NAME: Abhinav Sharma

UID: 22BCS11022

SECTION: KPIT_901

DAY 7

1. Find if Path Exists in Graph

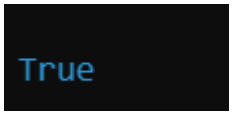
Question:
You are given an undirected graph represented by n nodes and an array of edges where each edge is represented by a pair [u, v]. Check if a path exists between two given nodes source and destination.

Code:

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

bool validPath(int n, vector<vector<int>> &edges, int source, int destination) {
    vector<vector<int>> adj(n);
    for (auto &edge : edges) {
        adj[edge[0]].push_back(edge[1]);
        adj[edge[1]].push_back(edge[0]);
    }
    vector<bool> visited(n, false);
    queue<int> q;
    q.push(source);
    visited[source] = true;
    while (!q.empty()) {
        int node = q.front();
        q.pop();
        if (node == destination) return true;
        for (int neighbor : adj[node]) {
            if (!visited[neighbor]) {
```

```
            visited[neighbor] = true;
            q.push(neighbor);
        }
    }
}
return false;
}

int main() {
    vector<vector<int>> edges = {{0, 1}, {1, 2}, {2, 0}};
    int n = 3, source = 0, destination = 2;
    cout << (validPath(n, edges, source, destination) ? "True" : "False") << endl;
    return 0;
}
```

Output:



2. BFS of Graph

Question:
Given a connected graph with V vertices and its adjacency list, perform a Breadth-First Search (BFS) traversal starting from vertex 0 and return the BFS traversal order.

Code:

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

vector<int> bfsOfGraph(int V, vector<vector<int>> &adj) {
    vector<int> bfs;
    vector<bool> visited(V, false);
    queue<int> q;
    q.push(0);
```

```
    visited[0] = true;
    while (!q.empty()) {
        int node = q.front();
        q.pop();
        bfs.push_back(node);
        for (int neighbor : adj[node]) {
            if (!visited[neighbor]) {
                visited[neighbor] = true;
                q.push(neighbor);
            }
        }
    }
    return bfs;
}

int main() {
    int V = 5;
    vector<vector<int>> adj = {
        {1, 2, 3},
        {0},
        {0, 4},
        {0},
        {2}
    };
    vector<int> bfs = bfsOfGraph(V, adj);
    for (int v : bfs) cout << v << " ";
    cout << endl;
    return 0;
}
```

Output :

3. DFS of Graph

Question:
Given a connected graph with V vertices and its adjacency list, perform a Depth-First Search (DFS) traversal starting from vertex 0 and return the DFS traversal order.

Code:

```
#include <iostream>

#include <vector>

using namespace std;


void dfsUtil(int node, vector<vector<int>> &adj, vector<bool> &visited, vector<int> &result) {

    visited[node] = true;

    result.push_back(node);

    for (int neighbor : adj[node]) {

        if (!visited[neighbor]) {

            dfsUtil(neighbor, adj, visited, result);

        }

    }

}


vector<int> dfsOfGraph(int V, vector<vector<int>> &adj) {

    vector<int> result;

    vector<bool> visited(V, false);

    dfsUtil(0, adj, visited, result);

    return result;

}


int main() {
```

```
    int V = 5;
    vector<vector<int>> adj = {
        {1, 2, 3},
        {0},
        {0, 4},
        {0},
        {2}
    };
    vector<int> dfs = dfsOfGraph(V, adj);
    for (int v : dfs) cout << v << " ";
    cout << endl;
    return 0;
}
```

Output:

```
0 1 2 4 3
```

4. 01 Matrix

Question:
You are given a binary matrix where 0 represents land and 1 represents water. Calculate the distance of each cell to the nearest 0 cell. Return the updated matrix.

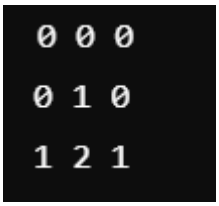
Code:

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;


vector<vector<int>> updateMatrix(vector<vector<int>> &mat) {
    int rows = mat.size(), cols = mat[0].size();
    vector<vector<int>> dist(rows, vector<int>(cols, INT_MAX));
    queue<pair<int, int>> q;
    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < cols; ++j) {
```

```
            if (mat[i][j] == 0) {
                dist[i][j] = 0;
                q.push({i, j});
            }
        }
    }
    vector<pair<int, int>> directions = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};
    while (!q.empty()) {
        auto [x, y] = q.front();
        q.pop();
        for (auto [dx, dy] : directions) {
            int newX = x + dx, newY = y + dy;
            if (newX >= 0 && newX < rows && newY >= 0 && newY < cols) {
                if (dist[newX][newY] > dist[x][y] + 1) {
                    dist[newX][newY] = dist[x][y] + 1;
                    q.push({newX, newY});
                }
            }
        }
    }
    return dist;
}

int main() {
    vector<vector<int>> mat = {
        {0, 0, 0},
        {0, 1, 0},
        {1, 1, 1}
    };
    vector<vector<int>> result = updateMatrix(mat);
    for (auto row : result) {
```

```
        for (int cell : row) cout << cell << " ";
        cout << endl;
    }
    return 0;
}
```

Output:



5. Course Schedule II

Question:
Given the number of courses numCourses and an array prerequisites where each pair [a, b] indicates that you must take course b before a, return the order of courses to finish all the courses. If it's not possible, return an empty array.

Code:

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

vector<int> findOrder(int numCourses, vector<vector<int>> &prerequisites) {
    vector<vector<int>> adj(numCourses);
    vector<int> inDegree(numCourses, 0);
    for (auto &prerequisite : prerequisites) {
        adj[prerequisite[1]].push_back(prerequisite[0]);
        ++inDegree[prerequisite[0]];
    }
    queue<int> q;
    for (int i = 0; i < numCourses; ++i) {
        if (inDegree[i] == 0) q.push(i);
    }
```

```
    vector<int> order;
    while (!q.empty()) {
        int course = q.front();
        q.pop();
        order.push_back(course);
        for (int neighbor : adj[course]) {
            if (--inDegree[neighbor] == 0) q.push(neighbor);
        }
    }
    return order.size() == numCourses ? order : vector<int>();
}

int main() {
    int numCourses = 4;
    vector<vector<int>> prerequisites = {{1, 0}, {2, 0}, {3, 1}, {3, 2}};
    vector<int> order = findOrder(numCourses, prerequisites);
    for (int course : order) cout << course << " ";
    cout << endl;
    return 0;
}
```

Output: