A. Math Problem

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

Your math teacher gave you the following problem:

There are n segments on the x-axis, $[l_1; r_1], [l_2; r_2], \ldots, [l_n; r_n]$. The segment $[l; r]$ includes the bounds, i.e. it is a set of such x that $l \le x \le r$. The length of the segment $[l; r]$ is equal to $r - l$.

Two segments $[a; b]$ and $[c; d]$ have a common point (intersect) if there exists x that $a \leq x \leq b$ and $c \leq x \leq d$. For example, $[2; 5]$ and $[3; 10]$ have a common point, but $[5; 6]$ and $[1; 4]$ don't have.

You should add one segment, which has at least one common point with each of the given segments and as short as possible (i.e. has minimal length). The required segment can degenerate to be a point (i.e a segment with length zero). The added segment may or may not be among the given n segments.

In other words, you need to find a segment $[a; b]$, such that $[a; b]$ and every $[l_i; r_i]$ have a common point for each i, and $b-a$ is minimal.

Input

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

The first line of each test case contains one integer n ($1 \le n \le 10^5$) — the number of segments. The following n lines contain segment descriptions: the i-th of them contains two integers l_i,r_i ($1 \le l_i \le r_i \le 10^9$).

The sum of all values n over all the test cases in the input doesn't exceed 10^5.

Output

For each test case, output one integer — the smallest possible length of the segment which has at least one common point with all given segments.

Example

input

```
4
3
4 5
5 9
7 7
5
11 19
4 17
16 16
3 12
14 17
1
1 10
1
1 1
```

output

```
2
4
0
0
```

Note

In the first test case of the example, we can choose the segment $[5; 7]$ as the answer. It is the shortest segment that has at least one common point with all given segments.

B. Box

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

Permutation p is a sequence of integers $p = [p_1, p_2, \ldots, p_n]$, consisting of n distinct (unique) positive integers between 1 and n, inclusive. For example, the following sequences are permutations: $[3, 4, 1, 2]$, $[1]$, $[1, 2]$. The following sequences are not permutations: $[0]$, $[1, 2, 1]$, $[2, 3]$, $[0, 1, 2]$.

The important key is in the locked box that you need to open. To open the box you need to enter secret code. Secret code is a permutation p of length n.

You don't know this permutation, you only know the array q of prefix maximums of this permutation. Formally:

- $q_1 = p_1$,
- $q_2 = \max(p_1, p_2)$,
- $q_3 = \max(p_1, p_2, p_3)$,
- ...
- $q_n = \max(p_1, p_2, \ldots, p_n)$.

You want to construct any possible suitable permutation (i.e. any such permutation, that calculated q for this permutation is equal to the given array).

Input

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

The first line of a test case contains one integer n $(1 \le n \le 10^5)$ — the number of elements in the secret code permutation p.

The second line of a test case contains n integers $q_1, q_2, \ldots, q_n$ $(1 \le q_i \le n)$ — elements of the array q for secret permutation. It is guaranteed that $q_i \le q_{i+1}$ for all i ($1 \le i < n$).

The sum of all values n over all the test cases in the input doesn't exceed 10^5.

Output

For each test case, print:

- If it's impossible to find such a permutation p, print "-1" (without quotes).
- Otherwise, print n distinct integers $p_1, p_2, \ldots, p_n$ ($1 \le p_i \le n$). If there are multiple possible answers, you can print any of them.

Example

input

```
4
5
1 3 4 5 5
4
1 1 3 4
2
2 2
1
1
```

output

```
1 3 4 5 2
-1
2 1
1
```

Note

In the first test case of the example answer $[1, 3, 4, 5, 2]$ is the only possible answer:

- $q_1 = p_1 = 1$;
- $q_2 = \max(p_1, p_2) = 3$;
- $q_3 = \max(p_1, p_2, p_3) = 4$;
- $q_4 = \max(p_1, p_2, p_3, p_4) = 5$;
- $q_5 = \max(p_1, p_2, p_3, p_4, p_5) = 5$.

