path to connect all nodes in a graph. Common applications of spanning trees are –

- Civil Network Planning
- Computer Network Routing Protocol
- Cluster Analysis
- Designing Local Area Networks
- Construct highways or railroads

Let us understand this through a small 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.

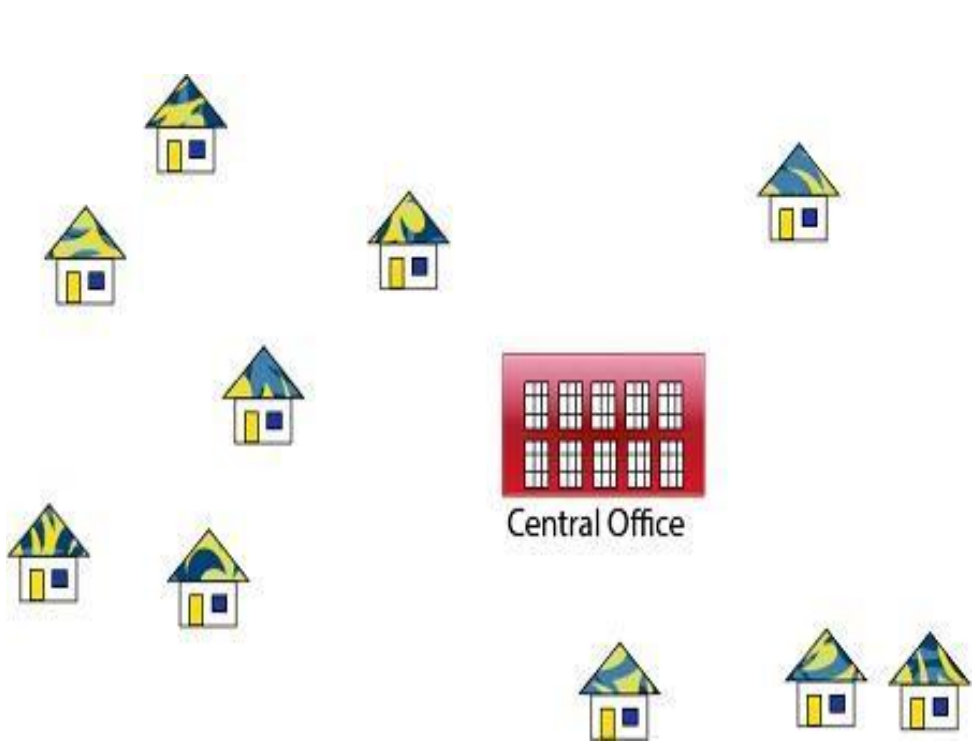
Application of Spanning Tree ...

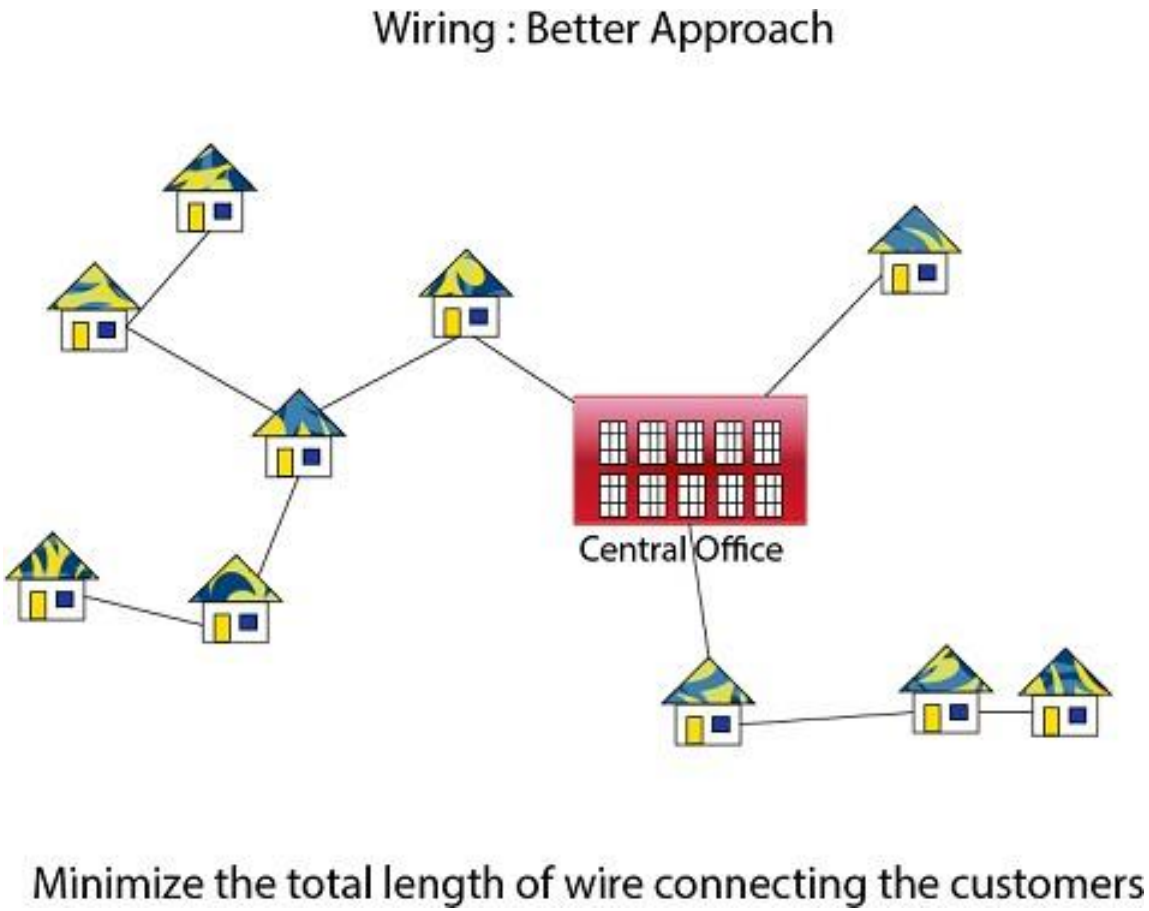
Example, Problem laying Telephone Wire



Application of Spanning Tree ...

Example, Problem laying Telephone Wire





Minimum Spanning Tree (MST)

- 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.

Two most important spanning tree algorithms here –

1.Kruskal's Algorithm

2.Prim's Algorithm

Both are greedy algorithms.

Minimum Spanning Tree (MST)

Kruskal's Algorithm

- Kruskal's algorithm to find the minimum cost spanning tree uses the **Greedy approach**.
- 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.

Algorithm Steps:

- Sort the graph edges with respect to their weights.
- Start adding edges to the MST from the edge with the smallest weight until the edge of the largest weight.
- Only add edges which doesn't form a cycle , edges which connect only disconnected components.

