A. Avoid Trygub

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

A string b is a subsequence of a string a if b can be obtained from a by deletion of several (possibly, zero or all) characters. For example, "xy" is a subsequence of "xzyw" and "xy", but not "yx".

You are given a string a. Your task is to reorder the characters of a so that "trygub" is not a subsequence of the resulting string.

In other words, you should find a string b which is a permutation of symbols of the string a and "trygub" is not a subsequence of b.

We have a truly marvelous proof that any string can be arranged not to contain "trygub" as a subsequence, but this problem statement is too short to contain it.

Input

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

The first line of each test case contains a single integer n $(1 \le n \le 200)$ — the length of a.

The next line contains the string a of length n, consisting of lowercase English letters.

Output

For each test case, output a string b of length n which is a permutation of characters of the string a, and such that "trygub" is not a subsequence of it.

If there exist multiple possible strings b, you can print any.

Example

input

```
3
11
antontrygub
15
bestcoordinator
19
trywatchinggurabruh
```

output

```
bugyrtnotna
bestcoordinator
bruhtrywatchinggura
```

Note

In the first test case, "bugyrtnotna" does not contain "trygub" as a subsequence. It does contain the letters of "trygub", but not in the correct order, so it is not a subsequence.

In the second test case, we did not change the order of characters because it is not needed.

In the third test case, "bruhtrywatchinggura" does contain "trygu" as a subsequence, but not "trygub".

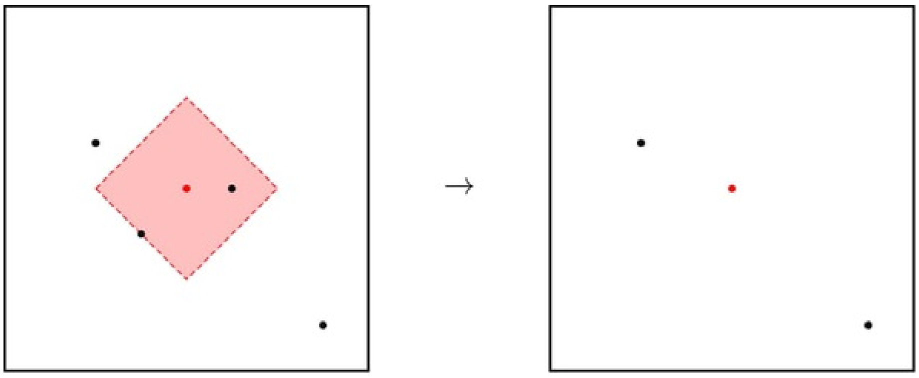
B. Balls of Steel

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

You have n **distinct** points $(x_1, y_1), \ldots, (x_n, y_n)$ on the plane and a non-negative integer parameter k. Each point is a microscopic steel ball and k is the attract power of a ball when it's charged. The attract power is the same for all balls.

In one operation, you can select a ball i to charge it. Once charged, **all** balls with Manhattan distance at most k from ball i move to the position of ball i. Many balls may have the same coordinate after an operation.

More formally, for all balls j such that $|x_i - x_j| + |y_i - y_j| \le k$, we assign $x_j := x_i$ and $y_j := y_i$.


An example of an operation. After charging the ball in the center, two other balls move to its position. On the right side, the red dot in the center is the common position of those balls.

Your task is to find the minimum number of operations to move all balls to the same position, or report that this is impossible.

Input

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

The first line of each test case contains two integers n, k ($2 \le n \le 100$, $0 \le k \le 10^6$) — the number of balls and the attract power of all balls, respectively.

The following n lines describe the balls' coordinates. The i-th of these lines contains two integers x_i, y_i ($0 \le x_i, y_i \le 10^5$) — the coordinates of the i-th ball.

It is guaranteed that all points are **distinct**.

Output

For each test case print a single integer — the minimum number of operations to move all balls to the same position, or -1 if it is impossible.

Example

input

```
3
3 2
0 0
3 3
1 1
3 3
6 7
8 8
6 9
4 1
0 0
0 1
0 2
0 3
```

output

```
-1
1
-1
```

Note

In the first test case, there are three balls at $(0,0)$, $(3,3)$, and $(1,1)$ and the attract power is 2. It is possible to move two balls together with one operation, but not all three balls together with any number of operations.

