15B17CI371 – Data Structures Lab
ODD 2024
Week 6-LAB B
Practice Lab

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

1		1	1			1	
			1			1	
	1		1			1	
	1		1	1		1	1
	1				1		
		1			1		
		1			1		
		1	1			1	

Output:

Eg size of below L shape is 5.

1	
1	
1	
1	1

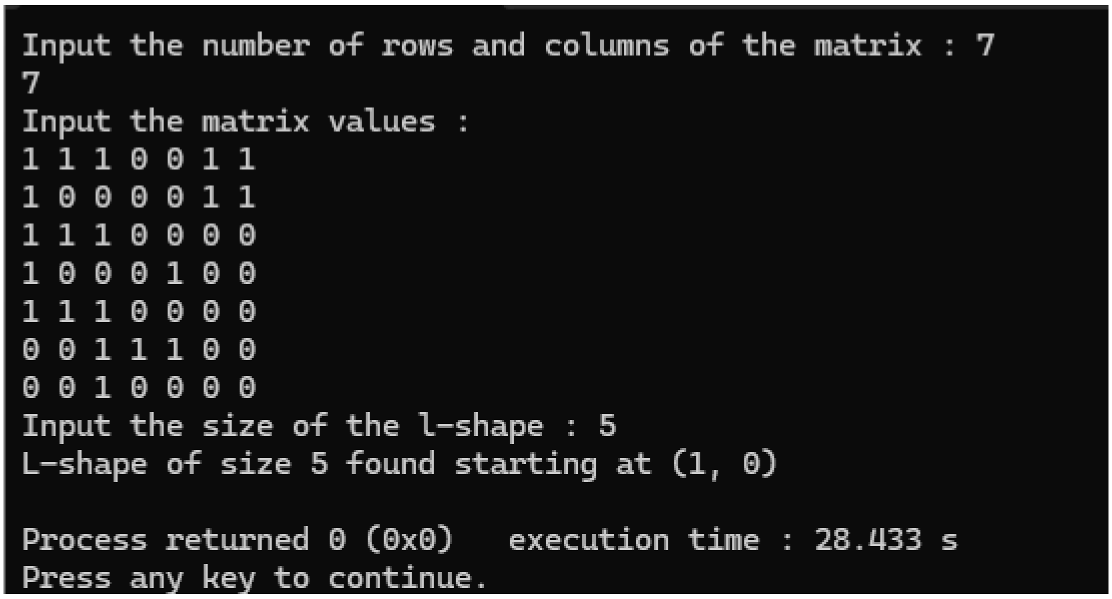
Below shape is not L shape

1	
1	1
1	
1	1

```
#include <iostream>
#include <stack>
using namespace std;
const int MAX_SIZE = 100;
bool checklshape(int matrix[MAX_SIZE][MAX_SIZE], int n, int m, int row, int col, int height) {
  stack<pair<int, int> > s;
  if (row + height - 1 < n && col + 1 < m)
  {
    for (int k = 0; k < height; k++)
    {
      if (matrix[row + k][col] == 1)
        s.push({row + k, col});
      else
        return false;
    }
    if (matrix[row + height - 1][col + 1] == 1)
      return true;
  }
  while (!s.empty())
    s.pop();
  if (row + 1 < n && col + height - 1 < m)
  {
        for (int k = 0; k < height; k++)
        {
            if (matrix[row][col + k] == 1)
                s.push({row, col + k});
            else
                return false;
        }
        if (matrix[row + 1][col + height - 1] == 1)
            return true;
    }
    return false;
}

int main() {
    int n, m, size;
    cout << "Input the number of rows and columns of the matrix : ";
    cin >> n >> m;
    int matrix[MAX_SIZE][MAX_SIZE];
    cout << "Input the matrix values :\n";
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            cin >> matrix[i][j];
    cout << "Input the size of the l-shape : ";
    cin >> size;
    if (size < 2)
    {
        cout << "Invalid size for l-shape. Size must be at least 2.\n";
        return 0;
    }
    int height = size - 1;
    bool found = false;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            if (matrix[i][j] == 1 && checklshape(matrix, n, m, i, j, height))
            {
                cout << "L-shape of size " << size << " found starting at (" << i << ", " << j << ")\n";
                found = true;
            }
    if (!found)
        cout << "no l-shape of size " << size << " found in the matrix.\n";
}
```

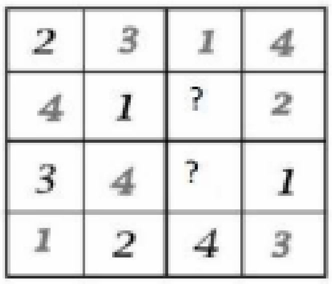
Output :



```
Input the number of rows and columns of the matrix : 7
7
Input the matrix values :
1 1 1 0 0 1 1
1 0 0 0 0 1 1
1 1 1 0 0 0 0
1 0 0 0 1 0 0
1 1 1 0 0 0 0
0 0 1 1 1 0 0
0 0 1 0 0 0 0
Input the size of the l-shape : 5
L-shape of size 5 found starting at (1, 0)

Process returned 0 (0x0)   execution time : 28.433 s
Press any key to continue.
```

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.**

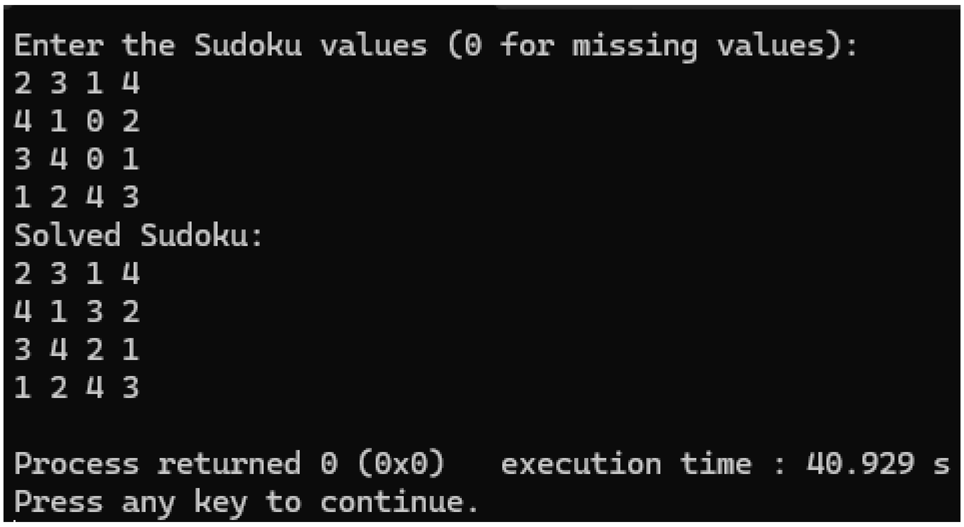


2	3	1	4
4	1	?	2
3	4	?	1
1	2	4	3

```
#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;
}
```

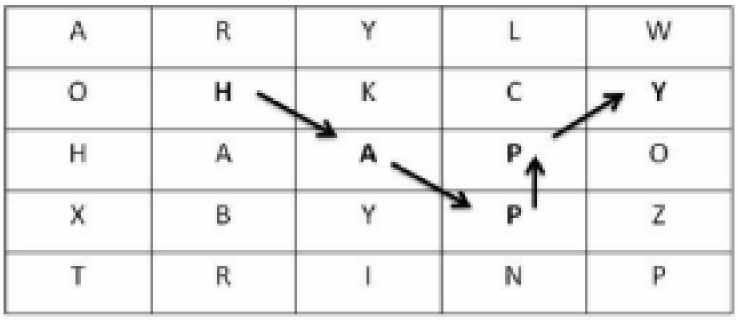
Output :



```
Enter the Sudoku values (0 for missing values):
2 3 1 4
4 1 0 2
3 4 0 1
1 2 4 3
Solved Sudoku:
2 3 1 4
4 1 3 2
3 4 2 1
1 2 4 3

Process returned 0 (0x0)   execution time : 40.929 s
Press any key to continue.
```

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**



A	R	Y	L	W
O	H	K	C	Y
H	A	A	P	O
X	B	Y	P	Z
T	R	I	N	P

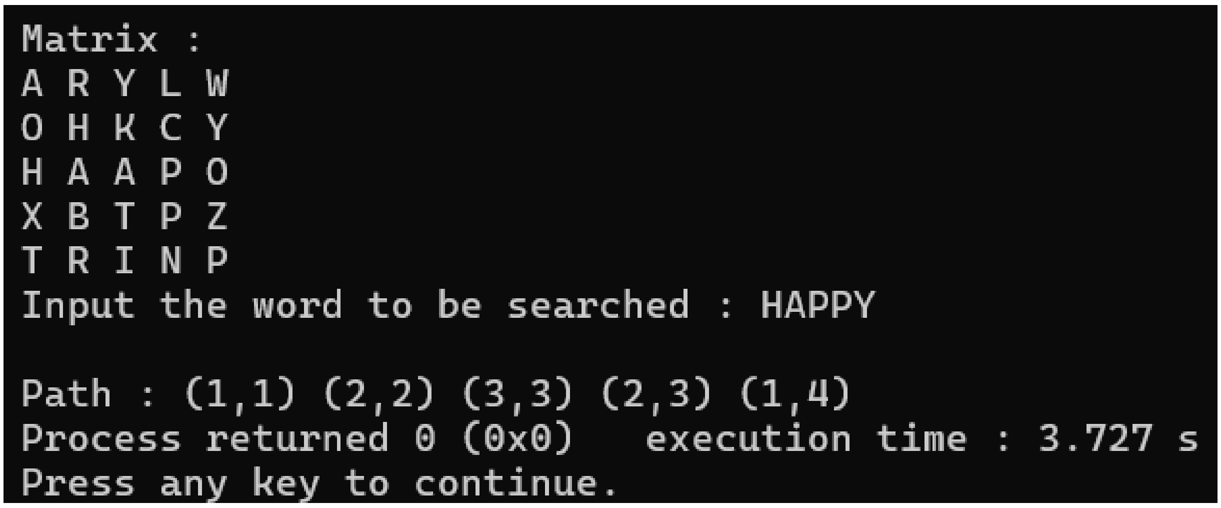
Assumption: a character once visited, cannot be visited again.
Output: Path: (1,1), (2,2), (3,3), (2,3), (1,4)

```
#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 :



```
Matrix :
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
Input the word to be searched : HAPPY

