**Graph Data Stucture**

A graph data structure is a collection of nodes that have data and are connected to other nodes.

Let's try to understand this through an example. On facebook, everything is a node. That includes User, Photo, Album, Event, Group, Page, Comment, Story, Video, Link, Note...anything that has data is a node.

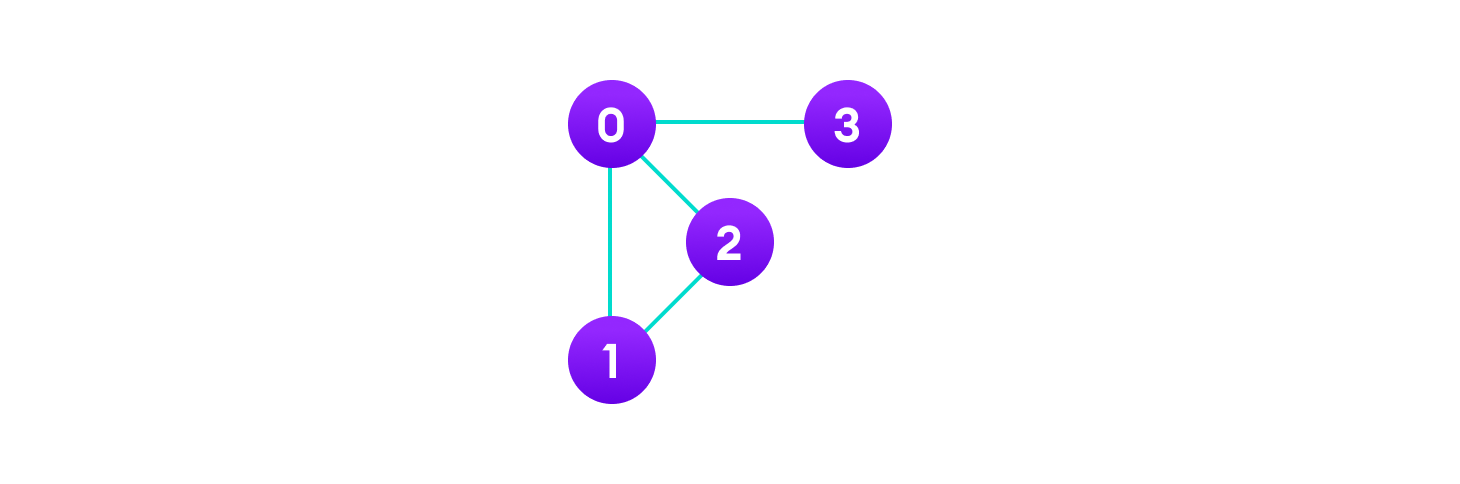
Every relationship is an edge from one node to another. Whether you post a photo, join a group, like a page, etc., a new edge is created for that relationship.

Example of graph data structure

All of facebook is then a collection of these nodes and edges. This is because facebook uses a graph data structure to store its data.

More precisely, a graph is a data structure (V, E) that consists of

* A collection of vertices V
* A collection of edges E, represented as ordered pairs of vertices (u,v)

Vertices and edges

In the graph,

V = {0, 1, 2, 3}

E = {(0,1), (0,2), (0,3), (1,2)}

G = {V, E}

**Graph Terminology**

* **Adjacency**: A vertex is said to be adjacent to another vertex if there is an edge connecting them. Vertices 2 and 3 are not adjacent because there is no edge between them.
* **Path**: A sequence of edges that allows you to go from vertex A to vertex B is called a path. 0-1, 1-2 and 0-2 are paths from vertex 0 to vertex 2.
* **Directed Graph**: A graph in which an edge (u,v) doesn't necessarily mean that there is an edge (v, u) as well. The edges in such a graph are represented by arrows to show the direction of the edge.

