PRIM'S ALGORITHM TO FIND MINIMUM SPANNING TREE

1. It is used to find Minimum Spanning Tree in the Graph and also it only works on Vertex
2. We uses MinHeap to implement it
3. Start by Selecting Any Random Vertex and Check all the adjacent vertex and push them in MinHeap and keep updating the visited array.
4. When the MinHeap is empty then finally the MST is achieved and also we can maintain the MST Edges in another data structures

```
class Pair implements Comparable<Pair>{
    int wt;
    int v;
    int parent;
    Pair(int wt, int v, int parent){
        this.wt = wt;
        this.v = v;
        this.parent = parent;
    }
    public int compareTo(Pair other){
        return this.wt - other.wt;
    }
}

class Solution {
    static int spanningTree(int V, int E, List<List<int[]>> adj) {
        int minCost = 0;
        boolean [] visited = new boolean[V];
        Arrays.fill(visited, false);
        PriorityQueue<Pair> queue = new PriorityQueue<>();
        queue.offer(new Pair(0,0, - 1)); // Start with 0 as root Node and wt = 0
        while(queue.size() > 0){
            Pair it = queue.poll();
            int destination = it.v;
            int wt = it.wt;
            int parent = it.parent;

            // this (parent, destination) is the vertex is to edge that can be added in
final MST if required

            if(visited[destination])continue;
            visited[destination] = true;
            minCost += wt;
            for(int i = 0; i < adj.get(destination).size(); i++){
                int [] adjacency = adj.get(destination).get(i);
                int v = adjacency[0];
                int weight = adjacency[1];
```

```
                if(!visited[v]){
                    queue.offer(new Pair(weight, v, destination));
                }
            }
        }
        return minCost;
    }}
```

Space Complexity = O(V + E)
Time Complexity = O(V + E) LOG V

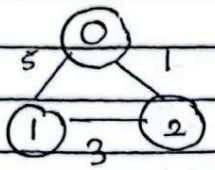
It is guaranteed that it always produces the best and minimum spanning tree for given pair of graph and also it can be used to produces edges by just Using Pair<WT, V, PARENT> Where <PARENT, V> is the pair of vertex in MST.

(i) (Create Priority Queue) (Min Heap)

(ii) Put All Vertex in ie (V, W) in PQ with Cost

< 0 → < [ ], [ ], [ ] >

0 → < {0, 1, 5}, {0, 2, 1}



[1, (0, 2)]
[3, (2, 1)] [5, (1, 0)]

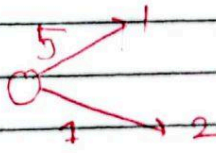
1	1	1
0	1	2

Algorithm

→ Select Any Arbitrary Index/Vertex As root (Usually We take 0)

1	1	1
0	1	2

(0, 0) - 1)
(Wt, V)



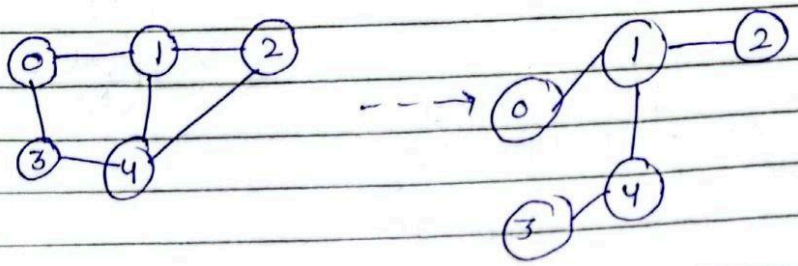
(1, 2)

(5, 1) (3, 2)

(1 + 3 = 4)

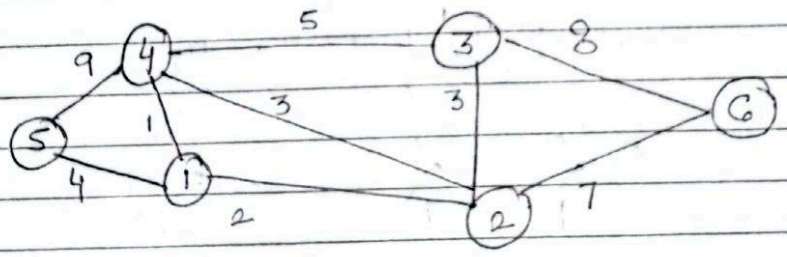
Minimum Spanning Tree and Disjoint Set in Graph

