**21. Write a C program to Graph traversal using Breadth First Search**

Input:

#include <stdio.h>

#include <stdlib.h>

#define MAX\_VERTICES 100

struct Node {

int data;

struct Node\* next;

};

struct Graph {

struct Node\* adjList[MAX\_VERTICES];

int visited[MAX\_VERTICES];

};

void initializeGraph(struct Graph\* graph, int numVertices) {

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

graph->adjList[i] = NULL;

graph->visited[i] = 0;

}

}

void addEdge(struct Graph\* graph, int src, int dest) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = dest;

newNode->next = graph->adjList[src];

graph->adjList[src] = newNode;

}

void bfs(struct Graph\* graph, int startVertex, int numVertices) {

int queue[MAX\_VERTICES];

int front = 0, rear = 0;

graph->visited[startVertex] = 1;

queue[rear++] = startVertex;

while (front < rear) {

int currentVertex = queue[front++];printf("%d ", currentVertex);

struct Node\* temp = graph->adjList[currentVertex];

while (temp != NULL) {

int adjVertex = temp->data;

if (graph->visited[adjVertex] == 0) {

graph->visited[adjVertex] = 1;

queue[rear++] = adjVertex;

}

temp = temp->next;

}

}

}

int main() {

struct Graph graph;

int numVertices, numEdges;

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

scanf("%d", &numVertices);

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

scanf("%d", &numEdges);

initializeGraph(&graph, numVertices);

printf("Enter edges (source destination):\n");

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

int src, dest;

scanf("%d %d", &src, &dest);

addEdge(&graph, src, dest);

addEdge(&graph, dest, src); // For undirected graph

}

int startVertex;

printf("Enter the starting vertex for BFS: ");

scanf("%d", &startVertex);

printf("BFS traversal starting from vertex %d: ", startVertex);

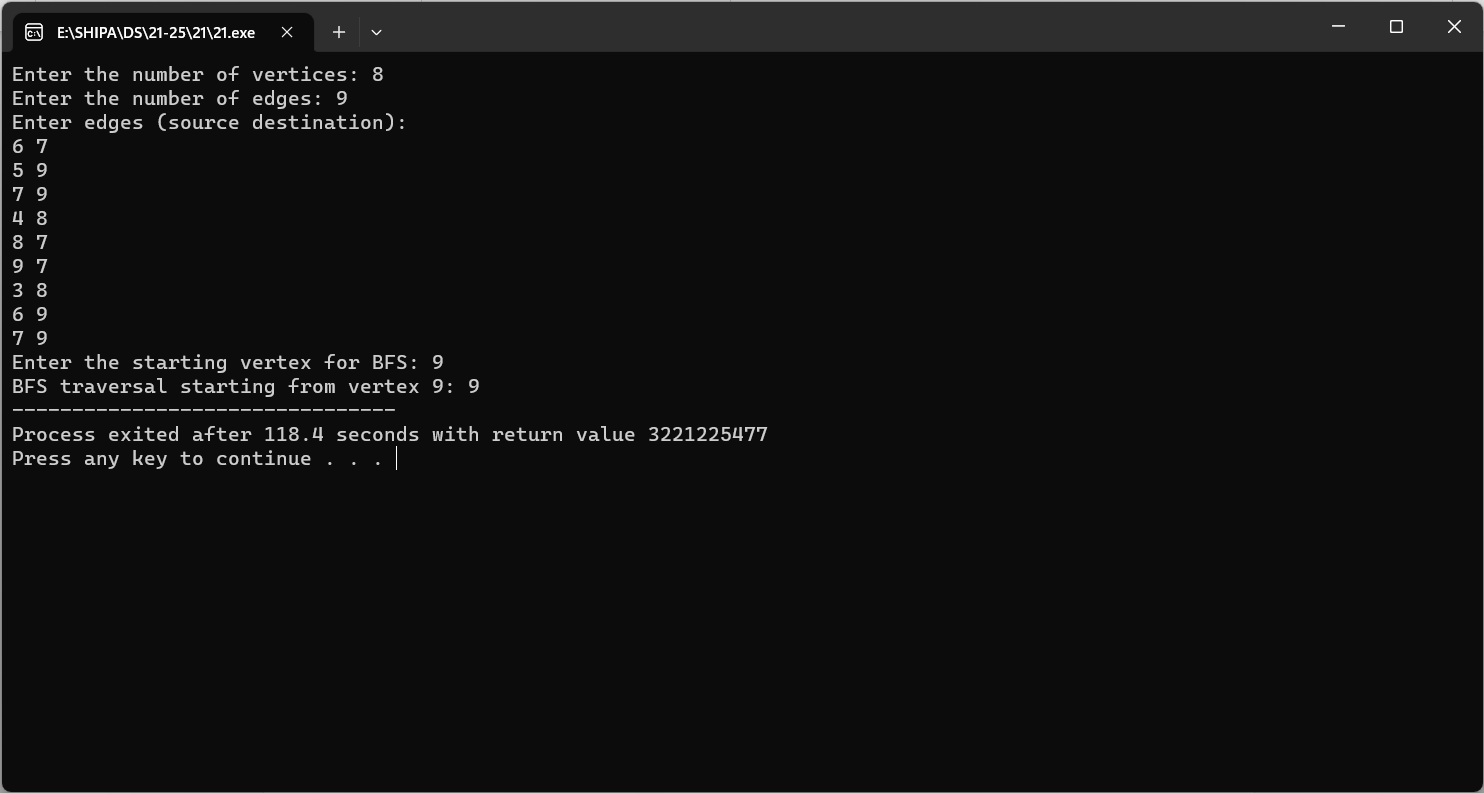
bfs(&graph, startVertex, numVertices);

