Codeforces Round #627 (Div. 3)

A. Yet Another Tetris Problem

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

You are given some Tetris field consisting of n columns. The initial height of the i-th column of the field is a_i blocks. On top of these columns you can place **only** figures of size 2×1 (i.e. the height of this figure is 2 blocks and the width of this figure is 1 block). Note that you **cannot** rotate these figures.

Your task is to say if you can clear the whole field by placing such figures.

More formally, the problem can be described like this:

The following process occurs while **at least one** a_i **is greater than** 0:

1. You place one figure 2×1 (choose some i from 1 to n and replace a_i with $a_i + 2$);
2. then, while all a_i are greater than zero, replace each a_i with $a_i - 1$.

And your task is to determine if it is possible to clear the whole field (i.e. finish the described process), choosing the places for new figures properly.

You have to answer t independent test cases.

Input

The first line of the input contains one integer t ($1 \le t \le 100$) — the number of test cases.

The next $2t$ lines describe test cases. The first line of the test case contains one integer n ($1 \le n \le 100$) — the number of columns in the Tetris field. The second line of the test case contains n integers $a_1, a_2, \dots, a_n$ ($1 \le a_i \le 100$), where a_i is the initial height of the i-th column of the Tetris field.

Output

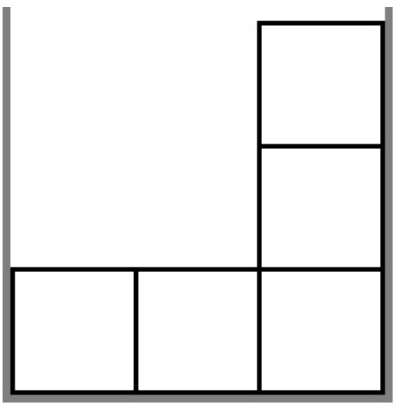
For each test case, print the answer — "YES" (without quotes) if you can clear the whole Tetris field and "NO" otherwise.

Example

input
4 3 1 1 3 4 1 1 2 1 2 11 11 1 100
output
YES NO YES YES

Note

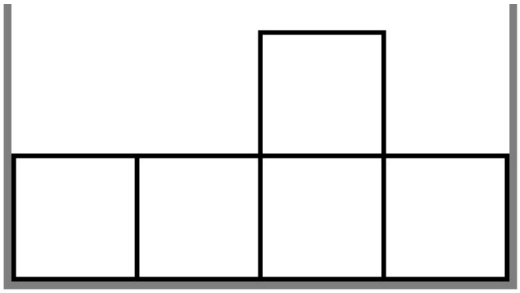
The first test case of the example field is shown below:



Gray lines are bounds of the Tetris field. Note that the field has no upper bound.

One of the correct answers is to first place the figure in the first column. Then after the second step of the process, the field becomes $[2, 0, 2]$. Then place the figure in the second column and after the second step of the process, the field becomes $[0, 0, 0]$.

And the second test case of the example field is shown below:



It can be shown that you cannot do anything to end the process.

In the third test case of the example, you first place the figure in the second column after the second step of the process, the field becomes $[0, 2]$. Then place the figure in the first column and after the second step of the process, the field becomes $[0, 0]$.

In the fourth test case of the example, place the figure in the first column, then the field becomes $[102]$ after the first step of the process, and then the field becomes $[0]$ after the second step of the process.

B. Yet Another Palindrome Problem

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.

Your task is to determine if a has some **subsequence** of length at least 3 that is a palindrome.

Recall that an array b is called a **subsequence** of the array a if b can be obtained by removing some (possibly, zero) elements from a (not necessarily consecutive) without changing the order of remaining elements. For example, $[2]$, $[1, 2, 1, 3]$ and $[2, 3]$ are subsequences of $[1, 2, 1, 3]$, but $[1, 1, 2]$ and $[4]$ are not.

Also, recall that a palindrome is an array that reads the same backward as forward. In other words, the array a of length n is the palindrome if $a_i = a_{n-i-1}$ for all i from 1 to n. For example, arrays $[1234]$, $[1, 2, 1]$, $[1, 3, 2, 2, 3, 1]$ and $[10, 100, 10]$ are palindromes, but arrays $[1, 2]$ and $[1, 2, 3, 1]$ are not.

You have to answer t independent test cases.

Input

The first line of the input contains one integer t ($1 \le t \le 100$) — the number of test cases.

Next $2t$ lines describe test cases. The first line of the test case contains one integer n ($3 \le n \le 5000$) — the length of a. The second line of the test case 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.

It is guaranteed that the sum of n over all test cases does not exceed 5000 ($\sum n \le 5000$).

Output

For each test case, print the answer — "YES" (without quotes) if a has some **subsequence** of length at least 3 that is a palindrome and "NO" otherwise.

Example

input
5 3 1 2 1 5 1 2 2 3 2 3 1 1 2 4 1 2 2 1 10 1 1 2 2 3 3 4 4 5 5
output
YES YES NO YES NO

Note
In the first test case of the example, the array a has a subsequence $[1, 2, 1]$ which is a palindrome.

In the second test case of the example, the array a has two subsequences of length 3 which are palindromes: $[2, 3, 2]$ and $[2, 2, 2]$.

In the third test case of the example, the array a has no subsequences of length at least 3 which are palindromes.

In the fourth test case of the example, the array a has one subsequence of length 4 which is a palindrome: $[1, 2, 2, 1]$ (and has two subsequences of length 3 which are palindromes: both are $[1, 2, 1]$).

In the fifth test case of the example, the array a has no subsequences of length at least 3 which are palindromes.

C. Frog Jumps

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

There is a frog staying to the left of the string $s = s_1 s_2 \ldots s_n$ consisting of n characters (to be more precise, the frog initially stays at the cell 0). Each character of s is either 'L' or 'R'. It means that if the frog is staying at the i-th cell and the i-th character is 'L', the frog can jump only to the left. If the frog is staying at the i-th cell and the i-th character is 'R', the frog can jump only to the right. **The frog can jump only to the right from the cell** 0.

Note that the frog can jump into the same cell twice and can perform as many jumps as it needs.

The frog wants to reach the $n + 1$-th cell. The frog chooses some **positive integer** value d **before the first jump** (and cannot change it later) and jumps by no more than d cells at once. I.e. if the i-th character is 'L' then the frog can jump to any cell in a range $[max(0, i - d); i - 1]$, and if the i-th character is 'R' then the frog can jump to any cell in a range $[i + 1; min(n + 1; i + d)]$.

The frog doesn't want to jump far, so your task is to find the minimum possible value of d such that the frog can reach the cell $n + 1$ from the cell 0 if it can jump by no more than d cells at once. **It is guaranteed that it is always possible to reach** $n + 1$ **from** 0.

You have to answer t independent test cases.

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

The next t lines describe test cases. The i-th test case is described as a string s consisting of at least 1 and at most $2 \cdot 10^5$ characters 'L' and 'R'.

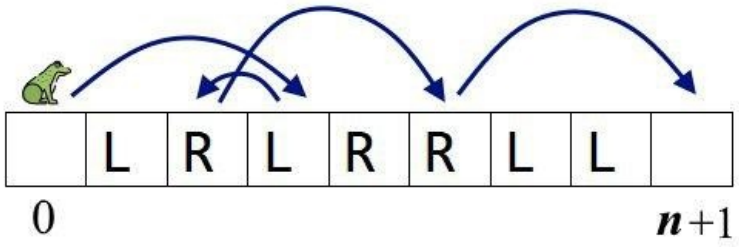
It is guaranteed that the sum of lengths of strings over all test cases does not exceed $2 \cdot 10^5$ ($\sum |s| \le 2 \cdot 10^5$).

Output
For each test case, print the answer — the minimum possible value of d such that the frog can reach the cell $n + 1$ from the cell 0 if it jumps by no more than d at once.

Example

input
6 LRLRRLL L LLR RRRR LLLLLL R
output
3 2 3 1 7 1

Note
The picture describing the first test case of the example and one of the possible answers:



In the second test case of the example, the frog can only jump directly from 0 to $n+1$.

In the third test case of the example, the frog can choose $d=3$, jump to the cell 3 from the cell 0 and then to the cell 4 from the cell 3.

In the fourth test case of the example, the frog can choose $d=1$ and jump 5 times to the right.

In the fifth test case of the example, the frog can only jump directly from 0 to $n+1$.

In the sixth test case of the example, the frog can choose $d=1$ and jump 2 times to the right.

D. Pair of Topics

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

The next lecture in a high school requires two topics to be discussed. The i-th topic is interesting by a_i units for the teacher and by b_i units for the students.

The pair of topics i and j ($i < j$) is called **good** if $a_i + a_j > b_i + b_j$ (i.e. it is more interesting for the teacher).

Your task is to find the number of **good** pairs of topics.

Input

The first line of the input contains one integer n ($2 \le n \le 2 \cdot 10^5$) — the number of topics.

The second line of the input contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$), where a_i is the interestingness of the i-th topic for the teacher.

The third line of the input contains n integers $b_1, b_2, \ldots, b_n$ ($1 \le b_i \le 10^9$), where b_i is the interestingness of the i-th topic for the students.

Output

Print one integer — the number of **good** pairs of topic.

Examples

input
5 4 8 2 6 2 4 5 4 1 3
output
7

input
4 1 3 2 4 1 3 2 4
output
0

E. Sleeping Schedule

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

Vova had a pretty weird sleeping schedule. There are h hours in a day. Vova will sleep exactly n times. The i-th time he will sleep exactly after a_i hours from the time he woke up. You can assume that Vova woke up exactly at the beginning of this story (the initial time is 0). Each time Vova sleeps **exactly one day** (in other words, h hours).