In the second test case, there are three balls at $(6,7)$, $(8,8)$, and $(6,9)$ and the attract power is 3. If we charge any ball, the other two will move to the same position, so we only require one operation.

In the third test case, there are four balls at $(0,0)$, $(0,1)$, $(0,2)$, and $(0,3)$, and the attract power is 1. We can show that it is impossible to move all balls to the same position with a sequence of operations.

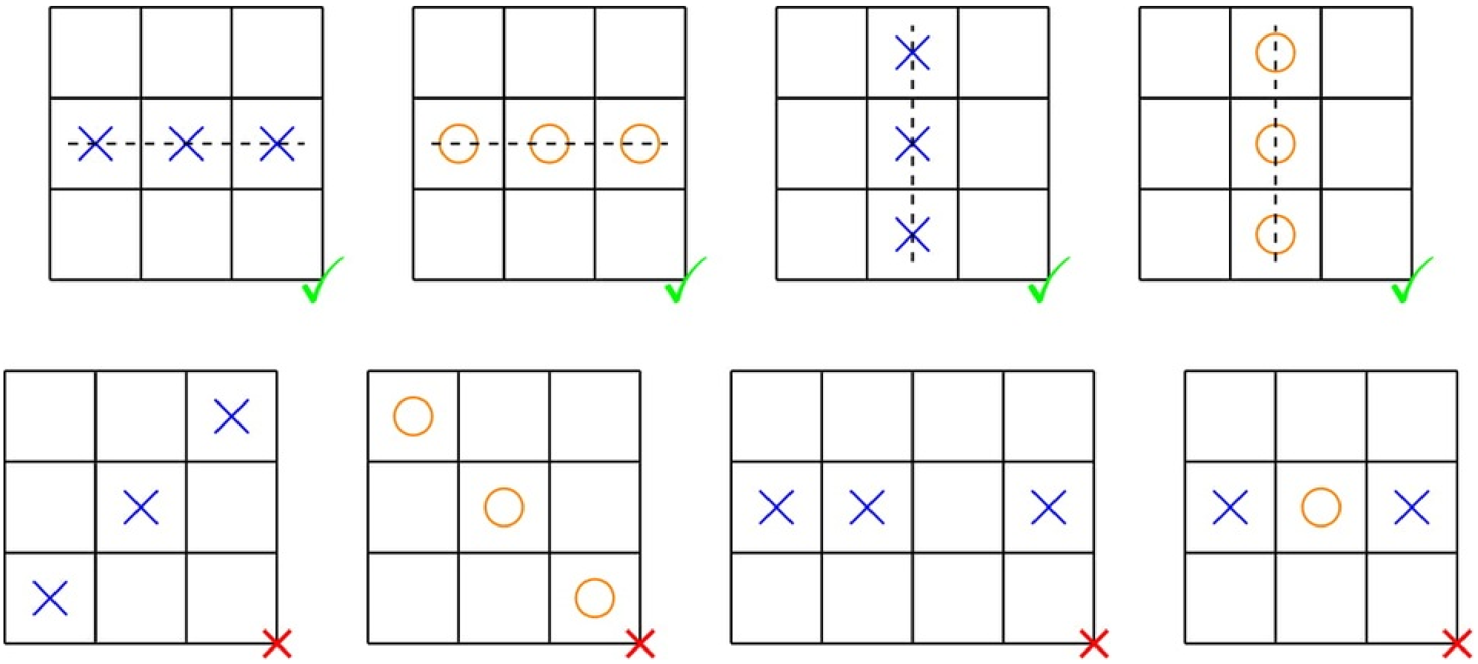
C1. Errich-Tac-Toe (Easy Version)

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

The only difference between the easy and hard versions is that tokens of type O do not appear in the input of the easy version.

Errichto gave Monogon the following challenge in order to intimidate him from taking his top contributor spot on Codeforces.

In a Tic-Tac-Toe grid, there are n rows and n columns. Each cell of the grid is either empty or contains a token. There are two types of tokens: X and O. If there exist three tokens of the same type consecutive in a row or column, it is a winning configuration. Otherwise, it is a draw configuration.


The patterns in the first row are winning configurations. The patterns in the second row are draw configurations.

