'''

A digit sum is the sum of all the digits of a number.

e.g., 123=> 1 + 2 + 3 => 6, So, digit sum of 123 is 6.

You are given an integer N.

Find the digit sum of each number from 1 to N.

And group them according to their digit sum.

Your task is to find and print the number of groups have the largest size.

Input Format:

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

An integer N

Output Format:

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

Print an integer, number of groups with largest size.

Sample Input-1:

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

13

Sample Output-1:

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

4

Explanation:

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

There are 9 groups formed: [1,10], [2,11], [3,12], [4,13], [5], [6], [7], [8], [9].

There are 4 groups having largest size-2.

Sample Input-2:

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

24

Sample Output-2:

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

5

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

Write your python code below

'''

n=int(input())

def fun(n):

r=0

sum=0

while(n>0):

r=int(n%10)

sum+=r

n=n//10

return sum

d=dict(())

# k=0

for i in range(1,n+1):

k=fun(i)

if k in d:

d[k]+=1

else:

d[k]=1

# print(d)

count=0

max=max(d.values())

for i in d:

if(d[i]==max):

count+=1

print(count)

Mr Hacker has given a color code system as a grid of size R\*C, the color codes

are in the range of 1 to 10 and the grid is 0-indexed.

He has given a task to crack the original grid by using three values, Ri, Cj, Code.

The way to crack the original color codes in the grid is as follows:

1. You have to start updating the color codes from (Ri,Cj) position in

the grid with the given color code, 'Code'.

2. Updating color codes means, replace the color codes from the cell (Ri,Cj)

in the grid, and replace all the cells connected in 4 directions

(Up, Down, Left, Right) and having same color-code of starting cell.

3. Repeat the step-2 from the updated cells until no more cells to update.

Your task is to help Mr.Hacker to get the original color-code system grid

and print it.

Input Format:

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

Line-1: Two space sepearted integers, R and C.

Next R lines: C space separated integers, grid[][].

Last line: Three space sepaarted integers, Ri,Cj and Code.

Output Format:

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

Print the resultant grid.

Sample Input-1:

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

3 4

1 0 1 1

0 1 1 1

1 1 0 1

1 3 3

Sample Output-1:

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

1 0 3 3

0 3 3 3

3 3 0 3

Sample Input-2:

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

3 4

1 0 1 1

0 1 1 1

1 1 0 1

0 0 3

Sample Output-2:

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

3 0 1 1

0 1 1 1

1 1 0 1

This code is correct but u are getting out of memory so try to optimize this-it is leetcode flood fill solution

import java.util.\*;

public class Main{

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

int rr=sc.nextInt();

int cr=sc.nextInt();

int[][] grid=new int[rr][cr];

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

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

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

}

}

int sr=sc.nextInt();

int scc=sc.nextInt();

int code=sc.nextInt();

Queue<Edge> q=new LinkedList<>();

q.add(new Edge(sr,scc));

while(!q.isEmpty()){

Edge curr=q.poll();

int r=curr.r;

int c=curr.c;

if(r-1>=0 && grid[r-1][c]==grid[r][c]){

grid[r-1][c]=code;

q.add(new Edge(r-1,c));

}

if(r+1<grid.length && grid[r+1][c]==grid[r][c]){

grid[r+1][c]=code;

q.add(new Edge(r+1,c));

}

if(c-1>=0 && grid[r][c-1]==grid[r][c]){

grid[r][c-1]=code;

q.add(new Edge(r,c-1));

}

if(c+1<grid[r].length && grid[r][c+1]==grid[r][c]){

grid[r][c+1]=code;

q.add(new Edge(r,c+1));

}

grid[r][c]=code;

}

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

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

System.out.print(grid[i][j]+" ");

}

System.out.println();

}

}

// public static int[][] fun()

static class Edge{

int r;

int c;

Edge(int r,int c){

this.r=r;

this.c=c;

}

}

}

For this below code I am getting only 87.5 per correct

import java.util.\*;

public class Main{

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

int rr=sc.nextInt();

int cr=sc.nextInt();

int[][] grid=new int[rr][cr];

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

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

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

}

}

int sr=sc.nextInt();

int scc=sc.nextInt();

int code=sc.nextInt();

dfs(grid,sr,scc,code,grid[sr][scc]);

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

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

System.out.print(grid[i][j]+" ");

}

System.out.println();

}

}

public static void dfs(int[][] grid,int r,int c,int code,int oldcode){

if(r<0 || r>=grid.length || c<0 || c>=grid[r].length|| grid[r][c]!=oldcode){

return;

}

grid[r][c]=code;

dfs(grid,r-1,c,code,oldcode);

dfs(grid,r+1,c,code,oldcode);

dfs(grid,r,c-1,code,oldcode);

dfs(grid,r,c+1,code,oldcode);

}

// grid[r][c]=code;

// }

}

Few people are living in a township of size N\*N, where each 1\*1 part

of the township is either a villa or a swimming pool.

You are given the structure of township as a 2d matrix of size N\*N,

filled with 0's and 1's, where '0'-indicates a swimming pool and

'1'- indiactes a villa.

Your task is to find a swimming pool, such that its distance to

the nearest villa(s) is maximized, and return the distance.

If the township contains only the villas or only swimming pools, print '-1'.

The distance used in this problem is the Manhattan distance:

the distance between two cells (a0, b0) and (a1, b1) is |a0 - a1| + |b0 - b1|

Input Format:

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

Line-1: An integer N, size of the 2d matrix.

Next N lines: N space separated integers, matrix[][] either 0 or 1.

Output Format:

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

An integer, maximum distance.

Sample Input-1:

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

4

1 0 1 1

0 0 0 0

1 0 1 0

1 0 0 1

Sample Output-1:

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

2

Explanation:

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

The swimming pool at (1, 1) is with distance 2 from the nearest villas.

Sample Input-2:

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

4

1 0 0 0

0 0 0 0

1 0 0 0

0 1 0 1

Sample Output-2:

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

3

Explanation:

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

The swimming pool at (0,3) or (1, 2) are with distance 3 from the nearest villas.



