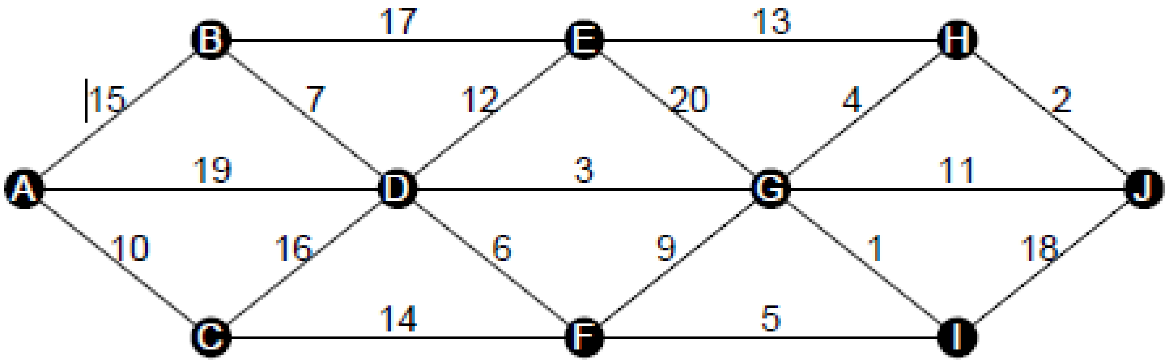
Name:Vishal Chauhan
PRN:2020BTECS00090
Sub:DAA
Assignment no:7
Topic:Greedy Method

Implement Kruskal’s algorithm & Prim's algorithm to find **M**inimum **S**panning **T**ree (*MST*) of the given an undirected, connected and weighted graph.



Implementation:

Kruskal's Algorithm:

```
// C++ program for the above approach

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



class DSU {
    int* parent;
    int* rank;

public:
    DSU(int n)
    {
```

```
            parent = new int[n];rank =
            new int[n];

            for (int i = 0; i < n; i++) {parent[i] = -1;
                  rank[i] = 1;
            }
      }

      int find(int i)
      {
            if (parent[i] == -1)return i;

            return parent[i] = find(parent[i]);
      }

      void unite(int x, int y)
      {
            int s1 = find(x);int s2 =
            find(y);

            if (s1 != s2) {
                  if (rank[s1] < rank[s2]) {parent[s1]
                        = s2; rank[s2] += rank[s1];
                  }
                  else {
                        parent[s2] = s1; rank[s1] +=
                        rank[s2];
                  }
            }
      }
};

class Graph {
      vector<vector<int> > edgelist;int V;

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

      void addEdge(int x, int y, int w)
      {
            edgelist.push_back({ w, x, y });
      }

      void kruskals_mst()
```

```
    {
        // 1. Sort all edges
        sort(edgelist.begin(), edgelist.end());

        // Initialize the DSU
        DSU s(V);
        int ans = 0;
        cout << "Following are the edges in the "
                "constructed MST"
            << endl;
        for (auto edge : edgelist) {
            int w = edge[0];
            int x = edge[1];
            int y = edge[2];

            // Take this edge in MST if it does
            // not forms a cycle
            if (s.find(x) != s.find(y)) {
                s.unite(x, y);
                ans += w;
                cout << x << " -- " << y << " == " << w
                    << endl;
            }
        }

        cout << "Minimum Cost Spanning Tree: " << ans;
    }
};


int   main()
{

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

```
    g.addEdge(6, 8, 1);
    g.addEdge(6, 9, 11);
    g.addEdge(7, 9, 2);
    g.addEdge(8, 9,18 );

    // Function call
    g.kruskals_mst();
    return 0;
}
```

Output:

```
Following are the edges in the constructed MST
6 -- 8 ==1
7 -- 9 ==2
3 -- 6 ==3
6 -- 7 == 4
5 -- 8 == 5
1 -- 3 == 7
0 -- 2 == 10
3 -- 4 == 12
2 -- 5 == 14
Minimum Cost Spanning Tree: 58
```

Prim's Algorithm:

```
// A C++ program for Prim's Minimum

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


#define V 10


int minKey(int key[], bool mstSet[])
{

        int min = INT_MAX, min_index;

        for (int v = 0; v < V; v++)
                if (mstSet[v] == false && key[v] < min)min =
                        key[v], min_index = v;

        return min_index;
}


void printMST(int parent[], int graph[V][V])
{
      cout << "Edge \tWeight\n"; for (int i =
                           1; i < V; i++)
                              cout << parent[i] << " - " << i << " \t"
                                   << graph[i][parent[i]] << " \n";
}


void primMST(int graph[V][V])
{


        int parent[V];


        int key[V];


        bool mstSet[V];


        for (int i = 0; i < V; i++)
                key[i] = INT_MAX, mstSet[i] = false;

        key[0] = 0;
```

```
    parent[0] = -1;

    for (int count = 0; count < V - 1; count++) {

        int u = minKey(key, mstSet);


        mstSet[u] = true;


        for (int v = 0; v < V; v++)


            if (graph[u][v] && mstSet[v] == false&&
                graph[u][v] < key[v])
                parent[v] = u, key[v] = graph[u][v];
    }


    printMST(parent, graph);
}


int main()
{

    int graph[V][V] = {
        { 0,15,10,19,0,0,0,0,0,0 },
        { 15,0,0,7,17,0,0,0,0,0 },
        { 10,0,0,16,0,14,0,0,0,0 },
        { 19,7,16,0,12,6,3,0,0,0 },
        { 0,17,0,12,0,0,20,13,0,0 },
        { 0,0,14,6,0,0,9,0,5,0 },
        { 0,0,0,3,20,9,0,4,1,11 },
        { 0,0,0,0,13,0,4,0,0,2 },
        { 0,0,0,0,0,5,1,0,0,18 },
        { 0,0,0,0,0,0,11,2,18,0 }
    };

    primMST(graph);

    return 0;
}
```

Output:

Edge	Weight
3 - 1	7
0 - 2	10
6 - 3	3
3 - 4	12
2 - 5	14
8 - 6	1
6 - 7	4
5 - 8	5
7 - 9	2

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

Answer: As 10 nodes are there so 10-1 edges required forminimum spanning tree

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.

Answer:

My method of contradiction we can prove this statement

. 1)Assume we have constructed a MST without including minimum edge(u,v).

2) **Now add the edge (u,v)**

In it then we get a cycle in MST now remove the node withmaximum weight .

3) **Hence previously MST is not the minimum one .**

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?

Answer:

If weight of edge e is less than previous weight then we can modify only those end points whose edge is updated and we can find this in O(1) by using Kruskal's DSU method

Q) Find order of edges for Kruskal's and

Prim's?Answer:

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. 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.