In an operation, you can change an X to an O, or an O to an X. Let k denote the total number of tokens in the grid. Your task is to make the grid a **draw** in at most $\lfloor \frac{k}{3} \rfloor$ (rounding down) operations.

You are **not required** to minimize the number of operations.

Input

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

The first line of each test case contains a single integer n ($1 \le n \le 300$) — the size of the grid.

The following n lines each contain a string of n characters, denoting the initial grid. The character in the i-th row and j-th column is '.' if the cell is empty, or it is the type of token in the cell: 'X' or 'O'.

It is guaranteed that not all cells are empty.

In the easy version, the character 'O' does not appear in the input.

The sum of n across all test cases does not exceed 300.

Output

For each test case, print the state of the grid after applying the operations.

We have proof that a solution always exists. If there are multiple solutions, print any.

Example

input

```
3
3
.X.
XXX
.X.
6
XX.XXX
XXXXXX
XXX.XX
XXXXXX
XX.X.X
XXXXXX
5
XXX.X
.X..X
XXX.X
..X..
..X..
```

output

```
.X.
XOX
.X.
XX.XXO
XOXXOX
OXX.XX
XOOXXO
XX.X.X
OXXOXX
XOX.X
.X..X
XXO.O
..X..
```

```
..X..
```

Note

In the first test case, there are initially three 'X' consecutive in the second row and the second column. By changing the middle token to '0' we make the grid a draw, and we only changed $1 \le \lfloor 5/3 \rfloor$ token.

In the second test case, we change only $9 \le \lfloor 32/3 \rfloor$ tokens, and there does not exist any three 'X' or '0' consecutive in a row or column, so it is a draw.

In the third test case, we change only $3 \le \lfloor 12/3 \rfloor$ tokens, and the resulting grid is a draw.

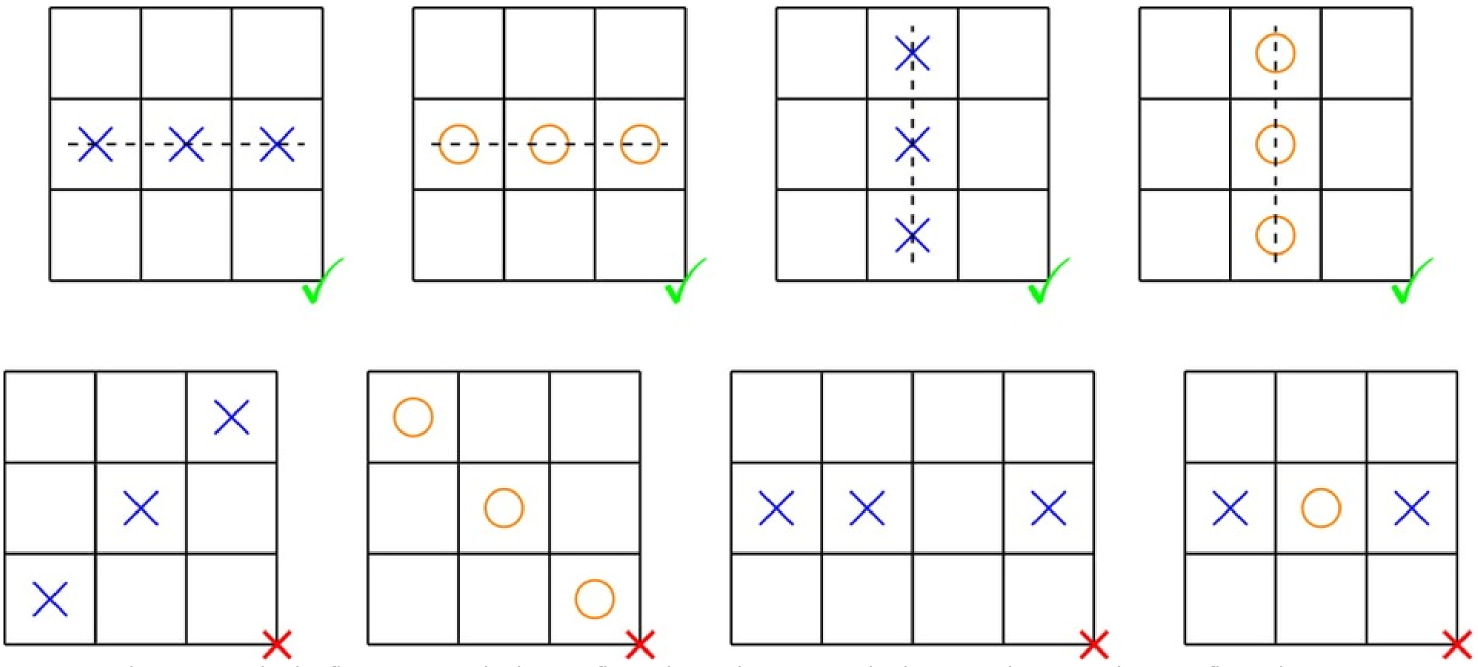
C2. Errich-Tac-Toe (Hard Version)

