ASSIGNMENT 4

AIM:-

For a weighted graph G, find the minimum spanning tree using Prim’s algorithm.

OBJECTIVE:-

Determine minimum spanning tree using Prim’s algorithm.

THEORY:-

The idea behind Prim’s algorithm is simple, a spanning tree means all vertices must be connected. So the two disjoint subsets (discussed above) of vertices must be connected to make a Spanning Tree. And they must be connected with the minimum weight edge to make it a Minimum Spanning Tree.

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.

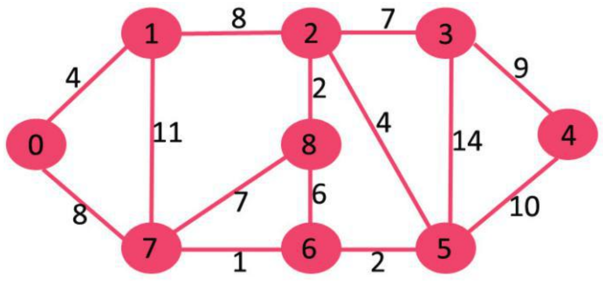
.**b)** Include u to mstSet.

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

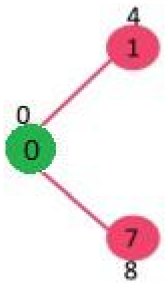
The idea of using key values is to pick the minimum weight edge from cut. The key values are used only for vertices which are not yet included in MST, the key value for these vertices indicate the minimum weight edges connecting them to the set of vertices included in MST.

EXAMPLE:

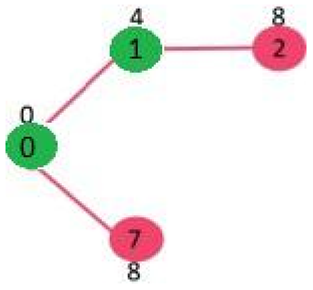
Let us understand with the following example:



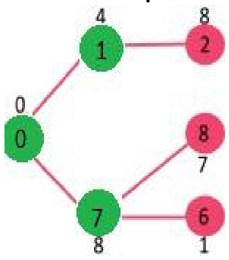
The set mstSet is initially empty and keys assigned to vertices are {0, INF, INF, INF, INF, INF, INF, INF} where INF indicates infinite. Now pick the vertex with minimum key value. The vertex 0 is picked, include it in mstSet. So mstSet becomes {0}. After including to mstSet, update key values of adjacent 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.



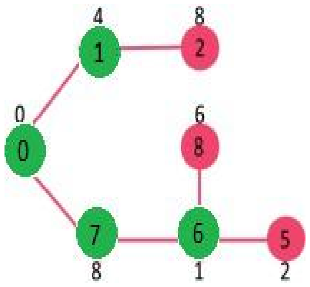
Pick the vertex with minimum key value and not already included in MST (not in mstSET). The vertex 1 is picked and added to mstSet. So mstSet now becomes {0, 1}. Update the key values of adjacent vertices of 1. The key value of vertex 2 becomes 8.



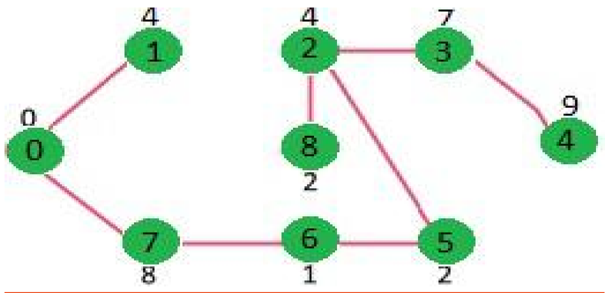
Pick the vertex with minimum key value and not already included in MST (not in mstSET). We can either pick vertex 7 or vertex 2, let vertex 7 is picked. So mstSet now becomes {0, 1, 7}. Update the key values of adjacent vertices of 7. The key value of vertex 6 and 8 becomes finite (1 and 7 respectively).



