A. Vasya and Coins

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

Vasya decided to go to the grocery store. He found in his wallet a coins of 1 burle and b coins of 2 burles. He does not yet know the total cost of all goods, so help him find out s ($s > 0$): the **minimum** positive integer amount of money he **cannot** pay without change or pay at all using only his coins.

For example, if $a = 1$ and $b = 1$ (he has one 1-burle coin and one 2-burle coin), then:

- he can pay 1 burle without change, paying with one 1-burle coin,
- he can pay 2 burle without change, paying with one 2-burle coin,
- he can pay 3 burle without change by paying with one 1-burle coin and one 2-burle coin,
- he cannot pay 4 burle without change (moreover, he cannot pay this amount at all).

So for $a = 1$ and $b = 1$ the answer is $s = 4$.

Input

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

The description of each test case consists of one line containing two integers a_i and b_i ($0 \le a_i, b_i \le 10^8$) — the number of 1-burle coins and 2-burles coins Vasya has respectively.

Output

For each test case, on a separate line print one integer s ($s > 0$): the minimum positive integer amount of money that Vasya cannot pay without change or pay at all.

Example

input

```
5
1 1
4 0
0 2
0 0
2314 2374
```

output

```
4
5
1
1
7063
```

Note

- The first test case of the example is clarified into the main part of the statement.
- In the second test case, Vasya has only 1 burle coins, and he can collect either any amount from 1 to 4, but 5 can't.
- In the second test case, Vasya has only 2 burle coins, and he cannot pay 1 burle without change.
- In the fourth test case you don't have any coins, and he can't even pay 1 burle.

B. Vlad and Candies

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

Not so long ago, Vlad had a birthday, for which he was presented with a package of candies. There were n types of candies, there are a_i candies of the type i ($1 \le i \le n$).

Vlad decided to eat exactly one candy every time, choosing any of the candies of a type that is currently the most frequent (if there are several such types, he can choose **any** of them). To get the maximum pleasure from eating, Vlad **does not want** to eat two candies of the same type in a row.

Help him figure out if he can eat all the candies without eating two identical candies in a row.

Input

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

The following is a description of t test cases of input, two lines for each.

The first line of the case contains the single number n ($1 \le n \le 2 \cdot 10^5$) — the number of types of candies in the package.

The second line of the case contains n integers a_i ($1 \le a_i \le 10^9$) — the number of candies of the type i.

It is guaranteed that the sum of n for all cases does not exceed $2 \cdot 10^5$.

Output

Output t lines, each of which contains the answer to the corresponding test case of input. As an answer, output "YES" if Vlad can eat candy as planned, and "NO" otherwise.

You can output the answer in any case (for example, the strings "yEs", "yes", "Yes" and "YES" will be recognized as a positive answer).

Example

input

```
6
2
2 3
1
2
5
1 6 2 4 3
4
2 2 2 1
3
1 1000000000 999999999
1
1
```

output

```
YES
NO
NO
YES
YES
YES
```

Note

In the first example, it is necessary to eat sweets in this order:

- a candy of the type 2, it is the most frequent, now $a = [2, 2]$;
- a candy of the type 1, there are the same number of candies of the type 2, but we just ate one, now $a = [1, 2]$;
- a candy of the type 2, it is the most frequent, now $a = [1, 1]$;
- a candy of the type 1, now $a = [0, 1]$;
- a candy of the type 2, now $a = [0, 0]$ and the candy has run out.

In the second example, all the candies are of the same type and it is impossible to eat them without eating two identical ones in a row.

In the third example, first of all, a candy of the type 2 will be eaten, after which this kind will remain the only kind that is the most frequent, and you will have to eat a candy of the type 2 again.

C. Get an Even String

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

A string $a = a_1a_2\dots a_n$ is called *even* if it consists of a concatenation (joining) of strings of length 2 consisting of the same characters. In other words, a string a is even if two conditions are satisfied **at the same time**:

- its length n is even;
- for all odd i ($1 \le i \le n-1$), $a_i = a_{i+1}$ is satisfied.

For example, the following strings are even: "" (empty string), "tt", "aabb", "oooo", and "ttrrrroouuuuuuuukk". The following strings are not even: "aaa", "abab" and "abba".

Given a string s consisting of lowercase Latin letters. Find the minimum number of characters to remove from the string s to make it even. The deleted characters do not have to be consecutive.

Input

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

The descriptions of the test cases follow.

Each test case consists of one string s ($1 \le |s| \le 2 \cdot 10^5$), where $|s|$ — the length of the string s. The string consists of lowercase Latin letters.

It is guaranteed that the sum of $|s|$ on all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, print a single number — the minimum number of characters that must be removed to make s even.

Example

input

```
6
aabbdabdccc
zyx
aaababbb
aabbcc
oaoaaaoo
bmefbmuyw
```

output

```
3
3
2
0
2
7
```

Note

In the first test case you can remove the characters with indices 6, 7, and 9 to get an even string "aabbddcc".