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

The only difference between the easy and hard versions is that tokens of type 0 do not appear in the input of the easy version.

Errichto gave Monogon the following challenge in order to intimidate him from taking his top contributor spot on Codeforces.

In a Tic-Tac-Toe grid, there are n rows and n columns. Each cell of the grid is either empty or contains a token. There are two types of tokens: X and 0. If there exist three tokens of the same type consecutive in a row or column, it is a winning configuration. Otherwise, it is a draw configuration.



The patterns in the first row are winning configurations. The patterns in the second row are draw configurations.

In an operation, you can change an X to an 0, or an 0 to an X. Let k denote the total number of tokens in the grid. Your task is to make the grid a **draw** in at most $\lfloor \frac{k}{3} \rfloor$ (rounding down) operations.

You are **not required** to minimize the number of operations.

Input

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

The first line of each test case contains a single integer n $(1 \le n \le 300)$ — the size of the grid.

The following n lines each contain a string of n characters, denoting the initial grid. The character in the i-th row and j-th column is '.' if the cell is empty, or it is the type of token in the cell: 'X' or '0'.

It is guaranteed that not all cells are empty.

The sum of n across all test cases does not exceed 300.

Output

For each test case, print the state of the grid after applying the operations.

We have proof that a solution always exists. If there are multiple solutions, print any.

Example

input

```
3
3
```

```
.O.
OOO
.O.
6
XXXOOO
XXXOOO
XX..OO
OO..XX
OOOXXX
OOOXXX
5
.OOO.
OXXXO
OXXXO
OXXXO
.OOO.
```

output

```
.O.
OXO
.O.
OXXOOX
XOXOXO
XX..OO
OO..XX
OXOXOX
XOOXXO
.OXO.
OOXXO
XXOXX
OXXOO
.OXO.
```

Note

In the first test case, there are initially three '0' consecutive in the second row and the second column. By changing the middle token to 'X' we make the grid a draw, and we only changed $1 \leq \lfloor 5/3 \rfloor$ token.

In the second test case, the final grid is a draw. We only changed $8 \leq \lfloor 32/3 \rfloor$ tokens.

In the third test case, the final grid is a draw. We only changed $7 \leq \lfloor 21/3 \rfloor$ tokens.

D. Rating Compression

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

On the competitive programming platform CodeCook, every person has a rating graph described by an array of integers a of length n. You are now updating the infrastructure, so you've created a program to compress these graphs.

The program works as follows. Given an integer parameter k, the program takes the minimum of each contiguous subarray of length k in a.

More formally, for an array a of length n and an integer k, define the k-compression array of a as an array b of length $n-k+1$, such that

$$b_j = \min_{j \le i \le j+k-1} a_i$$

For example, the 3-compression array of $[1,3,4,5,2]$ is $[\min\{1,3,4\}, \min\{3,4,5\}, \min\{4,5,2\}] = [1,3,2]$.

A permutation of length m is an array consisting of m distinct integers from 1 to m in arbitrary order. For example, $[2,3,1,5,4]$ is a permutation, but $[1,2,2]$ is not a permutation (2 appears twice in the array) and $[1,3,4]$ is also not a permutation ($m=3$ but there is 4 in the array).

A k-compression array will make CodeCook users happy if it will be a permutation. Given an array a, determine for all $1 \leq k \leq n$ if CodeCook users will be happy after a k-compression of this array or not.

Input

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

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

The second line of the description of each test case contains n integers $a_1, \ldots, a_n$ ($1 \leq a_i \leq n$) — the elements of the array.

