AlgorithmsandProblem-SolvingLab(15B17CI471) EVEN 2025

Week–3

**Q1.Givena maze inthe formofabinaryrectangularmatrix,findthe shortestpath’s length in the maze from a given source to a given destination. The path can only be constructed out of cells having value 1,and at any moment,we can only move one step in one of the four directions.**

**Thevalidmovesare:**

**GoTop:(x,y)——>(x–1,y)**

**GoLeft:(x,y)——>(x,y–1)**

**GoDown:(x,y)——>(x+1,y)**

**GoRight:(x,y)——>(x,y+1)**

**Forexample,considerthefollowingbinarymatrix.Ifsource=(0,0)anddestination**

**=(7,5),theshortestpath fromsourceto destinationhaslength 12.**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **[1** | **1** | **1** | **1** | **1** | **0** | **0** | **1** | **1** | **1]** |
| **[0** | **1** | **1** | **1** | **1** | **1** | **0** | **1** | **0** | **1]** |
| **[0** | **0** | **1** | **0** | **1** | **1** | **1** | **0** | **0** | **1]** |
| **[1** | **0** | **1** | **1** | **1** | **0** | **1** | **1** | **0** | **1]** |
| **[0** | **0** | **0** | **1** | **0** | **0** | **0** | **1** | **0** | **1]** |
| **[1** | **0** | **1** | **1** | **1** | **0** | **0** | **1** | **1** | **0]** |
| **[0** | **0** | **0** | **0** | **1** | **0** | **0** | **1** | **0** | **1]** |
| **[0** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **0** | **0]** |
| **[1** | **1** | **1** | **1** | **1** | **0** | **0** | **1** | **1** | **1]** |
| **[0** | **0** | **1** | **0** | **0** | **1** | **1** | **0** | **0** | **1]** |

#include <iostream>

#include <queue>

#include <vector>

using namespace std;

struct Point

{

int x,y;

};

bool isValid(int x,int y,int rows,int cols,vector<vector<int>> &matrix,vector<vector<bool>> &visited)

{

return x>=0&&x<rows&&y>=0&&y<cols&&matrix[x][y]==1&&!visited[x][y];

}

int shortestPath(vector<vector<int>> &matrix,Point source,Point destination)

{

int rows=matrix.size();

int cols=matrix[0].size();

if (matrix[source.x][source.y]==0||matrix[destination.x][destination.y]==0)

return -1;

vector<vector<bool>> visited(rows,vector<bool>(cols,false));

queue<pair<Point,int>> q;

q.push({source,0});

visited[source.x][source.y]=true;

int dx[]={-1,0,1,0};

int dy[]={0,-1,0,1};

while (!q.empty())

{

auto current=q.front();

q.pop();

Point pt=current.first;

int distance=current.second;

if (pt.x==destination.x&&pt.y==destination.y)

return distance;

for (int i=0;i<4;i++)

{

int newX=pt.x +dx[i];

int newY=pt.y +dy[i];

if (isValid(newX,newY,rows,cols,matrix,visited))

{

visited[newX][newY]=true;

q.push({{newX,newY},distance +1});

}

}

}

return -1;

}

void displayMatrix(vector<vector<int>> &matrix)

{

for (const auto &row : matrix)

{

for (int cell : row)

cout << cell << " ";

cout << endl;

}

}

int main()

{

vector<vector<int>> matrix={

{1,1,1,1,0,0,1,1,1,1},

{0,1,1,1,1,0,1,0,0,1},

{0,0,1,0,1,1,1,1,0,1},

{1,0,1,0,0,0,0,1,0,1},

{0,0,1,1,1,1,0,1,0,1},

{1,0,0,0,0,1,0,1,1,1},

{1,1,1,1,1,1,0,0,0,1},

{0,0,0,0,0,1,1,1,1,1}};

Point source={0,0};

