

Codeforces Round #616 (Div. 2)

A. Even But Not Even

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

Let's define a number *ebne* (*even but not even*) if and only if its sum of digits is divisible by 2 but the number itself is not divisible by 2. For example, 13, 1227, 185217 are *ebne* numbers, while 12, 2, 177013, 265918 are not. If you're still unsure what *ebne* numbers are, you can look at the sample notes for more clarification.

You are given a non-negative integer s , consisting of n digits. You can delete some digits (they are **not** necessary consecutive/successive) to make the given number *ebne*. You cannot change the order of the digits, that is, after deleting the digits the remaining digits collapse. The resulting number shouldn't contain leading zeros. You can delete any number of digits between 0 (do not delete any digits at all) and $n - 1$.

For example, if you are given $s = 222373204424185217171912$ then one of possible ways to make it *ebne* is: ~~222373204424~~185217171912 \rightarrow 2237344218521717191. The sum of digits of 2237344218521717191 is equal to 70 and is divisible by 2, but number itself is not divisible by 2: it means that the resulting number is *ebne*.

Find **any** resulting number that is *ebne*. If it's impossible to create an *ebne* number from the given number report about it.

Input

The input consists of multiple test cases. The first line contains a single integer t ($1 \leq t \leq 1000$) — the number of test cases. The description of the test cases follows.

The first line of each test case contains a single integer n ($1 \leq n \leq 3000$) — the number of digits in the original number.

The second line of each test case contains a non-negative integer number s , consisting of n digits.

It is guaranteed that s does not contain leading zeros and the sum of n over all test cases does not exceed 3000.

Output

For each test case given in the input print the answer in the following format:

- If it is impossible to create an *ebne* number, print "-1" (without quotes);
- Otherwise, print the resulting number after deleting some, possibly zero, but not all digits. This number should be *ebne*. If there are multiple answers, you can print **any** of them. Note that answers with leading zeros or empty strings are not accepted. **It's not necessary to minimize or maximize the number of deleted digits.**

Example

input
4 4 1227 1 0 6 177013 24 222373204424185217171912
output
1227 -1 17703 2237344218521717191

Note

In the first test case of the example, 1227 is already an *ebne* number (as $1 + 2 + 2 + 7 = 12$, 12 is divisible by 2, while in the same time, 1227 is not divisible by 2) so we don't need to delete any digits. Answers such as 127 and 17 will also be accepted.

In the second test case of the example, it is clearly impossible to create an *ebne* number from the given number.

In the third test case of the example, there are many *ebne* numbers we can obtain by deleting, for example, 1 digit such as 17703, 77013 or 17013. Answers such as 1701 or 770 will not be accepted as they are not *ebne* numbers. Answer 013 will not be accepted as it contains leading zeroes.

Explanation:

- $1 + 7 + 7 + 0 + 3 = 18$. As 18 is divisible by 2 while 17703 is not divisible by 2, we can see that 17703 is an *ebne* number.

- Same with 77013 and 17013;
- $1 + 7 + 0 + 1 = 9$. Because 9 is not divisible by 2, 1701 is not an *ebne* number;
- $7 + 7 + 0 = 14$. This time, 14 is divisible by 2 but 770 is also divisible by 2, therefore, 770 is not an *ebne* number.

In the last test case of the example, one of many other possible answers is given. Another possible answer is: 222373204424185217171912 \rightarrow 22237320442418521717191 (delete the last digit).

B. Array Sharpening

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

You're given an array a_1, \dots, a_n of n non-negative integers.

Let's call it *sharpened* if and only if there exists an integer $1 \leq k \leq n$ such that $a_1 < a_2 < \dots < a_k$ and $a_k > a_{k+1} > \dots > a_n$. In particular, any strictly increasing or strictly decreasing array is *sharpened*. For example:

- The arrays $[4]$, $[0, 1]$, $[12, 10, 8]$ and $[3, 11, 15, 9, 7, 4]$ are sharpened;
- The arrays $[2, 8, 2, 8, 6, 5]$, $[0, 1, 1, 0]$ and $[2, 5, 6, 9, 8, 8]$ are **not** sharpened.

You can do the following operation as many times as you want: choose any **strictly positive** element of the array, and decrease it by one. Formally, you can choose any i ($1 \leq i \leq n$) such that $a_i > 0$ and assign $a_i := a_i - 1$.

Tell if it's possible to make the given array *sharpened* using some number (possibly zero) of these operations.

Input

The input consists of multiple test cases. The first line contains a single integer t ($1 \leq t \leq 15\,000$) — the number of test cases. The description of the test cases follows.

The first line of each test case contains a single integer n ($1 \leq n \leq 3 \cdot 10^5$).

