Codeforces Round 979 (Div. 2)

A. A Gift From Orangutan

1 second, 256 megabytes

While exploring the jungle, you have bumped into a rare orangutan with a bow tie! You shake hands with the orangutan and offer him some food and water. In return...

The orangutan has gifted you an array a of length n. Using a, you will construct two arrays b and c, both containing n elements, in the following manner:

- $b_i = \min(a_1, a_2, \ldots, a_i)$ for each $1 \leq i \leq n$.
- $c_i = \max(a_1, a_2, \ldots, a_i)$ for each $1 \leq i \leq n$.

Define the *score* of a as $\sum_{i=1}^{n} c_i - b_i$ (i.e. the sum of $c_i - b_i$ over all $1 \leq i \leq n$). Before you calculate the *score*, you can **shuffle** the elements of a however you want.

Find the maximum *score* that you can get if you shuffle the elements of a optimally.

Input

The first line contains t $(1 \leq t \leq 100)$ — the number of test cases.

The first line of each test case contains an integer n $(1 \leq n \leq 1000)$ — the number of elements in a.

The following line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \leq a_i \leq 1000$) — the elements of the array a.

It is guaranteed that the sum of n over all test cases does not exceed 1000.

Output

For each test case, output the maximum *score* that you can get.

input

```
3
1
69
3
7 6 5
5
1 1 1 2 2
```

output

```
0
4
4
```

In the first test case, there is no other way to rearrange a. So, $b = [69]$ and $c = [69]$. The only possible *score* is $69 - 69 = 0$.

In the second test case, you can rearrange a as $[7, 5, 6]$. Here, $b = [7, 5, 5]$ and $c = [7, 7, 7]$. The *score* in this case is $(7-7) + (7-5) + (7-5) = 4$. It can be shown this is the maximum possible *score*.

The problem statement has recently been changed. View the changes. ×

B. Minimise Oneness

1.5 seconds, 256 megabytes

For an arbitrary binary string t^*, let $f(t)$ be the number of non-empty subsequences[†] of t that contain only `0`, and let $g(t)$ be the number of non-empty subsequences of t that contain at least one `1`.

Note that for $f(t)$ and for $g(t)$, each subsequence is counted as many times as it appears in t. E.g., $f(\texttt{000}) = 7, g(\texttt{100}) = 4$.

We define the *oneness* of the binary string t to be $|f(t) - g(t)|$, where for an arbitrary integer z, $|z|$ represents the absolute value of z.

You are given a positive integer n. Find a binary string s of length n such that its *oneness* is as small as possible. If there are multiple strings, you can print any of them.

* A binary string is a string that only consists of characters `0` and `1`.

† A sequence a is a subsequence of a sequence b if a can be obtained from b by the deletion of several (possibly, zero or all) elements. For example, subsequences of `1011101` are `0`, `1`, `11111`, `0111`, but not `000` nor `11100`.

Input

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

The only line of each test case contains an integer n ($1 \leq n \leq 2 \cdot 10^5$) — the length of s.

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 s on a new line. If multiple answers exist, output any.

input

```
3
1
2
3
```

output

```
0
01
010
```

In the first test case, for the example output, $f(t) = 1$ because there is one subsequence that contains only `0` (`0`), and $g(t) = 0$ because there are no subsequences that contain at least one 1. The *oneness* is $|1 - 0| = 1$. The output `1` is correct as well because its *oneness* is $|0 - 1| = 1$.

For the example output of the second test case, $f(t) = 1$ because there is one non-empty subsequence that contains only `0`, and $g(t) = 2$ because there are two non-empty subsequences that contain at least one `1` (`01` and `1`). The *oneness* is thus $|1 - 2| = 1$. It can be shown that 1 is the minimum possible value of its *oneness* over all possible binary strings of size 2.

C. A TRUE Battle

2 seconds, 256 megabytes

Alice and Bob are playing a game. There is a list of n booleans, each of which is either `true` or `false`, given as a binary string * of length n (where `1` represents `true`, and `0` represents `false`). Initially, there are no operators between the booleans.

Alice and Bob will take alternate turns placing `and` or `or` between the booleans, with Alice going first. Thus, the game will consist of $n - 1$ turns since there are n booleans. Alice aims for the final statement to evaluate to `true`, while Bob aims for it to evaluate to `false`. Given the list of boolean values, determine whether Alice will win if both players play optimally.

To evaluate the final expression, repeatedly perform the following steps until the statement consists of a single `true` or `false`:

- If the statement contains an `and` operator, choose any one and replace the subexpression surrounding it with its evaluation.
- Otherwise, the statement contains an `or` operator. Choose any one and replace the subexpression surrounding the `or` with its evaluation.