printf("\n");

return 0;

}

Output:



**22. Write a C program to Graph traversal using Depth First Search**

Input:

#include <stdio.h>

#include <stdlib.h>

#define MAX\_VERTICES 100

struct Node {

int data;

struct Node\* next;

};

struct Graph {

struct Node\* adjList[MAX\_VERTICES];

int visited[MAX\_VERTICES];

};

void initializeGraph(struct Graph\* graph, int numVertices) {

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

graph->adjList[i] = NULL;graph->visited[i] = 0;

}

}

void addEdge(struct Graph\* graph, int src, int dest) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = dest;

newNode->next = graph->adjList[src];

graph->adjList[src] = newNode;

}

void dfs(struct Graph\* graph, int vertex) {

graph->visited[vertex] = 1;

printf("%d ", vertex);

struct Node\* temp = graph->adjList[vertex];

while (temp != NULL) {

int adjVertex = temp->data;

if (graph->visited[adjVertex] == 0) {

dfs(graph, adjVertex);

}

temp = temp->next;

}

}

int main() {

struct Graph graph;

int numVertices, numEdges;

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

scanf("%d", &numVertices);

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

scanf("%d", &numEdges);

initializeGraph(&graph, numVertices);

printf("Enter edges (source destination):\n");

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

int src, dest;

scanf("%d %d", &src, &dest);

addEdge(&graph, src, dest);

addEdge(&graph, dest, src); // For undirected graph

}int startVertex;

printf("Enter the starting vertex for DFS: ");

scanf("%d", &startVertex);

printf("DFS traversal starting from vertex %d: ", startVertex);

