Name: Khushi Nitinkumar Patel

PRN: 2020BTECS00037

Batch: T5

Assignment no 7

Greedy method

Implement Kruskal’s algorithm & Prim's algorithm to find Minimum Spanning Tree (*MST*) of the given an undirected, connected and weighted graph.

Kruskal’s Algorithm:

Algorithm:

1. Sort all the edges according to increasing order of weights
2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.
3. To check the cycle check if both the nodes forming edge have same ultimate parents. If yes do not consider it in MST else include it.
4. Repeat step 2 & 3 until there are (V-1) edges in the spanning tree

Code:

```
#include<bits/stdc++.h>
using namespace std;

class DisjointSet{
    vector<int> rank,parent;
public:
    // constructor
    // initializes array of rank with 0 and parent with itself
    DisjointSet(int v){
        rank.resize(v+1,0);
        parent.resize(v+1);
        for(int i=0;i<=v;i++){
            parent[i]=i;
        }
    }

    // function to find the ultimate parent of node takes constant
time
    int findUparent(int node){
        if(node==parent[node]){
            return node;
        }
        return parent[node]=findUparent(parent[node]);
    }

    // Function to find Union
    void findUnionByRank(int u,int v){
        // find the ultimate parents of nodes
        int ultimateP_u=findUparent(u);
        int ultimateP_v=findUparent(v);

        // if same parents than return
        if(ultimateP_u==ultimateP_v){
            return;
        }

        // To connect the nodes check the rank
        // node with smaller rank is connected to node with larger
rank
        // if ranks are same then connect any one
        if(rank[ultimateP_u] < rank[ultimateP_v]){
            parent[ultimateP_u]=ultimateP_v;
        }else if(rank[ultimateP_u] > rank[ultimateP_v]){
            parent[ultimateP_v]=ultimateP_u;
        }else{
            parent[ultimateP_u]=ultimateP_v;
            rank[ultimateP_v]++;
        }
    }
};

class Graph{
    vector<vector<int>> edges;
    int V;
public:

    Graph(int V){
        this->V=V;
    }

    void addEdge(int u,int v,int w){
        edges.push_back({w,u,v});
    }

    void KruskalsMST(){
        // Sort the edges
        sort(edges.begin(),edges.end());

        DisjointSet ds(V);
        int cost=0;

        cout<<"The edges present in MST are: "<<endl;

        for(auto it: edges){
            int weight=it[0];
            int node1=it[1];
            int node2=it[2];

            // if Ultimate parents are not same means the nodes are
not in
            // same component and edge between them do not form
cycle
```

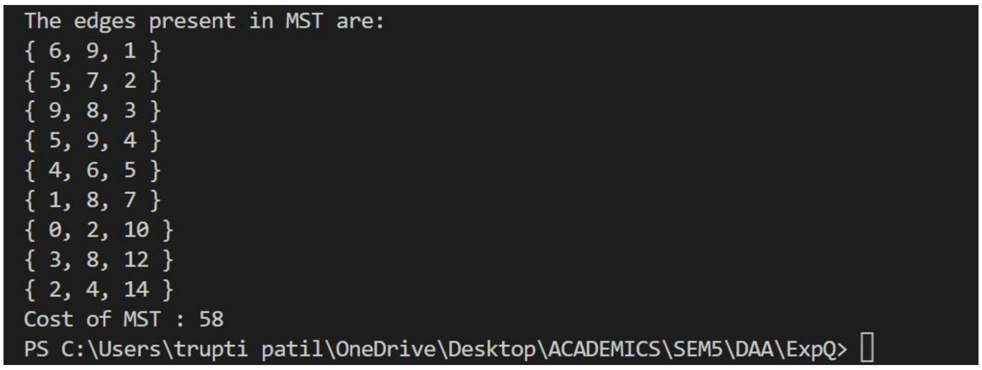
```cpp
            // include these nodes in MST
            if(ds.findUparent(node1)!=ds.findUparent(node2)){
                ds.findUnionByRank(node1,node2);
                cost+=weight;

                cout<<"{ "<<node1<<", "<<node2<<", "<<weight<<"
}"<<endl;
            }
        }

        cout<<"Cost of MST : "<<cost<<endl;
    }
};

int main(){
    Graph g(10);

    g.addEdge(0, 1, 15);
    g.addEdge(0, 8, 19);
    g.addEdge(0, 2, 10);
    g.addEdge(1, 8, 7);
    g.addEdge(1, 3, 17);
    g.addEdge(2, 8, 16);
    g.addEdge(2, 4, 14);
    g.addEdge(3, 1, 17);
    g.addEdge(3, 8, 12);
    g.addEdge(3, 9, 20);
    g.addEdge(3, 5, 13);
    g.addEdge(4, 2, 14);
    g.addEdge(4, 8, 6);
    g.addEdge(4, 9, 9);
    g.addEdge(4, 6, 5);
    g.addEdge(5, 3, 13);
    g.addEdge(5, 9, 4);
    g.addEdge(5, 7, 2);
    g.addEdge(7, 6, 18);
    g.addEdge(6, 9, 1);
    g.addEdge(7, 9, 11);
    g.addEdge(9, 8, 3);

    g.KruskalsMST();
}
```

