Experiment-25 : Kruskal Algorithm

25.Write a C Program For Implementation Of Minimum Spanning Tree Using Kruskal Algorithm.

Code :

#include <stdio.h>

#include <stdlib.h>

// Structure to represent an edge with source, destination, and weight

typedef struct Edge {

int src;

int dest;

int weight;

} Edge;

// Structure to represent a graph with vertices, edges, and an array of edges

typedef struct Graph {

int V;

int E;

Edge\* edges;

} Graph;

// Create a new graph with V vertices and E edges

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;

}

// Disjoint Set Union (DSU) structure with parent and rank

typedef struct DSU {

int\* parent;

int\* rank;

} DSU;

// Create a new DSU with V vertices

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 the representative (root) of a set 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 two sets 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 sorting edges by weight

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

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

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

return edgeA->weight - edgeB->weight;

}

// Kruskal's algorithm to find the Minimum Spanning Tree

void kruskalMST(Graph\* graph) {

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

DSU\* dsu = createDSU(graph->V);

Edge\* result = (Edge\*)malloc((graph->V - 1) \* sizeof(Edge));

int edgeCount = 0;

int i = 0;

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

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

int srcRoot = find(dsu, nextEdge.src);

int destRoot = find(dsu, nextEdge.dest);

if (srcRoot != destRoot) {

result[edgeCount++] = nextEdge;

unionSets(dsu, srcRoot, destRoot);

}

}

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

for (int j = 0; j < edgeCount; j++) {

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

}

free(result);

free(dsu->parent);

free(dsu->rank);

free(dsu);

}

int main() {

int V = 4; // Number of vertices

int E = 5; // Number of edges

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

// Initialize edges: (src, dest, weight)

graph->edges[0].src = 0;

graph->edges[0].dest = 1;

graph->edges[0].weight = 10;

// ... (similarly initialize other edges)

kruskalMST(graph);

free(graph->edges);

free(graph);

return 0;

}

Output :

