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

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

#define MAX\_NODES 100

#define INF INT\_MAX

// Function to find the vertex with minimum key value

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

int min = INF, min\_index;

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

if (mstSet[v] == 0 && key[v] < min) {

min = key[v];

min\_index = v;

}

}

return min\_index;

}

// Function to print the constructed MST

void printMST(int parent[], int graph[MAX\_NODES][MAX\_NODES], int n) {

int totalWeight = 0;

printf("\n=== Minimum Spanning Tree (Prim's Algorithm) ===\n");

printf("Edge \t\tWeight\n");

printf("-----------------------\n");

for (int i = 1; i < n; i++) {

printf("%d - %d \t\t%d\n", parent[i], i, graph[i][parent[i]]);

totalWeight += graph[i][parent[i]];

}

printf("-----------------------\n");

printf("Total Weight: %d\n", totalWeight);

}

// Prim's algorithm implementation

void primMST(int graph[MAX\_NODES][MAX\_NODES], int n) {

int parent[MAX\_NODES]; // Array to store constructed MST

int key[MAX\_NODES]; // Key values used to pick minimum weight edge

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

// Initialize all keys as INFINITE

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

key[i] = INF;

mstSet[i] = 0;

}

// Always include first vertex in MST

key[0] = 0; // Make key 0 so this vertex is picked first

parent[0] = -1; // First node is always root of MST

// The MST will have n vertices

for (int count = 0; count < n - 1; count++) {

// Pick the minimum key vertex from the set of vertices not yet included in MST

int u = minKey(key, mstSet, n);

// Add the picked vertex to the MST set

mstSet[u] = 1;

// Update key value and parent index of adjacent vertices

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

// Update key only if:

// 1. graph[u][v] is non-zero (edge exists)

// 2. v is not yet included in MST

// 3. graph[u][v] is smaller than current key of v

if (graph[u][v] && mstSet[v] == 0 && graph[u][v] < key[v]) {

parent[v] = u;

key[v] = graph[u][v];

}

}

}

// Print the constructed MST

printMST(parent, graph, n);

}

// Function to display the graph matrix

void displayGraph(int graph[MAX\_NODES][MAX\_NODES], int n) {

printf("\nGraph Matrix (0 means no edge):\n");

printf(" ");

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

printf("%4d", i);

}

printf("\n");

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

printf("%4d:", i);

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

if (graph[i][j] == INF || graph[i][j] == 0) {

printf(" -");

} else {

printf("%4d", graph[i][j]);

}

}

printf("\n");

}

}

// Function to initialize graph with no edges

void initializeGraph(int graph[MAX\_NODES][MAX\_NODES]) {

for (int i = 0; i < MAX\_NODES; i++) {

for (int j = 0; j < MAX\_NODES; j++) {

graph[i][j] = 0;

}

}

}

int main() {

int n, edges;

int graph[MAX\_NODES][MAX\_NODES];

printf("=== Prim's Minimum Spanning Tree Algorithm ===\n");

initializeGraph(graph);

printf("Enter number of nodes in the graph: ");

scanf("%d", &n);

if (n <= 0 || n > MAX\_NODES) {

printf("Invalid number of nodes! (1-%d)\n", MAX\_NODES);

return 1;

}

printf("Enter number of edges: ");

scanf("%d", &edges);

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

printf("Note: Enter weight 0 if no direct connection\n");

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

int u, v, w;

printf("Edge %d: ", i + 1);

scanf("%d %d %d", &u, &v, &w);

if (u < 0 || u >= n || v < 0 || v >= n) {

printf("Invalid node! Nodes must be between 0 and %d\n", n - 1);

i--;

continue;

}

if (w < 0) {

printf("Weight cannot be negative!\n");

i--;

continue;

}

// For undirected graph

graph[u][v] = w;

graph[v][u] = w;

}

int choice;

do {

printf("\n=== Menu ===\n");

printf("1. Find Minimum Spanning Tree\n");

printf("2. Display graph matrix\n");

printf("3. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

primMST(graph, n);

break;

case 2:

displayGraph(graph, n);

break;

case 3:

printf("Exiting program.\n");

break;

default:

printf("Invalid choice! Please try again.\n");

}

} while (choice != 3);

return 0;

}