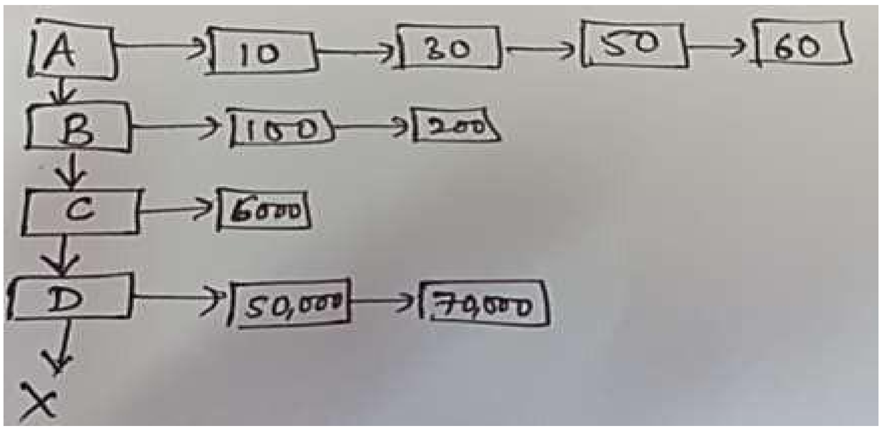
15B17CI371–Data Structures

Lab ODD 2024

Week10-LabA

Practice Lab

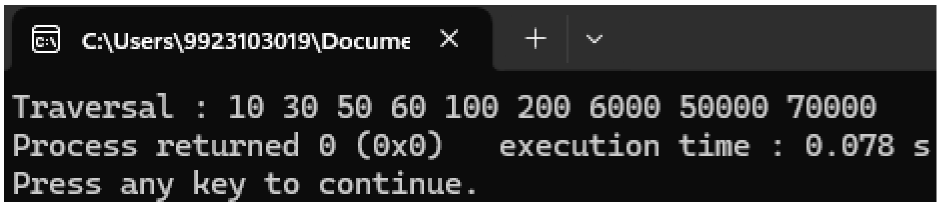
Q1. Write a program to
(a) Store data in a data structure to represent the below shown image.
(b) Write function to traverse in the manner such that the linked-list associated with A pointer is traversed first,then B pointer,followed by C pointer,and so on. Assume the NULL pointer is attached at the end of each row.



```
#include <iostream>
using namespace std;
struct Node
{
   int data;
   Node* next;
};
struct RowNode
{
   Node* head;
   RowNode* down;
};
RowNode* createRow(int arr[],int size)
{
   RowNode* row=new RowNode;
   row->down=NULL;
   row->head=NULL;
   Node* temp=NULL;
   for(int i=0;i<size;i++)
   {
      Node* newNode=new Node;
      newNode->data=arr[i];
      newNode->next=NULL;
      if(row->head==NULL)
```

```
        {
           row->head=newNode;
           temp=newNode;
        }
        else
        {
           temp->next=newNode;
           temp=newNode;
        }
    }
    return row;
}
void traverse(RowNode* row)
{
    cout<<"Traversal : ";
    while(row!=NULL)
    {
        Node* current=row->head;
        while(current!=NULL)
        {
           cout<<current->data<<" ";
           current=current->next;
        }
        row=row->down;
    }
}
int main()
{
    int a[]={10,30,50,60};
    int b[]={100,200};
    int c[]={6000};
    int d[]={50000,70000};
    RowNode* head=createRow(a,4);
    head->down=createRow(b,2);
    head->down->down=createRow(c,1);
    head->down->down->down=createRow(d,2);
    traverse(head);
}
```

