Educational Codeforces Round 90 (Rated for Div. 2)

A. Donut Shops

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

There are two rival donut shops.

The first shop sells donuts at retail: each donut costs a dollars.

The second shop sells donuts only in bulk: box of b donuts costs c dollars. So if you want to buy x donuts from this shop, then you have to buy the smallest number of boxes such that the total number of donuts in them is greater or equal to x.

You want to determine two **positive integer** values:

1. how many donuts can you buy so that they are strictly cheaper in the first shop than in the second shop?
2. how many donuts can you buy so that they are strictly cheaper in the second shop than in the first shop?

If any of these values doesn't exist then that value should be equal to -1. If there are multiple possible answers, then print any of them.

The printed values should be less or equal to 10^9. It can be shown that under the given constraints such values always exist if any values exist at all.

Input
The first line contains a single integer t ($1 \le t \le 1000$) — the number of testcases.

Each of the next t lines contains three integers a, b and c ($1 \le a \le 10^9$, $2 \le b \le 10^9$, $1 \le c \le 10^9$).

Output
For each testcase print two **positive** integers. For both shops print such x that buying x donuts in this shop is strictly cheaper than buying x donuts in the other shop. x should be greater than 0 and less or equal to 10^9.

If there is no such x, then print -1. If there are multiple answers, then print any of them.

Example

input
4 5 10 4 4 5 20 2 2 3 1000000000 1000000000 1000000000
output
-1 20 8 -1 1 2 -1 1000000000

Note
In the first testcase buying any number of donuts will be cheaper in the second shop. For example, for 3 or 5 donuts you'll have to buy a box of 10 donuts for 4 dollars. 3 or 5 donuts in the first shop would cost you 15 or 25 dollars, respectively, however. For 20 donuts you'll have to buy two boxes for 8 dollars total. Note that 3 and 5 are also valid answers for the second shop, along with many other answers.

In the second testcase buying any number of donuts will be either cheaper in the first shop or the same price. 8 donuts cost 32 dollars in the first shop and 40 dollars in the second shop (because you have to buy two boxes). 10 donuts will cost 40 dollars in both shops, so 10 is not a valid answer for any of the shops.

In the third testcase 1 donut costs 2 and 3 dollars, respectively. 2 donuts cost 4 and 3 dollars. Thus, 1 is a valid answer for the first shop and 2 is a valid answer for the second shop.

In the fourth testcase 10^9 donuts cost 10^{18} dollars in the first shop and 10^9 dollars in the second shop.

B. 01 Game

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input

output: standard output

Alica and Bob are playing a game.

Initially they have a binary string s consisting of only characters 0 and 1.

Alice and Bob make alternating moves: Alice makes the first move, Bob makes the second move, Alice makes the third one, and so on. During each move, the current player must choose two **different adjacent** characters of string s and delete them. For example, if $s = 1011001$ then the following moves are possible:

1. delete s_1 and s_2: $\mathbf{10}11001 \rightarrow 11001$;
2. delete s_2 and s_3: $1\mathbf{01}1001 \rightarrow 11001$;
3. delete s_4 and s_5: $1011\mathbf{10}01 \rightarrow 10101$;
4. delete s_6 and s_7: $1011\mathbf{01} \rightarrow 10110$.

If a player can't make any move, they lose. Both players play optimally. You have to determine if Alice can win.

Input
First line contains one integer t ($1 \leq t \leq 1000$) — the number of test cases.

Only line of each test case contains one string s ($1 \leq |s| \leq 100$), consisting of only characters 0 and 1.

Output
For each test case print answer in the single line.

If Alice can win print DA (YES in Russian) in any register. Otherwise print NET (NO in Russian) in any register.

Example

input
3 01 1111 0011
output
DA NET NET

Note
In the first test case after Alice's move string s become empty and Bob can not make any move.

In the second test case Alice can not make any move initially.

In the third test case after Alice's move string s turn into 01. Then, after Bob's move string s become empty and Alice can not make any move.

C. Pluses and Minuses

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given a string s consisting only of characters + and -. You perform some process with this string. This process can be described by the following pseudocode:

```
res = 0
for init = 0 to inf
   cur = init
   ok = true
   for i = 1 to |s|
      res = res + 1
      if s[i] == '+'
         cur = cur + 1
      else
         cur = cur - 1
      if cur < 0
         ok = false
         break
   if ok
      break
```

Note that the inf denotes infinity, and the characters of the string are numbered from 1 to $|s|$.

You have to calculate the value of the res after the process ends.

Input
The first line contains one integer t ($1 \le t \le 1000$) — the number of test cases.

The only lines of each test case contains string s ($1 \le |s| \le 10^6$) consisting only of characters + and -.

It's guaranteed that sum of $|s|$ over all test cases doesn't exceed 10^6.

Output
For each test case print one integer — the value of the res after the process ends.

Example

input
3 --+- --- ++--+-
output
7 9 6

D. Maximum Sum on Even Positions

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given an array a consisting of n integers. Indices of the array start from zero (i. e. the first element is a_0, the second one is a_1, and so on).

You can reverse **at most one** subarray (continuous subsegment) of this array. Recall that the subarray of a with borders l and r is $a[l;r] = a_l, a_{l+1}, \ldots, a_r$.

Your task is to reverse such a subarray that the sum of elements on **even** positions of the resulting array is **maximized** (i. e. the sum of elements $a_0, a_2, \ldots, a_{2k}$ for integer $k = \lfloor \frac{n-1}{2} \rfloor$ should be maximum possible).

