Codeforces Round #605 (Div. 3)

A. Three Friends

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

Three friends are going to meet each other. Initially, the first friend stays at the position $x = a$, the second friend stays at the position $x = b$ and the third friend stays at the position $x = c$ on the coordinate axis Ox.

In one minute **each friend independently** from other friends can change the position x by 1 to the left or by 1 to the right (i.e. set $x := x - 1$ or $x := x + 1$) or even don't change it.

Let's introduce the total pairwise distance — the sum of distances between each pair of friends. Let a', b' and c' be the final positions of the first, the second and the third friend, correspondingly. Then the total pairwise distance is $|a' - b'| + |a' - c'| + |b' - c'|$, where $|x|$ is the absolute value of x.

Friends are interested in the minimum total pairwise distance they can reach if they will move optimally. **Each friend will move no more than once**. So, more formally, they want to know the minimum total pairwise distance they can reach after one minute.

You have to answer q independent test cases.

Input

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

The next q lines describe test cases. The i-th test case is given as three integers a, b and c ($1 \le a, b, c \le 10^9$) — initial positions of the first, second and third friend correspondingly. The positions of friends can be equal.

Output

For each test case print the answer on it — the minimum total pairwise distance (the minimum sum of distances between each pair of friends) if friends change their positions optimally. **Each friend will move no more than once**. So, more formally, you have to find the minimum total pairwise distance they can reach after one minute.

Example

input
8 3 3 4 10 20 30 5 5 5 2 4 3 1 1000000000 1000000000 1 1000000000 999999999 3 2 5 3 2 6
output
0 36 0 0 1999999994 1999999994 2 4

B. Snow Walking Robot

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

Recently you have bought a snow walking robot and brought it home. Suppose your home is a cell $(0, 0)$ on an infinite grid.

You also have the sequence of instructions of this robot. It is written as the string s consisting of characters 'L', 'R', 'U' and 'D'. If the robot is in the cell (x, y) right now, he can move to one of the adjacent cells (depending on the current instruction).

- If the current instruction is 'L', then the robot can move to the left to $(x - 1, y)$;
- if the current instruction is 'R', then the robot can move to the right to $(x + 1, y)$;
- if the current instruction is 'U', then the robot can move to the top to $(x, y + 1)$;

- if the current instruction is 'D', then the robot can move to the bottom to $(x, y-1)$.

You've noticed the warning on the last page of the manual: if the robot visits some cell (**except** $(0, 0)$) twice then it breaks.

So the sequence of instructions is valid if the robot starts in the cell $(0, 0)$, performs the given instructions, visits no cell other than $(0, 0)$ two or more times and ends the path in the cell $(0, 0)$. Also cell $(0, 0)$ should be visited **at most** two times: at the beginning and at the end (if the path is empty then it is visited only once). For example, the following sequences of instructions are considered valid: "UD", "RL", "UUURULLDDDDLDDRRUU", and the following are considered invalid: "U" (the endpoint is not $(0, 0)$) and "UUDD" (the cell $(0, 1)$ is visited twice).

The initial sequence of instructions, however, might be not valid. You don't want your robot to break so you decided to reprogram it in the following way: you will remove some (possibly, all or none) instructions from the initial sequence of instructions, then rearrange the remaining instructions as you wish and turn on your robot to move.

Your task is to remove as few instructions from the initial sequence as possible and rearrange the remaining ones so that the sequence is valid. Report the valid sequence of the maximum length you can obtain.

Note that you can choose **any** order of remaining instructions (you don't need to minimize the number of swaps or any other similar metric).

You have to answer q independent test cases.

Input

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

The next q lines contain test cases. The i-th test case is given as the string s consisting of at least 1 and no more than 10^5 characters 'L', 'R', 'U' and 'D' — the initial sequence of instructions.

It is guaranteed that the sum of $|s|$ (where $|s|$ is the length of s) does not exceed 10^5 over all test cases ($\sum |s| \le 10^5$).

Output

For each test case print the answer on it. In the first line print the maximum number of remaining instructions. In the second line print the valid sequence of remaining instructions t the robot has to perform. The moves are performed from left to right in the order of the printed sequence. If there are several answers, you can print any. If the answer is 0, you are allowed to print an empty line (but you can don't print it).

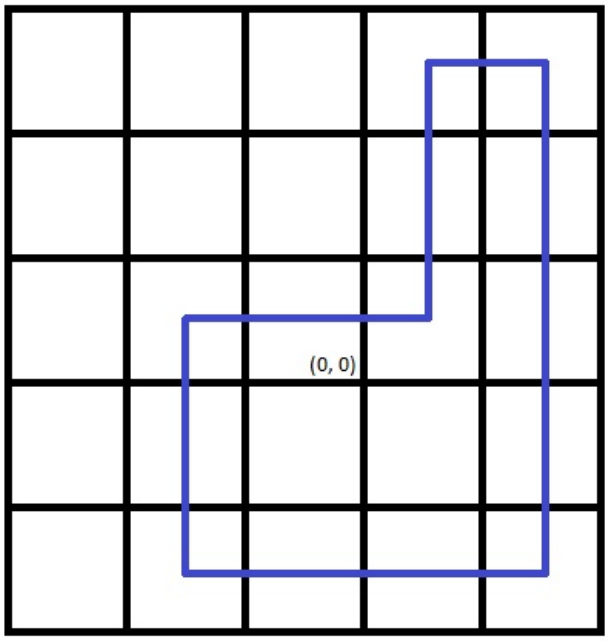
Example

input
6 LRU DURLDRUDRULRDURDDL LRUDDLRUDRUL LLLLRRRR URDUR LLL
output
2 LR 14 RUURDDDDLLLUUR 12 ULDDDRRRUULL 2 LR 2 UD 0

Note

There are only two possible answers in the first test case: "LR" and "RL".