Point destination={7,5};

cout << "Matrix:" << endl;

displayMatrix(matrix);

int result=shortestPath(matrix,source,destination);

if (result != -1)

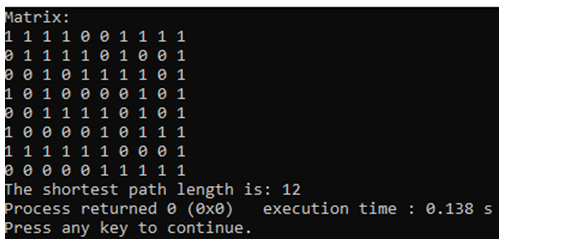
cout << "The shortest path length is: " << result;

else

cout << "No path exists.";

}

**Output :**

****

**Q2.Givenanarray nums ofdistinctintegers,return*allthepossiblepermutations*. You can return the answer in any order.**

# Example:

**Input:nums=[1,2,3]**

**Output:[[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]**

#include <iostream>

#include <vector>

using namespace std;

void generatePermutations(vector<int> &nums,int index,vector<vector<int>> &result)

{

if (index==nums.size())

{

result.push\_back(nums);

return;

}

for (int i=index;i<nums.size();i++)

{

swap(nums[index],nums[i]);

generatePermutations(nums,index +1,result);

swap(nums[index],nums[i]);

}

}

vector<vector<int>> permute(vector<int> &nums)

{

vector<vector<int>> result;

generatePermutations(nums,0,result);

return result;

}

int main()

{

int n;

cout<<"Input the size of the array : ";

cin>>n;

vector<int> nums(n);

cout<<"Input the elements : ";

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

cin>>nums[i];

vector<vector<int>> permutations=permute(nums);

cout << "All permutations:" << endl;

for (const auto &perm : permutations)

{

for (int num : perm)

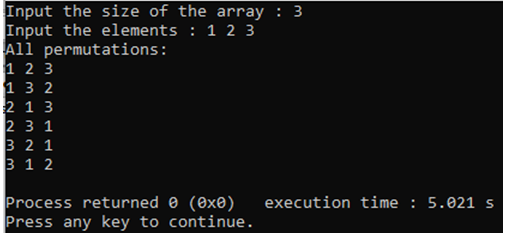
cout << num << " ";

cout << endl;

}

}

**Output :**

****

**Q3. The n-queens puzzle is the problem of placingn queens on an n x n chessboard such that no two queens attack each other.Given an integern,return *all distinct solutions to the n-queens puzzle*. You may return the answer in any order.Each solutioncontainsadistinctboardconfigurationofthen-queens'placement,where'Q'and'.'bothindicateaqueenandanemptyspace,respectively.**

**Input:n=4**

**Output:[[".Q..","...Q","Q...","..Q."],["..Q.","Q...","...Q",".Q.."]]**

**Explanation:Thereexisttwodistinctsolutionstothe4-queenspuzzle**

#include <iostream>

using namespace std;

class NQueens

{

int n;

char \*\*board;

bool isSafe(int row,int col)

{

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

if (board[row][i]=='Q')

return false;

for (int i=row,j=col;i>=0&&j>=0;i--,j--)

if (board[i][j]=='Q')

return false;

for (int i=row,j=col;i<n&&j>=0;i++,j--)

if (board[i][j]=='Q')

return false;

return true;

}

void solve(int col)

{

if (col==n)

{

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

{

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

cout << board[i][j];

cout << "\n";

}

cout << "\n";

return;

}

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

if (isSafe(i,col))

{

board[i][col]='Q';

solve(col +1);

board[i][col]='.';

}

}

public:

NQueens(int size) : n(size)

{

board=new char \*[n];

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

{

board[i]=new char[n];

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

board[i][j]='.';

}

}

~NQueens()

{

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

delete[] board[i];

delete[] board;

}

void solveNQueens()

{

solve(0);

}

};

int main()

{

int n;

cout<<"Input the value of n for n x n chessboard : ";

cin>>n;

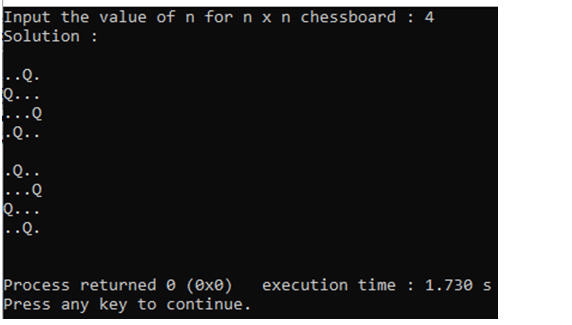
cout<<"Solution : "<<endl<<endl;

NQueens nq(n);

nq.solveNQueens();

}

**Output :**

**\**

**Q4.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.**

|  |  |
| --- | --- |
| **Example1:**  **Input:**  **N=4,M=4**  **Edges[][]={{1,2},{2,3},{3,4},{2,4}}**  **Output: 1 Explanation:**  **Thereisahamiltonianpath: 1 -> 2 -> 3 -> 4** | **Example2:**  **Input:**  **N=4,M=3**  **Edges[][]={{1,2},{2,3},{2,4}}**  **Output: 0 Explanation:**  **Itcanbeprovedthatthereisnohamiltonian path in the given graph** |

#include <iostream>

using namespace std;

class HamiltonianPath

{

int n,m;

bool \*\*adj;

bool \*visited;

int \*path;

bool dfs(int v,int count)

{

path[count-1]=v +1;

if (count==n)

{

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

cout << path[i] << (i==n-1 ? "\n" : " -> ");

return true;

}

visited[v]=true;

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

{

if (adj[v][i]&&!visited[i])

{

if (dfs(i,count +1))

return true;

}

}

visited[v]=false;

return false;

}

public:

HamiltonianPath(int vertices,int edges,int edgeList[][2]) : n(vertices),m(edges)

{

adj=new bool \*[n];

visited=new bool[n];

path=new int[n];

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

{

adj[i]=new bool[n];

visited[i]=false;

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

adj[i][j]=false;

}

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

{

int u=edgeList[i][0]-1,v=edgeList[i][1]-1;

adj[u][v]=adj[v][u]=true;

}

}

~HamiltonianPath()

{

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

delete[] adj[i];

delete[] adj;

delete[] visited;

delete[] path;

}

bool hasHamiltonianPath()

{

cout<<"Hamiltonian Path : ";

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

if (dfs(i,1))

return true;

cout<<"No Hamiltonian Path found";

return false;

}

};

int main()

{

int n,m;

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

cin>>n;

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

cin>>m;

int edges[m][2];

cout<<"Input the vertices connected to each other : ";

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

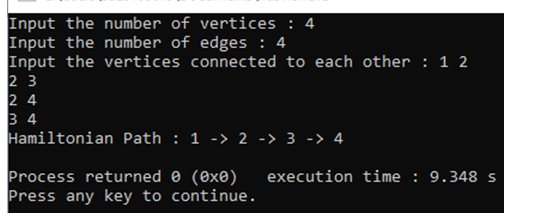
cin>>edges[i][0]>>edges[i][1];

HamiltonianPath hp(n,m,edges);

hp.hasHamiltonianPath();

}

**Output :**

****

**Q5.Given a dictionary,a method to do lookup in dictionary and a M x N board whereeverycellhasonecharacter.Findallpossiblewordsthatcanbeformedby a sequence of adjacent characters.**

**Note that we can move to any of 8 adjacent characters,but a word should not have multiple instances of same cell.**

|  |  |
| --- | --- |
| **Example1**  **MxN:3x3**  **Number of words in the dictionary: 4 Words: {"CAT",“ACT”,“DIP”}**  **Board:{{C,A,P},**  **{A,N,D},**  **{T,I,E}}**  **Output:WordfromtheDICTIONARY that can be found in the board=“CAT” Explanation:** | **Example 2 MxN:3x3**  **Number of words in the dictionary: 4 Words: {"SEEK","FOR","QUIZ","GO"}**  **Board:**  **{{S,I,Z},**  **{U,E,K},**  **{Q,S,E}}**  **Output: Word from the DICTIONARY that can be found in the board =“SEEK”,“QUIZ”**  **Explanation:’** |

#include <iostream>

using namespace std;

class WordSearch

{

int rows,cols;

char \*\*board;

bool \*\*visited;

string \*dictionary;

int dictSize;

int dx[8]={-1,-1,-1,0,0,1,1,1};

int dy[8]={-1,0,1,-1,1,-1,0,1};

bool isValid(int x,int y)

{

return x>=0&&x<rows&&y>=0&&y<cols&&!visited[x][y];

}

bool searchWord(int x,int y,string &word,int index)

{

if (index==word.length())

return true;

if (!isValid(x,y)||board[x][y] != word[index])

return false;

visited[x][y]=true;

for (int i=0;i<8;i++)

if (searchWord(x +dx[i],y +dy[i],word,index +1))

return true;

visited[x][y]=false;

return false;

}

bool exists(string &word)

{

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

for (int j=0;j<cols;j++)

if (board[i][j]==word[0]&&searchWord(i,j,word,0))

return true;

return false;

}

public:

WordSearch(int r,int c,char \*\*b,string \*d,int size) : rows(r),cols(c),dictSize(size)

{

board=b;

dictionary=d;

visited=new bool \*[rows];

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

{

visited[i]=new bool[cols];

for (int j=0;j<cols;j++)

visited[i][j]=false;

}

}

~WordSearch()

{

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

delete[] visited[i];

delete[] visited;

}

void findWords()

{

cout<<"Words Found :\n";

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

if (exists(dictionary[i]))

cout << dictionary[i] << "\n";

}

};

int main()

{

int rows=3,cols=3;

char \*boardData[]={new char[3]{'S','I','Z'},new char[3]{'U','E','K'},new char[3]{'Q','S','E'}};

string dictionary[]={"SEEK","FOR","QUIZ","GO"};

WordSearch ws(rows,cols,boardData,dictionary,4);

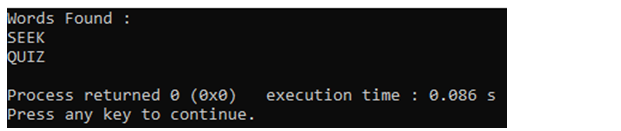
ws.findWords();

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

delete[] boardData[i];

}

**Output :**

****

**Q6.Mcoloringisawell-knownproblemwhere,nodesofagivengraphG={V,E}is to be colored with maximum M colors such that any two nodes adjacent to each other arecolored with different colors. Wecallthe graphGas M colourable,if all nodes of the graph G can be colored using maximum M colors. Write a program (backtracking based)tofindwhether,theinputtedgraphGisMcolourable(userinputtedM)or not.**

#include <iostream>

using namespace std;

class MColoring

{

int n,m;

bool \*\*graph;

int \*colors;

bool isSafe(int node,int color)

{

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

if (graph[node][i]&&colors[i]==color)

return false;

return true;

}

bool solve(int node)

{

if (node==n)

return true;

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

if (isSafe(node,c))

{

colors[node]=c;

if (solve(node +1))

return true;

colors[node]=0;

}

return false;

}

public:

MColoring(int vertices,int maxColors,int edges[][2],int edgeCount) : n(vertices),m(maxColors)

{

graph=new bool \*[n];

colors=new int[n];

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

{

graph[i]=new bool[n];

colors[i]=0;

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

graph[i][j]=false;

}

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

{

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

graph[u][v]=graph[v][u]=true;

}

}

~MColoring()

{

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

delete[] graph[i];

delete[] graph;

delete[] colors;