Output:

Complexity Analysis:

Time Complexity: O(ElogE) or O(ElogV)

Space Complexity: O(V+E)



```
The edges present in MST are:
{ 6, 9, 1 }
{ 5, 7, 2 }
{ 9, 8, 3 }
{ 5, 9, 4 }
{ 4, 6, 5 }
{ 1, 8, 7 }
{ 0, 2, 10 }
{ 3, 8, 12 }
{ 2, 4, 14 }
Cost of MST : 58
PS C:\Users\trupti patil\OneDrive\Desktop\ACADEMICS\SEM5\DAA\ExpQ>
```

Prim's Algorithm:

Algorithm:

1) Initialize keys of all vertices as infinite and parent of every vertex as -1.

2) Create an empty priority queue pq. Every item of pq is a pair (weight, vertex). Weight (or key) is used as first item of pair as first item is by default used to compare two pairs.

3) Initialize all vertices as not part of MST yet. We use int array visit[] for this purpose. This array is required to make sure that an already considered vertex is not included in pq again. This is where Prim's implementation differs from Dijkstra In Dijkstra's algorithm, we didn't need this array as distances always increase. We require this array here because key value of a processed vertex may decrease if not checked.

4) Insert source vertex into pq and make its key as 0.

```
5) While either pq doesn't become empty
   a) Extract minimum key vertex from pq.
      Let the extracted vertex be tempnode.
   b) Include tempnode in visit using visit [tempnode] = 1.
   c) Loop through all adjacent of tempnode and do following for every vertex
      node.
        // If weight of edge (tempnode,node) is smaller than
        // key of node and node is not already in MST
        If visit[node] = false && key[node] > weight(tempnode, node)
           (i) Update key of node, i.e., do
                 key[node] = weight(tempnode, node)
           (ii) Insert node into the pq
           (iii) parent[node] = tempnode
6) Print MST edges using parent array.
7) Find the cost of MST by looping the key array.
```

Code:

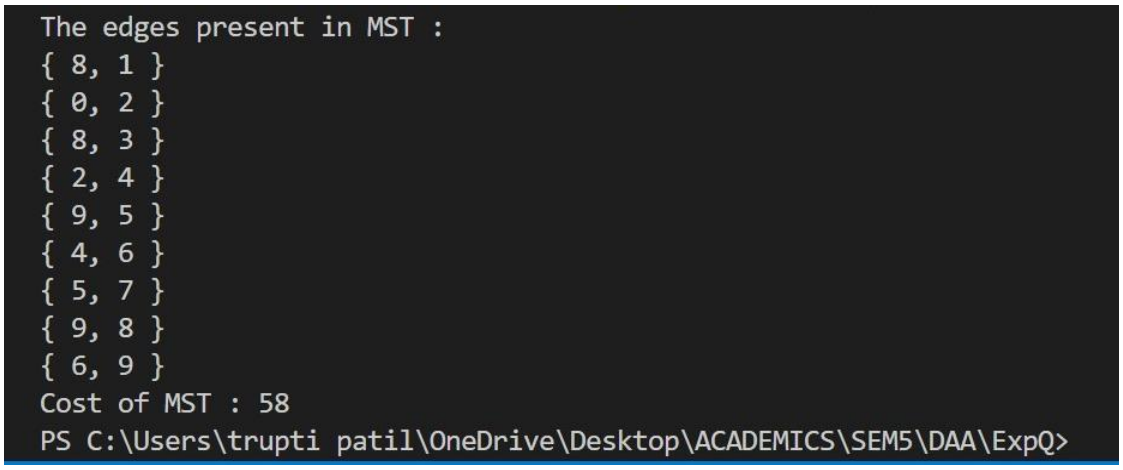
```cpp
#include<bits/stdc++.h>
using namespace std;

#define INF 0x3f3f3f3f

class Graph{
public:
    int v;
    list< pair<int, int> > *adj;

    Graph(int v){
        this->v=v;
        // array of type list<pair<int,int>> of size v
        adj=new list<pair<int,int>> [v];
    }

    void addEdge(int u,int v,int w){
        // in adjency list {node,weight}
        adj[u].push_back({v, w});
        adj[v].push_back({u, w});
    }

    void PrimsMST(){
        priority_queue<pair<int,int>,
        vector<pair<int,int>>,greater<pair<int,int>>> pq;

        int src=0;

        vector<int> key(v,INF); //stores the weight of edge at
particular index
        vector<int> parent(v,-1); //stores the parent node of
particular node determined by index
        vector<int> visit(v,0); //stores info or node is visited or
not

        // {weight,node} in priority queue
        pq.push({0,src});
        key[src]=0;


        while(!pq.empty()){
            int tempnode=pq.top().second;
            pq.pop();

            if(visit[tempnode]==1){
                continue;
            }

            visit[tempnode]=1;

            list<pair<int,int>> ::iterator itr;
            // traverse in adjency list
            for(itr=adj[tempnode].begin();itr!=adj[tempnode].end();i
tr++){
                // get the {node,weight} of adjacent node of present
node tempnode
                int node=(*itr).first; //node
                int weight=(*itr).second; //weight
                if(visit[node]==0 && key[node]>weight){
                    key[node]=weight;
                    pq.push({key[node],node});
                    parent[node]=tempnode;
                }
            }
        }

        // Edges present in MST
        cout<<"The edges present in MST : "<<endl;
        for(int i=1;i<v;i++){
            cout<<"{ "<<parent[i]<<", "<<i<<" }"<<endl;

        }

        // cost of MST
        int cost=0;
        for(int i=0;i<v;i++){
            cost+=key[i];
        }
        cout<<"Cost of MST : "<<cost<<endl;
    }
};

int main(){
    int v=10;
    Graph g(v);

    g.addEdge(0, 1, 15);
    g.addEdge(0, 8, 19);
    g.addEdge(0, 2, 10);
    g.addEdge(1, 8, 7);
    g.addEdge(1, 3, 17);
    g.addEdge(2, 8, 16);
    g.addEdge(2, 4, 14);
    g.addEdge(3, 1, 17);
    g.addEdge(3, 8, 12);
    g.addEdge(3, 9, 20);
    g.addEdge(3, 5, 13);
    g.addEdge(4, 2, 14);
    g.addEdge(4, 8, 6);
    g.addEdge(4, 9, 9);
    g.addEdge(4, 6, 5);
    g.addEdge(5, 3, 13);
    g.addEdge(5, 9, 4);
    g.addEdge(5, 7, 2);
    g.addEdge(7, 6, 18);
    g.addEdge(6, 9, 1);
    g.addEdge(7, 9, 11);
    g.addEdge(9, 8, 3);


    g.PrimsMST();

    return 0;
}
```

Output:

Complexity Analysis:

Time Complexity: O(E Log V))

where E is number of edges and V is number of vertice



```
The edges present in MST :
{ 8, 1 }
{ 0, 2 }
{ 8, 3 }
{ 2, 4 }
{ 9, 5 }
{ 4, 6 }
{ 5, 7 }
{ 9, 8 }
{ 6, 9 }
Cost of MST : 58
PS C:\Users\trupti patil\OneDrive\Desktop\ACADEMICS\SEM5\DAA\ExpQ>
```

Q) How many edges does a minimum spanning tree for above example?

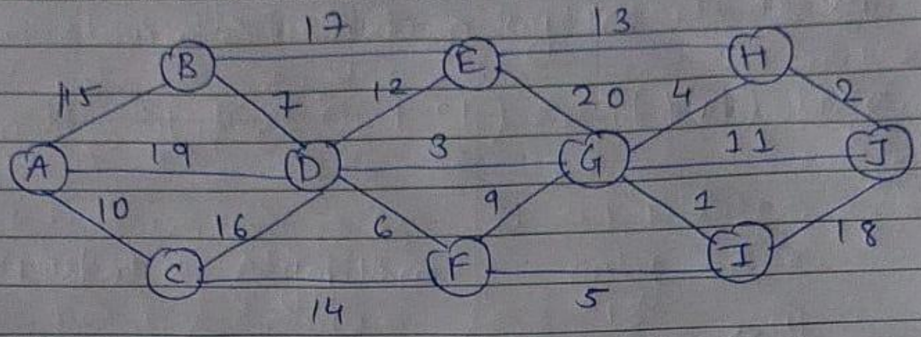
Q) In a graph *G*. let the edge *u v* have the least weight. is it true that *u v* is always part of any minimum spanning tree of *G*?.Justify your answers.

Q) Let *G* be a graph and T be a minimum spanning tree of *G*. Suppose that the weight of an edge e is decreased. How can you find the minimum spanning tree of the modified graph? What is the runtime of your solution?