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.



ALGORITHM:-
PRIM ALGORITHM:-

```
void prims::Prims_Algo(int** g,int 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,V);


        mstSet[u] = true;


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


        if (g[u][z] && mstSet[z] == false && g[u][z] < key[z])
        {
            parent[z] = u;
              key[z] = g[u][z];
        }
    }


    print(parent,g,V);

}
```

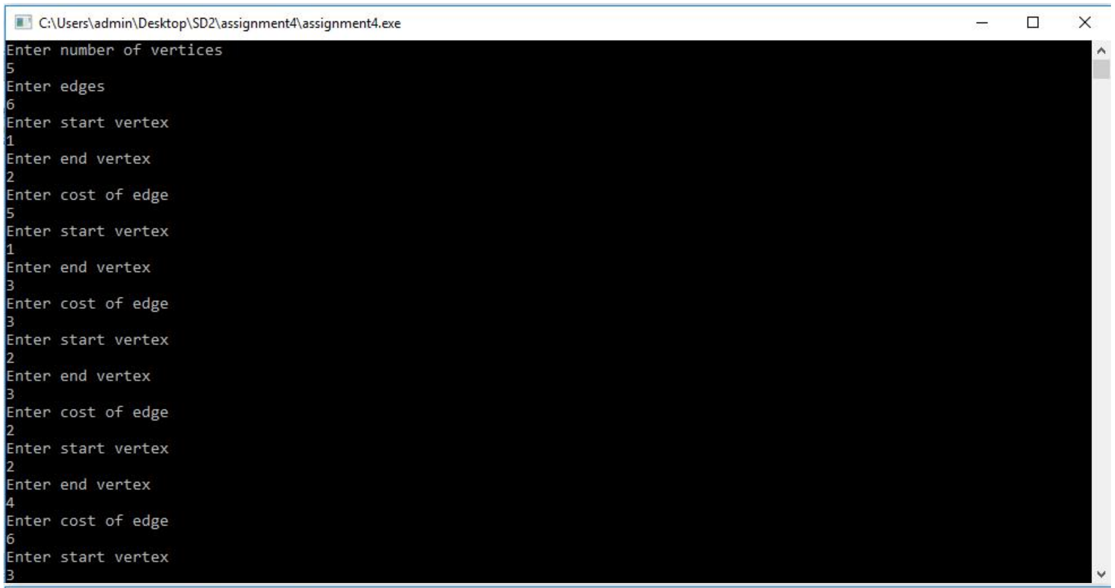
CODE:-

```
#include<iostream>
#define INT_MAX 999
using namespace std;

class prims
```

