Index - 8

Graphs 2

1) Dijkstra’s Algorithm - Single Source Shortest Path

i. In every iteration find the min node. Which has time complexity of n.
T.C = O(n^2)
S.C = O(n)

ii. Below we are using min heap so it only takes logn time.

```
class Pair implements Comparator<Pair>{
   int node;
   int dist;
   Pair(int n, int d){
      node = n;
      dist = d;
   }
   Pair(){

   }
    public int compare(Pair o1,Pair o2){
      if(o1.dist<o2.dist)
         return -1;
      else if(o1.dist>o2.dist)
         return 1;

      return 0;
   }
}
class Solution
{

   static int[] dijkstra(int v, ArrayList<ArrayList<ArrayList<Integer>>> adj, int s)
   {
      PriorityQueue<Pair> pq = new PriorityQueue<Pair>(v, new Pair());
      int distance[] = new int[v];
      Arrays.fill(distance,Integer.MAX_VALUE);
      distance[s] = 0;
      pq.add(new Pair(s,0));

      while(!pq.isEmpty()){
         Pair cur = pq.poll();
         int n = cur.node;
         int d = cur.dist;
```

```
            for(ArrayList<Integer> it:adj.get(n)){
                int adjNode = it.get(0);
                int weight = it.get(1);
                if(distance[n]+weight<distance[adjNode]){
                    distance[adjNode] = distance[n]+weight;
                    pq.add(new Pair(adjNode, distance[adjNode]));
                }
            }
        }

        return distance;
    }
```

T.C = O(nlogn)
S.C = O(n)

2) Prims Algorithm - Minimum Spanning Tree

```
class Node{
    int node;
    int weight;
    Node(int n, int w){
        node = n;
        weight = w;
    }
}
class Solution
{
    static int spanningTree(int v, ArrayList<ArrayList<ArrayList<Integer>>> adj)
    {

        int distance[] = new int[v];
        boolean inMST[] = new boolean[v];
        Arrays.fill(distance,Integer.MAX_VALUE);
        PriorityQueue<Node> pq = new PriorityQueue<>((o1,o2)-> o1.weight-o2.weight);
        pq.add(new Node(0,0));
        distance[0] = 0;

        while(!pq.isEmpty()){
            Node cur = pq.poll();
            int u = cur.node;
            inMST[u] = true;

            for(ArrayList<Integer> it : adj.get(u)){
                int adjNode = it.get(0);
                int weight = it.get(1);
                if(!inMST[adjNode] && distance[adjNode]>weight){
                    distance[adjNode] = weight;
                    pq.add(new Node(adjNode, weight));
                }
            }
        }
        int ans = 0;
        for(int i=0;i<v;i++)
            ans+=distance[i];

        return ans;
    }
```

T.C = O(nlogn)
S.C = O(n)