**Graph Representation**

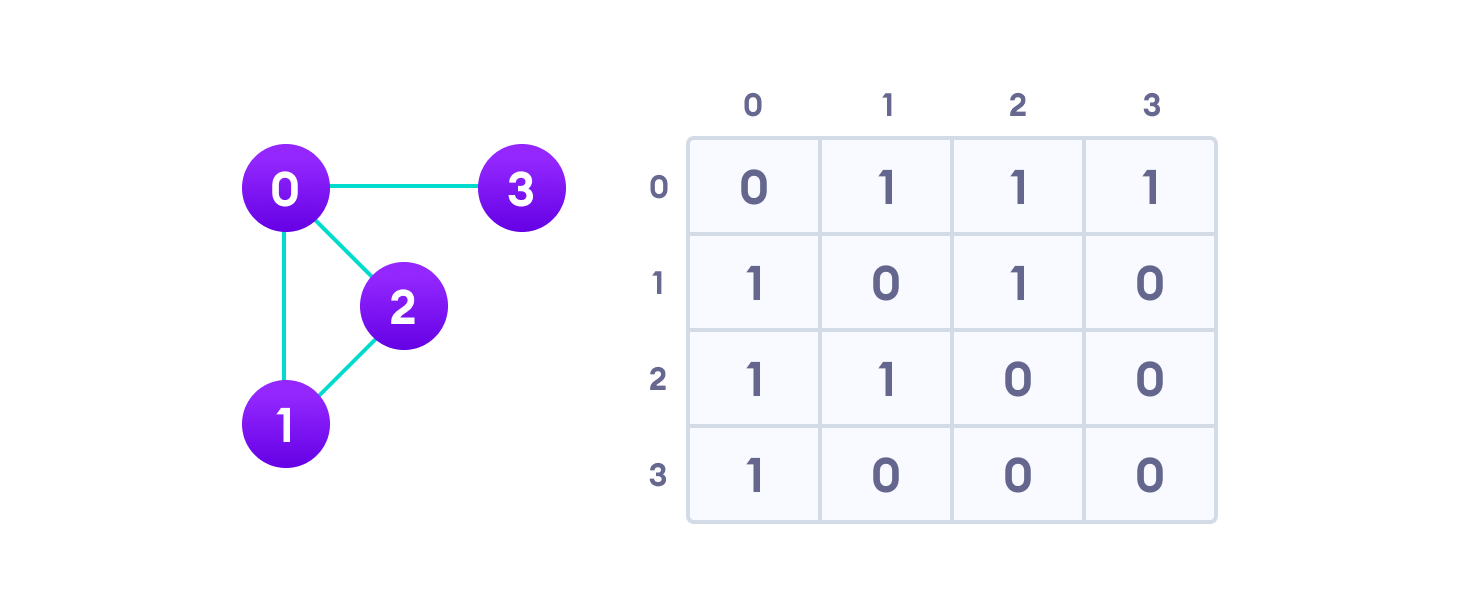
Graphs are commonly represented in two ways:

**1. Adjacency Matrix**

An adjacency matrix is a 2D array of V x V vertices. Each row and column represent a vertex.

If the value of any element a[i][j] is 1, it represents that there is an edge connecting vertex i and vertex j.

The adjacency matrix for the graph we created above is

Graph adjacency matrix

Since it is an undirected graph, for edge (0,2), we also need to mark edge (2,0); making the adjacency matrix symmetric about the diagonal.