}

bool canColor()

{

return solve(0);

}

};

int main()

{

int n,M,edgeCount;

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

cin>>n;

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

cin>>edgeCount;

cout<<"Input the value of M to check for M-colourable graph : ";

cin>>M;

int edges[M][2];

cout<<"Input the vertices connected to each other : ";

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

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

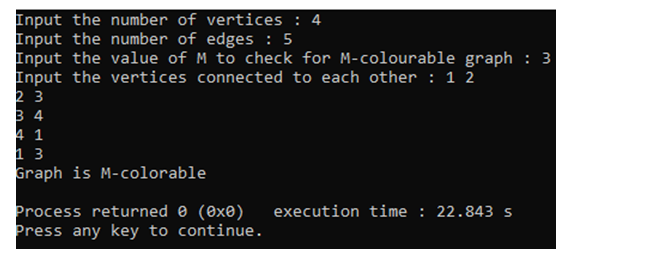
MColoring mc(n,M,edges,edgeCount);

cout << (mc.canColor() ? "Graph is M-colorable\n" : "Graph is not M-colorable\n");

return 0;

}

**Output :**

****

**Q10.GivenanundirectedgraphandanintegerM.Thetaskistodetermineifthe graph can be colored with at most M colors such that no two adjacent vertices of the grapharecoloredwiththesamecolor.Herecoloringofagraphmeansthe assignmentofcolorstoallvertices.Print1ifitispossibletocolourverticesand 0 otherwise.**

|  |  |
| --- | --- |
| **Example1:**  **Input:**  **N=4**  **M=3**  **E=5**  **Edges[]={(0,1),(1,2),(2,3),(3,0),(0,2)}**  **Output:1**  **Explanation:Itispossibletocolourthe given graph using 3 colours.** | **Example2:**  **Input:**  **N=3**  **M=2**  **E=3**  **Edges[]={(0,1),(1,2),(0,2)}**  **Output:0** |

#include <iostream>

using namespace std;

class MColoring

{

int n,m;

bool \*\*graph;

int \*colors;

bool isSafe(int node,int color)

{

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

if (graph[node][i]&&colors[i]==color)

return false;

return true;

}

bool solve(int node)

{

if (node==n)

return true;

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

if (isSafe(node,c))

{

colors[node]=c;

if (solve(node +1))

return true;

colors[node]=0;

}

return false;

}

public:

MColoring(int vertices,int maxColors,int edges[][2],int edgeCount) : n(vertices),m(maxColors)

{

graph=new bool \*[n];

colors=new int[n];

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

{

graph[i]=new bool[n];

colors[i]=0;

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

graph[i][j]=false;

}

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

{

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

graph[u][v]=graph[v][u]=true;

}

}

~MColoring()

{

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

delete[] graph[i];

delete[] graph;

delete[] colors;

}

bool canColor()

{

return solve(0);

}

};

int main()

{

int n,M,edgeCount;

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

cin>>n;

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

cin>>edgeCount;

cout<<"Input the value of M to check for M-colourable graph : ";

cin>>M;

int edges[M][2];

cout<<"Input the vertices connected to each other : ";

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

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

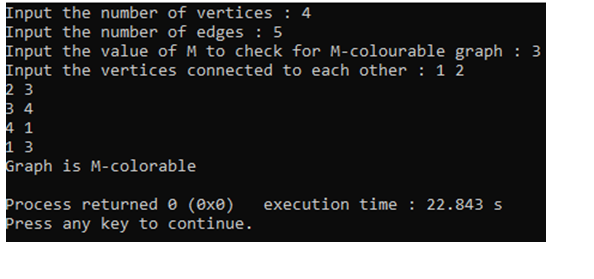
MColoring mc(n,M,edges,edgeCount);

cout << (mc.canColor() ? "Graph is M-colorable\n" : "Graph is not M-colorable\n");

return 0;

}

**Output :**

****