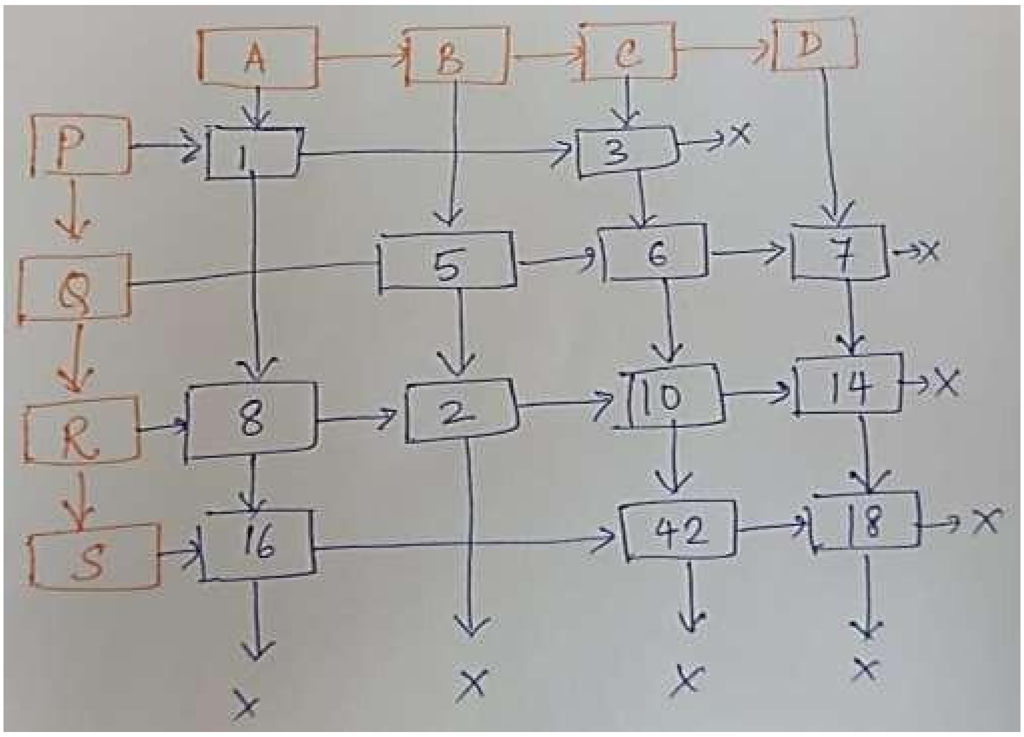
Output :



```
Traversal : 10 30 50 60 100 200 6000 50000 70000
Process returned 0 (0x0)   execution time : 0.078 s
Press any key to continue.
```

Q2. Write a program to
(a) construct the below given structure.
(b) Add a function to perform traversal such that elements are displayed top-tobottom,and left-to-right.
(c) Write a function to convert the given structure into a simple matrix. Given two such

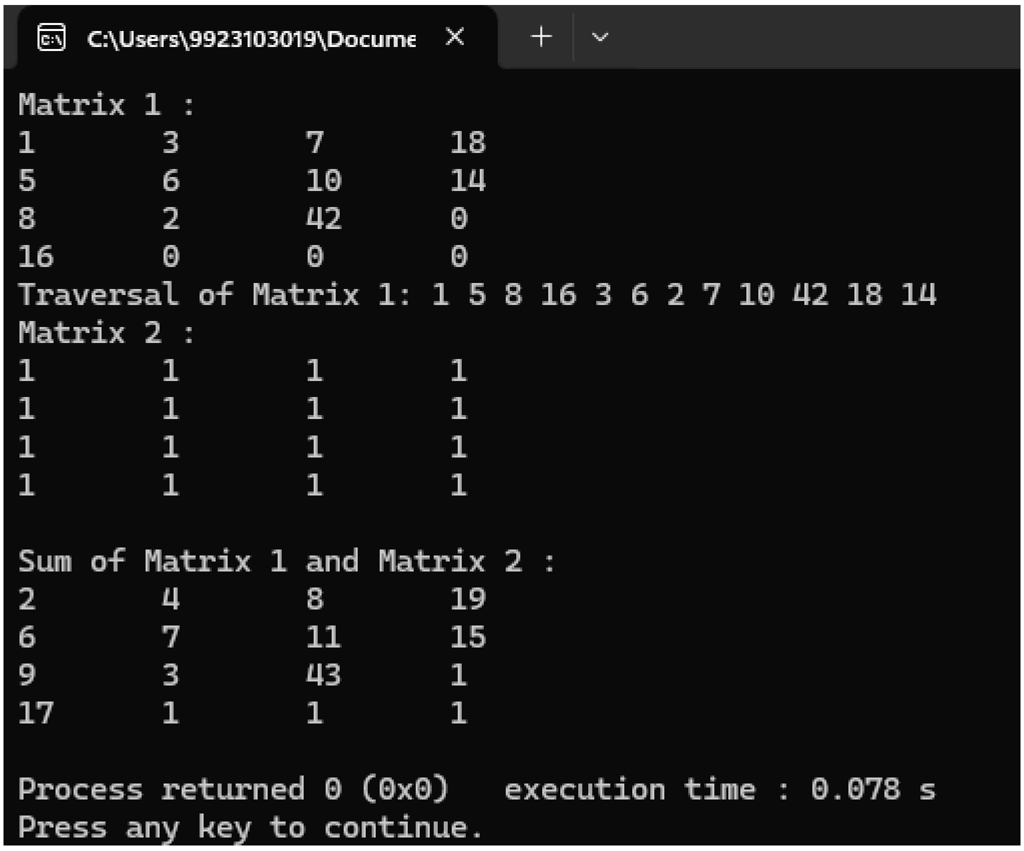
structures,s,perform addition and display result by using the function in(b)



```
#include <iostream>
using namespace std;
struct Node
{
   int value;
   Node *top,*bottom,*left,*right;
   Node(int val) : value(val),top(NULL),bottom(NULL),left(NULL),right(NULL) {}
};
class MultiLinkedList
{
   Node *head[4];
public:
   MultiLinkedList(int matrix[4][4])
   {
      Node *rows[4][4];
      for(int i=0;i<4;i++)
         for(int j=0;j<4;j++)
            rows[i][j]=matrix[i][j]!=0?new Node(matrix[i][j]):NULL;

      for(int i=0;i<4;i++)
         for(int j=0;j<4;j++)
            if(rows[i][j])
            {
               if(i>0) rows[i][j]->top=rows[i-1][j];
               if(i<3) rows[i][j]->bottom=rows[i+1][j];
               if(j>0) rows[i][j]->left=rows[i][j-1];
               if(j<3) rows[i][j]->right=rows[i][j+1];
               if(i==0) head[j]=rows[i][j];
            }
    }
    void traverse()
    {
        for(int i=0;i<4;i++)
        {
            Node *current=head[i];
            while(current!=NULL)
            {
                cout<<current->value<<" ";
                current=current->bottom;
            }
        }
    }
    void convertToMatrix(int matrix[4][4])
    {
        for(int i=0;i<4;i++)
        {
            Node *current=head[i];
            int row=0;
            while(current)
            {
                matrix[row++][i]=current->value;
                current=current->bottom;
            }
        }
    }
    static void addMatrices(int matrix1[4][4],int matrix2[4][4],int result[4][4])
    {
        for(int i=0;i<4;i++)
            for(int j=0;j<4;j++)
                result[i][j]=matrix1[i][j]+matrix2[i][j];
    }
};
int main()
{
    int matrix1[4][4]={{1,3,7,18},{5,6,10,14},{8,2,42,0},{16,0,0,0}};
    int matrix2[4][4]={{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1}};
    int result[4][4]={0};
    MultiLinkedList list1(matrix1);
    MultiLinkedList list2(matrix2);
    list1.convertToMatrix(matrix1);
    list2.convertToMatrix(matrix2);
    cout<<"Matrix 1 :"<<endl;
    for(int i=0;i<4;i++)
    {
        for(int j=0;j<4;j++)
            cout<<matrix1[i][j]<<"\t";
        cout<<endl;
    }
    cout<<"Traversal of Matrix 1 : ";
    list1.traverse();
    MultiLinkedList::addMatrices(matrix1,matrix2,result);
    cout<<endl<<"Matrix 2 :"<<endl;
    for(int i=0;i<4;i++)
    {
       for(int j=0;j<4;j++)
          cout<<matrix2[i][j]<<"\t";
       cout<<endl;
    }
    cout<<endl<<"Sum of Matrix 1 and Matrix 2 :"<<endl;
    for(int i=0;i<4;i++)
    {
       for(int j=0;j<4;j++)
          cout<<result[i][j]<<"\t";
       cout<<endl;
    }
}
```

Output :



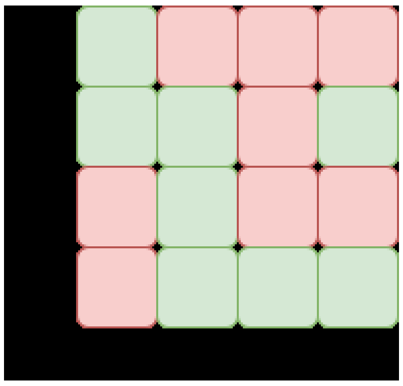
C:\Users\9923103019\Docume

```
Matrix 1 :
1       3       7       18
5       6       10      14
8       2       42      0
16      0       0       0
Traversal of Matrix 1: 1 5 8 16 3 6 2 7 10 42 18 14
Matrix 2 :
1       1       1       1
1       1       1       1
1       1       1       1
1       1       1       1

Sum of Matrix 1 and Matrix 2 :
2       4       8       19
6       7       11      15
9       3       43      1
17      1       1       1

Process returned 0 (0x0)   execution time : 0.078 s
Press any key to continue.
```

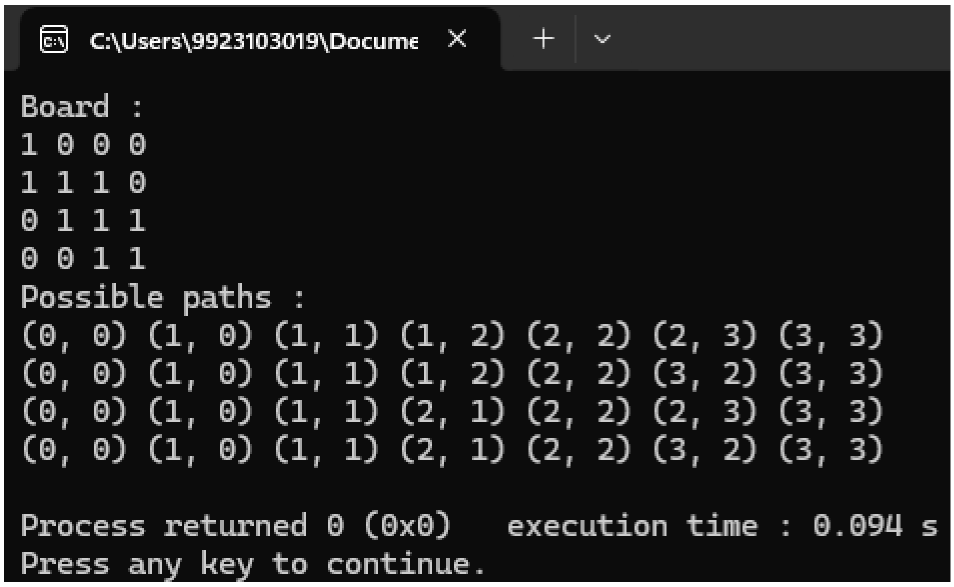
Q3.Consider that you are required to design a board game in which obstacles are placed in certain places in the square board.The“Green”blocks are the blocks through which you can traverse and the“Red”blocks are the obstacles,i.e.,you cannot traverse through these. Your game starts from the top-left corner(source) and ends in the bottom-right corner(destination). Write a program(using stack) to present all possible paths assuming that you can move only to only 1 block at a time(i.e.,one step either to right or left or up or down).



```
#include <iostream>
#include <stack>
#include <vector>
using namespace std;
struct Point
{
   int x,y;
};
bool isValid(int x,int y,vector<vector<int>>&board,vector<vector<bool>>&visited)
{
   return(x>=0&&x<4&&y>=0&&y<4&&board[x][y]==1&&!visited[x][y]);
}
void printPath(stack<Point> &path)
{
   stack<Point> temp=path;
   vector<Point> points;
   while(!temp.empty())
   {
      points.push_back(temp.top());
      temp.pop();
   }
   for(int i=points.size()-1;i>=0;i--)
      cout<<"("<<points[i].x<<","<<points[i].y<<") ";
   cout<<endl;
}
void dfs(int x,int y,vector<vector<int>>&board,vector<vector<bool>>&visited,stack<Point> &path)
{
   if(x==3&&y==3)
   {
      printPath(path);
      return;
   }
   if(isValid(x,y+1,board,visited))
   {
      visited[x][y+1]=true;
      path.push({x,y+1});
      dfs(x,y+1,board,visited,path);
      visited[x][y+1]=false;
      path.pop();
```

```
    }
    if(isValid(x+1,y,board,visited))
    {
        visited[x+1][y]=true;
        path.push({x+1,y});
        dfs(x+1,y,board,visited,path);
        visited[x+1][y]=false;
        path.pop();
    }
    if(isValid(x,y-1,board,visited))
    {
        visited[x][y-1]=true;
        path.push({x,y-1});
        dfs(x,y-1,board,visited,path);
        visited[x][y-1]=false;
        path.pop();
    }
    if(isValid(x-1,y,board,visited))
    {
        visited[x-1][y]=true;
        path.push({x-1,y});
        dfs(x-1,y,board,visited,path);
        visited[x-1][y]=false;
        path.pop();
    }
}
void findPaths(vector<vector<int>>&board)
{
    stack<Point> path;
    vector<vector<bool>>visited(4,vector<bool>(4,false));
    visited[0][0]=true;
    path.push({0,0});
    dfs(0,0,board,visited,path);
}
void display(vector<vector<int>>&board)
{
    cout<<"Board :"<<endl;
    for(int i=0;i<4;i++)
    {
        for(int j=0;j<4;j++)
            cout<<board[i][j]<<" ";
        cout<<endl;
    }
}
int main()
{
    vector<vector<int>>board =
    {
        {1,0,0,0},
        {1,1,1,0},
        {0,1,1,1},
        {0,0,1,1}
    };
    display(board);
    cout<<"Possible paths : "<<endl;
    findPaths(board);
}
```

Output :



```
C:\Users\9923103019\Docume
Board :
1 0 0 0
1 1 1 0
0 1 1 1
0 0 1 1
Possible paths :
(0, 0) (1, 0) (1, 1) (1, 2) (2, 2) (2, 3) (3, 3)
(0, 0) (1, 0) (1, 1) (1, 2) (2, 2) (3, 2) (3, 3)
(0, 0) (1, 0) (1, 1) (2, 1) (2, 2) (2, 3) (3, 3)
(0, 0) (1, 0) (1, 1) (2, 1) (2, 2) (3, 2) (3, 3)

Process returned 0 (0x0)   execution time : 0.094 s
Press any key to continue.
```

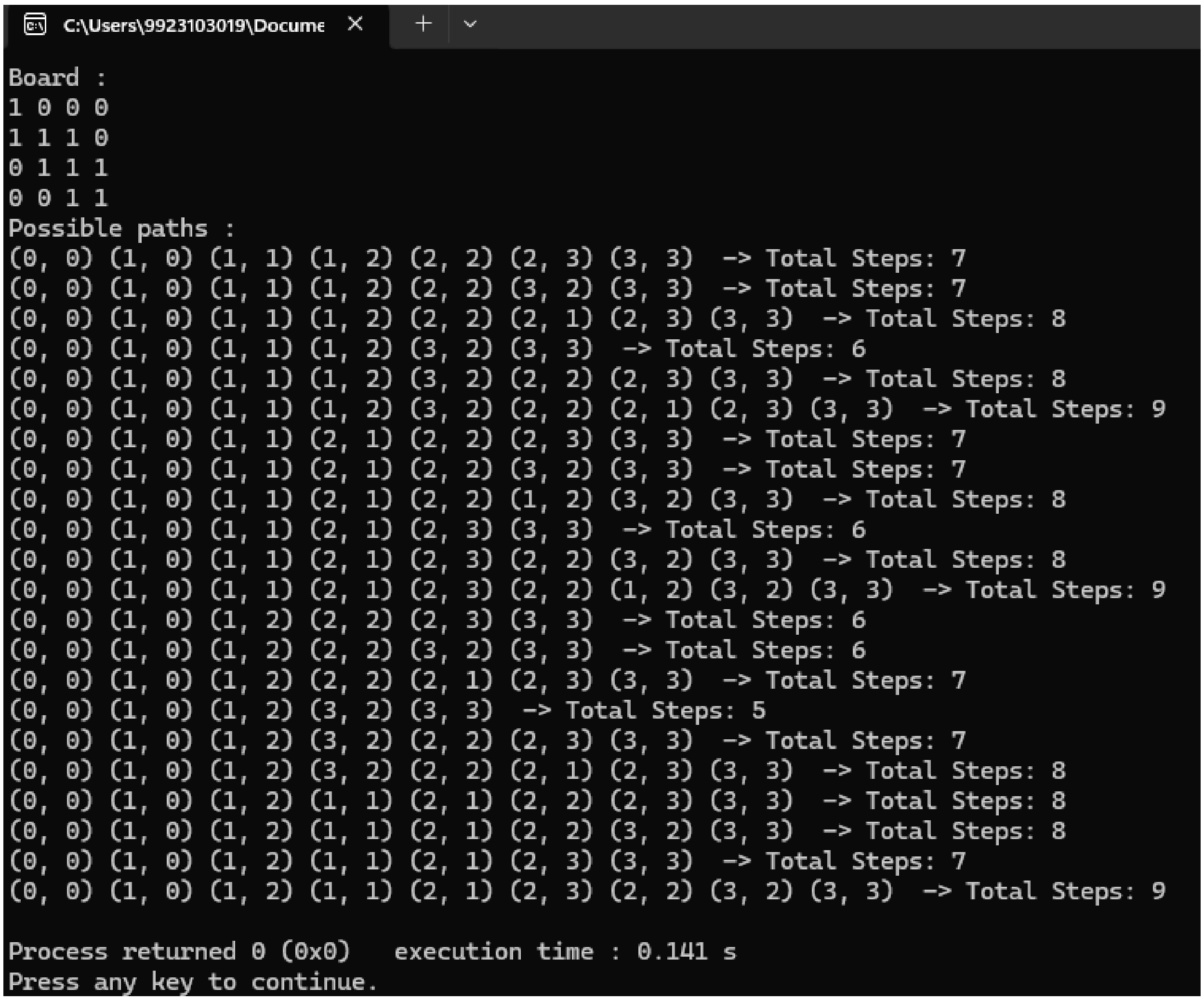
Q4. For the above question,suppose you are allowed to move either one step or maximum 2 steps in any direction in one move,write a program that would present all possible paths and the number of steps required to move from the source to the destination.

```
#include <iostream>
#include <stack>
#include <vector>
using namespace std;
struct Point
{
    int x,y,steps;
};

bool isValid(int x,int y,vector<vector<int>>&board,vector<vector<bool>>&visited)
{
    return(x>=0&&x<4&&y>=0&&y<4&&board[x][y]==1&&!visited[x][y]);
}
void printPath(stack<Point> &path)
{
    stack<Point> temp=path;
    vector<Point> points;
    int totalSteps=0;
    while(!temp.empty())
    {
        points.push_back(temp.top());
        temp.pop();
    }
    totalSteps=points.back().steps;
    for(int i=points.size()-1;i>=0;i--)
        cout<<"("<<points[i].x<<","<<points[i].y<<") ";
    cout<<" -> Total Steps: "<<points.size()<<endl;
}
void dfs(int x,int y,vector<vector<int>>&board,vector<vector<bool>>&visited,stack<Point> &path,int
stepCount)
{
    if(x==3&&y==3)
    {
        printPath(path);
        return;
    }
    if(isValid(x,y+1,board,visited))
    {
        visited[x][y+1]=true;
        path.push({x,y+1,stepCount+1});
        dfs(x,y+1,board,visited,path,stepCount+1);
        visited[x][y+1]=false;
        path.pop();
    }
    if(isValid(x,y+2,board,visited))
    {
        visited[x][y+2]=true;
        path.push({x,y+2,stepCount+2});
        dfs(x,y+2,board,visited,path,stepCount+2);
        visited[x][y+2]=false;
        path.pop();