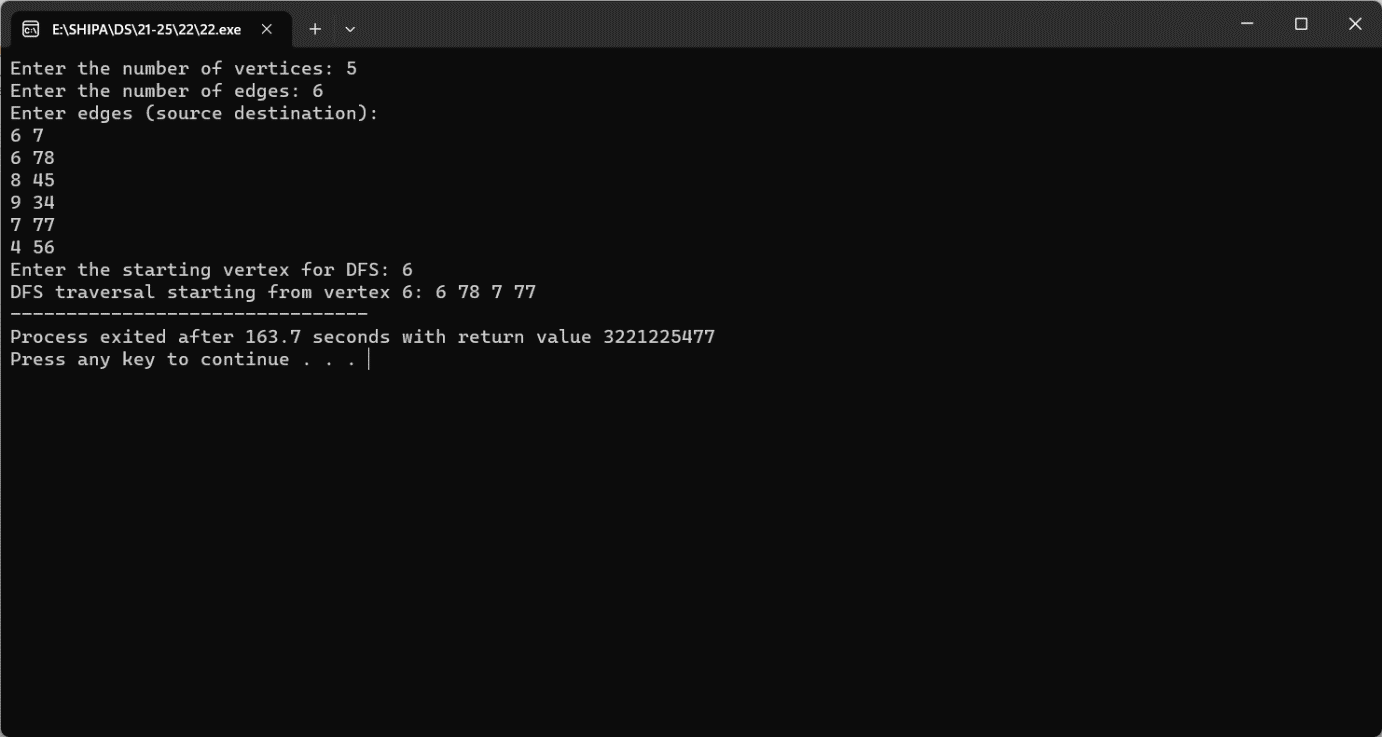
dfs(&graph, startVertex);

printf("\n");

return 0;

}

Output:

****

**23. Implementation of Shortest Path Algorithms using Dijkstra’s Algorithm**

Input:

#include <stdio.h>

#include <limits.h>

#define MAX\_VERTICES 100

struct Graph {

int numVertices;int adjMatrix[MAX\_VERTICES][MAX\_VERTICES];

};

void initializeGraph(struct Graph\* graph, int numVertices) {

graph->numVertices = numVertices;

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

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

graph->adjMatrix[i][j] = 0; // Initialize with no edges

}

}

}

void addEdge(struct Graph\* graph, int src, int dest, int weight) {

graph->adjMatrix[src][dest] = weight;

graph->adjMatrix[dest][src] = weight; // For undirected graph

}

int minDistance(int dist[], int visited[], int numVertices) {

int min = INT\_MAX, minIndex;

for (int v = 0; v < numVertices; v++) {

if (!visited[v] && dist[v] <= min) {

min = dist[v];

minIndex = v;

}

}

return minIndex;

}

void dijkstra(struct Graph\* graph, int src) {

int dist[MAX\_VERTICES];

int visited[MAX\_VERTICES];

for (int i = 0; i < graph->numVertices; i++) {

dist[i] = INT\_MAX;

visited[i] = 0;

}

dist[src] = 0;

for (int count = 0; count < graph->numVertices - 1; count++) {

int u = minDistance(dist, visited, graph->numVertices);

visited[u] = 1;for (int v = 0; v < graph->numVertices; v++) {

if (!visited[v] && graph->adjMatrix[u][v] && dist[u] != INT\_MAX &&

dist[u] + graph->adjMatrix[u][v] < dist[v]) {

dist[v] = dist[u] + graph->adjMatrix[u][v];

}

}

}

printf("Vertex Distance from Source\n");

for (int i = 0; i < graph->numVertices; i++) {

printf("%d

%d\n", i, dist[i]);

}

}

int main() {

struct Graph graph;

int numVertices, numEdges;

