ASSIGNMENT 5

1. **Aim** –You have a business with several offices.you want to lease phone lines to connect them up with each other and the phone company charges different amounts of money to connect different pairs of cities.You want a set of lines that connects all your offices with a minimum total cost.Solve the problem by suggesting data structures.

2. **Objective** – To find the minimum total cost for set of lines connecting all offices.

3. **Theory –**

There are n cities and there are roads in between some of the cities. Somehow all the roads are damaged simultaneously. We have to repair the roads to connect the cities again. There is a fixed cost to repair a particular road. Find out the minimum cost to connect all the cities by repairing roads. Input is in matrix(city) form, if city[i][j] = 0 then there is not any road between city i and city j, if city[i][j] = a > 0 then the cost to rebuild the path between city i and city j is a. Print out the minimum cost to connect all the cities.
It is sure that all the cities were connected before the roads were damaged.

If the minimum cost edge e *of a graph is unique, then this edge is included in any MST.*

Proof: if *e* was not included in the MST, removing any of the (larger cost) edges in the cycle formed after adding *e* to the MST, would yield a spanning tree of smaller weight.

Step 1 - Remove all loops and parallel edges

Remove all loops and parallel edges from the given graph. In case of parallel edges, keep the one which has the least cost associated and remove all others.

Step 2 - Choose any arbitrary node as root node

In this case, we choose **S** node as the root node of Prim's spanning tree. This node is arbitrarily chosen, so any node can be the root node. One may wonder why any video can be a root node. So the answer is, in the spanning tree all the nodes of a graph are included and because it is connected then there must be at least one edge, which will join it to the rest of the tree.

Step 3 - Check outgoing edges and select the one with less cost

After choosing the root node **S**, we see that S,A and S,C are two edges with weight 7 and 8, respectively. We choose the edge S,A as it is lesser than the other.

Now, the tree S-7-A is treated as one node and we check for all edges going out from it. We select the one which has the lowest cost and include it in the tree.

After this step, S-7-A-3-C tree is formed. Now we'll again treat it as a node and will check all the edges again. However, we will choose only the least cost edge. In this case, C-3-D is the new edge, which is less than other edges' cost 8, 6, 4, etc.

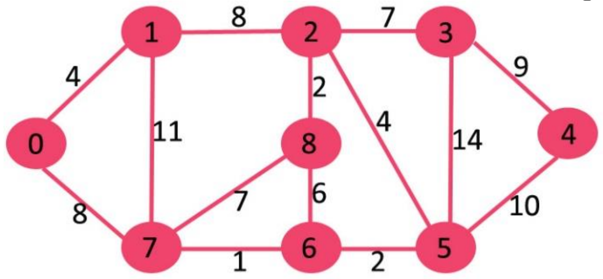
After adding node **D** to the spanning tree, we now have two edges going out of it having the same cost, i.e. D-2-T and D-2-B. Thus, we can add either one. But the next step will again yield edge 2 as the least cost. Hence, we are showing a spanning tree with both edges included.

We may find that the output spanning tree of the same graph using two different algorithms is same.

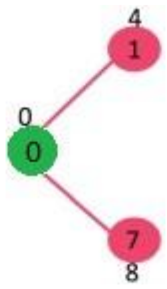
4. **Algorithm** –

1) Create a set *mstSet* that keeps track of vertices already included in MST.
2) Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3) While mstSet doesn’t include all vertices

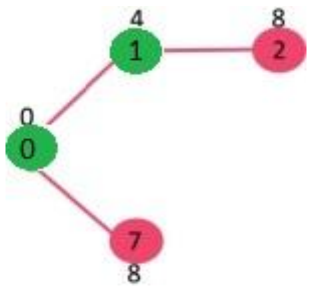
….**a)** Pick a vertex *u* which is not there in *mstSet* and has minimum key value.
….**b)** Include *u* to mstSet.
….**c)** Update key value of all adjacent vertices of *u*. To update the key values, iterate through all adjacent vertices. For every adjacent vertex *v*, if weight of edge *u*-*v* is less than the previous key value of *v*, update the key value as weight of *u*-*v*
The idea of using key values is to pick the minimum weight edge from cut. The key values are used only for vertices which are not yet included in MST, the key value for these vertices indicate the minimum weight edges connecting them to the set of vertices included in MST.
Let us understand with the following example:



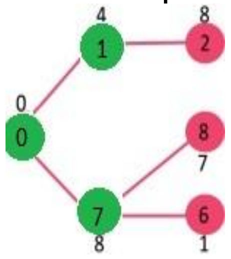
The set *mstSet* is initially empty and keys assigned to vertices are {0, INF, INF, INF, INF, INF, INF, INF} where INF indicates infinite. Now pick the vertex with minimum key value. The vertex 0 is picked, include it in *mstSet*. So *mstSet* becomes {0}. After including to *mstSet*, update key values of adjacent vertices. Adjacent vertices of 0 are 1 and 7. The key values of 1 and 7 are updated as 4 and 8. Following subgraph shows vertices and their key values, only the vertices with finite key values are shown. The vertices included in MST are shown in green color.



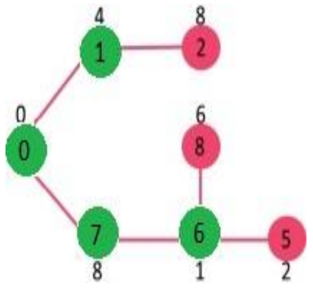
Pick the vertex with minimum key value and not already included in MST (not in mstSET). The vertex 1 is picked and added to mstSet. So mstSet now becomes {0, 1}. Update the key values of adjacent vertices of 1. The key value of vertex 2 becomes 8.



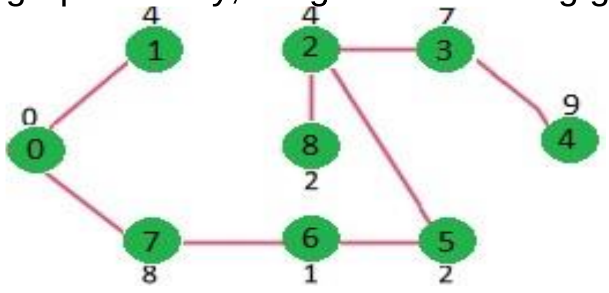
Pick the vertex with minimum key value and not already included in MST (not in mstSET). We can either pick vertex 7 or vertex 2, let vertex 7 is picked. So mstSet now becomes {0, 1, 7}. Update the key values of adjacent vertices of 7. The key value of vertex 6 and 8 becomes finite (1 and 7 respectively).



Pick the vertex with minimum key value and not already included in MST (not in mstSET). Vertex 6 is picked. So mstSet now becomes {0, 1, 7, 6}. Update the key values of adjacent vertices of 6. The key value of vertex 5 and 8 are updated.



We repeat the above steps until *mstSet* includes all vertices of given graph. Finally, we get the following graph.