For example, the expression `true or false and false` is evaluated as `true or (false and false)` $=$ `true or false` $=$ `true`. It can be shown that the result of any compound statement is unique.

* A binary string is a string that only consists of characters 0 and 1

Input

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

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

The second line contains a binary string of length n, consisting of characters 0 and 1 — the list of boolean values.

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

Output

For each testcase, output "YES" (without quotes) if Alice wins, and "NO" (without quotes) otherwise.

You can output "YES" and "NO" in any case (for example, strings "yES", "yes" and "Yes" will be recognized as a positive response).

input

```
5
2
11
3
010
12
101111111100
10
0111111011
8
01000010
```

output

```
YES
NO
YES
YES
NO
```

In the first testcase, Alice can place `and` between the two booleans. The game ends as there are no other places to place operators, and Alice wins because `true and true` is `true`.

In the second testcase, Alice can place `or` between the middle `true` and the left `false`. Bob can place `and` between the middle `true` and the right `false`. The statement `false or true and false` is `false`.

Note that these examples may not be the best strategies for either Alice or Bob.

D. QED's Favorite Permutation

2 seconds, 256 megabytes

QED is given a permutation* p of length n. He also has a string s of length n containing only characters $\texttt{L}$ and $\texttt{R}$. QED only likes permutations that are sorted in non-decreasing order. To sort p, he can select any of the following operations and perform them any number of times:

- Choose an index i such that $s_i = \texttt{L}$. Then, swap p_i and p_{i-1}. It is guaranteed that $s_1 \neq \texttt{L}$.
- Choose an index i such that $s_i = \texttt{R}$. Then, swap p_i and p_{i+1}. It is guaranteed that $s_n \neq \texttt{R}$.

He is also given q queries. In each query, he selects an index i and changes s_i from $\texttt{L}$ to $\texttt{R}$ (or from $\texttt{R}$ to $\texttt{L}$). Note that the changes are **persistent**.

After each query, he asks you if it is possible to sort p in non-decreasing order by performing the aforementioned operations any number of times. Note that before answering each query, the permutation p is reset to its original form.

---

*A permutation of length n is an array consisting of n distinct integers from 1 to n 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 ($n=3$ but there is 4 in the array).

Input

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

The first line of each test case contains two integers n and q ($3 \le n \le 2 \cdot 10^5$, $1 \le q \le 2 \cdot 10^5$) – the length of the permutation and the number of queries.

The following line contains n integers $p_1, p_2, \ldots, p_n$ $(1 \le p_i \le n$, p is a permutation).

The following line contains n characters $s_1 s_2 \ldots s_n$. It is guaranteed that s_i is either $\mathtt{L}$ or $\mathtt{R}$, $s_1 = \mathtt{R}$, and $s_n = \mathtt{L}$.

The following q lines contain an integer i ($2 \le i \le n-1$), denoting that s_i is changed from $\mathtt{L}$ to $\mathtt{R}$ (or from $\mathtt{R}$ to $\mathtt{L}$).

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

Output

For each query, output "YES" (without quotes) if it is possible, and "NO" (without quotes) otherwise.

You can output "YES" and "NO" in any case (for example, strings "yES", "yes" and "Yes" will be recognized as a positive response).

input

```
3
5 3
1 4 2 5 3
RLRLL
2
4
3
8 5
1 5 2 4 8 3 6 7
RRLLRRRL
4
3
5
3
4
6 2
1 2 3 4 5 6
RLRLRL
4
5
```

output

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

In the first testcase, $s = \mathtt{RRRLL}$ after the first query. QED may sort p using the following operations:

- Initially, $p = [1,4,2,5,3]$.
- Select $i = 2$ and swap p_2 with p_3. Now, $p = [1,2,4,5,3]$.
- Select $i = 5$ and swap p_5 with p_4. Now, $p = [1,2,4,3,5]$.
- Select $i = 4$ and swap p_4 with p_3. Now, $p = [1,2,3,4,5]$, which is in non-decreasing order.

It can be shown that it is impossible to sort the array after all three updates of the first testcase.

E. MEXimize the Score

2 seconds, 256 megabytes

Suppose we partition the elements of an array b into any number k of non-empty multisets $S_1, S_2, \ldots, S_k$, where k is an arbitrary positive integer. Define the *score* of b as the maximum value of $\operatorname{MEX}(S_1)^*$ $+ \operatorname{MEX}(S_2) + \ldots + \operatorname{MEX}(S_k)$ over all possible partitions of b for any integer k.

