**Name: Talha Abdullah Bangyal**

**Roll Number: 22F-3194**

**Section: BAI-3A**

**Lab: 15**

**--------------------------------------------------------------------**

**Task 1**

#include <iostream>

using namespace std;

void DFS(int node, int adjacencyMatrix[8][8], bool visited[8], int n) {

visited[node] = true;

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

if (adjacencyMatrix[node][i] == 1 && !visited[i]) {

DFS(i, adjacencyMatrix, visited, n);

}

}

}

void checkConnectivity(int adjacencyMatrix[8][8], int n) {

bool visited[8] = { false };

int isolatedGroups = 0;

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

if (!visited[i]) {

isolatedGroups++;

DFS(i, adjacencyMatrix, visited, n);

}

}

if (isolatedGroups == 1) {

cout << "Network is fully connected." << endl;

}

else {

cout << "Network is not fully connected. Number of isolated groups: " << isolatedGroups << endl;

}

}

int main() {

int n = 8;

int adjacencyMatrix[8][8] = { 0 };

int edges[][2] = { {1, 2}, {2, 3}, {4, 5}, {6, 7} };

int edgeCount = sizeof(edges) / sizeof(edges[0]);

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

int u = edges[i][0] - 1;

int v = edges[i][1] - 1;

adjacencyMatrix[u][v] = 1;

adjacencyMatrix[v][u] = 1;

}

checkConnectivity(adjacencyMatrix, n);

return 0;

}



**Task 2**

#include <iostream>

using namespace std;

void findShortestPath(int start, int end, int adjacencyList[6][6], int edgesPerNode[6], int n) {

bool visited[6] = { false };

int previous[6] = { -1, -1, -1, -1, -1, -1 };

int queue[6], front = 0, rear = 0;

queue[rear++] = start;

visited[start] = true;

while (front < rear) {

int current = queue[front++];

for (int i = 0; i < edgesPerNode[current]; i++) {

int neighbor = adjacencyList[current][i];

if (!visited[neighbor]) {

visited[neighbor] = true;

previous[neighbor] = current;

queue[rear++] = neighbor;

if (neighbor == end) break;

}

}

}

if (!visited[end]) {

cout << "No path exists between warehouses " << start + 1 << " and " << end + 1 << "." << endl;

return;

}

int path[6], pathLength = 0, currentNode = end;

while (currentNode != -1) {

path[pathLength++] = currentNode;

currentNode = previous[currentNode];

}

cout << "Shortest route length: " << pathLength - 1 << " and Route: ";

for (int i = pathLength - 1; i >= 0; i--) {

cout << path[i] + 1;

if (i > 0) cout << " -> ";

}

cout << endl;

}

int main() {

int n = 6;

int adjacencyList[6][6] = { 0 };

int edgesPerNode[6] = { 0 };

int edges[][2] = { {1, 2}, {2, 3}, {3, 4}, {4, 5}, {5, 6} };

int edgeCount = sizeof(edges) / sizeof(edges[0]);

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

int u = edges[i][0] - 1;

int v = edges[i][1] - 1;

adjacencyList[u][edgesPerNode[u]++] = v;

adjacencyList[v][edgesPerNode[v]++] = u;

}

int start = 0, end = 4;

findShortestPath(start, end, adjacencyList, edgesPerNode, n);

return 0;

}



**Task 3**

#include <iostream>

using namespace std;

#define MAX\_EDGES 100

struct Edge {

int u, v, weight;

};

void sortEdges(Edge edges[], int edgeCount) {

for (int i = 0; i < edgeCount - 1; i++) {

for (int j = 0; j < edgeCount - i - 1; j++) {

if (edges[j].weight > edges[j + 1].weight) {

Edge temp = edges[j];

edges[j] = edges[j + 1];

edges[j + 1] = temp;

}

}

}

}

int findParent(int node, int parent[]) {

while (node != parent[node]) {

node = parent[node];

}

return node;

}

void unionSets(int u, int v, int parent[], int rank[]) {

int rootU = findParent(u, parent);

int rootV = findParent(v, parent);

if (rank[rootU] > rank[rootV]) {

parent[rootV] = rootU;

}

else if (rank[rootU] < rank[rootV]) {

parent[rootU] = rootV;

}

else {

parent[rootV] = rootU;

rank[rootU]++;

}

}

void kruskal(int n, Edge edges[], int edgeCount) {

int parent[n], rank[n], totalCost = 0, mstEdges = 0;

Edge mst[MAX\_EDGES];

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

parent[i] = i;

rank[i] = 0;

}

sortEdges(edges, edgeCount);

for (int i = 0; i < edgeCount && mstEdges < n - 1; i++) {

int u = edges[i].u;

int v = edges[i].v;

int weight = edges[i].weight;

if (findParent(u, parent) != findParent(v, parent)) {

mst[mstEdges++] = edges[i];

totalCost += weight;

unionSets(u, v, parent, rank);

}

}

cout << "Total cost of the MST: " << totalCost << endl;

cout << "Selected connections:" << endl;

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

cout << mst[i].u + 1 << " - " << mst[i].v + 1 << " (Cost: " << mst[i].weight << ")" << endl;

}

}

int main() {

int n = 5; // Number of buildings

int edgeCount = 7;

Edge edges[] = {

{0, 1, 5}, {0, 3, 6}, {1, 3, 4}, {1, 2, 3}, {2, 3, 2},

{2, 4, 7}, {3, 4, 8}

};

kruskal(n, edges, edgeCount);

return 0;

}

**Task 4**

#include <iostream>

using namespace std;

#define INF 100000

void prim(int cost[5][5], int n, int start) {

bool selected[5] = { false };

int minCost = 0;

int edges = 0;

selected[start] = true;

cout << "Selected power lines:" << endl;

while (edges < n - 1) {

int minWeight = INF;

int u = -1, v = -1;

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

if (selected[i]) {

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

if (!selected[j] && cost[i][j] != 0 && cost[i][j] < minWeight) {

minWeight = cost[i][j];

u = i;

v = j;

}

}

}

}

if (u != -1 && v != -1) {

cout << u + 1 << " - " << v + 1 << " (Cost: " << minWeight << ")" << endl;

minCost += minWeight;

selected[v] = true;

edges++;

}

}

cout << "Minimum cost: " << minCost << endl;

}

int main() {

int n = 5;

int cost[5][5] = {

{0, 2, 3, 0, 7},

{2, 0, 0, 6, 4},

{3, 0, 0, 5, 0},

{0, 6, 5, 0, 1},

{7, 4, 0, 1, 0}

};

int start = 0;

prim(cost, n, start);

return 0;

}