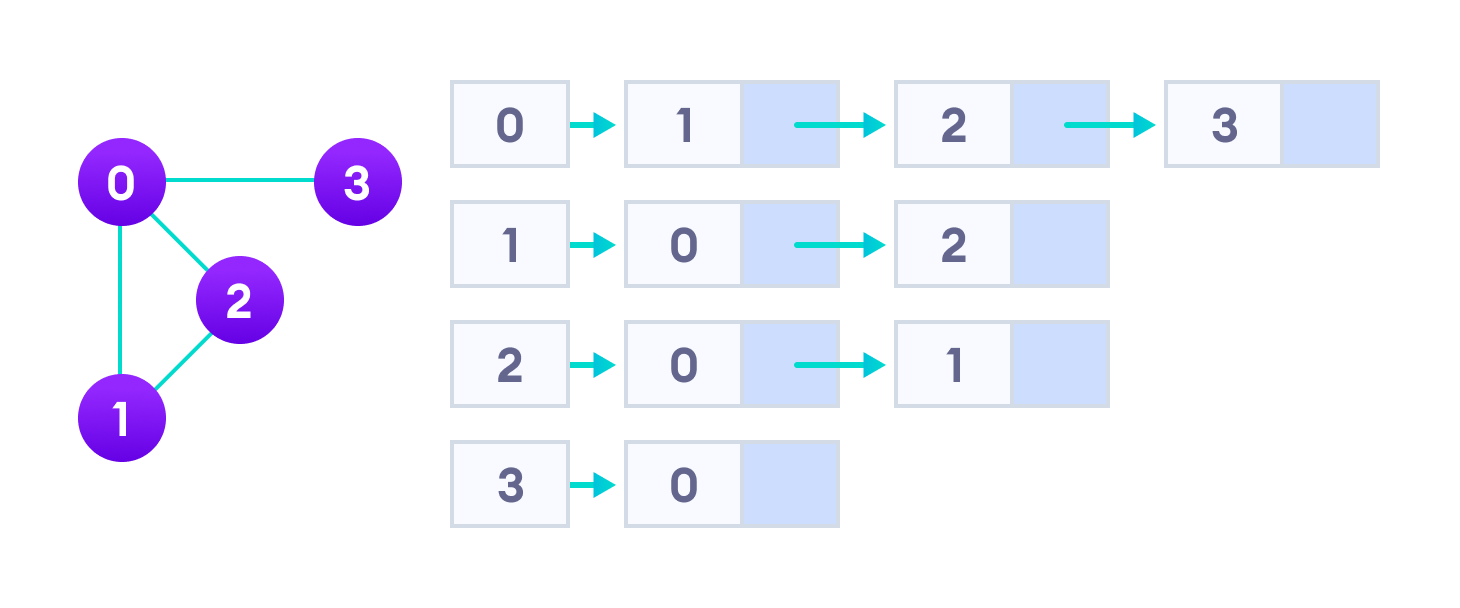
Edge lookup(checking if an edge exists between vertex A and vertex B) is extremely fast in adjacency matrix representation but we have to reserve space for every possible link between all vertices(V x V), so it requires more space.

**2. Adjacency List**

An adjacency list represents a graph as an array of linked lists.

The index of the array represents a vertex and each element in its linked list represents the other vertices that form an edge with the vertex.

The adjacency list for the graph we made in the first example is as follows:

Adjacency list representation

An adjacency list is efficient in terms of storage because we only need to store the values for the edges. For a graph with millions of vertices, this can mean a lot of saved space.

**Graph Operations**

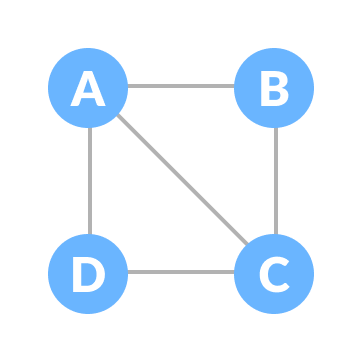
The most common graph operations are:

* Check if the element is present in the graph
* Graph Traversal
* Add elements(vertex, edges) to graph
* Finding the path from one vertex to another

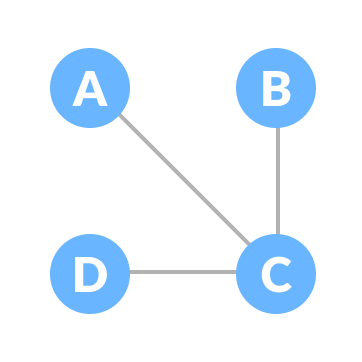
**Spanning Tree and Minimum Spanning Tree**

Before we learn about spanning trees, we need to understand two graphs: undirected graphs and connected graphs.

An **undirected graph** is a graph in which the edges do not point in any direction (ie. the edges are bidirectional).

Undirected Graph

A **connected graph** is a graph in which there is always a path from a vertex to any other vertex.