    }
    if(isValid(x+1,y,board,visited))
    {
        visited[x+1][y]=true;
        path.push({x+1,y,stepCount+1});
        dfs(x+1,y,board,visited,path,stepCount+1);
        visited[x+1][y]=false;
        path.pop();
    }
    if(isValid(x+2,y,board,visited))
    {
        visited[x+2][y]=true;
        path.push({x+2,y,stepCount+2});
        dfs(x+2,y,board,visited,path,stepCount+2);
        visited[x+2][y]=false;
        path.pop();
    }
    if(isValid(x,y-1,board,visited))
    {
        visited[x][y-1]=true;
        path.push({x,y-1,stepCount+1});
        dfs(x,y-1,board,visited,path,stepCount+1);
        visited[x][y-1]=false;
        path.pop();
    }
    if(isValid(x,y-2,board,visited))
    {
        visited[x][y-2]=true;
        path.push({x,y-2,stepCount+2});
        dfs(x,y-2,board,visited,path,stepCount+2);
        visited[x][y-2]=false;
        path.pop();
    }

    if(isValid(x-1,y,board,visited))
    {
        visited[x-1][y]=true;
        path.push({x-1,y,stepCount+1});
        dfs(x-1,y,board,visited,path,stepCount+1);
        visited[x-1][y]=false;
        path.pop();
    }
    if(isValid(x-2,y,board,visited))
    {
        visited[x-2][y]=true;
        path.push({x-2,y,stepCount+2});
        dfs(x-2,y,board,visited,path,stepCount+2);
        visited[x-2][y]=false;
        path.pop();
    }
}
void findPaths(vector<vector<int>>&board)
{
    stack<Point> path;
    vector<vector<bool>>visited(4,vector<bool>(4,false));
    visited[0][0]=true;
    path.push({0,0,0});
    dfs(0,0,board,visited,path,0);
}
void display(vector<vector<int>>&board)
{
    cout<<"Board :"<<endl;
    for(int i=0;i<4;i++)
    {
        for(int j=0;j<4;j++)
            cout<<board[i][j]<<" ";
        cout<<endl;
    }
}
int main()
{
    vector<vector<int>>board=
    {
        {1,0,0,0},
        {1,1,1,0},
        {0,1,1,1},
        {0,0,1,1}
    };
    display(board);
    cout<<"Possible paths : "<<endl;
    findPaths(board);
}
```

