**🔷 THEORY**

**📌 Problem Statement**

You have several office branches (represented as graph nodes) and need to connect them using leased phone lines. Each connection (edge) has a cost (weight). Your goal is to connect **all branches with the minimum total cost**, ensuring that all offices remain connected — this is a classic **Minimum Spanning Tree (MST)** problem.

**📌 Prim’s Algorithm**

Prim's Algorithm is a **greedy algorithm** used to find the MST of a connected, weighted, undirected graph.

**Characteristics:**

* Starts with a single vertex and **grows the MST one edge at a time** by adding the minimum-weight edge that connects a visited node to an unvisited one.
* Uses an **adjacency matrix** in this program.
* The result is a tree that connects all vertices with the **minimum possible total edge weight**, and **no cycles**.

**📌 Data Structures Used**

* adjacency matrix: To store connection costs between branches.
* visited[] array: To keep track of included nodes in MST.
* variables p, q: To store endpoints of the minimum-weight edge.
* total: To accumulate total cost of the MST.

**🔷 ALGORITHM**

**🔧 Prim's Algorithm**

text

CopyEdit

Input: Adjacency matrix of graph with V vertices and E edges

Output: Minimum Spanning Tree and its total cost

1. Initialize visited[] array with 0 (false)

2. Set visited[0] = 1 (Start from the first branch)

3. Repeat V-1 times:

a. Set min = ∞

b. For each visited node i:

For each unvisited node j:

- If adj\_mat[i][j] < min:

- min = adj\_mat[i][j]

- p = i, q = j

c. Mark visited[q] = 1

d. Add min to total cost

e. Output the edge (p -> q) and its cost

4. Print total cost

**🔧 Input Phase**

text

CopyEdit

1. Input number of branches (vertices) → v

2. Initialize adjacency matrix a[v][v] with large value (999)

3. Input number of connections (edges) → e

4. For each edge:

a. Input two vertices (l, u)

b. Input cost w

c. Set a[l-1][u-1] = a[u-1][l-1] = w

**🔧 Display Phase**

text

CopyEdit

1. Print the adjacency matrix