Connected Graph

**Spanning tree**

A spanning tree is a sub-graph of an undirected connected graph, which includes all the vertices of the graph with a minimum possible number of edges. If a vertex is missed, then it is not a spanning tree.

The edges may or may not have weights assigned to them.

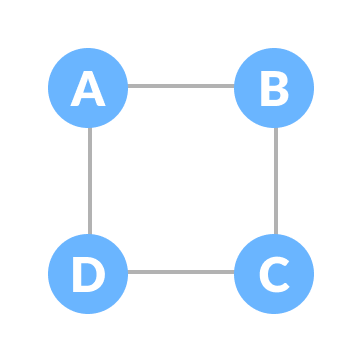
The total number of spanning trees with n vertices that can be created from a complete graph is equal to n(n-2).

If we have n = 4, the maximum number of possible spanning trees is equal to 44-2 = 16. Thus, 16 spanning trees can be formed from a complete graph with 4 vertices.

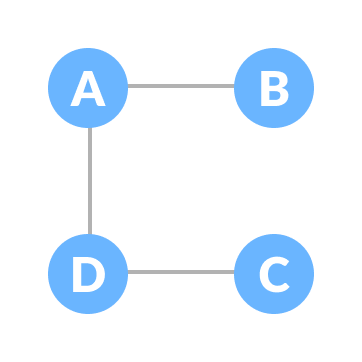
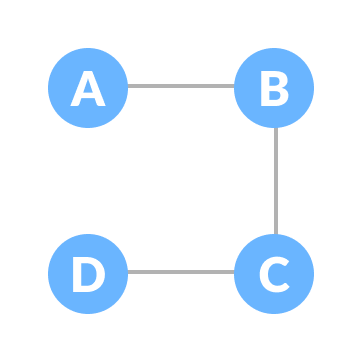
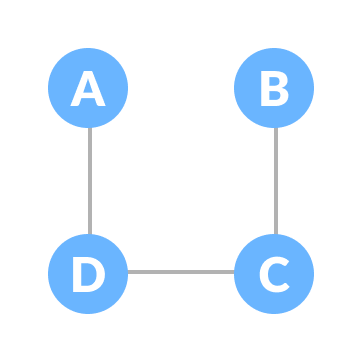
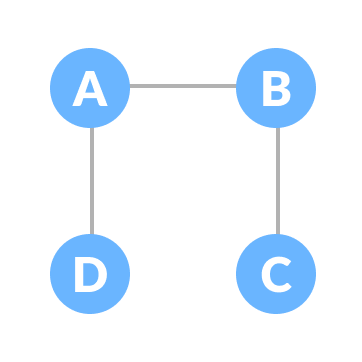
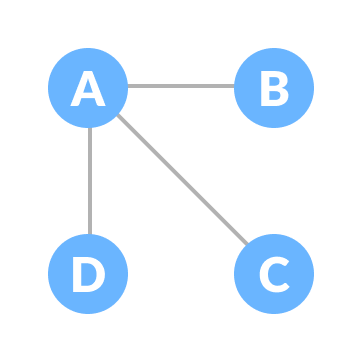
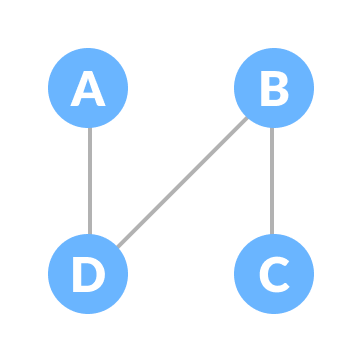
**Example of a Spanning Tree**

Let's understand the spanning tree with examples below:

Let the original graph be:

Normal graph

Some of the possible spanning trees that can be created from the above graph are:

A spanning treeA spanning treeA spanning treeA spanning treeA spanning treeA spanning tree

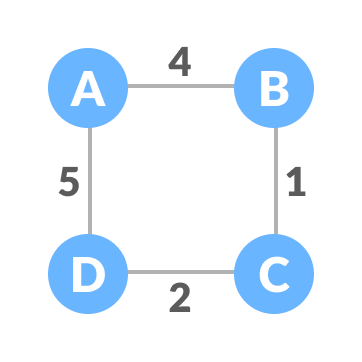
**Minimum Spanning Tree**

A minimum spanning tree is a spanning tree in which the sum of the weight of the edges is as minimum as possible.

