'''

A graduate at University of California (UoC) has to follow certain rules.

UoC offering K subjects, the subjects are indexed from 0 to K-1.

To complete the subjets, you have to follow the conditions, like

condition[i]= [Xi, Yi], which specifies you must take the subject Xi

before the l Yi.

Conditions can be indirect, i.e., take course-A to take course-B, and take

course-B to take course-C, then you have to take course-A to take course-C.

You are given total number of subjects, a list of C condition pairs and

a list of Q queries where queries[i] = [Ai, Bi]. For the ith query,

you should answer whether you have to take course Ai to take course Bi or not

and return the answers as a boolean list answers[].

NOTE:

It is guaranteed that the course conditional graph continas no cycle.

Input Format:

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

Line-1: 3 space separated integers, N, C and Q

Next C lines: Two space separated integers, conditions[]

Next Q lines: Two space separated integers, conditions[]

Output Format:

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

Print Q boolean values as result.

Sample Input-1:

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

3 3 2

1 2

1 0

2 0

1 0

1 2

Sample Output-1:

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

true true

Sample Input-2:

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

5 4 3

0 1

0 2

1 4

1 3

0 2

1 3

1 2

Sample Output-2:

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

true true false

'''

-Implemented Time Complexity Algorithm [O(C + Q)]

n,c,q=list(map(int,input().split()))

l=[]

for i in range(c):

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

l.append(l1)

r=[]

for i in range(q):

r1=list(map(int,input().split()))

r.append(r1)

d=dict()

for i in range(n):

d[i]=[]

for i in l:

d[i[1]].append(i[0])

if(i[0] in d.keys() ):

d[i[1]].extend(d[i[0]])

# print(d)

for i in r:

if(i[0] in d[i[1]] ):

print("true",end=" ")

else:

print("false",end=" ")

There is a Game where in fighter consumes power[i] power on the i-th hour.

Given an integer k, for every consecutive sequence of k hours (power[i],

power[i+1], ..., power[i+k-1] for all 0 <= i <= n-k), they look at T,

the total power consumed during that sequence of k hours (power[i] +

power[i+1] + ... + power[i+k-1]):

If T < lower, fighter performed poorly and lose 1 point;

If T > upper, fighter performed well and gain 1 point;

Otherwise, fighter performed normally and there is no change in points.

Initially, the fighter has zero points. Return the total number of points the

fighter has after consuming power for power.length hours.

Note that the total points can be negative.

Input Format

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

Line-1: An integer N, the arrray length.

Line-2: Array of integers, power[]

Line-3: Three integers -subarray length K, lower value and upper value

output Format

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

An integer total points

Sample Input-1:

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

4

7 10 0 0

2 1 12

Sample Output-1:

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

0

Explanation:

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

Since k = 2, we consider subarrays of length 2.

power[0] + power[1] > upper so 1 point is gained.

lower <= power[1] + power[2] <= upper so no change in points.

power[2] + power[3] < lower so 1 point is lost.

Sample Input-2:

2

5 10

2 0 1

Sample Output-2:

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

1

Explanation:

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

Since k = 2, we consider subarrays of length 2.

power[0] + power[1] > upper so 1 point is gained.

Example 3:

Sample Input-3:

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

5

1 2 3 4 5

1 3 3

Sample Output-3:

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

0

Explanation:

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

Since k = 1, Since k = 2, we consider subarrays of length 1. and compare it

to lower and upper. power[0] and power[1] are less than lower so 2 points are lost.

power[3] and power[4] are greater than upper so 2 points are gained.

import java.util.\*;

public class Main{

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int[] arr=new int[n];

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

arr[i]=sc.nextInt();

}

int k=sc.nextInt();

int lower=sc.nextInt();

int upper=sc.nextInt();

int sum=0;

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

sum+=arr[i];

}

int p=0;

if(sum<lower){

p-=1;

}

else if(sum>upper){

p+=1;

}

for(int i=1;i<n-k+1;i++){

sum-=arr[i-1];

sum+=arr[k+i-1];

if(sum<lower){

p-=1;

}

else if(sum>upper){

p+=1;

}

}

System.out.println(p);

}

}

Skandhanshi Infra Township constructed N Villas in a row.

