**15B17CI371 – Data Structures Lab**

**ODD 2024**

**Week 6-LAB B**

**Practice Lab**

**[CO: C270.3]**

**Instructions:**

**1. All students must save all their programs with the nomenclature**

**(Enroll No\_W6\_LabB\_QuestionNo.cpp). Also, store the Outputs as well.**

**2. Upload them as per the instructions given by your lab faculty.**

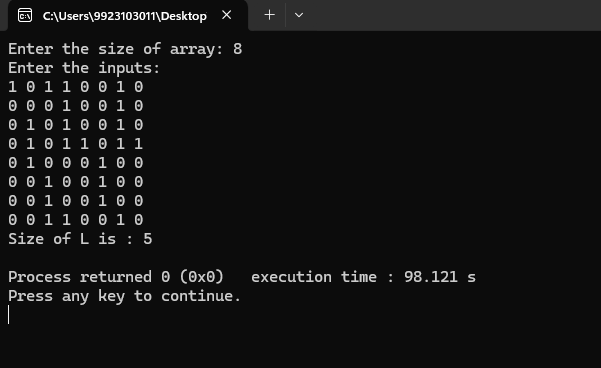
**Concepts: Experiment with backtracking using stacks for implementation of rat-**

**in-a-maze problem, n-Queens problem, etc.**

**Lab Questions:**

**1. Search shape L of given size (user input) from following matrix. And give**

**its size**



#include <iostream>

#include <bits/stdc++.h>

using namespace std;

int main()

{

stack<int> s;

int count=0;

int max=0;

int n;

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

cin>>n;

int \*\*arr=new int\*[n];

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

{

arr[i]=new int[n];

}

cout<<"Enter the inputs: "<<endl;

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

{

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

{

cin>>arr[i][j];

}

}

for(int i=0;i<n-1;i++)

{

for(int j=0;j<n-1;j++)

{

int r=i,c=j;

if(arr[i][j]==1)

{

while(arr[i][j]==1 && arr[i][j+1]==0 && j+1<n && i<n-1)

{

count++;

i++;

}

if(arr[i][j]==1 && j<n)

{

while(arr[i][j]==1 && j<n)

{

count++;

j++;

}

}

else

{

count==0;

}

}

i=r,j=c;

if(count>max)

{

max=count;

}

count=0;

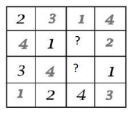
}

}

cout<<"Size of L is : "<<max<<endl;

}

**2. Consider 4x4 Sudoku problem. We have kept 14 correct values; however, 2 values are missing. But you have to find these missing values by implementing stack.**

****

#include <iostream>

#include <stack>

using namespace std;

const int SIZE = 4;

bool isSafe(int board[SIZE][SIZE], int row, int col, int num)

{

for (int x = 0; x < SIZE; x++)

if (board[row][x] == num || board[x][col] == num)

return false;

int startRow = row - row % 2, startCol = col - col % 2;

for (int i = 0; i < 2; i++)

for (int j = 0; j < 2; j++)

if (board[i + startRow][j + startCol] == num)

return false;

return true;

}

bool solveSudoku(int board[SIZE][SIZE])

{

stack<pair<int, int> > positions;

int missingCount = 0;

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

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

if (board[i][j] == 0)

{

positions.push(make\_pair(i, j));

missingCount++;

}

if (missingCount != 2)

return false;

while (!positions.empty())

{

pair<int, int> pos = positions.top();

positions.pop();

int row = pos.first;

int col = pos.second;

bool placed = false;

for (int num = 1; num <= 4; num++)

if (isSafe(board, row, col, num))

{

board[row][col] = num;

bool allFilled = true;

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

{

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

if (board[i][j] == 0)

{

allFilled = false;

break;

}

if (!allFilled)

break;

}

if (allFilled)

return true;

placed = true;

break;

}

if (!placed)

{

board[row][col] = 0;

if (positions.empty())

positions.push(make\_pair(row, col));

}

}

return false;

}

void printBoard(int board[SIZE][SIZE])