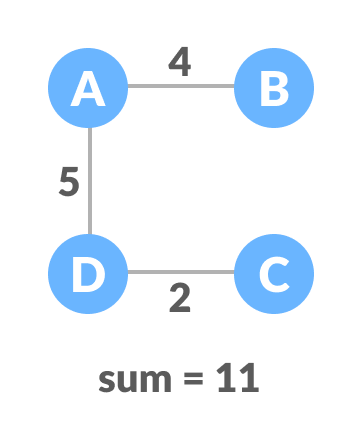
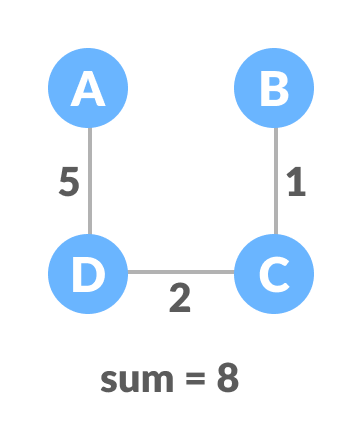
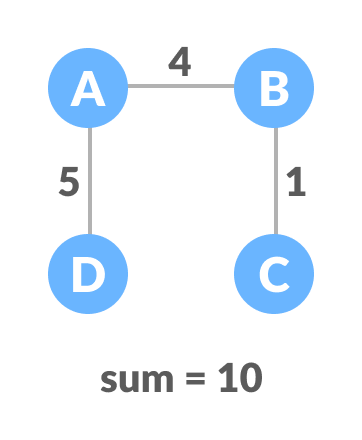
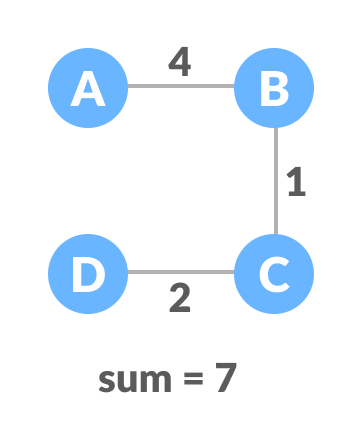
**Example of a Spanning Tree**

Let's understand the above definition with the help of the example below.

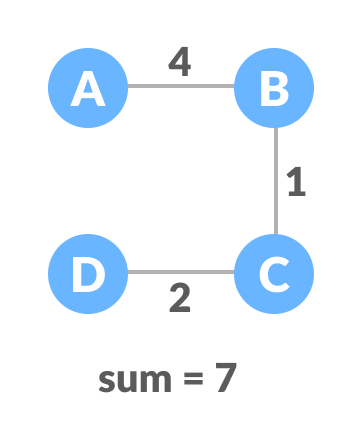
The initial graph is:

Weighted graph

The possible spanning trees from the above graph are:

Minimum spanning tree - 1Minimum spanning tree - 2Minimum spanning tree - 3Minimum spanning tree - 4

The minimum spanning tree from the above spanning trees is:

Minimum spanning tree

The minimum spanning tree from a graph is found using the following algorithms:

1. [Prim's Algorithm](https://www.programiz.com/dsa/prim-algorithm)
2. [Kruskal's Algorithm](https://www.programiz.com/dsa/kruskal-algorithm)

**Spanning Tree Applications**

* Computer Network Routing Protocol
* Cluster Analysis
* Civil Network Planning

**Minimum Spanning tree Applications**

* To find paths in the map
* To design networks like telecommunication networks, water supply networks, and electrical grids.