Minimum Spanning Tree (MST)

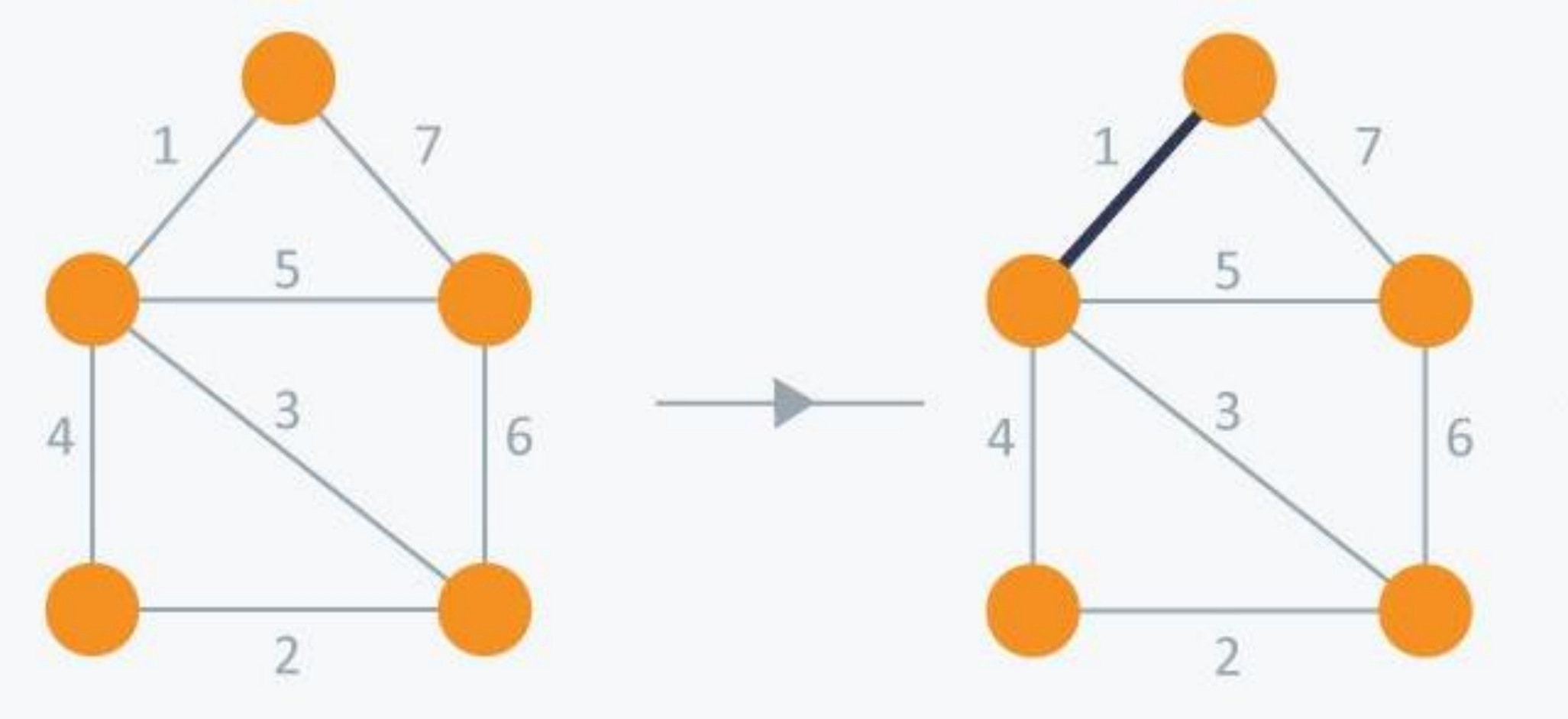
Kruskal's Algorithm

ALGORITHM *Kruskal*(G)

//Kruskal's algorithm for constructing a minimum spanning tree
//Input: A weighted connected graph $G = \langle V, E \rangle$
//Output: E_T, the set of edges composing a minimum spanning tree of G
sort E in nondecreasing order of the edge weights $w(e_{i_1}) \leq \cdots \leq w(e_{i_{|E|}})$
$E_T \leftarrow \varnothing$; $ecounter \leftarrow 0$ //initialize the set of tree edges and its size
$k \leftarrow 0$ //initialize the number of processed edges
while $ecounter < |V| - 1$ **do**
    $k \leftarrow k + 1$
    if $E_T \cup \{e_{i_k}\}$ is acyclic
        $E_T \leftarrow E_T \cup \{e_{i_k}\}$; $ecounter \leftarrow ecounter + 1$
return E_T

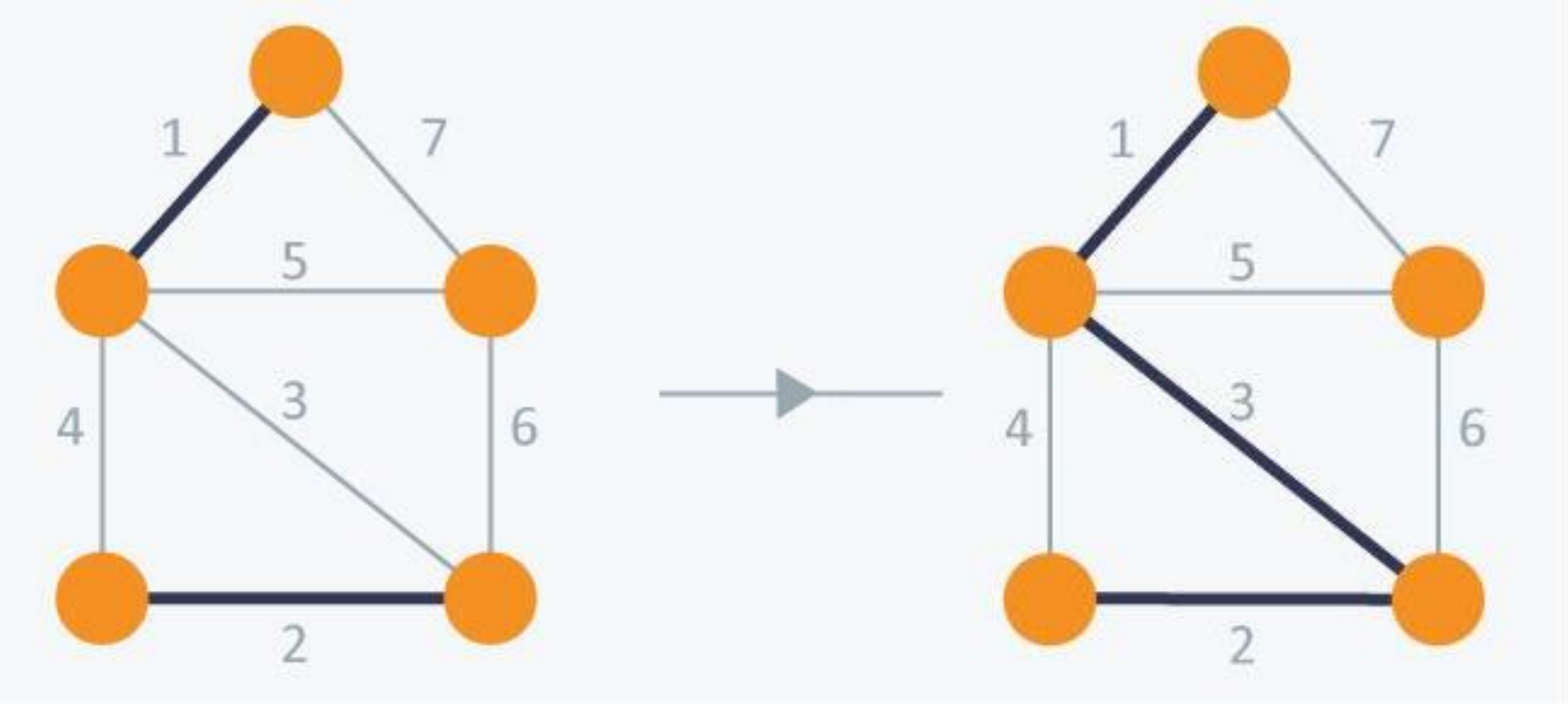
Minimum Spanning Tree (MST)

