Name-Mayank Srivastava batch-F6 enroll -9922103143

Ans 1-

#include<bits/stdc++.h>

using namespace std;

const int MAX = 20;

vector<int> adj[MAX];

bool visited[MAX];

bool checkHamiltonian(int u, int n, int rem) {

if (rem == 0)

return true;

visited[u] = true;

for(int v : adj[u]) {

if (!visited[v]) {

if (checkHamiltonian(v, n, rem - 1)) return true;

}

}

visited[u] = false;

return false;

}

bool hasHam(int n, int m, vector<vector<int>>&edges) {

memset(visited, false, sizeof(visited));

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

int u = edges[i][0];

int v = edges[i][1];

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

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

}

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

if (checkHamiltonian(i, n, n - 1))

return true;

}

return false;

}

int main() { int N, M;

cout << "Enter the number of vertices: "; cin >> N;

cout << "Enter the number of edges: "; cin >> M;

vector<vector<int>> Edges(M, vector<int>(2));

cout << "Enter the edges" << endl;

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

cin >> Edges[i][0] >> Edges[i][1];

}

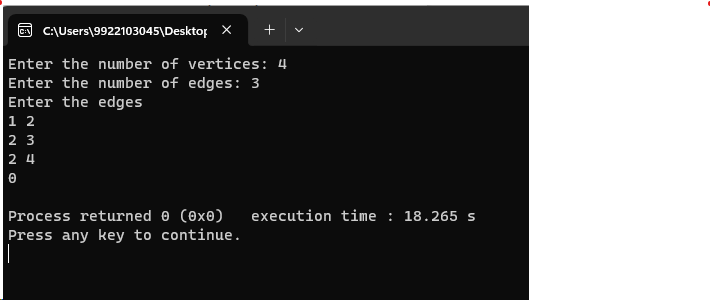
if (hasHam(N, M, Edges))

cout <<"1" << endl;

else

cout << "0" << endl;

return 0;

}

ANS 2-#include <iostream>

#include <vector>

#include <cstring>

using namespace std;

void findWordsUtil(char \*\*board, bool \*\*visited, int M, int N, int i, int j, string &str, vector<string> &dictionary) {

visited[i][j] = true;

str = str + board[i][j];

for (const string &word : dictionary) {

if (str.compare(word) == 0) {

cout << str << endl;

break;

}

}

for (int row = i - 1; row <= i + 1 && row < M; row++) {

for (int col = j - 1; col <= j + 1 && col < N; col++) {

if (row >= 0 && col >= 0 && !visited[row][col]) {

findWordsUtil(board, visited, M, N, row, col, str, dictionary);

}

}

}

str.erase(str.length() - 1);

visited[i][j] = false;

}

void findWords(char \*\*board, int M, int N, vector<string> &dictionary) {

bool \*\*visited = new bool \*[M];

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

visited[i] = new bool[N]{false};

}

string str = "";

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

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

findWordsUtil(board, visited, M, N, i, j, str, dictionary);

}

}

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

delete[] visited[i];

}

delete[] visited;

}

int main() {

int M, N;

cout << "Enter the number of rows (M): ";

cin >> M;

cout << "Enter the number of columns (N): ";

cin >> N;

int dictSize;

cout << "Enter the size of the dictionary: ";

cin >> dictSize;

vector<string> dictionary(dictSize);

cout << "Enter the words of the dictionary, one by one:\n";

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

cin >> dictionary[i];

}

char \*\*board = new char \*[M];

cout << "Enter the elements of the board matrix (" << M << "x" << N << "):\n";

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

board[i] = new char[N];

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

cin >> board[i][j];

}

}

cout << "Words present in the board are:\n";

findWords(board, M, N, dictionary);

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

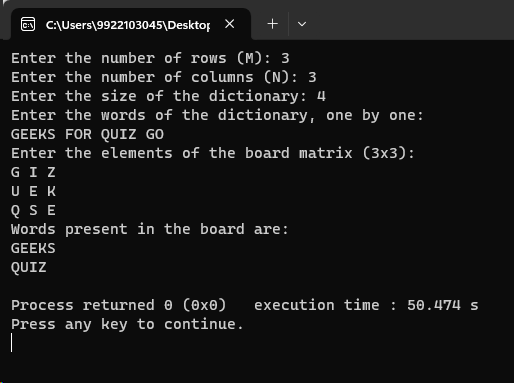
delete[] board[i];

}

delete[] board;

return 0;

}



Ans3,4-

#include<iostream>

#include<vector>

using namespace std;

bool isSafe(int node, vector<int> color, vector<vector<int> > graph, int N, int col)

{

for(int k=0; k<N; k++)

{

if(k!=node && graph[k][node]==1 && color[k]==col)

return false;

}

return true;

}

bool solve(int node, vector<vector<int> > graph, vector<int> color, int N, int c)

{

if(node==N)

return true;

for(int i=1; i<=c; i++)

{

if(isSafe(node, color, graph, N,i))

{

color[node] = i;

if(solve(node+1, graph, color,N,c))

return true;

color[node] = 0;

}

}

return false;

}

bool graphColoring( vector<vector<int> > graph, int N, int c)

{

vector<int> color(N,0);

if(solve(0,graph,color,N,c))

return true;

return false;

}

int main()

{

int N,M;

cout << "Enter the number of vertices: ";

cin >> N;

cout << "Enter the number of edges: ";

cin >> M;

vector<vector<int>> Edges(M, vector<int>(2));

int c;

cout<<"Enter number of colors: ";

cin>>c;

cout << "Enter the edges " << endl;

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

cin >> Edges[i][0] >> Edges[i][1];

}

vector<vector<int> > Adj(N+1, vector<int>(N+1,0));

for(int i=0; i<M; i++)

{

int u = Edges[i][0];

int v = Edges[i][1];

Adj[u][v]=1;

Adj[v][u]=1;

}

if(graphColoring(Adj,N,c))

cout<<"1";

else

cout<<"0";

return 0;

}