The second line of each test case contains a sequence of n non-negative integers a_1, \dots, a_n ($0 \leq a_i \leq 10^9$).

It is guaranteed that the sum of n over all test cases does not exceed $3 \cdot 10^5$.

Output

For each test case, output a single line containing "Yes" (without quotes) if it's possible to make the given array sharpened using the described operations, or "No" (without quotes) otherwise.

Example

input
10 1 248618 3 12 10 8 6 100 11 15 9 7 8 4 0 1 1 0 2 0 0 2 0 1 2 1 0 2 1 1 3 0 1 0 3 1 0 1
output
Yes Yes Yes No No Yes Yes Yes Yes Yes No

Note

In the first and the second test case of the first test, the given array is already sharpened.

In the third test case of the first test, we can transform the array into $[3, 11, 15, 9, 7, 4]$ (decrease the first element 97 times and decrease the last element 4 times). It is sharpened because $3 < 11 < 15$ and $15 > 9 > 7 > 4$.

In the fourth test case of the first test, it's impossible to make the given array sharpened.

C. Mind Control

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

You and your $n - 1$ friends have found an array of integers a_1, a_2, \dots, a_n . You have decided to share it in the following way: All n of you stand in a line in a particular order. Each minute, the person at the front of the line chooses either the first or the last element of the array, removes it, and keeps it for himself. He then gets out of line, and the next person in line continues the process.

You are standing in the m -th position in the line. **Before the process starts**, you may choose up to k different people in the line, and persuade them to always take either the first or the last element in the array on their turn (for each person his own choice, not necessarily equal for all people), no matter what the elements themselves are. **Once the process starts, you cannot persuade any more people, and you cannot change the choices for the people you already persuaded.**

Suppose that you're doing your choices optimally. What is the greatest integer x such that, no matter what are the choices of the friends you didn't choose to control, the element you will take from the array will be greater than or equal to x ?

Please note that the friends you don't control may do their choice **arbitrarily**, and they will not necessarily take the biggest element available.

Input

The input consists of multiple test cases. The first line contains a single integer t ($1 \leq t \leq 1000$) — the number of test cases. The description of the test cases follows.

The first line of each test case contains three space-separated integers n, m and k ($1 \leq m \leq n \leq 3500, 0 \leq k \leq n - 1$) — the number of elements in the array, your position in line and the number of people whose choices you can fix.

The second line of each test case contains n positive integers a_1, a_2, \dots, a_n ($1 \leq a_i \leq 10^9$) — elements of the array.

It is guaranteed that the sum of n over all test cases does not exceed 3500.

Output

For each test case, print the largest integer x such that you can guarantee to obtain at least x .

Example

input
4 6 4 2 2 9 2 3 8 5 4 4 1 2 13 60 4 4 1 3 1 2 2 1 2 2 0 1 2
output
8 4 1 1

Note

In the first test case, an optimal strategy is to force the first person to take the last element and the second person to take the first element.

- the first person will take the last element (5) because he or she was forced by you to take the last element. After this turn the remaining array will be $[2, 9, 2, 3, 8]$;
- the second person will take the first element (2) because he or she was forced by you to take the first element. After this turn the remaining array will be $[9, 2, 3, 8]$;
- if the third person will choose to take the first element (9), at your turn the remaining array will be $[2, 3, 8]$ and you will take 8 (the last element);
- if the third person will choose to take the last element (8), at your turn the remaining array will be $[9, 2, 3]$ and you will take 9 (the first element).

Thus, this strategy guarantees to end up with at least 8. We can prove that there is no strategy that guarantees to end up with at least 9. Hence, the answer is 8.

In the second test case, an optimal strategy is to force the first person to take the first element. Then, in the worst case, both the second and the third person will take the first element: you will end up with 4.

D. Irreducible Anagrams

time limit per test: 2 seconds

memory limit per test: 256 megabytes
input: standard input
output: standard output

Let's call two strings s and t *anagrams* of each other if it is possible to rearrange symbols in the string s to get a string, equal to t .

Let's consider two strings s and t **which are anagrams of each other**. We say that t is a *reducible anagram* of s if there exists an integer $k \geq 2$ and $2k$ non-empty strings $s_1, t_1, s_2, t_2, \dots, s_k, t_k$ that satisfy the following conditions:

1. If we write the strings s_1, s_2, \dots, s_k in order, the resulting string will be equal to s ;
2. If we write the strings t_1, t_2, \dots, t_k in order, the resulting string will be equal to t ;
3. For all integers i between 1 and k inclusive, s_i and t_i are anagrams of each other.

