

**Data Structure and Algorithm (MCA 271)**

**Lab Practical –**

***BY***

**Himanshu Heda (24225013)**

**SUBMITTED TO**

**Prof. Vandna Kansal**

**SCHOOL OF SCIENCES**

**2024-2025**

**Program Description:**

**Code of the program**

**Output**: - Paste the o/p of the program.

1. Spanning Tree: --

#include <stdio.h>

#include <stdlib.h>

// Structure to represent an edge in the graph

struct Edge {

    int src, dest, weight;

};

// Structure to represent a graph

struct Graph {

    int V, E; // Number of vertices and edges

    struct Edge\* edges; // Array of edges

};

// Structure to represent a subset for union-find

struct Subset {

    int parent;

    int rank;

};

// Function to create a graph with V vertices and E edges

struct Graph\* createGraph(int V, int E) {

    struct Graph\* graph = (struct Graph\*)malloc(sizeof(struct Graph));

    graph->V = V;

    graph->E = E;

    graph->edges = (struct Edge\*)malloc(E \* sizeof(struct Edge));

    return graph;

}

// Find function for union-find

int find(struct Subset subsets[], int i) {

    if (subsets[i].parent != i) {

        subsets[i].parent = find(subsets, subsets[i].parent);

    }

    return subsets[i].parent;

}

// Union function for union-find

void Union(struct Subset subsets[], int x, int y) {

    int rootX = find(subsets, x);

    int rootY = find(subsets, y);

    if (subsets[rootX].rank < subsets[rootY].rank) {

        subsets[rootX].parent = rootY;

    } else if (subsets[rootX].rank > subsets[rootY].rank) {

        subsets[rootY].parent = rootX;

    } else {

        subsets[rootY].parent = rootX;

        subsets[rootX].rank++;

    }

}

// Comparison function for qsort

int compareEdges(const void\* a, const void\* b) {

    struct Edge\* edgeA = (struct Edge\*)a;

    struct Edge\* edgeB = (struct Edge\*)b;

    return edgeA->weight - edgeB->weight;

}

// Function to construct MST using Kruskal's algorithm

void KruskalMST(struct Graph\* graph) {

    int V = graph->V;

    struct Edge result[V]; // Array to store the resultant MST

    int e = 0; // Index variable for result[]

    int i = 0; // Index variable for sorted edges

    // Step 1: Sort all edges by weight

    qsort(graph->edges, graph->E, sizeof(graph->edges[0]), compareEdges);

    // Allocate memory for union-find subsets

    struct Subset\* subsets = (struct Subset\*)malloc(V \* sizeof(struct Subset));

    for (int v = 0; v < V; ++v) {

        subsets[v].parent = v;

        subsets[v].rank = 0;

    }

    // Step 2: Iterate through sorted edges and add them to the MST if no cycle

    while (e < V - 1 && i < graph->E) {

        struct Edge nextEdge = graph->edges[i++];

        int x = find(subsets, nextEdge.src);

        int y = find(subsets, nextEdge.dest);

        if (x != y) { // If including this edge doesn't form a cycle

            result[e++] = nextEdge;

            Union(subsets, x, y);

        }

    }

    // Print the resultant MST

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

    int minimumCost = 0;

    for (i = 0; i < e; ++i) {

        printf("%d -- %d == %d\n", result[i].src, result[i].dest, result[i].weight);

        minimumCost += result[i].weight;

    }

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

    free(subsets);

}

int main() {

    int V = 4; // Number of vertices

    int E = 5; // Number of edges

    struct Graph\* graph = createGraph(V, E);

    // Add edges to the graph

    graph->edges[0].src = 0;

    graph->edges[0].dest = 1;

    graph->edges[0].weight = 10;

    graph->edges[1].src = 0;

    graph->edges[1].dest = 2;

    graph->edges[1].weight = 6;

    graph->edges[2].src = 0;

    graph->edges[2].dest = 3;

    graph->edges[2].weight = 5;

    graph->edges[3].src = 1;

    graph->edges[3].dest = 3;

    graph->edges[3].weight = 15;

    graph->edges[4].src = 2;

    graph->edges[4].dest = 3;

    graph->edges[4].weight = 4;

    KruskalMST(graph);

    free(graph->edges);

    free(graph);

    return 0;

}

OUTPUT : --