Envy is given an array a of size n. Since he knows that calculating the *score* of a is too easy for you, he instead asks you to calculate the sum of *scores* of all $2^n - 1$ non-empty subsequences of a.† Since this answer may be large, please output it modulo $998\,244\,353$.

*MEX of a collection of integers $c_1, c_2, \ldots, c_k$ is defined as the smallest non-negative integer x that does not occur in the collection c. For example, $\operatorname{MEX}([0,1,2,2]) = 3$ and $\operatorname{MEX}([1,2,2]) = 0$

†A sequence x is a subsequence of a sequence y if x can be obtained from y by deleting several (possibly, zero or all) elements.

Input

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

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

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($0 \le a_i < n$) — the elements of the 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 the answer, modulo $998\,244\,353$.

input

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

output

```
11
26
53
0
```

In the first testcase, we must consider seven subsequences:

- $[0]$: The score is 1.
- $[0]$: The score is 1.
- $[1]$: The score is 0.
- $[0,0]$: The score is 2.
- $[0,1]$: The score is 2.
- $[0,1]$: The score is 2.
- $[0,0,1]$: The score is 3.

The answer for the first testcase is $1+1+2+2+2+3 = 11$.
In the last testcase, all subsequences have a score of 0.

F. Orangutan Approved Subarrays

3 seconds, 512 megabytes

Suppose you have an array b. Initially, you also have a set S that contains all distinct elements of b. The array b is called *orangutan-approved* if it can be **emptied** by repeatedly performing the following operation:

- In one operation, select indices l and r ($1 \le l \le r \le |b|$) such that $v = b_l = b_{l+1} = \ldots = b_r$ and v is present in S. Remove v from S, and simultaneously remove all b_i such that $l \le i \le r$. Then, reindex the elements $b_{r+1}, b_{r+2}, \ldots$ as $b_l, b_{l+1}, \ldots$ accordingly.

You are given an array a of length n and q queries.

Each query consists of two indices l and r ($1 \le l \le r \le n$), and you need to determine whether or not the subarray $a_l, a_{l+1}, \ldots, a_r$ is *orangutan-approved*.

Input

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

The first line of each test case contains integers n and q ($1 \le n, q \le 2 \cdot 10^5$) — the size of a and the number of queries, respectively.

The following line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le n$) — the elements of the array a.

The following q lines contain two integers l and r — the endpoints of the subarray for each query ($1 \le l \le r \le n$).

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

Output

For each query, output "YES" (without quotes) if the subarray from l to r is *orangutan-approved*, and "NO" (without quotes) otherwise.

You can output "YES" and "NO" in any case (for example, strings "yES", "yes" and "Yes" will be recognized as a positive response).

input

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

output

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

In the first query of the first testcase, the answer is YES.

- Initially, $S = \{1, 2\}$ and $b = [1, 2, 2, 1]$
- Select $l = 2$ and $r = 3$. Since $b_2 = b_3 = 2$ is in S, we may erase b_2 and b_3 from the array, as well as erasing 2 from S. The set S becomes $\{1\}$ and the array becomes $[1, 1]$.
- Select $l = 1$ and $r = 2$. Since $b_1 = b_2 = 1$ is in S, we may erase b_1 and b_2 from the array, as well as erasing 1 from S. The set S becomes $\{\}$ and the array becomes $[]$.
- Since the array is now empty, we can say the original array is orangutan-approved.

In the first query of the second testcase, the answer is `NO`, because it can be shown that the subarray $[2, 1, 2, 1]$ cannot become empty through any sequence of valid operations.

G1. The Destruction of the Universe (Easy Version)

4 seconds, 512 megabytes

This is the easy version of the problem. In this version, $n \le 5000$. You can only make hacks if both versions of the problem are solved.

Orangutans are powerful beings—so powerful that they only need 1 unit of time to destroy every vulnerable planet in the universe!

There are n planets in the universe. Each planet has an *interval of vulnerability* $[l, r]$, during which it will be exposed to destruction by orangutans. Orangutans can also expand the *interval of vulnerability* of any planet by 1 unit.

Specifically, suppose the expansion is performed on planet p with *interval of vulnerability* $[l_p, r_p]$. Then, the resulting *interval of vulnerability* may be either $[l_p - 1, r_p]$ or $[l_p, r_p + 1]$.

Given a set of planets, orangutans can destroy all planets if the *intervals of vulnerability* of all planets in the set intersect at least one common point. Let the *score* of such a set denote the minimum number of expansions that must be performed.

Orangutans are interested in the sum of **scores** of all *non-empty* subsets of the planets in the universe. As the answer can be large, output it modulo $998\,244\,353$.

