A3-B1-01

Aim: Construction of Minimum Spanning Tree

Kruskal's Algo:

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_VERTICES 10

const char *city[] = {"Delhi", "Mumbai", "Hydrabad", "Pune", "Nagpur"};

int getRoot(int parent[], int node) {
    if (parent[node] != node)
        parent[node] = getRoot(parent, parent[node]);
    return parent[node];
}

void mergeSets(int parent[], int rank[], int rootA, int rootB) {
    if (rank[rootA] < rank[rootB])
        parent[rootA] = rootB;
    else if (rank[rootA] > rank[rootB])
        parent[rootB] = rootA;
    else {
        parent[rootB] = rootA;
        rank[rootA]++;
    }
```

```
}

void applyKruskal(int matrix[MAX_VERTICES][MAX_VERTICES], int vertices) {
    int from[MAX_VERTICES * MAX_VERTICES], to[MAX_VERTICES * MAX_VERTICES],
        weights[MAX_VERTICES * MAX_VERTICES];
    int totalEdges = 0;

    for (int a = 0; a < vertices; a++) {
        for (int b = a + 1; b < vertices; b++) {
            if (matrix[a][b] != 0) {
                from[totalEdges] = a;
                to[totalEdges] = b;
                weights[totalEdges] = matrix[a][b];
                totalEdges++;
            }
        }
    }

    for (int i = 0; i < totalEdges - 1; i++) {
        for (int j = 0; j < totalEdges - i - 1; j++) {
            if (weights[j] > weights[j + 1]) {
                int temp = weights[j];
                weights[j] = weights[j + 1];
                weights[j + 1] = temp;

                int swap = from[j];
                from[j] = from[j + 1];
                from[j + 1] = swap;
```

```
            swap = to[j];
            to[j] = to[j + 1];
            to[j + 1] = swap;
        }
    }
}

int parent[MAX_VERTICES], rank[MAX_VERTICES];
for (int i = 0; i < vertices; i++) {
    parent[i] = i;
    rank[i] = 0;
}

printf("\nEdges in Minimum Spanning Tree (Kruskal's Algorithm):\n");
int totalCost = 0, count = 0;

for (int i = 0; i < totalEdges && count < vertices - 1; i++) {
    int node1 = from[i];
    int node2 = to[i];

    int root1 = getRoot(parent, node1);
    int root2 = getRoot(parent, node2);

    if (root1 != root2) {
        printf("%s - %s : %d\n", city[node1], city[node2], weights[i]);
        totalCost += weights[i];
        mergeSets(parent, rank, root1, root2);
```

```
            count++;
        }
    }

    printf("Total Minimum Cost of Spanning Tree: %d\n", totalCost);
}

int main() {
    int vertices;
    printf("Enter The No. Of Vertices (max 5): ");
    scanf("%d", &vertices);

    if (vertices > 5 || vertices < 1) {
        printf("Error: Number of vertices must be between 1 and 5.\n");
        return 1;
    }

    int graph[MAX_VERTICES][MAX_VERTICES] = {0};

    printf("\nEnter The Weight Of Edges (0 for no edge):\n");

    for (int x = 0; x < vertices; x++) {
        for (int y = x + 1; y < vertices; y++) {
            printf("Graph[%s][%s] = ", city[x], city[y]);
            scanf("%d", &graph[x][y]);
            graph[y][x] = graph[x][y];
        }
    }
```

```
    printf("\nAdjacency Matrix:\n\t");
    for (int i = 0; i < vertices; i++)
        printf("%s\t", city[i]);
    printf("\n");

    for (int i = 0; i < vertices; i++) {
        printf("%s\t", city[i]);
        for (int j = 0; j < vertices; j++) {
            printf("%d\t", graph[i][j]);
        }
        printf("\n");
    }

    applyKruskal(graph, vertices);

    return 0;
}

/*Sample Output:
Enter The No. Of Vertices (max 5): 4

Enter The Weight Of Edges (0 for no edge):
Graph[Delhi][Mumbai] = 3
Graph[Delhi][Hydrabad] = 6
Graph[Delhi][Pune] = 8
Graph[Mumbai][Hydrabad] = 4
Graph[Mumbai][Pune] = 6
```

Graph[Hydrabad][Pune] = 9

Adjacency Matrix:

	Delhi	Mumbai	Hydrabad	Pune
Delhi	0	3	6	8
Mumbai	3	0	4	6
Hydrabad	6	4	0	9
Pune	8	6	9	0

Edges in Minimum Spanning Tree (Kruskal's Algorithm):

Delhi - Mumbai : 3

Mumbai - Hydrabad : 4

Mumbai - Pune : 6

Total Minimum Cost of Spanning Tree: 13

*/

Prim’s Algo:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdbool.h>

float primAlgorithm(int size, float adjMatrix[][size]) {
   bool included[size];
   for (int i = 0; i < size; i++) {
      included[i] = false;
   }
```

```
included[0] = true;
int count = 0;
float totalWeight = 0.0;

while (count < size - 1) {
    float smallest = 1e9;
    int src = -1, dest = -1;

    for (int i = 0; i < size; i++) {
        if (included[i]) {
            for (int j = 0; j < size; j++) {
                if (!included[j] && adjMatrix[i][j] != 0 && adjMatrix[i][j] < smallest) {
                    smallest = adjMatrix[i][j];
                    src = i;
                    dest = j;
                }
            }
        }
    }

    if (src != -1 && dest != -1) {
        totalWeight += smallest;
        included[dest] = true;
        count++;
    }
}

return totalWeight;
```

```
}

int main() {
    int n;
    printf("Enter number of freckles (points): ");
    scanf("%d", &n);

    int coords[n][2];
    for (int i = 0; i < n; i++) {
        printf("Enter coordinates of point %d: ", i + 1);
        scanf("%d %d", &coords[i][0], &coords[i][1]);
    }

    float graph[n][n];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            float dx = coords[i][0] - coords[j][0];
            float dy = coords[i][1] - coords[j][1];
            graph[i][j] = sqrt(dx * dx + dy * dy);
        }
    }

    printf("\nAdjacency Matrix (distances):\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%7.3f ", graph[i][j]);
        }
        printf("\n");
```

```
    }

    float mstWeight = primAlgorithm(n, graph);
    printf("\nMinimum Spanning Distance: %.3f\n", mstWeight);

    return 0;
}

/*Sample Output:
Enter number of freckles (points): 6
Enter coordinates of point 1: 2
4
Enter coordinates of point 2: 5
3
Enter coordinates of point 3: 5
4
Enter coordinates of point 4: 7
9
Enter coordinates of point 5: 2
5
Enter coordinates of point 6: 3
7

Adjacency Matrix (distances):
  0.000   3.162   3.000   7.071   1.000   3.162
  3.162   0.000   1.000   6.325   3.606   4.472
  3.000   1.000   0.000   5.385   3.162   3.606
  7.071   6.325   5.385   0.000   6.403   4.472
```

1.000 3.606 3.162 6.403 0.000 2.236

3.162 4.472 3.606 4.472 2.236 0.000

Minimum Spanning Distance: 11.708

*/