MIT WORLD PEACE UNIVERSITY

Advanced Data Structures
Second Year B. Tech, Semester 4

MINIMUM COST SPANNING TREE USING *Prim's Algorithm*

ASSIGNMENT NO. 6

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

April 12, 2023

Contents

1 Objectives

1. To study data structure Graph and its representation using cost adjacency Matrix
2. To study and implement algorithm for minimum cost spanning tree
3. To study and implement Prim's Algorithm for minimum cost spanning tree
4. To study how graph can be used to model real world problems

2 Problem Statement

A business house has several offices in different countries; they want to lease phone lines to connect them with each other and the phone company charges different rent to connect different pairs of cities. Business house wants to connect all its offices with a minimum total cost. Solve the problem using Prim's algorithm.

3 Theory

3.1 Spanning Tree

A spanning tree of a graph is a subgraph that is a tree and connects all the vertices together. A single graph can have many different spanning trees. A minimum spanning tree (MST) or minimum weight spanning tree for a weighted, connected and undirected graph is a spanning tree with weight less than or equal to the weight of every other spanning tree. The weight of a spanning tree is the sum of weights given to each edge of the spanning tree.

3.1.1 Example of Spanning Tree

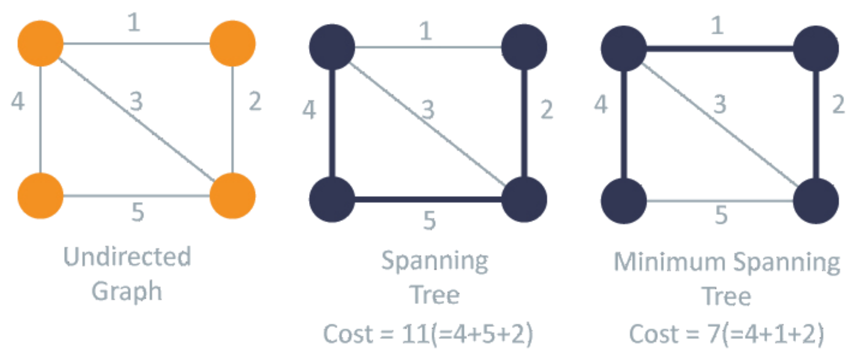

Figure 1: Example of Spanning Tree

3.2 Weighted Graph

A weighted graph is a graph in which each edge is assigned a weight or cost. The weight of an edge is a non-negative real number. The weight of a path is the sum of the weights of its constituent edges. The weight of a cycle is the sum of the weights of its constituent edges. The weight of a graph is the sum of the weights of its edges.

3.2.1 Example of Weighted Graph

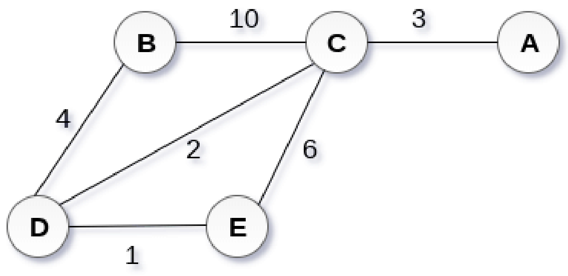

Figure 2: Example of Weighted Graph

3.3 Cost Adjacency Matrix

A cost adjacency matrix is a square matrix that represents the weighted edges of a graph. The matrix is symmetric about the main diagonal. The main diagonal of the matrix is all zeros. The matrix is filled with the weights of the edges. If there is no edge between two vertices, the corresponding entry in the matrix is zero.

3.3.1 Example of Cost Adjacency Matrix

	A	**B**	**C**	**D**	**E**
A	0	-1	3	-1	-1
B	-1	0	10	4	-1
C	3	10	0	2	6
D	-1	4	2	0	1
E	-1	-1	6	1	0

Table 1: Adjacency Matrix of the Graph

3.4 Prim's Algorithm

Prim'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.

3.4.1 Steps

1. Choose a starting vertex B.
2. Add the vertices that are adjacent to A. the edges that connecting the vertices are shown by dotted lines.
3. Choose the edge with the minimum weight among all. i.e. BD and add it to MST. Add the adjacent vertices of D i.e. C and E.

