'''

Arjun wants to build a swimming pool, in the backyard of his farm-house.

The backyard has an empty land of size m\*n.

Some part of the backyard is used to build the swimming pool

You will be given the m\*n grid values (0's and 1's).

where 1 indicates swimming pool, and 0 indiactes empty land.

Your task to find the perimeter of the swimming pool.

Note: There is only one swimming pool.

Input Format:

-------------

Line-1: Two integers M and N, size of the backyard.

Next M lines: N space separated integers, either 0 or 1

0- represents empty land and 1- represents the swimming pool

Output Format:

--------------

Print an integer, the perimeter of the swimming pool

Sample Input-1:

---------------

4 4

0 1 0 0

1 1 1 0

0 1 0 0

1 1 0 0

Sample Output-1:

----------------

16

Sample Input-2:

---------------

1 2

1 0

Sample Output-2:

----------------

4

For explanation of sample test cases given refer Hint-1.

Write your python code below

'''

m,n=list(map(int,input().split()))

l=[]

# l1=[]

p=0

i=0

j=0

for i in range(0,m):

l1=list(map(int,input().split()))

l.append(l1)

for k in range(0,m):

for w in range(0,n):

if(l[k][w]==1):

i=k

j=w

break

# print(i,j)

def dfs(l,r,c,m,n):

if(r< 0 or r>=m or c<0 or c>=n or l[r][c]==0):

return 1

if(l[r][c]==-1):

return 0;

l[r][c]=-1

# if(l[r-1][c]==0):

# p+=1

# if(l[r][c+1]==0):

# p+=1

# if(l[r][c-1]==0):

# p+=1

# if(l[r+1][c]==0):

# p+=1

p=dfs(l,r-1,c,m,n)

p+=dfs(l,r,c+1,m,n)

p+=dfs(l,r,c-1,m,n)

p+=dfs(l,r+1,c,m,n)

return p

res=dfs(l,i,j,m,n)

print(res)

another more accurate soln:

# Online Python compiler (interpreter) to run Python online.

# Write Python 3 code in this online editor and run it.

'''

Arjun wants to build a swimming pool, in the backyard of his farm-house.

The backyard has an empty land of size m\*n.

Some part of the backyard is used to build the swimming pool

You will be given the m\*n grid values (0's and 1's).

where 1 indicates swimming pool, and 0 indiactes empty land.

Your task to find the perimeter of the swimming pool.

Note: There is only one swimming pool.

Input Format:

-------------

Line-1: Two integers M and N, size of the backyard.

Next M lines: N space separated integers, either 0 or 1

0- represents empty land and 1- represents the swimming pool

Output Format:

--------------

Print an integer, the perimeter of the swimming pool

Sample Input-1:

---------------

4 4

0 1 0 0

1 1 1 0

0 1 0 0

1 1 0 0

Sample Output-1:

----------------

16

Sample Input-2:

---------------

1 2

1 0

Sample Output-2:

----------------

4

For explanation of sample test cases given refer Hint-1.

Write your python code below

'''

m,n=list(map(int,input().split()))

l=[]

# l1=[]

# k=0

max\_per=0

p=0

i=0

j=0

w=0

for i in range(0,m):

l1=list(map(int,input().split()))

l.append(l1)

# print(i,j)

def dfs(l,r,c,m,n):

if(r< 0 or r>=m or c<0 or c>=n or l[r][c]==0):

return 1

if(l[r][c]==-1):

return 0;

l[r][c]=-1

p=dfs(l,r-1,c,m,n)

p+=dfs(l,r,c+1,m,n)

p+=dfs(l,r,c-1,m,n)

p+=dfs(l,r+1,c,m,n)

return p

for k in range(m):

for w in range(n):

if(l[k][w]==1):

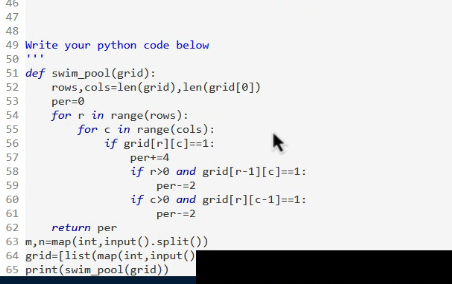
w=dfs(l,k,w,m,n)

max\_per=max(max\_per,w)

print(max\_per)

# res=dfs(l,i,j,m,n)

# print(res)



Kiran is playing a Jumping game. He is playing the game on a flat surface XY.

Initially, Kiran's position is at the center (x,y) = (0,0).

He can jump one unit of length and the direction of the jump

is either upside U,downside D, rightside R, or leftside L.

You are given jumping sequence of Kiran as a string, contains only UDLR characters.

Now your task is to find out, at the end of all the jumps in the sequence,

the position of Kiran is his initial position or not.

If yes, print true.

Otherwise, print false

Input Format:

-------------

A String jumps, jumping sequence.

(jumps contains only U, R, L, D letters)

Output Format:

--------------

Print a boolean value.

Sample Input-1:

---------------

UDRL

Sample Output-1:

----------------

true

Explanation:

------------

Kiran's initial position is (0,0) and jumps are UDRL:

U -> jump from (0,0) to (0,1)

D -> jump from (0,1) to (0,0)

R -> jump from (0,0) to (1,0)

L -> jump from (1,0) to (0,0)

His final position is (0,0). So, return true.

