Assignment 7

Name: Soham Mahajan

Roll No:SYCOC160

Program:

```
#include <iostream>
#define infinity 9999
using namespace std;
class spt
{
   int vertex;
   int Edges;
   int adj[100][100];
   int sp[100][100];
   int minCost;
public:
   spt(int V)
   {
      vertex = V;
      for (int i = 0; i < V; i++)
         for (int j = 0; j < V; j++)
         {
            adj[i][j] = 0;
            sp[i][j] = 0;
         }
   }
```

```
   int getMinCost()
   {
      return this->minCost;
   }
   void createGraph()
   {
      int source, destination, weight;
      char choice = 'y';
      cout << "Enter the details of edges (source, destination and weight)"
<< endl;
      do
      {
         cout << "Enter edge - source, destination, weight" << endl;
         cin >> source >> destination >> weight;
         adj[source][destination] = weight;
         adj[destination][source] = weight;
         cout << "Do you want to enter another edge? (Y-yes and N-No)" <<
endl;
         cin >> choice;
      } while (choice == 'y' || choice == 'Y');
   }
void printGraph();
   void printSpanningTree();
   void mst();
};
void spt::printGraph()
```

```
{
   for (int i = 0; i < vertex; i++)
   {
      for (int j = 0; j < vertex; j++)
         cout << " " << adj[i][j];
      cout << endl;
   }
}
void spt::printSpanningTree()
{
   for (int i = 0; i < vertex; i++)
   {
      for (int j = 0; j < vertex; j++)
         cout << " " << sp[i][j];
      cout << endl;
   }
}
void spt::mst()
{
   int cost[vertex][vertex] = {0};
   int visited[10] = {0};
   int distance[vertex] = {infinity};
   int source[vertex] = {0};
   int minDist = 0;
   for (int i = 0; i < vertex; i++)
      for (int j = 0; j < vertex; j++)
```

```
        if (adj[i][j] == 0)
            cost[i][j] = infinity;
        else
            cost[i][j] = adj[i][j];
    distance[0] = 0;
    visited[0] = 1;
    int source_vertex, dest_vertex = 0;
    for (int i = 0; i < vertex; i++)
    {
        distance[i] = cost[0][i];
        source[i] = 0;
    }
    minCost = 0;
Edges = vertex - 1;
    while (Edges > 0)
    {
        minDist = infinity;

        for (int i = 0; i < vertex; i++)
        {
            if (visited[i] == 0 && distance[i] < minDist)
            {
                minDist = distance[i];
                dest_vertex = i;
            }
        }
```

```
        source_vertex = source[dest_vertex];
        sp[source_vertex][dest_vertex] = sp[dest_vertex][source_vertex] =
distance[dest_vertex];
        cout << "Added (" << source_vertex << "," << dest_vertex << ")" <<
endl;
        visited[dest_vertex] = 1;
        minCost += cost[source_vertex][dest_vertex];
        Edges--;

        for (int i = 0; i < vertex; i++)
        {
          if (visited[i] == 0 && cost[dest_vertex][i] < distance[i])
          {
            distance[i] = cost[dest_vertex][i];
            source[i] = dest_vertex;
          }
        }
    }
}
int main()
{
    spt spObj(7);
    spObj.createGraph();
    spObj.printGraph();
    spObj.mst();
```

```
    cout << "Done" << endl;
    spObj.printSpanningTree();
    cout << "The cost of minimum spanning tree is: " << spObj.getMinCost();
    return 0;
}
```

Output:

```
Enter the details of edges (source, destination and weight)
Enter edge - source, destination, weight
1 3 1
Do you want to enter another edge? (Y-yes and N-No)
Y
Enter edge - source, destination, weight
1 2 2
Do you want to enter another edge? (Y-yes and N-No)
Y
Enter edge - source, destination, weight
2 4 1
Do you want to enter another edge? (Y-yes and N-No)
Y
Enter edge - source, destination, weight
3 5 2
Do you want to enter another edge? (Y-yes and N-No)
Y
Enter edge - source, destination, weight
4 6 2
Do you want to enter another edge? (Y-yes and N-No)
```

Y

Enter edge - source, destination, weight

5 6 3

Do you want to enter another edge? (Y-yes and N-No)

Y

Enter edge - source, destination, weight

1 6 7

Do you want to enter another edge? (Y-yes and N-No)

Y

Enter edge - source, destination, weight

2 5 3

Do you want to enter another edge? (Y-yes and N-No)

N

0 0 0 0 0 0 0

0 0 2 1 0 0 7

0 2 0 0 1 3 0

0 1 0 0 0 2 0

0 0 1 0 0 0 2

0 0 3 2 0 0 3

0 7 0 0 2 3 0

Added (1,3)

Added (1,2)

Added (2,4)

Added (3,5)

Added (4,6)

Done

0230000

0001000

0000200

0000020

0000000

0000000