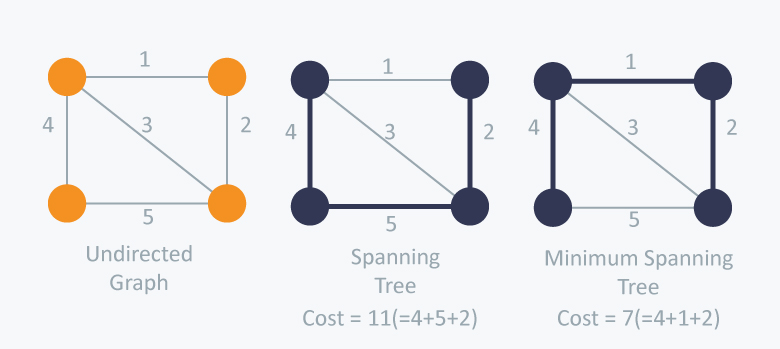
# Minimum Spanning Tree

What is a Minimum Spanning Tree?

The cost of the spanning tree is the sum of the weights of all the edges in the tree. There can be many spanning trees. Minimum spanning tree is the spanning tree where the cost is minimum among all the spanning trees. There also can be many minimum spanning trees.

Minimum spanning tree has direct application in the design of networks. It is used in algorithms approximating the travelling salesman problem, multi-terminal minimum cut problem and minimum-cost weighted perfect matching. Other practical applications are:

1. Cluster Analysis
2. Handwriting recognition
3. Image segmentation



There are two famous algorithms for finding the Minimum Spanning Tree:

Kruskal’s Algorithm

Kruskal’s Algorithm builds the spanning tree by adding edges one by one into a growing spanning tree. Kruskal's algorithm follows greedy approach as in each iteration it finds an edge which has least weight and add it to the growing spanning tree.

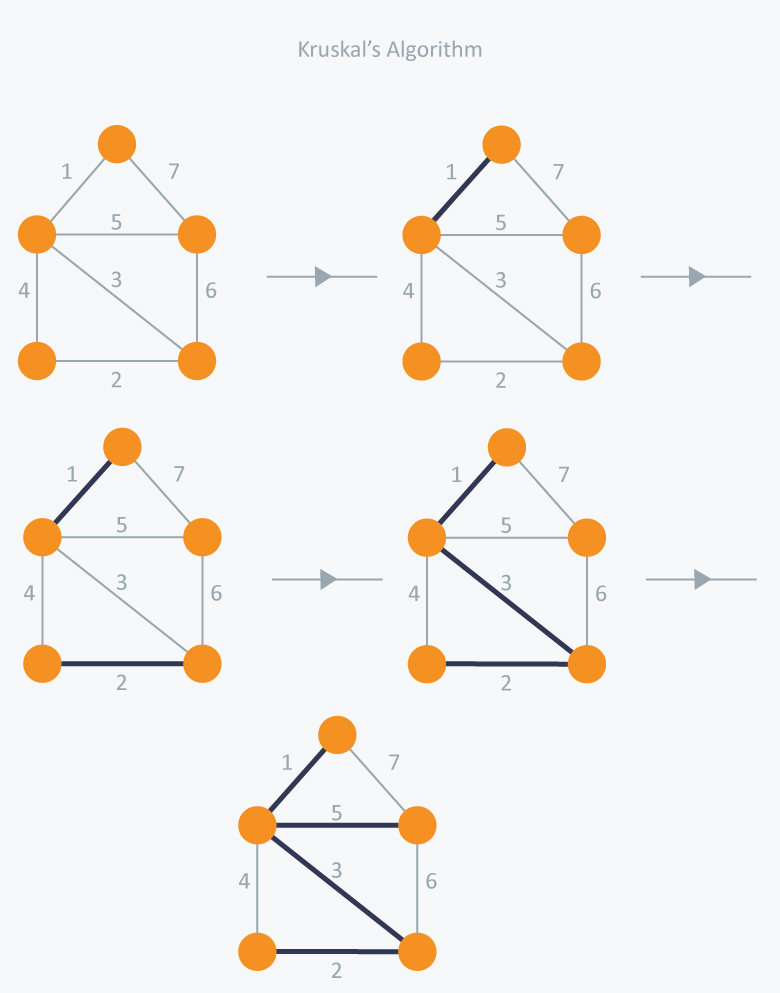
**Algorithm Steps:**

* 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.