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

scanf("%d", &numVertices);

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

scanf("%d", &numEdges);

initializeGraph(&graph, numVertices);

printf("Enter edges and weights (source destination weight):\n");

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

int src, dest, weight;

scanf("%d %d %d", &src, &dest, &weight);

addEdge(&graph, src, dest, weight);

}

int startVertex;

printf("Enter the starting vertex for Dijkstra's algorithm: ");

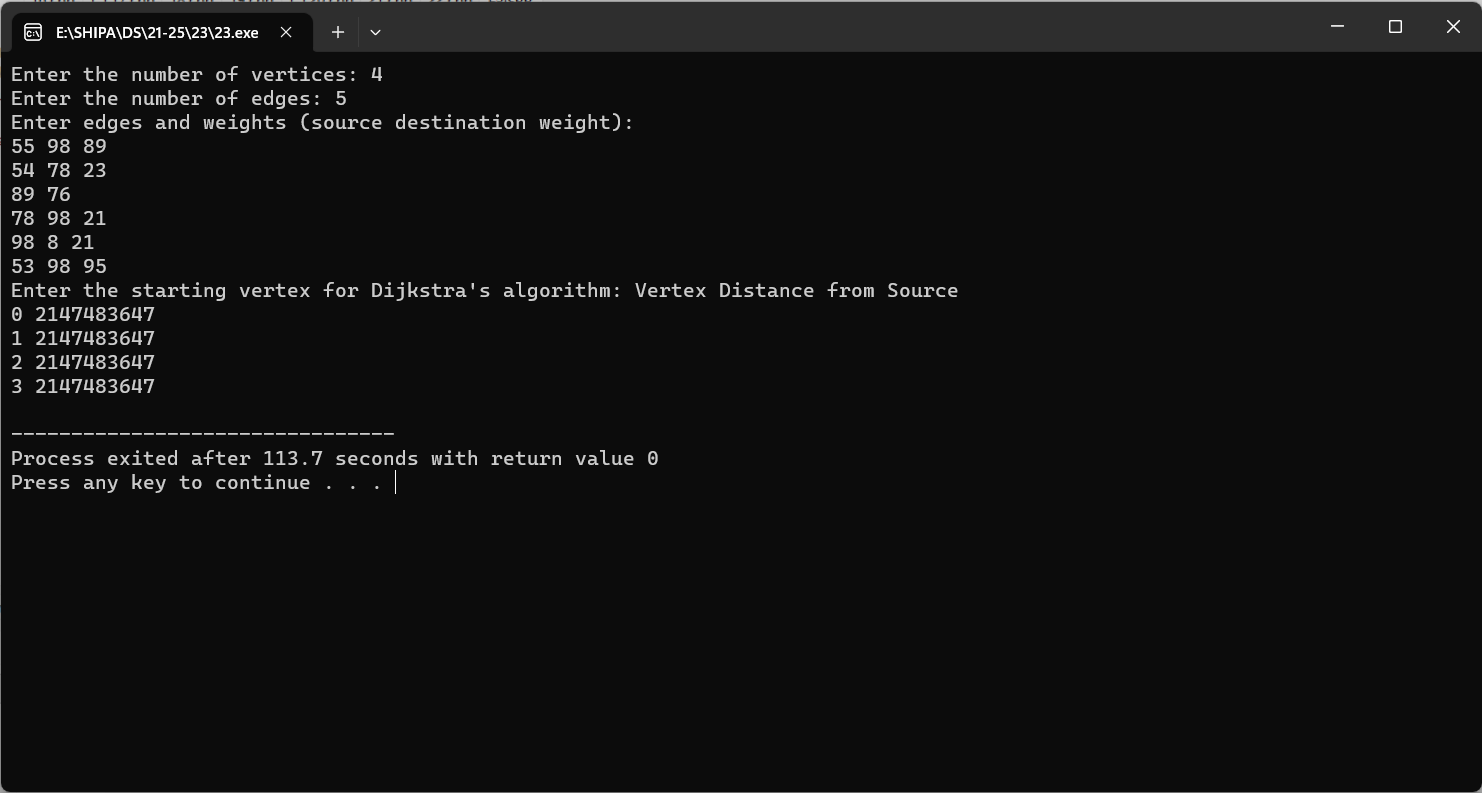
scanf("%d", &startVertex);

dijkstra(&graph, startVertex);

return 0;