The picture corresponding to the second test case:



Note that the direction of traverse does not matter

Another correct answer to the third test case: "URDDLLLUURDR".

C. Yet Another Broken Keyboard

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

Recently, Norge found a string $s = s_1 s_2 \ldots s_n$ consisting of n lowercase Latin letters. As an exercise to improve his typing speed, he decided to type all substrings of the string s. Yes, all $\frac{n(n+1)}{2}$ of them!

A substring of s is a non-empty string $x = s[a \ldots b] = s_a s_{a+1} \ldots s_b$ ($1 \le a \le b \le n$). For example, "auto" and "ton" are substrings of "automaton".

Shortly after the start of the exercise, Norge realized that his keyboard was broken, namely, he could use only k Latin letters $c_1, c_2, \ldots, c_k$ out of 26.

After that, Norge became interested in how many substrings of the string s he could still type using his broken keyboard. Help him to find this number.

Input

The first line contains two space-separated integers n and k ($1 \le n \le 2 \cdot 10^5$, $1 \le k \le 26$) — the length of the string s and the number of Latin letters still available on the keyboard.

The second line contains the string s consisting of exactly n lowercase Latin letters.

The third line contains k space-separated distinct lowercase Latin letters $c_1, c_2, \ldots, c_k$ — the letters still available on the keyboard.

Output

Print a single number — the number of substrings of s that can be typed using only available letters $c_1, c_2, \ldots, c_k$.

Examples

input
7 2 abacaba a b
output
12

input
10 3 sadfaasdda f a d
output
21

input

7 1 aaaaaaa b
output
0

Note
In the first example Norge can print substrings $s[1\ldots2]$, $s[2\ldots3]$, $s[1\ldots3]$, $s[1\ldots1]$, $s[2\ldots2]$, $s[3\ldots3]$, $s[5\ldots6]$, $s[6\ldots7]$, $s[5\ldots7]$, $s[5\ldots5]$, $s[6\ldots6]$, $s[7\ldots7]$.

D. Remove One Element

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.

You can remove **at most one** element from this array. Thus, the final length of the array is $n-1$ or n.

Your task is to calculate the maximum possible length of the **strictly increasing** contiguous subarray of the remaining array.

Recall that the contiguous subarray a with indices from l to r is $a[l\ldots r]=a_l,a_{l+1},\ldots,a_r$. The subarray $a[l\ldots r]$ is called strictly increasing if $a_l<a_{l+1}<\cdots<a_r$.

Input
The first line of the input contains one integer n ($2\le n\le 2\cdot10^5$) — the number of elements in a.

The second line of the input contains n integers $a_1,a_2,\ldots,a_n$ ($1\le a_i\le10^9$), where a_i is the i-th element of a.

Output
Print one integer — the maximum possible length of the **strictly increasing** contiguous subarray of the array a after removing at most one element.

Examples

input
5 1 2 5 3 4
output
4

input
2 1 2
output
2

input
7 6 5 4 3 2 4 3
output
2

Note
In the first example, you can delete $a_3=5$. Then the resulting array will be equal to $[1,2,3,4]$ and the length of its largest increasing subarray will be equal to 4.

E. Nearest Opposite Parity

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. In one move, you can jump from the position i to the position $i-a_i$ (if $1\le i-a_i$) or to the position $i+a_i$ (if $i+a_i\le n$).

For each position i from 1 to n you want to know the minimum the number of moves required to reach any position j such that a_j has the opposite parity from a_i (i.e. if a_i is odd then a_j has to be even and vice versa).

Input
The first line of the input contains one integer n ($1 \le n \le 2 \cdot 10^5$) — the number of elements in a.

The second line of the input contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le n$), where a_i is the i-th element of a.

Output
Print n integers $d_1, d_2, \ldots, d_n$, where d_i is the minimum the number of moves required to reach any position j such that a_j has the opposite parity from a_i (i.e. if a_i is odd then a_j has to be even and vice versa) or -1 if it is impossible to reach such a position.

Example

input
10 4 5 7 6 7 5 4 4 6 4
output
1 1 1 2 -1 1 1 3 1 1

F. Two Bracket Sequences

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

You are given two bracket sequences (not necessarily regular) s and t consisting only of characters '(' and ')'. You want to construct the shortest **regular** bracket sequence that contains both given bracket sequences as **subsequences** (not necessarily contiguous).

Recall what is the regular bracket sequence:

- () is the regular bracket sequence;
- if S is the regular bracket sequence, then (S) is a regular bracket sequence;
- if S and T regular bracket sequences, then ST (concatenation of S and T) is a regular bracket sequence.

Recall that the subsequence of the string s is such string t that can be obtained from s by removing some (possibly, zero) amount of characters. For example, "coder", "force", "cf" and "cores" are subsequences of "codeforces", but "fed" and "z" are not.

Input
The first line of the input contains one bracket sequence s consisting of no more than 200 characters '(' and ')'.

The second line of the input contains one bracket sequence t consisting of no more than 200 characters '(' and ')'.

Output
Print one line — the shortest **regular** bracket sequence that contains both given bracket sequences as **subsequences** (not necessarily contiguous). If there are several answers, you can print any.

Examples

input
(())(() ()))()
output
(())()()

input
) ((
output
(())

input
))))
output
((()))

input
()) (()(()((
output
(()()(()()))