Sample Input-2:

---------------

UURRLD

Sample Output-2:

----------------

false

Explanation:

------------

Kiran's initial position is (0,0) and jumps are UURRLD:

U -> jump from (0,0) to (0,1)

U -> jump from (0,1) to (0,2)

R -> jump from (0,2) to (1,2)

R -> jump from (1,2) to (2,2)

L -> jump from (2,2) to (1,2)

D> jump from (1,2) to (1,1)

His final position is (1,1). So, return false..

import java.util.\*;

public class Main{

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

String s=sc.next();

int r=0;

int c=0;

for(int i=0;i<s.length();i++){

if(s.charAt(i)=='U'){

r-=1;

}

else if(s.charAt(i)=='R'){

c+=1;

}

else if(s.charAt(i)=='L'){

c-=1;

}

else{

r+=1;

}

}

if(r==0 && c==0){

System.out.println(true);

}

else {

System.out.println(false);

}

}

}

Kiran is playing a Jumping game. He is playing the game on a flat surface XY.

Initially, Kiran's position is at the center (x,y) = (0,0).

He can jump one unit of length and the direction of the jump

is either upside U,downside D, rightside R, or leftside L.

You are given jumping sequence of Kiran as a string, contains only UDLR characters.

Now your task is to find out, at the end of all the jumps in the sequence,

the position of Kiran is his initial position or not.

If yes, print true.

Otherwise, print false

Input Format:

-------------

A String jumps, jumping sequence.

(jumps contains only U, R, L, D letters)

Output Format:

--------------

Print a boolean value.

Sample Input-1:

---------------

UDRL

Sample Output-1:

----------------

true

Explanation:

------------

Kiran's initial position is (0,0) and jumps are UDRL:

U -> jump from (0,0) to (0,1)

D -> jump from (0,1) to (0,0)

R -> jump from (0,0) to (1,0)

L -> jump from (1,0) to (0,0)

His final position is (0,0). So, return true.

Sample Input-2:

---------------

UURRLD

Sample Output-2:

----------------

false

Explanation:

------------

Kiran's initial position is (0,0) and jumps are UURRLD:

U -> jump from (0,0) to (0,1)

U -> jump from (0,1) to (0,2)

R -> jump from (0,2) to (1,2)

R -> jump from (1,2) to (2,2)

L -> jump from (2,2) to (1,2)

D> jump from (1,2) to (1,1)

His final position is (1,1). So, return false..

import java.util.\*;

public class Main{

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

String s=sc.next();

int r=0;

int c=0;

for(int i=0;i<s.length();i++){

if(s.charAt(i)=='U'){

r-=1;

}

else if(s.charAt(i)=='R'){

c+=1;

}

else if(s.charAt(i)=='L'){

c-=1;

}

else{

r+=1;

}

}

if(r==0 && c==0){

System.out.println(true);

}

else {

System.out.println(false);

}

}

}

Given an undirected graph with V vertices and a number M, the task is to color

the given graph with at most M colors such that no two adjacent vertices of

the graph are colored with the same color. Color codes are start from 1 to M.

You need to assign the color code to any vertex V, whihch is the least possible

color code value.

Your task is to check whether it is possible or not.

Print 'true' if it is possible, Otherwise print 'false'.

Note:

- In the given graph, there are no self edges.

- Coloring of a graph means the assignment of colors to all vertices.

Input Format:

-------------

Line-1: An integer V and M,

Next V lines: V space separated integers, only 1's and 0's.

Output Format:

--------------

if possible, print the coloring arrangement of V vertices.

Otherwise, print "No Solution"

Sample Input-1:

---------------

4 3

0 1 1 1

1 0 1 0

1 1 0 1

1 0 1 0

Sample Output-1:

----------------

1 2 3 2

Sample Input-2:

---------------

4 2

0 1 1 1

1 0 1 1

1 1 0 1

1 1 1 0

Sample Output-2:

----------------

No Solution

import java.util.\*;

public class Main{

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

int v=sc.nextInt();

int m=sc.nextInt();

int[][] arr=new int[v][v];

for(int i=0;i<v;i++){

for(int j=0;j<v;j++){

arr[i][j]=sc.nextInt();

}

}

ArrayList<Edge>[] graph=new ArrayList[v];

int[] colors=new int[v];

creategraph(arr,graph);

// for(int i=0;i<graph.length;i++){

// System.out.println(fun(0,graph,m,colors));

// }

if(fun(0,graph,m,colors)){

System.out.println(Arrays.toString(colors));

}

else{

System.out.println("No Solution");

}

}

public static boolean fun(int n,ArrayList<Edge>[] graph,int m,int[] colors){

if(n==graph.length){

return true;

}

for(int i=1;i<=m;i++){

if(check(graph[n],colors,i)){

colors[n]=i;

if(fun(n+1,graph,m,colors)){

return true;

}

colors[n]=0;

}

}

return false;

}

public static boolean check(ArrayList<Edge> l,int[] colors,int col){

for(int i=0;i<l.size();i++){

Edge e=l.get(i);

if(colors[e.dest]==col){

return false;

}

}

return true;

}

public static class Edge{

int src;

int dest;

Edge(int src,int dest){

this.src=src;

this.dest=dest;

}

}

public static void creategraph(int[][] arr,ArrayList<Edge> [] graph){

for(int i=0;i<arr.length;i++){

graph[i]=new ArrayList<Edge>();

}

for(int i=0;i<arr.length;i++){

for(int j=0;j<arr.length;j++){

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

graph[i].add(new Edge(i,j));

}

}

}

}

}