So now the question is how to check if 2 vertices are connected or not ?

This could be done using DFS which starts from the first vertex, then check if the second vertex is visited or not. But DFS will make time complexity large as it has an order of �(�+�) where � is the number of vertices, � is the number of edges. So the best solution is **"Disjoint Sets":**  
Disjoint sets are sets whose intersection is the empty set so it means that they don't have any element in common.

Consider following example:



In Kruskal’s algorithm, at each iteration we will select the edge with the lowest weight. So, we will start with the lowest weighted edge first i.e., the edges with weight 1. After that we will select the second lowest weighted edge i.e., edge with weight 2. Notice these two edges are totally disjoint. Now, the next edge will be the third lowest weighted edge i.e., edge with weight 3, which connects the two disjoint pieces of the graph. Now, we are not allowed to pick the edge with weight 4, that will create a cycle and we can’t have any cycles. So we will select the fifth lowest weighted edge i.e., edge with weight 5. Now the other two edges will create cycles so we will ignore them. In the end, we end up with a minimum spanning tree with total cost 11 ( = 1 + 2 + 3 + 5).

**Time Complexity:**  
In Kruskal’s algorithm, most time consuming operation is sorting because the total complexity of the Disjoint-Set operations will be O(E logV), which is the overall Time Complexity of the algorithm.

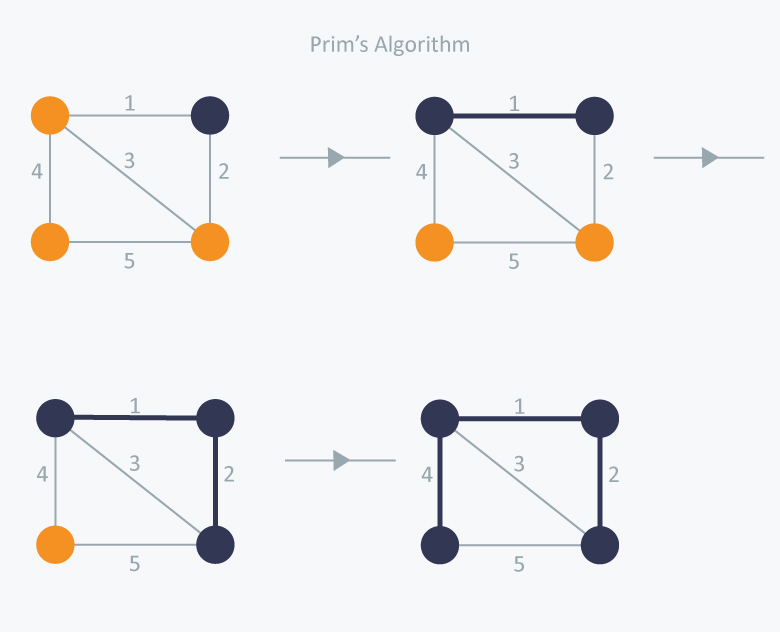
Prim’s Algorithm

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.
* Check for cycles. To do that, mark the nodes which have been already selected and insert only those nodes in the Priority Queue that are not marked.

Consider the example below:



In Prim’s Algorithm, we will start with an arbitrary node (it doesn’t matter which one) and mark it. In each iteration we will mark a new vertex that is adjacent to the one that we have already marked. As a greedy algorithm, Prim’s algorithm will select the cheapest edge and mark the vertex. So we will simply choose the edge with weight 1. In the next iteration we have three options, edges with weight 2, 3 and 4. So, we will select the edge with weight 2 and mark the vertex. Now again we have three options, edges with weight 3, 4 and 5. But we can’t choose edge with weight 3 as it is creating a cycle. So we will select the edge with weight 4 and we end up with the minimum spanning tree of total cost 7 ( = 1 + 2 +4).

**Time Complexity:**  
The time complexity of the Prim’s Algorithm is O(V + E) logV because each edge is inserted in the priority queue only once and insertion in priority queue take logarithmic time.