/\*

Problem Statement: Represent a given graph using adjacency matrix/list to perform DFS and using adjacency list to perform BFS. Use the map of the area around the college as the graph. Identify the prominent land marks as nodes and perform DFS and BFS on that.

\*/

#include <iostream>

#include <vector>

#include <queue>

#include <map>

using namespace std;

// Number of nodes (landmarks)

const int N = 6;

// For DFS using adjacency matrix

int adjMatrix[N][N];

// For BFS and DFS using adjacency list

vector<vector<int>> adjList(N);

// Visited arrays for DFS

bool visitedDFSMatrix[N];

bool visitedDFSList[N];

// Landmark mapping

map<int, string> landmarks = {

{0, "College Main Gate"},

{1, "Library"},

{2, "Canteen"},

{3, "Auditorium"},

{4, "Hostel"},

{5, "Playground"}

};

// Function to add edges to both matrix and list

void addEdge(int u, int v) {

// For adjacency matrix

adjMatrix[u][v] = 1;

adjMatrix[v][u] = 1;

// For adjacency list

adjList[u].push\_back(v);

adjList[v].push\_back(u);

}

// DFS using adjacency matrix

void DFS\_Matrix(int node) {

visitedDFSMatrix[node] = true;

cout << landmarks[node] << " -> ";

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

if (adjMatrix[node][i] && !visitedDFSMatrix[i]) {

DFS\_Matrix(i);

}

}

}

// DFS using adjacency list

void DFS\_List(int node) {

visitedDFSList[node] = true;

cout << landmarks[node] << " -> ";

for (int neighbor : adjList[node]) {

if (!visitedDFSList[neighbor]) {

DFS\_List(neighbor);

}

}

}

// BFS using adjacency list

void BFS(int start) {

vector<bool> visited(N, false);

queue<int> q;

visited[start] = true;

q.push(start);

while (!q.empty()) {

int node = q.front();

q.pop();

cout << landmarks[node] << " -> ";

for (int neighbor : adjList[node]) {

if (!visited[neighbor]) {

visited[neighbor] = true;

q.push(neighbor);

}

}

}

}

int main() {

// Build the graph

addEdge(0, 1); // College Main Gate - Library

addEdge(0, 2); // College Main Gate - Canteen

addEdge(1, 3); // Library - Auditorium

addEdge(2, 5); // Canteen - Playground

addEdge(3, 4); // Auditorium - Hostel

addEdge(4, 5); // Hostel - Playground

cout << "DFS using Adjacency Matrix:\n";

DFS\_Matrix(0);

cout << "END\n\n";

cout << "DFS using Adjacency List:\n";

DFS\_List(0);

cout << "END\n\n";

cout << "BFS using Adjacency List:\n";

BFS(0);

cout << "END\n";

return 0;

}