Output :



```
C:\Users\9923103019\Docume

Board :
1 0 0 0
1 1 1 0
0 1 1 1
0 0 1 1
Possible paths :
(0, 0) (1, 0) (1, 1) (1, 2) (2, 2) (2, 3) (3, 3)  -> Total Steps: 7
(0, 0) (1, 0) (1, 1) (1, 2) (2, 2) (3, 2) (3, 3)  -> Total Steps: 7
(0, 0) (1, 0) (1, 1) (1, 2) (2, 2) (2, 1) (2, 3) (3, 3)  -> Total Steps: 8
(0, 0) (1, 0) (1, 1) (1, 2) (3, 2) (3, 3)  -> Total Steps: 6
(0, 0) (1, 0) (1, 1) (1, 2) (3, 2) (2, 2) (2, 3) (3, 3)  -> Total Steps: 8
(0, 0) (1, 0) (1, 1) (1, 2) (3, 2) (2, 2) (2, 1) (2, 3) (3, 3)  -> Total Steps: 9
(0, 0) (1, 0) (1, 1) (2, 1) (2, 2) (2, 3) (3, 3)  -> Total Steps: 7
(0, 0) (1, 0) (1, 1) (2, 1) (2, 2) (3, 2) (3, 3)  -> Total Steps: 7
(0, 0) (1, 0) (1, 1) (2, 1) (2, 2) (1, 2) (3, 2) (3, 3)  -> Total Steps: 8
(0, 0) (1, 0) (1, 1) (2, 1) (2, 3) (3, 3)  -> Total Steps: 6
(0, 0) (1, 0) (1, 1) (2, 1) (2, 3) (2, 2) (3, 2) (3, 3)  -> Total Steps: 8
(0, 0) (1, 0) (1, 1) (2, 1) (2, 3) (2, 2) (1, 2) (3, 2) (3, 3)  -> Total Steps: 9
(0, 0) (1, 0) (1, 2) (2, 2) (2, 3) (3, 3)  -> Total Steps: 6
(0, 0) (1, 0) (1, 2) (2, 2) (3, 2) (3, 3)  -> Total Steps: 6
(0, 0) (1, 0) (1, 2) (2, 2) (2, 1) (2, 3) (3, 3)  -> Total Steps: 7
(0, 0) (1, 0) (1, 2) (3, 2) (3, 3)  -> Total Steps: 5
(0, 0) (1, 0) (1, 2) (3, 2) (2, 2) (2, 3) (3, 3)  -> Total Steps: 7
(0, 0) (1, 0) (1, 2) (3, 2) (2, 2) (2, 1) (2, 3) (3, 3)  -> Total Steps: 8
(0, 0) (1, 0) (1, 2) (1, 1) (2, 1) (2, 2) (2, 3) (3, 3)  -> Total Steps: 8
(0, 0) (1, 0) (1, 2) (1, 1) (2, 1) (2, 2) (3, 2) (3, 3)  -> Total Steps: 8
(0, 0) (1, 0) (1, 2) (1, 1) (2, 1) (2, 3) (3, 3)  -> Total Steps: 7
(0, 0) (1, 0) (1, 2) (1, 1) (2, 1) (2, 3) (2, 2) (3, 2) (3, 3)  -> Total Steps: 9

Process returned 0 (0x0)   execution time : 0.141 s
Press any key to continue.
```