4. Choose the edge with the minimum weight among all. In this case, the edges DE and CD are such edges. Add them to MST and explore the adjacent of C i.e. E and A.

5. Choose the edge with the minimum weight i.e. CA. We can't choose CE as it would cause cycle in the graph.

3.4.2 Example of Prim's Algorithm

This example is based on the above example of weighted graph and its cost adjacency matrix.

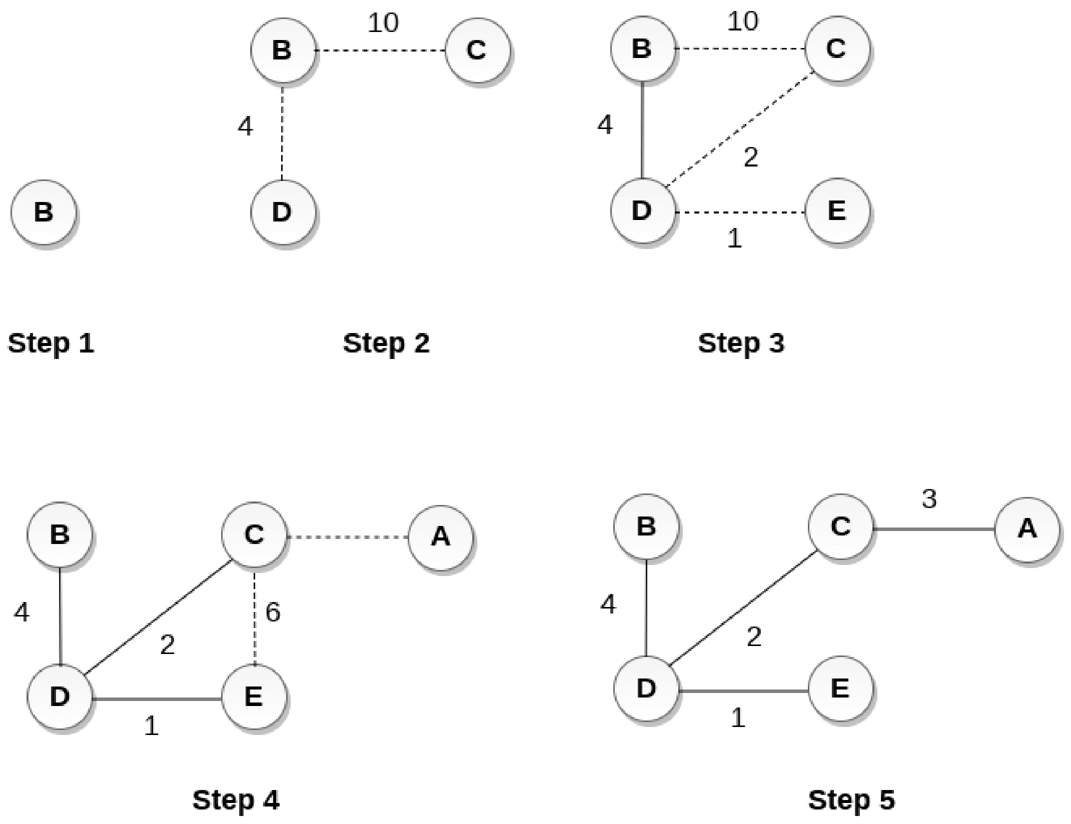

Figure 3: Example of Prim's Algorithm

4 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Visual Studio Code
Compilers : g++ and gcc on linux for C++