It can be proved that there are no answers for the second test case of the example.

C. Messy

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

You are fed up with your messy room, so you decided to clean it up.

Your room is a bracket sequence $s = s_1 s_2 \ldots s_n$ of length n. Each character of this string is either an opening bracket '(' or a closing bracket ')'.

In one operation you can choose any consecutive substring of s and reverse it. In other words, you can choose any substring $s[l \ldots r] = s_l, s_{l+1}, \ldots, s_r$ and change the order of elements in it into $s_r, s_{r-1}, \ldots, s_l$.

For example, if you will decide to reverse substring $s[2 \dots 4]$ of string $s=$"((()))" it will be equal to $s=$"()(())".

A *regular* (aka balanced) bracket sequence is a bracket sequence that can be transformed into a correct arithmetic expression by inserting characters '1' and '+' between the original characters of the sequence. For example, bracket sequences "()()", "(())" are regular (the resulting expressions are: "(1)+(1)", "((1+1)+1)"), and ")(" and "(" are not.

A prefix of a string s is a substring that starts at position 1. For example, for $s=$"(())()" there are 6 prefixes: "(", "((", "(()", "(())", "(())(" and "(())()".

In your opinion, a neat and clean room s is a bracket sequence that:

- the whole string s is a *regular* bracket sequence;
- **and** there are exactly k prefixes of this sequence which are regular (including whole s itself).

For example, if $k=2$, then "(())()" is a neat and clean room.

You want to use at most n operations to make your room neat and clean. Operations are applied one after another sequentially.

It is guaranteed that the answer exists. Note that you **do not need** to minimize the number of operations: find any way to achieve the desired configuration in n or less operations.

Input

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

The first line of a test case contains two integers n and k ($1 \le k \le \frac{n}{2}, 2 \le n \le 2000$, n is even) — length of s and required number of regular prefixes.

The second line of a test case contains s of length n — the given bracket sequence. It contains only '(' and ')'.

It is guaranteed that there are exactly $\frac{n}{2}$ characters '(' and exactly $\frac{n}{2}$ characters ')' in the given string.

The sum of all values n over all the test cases in the input doesn't exceed 2000.

Output

For each test case print an answer.

In the first line print integer m ($0 \le m \le n$) — the number of operations. You **do not need** to minimize m, any value is suitable.

In the following m lines print description of the operations, each line should contain two integers l, r ($1 \le l \le r \le n$), representing single reverse operation of $s[l \dots r]=s_l s_{l+1} \dots s_r$. Operations are applied one after another sequentially.

The final s after all operations should be a regular, also it should be exactly k prefixes (including s) which are regular.

It is guaranteed that the answer exists. If there are several possible answers you can print any.

Example

input

```
4
8 2
()(())()
10 3
))()()()((
2 1
()
2 1
)(
```

output

```
4
3 4
1 1
5 8
2 2
3
4 10
1 4
6 7
0
1
1 2
```

Note

In the first example, the final sequence is "()(()())", where two prefixes are regular, "()" and "()(()())". Note, that all the operations except "5 8" in the example output are useless (they do not change s).

D1. Optimal Subsequences (Easy Version)

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

This is the easier version of the problem. In this version $1 \le n, m \le 100$. You can hack this problem only if you solve and lock both problems.

You are given a sequence of integers $a = [a_1, a_2, \dots, a_n]$ of length n. Its *subsequence* is obtained by removing zero or more elements from the sequence a (they do not necessarily go consecutively). For example, for the sequence $a = [11, 20, 11, 33, 11, 20, 11]$:

- $[11, 20, 11, 33, 11, 20, 11]$, $[11, 20, 11, 33, 11, 20]$, $[11, 11, 11, 11]$, $[20]$, $[33, 20]$ are subsequences (these are just some of the long list);
- $[40]$, $[33, 33]$, $[33, 20, 20]$, $[20, 20, 11, 11]$ are not subsequences.

