

Experiment No.5

Title : Minimum Spanning Tree

Reg.No.24141028

Kruskal's Algorithm

Program :

```
#include <stdio.h>

#define MAX 30

// Structure to store edges
struct Edge {
    int u, v, w; // u = start vertex, v = end vertex, w = weight
};

int parent[MAX];

// Function to find the parent of a vertex
int findParent(int i) {
    while (parent[i] != i)
        i = parent[i];
    return i;
}

// Function to join two edges
void unionSets(int u, int v) {
    int parentA = findParent(u);
    int parentB = findParent(v);
    parent[parentA] = parentB;
```

```
}
```

```
int main() {
```

```
    struct Edge edges[MAX];
```

```
    int n, e;
```

```
    int i, j;
```

```
    int totalCost = 0;
```

```
    printf("Enter number of vertices: ");
```

```
    scanf("%d", &n);
```

```
    printf("Enter number of edges: ");
```

```
    scanf("%d", &e);
```

```
    printf("Enter edges (u v w):\n");
```

```
    for (i = 0; i < e; i++) {
```

```
        scanf("%d %d %d", &edges[i].u, &edges[i].v, &edges[i].w);
```

```
    }
```

```
    // Initialize each vertex as its own parent
```

```
    for (i = 0; i < n; i++)
```

```
        parent[i] = i;
```

```
    // Sort edges based on weight (ascending)
```

```
    for (i = 0; i < e - 1; i++) {
```

```
        for (j = 0; j < e - i - 1; j++) {
```

```
            if (edges[j].w > edges[j + 1].w) {
```

```
                struct Edge temp = edges[j];
```

```
                edges[j] = edges[j + 1];
```

```
                edges[j + 1] = temp;
```

```

        }
    }
}

printf("\nEdges included in the Minimum Spanning Tree:\n");

// Kruskal's Algorithm main part
for (i = 0, j = 0; j < n - 1 && i < e; i++) {
    int a = findParent(edges[i].u);
    int b = findParent(edges[i].v);
    if (a != b) {
        printf("Edge: (%d - %d)   Cost: %d\n", edges[i].u, edges[i].v,
edges[i].w);
        totalCost += edges[i].w;
        unionSets(a, b);
        j++;
    }
}

printf("\nTotal Minimum Cost = %d\n", totalCost);

return 0;
}

```

Output:

```
C:\Users\USER\Desktop\241410128\Kruskal's Algorithm.exe
Enter number of vertices: 4
Enter number of edges: 5
Enter edges (u v w):
0 10 6
0 5 2
0 8 4
3 33 7
5 6 9

Edges included in the Minimum Spanning Tree:
Edge: (3 - 33) Cost: 7

Total Minimum Cost = 7

-----
Process exited after 53.39 seconds with return value 0
Press any key to continue . . .
```

Prim's Algorithm

Program :

```
#include <stdio.h>

#define V 5

#define INF 9999

// Function to find the vertex with minimum key value
int minKey(int key[], int mstSet[]) {
    int v;

    int min = INF, min_index = -1;

    for (v = 0; v < V; v++) {
        if (mstSet[v] == 0 && key[v] < min) {
            min = key[v];
            min_index = v;
        }
    }

    return min_index;
}
```

```

}

// Function to print the constructed MST and its total cost
void printMST(int parent[], int graph[V][V]) {
    int i;
    int totalCost = 0;
    printf("Edge \tWeight\n");
    for (i = 1; i < V; i++) {
        printf("%d - %d \t%d\n", parent[i], i, graph[i][parent[i]]);
        totalCost += graph[i][parent[i]];
    }
    printf("\nTotal Minimum Cost = %d\n", totalCost);
}

// Function that constructs and prints MST using Prim's algorithm
void primMST(int graph[V][V]) {
    int parent[V];    // Array to store constructed MST
    int key[V];       // Key values to pick minimum weight edge
    int mstSet[V];    // To represent set of vertices included in MST
    int i, v, count, u;

    // Initialize all keys as infinite and mstSet[] as false
    for (i = 0; i < V; i++) {
        key[i] = INF;
        mstSet[i] = 0;
    }

    key[0] = 0;       // Start from the first vertex
    parent[0] = -1;   // First node is root of MST

    // Build MST with V vertices
    for (count = 0; count < V - 1; count++) {
        u = minKey(key, mstSet);

```

```

        mstSet[u] = 1; // Include vertex in MST
    // Update key and parent for adjacent vertices
    for (v = 0; v < V; v++) {
        if (graph[u][v] && mstSet[v] == 0 && graph[u][v] < key[v]) {
            parent[v] = u;
            key[v] = graph[u][v];
        }
    }
}

// Print the constructed MST and total cost
printMST(parent, graph);
}

int main() {
    int graph[V][V] = {
        {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}
    };

    primMST(graph); // Run Prim's Algorithm

    return 0;
}

```

Output:

```
C:\Users\USER\Desktop\241410128\Prim's Algorithm.exe
Edge    Weight
0 - 1    2
1 - 2    3
0 - 3    6
1 - 4    5

Total Minimum Cost = 16

-----
Process exited after 0.09797 seconds with return value 0
Press any key to continue . . .
```

Application : Simplified Railway Planning using MST

Program :

```
#include <stdio.h>

#define V 5          // Number of railway stations
#define INF 9999     // Represents no direct track

int main() {
    int i, j, edges;
    int min, x, y;
    // Cost matrix in km
    int cost[V][V] = {
        {INF, 2, INF, 6, INF},
        {2, INF, 3, 8, 5},
        {INF, 3, INF, INF, 7},
        {6, 8, INF, INF, 9},
        {INF, 5, 7, 9, INF}
    };

    int selected[V] = {0}; // Track stations included in MST
    selected[0] = 1;       // Start from first station
    int totalCost = 0;
```

```

printf("Railway tracks selected in MST (in km):\n");

for (edges = 0; edges < V - 1; edges++) {
    min = INF;
    x = -1;
    y = -1;
    for (i = 0; i < V; i++) {
        if (selected[i]) {
            for (j = 0; j < V; j++) {
                if (!selected[j] && cost[i][j] < min) {
                    min = cost[i][j];
                    x = i;
                    y = j;
                }
            }
        }
    }

    printf("Station %d - Station %d : %d km\n", x, y, cost[x][y]);
    totalCost += cost[x][y];
    selected[y] = 1;
}

printf("Total minimum track length to connect all stations: %d km\n",
totalCost);

return 0;
}

```

Output:

```
C:\Users\USER\Desktop\241410128\Application using MST.exe

Railway tracks selected in MST (in km):
Station 0 - Station 1 : 2 km
Station 1 - Station 2 : 3 km
Station 1 - Station 4 : 5 km
Station 0 - Station 3 : 6 km
Total minimum track length to connect all stations: 16 km

-----
Process exited after 0.1172 seconds with return value 0
Press any key to continue . . .
```

Time Complexity :

Kruskal's Algorithm

Sorting edges: $O(E \log E)$, where E = number of edges.

Initialize Union-Find: $O(V)$, where V = number of vertices.

For each edge, perform find and union operations: $O(E \alpha(V))$, where $\alpha(V) \approx 1$ in practice.

Total Time Complexity: $O(E \log E + E \alpha(V)) \approx O(E \log E)$

Notes:

Sorting dominates the complexity.

Sparse graph ($E \approx V$): $O(V \log V)$

Dense graph ($E \approx V^2$): $O(V^2 \log V)$

Prim's Algorithm

Using adjacency matrix:

For each vertex (V times), we scan all vertices to find the minimum edge.

Each scan takes $O(V)$, so total = $O(V \times V) = O(V^2)$.

Works fine for dense graphs.

Using min-heap / priority queue:

Each vertex is inserted or updated in the heap: $O(\log V)$.

For all edges (E edges), decrease-key operations take $O(E \log V)$.

Total time = $O((V + E) \log V) \approx O(E \log V)$ for connected graphs.

Efficient for sparse graphs.

Real Time Applications of MST :

1. Network Design

Telecommunication Networks: Connecting telephone exchanges, cell towers, or internet routers with minimum total cable length.

Computer Networks: Laying out LANs or WANs to minimize wiring costs.

2. Railway and Road Planning

Railway Network: Connecting stations with minimum total track length to reduce construction costs.

Road Networks: Designing highways or local roads to connect cities efficiently.

3. Electrical Grid / Power Distribution

Connecting power stations and substations with minimum wiring or cable length.

Reduces cost of laying transmission lines while ensuring all areas are connected.

4. Water Supply / Pipeline Networks

Designing water pipelines to connect reservoirs, treatment plants, and cities efficiently.

Ensures minimum total pipe length and construction cost.

5. Clustering & Image Processing

Clustering Data: MST helps in forming clusters by connecting points with minimum distance.

Image Segmentation: Identifying edges and regions in image analysis.

6. Approximation in Complex Problems

Traveling Salesman Problem (TSP) Approximation: MST is used to approximate minimum routes.

Network Reliability Optimization: Helps in building resilient networks with minimal resources.