**25. Implementation of Minimum Spanning Tree using Kruskal Algorithm**

#include <stdio.h>

#include <stdlib.h>

// Structure to represent an edge

typedef struct {

int src, dest, weight;

} Edge;

// Structure to represent a graph

typedef struct {

int V, E;

Edge\* edges;

} Graph;

// Structure for Union-Find (Disjoint Set Union)

typedef struct {

int\* parent;

int\* rank;

} DSU;

// Create a graph

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

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

graph->V = V;

graph->E = E;

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

return graph;

}

// Create DSU

DSU\* createDSU(int V) {

DSU\* dsu = (DSU\*)malloc(sizeof(DSU));

dsu->parent = (int\*)malloc(V \* sizeof(int));

dsu->rank = (int\*)malloc(V \* sizeof(int));

for (int i = 0; i < V; i++) {

dsu->parent[i] = i;

dsu->rank[i] = 0;

}

return dsu;

}

// Find with path compression

int find(DSU\* dsu, int x) {

if (dsu->parent[x] != x)

dsu->parent[x] = find(dsu, dsu->parent[x]);

return dsu->parent[x];

}

// Union by rank

void unionSets(DSU\* dsu, int x, int y) {

int rootX = find(dsu, x);

int rootY = find(dsu, y);

if (rootX != rootY) {

if (dsu->rank[rootX] < dsu->rank[rootY])

dsu->parent[rootX] = rootY;

else if (dsu->rank[rootX] > dsu->rank[rootY])

dsu->parent[rootY] = rootX;

else {

dsu->parent[rootY] = rootX;

dsu->rank[rootX]++;

}

}

}

// Compare function for qsort

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

Edge\* e1 = (Edge\*)a;

Edge\* e2 = (Edge\*)b;

return e1->weight - e2->weight;

}

// Kruskal's algorithm

void kruskalMST(Graph\* graph) {

int V = graph->V;

qsort(graph->edges, graph->E, sizeof(Edge), compareEdges);

DSU\* dsu = createDSU(V);

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

int count = 0;

for (int i = 0; i < graph->E && count < V - 1; i++) {

Edge e = graph->edges[i];

int setU = find(dsu, e.src);

int setV = find(dsu, e.dest);

if (setU != setV) {

printf("%d -- %d == %d\n", e.src, e.dest, e.weight);

unionSets(dsu, setU, setV);

count++;

}

}

free(dsu->parent);

free(dsu->rank);

free(dsu);

}

// Example usage

int main() {

int V = 4; // Number of vertices

int E = 5; // Number of edges

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

// Define edges

graph->edges[0] = (Edge){0, 1, 10};

graph->edges[1] = (Edge){0, 2, 6};

graph