Suppose that an additional non-negative integer k ($1 \le k \le n$) is given, then the subsequence is called *optimal* if:

- it has a length of k and the sum of its elements is the maximum possible among all subsequences of length k;
- and among all subsequences of length k that satisfy the previous item, it is *lexicographically* minimal.

Recall that the sequence $b = [b_1, b_2, \dots, b_k]$ is lexicographically smaller than the sequence $c = [c_1, c_2, \dots, c_k]$ if the first element (from the left) in which they differ less in the sequence b than in c. Formally: there exists t ($1 \le t \le k$) such that $b_1 = c_1$, $b_2 = c_2$, ..., $b_{t-1} = c_{t-1}$ and at the same time $b_t < c_t$. For example:

- $[10, 20, 20]$ lexicographically less than $[10, 21, 1]$,
- $[7, 99, 99]$ is lexicographically less than $[10, 21, 1]$,
- $[10, 21, 0]$ is lexicographically less than $[10, 21, 1]$.

You are given a sequence of $a = [a_1, a_2, \dots, a_n]$ and m requests, each consisting of two numbers k_j and pos_j ($1 \le k \le n$, $1 \le pos_j \le k_j$). For each query, print the value that is in the index pos_j of the optimal subsequence of the given sequence a for $k = k_j$.

For example, if $n = 4$, $a = [10, 20, 30, 20]$, $k_j = 2$, then the optimal subsequence is $[20, 30]$ — it is the minimum lexicographically among all subsequences of length 2 with the maximum total sum of items. Thus, the answer to the request $k_j = 2$, $pos_j = 1$ is the number 20, and the answer to the request $k_j = 2$, $pos_j = 2$ is the number 30.

Input

The first line contains an integer n ($1 \le n \le 100$) — the length of the sequence a.

The second line contains elements of the sequence a: integer numbers $a_1, a_2, \dots, a_n$ ($1 \le a_i \le 10^9$).

The third line contains an integer m ($1 \le m \le 100$) — the number of requests.

The following m lines contain pairs of integers k_j and pos_j ($1 \le k \le n$, $1 \le pos_j \le k_j$) — the requests.

Output

Print m integers $r_1, r_2, \dots, r_m$ ($1 \le r_j \le 10^9$) one per line: answers to the requests in the order they appear in the input. The value of r_j should be equal to the value contained in the position pos_j of the optimal subsequence for $k = k_j$.

Examples

input

```
3
10 20 10
6
1 1
2 1
2 2
3 1
3 2
3 3
```

output

```
20
10
20
10
20
10
```

input

```
7
1 2 1 3 1 2 1
9
2 1
2 2
3 1
3 2
3 3
1 1
7 1
7 7
7 4
```

output

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

Note

In the first example, for $a = [10, 20, 10]$ the optimal subsequences are:

- for $k = 1$: $[20]$,
- for $k = 2$: $[10, 20]$,
- for $k = 3$: $[10, 20, 10]$.

D2. Optimal Subsequences (Hard Version)

time limit per test: 3 seconds

memory limit per test: 256 megabytes

input: standard input

output: standard output

This is the harder version of the problem. In this version, $1 \le n, m \le 2 \cdot 10^5$. You can hack this problem if you locked it. But you can hack the previous problem only if you locked both problems.

You are given a sequence of integers $a = [a_1, a_2, \dots, a_n]$ of length n. Its *subsequence* is obtained by removing zero or more elements from the sequence a (they do not necessarily go consecutively). For example, for the sequence $a = [11, 20, 11, 33, 11, 20, 11]$:

- $[11, 20, 11, 33, 11, 20, 11]$, $[11, 20, 11, 33, 11, 20]$, $[11, 11, 11, 11]$, $[20]$, $[33, 20]$ are subsequences (these are just some of the long list);
- $[40]$, $[33, 33]$, $[33, 20, 20]$, $[20, 20, 11, 11]$ are not subsequences.

Suppose that an additional non-negative integer k ($1 \le k \le n$) is given, then the subsequence is called *optimal* if:

- it has a length of k and the sum of its elements is the maximum possible among all subsequences of length k;
- and among all subsequences of length k that satisfy the previous item, it is *lexicographically* minimal.

