CSE 4404-Algorithms Lab. Winter 2022
Date: January 10, 2023.
Target Group: A
Topic: BFS/DFS

Instructions:
- Task naming format: fullID_L02_T01_1A.c/CPP
- **Solutions with less efficient approaches will be considered for partial marks.**

Task 1

A friend of yours is doing research on the Traveling Knight Problem (TKP) where you are to find the shortest closed tour of knight moves that visits each square of a given set of 64 squares on a chessboard exactly once. He thinks that the most difficult part of the problem is determining the smallest number of knight moves between two given squares and that, once you have accomplished this, finding the tour would be easy. Of course you know that it is vice versa. So you offer him to write a program that solves the ”difficult” part. Your job is to write a program that takes two squares **a** and **b** as input and then determines the number of knight moves on a shortest route from **a** to **b**.

Mandatory Note: Apply BFS

Input

The input file will contain one or more test cases. Each test case consists of one line containing two squares separated by one space. A square is a string consisting of a letter **(a..h)** representing the column and a digit **(1..8)** representing the row on the chessboard.

Output

For each test case, print one line saying ‘To get from **xx** to **yy** takes **n** knight moves.’.

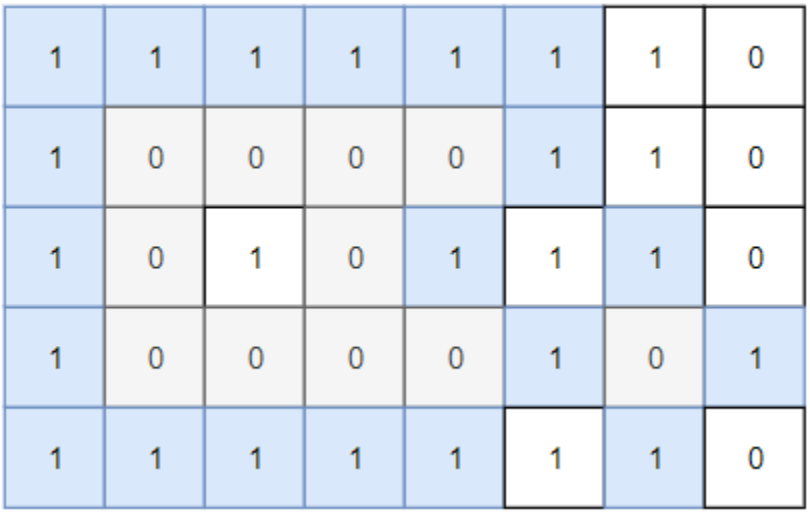
Sample Input	**Sample Output**
e2 e4 a1 b2 b2 c3 a1 h8 a1 h7 h8 a1 b1 c3 f6 f6	To get from e2 to e4 takes 2 knight moves. To get from a1 to b2 takes 4 knight moves. To get from b2 to c3 takes 2 knight moves. To get from a1 to h8 takes 6 knight moves. To get from a1 to h7 takes 5 knight moves. To get from h8 to a1 takes 6 knight moves. To get from b1 to c3 takes 1 knight move. To get from f6 to f6 takes 0 knight moves.

Task 2

Given a 2D grid consists of **0s** (land) and **1s** (water). An *island* is a maximal 4-directionally connected group of `0`s and a *closed island* is an island totally (all left, top, right, bottom) surrounded by **1s**. Your task is to output the number of ***closed islands*.**

Mandatory Note: Apply DFS

Example:



Sample Input	**Sample Output**
Grid = [[1,1,1,1,1,1,1,0],[1,0,0,0,0,1,1,0],[1,0,1,0,1,1,1,0],[1,0,0,0,0,1,0,1],[1,1,1,1,1,1,1,0]]	2

Explanation: Islands in gray are closed because they are completely surrounded by water (group of 1s).

BFS PseudoCode:

```
1  procedure BFS(G,source):
2      Q=queue(), level[]=infinity
3      Q.enqueue(source)
4      level[source]=0
5      while Q is not empty
6         u ← Q.pop()
7         for all edges from u to v in G.adjacentEdges(v) do
8             if level[v] = infinity:
9                       level[v] = level[u] + 1;
10                      Q.enqueue(v)
11            end if
12        end for
13       end while
14.   Return distance;
```

2D BFS PseudoCode:

```
#define pii pair<int,int>
int fx[]={1,-1,0,0}; //ডিরেকশন অ্যারে
int fy[]={0,0,1,-1};
int vis[100][100];
int d[100][100]; //d means destination from source, also known as level
int row,col;
void bfs(int sx,int sy) //Source node is in [sx][sy] cell.
{
	memset(vis,0,sizeof vis);
	vis[sx][sy]=1;
	queue<pii>q; //A queue containing STL pairs
	q.push(pii(sx,sy));
	while(!q.empty())
	{
		int x = q.front().first;
		int y = q.front().second;
		q.pop();
		for(int k=0;k<4;k++)
		{
			int tx = x + fx[k];
			int ty = y + fy[k]; //Neighbor cell [tx][ty]
			if(tx>=0 and tx<row and ty>=0 and ty<col and vis[tx][ty]==0)
//Check if the neighbor is valid and not visited before.
			{
				vis[tx][ty]=1;
				d[tx][ty]=d[x][y] + 1;
				q.push(pii(tx,ty)); //Pushing a new pair in the queue
			}
		}
	}
}
```