If such strings don't exist, then t is said to be an *irreducible anagram* of s . **Note that these notions are only defined when s and t are anagrams of each other.**

For example, consider the string $s = \text{"gamegame"}$. Then the string $t = \text{"megamage"}$ is a reducible anagram of s , we may choose for example $s_1 = \text{"game"}$, $s_2 = \text{"gam"}$, $s_3 = \text{"e"}$ and $t_1 = \text{"mega"}$, $t_2 = \text{"mag"}$, $t_3 = \text{"e"}$:

$$\begin{array}{l} s = \overbrace{\text{game}}^{s_1} | \overbrace{\text{gam}}^{s_2} | \overbrace{\text{e}}^{s_3} \\ t = \underbrace{\text{mega}}_{t_1} | \underbrace{\text{mag}}_{t_2} | \underbrace{\text{e}}_{t_3} \end{array}$$

On the other hand, we can prove that $t = \text{"memegaga"}$ is an irreducible anagram of s .

You will be given a string s and q queries, represented by two integers $1 \leq l \leq r \leq |s|$ (where $|s|$ is equal to the length of the string s). For each query, you should find if the substring of s formed by characters from the l -th to the r -th has at least one irreducible anagram.

Input

The first line contains a string s , consisting of lowercase English characters ($1 \leq |s| \leq 2 \cdot 10^5$).

The second line contains a single integer q ($1 \leq q \leq 10^5$) — the number of queries.

Each of the following q lines contain two integers l and r ($1 \leq l \leq r \leq |s|$), representing a query for the substring of s formed by characters from the l -th to the r -th.

Output

For each query, print a single line containing "Yes" (without quotes) if the corresponding substring has at least one irreducible anagram, and a single line containing "No" (without quotes) otherwise.

Examples

input
aaaaa 3 1 1 2 4 5 5
output
Yes No Yes

input
aabbbbbbc 6 1 2 2 4 2 2 1 9 5 7 3 5
output
No Yes Yes Yes No No

Note

In the first sample, in the first and third queries, the substring is "a", which has itself as an irreducible anagram since two or more non-empty strings cannot be put together to obtain "a". On the other hand, in the second query, the substring is "aaa", which has no

irreducible anagrams: its only anagram is itself, and we may choose $s_1 = \text{"a"}, s_2 = \text{"aa"}, t_1 = \text{"a"}, t_2 = \text{"aa"}$ to show that it is a reducible anagram.

In the second query of the second sample, the substring is "abb", which has, for example, "bba" as an irreducible anagram.

E. Prefix Enlightenment

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

There are n lamps on a line, numbered from 1 to n . Each one has an initial state off (0) or on (1).

You're given k subsets A_1, \dots, A_k of $\{1, 2, \dots, n\}$, such that the intersection of any three subsets is empty. In other words, for all $1 \leq i_1 < i_2 < i_3 \leq k$, $A_{i_1} \cap A_{i_2} \cap A_{i_3} = \emptyset$.

In one operation, you can choose one of these k subsets and switch the state of all lamps in it. It is guaranteed that, with the given subsets, it's possible to make all lamps be simultaneously on using this type of operation.

Let m_i be the minimum number of operations you have to do in order to make the i first lamps be simultaneously on. Note that there is no condition upon the state of other lamps (between $i + 1$ and n), they can be either off or on.

You have to compute m_i for all $1 \leq i \leq n$.

Input

The first line contains two integers n and k ($1 \leq n, k \leq 3 \cdot 10^5$).

The second line contains a binary string of length n , representing the initial state of each lamp (the lamp i is off if $s_i = 0$, on if $s_i = 1$).

The description of each one of the k subsets follows, in the following format:

The first line of the description contains a single integer c ($1 \leq c \leq n$) — the number of elements in the subset.

The second line of the description contains c distinct integers x_1, \dots, x_c ($1 \leq x_i \leq n$) — the elements of the subset.

It is guaranteed that:

- The intersection of any three subsets is empty;
- It's possible to make all lamps be simultaneously on using some operations.

Output

You must output n lines. The i -th line should contain a single integer m_i — the minimum number of operations required to make the lamps 1 to i be simultaneously on.

Examples

input
7 3 0011100 3 1 4 6 3 3 4 7 2 2 3
output
1 2 3 3 3 3 3

input
8 6 00110011 3 1 3 8 5 1 2 5 6 7 2 6 8 2 3 5 2 4 7 1 2

output
1 1 1 1 1 1 1 4 4

input
5 3 00011 3 1 2 3 1 4 3 3 4 5

output
1 1 1 1 1

input
19 5 1001001001100000110 2 2 3 2 5 6 2 8 9 5 12 13 14 15 16 1 19

output
0 1 1 1 2 2 2 3 3 3 3 4 4 4 4 4 4 4 5

Note

In the first example:

- For $i = 1$, we can just apply one operation on A_1 , the final states will be 1010110;
- For $i = 2$, we can apply operations on A_1 and A_3 , the final states will be 1100110;
- For $i \geq 3$, we can apply operations on A_1 , A_2 and A_3 , the final states will be 1111111.

