'''

In Dubai's Burj Khalifa, there is an elevator moves only in upwards direction,

the elevator can carry N members.

The people are waiting for the elevator at different floors, made P requests,

request[i] = [ num\_people, enter\_floor, exit\_floor ], each request indicates,

number of people to enter into elevator, entering floor number,

exiting floor number.

Initially the elevator is empty.

Your task is to find and return true, iff it is possible to enter the people

into elevator and exit from elevator of all the requests made by the people.

Input Format:

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

Line-1 -> Two integers P and N, number of requests and number of members

can be carried by elevator.

Next N Lines -> three space separated integers, num\_people, enter\_floor, exit\_floor.

Output Format:

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

Print a boolean value.

Sample Input-1:

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

2 5

2 1 5

3 3 7

Sample Output-1:

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

true

Sample Input-2:

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

2 4

2 1 5

3 3 7

Sample Output-2:

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

false

Sample Input-3:

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

3 11

3 2 7

3 7 9

8 3 9

Sample Output-3:

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

true

'''

d=dict()

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

flag=True

req=[]

for i in range(r):

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

req.append(r1);

# print(req)

for i in range(r):

for j in range(req[i][1],req[i][2]):

if j not in d:

d[j]=req[i][0]

if(d[j]>n):

# print(d[j])

flag=False

else:

d[j]+=req[i][0]

if(d[j]>n):

# print(d[j])

flag=False

# print(d)

print(flag)

In a joint family, every person assigned with an ID, an integer value.

and the entire family is arranged in the from of tree.

You will be given the family tree and two persons IDs P1 and P2.

Your task is to find the person ID, who is the latest common ascendant of P1 and P2.

and return the Lowest Common Ascendant ID.

Implement the class Solution:

- Node lowestCommonAscendant(Node root, Node P1, Node P2):

return the person ID who is the latest common ascendant of P1 and P2.

Input Format:

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

Line-1: Single line of space separated integers, person ID's in the family.

Line-2: Two Person IDs, P1 and P2.

Output Format:

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

Return the latest common ascendant of P1 and P2.

Sample Input-1:

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

3 5 1 6 2 0 8 -1 -1 7 4

6 4

Sample Output-1:

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

5

Sample Input-2:

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

11 99 88 77 22 33 66 55 10 20 30 40 50 60 44

66 55

Sample Output-2:

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

11

You are given a phrase and a paper of size m\*n.

So you can type exactly m \* n characters on that paper.

i.e,. there are 'm' rows and in each row you can type 'n' characters

You need to type the phrase on the paper with some rules.

The rules are as follows:

- A word in the phrase cannot be split into two lines.

- The order of words in the phrase shuld not change.

- Two consecutive words in a line must be separated by a single space.

Your task is to find out how many times the phrase can be typed on the paper.

Constraint:

-----------

Length of each word is <=10.

Input Format:

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

Line-1: Three space separated integers m, n, and s, grid size m\*n, number of words.

Line-2: 's' space separated strings, set of words

Output Format:

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

Print an integer, no.of times set of words fit into the grid.

Sample Input-1:

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

2 8 2

socail distance

Sample Output-1:

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

1

Explanation:

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

social\_\_

distance

Sample Input-2:

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

3 6 3

a bc def

Sample Output-2:

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

2

Explanation:

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

a\_bc\_\_

def\_a\_

bc\_def

import java.util.\*;

public class Main{

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

int m=sc.nextInt();

int n=sc.nextInt();

int s=sc.nextInt();

sc.nextLine();

String str=sc.nextLine();

String[] arr=str.split(" ");

int l=arr.length;

int j=0;

int count=0;

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

int size=n;

boolean flag=true;

while(flag){

if(size<=0){

break;

}

if(arr[j%l].length()<=size){

size-=arr[j%l].length();

if(size>0){

size-=1;

}

j++;

}

else{

flag=false;

break;

}

if(j%l==0){

count+=1;

}

// j+=1;

}

}

System.out.println(count);

}

}

Clavius, the person who suggested grouping of data using brackets.

He has suggested that the group of brackets should be Well-Formed.

A string is said to be Well-Formed, if:

- The string is empty.

- The string can be written as MN, (M is appended to N),

where M and N are Well-Formed Strings, or

- The string can be written as {M}, where M is Well-Formed string.

You will be given an integer N. Your task is to return the list of all

Well-Formed strings of length 2\*N.

Input Format:

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

An integer N.

Output Format:

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

Print the list of well-formed strings.

Sample Input-1:

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

3

Sample Output-1:

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

[{{{}}},{{}{}},{{}}{},{}{{}},{}{}{}]

Sample Input-2:

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

1

Sample Output-2:

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

[{}]

import java.util.\*;

public class Main{

static ArrayList<String> res=new ArrayList<>();

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

fun(n,"",0,0);

System.out.println(res);

}

public static void fun(int n,String s,int left,int right){

if(s.length()==2\*n){

res.add(s);

return;

}

if(left<n){

fun(n,s+"{",left+1,right);

}

if(right<left){

fun(n,s+"}",left,right+1);

}

}

}