Kruskal's Algorithm



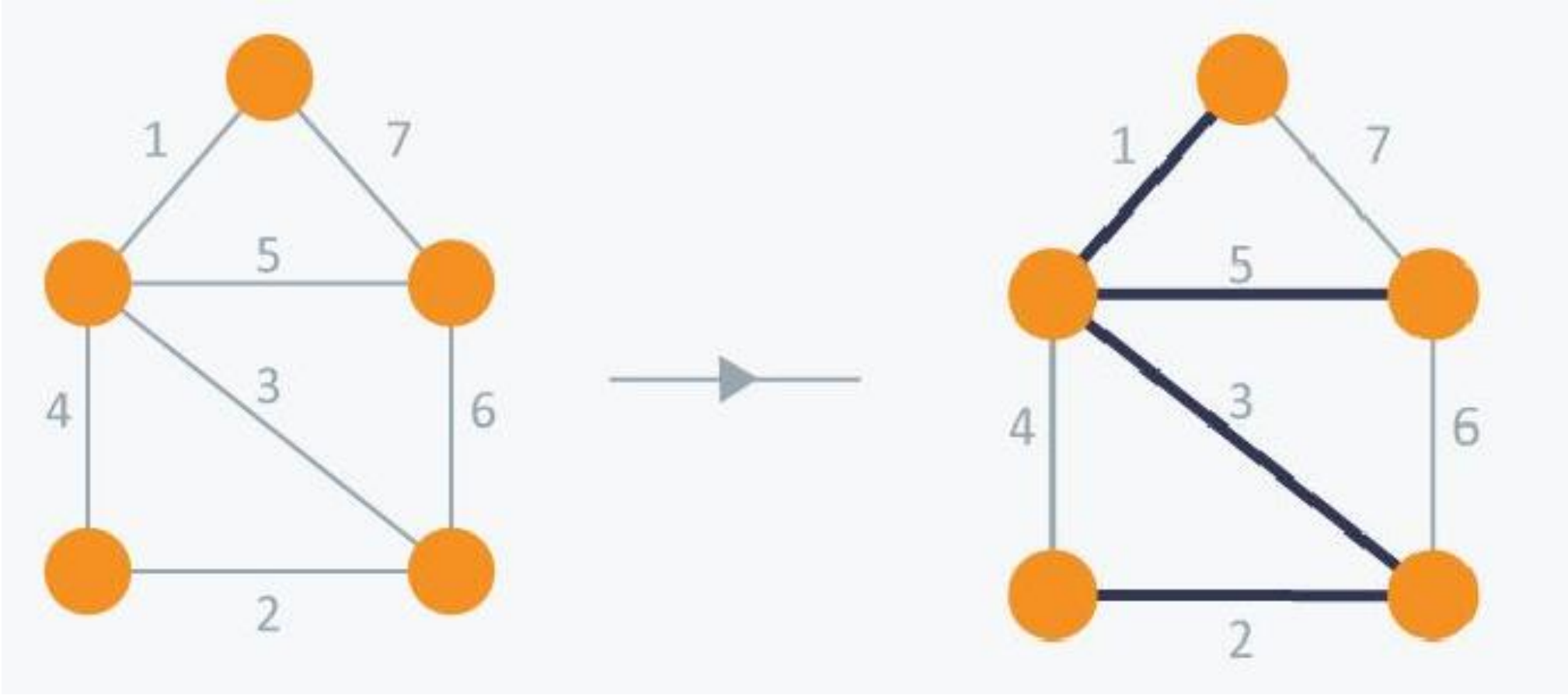
Minimum Spanning Tree (MST)

Kruskal's Algorithm



Minimum Spanning Tree (MST)