In the second test case, each character occurs exactly once, so in order to get an even string, you must remove all characters from the string.

In the third test case, you can get an even string "aaaabb" by removing, for example, 4-th and 6-th characters, or a string "aabbbb" by removing the 5-th character and any of the first three.

D. Maximum Product Strikes Back

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. For each i ($1 \le i \le n$) the following inequality is true: $-2 \le a_i \le 2$.

You can remove any number (possibly 0) of elements from the beginning of the array and any number (possibly 0) of elements from the end of the array. You are allowed to delete the whole array.

You need to answer the question: how many elements should be removed from the beginning of the array, and how many elements should be removed from the end of the array, so that the result will be an array whose product (multiplication) of elements is **maximal**. If there is more than one way to get an array with the maximum product of elements on it, you are allowed to output **any** of them.

The product of elements of an **empty** array (array of length 0) should be assumed to be 1.

Input

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

Then the descriptions of the input test cases follow.

The first line of each test case description contains an integer n ($1 \le n \le 2 \cdot 10^5$) —the length of array a.

The next line contains n integers $a_1, a_2, \dots, a_n$ ($|a_i| \le 2$) — elements of array a.

It is guaranteed that the sum of n over all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, output two non-negative numbers x and y ($0 \le x + y \le n$) — such that the product (multiplication) of the array numbers, after removing x elements from the beginning and y elements from the end, is maximal.

If there is more than one way to get the maximal product, it is allowed to output **any** of them. Consider the product of numbers on **empty** array to be 1.

Example

input

```
5
4
1 2 -1 2
```

```
3
1 1 -2
5
2 0 -2 2 -1
3
-2 -1 -1
3
-1 -2 -2
```

output

```
0 2
3 0
2 0
0 1
1 0
```

Note

In the first case, the maximal value of the product is 2. Thus, we can either delete the first three elements (obtain array $[2]$), or the last two and one first element (also obtain array $[2]$), or the last two elements (obtain array $[1, 2]$). Thus, in the first case, the answers fit: "`3 0`", or "`1 2`", or "`0 2`".

In the second case, the maximum value of the product is 1. Then we can remove all elements from the array because the value of the product on the empty array will be 1. So the answer is "`3 0`", but there are other possible answers.

In the third case, we can remove the first two elements of the array. Then we get the array: $[-2, 2, -1]$. The product of the elements of the resulting array is $(-2) \cdot 2 \cdot (-1) = 4$. This value is the maximum possible value that can be obtained. Thus, for this case the answer is: "`2 0`".

E. Matrix and Shifts

time limit per test: 2 seconds

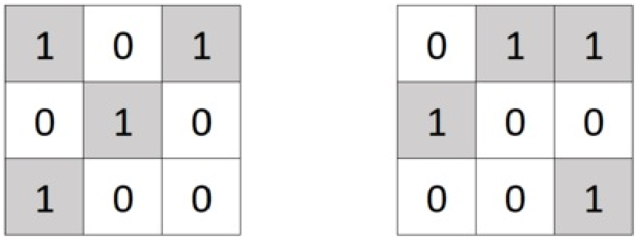
memory limit per test: 256 megabytes

input: standard input

output: standard output

You are given a binary matrix A of size $n \times n$. Rows are numbered from top to bottom from 1 to n, columns are numbered from left to right from 1 to n. The element located at the intersection of row i and column j is called A_{ij}. Consider a set of 4 operations:

1. Cyclically shift all rows up. The row with index i will be written in place of the row $i-1$ ($2 \le i \le n$), the row with index 1 will be written in place of the row n.
2. Cyclically shift all rows down. The row with index i will be written in place of the row $i+1$ ($1 \le i \le n-1$), the row with index n will be written in place of the row 1.
3. Cyclically shift all columns to the left. The column with index j will be written in place of the column $j-1$ ($2 \le j \le n$), the column with index 1 will be written in place of the column n.
4. Cyclically shift all columns to the right. The column with index j will be written in place of the column $j+1$ ($1 \le j \le n-1$), the column with index n will be written in place of the column 1.



The 3×3 matrix is shown on the left before the 3-rd operation is applied to it, on the right — after.

You can perform an arbitrary (possibly zero) number of operations on the matrix; the operations can be performed in any order.

After that, you can perform an arbitrary (possibly zero) number of new xor-operations:

- Select any element A_{ij} and assign it with new value $A_{ij} \oplus 1$. In other words, the value of $(A_{ij} + 1) \bmod 2$ will have to be written into element A_{ij}.

Each application of this xor-operation costs one burl. Note that the 4 shift operations — are free. These 4 operations can only be performed before xor-operations are performed.

Output the minimum number of burles you would have to pay to make the A matrix unitary. A *unitary matrix* is a matrix with ones on the main diagonal and the rest of its elements are zeros (that is, $A_{ij} = 1$ if $i = j$ and $A_{ij} = 0$ otherwise).

Input

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

The descriptions of the test cases follow. Before each test case, an empty line is written in the input.