Vova thinks that the i-th sleeping time is **good** if he starts to sleep between hours l and r inclusive.

Vova can control himself and before the i-th time can choose between two options: go to sleep after a_i hours or after $a_i - 1$ hours.

Your task is to say the maximum number of **good** sleeping times Vova can obtain if he acts optimally.

Input
The first line of the input contains four integers n, h, l and r ($1 \le n \le 2000, 3 \le h \le 2000, 0 \le l \le r < h$) — the number of times Vova goes to sleep, the number of hours in a day and the segment of the **good** sleeping time.

The second line of the input contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i < h$), where a_i is the number of hours after which Vova goes to sleep the i-th time.

Output
Print one integer — the maximum number of **good** sleeping times Vova can obtain if he acts optimally.

Example

input
7 24 21 23 16 17 14 20 20 11 22
output
3

Note
The maximum number of **good** times in the example is 3.

The story starts from $t = 0$. Then Vova goes to sleep after $a_1 - 1$ hours, now the time is 15. This time is not good. Then Vova goes to sleep after $a_2 - 1$ hours, now the time is $15 + 16 = 7$. This time is also not good. Then Vova goes to sleep after a_3 hours, now the time is $7 + 14 = 21$. This time is **good**. Then Vova goes to sleep after $a_4 - 1$ hours, now the time is $21 + 19 = 16$. This time is not good. Then Vova goes to sleep after a_5 hours, now the time is $16 + 20 = 12$. This time is not good. Then Vova goes to sleep after a_6 hours, now the time is $12 + 11 = 23$. This time is **good**. Then Vova goes to sleep after a_7 hours, now the time is $23 + 22 = 21$. This time is also **good**.

F. Maximum White Subtree

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

You are given a tree consisting of n vertices. A tree is a connected undirected graph with $n - 1$ edges. Each vertex v of this tree has a color assigned to it ($a_v = 1$ if the vertex v is white and 0 if the vertex v is black).

You have to solve the following problem for each vertex v: what is the maximum difference between the number of white and the number of black vertices you can obtain if you choose some subtree of the given tree that **contains** the vertex v? The subtree of the tree is the connected subgraph of the given tree. More formally, if you choose the subtree that contains cnt_w white vertices and cnt_b black vertices, you have to maximize $cnt_w - cnt_b$.

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

The second line of the input contains n integers $a_1, a_2, \ldots, a_n$ ($0 \le a_i \le 1$), where a_i is the color of the i-th vertex.

Each of the next $n - 1$ lines describes an edge of the tree. Edge i is denoted by two integers u_i and v_i, the labels of vertices it connects ($1 \le u_i, v_i \le n, u_i \ne v_i$).

It is guaranteed that the given edges form a tree.

Output
Print n integers $res_1, res_2, \ldots, res_n$, where res_i is the maximum possible difference between the number of white and black vertices in some subtree that contains the vertex i.

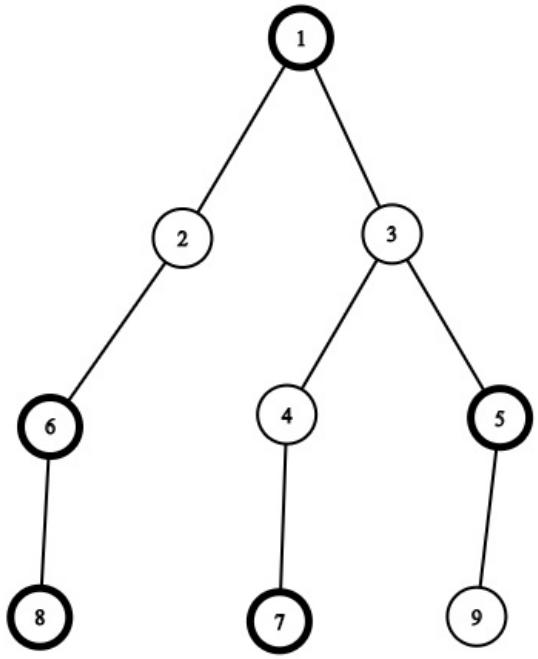
Examples

input
9 0 1 1 1 0 0 0 0 1 1 2 1 3 3 4 3 5 2 6 4 7 6 8 5 9
output
2 2 2 2 2 1 1 0 2

input
4 0 0 1 0 1 2 1 3 1 4
output
0 -1 1 -1

Note

The first example is shown below:



The black vertices have bold borders.

In the second example, the best subtree for vertices $2, 3$ and 4 are vertices $2, 3$ and 4 correspondingly. And the best subtree for the vertex 1 is the subtree consisting of vertices 1 and 3.