Recall that the sequence $b = [b_1, b_2, \dots, b_k]$ is lexicographically smaller than the sequence $c = [c_1, c_2, \dots, c_k]$ if the first element (from the left) in which they differ less in the sequence b than in c. Formally: there exists t ($1 \le t \le k$) such that $b_1 = c_1$, $b_2 = c_2$, ..., $b_{t-1} = c_{t-1}$ and at the same time $b_t < c_t$. For example:

- $[10, 20, 20]$ lexicographically less than $[10, 21, 1]$,
- $[7, 99, 99]$ is lexicographically less than $[10, 21, 1]$,
- $[10, 21, 0]$ is lexicographically less than $[10, 21, 1]$.

You are given a sequence of $a = [a_1, a_2, \dots, a_n]$ and m requests, each consisting of two numbers k_j and pos_j ($1 \le k \le n$, $1 \le pos_j \le k_j$). For each query, print the value that is in the index pos_j of the optimal subsequence of the given sequence a for $k = k_j$.

For example, if $n = 4$, $a = [10, 20, 30, 20]$, $k_j = 2$, then the optimal subsequence is $[20, 30]$ — it is the minimum lexicographically among all subsequences of length 2 with the maximum total sum of items. Thus, the answer to the request $k_j = 2$, $pos_j = 1$ is the number 20, and the answer to the request $k_j = 2$, $pos_j = 2$ is the number 30.

Input

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

The second line contains elements of the sequence a: integer numbers $a_1, a_2, \dots, a_n$ ($1 \le a_i \le 10^9$).

The third line contains an integer m ($1 \le m \le 2 \cdot 10^5$) — the number of requests.

The following m lines contain pairs of integers k_j and pos_j ($1 \le k \le n$, $1 \le pos_j \le k_j$) — the requests.

Output

Print m integers $r_1, r_2, \dots, r_m$ ($1 \le r_j \le 10^9$) one per line: answers to the requests in the order they appear in the input. The value of r_j should be equal to the value contained in the position pos_j of the optimal subsequence for $k = k_j$.

Examples

input

```
3
10 20 10
6
1 1
2 1
2 2
```

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

output

```
20
10
20
10
20
10
```

input

```
7
1 2 1 3 1 2 1
9
2 1
2 2
3 1
3 2
3 3
1 1
7 1
7 7
7 4
```

output

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

Note

In the first example, for $a = [10, 20, 10]$ the optimal subsequences are:

- for $k = 1$: $[20]$,
- for $k = 2$: $[10, 20]$,
- for $k = 3$: $[10, 20, 10]$.

E. Arson In Berland Forest

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

The Berland Forest can be represented as an infinite cell plane. Every cell contains a tree. That is, contained before the recent events.

A destructive fire raged through the Forest, and several trees were damaged by it. Precisely speaking, you have a $n \times m$ rectangle map which represents the damaged part of the Forest. The damaged trees were marked as "X" while the remaining ones were marked as "`.`". **You are sure that all burnt trees are shown on the map. All the trees outside the map are undamaged.**

The firemen quickly extinguished the fire, and now they are investigating the cause of it. The main version is that there was an arson: at some moment of time (let's consider it as 0) some trees were set on fire. At the beginning of minute 0, only the trees that were set on fire initially were burning. At the end of each minute, the fire spread from every burning tree to each of 8 neighboring trees. At the beginning of minute T, the fire was extinguished.

The firemen want to find the arsonists as quickly as possible. The problem is, they know neither the value of T (how long the fire has been raging) nor the coordinates of the trees that were initially set on fire. They want you to find the maximum value of T (to know how far could the arsonists escape) and a possible set of trees that could be initially set on fire.

Note that you'd like to maximize value T but the set of trees can be arbitrary.

Input

The first line contains two integer n and m ($1 \le n, m \le 10^6$, $1 \le n \cdot m \le 10^6$) — the sizes of the map.