The first line of each test case contains a single number n ($1 \le n \le 2000$)

This is followed by n lines, each containing exactly n characters and consisting only of zeros and ones. These lines describe the values in the elements of the matrix.

It is guaranteed that the sum of n^2 values over all test cases does not exceed $4 \cdot 10^6$.

Output

For each test case, output the minimum number of burles you would have to pay to make the A matrix unitary. In other words, print the minimum number of xor-operations it will take after applying cyclic shifts to the matrix for the A matrix to become unitary.

Example

input

```
4

3
010
011
100

5
00010
00001
10000
01000
00100

2
10
10

4
1111
1011
1111
1111
```

output

```
1
0
2
11
```

Note

In the first test case, you can do the following: first, shift all the rows down cyclically, then the main diagonal of the matrix will contain only "1". Then it will be necessary to apply xor-operation to the only "1" that is not on the main diagonal.

In the second test case, you can make a unitary matrix by applying the operation 2 — cyclic shift of rows upward twice to the matrix.

F1. Promising String (easy version)

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

This is the easy version of Problem F. The only difference between the easy version and the hard version is the constraints.

We will call a non-empty string **balanced** if it contains the same number of plus and minus signs. For example: strings "+--+" and "++-+--" are balanced, and strings "+--", "--" and "" are not balanced.

We will call a string **promising** if the string can be made balanced by several (possibly zero) uses of the following operation:

- replace two **adjacent** minus signs with one plus sign.

In particular, every balanced string is promising. However, the converse is not true: not every promising string is balanced.

For example, the string "-+---" is promising, because you can replace two adjacent minuses with plus and get a balanced string "-++-", or get another balanced string "-+-+".

How many non-empty substrings of the given string s are promising? Each non-empty promising substring must be counted in the answer as many times as it occurs in string s.

Recall that a substring is a sequence of consecutive characters of the string. For example, for string "+-+" its substring are: "+-", "-+", "+", "+-+" (the string is a substring of itself) and some others. But the following strings are not its substring: "--", "++", "-++".

Input

The first line of the input contains an integer t $(1 \le t \le 500)$ —the number of test cases in the test.

Then the descriptions of test cases follow.

Each test case of input data consists of two lines. The first line consists of the number n $(1 \le n \le 3000)$: the length of s.

The second line of the test case contains the string s of length n, consisting only of characters "+" and "-".

It is guaranteed that the sum of values n over all test cases does not exceed 3000.

Output

For each test case, print a single number: the number of the promising non-empty substrings of string s. Each non-empty promising substring must be counted in the answer as many times as it occurs in string s.

Example

input

```
5
3
+-+
5
-+---
4
----
7
--+---+
6
+++---
```

output

```
2
4
2
7
4
```

Note

The following are the promising substrings for the first three test cases in the example:

1. $s[1\dots2]$="+-", $s[2\dots3]$="-+";
2. $s[1\dots2]$="-+", $s[2\dots3]$="+-", $s[1\dots5]$="-+---", $s[3\dots5]$="---";
3. $s[1\dots3]$="---", $s[2\dots4]$="---".

F2. Promising String (hard version)

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

This is the hard version of Problem F. The only difference between the easy version and the hard version is the constraints.

We will call a non-empty string **balanced** if it contains the same number of plus and minus signs. For example: strings "+--+" and "++-+--" are balanced, and strings "+--", "--" and "" are not balanced.

We will call a string **promising** if the string can be made balanced by several (possibly zero) uses of the following operation:

- replace two **adjacent** minus signs with one plus sign.

In particular, every balanced string is promising. However, the converse is not true: not every promising string is balanced.

For example, the string "-+---" is promising, because you can replace two adjacent minuses with plus and get a balanced string "-++-", or get another balanced string "-+-+".

How many non-empty substrings of the given string s are promising? Each non-empty promising substring must be counted in the answer as many times as it occurs in string s.

Recall that a substring is a sequence of consecutive characters of the string. For example, for string "+-+" its substring are: "+-", "-+", "+", "+-+" (the string is a substring of itself) and some others. But the following strings are not its substring: "--", "++", "-++".

Input

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

Then the descriptions of test cases follow.

Each test case of input data consists of two lines. The first line consists of the number n ($1 \le n \le 2 \cdot 10^5$): the length of s.

The second line of the test case contains the string s of length n, consisting only of characters "+" and "-".

It is guaranteed that the sum of values n over all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, print a single number: the number of the promising non-empty substrings of string s. Each non-empty promising substring must be counted in the answer as many times as it occurs in string s.

Example

input

```
5
3
+-+
5
-+---
4
----
7
--+---+
6
+++---
```

output

```
2
4
2
7
4
```

Note

The following are the promising substrings for the first three test cases in the example:

1. $s[1 \dots 2]$="+-", $s[2 \dots 3]$="-+";
2. $s[1 \dots 2]$="-+", $s[2 \dots 3]$="+-", $s[1 \dots 5]$="-+---", $s[3 \dots 5]$="---";
3. $s[1 \dots 3]$="---", $s[2 \dots 4]$="---".