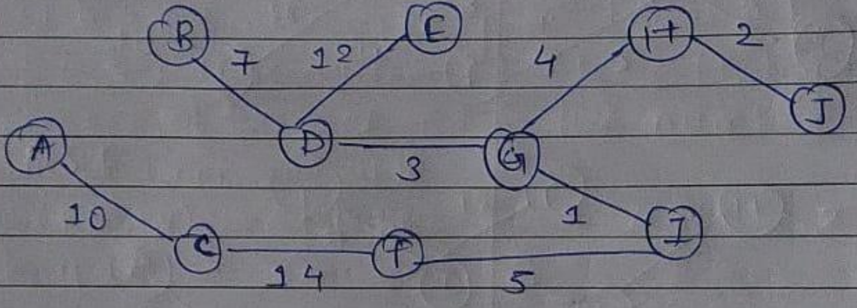
Q) Find order of edges for Kruskal's and Prim's?

PRN - 2020BTECS00037
Khushi N Patel
Assignment NO : 7.

① Prim's Algorithm



→ MST



Minimum cost = 10 + 14 + 5 + 1 + 3 + 7 + 12 + 4 + 2 = 58

$\boxed{\therefore \text{ Min cost } = 58}$

Q1. How many edges does a minimum spanning tree have in above example?

→ No. of edges $|E| = |V| - 1$.
$|E| = 10 - 1 \quad \because |V| = 10$
$\boxed{\therefore |E| = 9}$

Lucky Page No.:
Date: / /

Q2. In a graph G. Let the edge U,V have the least weight. Is it true that UV is always part of any minimum spanning tree of G? Justify your answer.

→ For edge UV to be part of MST, there are two cases :-

1. If there is no other edge or edges whose weight is same as edge UV present, then edge UV will be present mandatory in the MST.
2. If there are one or more edges with the same weight as UV, then we consider only UV edge iff it doesn't make any cycle.

Q3. Let G be a graph and T be a minimum spanning tree of G. Suppose that the weight of edge e is decreased. How can you find the minimum spanning tree of modified graph? What is the runtime of your solution?

→ The edge with decreased weight may or may not be present in final result on certain conditions.

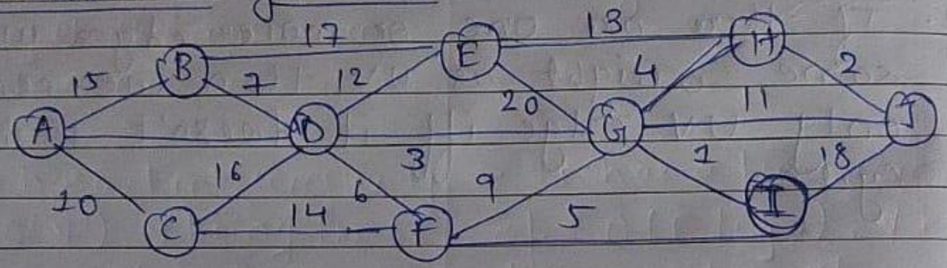
1) If edge belongs to MST and it is decreased later then it still remains in MST.

Lucky Page No.:
Date: / /

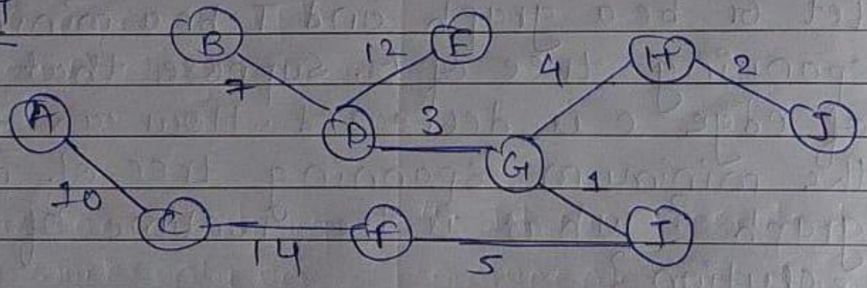
2) If edge does not belongs to MST, then include the edge if there is no cycle formation.
If there is cycle formation by including the edge then remove the edge with maximum weight in cycle.

This takes $O(|T|) = O(|V|)$ time.

② Kruskal's Algorithm



→ MST



$\boxed{\text{Min cost} = 58}$

Q. How many ~~span~~ edges does a minimum spanning tree have in the above example?

→ No. of edges $|E| = |V| - 1$
$|E| = 10 - 1 \Rightarrow \boxed{|E| = 9}$

Weight	Source	Destination	Edge.
1	G	I	GI
2	H	J	HJ
3	D	G	DG
4	G	H	GH
5	F	I	FI
7	B	D	BD
10	A	C	AC
12	D	E	DE
14	C	F	CF.

Q4. Find order of edges for Kruskal's & prim's ?

→ GI, HJ, DJ, GH, FI, BD, AC, DE, CF.
For kruskal's.

AC, CF, FI, IG, GD, GH, HJ, DB, DE.
For prim's.