DSA **Assignment 04**
Deadline: 10/05/2020

Detailed description of Prim's and Kruskal's algorithms with examples is given below. Read it carefully and by using your **graph** class implementations (Adjacency List + Adjacency Matrix), implement the following methods:

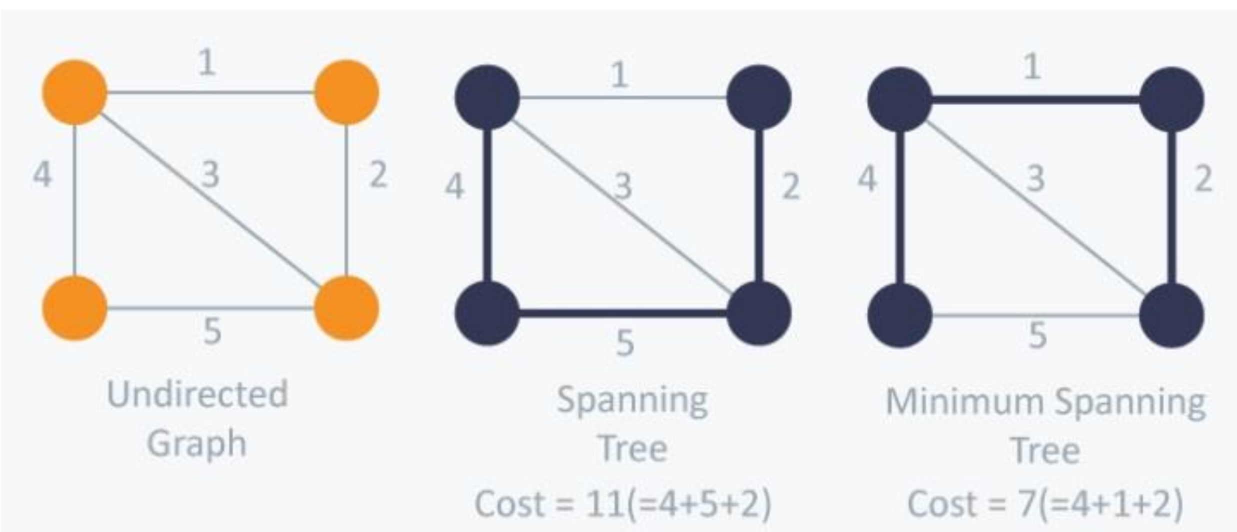
1. Prim(startingNode) // using Adjacency Matrix
2. Prim(startingNode) // using Adjacency List
3. Kruskal Algorithm // using Adjacency Matrix
4. Kruskal Algorithm // using Adjacency List

Spanning Tree

Given an undirected and connected graph G = (V, E), a spanning tree of the graph G is a tree that spans G (that is, it includes every vertex of G) and is a **subgraph of G** (every edge in the tree belongs to G)

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.



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:

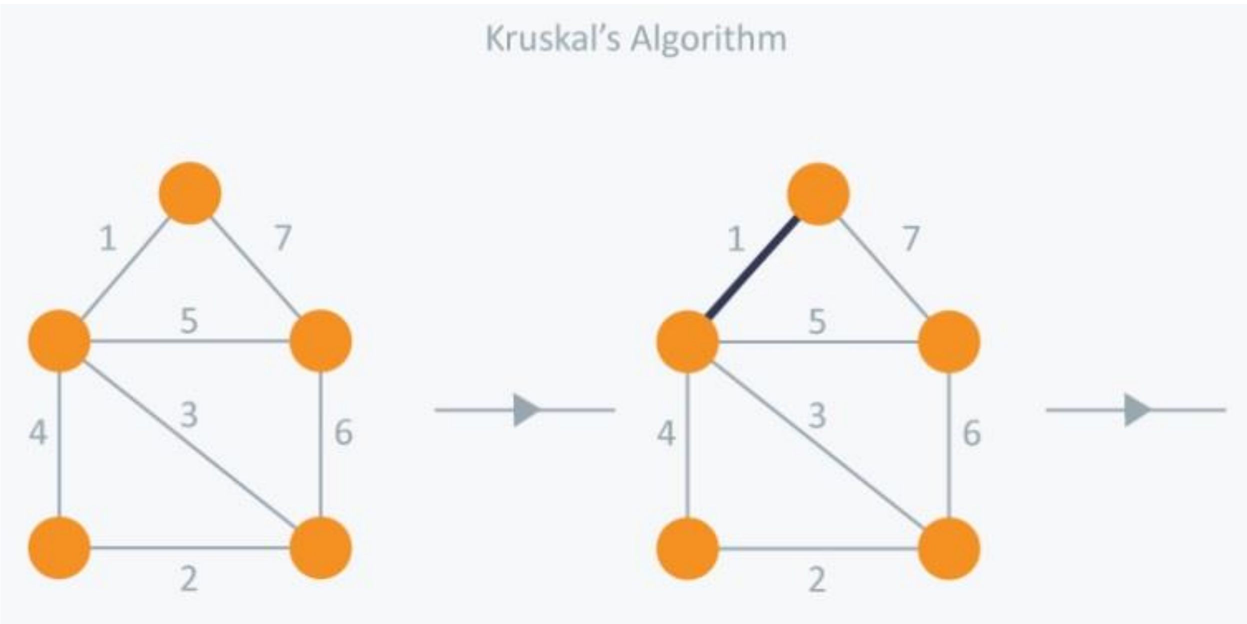
- Sort the graph edges with respect to their weights.
- Start adding edges to the MST from the edge with the smallest weight until the edge of the largest weight.
- Only add edges which doesn't form a cycle, edges which connect only disconnected components.