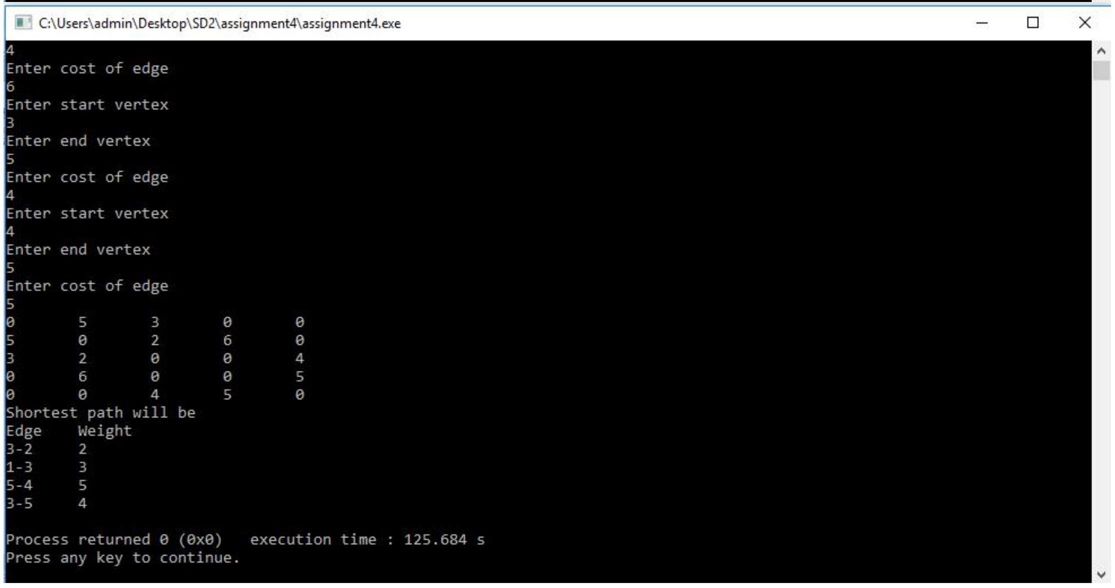
```
{
    public:

    prims()
    {}
    int** build(int);
    void print(int[],int**,int);
    int minKey(int[],bool[], int);
    void Prims_Algo(int**,int);
};

int** prims::build(int V)
{
    int i,j,cost,e;
    int** graph=new int*[V];


    for(int i=0;i<V;i++)
    {
        graph[i]=new int [V];
        for(int j=0;j<V;j++)
        {
        graph[i][j]=0;
        }

    }

    cout<<"Enter edges"<<endl;
    cin>>e;

    for(int k=1;k<=e;k++)
    {
        cout<<"Enter start vertex"<<endl;
        cin>>i;
        cout<<"Enter end vertex"<<endl;
        cin>>j;
        cout<<"Enter cost of edge"<<endl;
        cin>>cost;
        graph[i-1][j-1]=cost;
        graph[j-1][i-1]=cost;
    }

    for(int k=0;k<V;k++)
    {
        for(int l=0;l<V;l++)
        {
        cout<<graph[k][l]<<"\t";
        }
        cout<<"\n";
    }

    return graph;
}

void prims::print(int parent[],int** g,int V)
{
    cout<<"Shortest path will be"<<endl;
    cout<<"Edge \tWeight\n";
    for (int i = 1; i < V;i++)
    cout<< parent[i]+1<<"-"<< i+1<<"\t"<< g[i][parent[i]]<<endl;

}

int prims::minKey(int key[], bool mstSet[], int V)
{

    int min = INT_MAX;
    int 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 prims::Prims_Algo(int** g,int 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,V);


        mstSet[u] = true;


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


        if (g[u][z] && mstSet[z] == false && g[u][z] < key[z])
        {
            parent[z] = u;
              key[z] = g[u][z];
        }
    }


    print(parent,g,V);

}
```

```
int main()
{
    prims p;
    int V;
    cout<<"Enter number of vertices"<<endl;
    cin>>V;
    //p.build(V);
    int **g=p.build(V);
    p.Prims_Algo(g,V);
    return 0;
}
```

OUTPUT:-



```
C:\Users\admin\Desktop\SD2\assignment4\assignment4.exe
Enter number of vertices
5
Enter edges
6
Enter start vertex
1
Enter end vertex
2
Enter cost of edge
5
Enter start vertex
1
Enter end vertex
3
Enter cost of edge
3
Enter start vertex
2
Enter end vertex
3
Enter cost of edge
2
Enter start vertex
2
Enter end vertex
4
Enter cost of edge
6
Enter start vertex
3
```



```
C:\Users\admin\Desktop\SD2\assignment4\assignment4.exe
4
Enter cost of edge
6
Enter start vertex
3
Enter end vertex
5
Enter cost of edge
4
Enter start vertex
4
Enter end vertex
5
Enter cost of edge
5
0       5       3       0       0
5       0       2       6       0
3       2       0       0       4
0       6       0       0       5
0       0       4       5       0
Shortest path will be
Edge    Weight
3-2     2
1-3     3
5-4     5
3-5     4

Process returned 0 (0x0)   execution time : 125.684 s
Press any key to continue.
```

CONCLUSION:-

We have successfully implemented prim’s algorithm to find minimum spanning tree.