In front of a kirana shop, there are N persons in a line,

Each of them waiting to pay some amount(-ve) or to recieve some amount(+ve).

You are given an array of N integers amounts[] , amount[i] is the amount

to pay or recieve. You have to choose a positive amount 'startAmount', and start.

In each iteration, you calculate the step by step addition of

amount of 'startAmount' with amounts in nums[] (from left to right).

You have to return the minimum positive amount of startAmount

such that the step by step amount is never less than 1.

Input Format:

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

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

Line-2: N space separated integers, amounts[].

Output Format:

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

Print an integer, minimum positive amount.

Sample Input-1:

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

5

-3 2 -3 4 2

Sample Output-1:

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

5

Explanation:

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

If you choose startAmount = 4, in the third iteration your step by step sum

is less than 1.

step by step sum startAmount = 4 | startAmount = 5| nums

(4 -3 ) = 1 | (5 -3 ) = 2 | -3

(1 +2 ) = 3 | (2 +2 ) = 4 | 2

(3 -3 ) = 0 | (4 -3 ) = 1 | -3

(0 +4 ) = 4 | (1 +4 ) = 5 | 4

(4 +2 ) = 6 | (5 +2 ) = 7 | 2

Sample Input-2:

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

3

1 -2 -3

Sample Output-2:

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

5

My soln:

n=int(input())

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

max=0

for i in l:

max+=int(abs(i))

# print(max)

for i in range(1,max):

sum=i

for j in l:

sum+=j

if(sum<1):

break

if(sum>=1):

print(i)

break

# print(k)

A number is called self-supportive if all the digits of the number are

factors of the number. For example, 48 is a self-supportive number,

because 48 % 4 == 0, and 48 % 8 == 0.

A number is not a self-supportive if it has any digit as zero.

Given two Positive numbers start and end, return a set of all

the self-supportive numbers in between start and end (both inclusive).

1<=start<=end<=10^5

Input Format:

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

Line: 2 space seperated integers start and end.

Output Format:

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

Print a space seperated list.

Sample Input-1:

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

20 25

Sample Output-1:

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

22 24

Explanation:

-----------

20 has 0 as digit so it's not self - supportive.

21 is not divisible by 2. so it's not self - supportive.

22 is divisible 2. so it's self - supportive.

23 is not divisible by both the digits 2 and 3. so it's not self - supportive.

24 is divisible by both 2 and 4. so it is self - supportive.

So 22 and 24 are self-supportive.

Sample Input-2:

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

50 80

Sample Output-2:

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

55 66 77

My soln:

import java.util.\*;

public class Main{

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

int n1=sc.nextInt();

int n2=sc.nextInt();

for(int i=n1;i<n2;i++){

if(fun(i,i

)){

System.out.print(i+" ");

}

}

}

public static boolean fun(int k,int n){

if(k==0){

return false;

}

if(n<=0){

return true;

}

int r=n%10;

if(r==0){

return false;

}

if(k%r!=0){

return false;

}

else {

return fun(k,n/10);

}

}

}

Karthik is going on a world tour, He prefers to travel in airplanes.

He is given a list of planes avaialble , where a plane[i]=[src-i, dest-i].

i.e, plane[i] indiactes airplane from source city to destination city.

Your mission is to discover both the starting and ending cities of Karthk's

world tour.

Note:

-------

Karthik visits all of the cities listed, there will be no loop.

Input Format:

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

Line-1: An integer N, number of airplanes routes.

next N lines: two comma (',') separated Strings, source and destination.

Output Format:

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

Print the output according to the sample.

e.g., startcity-endcity

Sample Input-1:

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

3

California,New York

New York,Sydney

Sydney,New Delhi

Sample Output-1:

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

California-New Delhi

Explanation:

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

If you start from 'New York' you cannot visit California.

If you start from 'Sydney' you cannot visit New York and California cities.

So, neither 'New York' nor 'Sydney' is your starting city.