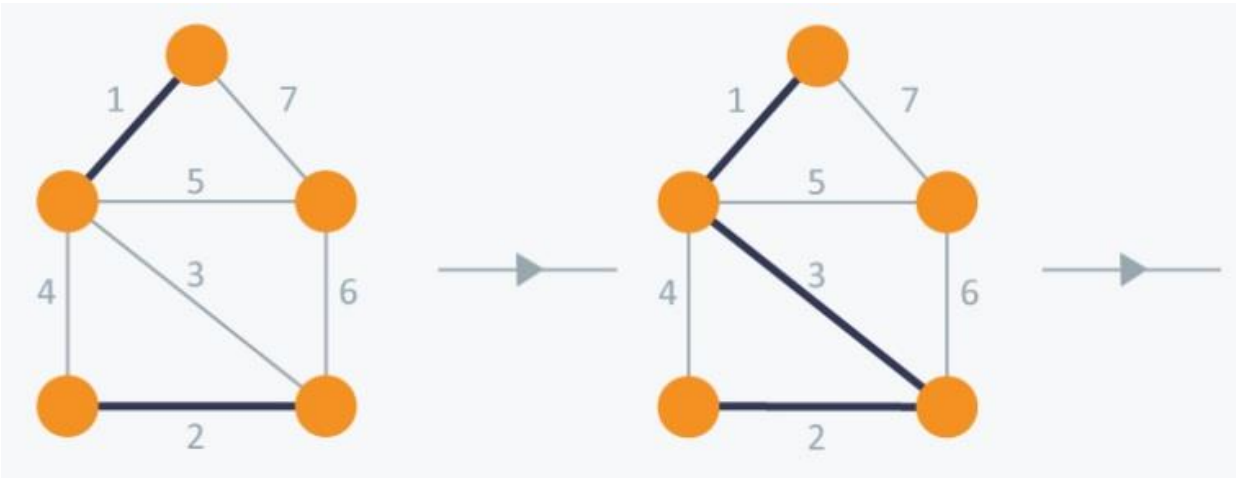
So now the question is how to check if 2 vertices are connected or not?

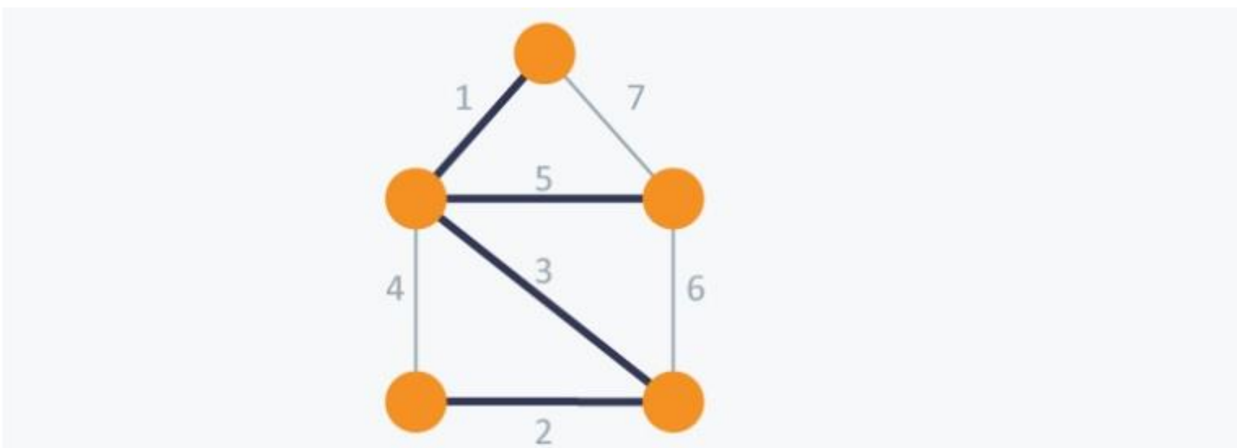
This could be done using DFS which starts from the first vertex, then check if the second vertex is visited or not. But DFS will make time complexity large as it has an order of O(V+E) where V is the number of vertices, E is the number of edges. So the best solution is **"Disjoint Sets":**

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).