You have to answer t independent test cases.

Input
The first line of the input contains one integer t ($1 \le t \le 2 \cdot 10^4$) — the number of test cases. Then t test cases follow.

The first line of the test case contains one integer n ($1 \le n \le 2 \cdot 10^5$) — the length of a. The second line of the test case contains n integers $a_0, a_1, \ldots, a_{n-1}$ ($1 \le a_i \le 10^9$), where a_i is the i-th element of a.

It is guaranteed that the sum of n does not exceed $2 \cdot 10^5$ ($\sum n \le 2 \cdot 10^5$).

Output
For each test case, print the answer on the separate line — the **maximum** possible sum of elements on **even** positions after reversing **at most one** subarray (continuous subsegment) of a.

Example

input
4 8 1 7 3 4 7 6 2 9 5 1 2 1 2 1 10 7 8 4 5 7 6 8 9 7 3 4 3 1 2 1
output
26 5 37 5

E. Sum of Digits

time limit per test: 2 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

Let $f(x)$ be the sum of digits of a decimal number x.

Find the smallest non-negative integer x such that $f(x) + f(x+1) + \cdots + f(x+k) = n$.

Input
The first line contains one integer t ($1 \le t \le 150$) — the number of test cases.

Each test case consists of one line containing two integers n and k ($1 \le n \le 150$, $0 \le k \le 9$).

Output
For each test case, print one integer without leading zeroes. If there is no such x that $f(x) + f(x+1) + \cdots + f(x+k) = n$, print -1; otherwise, print the minimum x meeting that constraint.

Example

input
7 1 0 1 1 42 7 13 7 99 1 99 0 99 2
output
1 0 4 -1 599998 99999999999 7997

F. Network Coverage

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

The government of Berland decided to improve network coverage in his country. Berland has a unique structure: the capital in the center and n cities *in a circle* around the capital. The capital already has a good network coverage (so the government ignores it), but the i-th city contains a_i households that require a connection.

The government designed a plan to build n network stations between all pairs of neighboring cities which will maintain connections only for these cities. In other words, the i-th network station will provide service only for the i-th and the $(i+1)$-th city (the n-th station is connected to the n-th and the 1-st city).

All network stations have capacities: the i-th station can provide the connection to at most b_i households.

Now the government asks you to check can the designed stations meet the needs of all cities or not — that is, is it possible to assign each household a network station so that each network station i provides the connection to at most b_i households.

Input
The first line contains a single integer t ($1 \le t \le 10^4$) — the number of test cases.

The first line of each test case contains the single integer n ($2 \le n \le 10^6$) — the number of cities and stations.

The second line of each test case contains n integers ($1 \le a_i \le 10^9$) — the number of households in the i-th city.

The third line of each test case contains n integers ($1 \le b_i \le 10^9$) — the capacities of the designed stations.

It's guaranteed that the sum of n over test cases doesn't exceed 10^6.

Output
For each test case, print YES, if the designed stations can meet the needs of all cities, or NO otherwise (case insensitive).

Example

input
5 3 2 3 4 3 3 3 3 3 3 3 2 3 4 4 2 3 4 5 3 7 2 2 4

```
4 5 2 3
2 3 2 7
2
1 1
10 10
```

output

```
YES
YES
NO
YES
YES
```

Note

In the first test case:

- the first network station can provide 2 connections to the first city and 1 connection to the second city;
- the second station can provide 2 connections to the second city and 1 connection to the third city;
- the third station can provide 3 connections to the third city.

In the second test case:

- the 1-st station can provide 2 connections to the 1-st city;
- the 2-nd station can provide 3 connections to the 2-nd city;
- the 3-rd station can provide 3 connections to the 3-rd city and 1 connection to the 1-st station.

In the third test case, the fourth city needs 5 connections, but the third and the fourth station has 4 connections in total.

G. Pawns

time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given a chessboard consisting of n rows and n columns. Rows are numbered from bottom to top from 1 to n. Columns are numbered from left to right from 1 to n. The cell at the intersection of the x-th column and the y-th row is denoted as (x, y). Furthermore, the k-th column is a special column.

Initially, the board is empty. There are m changes to the board. During the i-th change one pawn is added or removed from the board. The current board is good if we can move all pawns to the special column by the followings rules:

- Pawn in the cell (x, y) can be moved to the cell $(x, y + 1)$, $(x - 1, y + 1)$ or $(x + 1, y + 1)$;
- You can make as many such moves as you like;
- Pawns can not be moved outside the chessboard;
- Each cell can not contain more than one pawn.

The current board may not always be good. To fix it, you can add new rows to the board. New rows are added at the top, i. e. they will have numbers $n + 1, n + 2, n + 3, \ldots$.

After each of m changes, print one integer — the minimum number of rows which you have to add to make the board good.

Input

The first line contains three integers n, k and m ($1 \le n, m \le 2 \cdot 10^5; 1 \le k \le n$) — the size of the board, the index of the special column and the number of changes respectively.

Then m lines follow. The i-th line contains two integers x and y ($1 \le x, y \le n$) — the index of the column and the index of the row respectively. If there is no pawn in the cell (x, y), then you add a pawn to this cell, otherwise — you remove the pawn from this cell.

Output

After each change print one integer — the minimum number of rows which you have to add to make the board good.

Example

input

```
5 3 5
4 4
3 5
2 4
3 4
3 5
```

output

```
0
1
2
2
1
```