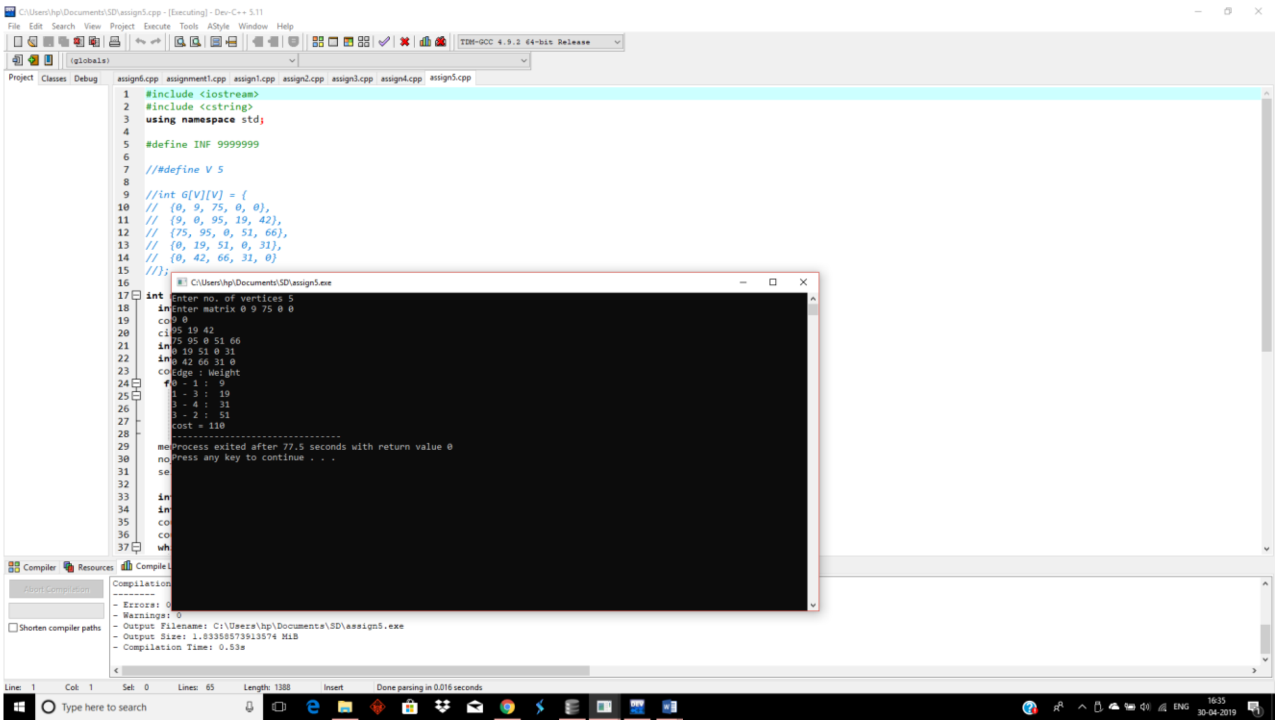
5. **Program code –**

```
#include <iostream>
```

```
#include <cstring>
using namespace std;

#define INF 9999999

//#define V 5

//int G[V][V] = {
//  {0, 9, 75, 0, 0},
//  {9, 0, 95, 19, 42},
//  {75, 95, 0, 51, 66},
//  {0, 19, 51, 0, 31},
//  {0, 42, 66, 31, 0}
//};

int main () {
  int no_edge,sum=0,V;
  cout<<"Enter no. of vertices";
  cin>>V;
  int G[V][V];
  int selected[V];
  cout<<"Enter matrix";
   for (int i = 0; i < V; i++) {
          for (int j = 0; j < V; j++) {
                   cin>>G[i][j];
           }
```

```
                          }
  memset (selected, false, sizeof (selected));
  no_edge = 0;
  selected[0] = true;

  int x;
  int y;
  cout << "Edge" << " : " << "Weight";
  cout << endl;
  while (no_edge < V - 1) {

     int min = INF;
     x = 0;
     y = 0;

     for (int i = 0; i < V; i++) {
       if (selected[i]) {
          for (int j = 0; j < V; j++) {
            if (!selected[j] && G[i][j]) {
               if (min > G[i][j]) {
                  min = G[i][j];
                  x = i;
                  y = j;
               }

            }
          }
        }
      }
      cout << x <<  " - " << y << " :  " << G[x][y];
      sum=sum+G[x][y];
      cout << endl;
      selected[y] = true;
      no_edge++;
    }
 cout<<"cost = "<<sum;
  return 0;
}
```

6. **Output screen shots/Copy Past From Terminal –**



```
#include <iostream>
#include <cstring>
using namespace std;

#define INF 9999999

//#define V 5

//int G[V][V] = {
//  {0, 9, 75, 0, 0},
//  {9, 0, 95, 19, 42},
//  {75, 95, 0, 51, 66},
//  {0, 19, 51, 0, 31},
//  {0, 42, 66, 31, 0}
//};
```

```
Enter no. of vertices 5
Enter matrix 0 9 75 0 0
9 0
95 19 42
75 95 0 51 66
0 19 51 0 31
0 42 66 31 0
Edge : Weight
0 - 1 :  9
1 - 3 :  19
3 - 4 :  31
3 - 2 :  51
cost = 110
--------------------------------
Process exited after 77.5 seconds with return value 0
Press any key to continue . . .
```

```
Compilation
--------
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\hp\Documents\SD\assign5.exe
- Output Size: 1.83358573913574 MiB
- Compilation Time: 0.53s
```

7. **Conclusion –**

Worst case time complexity of Prim’s Algorithm

= O(ElogV) using binary heap

= O(E + VlogV) using Fibonacci heap

Prim's algorithm, in contrast with Kruskal's algorithm, treats the nodes as a single tree and keeps on adding new nodes to the spanning tree from the given graph.