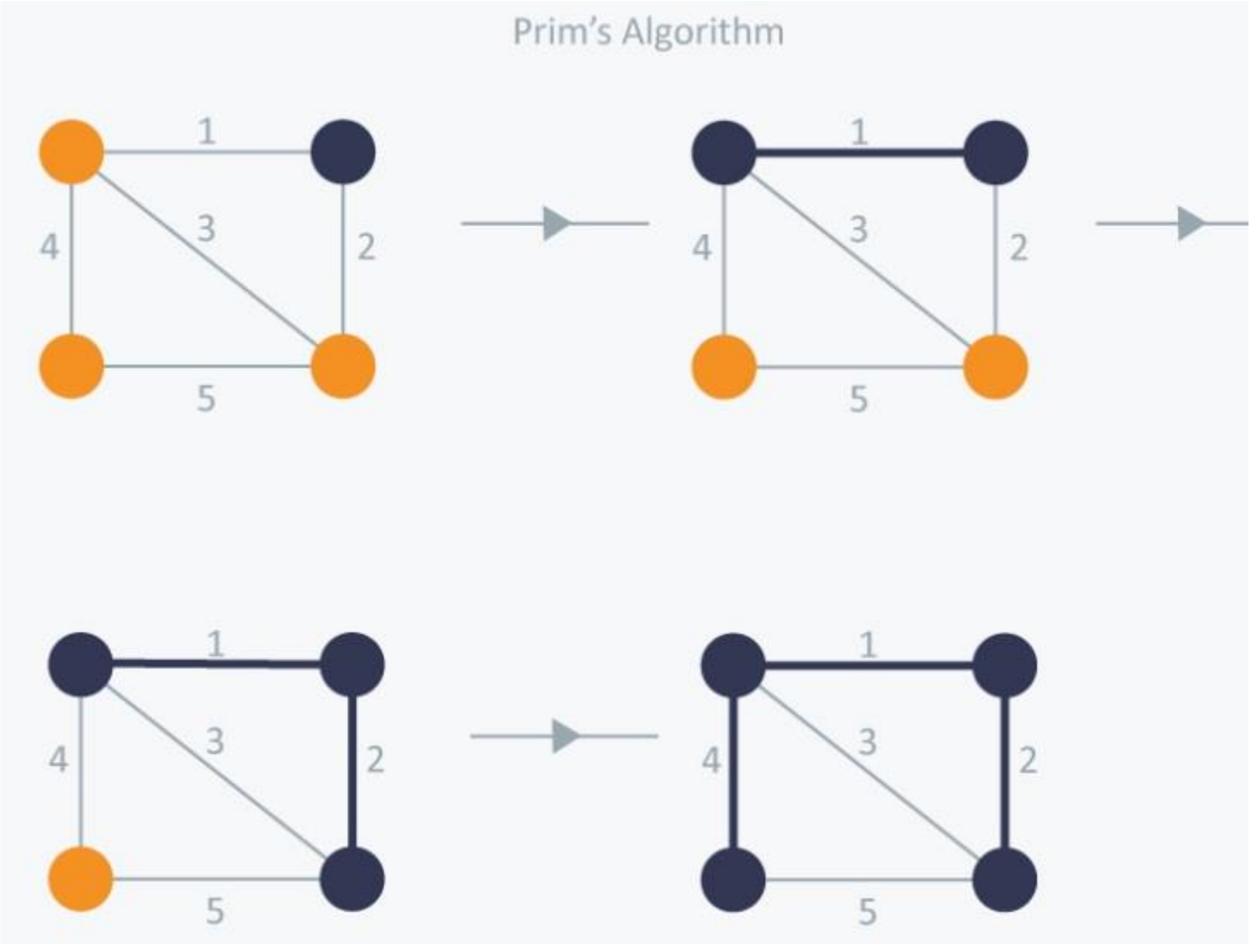
Prim’s Algorithm

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).