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

Output

For each test case, print a binary string of length n.

The k-th character of the string should be 1 if CodeCook users will be happy after a k-compression of the array a, and 0 otherwise.

Example

input

```
5
5
1 5 3 4 2
4
1 3 2 1
5
1 3 3 3 2
10
1 2 3 4 5 6 7 8 9 10
3
3 3 2
```

output

```
10111
0001
00111
1111111111
000
```

Note

In the first test case, $a = [1, 5, 3, 4, 2]$.

- The 1-compression of a is $[1, 5, 3, 4, 2]$ and it is a permutation.
- The 2-compression of a is $[1, 3, 3, 2]$ and it is not a permutation, since 3 appears twice.
- The 3-compression of a is $[1, 3, 2]$ and it is a permutation.
- The 4-compression of a is $[1, 2]$ and it is a permutation.
- The 5-compression of a is $[1]$ and it is a permutation.

E. Capitalism

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

A society can be represented by a connected, undirected graph of n vertices and m edges. The vertices represent people, and an edge (i, j) represents a friendship between people i and j.

In society, the i-th person has an income a_i. A person i is envious of person j if $a_j = a_i + 1$. That is if person j has exactly 1 more unit of income than person i.

The society is called **capitalist** if for every pair of friends one is envious of the other. For some friendships, you know which friend is envious of the other. For the remaining friendships, you do not know the direction of envy.

The **income inequality** of society is defined as $\max\limits_{1 \le i \le n} a_i - \min\limits_{1 \le i \le n} a_i$.

You only know the friendships and not the incomes. If it is impossible for this society to be capitalist with the given knowledge, you should report about it. Otherwise, you should find an assignment of incomes in which the society is capitalist, and the income inequality is maximized.

Input

The first line contains two integers n, m ($1 \le n \le 200$, $n - 1 \le m \le 2000$) — the number of people and friendships, respectively.

The following m lines describe the friendships. Each friendship is described by three integers i, j, b ($1 \le i, j \le n, i \ne j, 0 \le b \le 1$). This denotes that people i and j are friends. If $b = 1$, we require that person i is envious of person j. If $b = 0$, one friend should be envious of the other in either direction.

There is at most one friendship between each pair of people. It is guaranteed that if we consider the friendships as undirected edges, the graph is connected.

Output

Print "YES" if it is possible that the society is capitalist, or "NO" otherwise. You can print characters in any case (upper or lower).

If the answer is "YES", you should print two additional lines. In the first line, print the maximum possible income inequality. On the next line you should print n integers $a_1, \ldots, a_n$ ($0 \le a_i \le 10^6$), where a_i denotes the income of the i-th person.

We can prove that if there exists a solution, there exists one where $0 \le a_i \le 10^6$ for all i.

If there exist multiple solutions, print any.

Examples

input

```
6 6
1 2 0
3 2 0
2 5 0
```

```
6 5 1
6 3 0
2 4 1
```

output

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

input

```
4 4
1 2 1
2 3 0
3 4 1
4 1 1
```

output

```
NO
```

input

```
1 0
```

output

```
YES
0
0
```

Note

In the first test, we can show that an income inequality greater than 3 is impossible for the given society. In the given answer with income inequality equal to 3:

- Person 2 is envious of person 1.
- Person 3 is envious of person 2.
- Person 5 is envious of person 2.
- Person 6 is envious of person 5 (the required direction is satisfied).
- Person 6 is envious of person 3.
- Person 2 is envious of person 4 (the required direction is satisfied).

In the second test, we can show that there is no way to assign incomes to satisfy all requirements.

F. The Struggling Contestant

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

To help those contestants who struggle a lot in contests, the headquarters of Codeforces are planning to introduce Division 5. In this new division, the tags of all problems will be announced prior to the round to help the contestants.

The contest consists of n problems, where the tag of the i-th problem is denoted by an integer a_i.

You want to AK (solve all problems). To do that, you must solve the problems in some order. To make the contest funnier, you created extra limitations on yourself. You do not want to solve two problems consecutively with the same tag since it is boring. Also, you are afraid of big jumps in difficulties while solving them, so you want to minimize the number of times that you solve two problems consecutively that are not adjacent in the contest order.