Next n lines contain the map. The i-th line corresponds to the i-th row of the map and contains m-character string. The j-th character of the i-th string is "X" if the corresponding tree is burnt and "`.`" otherwise.

It's guaranteed that the map contains at least one "X".

Output

In the first line print the single integer T — the maximum time the Forest was on fire. In the next n lines print the certificate: the map ($n \times m$ rectangle) where the trees that were set on fire are marked as "X" and all other trees are marked as "`.`".

Examples

input

```
3 6
XXXXXX
XXXXXX
XXXXXX
```

output

```
1
......
.X.XX.
......
```

input

```
10 10
.XXXXXX...
.XXXXXX...
.XXXXXX...
.XXXXXX...
.XXXXXXXX.
...XXXXXX.
...XXXXXX.
...XXXXXX.
...XXXXXX.
..........
```

output

```
2
..........
..........
...XX.....
..........
..........
..........
.....XX...
..........
..........
..........
```

input

```
4 5
X....
..XXX
..XXX
..XXX
```

output

```
0
X....
..XXX
..XXX
..XXX
```

F1. Wrong Answer on test 233 (Easy Version)

time limit per test: 1 second

memory limit per test: 256 megabytes

input: standard input

output: standard output

Your program fails again. This time it gets "`Wrong answer on test 233`"

.

This is the easier version of the problem. In this version $1 \le n \le 2000$*. You can hack this problem only if you solve and lock both problems.*

The problem is about a test containing n one-choice-questions. Each of the questions contains k options, and only one of them is correct. The answer to the i-th question is h_i, and if your answer of the question i is h_i, you earn 1 point, otherwise, you earn 0 points for this question. The values $h_1, h_2, \dots, h_n$ are known to you in this problem.

However, you have a mistake in your program. It moves the answer clockwise! Consider all the n answers are written in a circle. Due to the mistake in your program, they are shifted by one cyclically.

Formally, the mistake moves the answer for the question i to the question $i \bmod n + 1$. So it moves the answer for the question 1 to question 2, the answer for the question 2 to the question 3, ..., the answer for the question n to the question 1.

We call all the n answers together an *answer suit*. There are k^n possible answer suits in total.

You're wondering, how many answer suits satisfy the following condition: *after moving clockwise by* 1*, the total number of points of the new answer suit is strictly larger than the number of points of the old one*. You need to find the answer modulo $998\,244\,353$.

For example, if $n = 5$, and your answer suit is $a = [1, 2, 3, 4, 5]$, it will submitted as $a' = [5, 1, 2, 3, 4]$ because of a mistake. If the correct answer suit is $h = [5, 2, 2, 3, 4]$, the answer suit a earns 1 point and the answer suite a' earns 4 points. Since $4 > 1$, the answer suit $a = [1, 2, 3, 4, 5]$ should be counted.

Input

The first line contains two integers n, k ($1 \le n \le 2000$, $1 \le k \le 10^9$) — the number of questions and the number of possible answers to each question.

The following line contains n integers $h_1, h_2, \dots, h_n$, $(1 \le h_i \le k)$ — answers to the questions.

Output

Output one integer: the number of answers suits satisfying the given condition, modulo $998\,244\,353$.

Examples

input

```
3 3
1 3 1
```

output

```
9
```

input

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

output

```
1000
```

Note

For the first example, valid answer suits are $[2, 1, 1], [2, 1, 2], [2, 1, 3], [3, 1, 1], [3, 1, 2], [3, 1, 3], [3, 2, 1], [3, 2, 2], [3, 2, 3]$.

F2. Wrong Answer on test 233 (Hard Version)

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

Your program fails again. This time it gets "Wrong answer on test 233"

.

This is the harder version of the problem. In this version, $1 \le n \le 2 \cdot 10^5$. You can hack this problem if you locked it. But you can hack the previous problem only if you locked both problems.

The problem is to finish n one-choice-questions. Each of the questions contains k options, and only one of them is correct. The answer to the i-th question is h_i, and if your answer of the question i is h_i, you earn 1 point, otherwise, you earn 0 points for this question. The values $h_1, h_2, \dots, h_n$ are known to you in this problem.