Kruskal's Algorithm

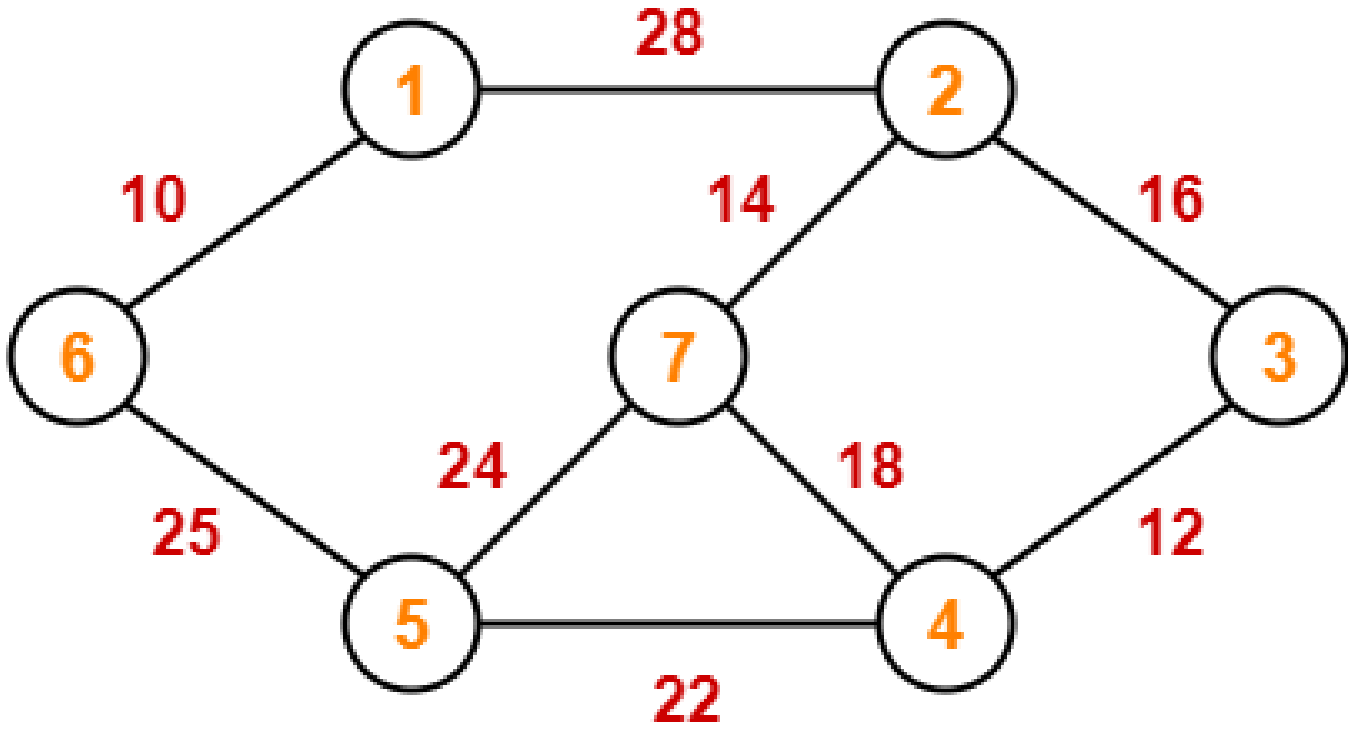


Minimum spanning tree with total cost 11 (= 1 + 2 + 3 + 5).

Minimum Spanning Tree (MST)

Kruskal's Algorithm

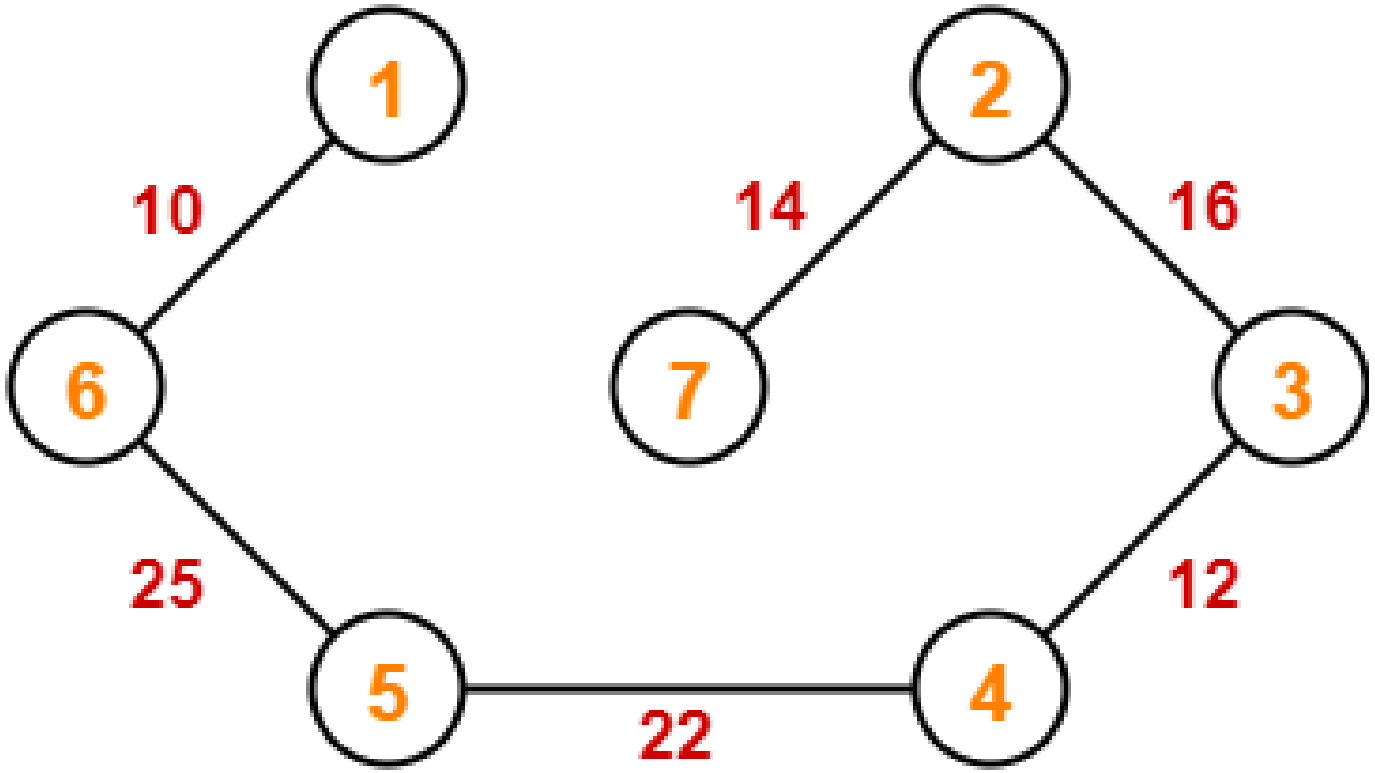
Problem: Construct the minimum spanning tree (MST) for the given graph using Kruskal's Algorithm-



Minimum Spanning Tree (MST)

Kruskal's Algorithm

Solution



Weight of the MST = Sum of all edge weights = 10 + 25 + 22 + 12 + 16 + 14 = 99 units

Minimum Spanning Tree (MST)

Prim's Algorithm

- Prim's Algorithm also use **Greedy approach** to find the minimum spanning tree.
- In Prim's Algorithm we grow the spanning tree from a starting position.
- Unlike an edge in Kruskal's, we **add vertex** to the growing spanning tree in Prim's.

Algorithm Steps:

- Maintain two disjoint sets of vertices. One containing vertices that are in the growing spanning tree and other that are not in the growing spanning tree.
- Select the cheapest vertex that is connected to the growing spanning tree and is not in the growing spanning tree and add it into the growing spanning tree. This can be done using Priority Queues. Insert the vertices, that are connected to growing spanning tree, into the Priority Queue.

