Autonomous Institute of Government of Maharashtra)	
Name :	**PRN :** 2241004
Class : T. Y. B.Tech Computer	**Academic Year :** 2024-25
Subject : DAA Lab	**Course Teacher** : Mr. Vinit Kakde
Date of Performance :	**Date of Completion :**

PRACTICAL NO.

AIM:

From a given weighted connected graph, find the Minimum Spanning Tree using **Prim's Algorithm**.

THEORY:

A Minimum Spanning Tree (MST) of a weighted, connected, undirected graph is a subset of edges that connects all vertices without forming a cycle and with the minimum possible total edge weight.

Prim's Algorithm is a Greedy Algorithm. It builds the MST by starting from an arbitrary node and at each step, adding the smallest weight edge that connects a visited vertex to an unvisited vertex.

ALGORITHM:

1. Start with any node and mark it as visited.
2. Initialize a priority queue (or min-heap) to store edges connected to the visited nodes.
3. Pick the edge with the minimum weight that connects to an unvisited node.
4. Mark that node as visited and add its edges to the priority queue.
5. Repeat steps 3–4 until all vertices are included.

TIME COMPLEXITY:

- Using Min Heap with adjacency list: $O(E \log V)$
- Using adjacency matrix: $O(V^2)$

ADVANTAGES:

1. Better performance for dense graphs
2. Always finds the globally optimal MST
3. No need to sort all edges initially

DISADVANTAGES:

1. Not as efficient for sparse graphs
2. Requires more memory due to adjacency structures
3. Needs a good priority queue for optimal performance

APPLICATIONS:

1. Network design (telecom, electrical grids)
2. Image processing and segmentation
3. Cluster analysis
4. Circuit design and layout
5. Map and route optimizations

EXAMPLE:
Graph (Edge - Weight):
B–D - 5
A–B - 6
C–F - 9
F–E - 10

B–C - 11
G–F - 12
A–G - 15
C–D - 17
D–E - 22
C–G - 25

Steps (starting from A):

1. A–B → 6
2. B–D → 5 → Total = 11
3. B–C → 11 → Total = 22
4. C–F → 9 → Total = 31
5. F–E → 10 → Total = 41
6. F–G → 12 → Total = 53

Minimum Cost = 53

PROGRAM:

```
#include <stdio.h>

#include <limits.h>

#include <stdbool.h>


#define MAX 100


int minKey(int key[], bool mstSet[], int V) {

    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)

        if (!mstSet[v] && key[v] < min)

            min = key[v], min_index = v;

    return min_index;
```

```
}

void printMST(int parent[], int graph[MAX][MAX], int V) {
    int totalWeight = 0;
    printf("Edges in Minimum Spanning Tree:\n");
    for (int i = 1; i < V; i++) {
        printf("%d -- %d == %d\n", parent[i], i, graph[i][parent[i]]);
        totalWeight += graph[i][parent[i]];
    }
    printf("Total Weight of MST: %d\n", totalWeight);
}

void primMST(int graph[MAX][MAX], int V) {
    int parent[MAX];
    int key[MAX];
    bool mstSet[MAX];

    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, V);
        mstSet[u] = true;
```

```
        for (int v = 0; v < V; v++)
            if (graph[u][v] && !mstSet[v] && graph[u][v] < key[v])
                parent[v] = u, key[v] = graph[u][v];
    }

    printMST(parent, graph, V);
}

int main() {
    int V;
    printf("Enter number of vertices: ");
    scanf("%d", &V);

    int graph[MAX][MAX];
    printf("Enter adjacency matrix (enter 0 if no edge exists):\n");
    for (int i = 0; i < V; i++)
        for (int j = 0; j < V; j++)
            scanf("%d", &graph[i][j]);

    primMST(graph, V);
    return 0;
}
```

#OUTPUT:

```
C:\Users\HP\Desktop\sem6\D
Enter number of vertices: 7
Enter adjacency matrix (enter 0 if no edge exists):
0 6 0 0 0 0 15
6 0 11 5 0 0 0
0 11 0 17 0 9 25
0 5 17 0 22 0 0
0 0 0 22 0 10 0
0 0 9 0 10 0 12
15 0 25 0 0 12 0
Edges in Minimum Spanning Tree:
0 -- 1 == 6
1 -- 2 == 11
1 -- 3 == 5
5 -- 4 == 10
2 -- 5 == 9
5 -- 6 == 12
Total Weight of MST: 53

--------------------------------
Process exited after 83.42 seconds with return value 0
Press any key to continue . . .
```

QUESTIONS:

1. What is the difference between Prim's and Kruskal's algorithm?
   Ans:
   - Prim's grows a single tree by selecting the smallest edge from visited to unvisited.
   - Kruskal's adds edges in order of increasing weight, avoiding cycles using Union-Find.
2. Why is Prim's preferred for dense graphs?
   Ans:
   - It checks all adjacent vertices, making it faster for graphs with many edges.

3. **How does Prim's Algorithm ensure no cycles are formed?**
   Ans:
   - It only adds edges from visited to unvisited nodes, ensuring a tree structure.
4. **Explain how Prim's Algorithm works step by step with an example.**
   Ans:
   - Start from any node, choose the smallest connecting edge to an unvisited node, repeat until all nodes are included.
5. **List applications where Prim's is more effective than Kruskal's.**
   Ans:
   - In network design and dense graphs where adjacency matrices or heaps are efficient.

Conclusion:
Prim's Algorithm is a greedy approach that finds the minimum spanning tree by growing one edge at a time, always choosing the smallest edge that adds a new vertex. It is efficient and reliable for various real-world applications, especially for dense graphs.

Sign of course Teacher

Mr. Vinit Kakde