ASSIGNMENT 4

AIM: To implement prims algorithm for minimum spanning tree.

OBJECTIVE: For a weighted graph G, find the minimum spanning tree using prims algorithm.

THEORY:

In computer science, Prim's (also known as Jarník's) algorithm is a greedy algorithm that finds a minimum spanning tree for a weighted undirected graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. The algorithm operates by building this tree one vertex at a time, from an arbitrary starting vertex, at each step adding the cheapest possible connection from the tree to another vertex.

The algorithm was developed in 1930 by Czech mathematician Vojtěch Jarník and later rediscovered and republished by computer scientists Robert C. Prim in 1957 and Edsger W. Dijkstra in 1959.Therefore, it is also sometimes called the Jarník's algorithm, Prim–Jarník algorithm, Prim–Dijkstra algorithm or the DJP algorithm.

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.

PROGRAM:

```
#include <iostream>

using namespace std;

class graph

{

   int a[100][100];

   int v;

public:

   void insert_edge(int n1,int n2,int wt)
```

```
{
    if(n1-1>=v||n2-1>=v)
        cout<<"Vertex request out of range\n";
    else
    {
        a[n1-1][n2-1]=wt;
        a[n2-1][n1-1]=wt;
    }
}
void display()
{
    for(int i=0;i<v;i++)
    {
        for(int j=0;j<v;j++)
        {
            cout<<a[i][j]<<"\t";
        }
        cout<<endl;
    }
}
void update_v(int n)
{
    v=n;
}
void prims(int src)
{
```

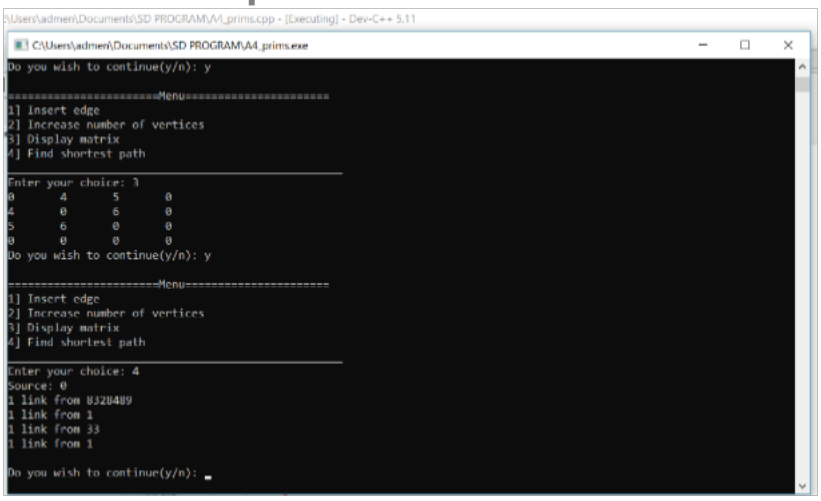
```
int sp[v],dist[v],visited[v],parent[v],c=0;
for(int i=0;i<v;i++)
{
   visited[i]=0;
   dist[i]=9999;
}
dist[src-1]=0;
parent[src-1]=-1;
for(int i=0;i<v;i++)
{
   int min=9999,min_ind;
   for(int j=0;j<v;j++)
   {
      if(!visited[j] && dist[j]<min )
      {
         min=dist[j];
         min_ind=j;
      }
   }
   int U=min_ind;
   visited[U]=1;
   sp[c]=U;
   c++;
   for(int V=0;V<v;V++)
   {
      if(!visited[V] && a[U][V] && a[U][V]<dist[V] && dist[U]!=9999)
```

```
            {
              parent[V]=U;
              dist[V]=a[U][V];
            }
          }
        }
        for(int i=0;i<c;i++){
          cout<<sp[i]+1<<" link from "<<parent[i]+1<<endl;
        }
        cout<<endl;
    }
};
int main(){
    char r;
    do
    {
        graph g;
        char op;
        int v;
        cout<<"Enter number of vertices: ";
        cin>>v;
        g.update_v(v);
        do{
            int c;
            cout<<"\n=======================Menu=======================\n";
            cout<<"1] Insert edge\n2] Increase number of vertices\n3] Display matrix\n4] Find
```

```
shortest path\n";
        cout<<"_____\n";
        cout<<"Enter your choice: ";
        cin>>c;
        switch(c){
           case 1: {
                    int n1,n2,wt;
                    cout<<"Enter the nodes between which there is an edge\n";
                    cin>>n1>>n2;
                    cout<<"Enter weight: ";
                    cin>>wt;
                    g.insert_edge(n1,n2,wt);
                }
                break;
           case 2: {
                    int n;
                    cout<<"Enter the number by which you wish to increase the vertices: ";
                    cin>>n;
                    v+=n;
                    g.update_v(v);
                } break;
           case 3: {
                    g.display();
                }
                break;
           case 4: {
```

```
                int src,dst;
                cout<<"Source: ";
                cin>>src;
                g.prims(src);
              }
              break;
        default:cout<<"Error 404.....page not found\n";
      }
      cout<<"Do you wish to continue(y/n): ";
      cin>>op;
    }while(op=='y' || op=='Y');
    cout<<"Test pass(y/n): ";
    cin>>r;
  }while(r=='n' || r=='N');
  cout<<"******************\n";
  cout<<"*   Thank You!   *\n";
  cout<<"******************\n";
  return 0;
}
```

OUTPUT:



CONCLUSION:

Time Complexity of the above program is O(V^2). If the input graph is represented using adjacency list, then the time complexity of Prim's algorithm can be reduced to O(E log V) with the help of binary heap.