However, you have a mistake in your program. It moves the answer clockwise! Consider all the n answers are written in a circle. Due to the mistake in your program, they are shifted by one cyclically.

Formally, the mistake moves the answer for the question i to the question $i \bmod n + 1$. So it moves the answer for the question 1 to question 2, the answer for the question 2 to the question 3, ..., the answer for the question n to the question 1.

We call all the n answers together an *answer suit*. There are k^n possible answer suits in total.

You're wondering, how many answer suits satisfy the following condition: *after moving clockwise by 1, the total number of points of the new answer suit is strictly larger than the number of points of the old one*. You need to find the answer modulo $998\,244\,353$.

For example, if $n = 5$, and your answer suit is $a = [1, 2, 3, 4, 5]$, it will submitted as $a' = [5, 1, 2, 3, 4]$ because of a mistake. If the correct answer suit is $h = [5, 2, 2, 3, 4]$, the answer suit a earns 1 point and the answer suite a' earns 4 points. Since $4 > 1$, the answer suit $a = [1, 2, 3, 4, 5]$ should be counted.

Input

The first line contains two integers n, k ($1 \le n \le 2 \cdot 10^5$, $1 \le k \le 10^9$) — the number of questions and the number of possible answers to each question.

The following line contains n integers $h_1, h_2, \dots, h_n$, $(1 \le h_i \le k)$ — answers to the questions.

Output

Output one integer: the number of answers suits satisfying the given condition, modulo $998\,244\,353$.

Examples

input

```
3 3
```

```
1 3 1
```

output

```
9
```

input

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

output

```
1000
```

input

```
6 2
1 1 2 2 1 1
```

output

```
16
```

Note

For the first example, valid answer suits are $[2,1,1]$, $[2,1,2]$, $[2,1,3]$, $[3,1,1]$, $[3,1,2]$, $[3,1,3]$, $[3,2,1]$, $[3,2,2]$, $[3,2,3]$.

G. Not Same

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

You are given an integer array $a_1, a_2, \dots, a_n$, where a_i represents the number of blocks at the i-th position. It is guaranteed that $1 \le a_i \le n$.

In one operation you can choose a subset of indices of the given array and remove one block in each of these indices. You can't remove a block from a position without blocks.

All subsets that you choose should be different (unique).

You need to remove all blocks in the array using at most $n+1$ operations. It can be proved that the answer always exists.

Input

The first line contains a single integer n ($1 \le n \le 10^3$) — length of the given array.

The second line contains n integers $a_1, a_2, \dots, a_n$ ($1 \le a_i \le n$) — numbers of blocks at positions $1, 2, \dots, n$.

Output

In the first line print an integer op ($0 \le op \le n+1$).

In each of the following op lines, print a binary string s of length n. If $s_i =$'0', it means that the position i is not in the chosen subset. Otherwise, s_i should be equal to '1' and the position i is in the chosen subset.

All binary strings should be distinct (unique) and a_i should be equal to the sum of s_i among all chosen binary strings.

If there are multiple possible answers, you can print any.

It can be proved that an answer always exists.

Examples

input

```
5
5 5 5 5 5
```

output

```
6
11111
01111
10111
11011
11101
11110
```

input

```
5
5 1 1 1 1
```

output

```
5
11000
10000
```

```
10100
10010
10001
```

input

```
5
4 1 5 3 4
```

output

```
5
11111
10111
10101
00111
10100
```

Note

In the first example, the number of blocks decrease like that:

$\{5,5,5,5,5\} \to \{4,4,4,4,4\} \to \{4,3,3,3,3\} \to \{3,3,2,2,2\} \to \{2,2,2,1,1\} \to \{1,1,1,1,0\} \to \{0,0,0,0,0\}$. And we can note that each operation differs from others.

Codeforces (c) Copyright 2010-2022 Mike Mirzayanov
The only programming contests Web 2.0 platform