{

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

{

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

cout << board[i][j] << " ";

cout << endl;

}

}

int main()

{

int board[SIZE][SIZE];

cout << "Enter the Sudoku values (0 for missing values):\n";

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

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

cin >> board[i][j];

if (solveSudoku(board))

{

cout << "Solved Sudoku:\n";

printBoard(board);

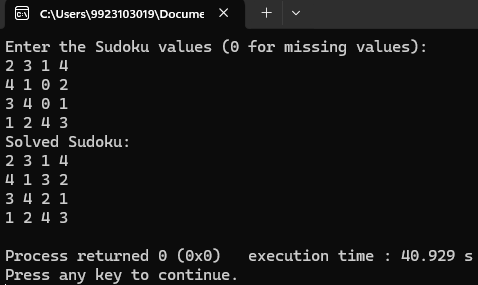
}

else

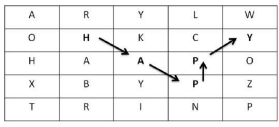
cout << "No solution exists." << endl;

}

**Output :**

****

**4. Write a function to check whether a given string exists in the two-dimensional character matrix or not. Print the path if the string exists. You may use all movements such as left, right, up, and down. Consider the following two dimensional character matrix and the string to be searched is “HAPPY”. Example is shown below**

****

**Assumption: a character once visited, cannot be visited again.**

**Output: Path: (1,1), (2,2), (3,3), (2,3), (1,4)**

#include<iostream>

using namespace std;

bool searchPath(char mat[5][5], int row, int col, string word, int index, int n, bool visited[5][5], int path[][2])

{

if(index == word.length()) return true;

if(row < 0 || col < 0 || row >= 5 || col >= 5 || visited[row][col] || mat[row][col] != word[index])

return false;

visited[row][col] = true;

path[index][0] = row;

path[index][1] = col;

if(searchPath(mat, row-1, col, word, index+1, n, visited, path))

return true;

if(searchPath(mat, row+1, col, word, index+1, n, visited, path))

return true;

if(searchPath(mat, row, col-1, word, index+1, n, visited, path))

return true;

if(searchPath(mat, row, col+1, word, index+1, n, visited, path))

return true;

if(searchPath(mat, row-1, col-1, word, index+1, n, visited, path))

return true;

if(searchPath(mat, row-1, col+1, word, index+1, n, visited, path))

return true;

if(searchPath(mat, row+1, col-1, word, index+1, n, visited, path))

return true;

if(searchPath(mat, row+1, col+1, word, index+1, n, visited, path))

return true;

visited[row][col] = false;

return false;

}

void findString(char mat[5][5], string word)

{

int n = word.length();

bool visited[5][5] = {false};

int path[5][2];

for(int i=0; i<5; i++)

for(int j=0; j<5; j++)

if(mat[i][j] == word[0] && searchPath(mat, i, j, word, 0, n, visited, path))

{

cout<<"\nPath : ";

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

cout<<"("<<path[k][0]<<","<<path[k][1]<<") ";

return;

}

cout<<"String not found";

}

int main()

{

char mat[5][5] =

{

{'A', 'R', 'Y', 'L', 'W'},

{'O', 'H', 'K', 'C', 'Y'},

{'H', 'A', 'A', 'P', 'O'},

{'X', 'B', 'T', 'P', 'Z'},

{'T', 'R', 'I', 'N', 'P'}

};

cout<<"Matrix :\n";

for(int i=0;i<5;i++)

{

for(int j=0;j<5;j++)

cout<<mat[i][j]<<" ";

cout<<endl;

}

string word;

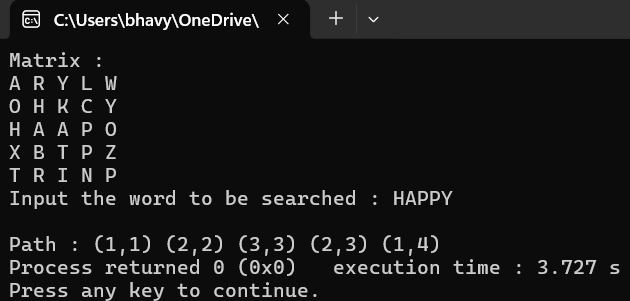
cout<<"Input the word to be searched : ";

cin>>word;

findString(mat, word);