In the second example:

- For $i \leq 6$, we can just apply one operation on A_2 , the final states will be 11111101;
- For $i \geq 7$, we can apply operations on A_1, A_3, A_4, A_6 , the final states will be 11111111.

F. Coffee Varieties (easy version)

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

This is the easy version of the problem. You can find the hard version in the Div. 1 contest. Both versions only differ in the number of times you can ask your friend to taste coffee.

This is an interactive problem.

You're considering moving to another city, where one of your friends already lives. There are n cafés in this city, where n is a power of two. The i -th café produces a single variety of coffee a_i .

As you're a coffee-lover, before deciding to move or not, **you want to know the number d of distinct varieties of coffees** produced in this city.

You don't know the values a_1, \dots, a_n . Fortunately, your friend has a memory of size k , where k is a power of two.

Once per day, you can ask him to taste a cup of coffee produced by the café c , and he will tell you if he tasted a similar coffee during the last k days.

You can also ask him to take a medication that will reset his memory. He will forget all previous cups of coffee tasted. You can reset his memory at most 30 000 times.

More formally, the memory of your friend is a queue S . Doing a query on café c will:

- Tell you if a_c is in S ;
- Add a_c at the back of S ;
- If $|S| > k$, pop the front element of S .

Doing a reset request will pop all elements out of S .

Your friend can taste at most $\frac{2n^2}{k}$ cups of coffee in total. Find the diversity d (number of distinct values in the array a).

Note that asking your friend to reset his memory **does not count** towards the number of times you ask your friend to taste a cup of coffee.

In some test cases the behavior of the interactor **is adaptive**. It means that the array a may be **not fixed** before the start of the interaction and may **depend on your queries**. It is guaranteed that at any moment of the interaction, there is at least one array a consistent with all the answers given so far.

Input

The first line contains two integers n and k ($1 \leq k \leq n \leq 1024$, k and n are powers of two).

It is guaranteed that $\frac{2n^2}{k} \leq 20\,000$.

Interaction

You begin the interaction by reading n and k .

- To ask your friend to taste a cup of coffee produced by the café c , in a separate line output
? c

Where c must satisfy $1 \leq c \leq n$. Don't forget to flush, to get the answer.

In response, you will receive a single letter Y (yes) or N (no), telling you if variety a_c is one of the last k varieties of coffee in his memory.

- To reset the memory of your friend, in a separate line output the single letter **R in upper case**. You can do this operation at most 30 000 times.
- When you determine the number d of different coffee varieties, output
! d

In case your query is invalid, you asked more than $\frac{2n^2}{k}$ queries of type ? or you asked more than 30 000 queries of type R, the program will print the letter E and will finish interaction. You will receive a **Wrong Answer** verdict. Make sure to exit immediately to avoid getting other verdicts.

After printing a query do not forget to output end of line and flush the output. Otherwise, you will get **Idleness limit exceeded**. To do this, use:

- `fflush(stdout)` or `cout.flush()` in C++;
- `System.out.flush()` in Java;
- `flush(output)` in Pascal;
- `stdout.flush()` in Python;
- see documentation for other languages.

Hack format

The first line should contain the word `fixed`

The second line should contain two integers n and k , separated by space ($1 \leq k \leq n \leq 1024$, k and n are powers of two).

It must hold that $\frac{2n^2}{k} \leq 20\,000$.

The third line should contain n integers a_1, a_2, \dots, a_n , separated by spaces ($1 \leq a_i \leq n$).

Examples

input
4 2 N N Y N N N N
output
? 1 ? 2 ? 3 ? 4 R ? 4 ? 1 ? 2 ! 3

input
8 8 N N N N Y Y
output
? 2 ? 6 ? 4 ? 5 ? 2 ? 5 ! 6

Note

In the first example, the array is $a = [1, 4, 1, 3]$. The city produces 3 different varieties of coffee (1, 3 and 4).

The successive varieties of coffee tasted by your friend are 1, 4, **1**, 3, 3, 1, 4 (bold answers correspond to Y answers). Note that between the two ? 4 asks, there is a reset memory request R, so the answer to the second ? 4 ask is N. Had there been no reset memory request, the answer to the second ? 4 ask is Y.

In the second example, the array is $a = [1, 2, 3, 4, 5, 6, 6, 6]$. The city produces 6 different varieties of coffee.

The successive varieties of coffee tasted by your friend are 2, 6, 4, 5, **2**, **5**.