**Experiment No 5**

**AIM:** Study of Prims and Kruskal algorithms for minimum spanning tree.

**PROBLEM STATEMENT:** Design a program to represent a transportation network using graphs. The program should allow users to input cities and routes between them, and then visualize the network graphically. Additionally, it should support operations like adding new cities and routes and determining the minimum spanning tree of the entire network.

**REQUIREMENT:**Turbo C/ GCC Compiler

**OPERATING SYSTEM:** Windows/Linux/Unix.

**THEORY:**

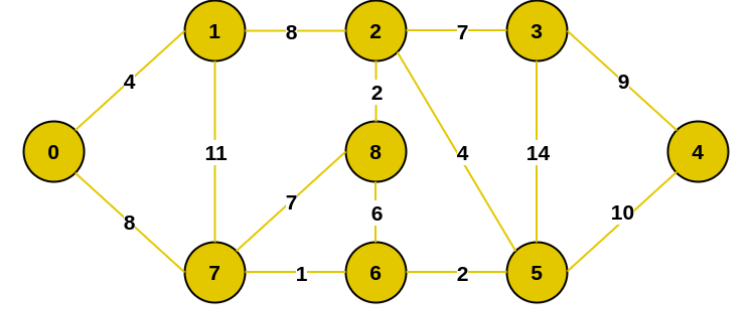
**Prims Algorithm:**

Prim’s algorithm is a Greedy algorithm like Kruskal’s algorithm. This algorithm always starts with a single node and moves through several adjacent nodes, in order to explore all of the connected edges along the way.

**Working of Prim’s Algorithm**

**Step 1:** Determine an arbitrary vertex as the starting vertex of the MST.  
**Step 2:** Follow steps 3 to 5 till there are vertices that are not included in the MST (known as fringe vertex).  
**Step 3:** Find edges connecting any tree vertex with the fringe vertices.  
**Step 4:** Find the minimum among these edges.  
**Step 5:** Add the chosen edge to the MST if it does not form any cycle.  
**Step 6:** Return the MST and exit

**Example:**



Step 1: Firstly, we select an arbitrary vertex that acts as the starting vertex of the Minimum Spanning Tree. Here we have selected vertex 0 as the starting vertex.

Step 2: All the edges connecting the incomplete MST and other vertices are the edges {0, 1} and {0, 7}. Between these two the edge with minimum weight is {0, 1}. So include the edge and vertex 1 in the MST.

Step 3: The edges connecting the incomplete MST to other vertices are {0, 7}, {1, 7} and {1, 2}. Among these edges the minimum weight is 8 which is of the edges {0, 7} and {1, 2}. Let us here include the edge {0, 7} and the vertex 7 in the MST. [We could have also included edge {1, 2} and vertex 2 in the MST].

Step 4: The edges that connect the incomplete MST with the fringe vertices are {1, 2}, {7, 6} and {7, 8}. Add the edge {7, 6} and the vertex 6 in the MST as it has the least weight (i.e., 1).

Step 5: The connecting edges now are {7, 8}, {1, 2}, {6, 8} and {6, 5}. Include edge {6, 5} and vertex 5 in the MST as the edge has the minimum weight (i.e., 2) among them.

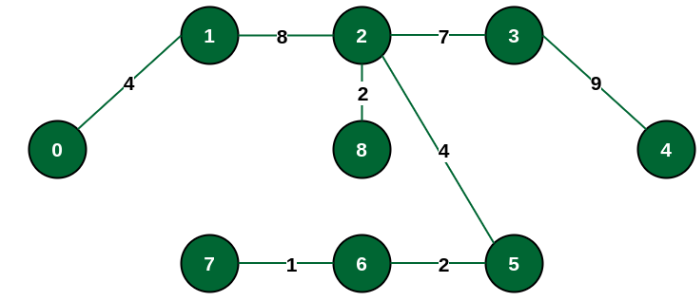
Step 6: Among the current connecting edges, the edge {5, 2} has the minimum weight. So include that edge and the vertex 2 in the MST.

Step 7: The connecting edges between the incomplete MST and the other edges are {2, 8}, {2, 3}, {5, 3} and {5, 4}. The edge with minimum weight is edge {2, 8} which has weight 2. So include this edge and the vertex 8 in the MST.

Step 8: See here that the edges {7, 8} and {2, 3} both have same weight which are minimum. But 7 is already part of MST. So we will consider the edge {2, 3} and include that edge and vertex 3 in the MST.

Step 9: Only the vertex 4 remains to be included. The minimum weighted edge from the incomplete MST to 4 is {3, 4}.

The final structure of the MST is as follows and the weight of the edges of the MST is (4 + 8 + 1 + 2 + 4 + 2 + 7 + 9) = 37.



**Kruskal Algorithm:**

In Kruskal’s algorithm, sort all edges of the given graph in increasing order. Then it keeps on adding new edges and nodes in the MST if the newly added edge does not form a cycle. It picks the minimum weighted edge at first and the maximum weighted edge at last. Thus we can say that it makes a locally optimal choice in each step in order to find the optimal solution. Hence this is a Greedy Algorithm.

**How to find MST using Kruskal’s algorithm?**

Below are the steps for finding MST using Kruskal’s algorithm:

1. Sort all the edges in non-decreasing order of their weight.
2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If the cycle is not formed, include this edge. Else, discard it.
3. Repeat step#2 until there are (V-1) edges in the spanning tree.
4. *The graph contains 9 vertices and 14 edges. So, the minimum spanning tree formed will be having (9 – 1) = 8 edges.   
   After sorting:*

|  |  |  |
| --- | --- | --- |
| *Weight* | *Source* | *Destination* |
| *1* | *7* | *6* |
| *2* | *8* | *2* |
| *2* | *6* | *5* |
| *4* | *0* | *1* |
| *4* | *2* | *5* |
| *6* | *8* | *6* |
| *7* | *2* | *3* |
| *7* | *7* | *8* |
| *8* | *0* | *7* |
| *8* | *1* | *2* |
| *9* | *3* | *4* |
| *10* | *5* | *4* |
| *11* | *1* | *7* |
| *14* | *3* | *5* |

*Now pick all edges one by one from the sorted list of edges*

***Step 1:*** *Pick edge 7-6. No cycle is formed, include it.*

***Step 2:*** *Pick edge 8-2. No cycle is formed, include it.*

***Step 3:*** *Pick edge 6-5. No cycle is formed, include it.*

***Step 4:*** *Pick edge 0-1. No cycle is formed, include it.*

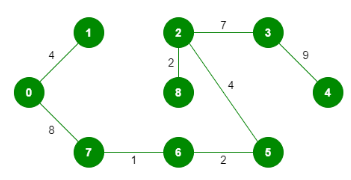
***Step 5:*** *Pick edge 2-5. No cycle is formed, include it.*

***Step 6:*** *Pick edge 8-6. Since including this edge results in the cycle, discard it. Pick edge 2-3: No cycle is formed, include it.*

***Step 7:*** *Pick edge 7-8. Since including this edge results in the cycle, discard it. Pick edge 0-7. No cycle is formed, include it.*

***Step 8:*** *Pick edge 1-2. Since including this edge results in the cycle, discard it. Pick edge 3-4. No cycle is formed, include it.*

***Note:*** *Since the number of edges included in the MST equals to (V – 1), so the algorithm stops here*



**CONCLUSION:** Program for Prims and Kruskal algorithms are implemented successfully.

#include <bits/stdc++.h>

using namespace std;

#define V 5

int minKey(vector<int> &key, vector<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(vector<int> &parent, vector<vector<int>> &graph) {

cout << "Edge \tWeight\n";

for (int i = 1; i < V; i++)

cout << parent[i] << " - " << i << " \t"

<< graph[parent[i]][i] << " \n";

}

void primMST(vector<vector<int>> &graph) {

vector<int> parent(V);

vector<int> key(V);

vector<bool> mstSet(V);

// Initialize all keys as INFINITE

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);

}

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;

}

else if (rank[s1] > rank[s2]) {

parent[s2] = s1;

}

else {

parent[s2] = s1;

rank[s1] += 1;

}

}

}

};

class Graph {

vector<vector<int> > edgelist;

int V1;

public:

Graph(int V1) { this->V1 = V1; }

void addEdge(int x, int y, int w)

{

edgelist.push\_back({ w, x, y });

}

void kruskals\_mst()

{

sort(edgelist.begin(), edgelist.end());

DSU s(V1);

int ans = 0;

int count = 0; // To keep track of the number of edges in MST

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];

if (s.find(x) != s.find(y)) {

s.unite(x, y);

ans += w;

cout << x << " -- " << y << " == " << w

<< endl;

count++;

}

if (count == V1 - 1) {

break;

}

}

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

}

};

int main() {

int ch;

vector<vector<int>> graph = { { 0, 2, 0, 6, 0 },

{ 2, 0, 3, 8, 5 },

{ 0, 3, 0, 0, 7 },

{ 6, 8, 0, 0, 9 },

{ 0, 5, 7, 9, 0 } };

Graph g(4);

while(1)

{

cout<<"\nEnter 1: Prims 2: Kruskal 3: Exit";

cin>>ch;

switch(ch)

{

case 1:

primMST(graph);

break;

case 2:

g.addEdge(0, 1, 10);

g.addEdge(1, 3, 15);

g.addEdge(2, 3, 4);

g.addEdge(2, 0, 6);

g.addEdge(0, 3, 5);

g.kruskals\_mst();

break;

case 3:

return 0;

}

}

return 0;

}