}

**Output :**



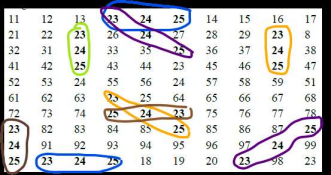
**5. Consider following two matrices**

| **M1 =** |  | **11 12 13 23 24 25 14 15 16 17 21 22 23 26 24 27 28 29 23 8 32 31 24 33 35 25 36 37 24 38 41 42 25 43 44 23 45 46 25 47 52 53 24 55 56 24 57 58 59 51 61 62 63 23 25 64 65 66 67 68 72 73 74 25 24 23 75 76 77 78 23 82 83 84 85 25 85 86 87 25 24 91 92 93 94 95 96 97 24 99 25 23 24 25 18 19 20 23 98 23** |  |
| --- | --- | --- | --- |

**M2 = [23, 24, 25]**

**Write a function to check whether elements of M2 are present in M1. If elements exist, print the count that how many times M2 elements present in M1. You may use all movements such as left, right, up, and down.**

**Output: 9**

****

#include <iostream>

using namespace std;

int M1[10][10] =

{

{11, 12, 13, 23, 24, 25, 14, 15, 16, 17},

{21, 22, 23, 26, 24, 27, 28, 29, 23, 8},

{32, 31, 24, 33, 35, 25, 36, 37, 24, 38},

{41, 42, 25, 43, 44, 23, 45, 46, 25, 47},

{52, 53, 24, 55, 56, 24, 57, 58, 59, 51},

{61, 62, 63, 23, 25, 64, 65, 66, 67, 68},

{72, 73, 74, 25, 24, 23, 75, 76, 77, 78},

{23, 82, 83, 84, 85, 25, 85, 86, 87, 25},

{24, 91, 92, 93, 94, 95, 96, 97, 24, 99},

{25, 23, 24, 25, 18, 19, 20, 23, 98, 23}

};

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

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

int M2[] = {23, 24, 25};

int M2\_len = 3;

bool isValid(int x, int y)

{

return x >= 0 && x < 10 && y >= 0 && y < 10;

}

bool searchInDirection(int x, int y, int dir)

{

for (int i = 0; i < M2\_len; i++)

{

int nx = x + i \* dx[dir], ny = y + i \* dy[dir];

if (!isValid(nx, ny) || M1[nx][ny] != M2[i])

return false;

}

return true;

}

int countPatterns()

{

int count = 0,dir;

for (int i = 0; i < 10; i++)

for (int j = 0; j < 10; j++)

if (M1[i][j] == M2[0])

for (dir = 0; dir < 8; dir++)

{

if (searchInDirection(i, j, dir))

count++;

}

count=dir+1;

return count;

}

int main()

{

for (int i = 0; i < 10; i++)

{

for (int j = 0; j < 10; j++)

cout<<M1[i][j]<<" ";

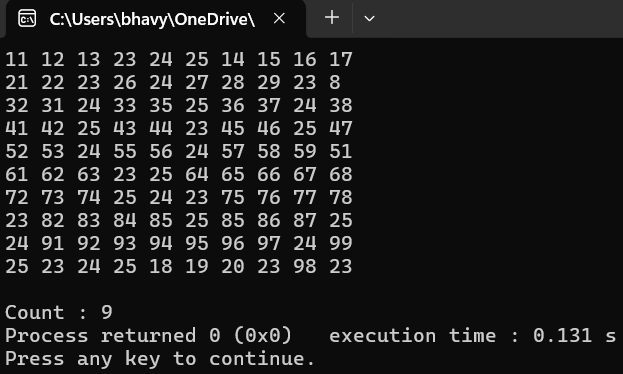
cout<<endl;

}

cout<<endl<<"Count : "<<countPatterns();

}

**Output :**

****