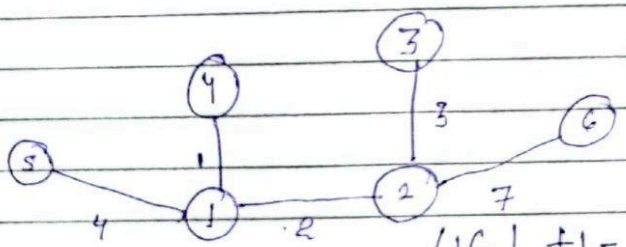
Spanning Tree has 'n' nodes (n-1) edges & All Are Connected



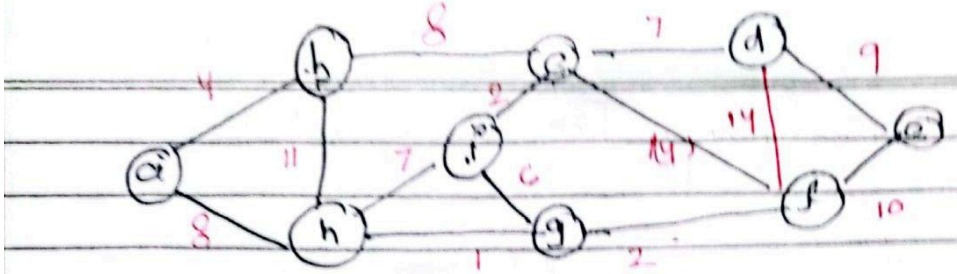
→ A Graph Can Have More than One Spanning Tree

Among All Spanning Tree a Graph With Minimum Edge Cost is Minimum Spanning Tree





(16) +1 = 17



(4, a, b)

(8, a, h) (8, b, c)

(11, b, h) (2, c, i)

(2, c, i)

(4, a, b) (8, b, h)

(7, c, d)

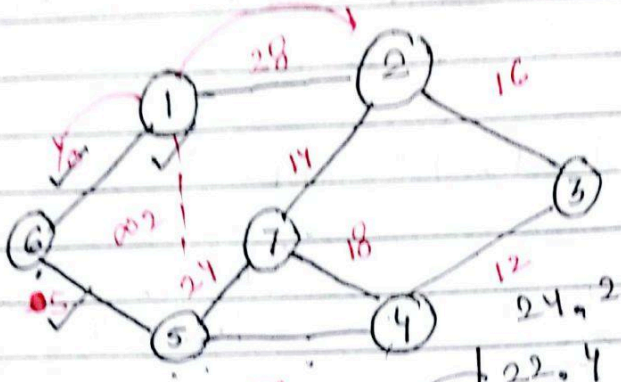
(11, b, h) (8, a, h)

(2, c, i)

(4, a, b) (7, c, d)

(1, b, h) (8, a, h) (8, b, h)

Example Question



22, 4
(5, 5)
(28, 2)
(10, 6)
(0, 1)

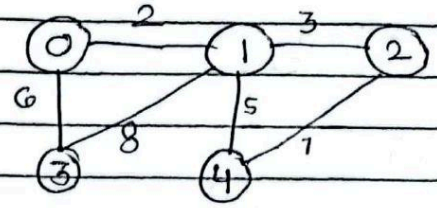
1	1	1	1	1	1	1
1	2	3	4	5	6	7

10
5
2-2
12
16
14
79

Minimum Spanning Tree

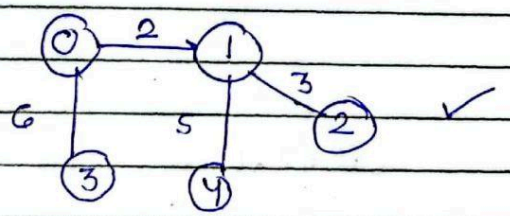
- Prims
- Kruskal

(1) Prim's Algorithm for MST (Works On Vertex)



[[0,1),2] (1) ✓
[(1,2),3] (2) ✓
[(0,3),6] (4) ✓
[(3,1),8] (6)
[(1,4),5] (3) ✓
[(2,4),7] (5)

{0,1}



✓

{0,1,2,3,4} if(Set Size == n) then We get Our MST

Data Structure To be Used

1) Min Heap
2) Visited Array
3) MST Array