Sample Input-2:

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

5

New Delhi,Kerala

Kerala,Kashmir

Japan,New York

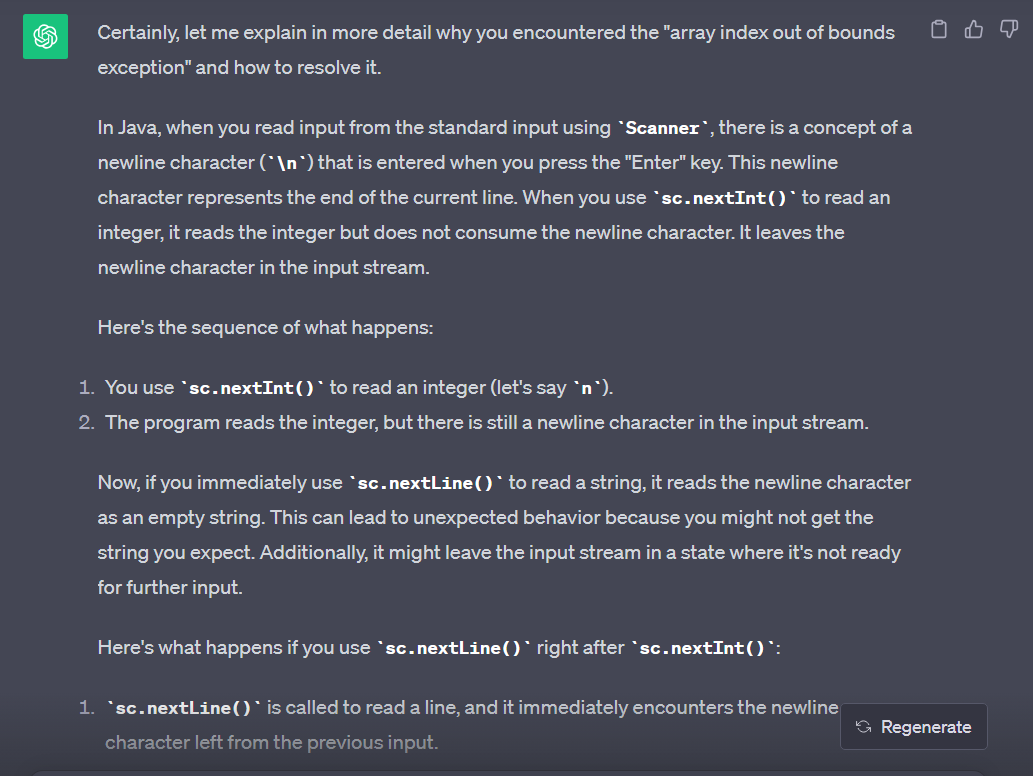
New York,Sydney

Sydney,New Delhi

Sample Output-2:

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

Japan-Kashmir



My soln:

import java.util.\*;

public class Main{

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

sc.nextLine();

String[] arr=new String[n];

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

arr[i]=sc.nextLine();

}

String[][] loc=new String[n][2];

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

loc[i]=arr[i].split(",");

}

Set<String> s=new LinkedHashSet<>();

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

if(s.contains(loc[i][0])){

s.remove(loc[i][0]);

}

else{

s.add(loc[i][0]);

}

if(s.contains(loc[i][1])){

s.remove(loc[i][1]);

}

else{

s.add(loc[i][1]);

}

}

ArrayList<String> k1=new ArrayList<>();

ArrayList<String> k2=new ArrayList<>();

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

k1.add(loc[i][0]);

k2.add(loc[i][1]);

}

String [] sss=new String[2];

int count=0;

if(s.size()==2){

for(String i: s){

sss[count]=i;

count+=1;

}

}

// for (String i: sss)

if(k1.contains(sss[0])){

System.out.println(sss[0]+"-"+sss[1]);

// break;

}

else{

System.out.println(sss[1]+"-"+sss[0]);

// break;

}

}

}