5 Test Conditions

1. Input at least 5 nodes.

2. Display minimum cost spanning tree and minimum cost for its graph.

6 Input and Output

1. The minimum cost of the spanning tree.

7 Pseudo Code

7.1 Accepting the Spanning Tree using its adjacency matrix

```
// Graph class with constructor
class Graph
    int V // Number of vertices in the graph
    int** adj // Adjacency matrix to store the graph

    // Constructor to initialize the graph
    Graph(int V)
        this->V = V // Set the number of vertices in the graph
        adj = new int*[V] // Create a 2D array for adjacency matrix with V rows
        for (int i = 0; i < V; i++)
            adj[i] = new int[V] // Create V columns for each row
            memset(adj[i], 0, sizeof(int) * V) // Initialize all entries in the matrix to 0


    // Function to add an edge between two vertices with weight w
    void addEdge(int u, int v, int w)
        adj[u][v] = w // Set the weight of edge (u, v)
        adj[v][u] = w // Set the weight of edge (v, u) (since it's an undirected graph)


    // Function to accept the graph as input from the user
    void acceptGraph()
        int E // Number of edges in the graph
        cout << "Enter the number of edges: "
        cin >> E // Take input from user for the number of edges

        // Loop through all the edges and add them to the graph using the addEdge function
        for (int i = 0; i < E; i++)
            int u, v, w // Variables to store the vertices and weight of the edge
            cout << "Enter the edge (u, v, w): "
            cin >> u >> v >> w // Take input from user for the edge
            addEdge(u, v, w) // Add the edge to the graph
```

7.2 Display of the Spanning Tree using its adjacency matrix

```
void Graph::displayMST(int parent[])
{
    cout << "Minimum Cost Spanning Tree:" << endl;
    for (int i = 1; i < V; i++)
    {
        // Display the edge from parent[i] to i in the MST with weight adj[parent[i]][i]
        cout << parent[i] << " - " << i << " \t" << adj[parent[i]][i] << endl;
    }
}
```

7.3 Prim's Algorithm

```
// Function to find the minimum cost spanning tree using Prim's algorithm
void primsMST()
// Variables to store the parent of each vertex in the MST and the key value of
    each vertex
int parent[V], key[V]
// Array to keep track of whether each vertex is included in the MST or not
bool inMST[V]
// Initialize all key values to infinity and inMST to false
for (int i = 0; i < V; i++)
    key[i] = INT_MAX, inMST[i] = false
// Set the key value of the first vertex to 0
key[0] = 0
parent[0] = -1 // Set the parent of the first vertex as -1

// Loop through all vertices to construct the MST
for (int i = 0; i < V - 1; i++)
    // Find the vertex with the minimum key value that is not yet included in the
    MST
    int u = minKey(key, inMST)
    // Add the vertex to the MST
    inMST[u] = true
    // Loop through all adjacent vertices of the selected vertex
    for (int v = 0; v < V; v++)
        // If v is adjacent to u, has not been included in the MST, and has a
    smaller key value than its current value
        if (adj[u][v] && inMST[v] == false && adj[u][v] < key[v])
            parent[v] = u // Set u as the parent of v in the MST
            key[v] = adj[u][v] // Update the key value of v
```

8 Time Complexity

Let V be the number of vertices in the graph and E be the number of edges in the graph, then:

8.1 Creation of Minimum Cost Spanning Tree using its adjacency matrix

- **Time Complexity:**
$$O(V^2)$$
- **Space Complexity:**
$$O(V^2)$$

8.2 Display of Minimum Cost Spanning Tree using its adjacency matrix

- **Time Complexity:**
$$O(V^2)$$
- **Space Complexity:**
$$O(V^2)$$

8.3 Prim's Algorithm

- **Time Complexity:**
$$O(V^2 \log V)$$
: We maintain a priority queue, and we also maintain a matrix of size VxV. The algorithm runs in logV time, for each edge. There can be VxV edges in the graph. Hence the time complexity.

- **Space Complexity:**
$$O(V^2)$$
: For storing the graph in the form of an adjacency matrix.

9 Code

9.1 Program

```
// Graph class with constructor
class Graph {
    int V; // Number of vertices in the graph
    int** adj; // Adjacency matrix to store the graph

public:
    // Constructor to initialize the graph
    Graph(int V) {
        this->V = V; // Set the number of vertices in the graph
        adj = new int*[V]; // Create a 2D array for adjacency matrix with V rows
        for (int i = 0; i < V; i++) {
            adj[i] = new int[V]; // Create V columns for each row
            memset(adj[i], 0, sizeof(int) * V); // Initialize all entries in the
    matrix to 0
        }
    }

    // Function to add an edge between two vertices with weight w
    void addEdge(int u, int v, int w) {
        adj[u][v] = w; // Set the weight of edge (u, v)
        adj[v][u] = w; // Set the weight of edge (v, u) (since it's an undirected
    graph)
    }

    // Function to accept the graph as input from the user
    void acceptGraph() {
        int E; // Number of edges in the graph
        cout << "Enter the number of edges: ";
        cin >> E; // Take input from user for the number of edges

        // Loop through all the edges and add them to the graph using the addEdge
    function
        for (int i = 0; i < E; i++) {
            int u, v, w; // Variables to store the vertices and weight of the edge
            cout << "Enter the edge (u, v, w): ";
            cin >> u >> v >> w; // Take input from user for the edge
            addEdge(u, v, w); // Add the edge to the graph
        }
    }
};
```

10 Conclusion

Thus, we have represented graph using cost adjacency matrix and implemented Prim's algorithm for MCST.