Formally, your solve order can be described by a permutation p of length n. The **cost** of a permutation is defined as the number of indices i ($1 \le i < n$) where $|p_{i+1} - p_i| > 1$. You have the requirement that $a_{p_i} \ne a_{p_{i+1}}$ for all $1 \le i < n$.

You want to know the minimum possible cost of permutation that satisfies the requirement. If no permutations meet this requirement, you should report about it.

Input

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

The first line of the description of each test case contains a single integer n ($1 \le n \le 10^5$) — the number of problems in the contest.

The next line contains n integers $a_1, a_2, \dots a_n$ ($1 \le a_i \le n$) — the tags of the problems.

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

Output

For each test case, if there are no permutations that satisfy the required condition, print -1. Otherwise, print the minimum possible cost of a permutation that satisfies the required condition.

Example

input

```
4
6
2 1 2 3 1 1
5
1 1 1 2 2
8
7 7 2 7 7 1 8 7
10
1 2 3 4 1 1 2 3 4 1
```

output

```
1
3
-1
2
```

Note

In the first test case, let $p = [5, 4, 3, 2, 1, 6]$. The cost is 1 because we jump from $p_5 = 1$ to $p_6 = 6$, and $|6 - 1| > 1$. This permutation is valid because we don't solve problems with the same tag twice in a row. We cannot find a permutation with a cost smaller than 1.

In the second test case, let $p = [1, 5, 2, 4, 3]$. The cost is 3 because $|p_2 - p_1| > 1$, $|p_3 - p_2| > 1$, and $|p_4 - p_3| > 1$. The permutation is valid because we don't solve problems with the same tag twice in a row. We cannot find a permutation with a cost smaller than 3.

In the third test case, for any order of solving the problems, we will solve two problems with the same tag consecutively, so the answer is -1.

G. Communism

time limit per test: 1.5 seconds
memory limit per test: 32 megabytes
input: standard input
output: standard output

Please pay attention to the unusual memory limit in this problem.

In a parallel universe, Satan is called "Trygub". For that reason, the letters of his namesake were deleted from the alphabet in ancient times.

The government has n workers standing in a row and numbered with integers from 1 to n from left to right. Their job categories can be represented as a string s of length n, where the character s_i represents the job category of the i-th worker.

A new law will be approved to increase the equality between the workers. The government decided to make everyone have the same job category by performing the following operation any number of times (possibly zero).

There is a fixed **rational** parameter $k = \frac{a}{b}$ describing how easy it is to convince the public, and it will be used to determine the success of an operation.

In an operation, the government first selects a job category x with at least one worker at the current moment. Suppose $i_1, \ldots, i_m$ ($i_1 < \ldots < i_m$) are the positions of all the workers with job category x. If $k \cdot (i_m - i_1 + 1) \le m$, the government is able to choose any job category y with at least one worker at the current moment and change the job category of **all** workers with job category x to job category y.

If it is possible to make all workers have job category x, we say that x is obtainable. Can you tell the government the set of obtainable job categories?

Input

The first line contains three integers n, a, b ($1 \le n \le 5000$, $1 \le a \le b \le 10^5$) — the number of workers and the numerator and denominator of the parameter k, respectively.

The second line contains a string s of length n, consisting of lowercase English characters — the job categories of each worker. **The characters 't', 'r', 'y', 'g', 'u', and 'b' do not appear in the string s.**

Output

Print an integer c equal to the number of obtainable job categories followed by c space-separated characters — the obtainable job categories sorted in the lexicographical order.

Example

input

```
7 1 2
comicom
```

output

```
3 c m o
```

Note

The first operation must select the job category 'i' because all other job categories cannot satisfy the condition, therefore 'i' is not obtainable.

Below is showed how to obtain 'c', 'm', and 'o'. The square brackets denote the segment containing all workers of the selected category, the red color denotes this category and the blue color denotes the new category after the change.

- Get 'c':
  1. $(\texttt{com[i]com} \to \texttt{com[o]com})$
  2. $(\texttt{c[omoco]m} \to \texttt{c[mmmcm]m})$
  3. $(\texttt{c[mmmcmm]} \to \texttt{c[cccccc]})$
- Get 'm':
  1. $(\texttt{com[i]com} \to \texttt{com[o]com})$
  2. $(\texttt{c[omoco]m} \to \texttt{c[cmccc]m})$
  3. $(\texttt{[ccmccc]m} \to \texttt{[mmmmmm]m})$
- Get 'o':
  1. $(\texttt{com[i]com} \to \texttt{com[c]com})$
  2. $(\texttt{[comcc]om} \to \texttt{[mommm]om})$
  3. $(\texttt{[mommmom]} \to \texttt{[ooooooo]})$

H1. Multithreading (Easy Version)

time limit per test: 4 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

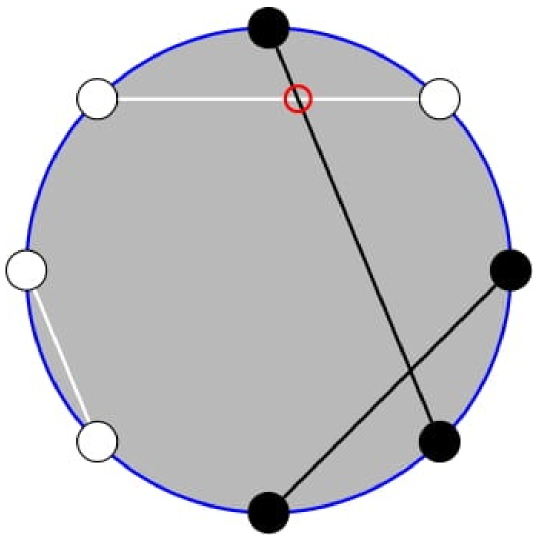
The only difference between the two versions of the problem is that there are no updates in the easy version.

There are n spools of thread placed on the rim of a circular table. The spools come in two types of thread: the first thread is black and the second thread is white.

For any two spools of the same color, you can attach them with a thread of that color in a straight line segment. Define a matching as a way to attach spools together so that each spool is attached to exactly one other spool.

Coloring is an assignment of colors (white and black) to the spools. A coloring is called **valid** if it has at least one matching. That is if the number of black spools and the number of white spools are both even.

Given a matching, we can find the number of times some white thread intersects some black thread. We compute the number of pairs of differently colored threads that intersect instead of the number of intersection points, so one intersection point may be counted multiple times if different pairs of threads intersect at the same point. If c is a valid coloring, let $f(c)$ denote the minimum number of such intersections out of all possible matchings.



The circle above is described by the coloring bwbbbwww. After matching the spools as shown, there is one intersection between differently colored threads. It can be proven that it is the minimum possible, so $f(\text{bwbbbwww}) = 1$.

You are given a string s representing an **unfinished** coloring, with black, white, and uncolored spools. A coloring c is called s-reachable if you can achieve it by assigning colors to the uncolored spools of s without changing the others.

A coloring c is chosen uniformly at random among all valid, s-reachable colorings. Compute the expected value of $f(c)$. You should find it by modulo 998244353.

We can show that the answer can be written in the form $\frac{p}{q}$ where p and q are relatively prime integers and $q \not\equiv 0 \pmod{998244353}$. The answer by modulo 998244353 is equal to $(p \cdot q^{-1})$ modulo 998244353.

Input

The first line contains two integers n, m ($2 \le n \le 2 \cdot 10^5$, n is even, $m = 0$) — the number of spools and updates, respectively. There are no updates in the easy version, so it is always 0.

The second line contains a string s of length n — the unfinished coloring of the spools. The i-th character will be 'w', 'b', or '?', describing if the i-th spool is white, black, or uncolored, respectively.

It is guaranteed there exists at least one uncolored spool.

Output

Print the expected value of $f(c)$ by modulo 998244353.

Examples

input

```
8 0
bwbb?www
```

output

```
1
```

input

```
10 0
???ww?wb??
```

output

```
436731905
```

input

```
4 0
bw?b
```

output

```
0
```

Note

The first test corresponds closely to the image. Coloring '?' as 'w' does not create a valid coloring because the number of black spools is odd. Then the only reachable valid coloring is 'bwbbbwww' and $f(\text{bwbbbwww}) = 1$, so the expected value is 1.

It can be shown that the expected value for the second test is $\frac{9}{16}$.

H2. Multithreading (Hard Version)

time limit per test: 4 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

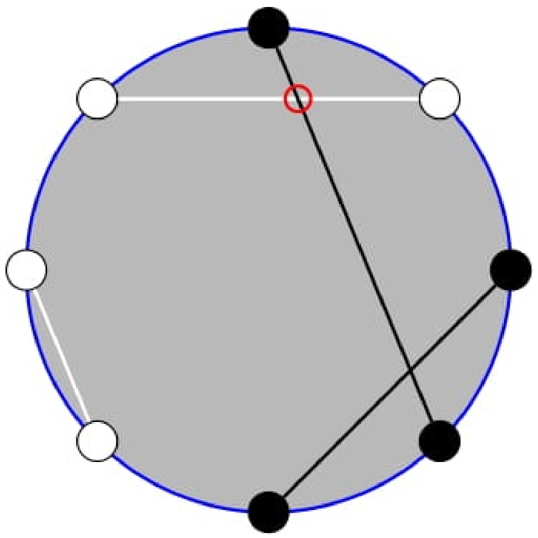
The only difference between the two versions of the problem is that there are no updates in the easy version.

There are n spools of thread placed on the rim of a circular table. The spools come in two types of thread: the first thread is black and the second thread is white.

For any two spools of the same color, you can attach them with a thread of that color in a straight line segment. Define a matching as a way to attach spools together so that each spool is attached to exactly one other spool.

Coloring is an assignment of colors (white and black) to the spools. A coloring is called **valid** if it has at least one matching. That is if the number of black spools and the number of white spools are both even.

Given a matching, we can find the number of times some white thread intersects some black thread. We compute the number of pairs of differently colored threads that intersect instead of the number of intersection points, so one intersection point may be counted multiple times if different pairs of threads intersect at the same point. If c is a valid coloring, let $f(c)$ denote the minimum number of such intersections out of all possible matchings.



The circle above is described by the coloring bwbbbwww. After matching the spools as shown, there is one intersection between differently colored threads. It can be proven that it is the minimum possible, so $f(\text{bwbbbwww}) = 1$.

You are given a string s representing an **unfinished** coloring, with black, white, and uncolored spools. A coloring c is called s-reachable if you can achieve it by assigning colors to the uncolored spools of s without changing the others.

A coloring c is chosen uniformly at random among all valid, s-reachable colorings. Compute the expected value of $f(c)$. You should find it by modulo 998244353.

There will be m updates to change one character of s. After each update, you should again compute the expected value of $f(c)$.

We can show that each answer can be written in the form $\frac{p}{q}$ where p and q are relatively prime integers and $q \not\equiv 0 \pmod{998244353}$. The answer by modulo 998244353 is equal to $(p \cdot q^{-1})$ modulo 998244353.

Input

The first line contains two integers n, m ($2 \le n \le 2 \cdot 10^5$, n is even, $0 \le m \le 2 \cdot 10^5$) — the number of spools and the number of updates, respectively.

The second line contains a string s of length n — the unfinished coloring of the spools. The i-th character will be 'w', 'b', or '?', describing if the i-th spool is white, black, or uncolored, respectively.

Each of the next m lines contains an integer i ($1 \le i \le n$) — the position of the character in s to be updated, and a character c ($c \in \{\text{w}, \text{b}, ?\}$) — the new color of the spool i after the update.

It is guaranteed there exists at least one uncolored spool initially and after each update.

Output

Print $m+1$ lines: the expected value of $f(c)$ initially and after each update. All values should be found by modulo 998244353.

Examples

input

```
8 0
bwbb?www
```

output

```
1
```

input

```
10 3
???ww?wb??
4 ?
5 ?
2 w
```

output

```
436731905
218365953
374341633
530317313
```

input

```
4 3
bw?b
1 w
2 b
1 w
```

output

```
0
0
```

```
1
1
```

Note

The first test corresponds closely to the image. Coloring '?' as 'w' does not create a valid coloring because the number of black spools is odd. Then the only reachable valid coloring is 'bwbbbwww' and $f(\text{bwbbbwww}) = 1$, so the expected value is 1.

In the second test, the string after each update is:

1. `????w?wb??`
2. `??????wb??`
3. `?w????wb??`

In the third test, the string after each update is:

1. `ww?b`
2. `wb?b`
3. `wb?b`