Minimum Spanning Tree (MST)

Prim's Algorithm

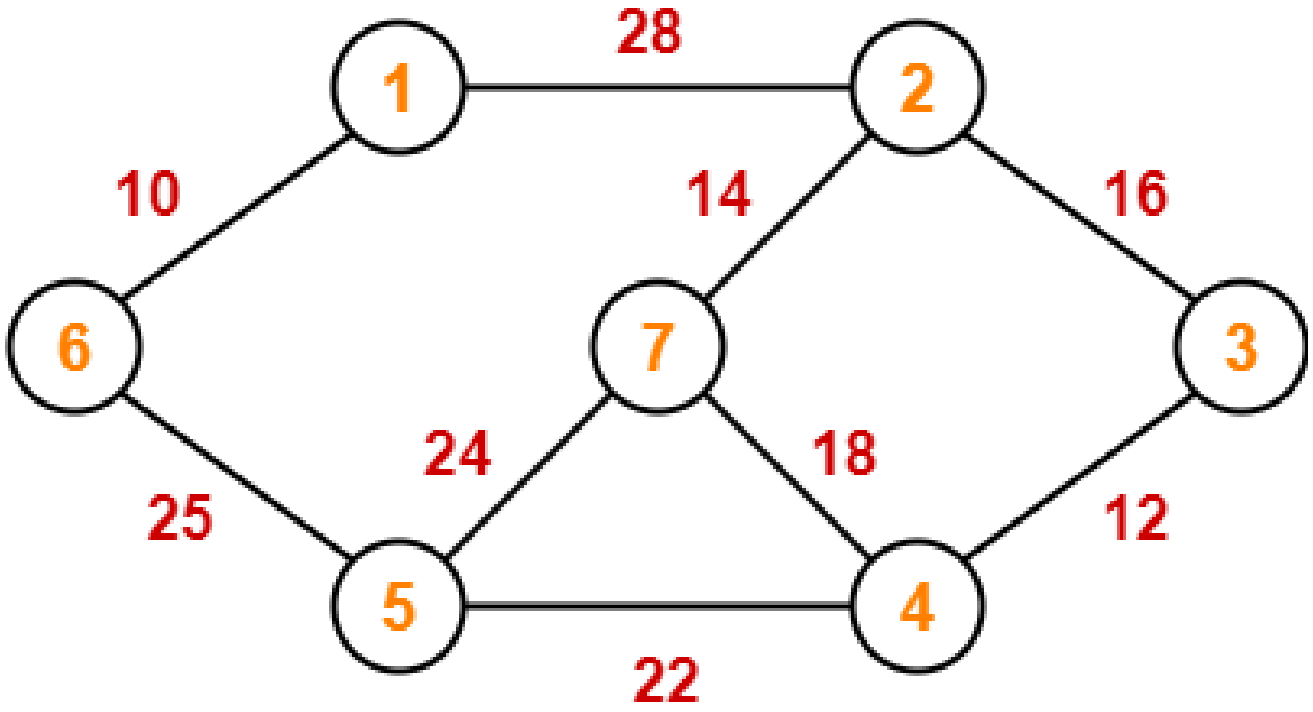
ALGORITHM *Prim*(G)

//Prim's algorithm for constructing a minimum spanning tree
//Input: A weighted connected graph $G = \langle V, E \rangle$
//Output: E_T, the set of edges composing a minimum spanning tree of G
$V_T \leftarrow \{v_0\}$ //the set of tree vertices can be initialized with any vertex
$E_T \leftarrow \varnothing$
for $i \leftarrow 1$ **to** $|V| - 1$ **do**
    find a minimum-weight edge $e^* = (v^*, u^*)$ among all the edges (v, u)
    such that v is in V_T and u is in $V - V_T$
    $V_T \leftarrow V_T \cup \{u^*\}$
    $E_T \leftarrow E_T \cup \{e^*\}$
return E_T

Minimum Spanning Tree (MST)

Prim's Algorithm

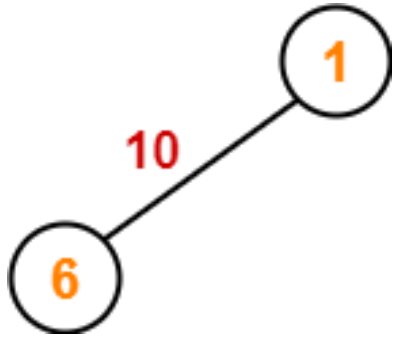
Problem: Construct the minimum spanning tree (MST) for the given graph using Prim's Algorithm-



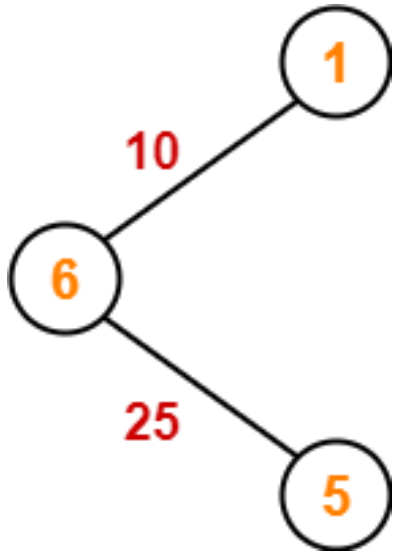
Minimum Spanning Tree (MST)

Prim's Algorithm

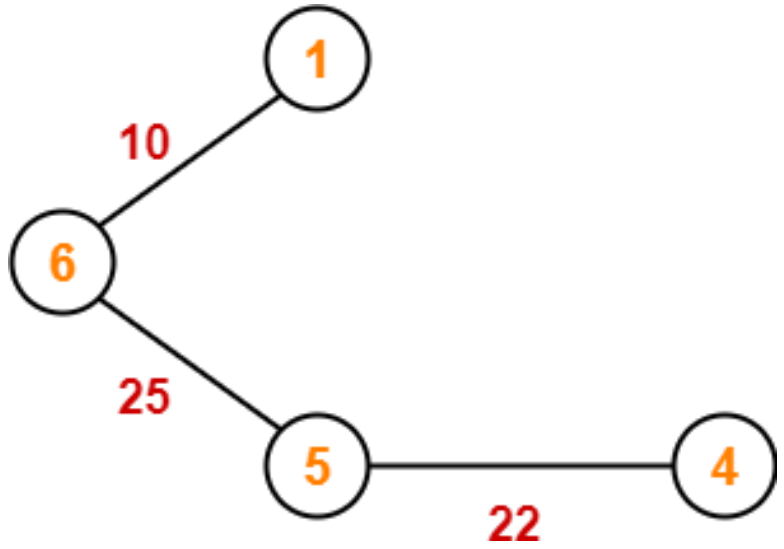
Step-01:



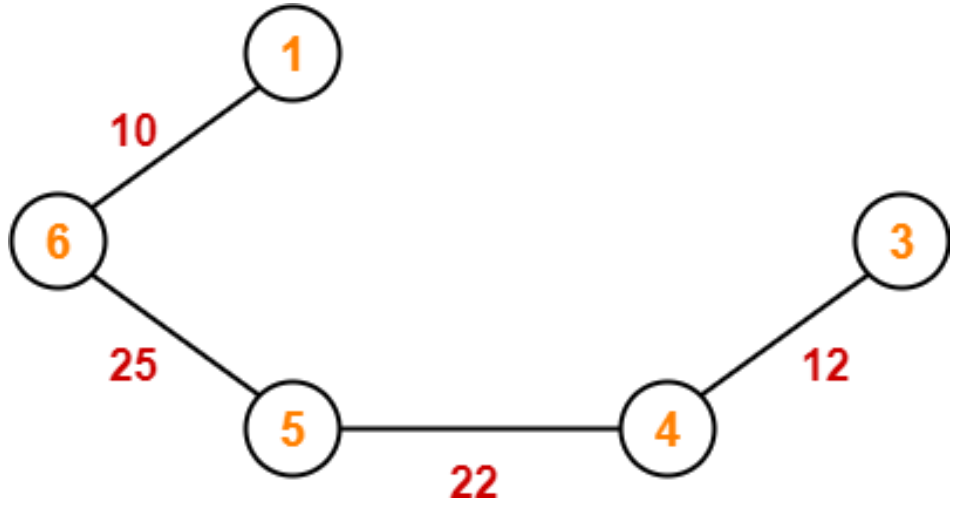
Step-02:



Step-03:



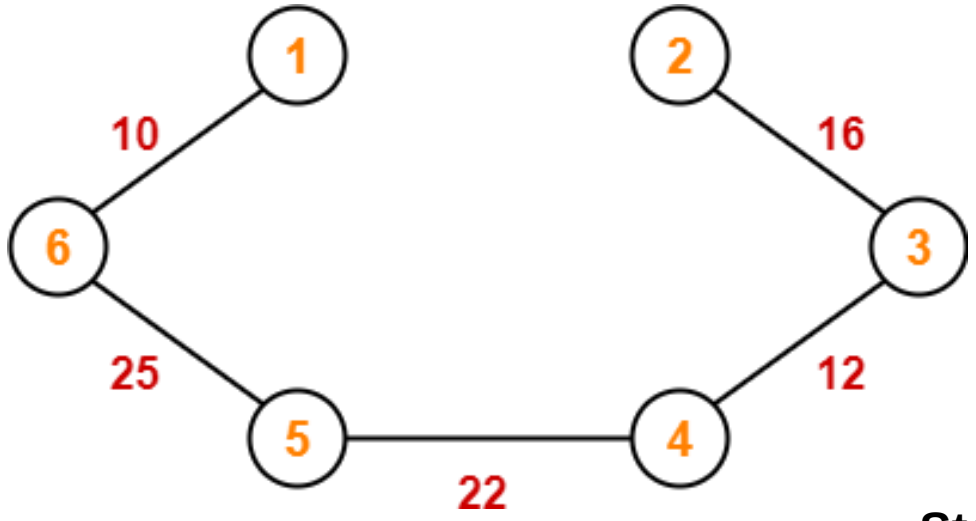
Step-04:



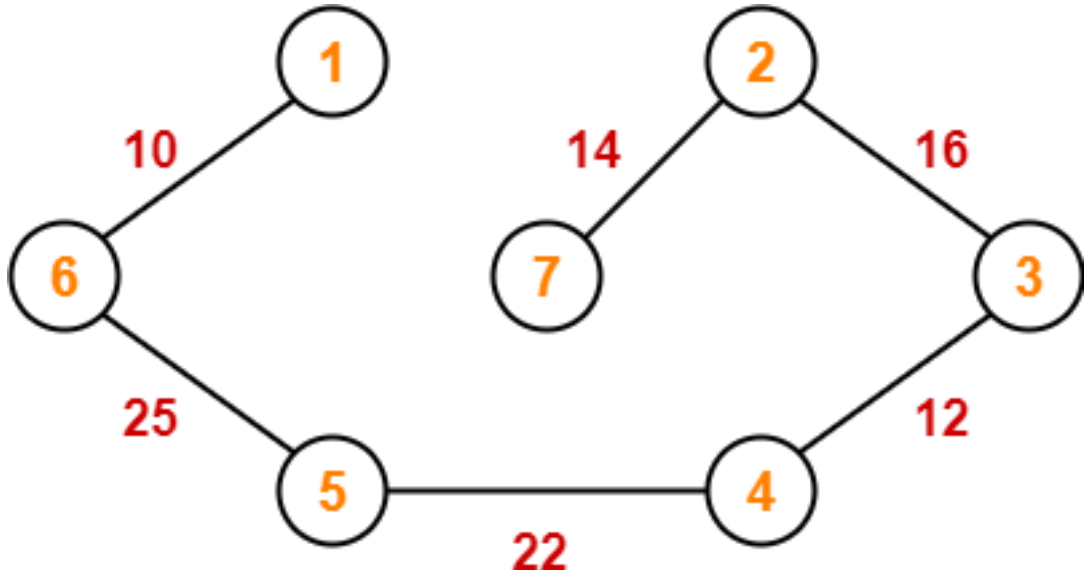
Minimum Spanning Tree (MST)

Prim's Algorithm

Step-05:



Step-06:

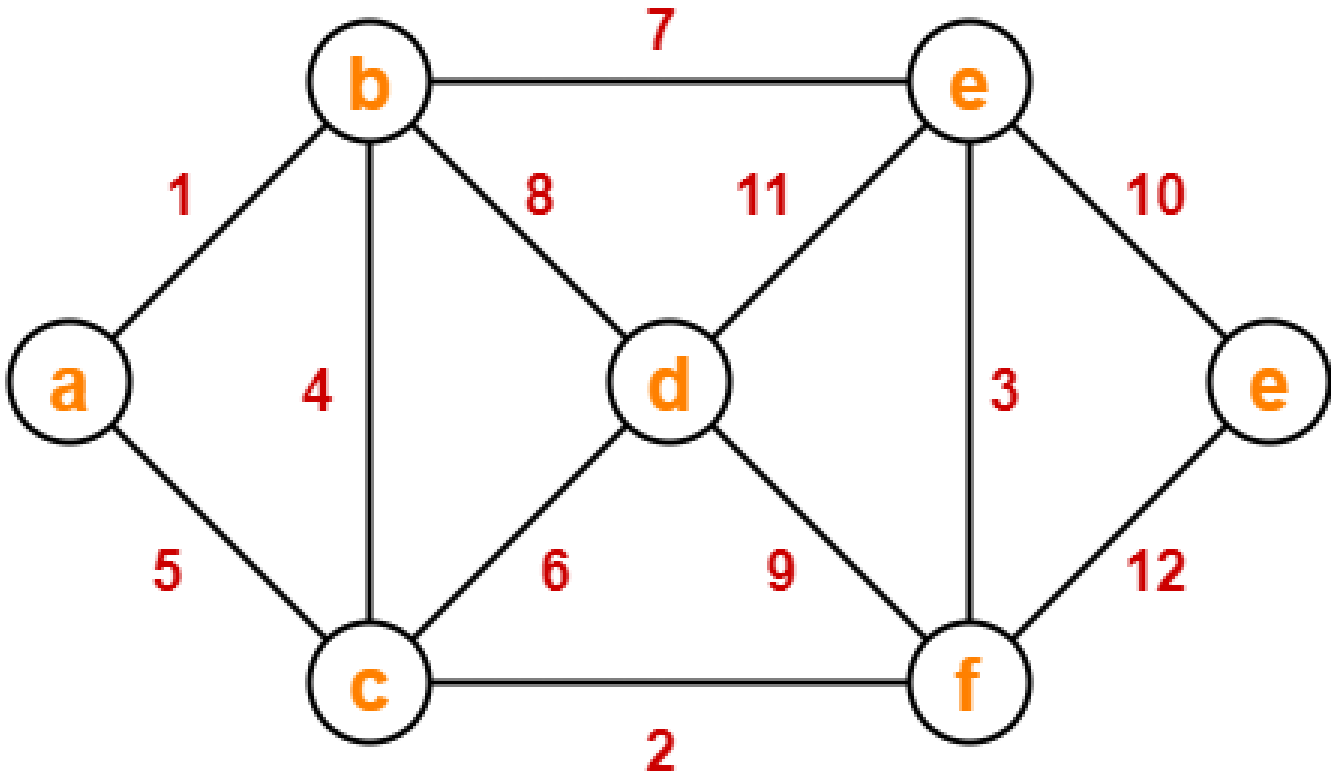


Cost of Minimum Spanning Tree= Sum of all edge weights= 10 + 25 + 22 + 12 + 16 + 14= 99 units

Minimum Spanning Tree (MST)

Prim's Algorithm

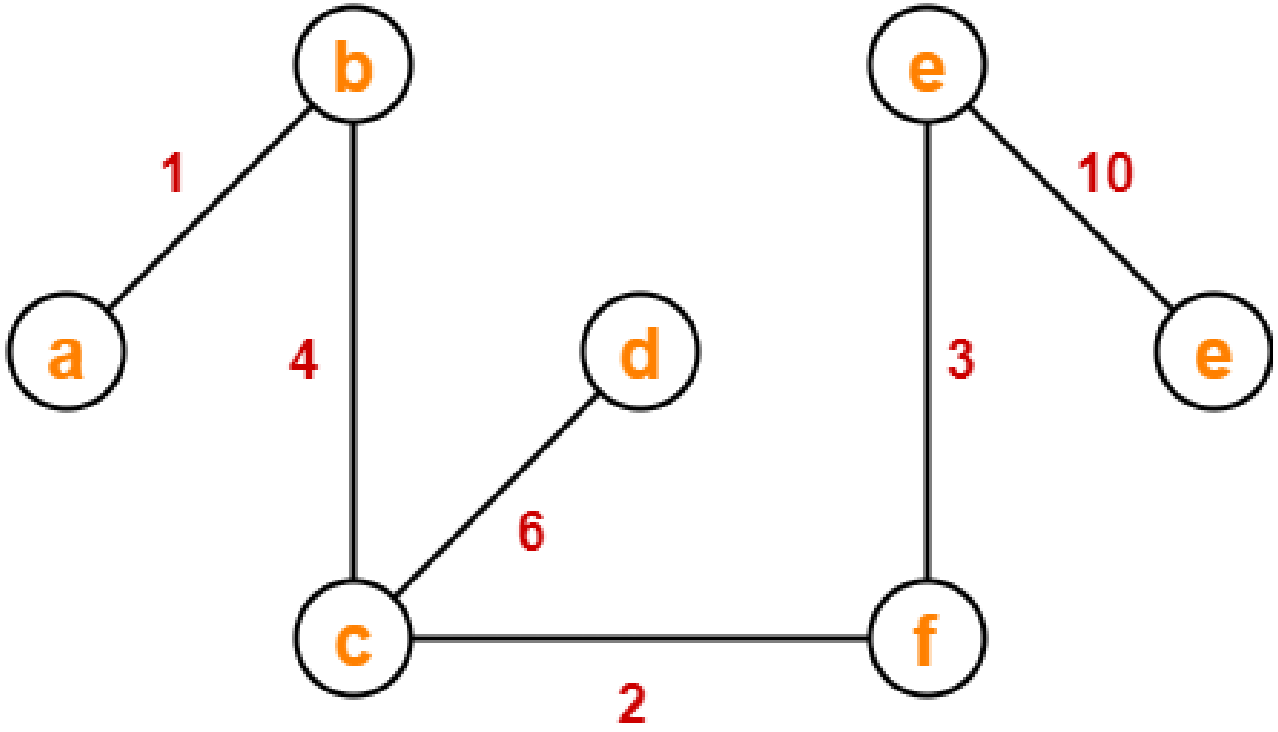
Problem: Construct the minimum spanning tree (MST) for the given graph using Prim's Algorithm-



Minimum Spanning Tree (MST)

Prim's Algorithm

Solution



Cost of Minimum Spanning Tree = Sum of all edge weights = 1 + 4 + 2 + 6 + 3 + 10 = 26 units