}

Output:



**24. Implementation of Minimum Spanning Tree using Prim’s Algorithm**

Input:

#include <stdio.h>

#include <limits.h>

#define MAX\_VERTICES 100

struct Graph {

int numVertices;

int adjMatrix[MAX\_VERTICES][MAX\_VERTICES];

};

void initializeGraph(struct Graph\* graph, int numVertices) {

graph->numVertices = numVertices;

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

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

graph->adjMatrix[i][j] = 0; // Initialize with no edges

}

}

}

void addEdge(struct Graph\* graph, int src, int dest, int weight) {

graph->adjMatrix[src][dest] = weight;

graph->adjMatrix[dest][src] = weight; // For undirected graph}

int minKey(int key[], int mstSet[], int numVertices) {

int min = INT\_MAX, minIndex;

for (int v = 0; v < numVertices; v++) {

if (!mstSet[v] && key[v] < min) {

min = key[v];

minIndex = v;

}

}

return minIndex;

}

void primMST(struct Graph\* graph) {

int parent[MAX\_VERTICES]; // To store the constructed MST

int key[MAX\_VERTICES];

// Key values used to pick minimum weight edge

int mstSet[MAX\_VERTICES]; // To represent set of vertices included in MST

for (int i = 0; i < graph->numVertices; i++) {

key[i] = INT\_MAX;

mstSet[i] = 0;

}

key[0] = 0;

// Start from the first vertex

parent[0] = -1; // First vertex is the root of MST

for (int count = 0; count < graph->numVertices - 1; count++) {

int u = minKey(key, mstSet, graph->numVertices);

mstSet[u] = 1;

for (int v = 0; v < graph->numVertices; v++) {

if (graph->adjMatrix[u][v] && !mstSet[v] &&

graph->adjMatrix[u][v] < key[v]) {

parent[v] = u;

key[v] = graph->adjMatrix[u][v];

}

}

}

printf("Edge Weight\n");

for (int i = 1; i < graph->numVertices; i++) {

printf("%d - %d

%d\n", parent[i], i, graph->adjMatrix[i][parent[i]]);

}}

int main() {

struct Graph graph;

int numVertices, numEdges;

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

scanf("%d", &numVertices);

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

scanf("%d", &numEdges);

initializeGraph(&graph, numVertices);

printf("Enter edges and weights (source destination weight):\n");

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

int src, dest, weight;

scanf("%d %d %d", &src, &dest, &weight);

addEdge(&graph, src, dest, weight);

