

*Assignment on shortest path Algorithm*

*Course name : Discrete Mathematics*

*Course code: CSE 212*

*Submitted to: Md Sadekur Rahman ( Lecturer of cse)*

*Submitted by:*

*Name : Md. Mainul Hasan*

*Id:0242310005101021*

*Section : 64\_A*

*Department: Computer Science and Engineering*

Assignment

* **Introduction:**
* Prim's Algorithm, an essential method in graph theory for finding Minimum Spanning Trees
* **Understanding of MST:**
* A Minimum Spanning Tree (**MST**) of a graph is a subgraph that connects all vertices together without any cycles and with the minimum possible total edge weight.

**Connected:** All vertices are connected in a single connected component.

**Acyclic:** There are no cycles in the tree.

**Minimum Total Edge Weight:** The sum of the edge weights is the smallest among all possible spanning trees of the graph.

* Importance of Prim’s algorithms in real-world scenarios:

**Networking:**

* **Computer Networks:** Designing minimum-cost networks for connecting computers or data centers.
* **Telecommunications:** Optimizing the layout of cables and connections for telephone networks.

**Infrastructure Development:**

* **Road Networks:** Planning the construction of roads to minimize cost while ensuring all cities are connected.
* **Electrical Grids:** Designing cost-effective power distribution networks.

**Clustering:**

**Data Organization:** Creating hierarchical clusterings in data mining and machine learning

**Prim’s Algorithm (Pseudocode)**

**Graph Representation**:

* Display a weighted, undirected graph with vertices and edges labeled with weights.

**Step-by-Step Execution:**

**1)Initialization:**

* Choose an arbitrary starting vertex (e.g., A).
* Set initial key values for all vertices (key[A] = 0, others = ∞).

**2) Iteration 1:**

* Select vertex A (minimum key value).
* Update key values for adjacent vertices (e.g., B, C).

**3) Iteration 2:**

* Select vertex with the next minimum key value (e.g., B).
* Update key values for adjacent vertices not yet in MST.

**4) Continue:**

* Repeat the process until all vertices are included in the MST.

**5) Final MST:**

* Highlight the edges included in the MST and their total weight.

**Explanation:**

1. We initialize arrays to store distances and predecessors (optional for path reconstruction). We also create a set to track visited vertices.
2. We iterate until all vertices are visited. In each iteration, we find the unvisited vertex with the minimum distance.
3. We explore the neighbors of the current vertex. If a neighbor is unvisited and the tentative distance through the current vertex is less than the current distance to that neighbor, we update the distance and predecessor (optional).
4. Finally, we return the calculated distances and the predecessor information

Code

#include<bits/stdc++.h>

using namespace std;

# define INF 0x3f3f3f3f

typedef pair<int, int> iPair;

void addEdge(vector <pair<int, int> > adj[], int u,int v, int wt)

{

adj[u].push\_back(make\_pair(v, wt));

adj[v].push\_back(make\_pair(u, wt));

}

void primMST(vector<pair<int,int> > adj[], int V)

{

priority\_queue< iPair, vector <iPair> , greater<iPair> > pq;

int src = 0;

vector<int> key(V, INF);

vector<int> parent(V, -1);

vector<bool> inMST(V, false);

pq.push(make\_pair(0, src));

key[src] = 0;

while (!pq.empty())

{

int u = pq.top().second;

pq.pop();

if(inMST[u] == true){

continue;

}

inMST[u] = true;

for (auto x : adj[u])

{

int v = x.first;

int weight = x.second;

if (inMST[v] == false && key[v] > weight)

{

key[v] = weight;

pq.push(make\_pair(key[v], v));

parent[v] = u;

}

}

}

cout<<"Edges weight "<<endl;

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

printf("%d - %d %d\n", parent[i], i,key[i]);

}

int main()

{

/\* int V = 9;

vector<iPair > adj[V];

addEdge(adj, 0, 1, 4); //adj[], int u,int v, int wt

addEdge(adj, 0, 7, 8);

addEdge(adj, 1, 2, 8);

addEdge(adj, 1, 7, 11);

addEdge(adj, 2, 3, 7);

addEdge(adj, 2, 8, 2);

addEdge(adj, 2, 5, 4);

addEdge(adj, 3, 4, 9);

addEdge(adj, 3, 5, 14);

addEdge(adj, 4, 5, 10);

addEdge(adj, 5, 6, 2);

addEdge(adj, 6, 7, 1);

addEdge(adj, 6, 8, 6);

addEdge(adj, 7, 8, 7);

primMST(adj, V);

\*/

int node\_num ;

cin>>node\_num;

vector<iPair > adj[node\_num];

int edge\_num;

cin>>edge\_num;

while(edge\_num--)

{

int u,v,wt;

cin>>u>>v>>wt;

addEdge(adj,u,v,wt);

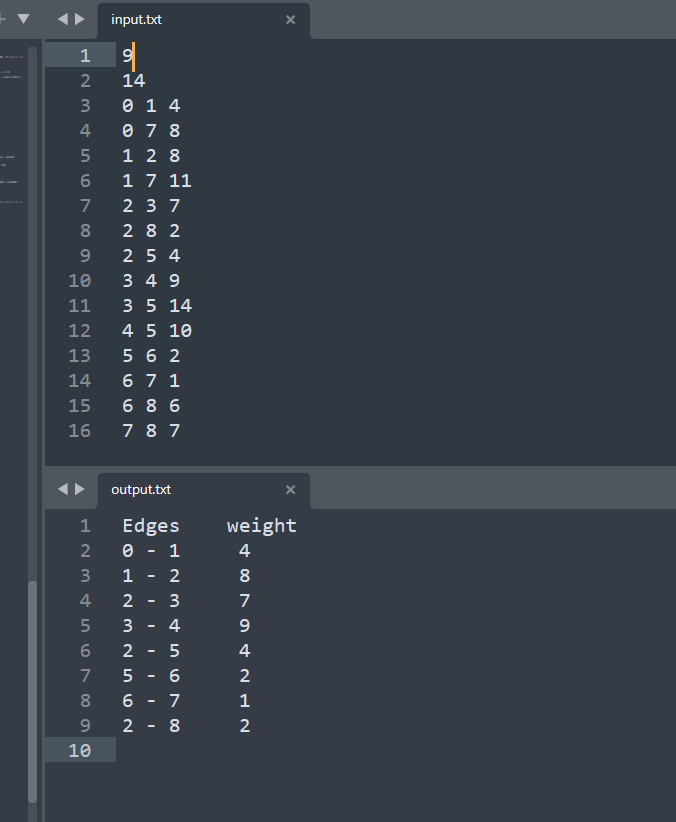
}

primMST(adj, node\_num);

return 0;

}

Output of the code



**Conclusion**

Prim's Algorithm is an efficient and effective method for finding the Minimum Spanning Tree of a graph. It builds the MST incrementally by selecting the smallest edge at each step, ensuring the total weight is minimized.