Path : (1,1) (2,2) (3,3) (2,3) (1,4)
Process returned 0 (0x0)   execution time : 3.727 s
Press any key to continue.
```

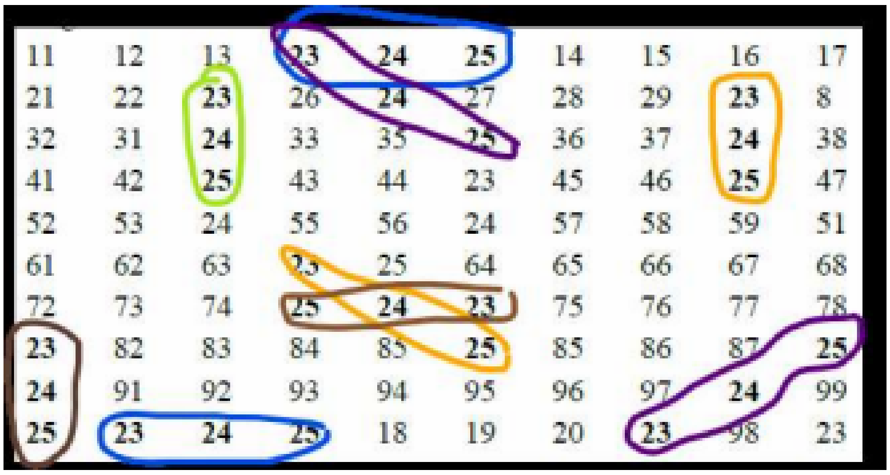
5. Consider following two matrices

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

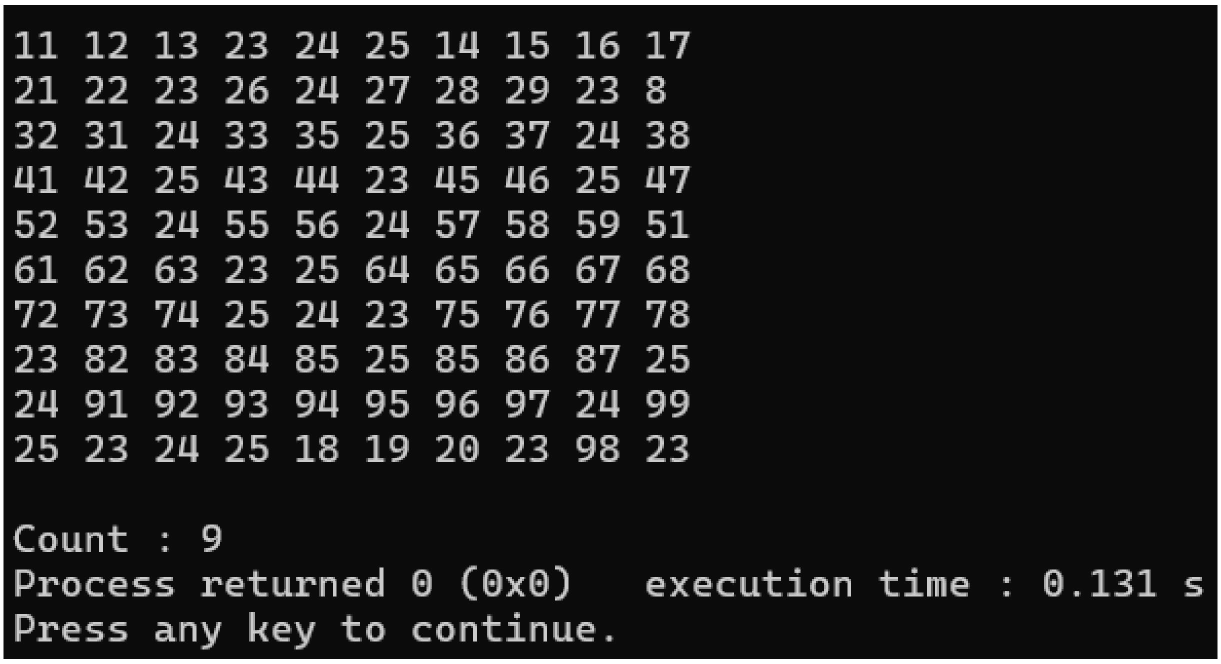


11	12	13	23	24	25	14	15	16	17
21	22	23	26	24	27	28	29	23	8
32	31	24	33	35	25	36	37	24	38
41	42	25	43	44	23	45	46	25	47
52	53	24	55	56	24	57	58	59	51
61	62	63	23	25	64	65	66	67	68
72	73	74	25	24	23	75	76	77	78
23	82	83	84	85	25	85	86	87	25
24	91	92	93	94	95	96	97	24	99
25	23	24	25	18	19	20	23	98	23

```
#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 :



```
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

Count : 9
Process returned 0 (0x0)   execution time : 0.131 s
Press any key to continue.
```