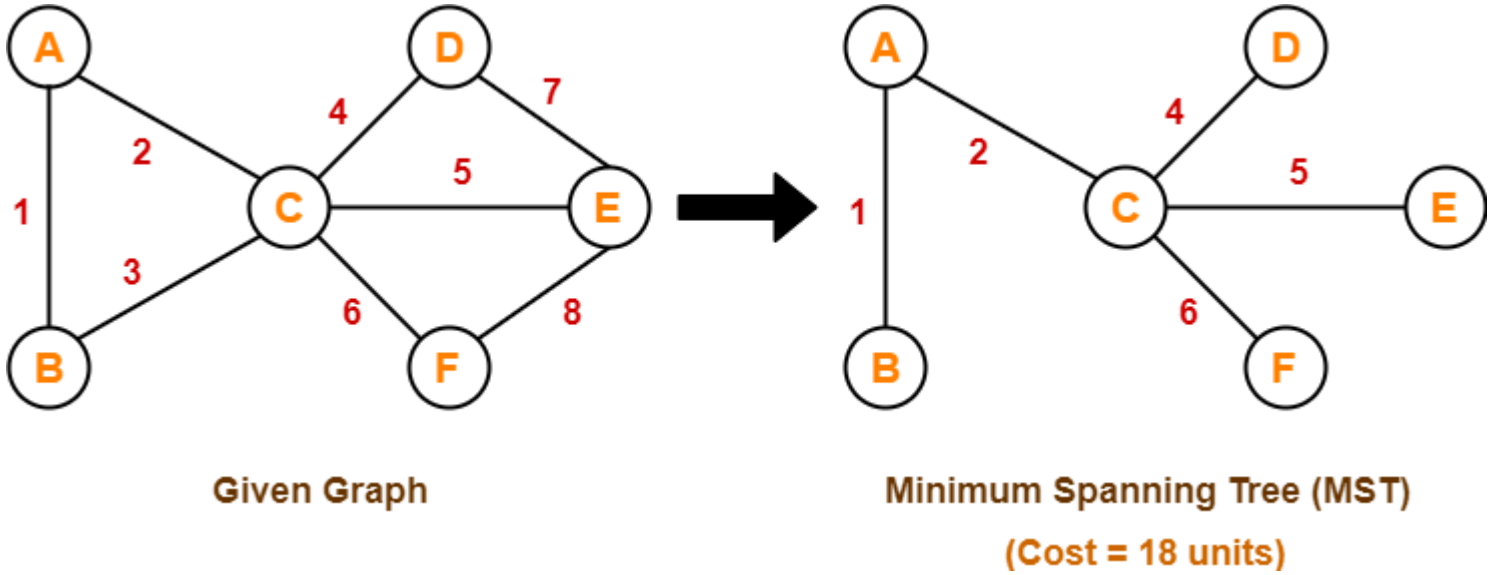
Difference Between Prim's and Kruskal's Algorithm

Important concepts

Concept-01:

If all the edge weights are distinct, then both the algorithms are guaranteed to find the same MST.

Example:



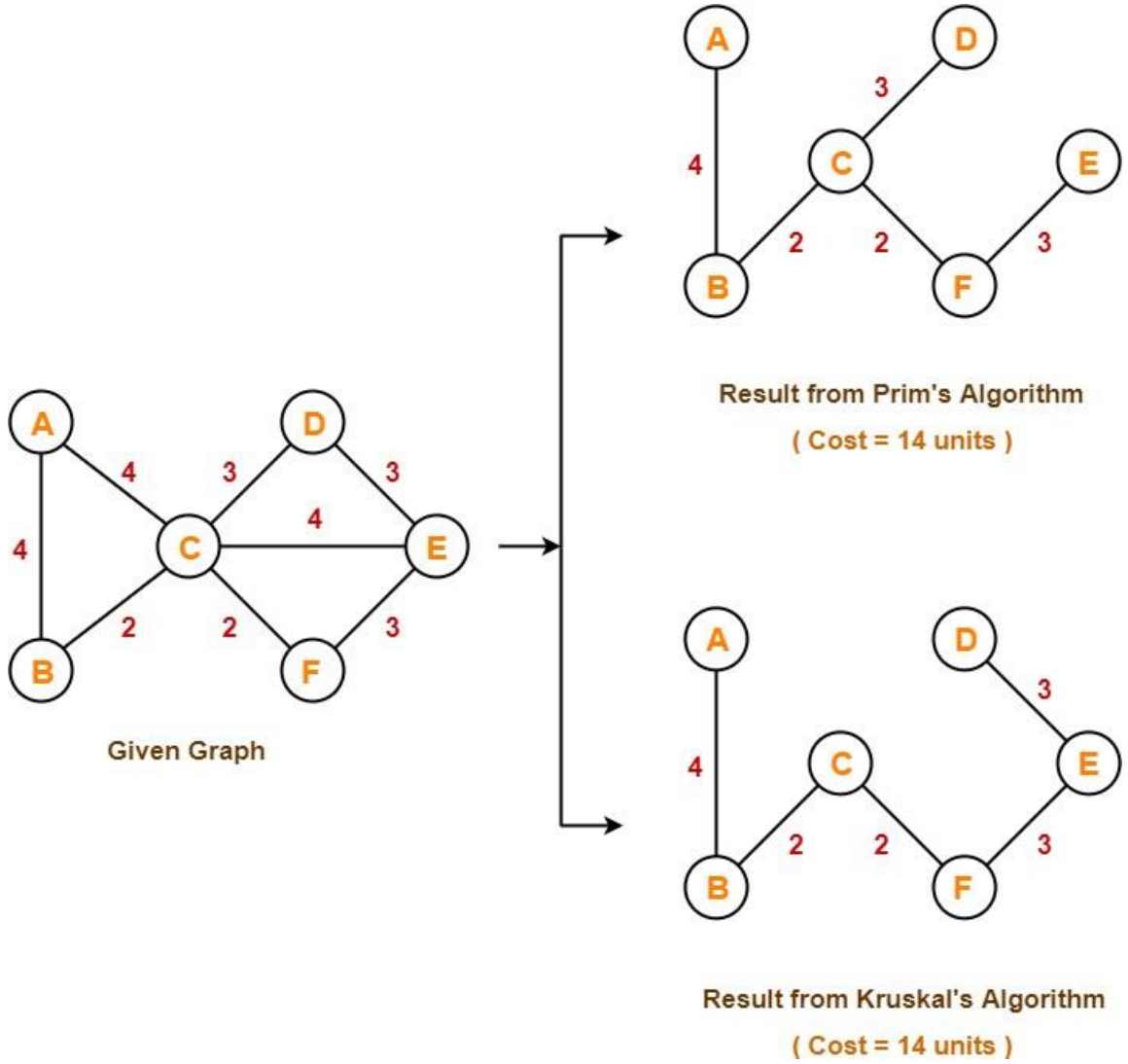
Here, both the algorithms on the above given graph produces the same MST

Difference Between Prim's and Kruskal's Algorithm

Important concepts

Concept-02:

- If all the edge weights are not distinct, then both the algorithms may not always produce the same MST.
- However, cost of both the MSTs would always be same in both the cases.



Difference Between Prim's and Kruskal's Algorithm

Important concepts

Concept-03:

Kruskal's Algorithm is preferred when-

- The graph is sparse.
- There are less number of edges in the graph like $E = O(V)$
- The edges are already sorted or can be sorted in linear time.

Prim's Algorithm is preferred when-

- The graph is dense.
- There are large number of edges in the graph like $E = O(V^2)$.

Difference Between Prim's and Kruskal's Algorithm

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
Prim's Algorithm is faster for dense graphs.	Kruskal's Algorithm is faster for sparse graphs.
Prim's algorithm has a time complexity of $O(V^2)$, V being the number of vertices and can be improved up to O(E log V) using Fibonacci heaps.	Kruskal's algorithm's time complexity is O(E log V), V being the number of vertices.

Any Questions ?

Dr. Anand Singh Jalal
Professor
Email: asjalal@gla.ac.in