Nitin Chaudhary
9921103163
F7
APS Lab Week 7

Q1. A Hamiltonian path, is a path in an undirected or directed graph that visits each vertex exactly once. Given an undirected graph the task is to check if a Hamiltonian path is present in it or not.

Example 1:

Input:

N = 4, M = 4

Edges[][]= { {1,2}, {2,3}, {3,4}, {2,4} }

Output:

1

Explanation:

There is a hamiltonian path:

1 -> 2 -> 3 -> 4

Example 2:

Input:

N = 4, M = 3

Edges[][] = { {1,2}, {2,3}, {2,4} }

Output:

0

Explanation:

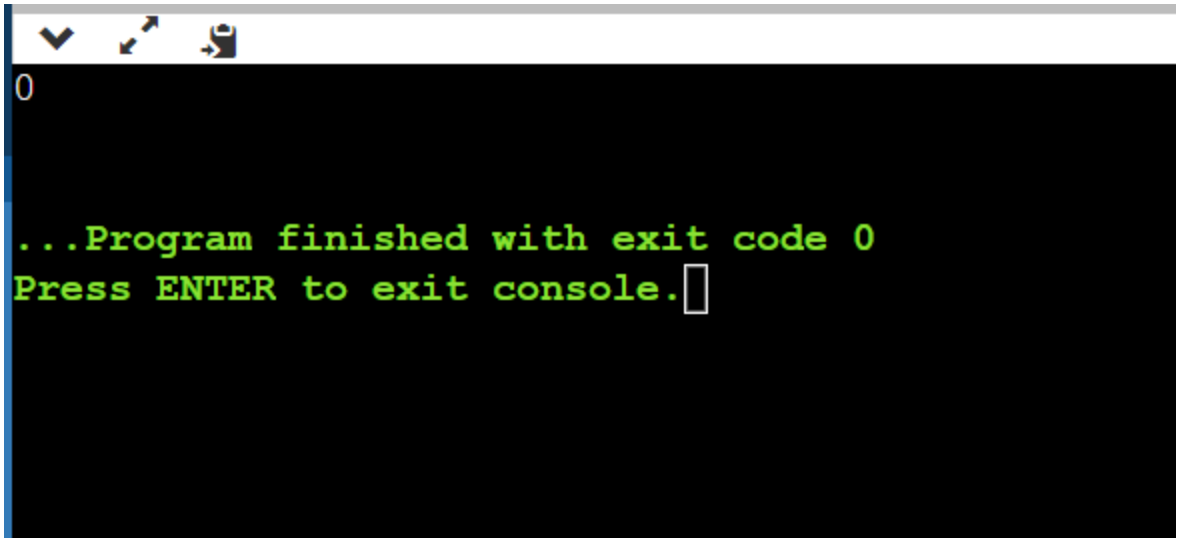
It can be proved that there is no hamiltonian path in the given graph

Code for the answer-

```
#include <iostream>
#include <vector>

using namespace std;

bool valid(vector<vector<int>>& graph, vector<int>& path, int pos, int v) {
    int n = graph.size();
    if (graph[path[pos-1]][v] == 0) {
        return false;
    }
    for (int i = 0; i < pos; i++) {
        if (path[i] == v) {
            return false;
        }
    }
    return true;
}

bool Path_helper(vector<vector<int>>& graph, vector<int>& path, int pos) {
    int n = graph.size();
    if (pos == n) {
        return true;
    }
    for (int v = 1; v < n; v++) {
```

```
        if (valid(graph, path, pos, v)) {
            path[pos] = v;
            if (Path_helper(graph, path, pos+1)) {
                return true;
            }
            path[pos] = -1;
        }
    }
    return false;
}
//This function will return true if path exists other wise this will return false;
bool hamiltonian_path(vector<vector<int>>& graph) {
    int n = graph.size();
    vector<int> path(n, -1);
    path[0] = 0;
    if (!Path_helper(graph, path, 1)) return false;
    return true;
}


int main() {
    vector<vector<int>> graph = {{1,2},{2,3},{2,4}};
    if(hamiltonian_path(graph))cout<<1<<endl;
    else cout<<0<<endl;
    return 0;
}
```

Output of the above program-



Q2. Given a dictionary of distinct words and an M x N board where every cell has one character. Find all possible words from the dictionary that can be formed by a sequence of adjacent characters on the board. We can move to any of 8 adjacent characters

Note: While forming a word we can move to any of the 8 adjacent cells. A cell can be used only once in one word.

Example 1:

Input:

N = 1

dictionary = {"CAT"}

R = 3, C = 3

board = {{C,A,P},{A,N,D},{T,I,E}}

Output:

CAT

Explanation:

C A P

A N D

T I E

Words we got is denoted using same color.

Example 2:

Input:

N = 4

dictionary = {"GEEKS","FOR","QUIZ","GO"}

R = 3, C = 3

board = {{G,I,Z},{U,E,K},{Q,S,E}}

Output:

GEEKS QUIZ

Explanation:

G I Z

U E K

Q S E

Words we got is denoted using same color.