They planned to paint the villas with three colors,

snow-white, sky-blue and light-green.

And adjacent villas should not paint with same color.

You are given the cost of painting each villa with these colors, cost[N][3].

cost[i][0] is cost of painting with snow-white.

cost[i][1] is cost of painting with sky-blue.

cost[i][2] is cost of painting with light-green.

where, 0 <= i < N.

Your task is to find the minimum cost to paint all the villas.

Input Format:

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

Line-1: An integer N, number of villas.

Next N lines: 3 space separated integers, cost to paint the villas.

Output Format:

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

Print an integer, minimum cost to paint all the villas.

Sample Input:

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

3

17 2 17

16 4 5

14 3 19

Sample Output:

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

10

Explanation:

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

1st Villa painted with sky-blue, 2nd Villa painted with light-green,

3rd Villa painted with sky-blue.

Minimum cost: 2 + 5 + 3 = 10.

This is done using dp (learn dp first to understand this)

import java.util.\*;

public class Main{

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int[][] arr=new int[n][3];

int[][] dp=new int[n][4];

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

Arrays.fill(dp[i],-1);

}

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

arr[i][0]=sc.nextInt();

arr[i][1]=sc.nextInt();

arr[i][2]=sc.nextInt();

}

System.out.println(fun(0,-1,n,dp,arr));

}

public static int fun(int j,int k,int n,int[][] dp,int[][] c){

if(j==n){

return 0;

}

if(dp[j][k+1]!=-1){

return dp[j][k+1];

}

int s=Integer.MAX\_VALUE;

for(int i=0;i<3;i++){

if(i!=k){

s=Math.min(s,c[j][i]+fun(j+1,i,n,dp,c));

}

}

return dp[j][k+1]=s;

}

}

In AMB mall, to attract kids and to increase their economy they setup

a separate Fun Zone. They opened a stall by name Soda Bears. We have an

LED display panel with different colours of Soda bears.

You will be given a 2D integer panel representing the grid of a

Soda bears, we have different color-codes as positive integer in

panel [p][q] of each coloured soda bears. If a cell in

panel[p][q]=0 representing at position (p,q) is empty.

The given panel represents the state of game according to participants move.

Now it’s your aim to make panel to a stable state by merging Soda Bears

with certain conditions:

1. If three or more soda bears of the same colour are adjacent

vertically or horizontally,"merge" them all at the same time -

these locations become empty.

2. After merging all Soda Bears simultaneously, if an empty space on

the panel has soda bears on top of itself, then these bears will drop

until they hit another bear or bottom at the same time.

(No new soda bear will drop outside the top boundary.)

3. After the above steps, there may exist more bears that can be merged.

If so, you need to repeat the above steps.

4. If there does not exist more bears for merge (ie. the panel is stable),

then return the current panel.

Repeat the procedure for stable panel, then return the current panel state.

Input Format:

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

Line-1: Two space separated integers, M and N size of panel

Next M lines: N space separated integers, color codes of soda bears.

Output Format:

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

Print the stable format of panel.

Sample Input-1:

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

8 4

11 5 13 5

12 13 5 13

1 2 3 4

11 2 3 4

2 2 2 5

13 13 3 4

14 13 13 14

12 12 11 13

Sample Output-1:

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

0 0 0 5

11 0 0 13

12 0 0 4

1 0 0 4

11 0 13 5

13 0 5 4

14 5 13 14

12 12 11 13

Sample Input-2:

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

7 8

1 1 1 1 1 1 1 1

2 1 1 1 1 1 1 2

3 2 1 1 1 1 2 3

4 3 2 1 1 2 3 4

3 2 1 1 1 1 2 3

2 1 1 1 1 1 1 2

1 1 1 1 1 1 1 1

Sample Output-2:

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

0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0

2 0 0 0 0 0 0 2

3 0 0 0 0 0 0 3

4 2 0 0 0 0 2 4

3 3 0 0 0 0 3 3