Input

The first line contains an integer t ($1 \le t \le 1000$) — the number of test cases.

The first line of each test case contains an integer n ($1 \le n \le 5000$) — the number of planets in the universe.

The following n lines contain two integers l_i and r_i ($1 \le l_i \le r_i \le n$) — the initial *interval of vulnerability* of the i-th planet.

It is guaranteed that the sum of n does not exceed 5000 over all test cases.

Output

For each test case, output an integer — the sum of *scores* to destroy all **non-empty** subsets of the planets in the universe, modulo $998\,244\,353$.

input

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

output

```
5
6
24
```

In the first testcase, there are seven non-empty subsets of planets we must consider:

- For each of the subsets $\{[1,1]\}, \{[2,3]\}, \{[3,3]\}$, the score is 0.
- For the subset $\{[2,3],[3,3]\}$, the score is 0, because the point 3 is already contained in both planets' *interval of vulnerability*.
- For the subset $\{[1,1],[2,3]\}$, the score is 1. By using one operation on changing the *interval of vulnerability* of the second planet to be

$[1, 3]$, the two planets now both have the point 1 in their interval.

- For the subset $\{[1, 1], [3, 3]\}$, the score is 2. By using two operations on changing the *interval of vulnerability* of the first planet to be $[1, 3]$, the two planets now both have the point 3 in their interval.
- For the subset $\{[1, 1], [2, 3], [3, 3]\}$, the score is 2. By using one operation on changing the *interval of vulnerability* of the first planet to be $[1, 2]$ and one operation on changing the *interval of vulnerability* of the third planet to $[2, 3]$, all three planets will have the point 2 in their interval.

The sum of scores of all non-empty subsets of the first testcase is $0 \cdot 4 + 1 \cdot 1 + 2 \cdot 2 = 5$.

G2. The Destruction of the Universe (Hard Version)

4 seconds, 512 megabytes

This is the hard version of the problem. In this version, $n \le 10^6$. You can only make hacks if both versions of the problem are solved.

Orangutans are powerful beings—so powerful that they only need 1 unit of time to destroy every vulnerable planet in the universe!

There are n planets in the universe. Each planet has an *interval of vulnerability* $[l, r]$, during which it will be exposed to destruction by orangutans. Orangutans can also expand the *interval of vulnerability* of any planet by 1 unit.

Specifically, suppose the expansion is performed on planet p with *interval of vulnerability* $[l_p, r_p]$. Then, the resulting *interval of vulnerability* may be either $[l_p - 1, r_p]$ or $[l_p, r_p + 1]$.

Given a set of planets, orangutans can destroy all planets if the *intervals of vulnerability* of all planets in the set intersect at least one common point. Let the *score* of such a set denote the minimum number of expansions that must be performed.

Orangutans are interested in the sum of **scores** of all *non-empty* subsets of the planets in the universe. As the answer can be large, output it modulo $998\,244\,353$.

Input

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

The first line of each test case contains an integer n ($1 \le n \le 10^6$) — the number of planets in the universe.

The following n lines contain two integers l_i and r_i ($1 \le l_i \le r_i \le n$) — the initial *interval of vulnerability* of the i-th planet.

It is guaranteed that the sum of n does not exceed 10^6 over all test cases.

Output

For each test case, output an integer — the sum of *scores* to destroy all **non-empty** subsets of the planets in the universe, modulo $998\,244\,353$.

input

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

output

```
5
6
24
```

In the first testcase, there are seven non-empty subsets of planets we must consider:

- For each of the subsets $\{[1,1]\}, \{[2,3]\}, \{[3,3]\}$, the score is 0.
- For the subset $\{[2,3],[3,3]\}$, the score is 0, because the point 3 is already contained in both planets' *interval of vulnerability*.
- For the subset $\{[1,1],[2,3]\}$, the score is 1. By using one operation on changing the *interval of vulnerability* of the second planet to be $[1,3]$, the two planets now both have the point 1 in their interval.
- For the subset $\{[1,1],[3,3]\}$, the score is 2. By using two operations on changing the *interval of vulnerability* of the first planet to be $[1,3]$, the two planets now both have the point 3 in their interval.
- For the subset $\{[1,1],[2,3],[3,3]\}$, the score is 2. By using one operation on changing the *interval of vulnerability* of the first planet to be $[1,2]$ and one operation on changing the *interval of vulnerability* of the third planet to $[2,3]$, all three planets will have the point 2 in their interval.

The sum of scores of all non-empty subsets of the first testcase is $0 \cdot 4 + 1 \cdot 1 + 2 \cdot 2 = 5$.