11 FAQ

1. *Explain two applications of minimum cost spanning tree.*

   Minimum cost spanning trees have a wide range of applications in computer science, operations research, and engineering. Here are two common applications of minimum cost spanning trees:

   (a) **Network design and optimization**:
   Minimum cost spanning trees are often used to design and optimize communication networks, such as the internet or a telephone network.

   For example, a minimum cost spanning tree can be used to find the most cost-effective way to connect a set of cities with a fiber-optic network, or to design a network of sensors to monitor a given area with minimum cost.

   By minimizing the cost of connecting nodes while ensuring that every node is reachable from every other node, minimum cost spanning trees can help improve network performance and reduce operating costs.

   (b) **Circuit design**:
   In circuit design, minimum cost spanning trees are often used to solve the problem of routing wires between electronic components on a printed circuit board.

   By constructing a minimum cost spanning tree that connects all the components, engineers can minimize the length and cost of the wires needed to connect the components, while ensuring that every component is connected to every other component through a path of wires.

   This can lead to more efficient and reliable circuit designs with lower production costs.

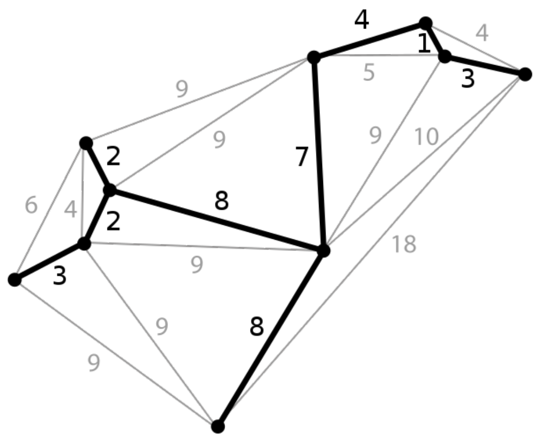

Figure 4: Minimum cost spanning tree for a network

2. *Explain the difference between Prim's and Kruskal's Algorithm for finding minimum cost spanning tree with a small example graph.*

The Main differences between the Two Algorithms are:

(a) **Approach:** Prim's algorithm follows a *greedy* approach where it starts from an arbitrary vertex and builds the tree by adding the next vertex with the lowest cost edge. On the other hand, Kruskal's algorithm is also a greedy approach, but it starts with the minimum edge and adds the next minimum edge which does not create a cycle.

(b) **Data structure:** Prim's algorithm uses a *priority queue* to maintain the vertices with their minimum distance values, while Kruskal's algorithm uses a *disjoint set data structure* to keep track of the connected components.

(c) **Complexity:** Prim's algorithm has a time complexity of $O(E \log V)$ for implementing with a priority queue, while Kruskal's algorithm has a time complexity of $O(E \log E)$ for implementing with a disjoint set data structure.

(d) **Connectivity:** Prim's algorithm generates a single tree rooted at the selected starting vertex, while Kruskal's algorithm generates multiple trees (i.e., forest) and later combines them into a single tree.

(e) **Graph type:** Prim's algorithm is more suitable for dense graphs (i.e., where E is close to V^2), while Kruskal's algorithm works better for sparse graphs (i.e., where E is much smaller than V^2).

(f) **Application:** Prim's algorithm is often used in network routing protocols, where finding the shortest path between two points is essential. On the other hand, Kruskal's algorithm is useful in applications like clustering, where finding the minimum cost connected subgraphs is required.

