20 MST

#include <stdio.h>

#include <stdlib.h>

// Structure to represent an edge in the graph

typedef struct Edge {

int source;

int destination;

int weight;

} Edge;

// Function to compare edges based on their weights (used for sorting)

int compare\_edges(const void\* a, const void\* b) {

const Edge\* edge1 = (Edge\*)a;

const Edge\* edge2 = (Edge\*)b;

return edge1->weight - edge2->weight;

}

// Function to find the parent node using path compression (optimization)

int find\_parent(int parent[], int i) {

if (parent[i] != i) {

parent[i] = find\_parent(parent, parent[i]); // Path compression optimization

}

return parent[i];

}

// Function to perform union of two sets (used for merging disjoint sets)

void union\_sets(int parent[], int rank[], int x, int y) {

int x\_root = find\_parent(parent, x);

int y\_root = find\_parent(parent, y);

// Attach the smaller rank tree under the root of the larger rank tree

if (rank[x\_root] < rank[y\_root]) {

parent[x\_root] = y\_root;

} else if (rank[x\_root] > rank[y\_root]) {

parent[y\_root] = x\_root;

} else {

// If ranks are same, increment rank of one tree

parent[y\_root] = x\_root;

rank[x\_root]++;

}

}

// Function to find the minimum spanning tree using Kruskal's algorithm

void kruskal\_mst(Edge edges[], int n, int e) {

// Sort edges in non-decreasing order of their weight

qsort(edges, e, sizeof(Edge), compare\_edges);

// Create parent and rank arrays for union-find operations

int parent[n];

int rank[n];

// Initialize parent and rank arrays

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

parent[i] = i;

rank[i] = 0;

}

// Include edges in the MST one by one

int included\_edges = 0;

int total\_weight = 0;

while (included\_edges < n - 1) {

Edge current\_edge = edges[included\_edges];

// Check if adding the current edge creates a cycle

int x = find\_parent(parent, current\_edge.source);

int y = find\_parent(parent, current\_edge.destination);

if (x != y) {

// Include the edge in the MST as it doesn't form a cycle

total\_weight += current\_edge.weight;

included\_edges++;

// Merge the two sets (vertices) connected by the current edge

union\_sets(parent, rank, x, y);

}

}

// Print the MST edges

printf("Edges in the minimum spanning tree:\n");

for (int i = 0; i < included\_edges; i++) {

printf("(%d, %d, %d)\n", edges[i].source, edges[i].destination, edges[i].weight);

}

printf("Total weight of MST: %d\n", total\_weight);

}

int main() {

int n, e; // Number of vertices and edges

printf("Enter the number of vertices and edges: ");

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

Edge edges[e]; // Array to store edges

printf("Enter the source, destination, and weight of each edge:\n");

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

scanf("%d %d %d", &edges[i].source, &edges[i].destination, &edges[i].weight);

}

kruskal\_mst(edges, n, e);

return 0;

}