Code:

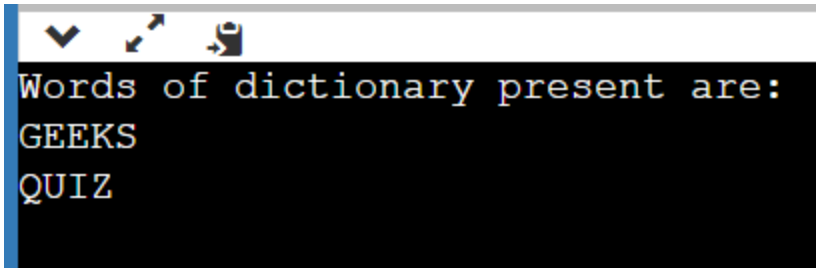
```
#include<bits/stdc++.h>
using namespace std;

bool dfs(vector<vector<char> >& board, string &s, int i, int j, int n, int m, int idx){

    if(i<0 || i>=n||j<0||j>=m){
        return false;
    }
```

```
        if(s[idx]!= board[i][j]){
                return false;
        }
        if(idx == s.size()-1){
                return true;
        }

        char temp = board[i][j];
        board[i][j]='*';

        bool a = dfs(board,s,i,j+1,n,m,idx+1);
        bool b= dfs(board,s,i,j-1,n,m,idx+1);
        bool c = dfs(board,s,i+1,j,n,m,idx+1);
        bool d = dfs(board,s,i-1,j,n,m,idx+1);
        bool e = dfs(board,s,i+1,j+1,n,m,idx+1);
        bool f = dfs(board,s,i-1,j+1,n,m,idx+1);
        bool g = dfs(board,s,i+1,j-1,n,m,idx+1);
        bool h = dfs(board,s,i-1,j-1,n,m,idx+1);

        board[i][j]=temp;
        return a||b||c||e||f||g||h||d;


}
void wordBoard(vector<vector<char> >& board, vector<string>& dictionary) {
                int n= board.size();
```

```
        int m = board[0].size();
        vector<string> ans;
        set<string> store;
            for(int i=0;i<dictionary.size();i++){
                string s = dictionary[i];
                int l = s.size();
                for(int j = 0 ; j < n;j++){
                    for(int k=0;k<m;k++){
                        if(dfs(board,s,j,k,n,m,0)){
                            store.insert(s);
                        }
                    }
                }
            }

            for(auto i:store){
                cout<<i<<"\n";
            }

            return ;

}

int main()
{
//     vector<vector<char>> board;
//         vector<string> dictionary;
```

```
//        int N;
//    cin>>N;
//    string s;
//        while(N--){
//            cin>>s;
//            dictionary.push_back(s);
//        }
//        int R,C;
//        cin>>R>>C;
//        char temp;
//        vector<char> t;
//        for(int i=0;i<R;i++){
//            for(int j=0;j<C;j++){
//                cin>>temp;
//                t.push_back(temp);
//            }
//            board.push_back(t);
//        }
vector<vector<char>> board{ { 'G', 'I', 'Z' },{ 'U', 'E', 'K' },{ 'Q', 'S', 'E' } };
vector<string> dictionary{ "GEEKS", "FOR", "QUIZ", "GO" };
cout << "Words of dictionary present are:\n";
wordBoard(board,dictionary);
return 0;
}
```

Output:



Q4. Given an undirected graph and an integer M. The task is to determine if the graph can be colored with at most M colors such that no two adjacent vertices of the graph are colored with the same color. Here coloring of a graph means the assignment of colors to all vertices. Print 1 if it is possible to colour vertices and 0 otherwise.

Example 1:

Input:

N = 4

M = 3

E = 5

Edges[] = {(0,1),(1,2),(2,3),(3,0),(0,2)}

Output: 1

Explanation: It is possible to colour the

given graph using 3 colours.

Example 2:

Input:

N = 3

M = 2

E = 3

Edges[] = {(0,1),(1,2),(0,2)}

Output: 0

Code:

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

vector<vector<int>> graph;
vector<int> colors;

bool isSafe(int node, int color) {
    for (int neighbor : graph[node]) {
        if (colors[neighbor] == color) {
            return false;
        }
    }
    return true;
}

bool canColorGraph(int node, int M) {
    if (node == graph.size()) {
        return true;
    }
    for (int i = 1; i <= M; i++) {
        if (isSafe(node, i)) {
            colors[node] = i;
            if (canColorGraph(node+1, M)) {
                return true;
            }
            colors[node] = 0;
        }
    }
    return false;
}

int main() {
    int N, M, E;
    cin >> N >> M >> E;
    graph.resize(N);
    colors.resize(N, 0);
    for (int i = 0; i < E; i++) {
        int u, v;
```

```
        cin >> u >> v;
        graph[u].push_back(v);
        graph[v].push_back(u);
    }
    if (canColorGraph(0, M)) {
        cout << "\nThe answer is:1\n";
    } else {
        cout << "\nThe answer is:0\n";
    }
    return 0;
}
```