Prim's Algorithm	**Kruskal's Algorithm**
The tree that we are making or growing always remains connected.	The tree that we are making or growing usually remains disconnected.
Prim's Algorithm grows a solution from a random vertex adding the next cheapest vertex to the existing tree.	Kruskal's Algorithm grows a solution from the cheapest edge by adding the next cheapest edge to the tree / forest.
Prim's Algorithm is faster for dense graphs.	Kruskal's Algorithm is faster for sparse graphs.

Example:
Consider the following graph:

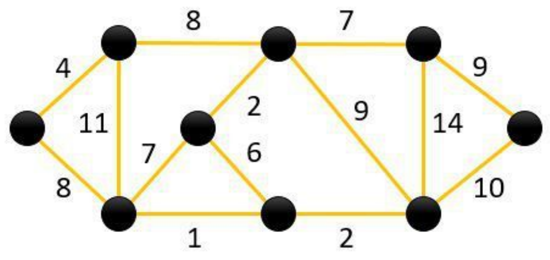

Figure 5: Example graph

Prim's solution:

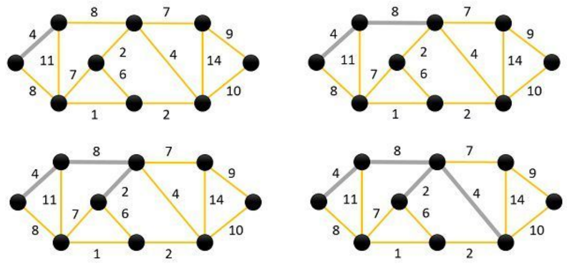

Figure 6: Prim's solution

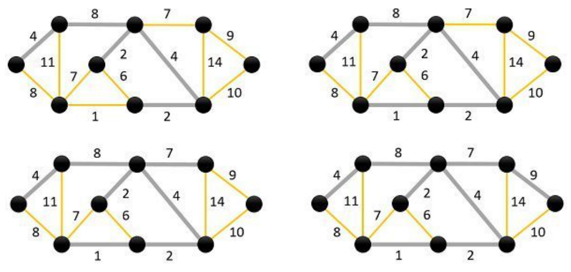

Figure 7: Prim's solution

Kruskal's solution:

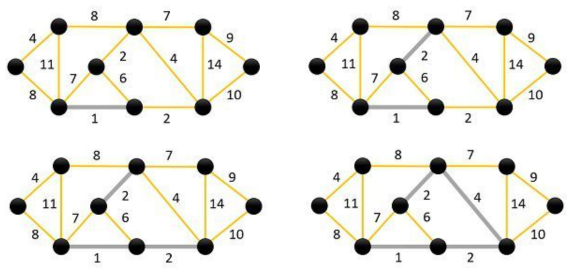

Figure 8: Kruskal's solution

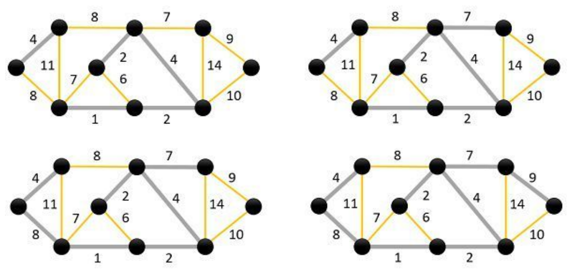

Figure 9: Kruskal's solution

3. *Which algorithmic strategy is used in Prim's algorithm. Explain that algorithmic strategy in brief.*

- Prim's algorithm for finding the minimum cost spanning tree uses a **Greedy approach**, which means that it makes the locally optimal choice at each step with the hope of finding a globally optimal solution.
- The algorithm starts with a single vertex and keeps adding edges to the tree one by one, such that the tree grows gradually and remains connected at all times. At each step, the algorithm selects the edge with the minimum weight that connects a vertex in the tree to a vertex outside the tree. This process continues until all the vertices are included in the tree.
- The key idea behind the algorithm is to start with a small tree and gradually add vertices to it in a way that minimizes the total cost of the tree. The algorithm achieves this by always selecting the edge with the lowest weight that connects a vertex in the tree to a vertex outside the tree. By following this approach, the algorithm ensures that the tree is connected and has the minimum possible cost.
- Overall, Prim's algorithm is an efficient and widely used algorithm for finding the minimum cost spanning tree of a graph, especially when the graph is dense (i.e., has many edges).