addEdge(&graph, dest, src, weight); // For undirected graph

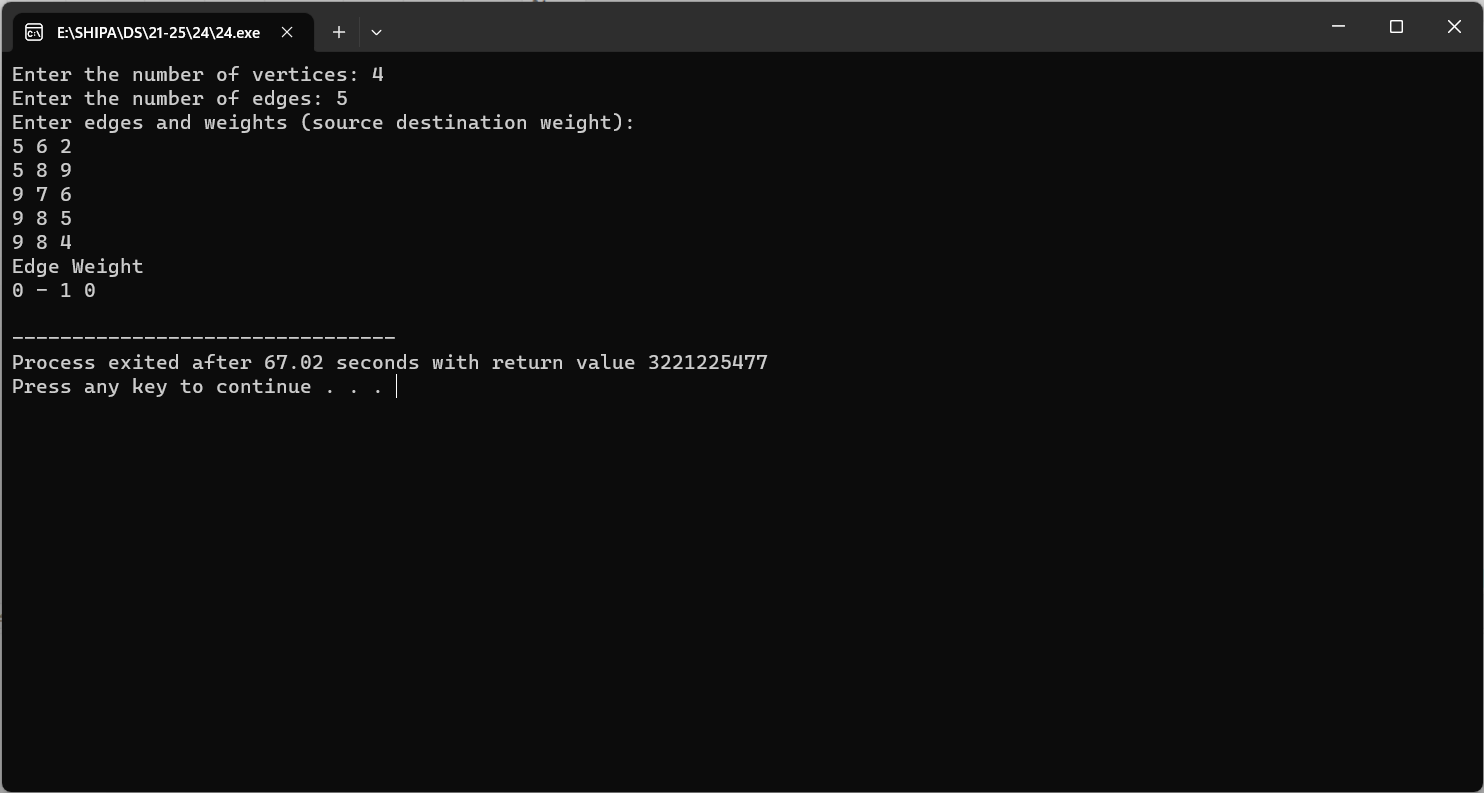
}

primMST(&graph);

return 0;

}

Output:



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

Input:

#include <stdio.h>

#include <stdlib.h>

#define MAX\_VERTICES 100

struct Edge {

int src, dest, weight;

};

struct Graph {

int numVertices, numEdges;

struct Edge edges[MAX\_VERTICES];

};

void initializeGraph(struct Graph\* graph, int numVertices, int numEdges) {

graph->numVertices = numVertices;

graph->numEdges = numEdges;

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

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

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

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

}

}

int find(int parent[], int vertex) {

if (parent[vertex] == -1) {

return vertex;

}

return find(parent, parent[vertex]);

}

void unionSets(int parent[], int x, int y) {

int xroot = find(parent, x);

int yroot = find(parent, y);

parent[xroot] = yroot;

}

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

return ((struct Edge\*)a)->weight - ((struct Edge\*)b)->weight;

}void kruskalMST(struct Graph\* graph) {

int parent[MAX\_VERTICES];

for (int i = 0; i < graph->numVertices; i++) {

parent[i] = -1;

}

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

printf("Edge Weight\n");

for (int i = 0; i < graph->numEdges; i++) {

int srcRoot = find(parent, graph->edges[i].src);

int destRoot = find(parent, graph->edges[i].dest);

if (srcRoot != destRoot) {

printf("%d - %d

%d\n", graph->edges[i].src, graph->edges[i].dest,

graph->edges[i].weight);

unionSets(parent, srcRoot, destRoot);

}

}

}

int main() {

struct Graph graph;

int numVertices, numEdges;

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

scanf("%d", &numVertices);

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

scanf("%d", &numEdges);

initializeGraph(&graph, numVertices, numEdges);

printf("Enter edges and weights (source destination weight):\n");

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

scanf("%d %d %d", &graph.edges[i].src, &graph.edges[i].dest, &graph.edges[i].weight);

}

kruskalMST(&graph);

return 0;

}

Output:

