Codeforces Round 1029 (Div. 3)

A. False Alarm

1 second, 256 megabytes

Yousef is at the entrance of a long hallway with n doors in a row, numbered from 1 to n. He needs to pass through all the doors from 1 to n in order of numbering and reach the exit (past door n).

Each door can be open or closed. If a door is open, Yousef passes through it in 1 second. If the door is closed, Yousef can't pass through it.

However, Yousef has a special button which he can use **at most once** at any moment. This button makes all closed doors become open for x seconds.

Your task is to determine if Yousef can pass through all the doors if he can use the button at most once.

Input

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

The first line of each test case contains two integers n, x ($1 \le n, x \le 10$) — the number of doors and the number of seconds of the button, respectively.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($a_i \in \{0, 1\}$) — the state of each door. Open doors are represented by `'0'`, while closed doors are represented by `'1'`.

It is guaranteed that each test case contains at least one closed door.

Output

For each test case, output "`YES`" if Yousef can reach the exit, and "`NO`" otherwise.

You can output the answer in any case (upper or lower). For example, the strings "`yEs`", "`yes`", "`Yes`", and "`YES`" will be recognized as positive responses.

input

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

output

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

In the first test case, the optimal way is as follows:

- At time 0, the door is open, so Yousef passes.
- At time 1, the door is closed, Yousef can use the button now and pass through the door.
- At time 2, the button's effect is still on, so Yousef can still pass.
- At time 3, the button's effect has finished, but the door is open. Yousef passes and reaches the exit.

In the second test case, Yousef has a 3-second button, but he would need at least a 4-second button to reach the exit. Therefore, the answer is `NO`.

In the third test case, Yousef can turn on the button before starting to move. All the doors will stay open until he reaches the exit.

B. Shrink

2 seconds, 256 megabytes

A shrink operation on an array a of size m is defined as follows:

- Choose an index i ($2 \le i \le m - 1$) such that $a_i > a_{i-1}$ and $a_i > a_{i+1}$.
- Remove a_i from the array.

Define the *score* of a permutation* p as the *maximum* number of times that you can perform the shrink operation on p.

Yousef has given you a single integer n. Construct a permutation p of length n with the **maximum** possible *score*. If there are multiple answers, you can output any of them.

*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 of the input contains an integer t ($1 \le t \le 10^3$) — the number of test cases.

Each test case contains an integer n ($3 \le n \le 2 \cdot 10^5$) — the size of the permutation.

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 any permutation $p_1, p_2, \ldots, p_n$ that maximizes the number of shrink operations.

input

```
2
3
6
```

output

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

In the first test case:

- We choose $p = [1, 3, 2]$.
- Choose index 2, and remove p_2 from the array. The array becomes $p = [1, 2]$.

It can be shown that the maximum number of operations we can perform is 1. Another valid answer is $p = [2, 3, 1]$.

In the second test case:

- We choose $p = [2, 3, 6, 4, 5, 1]$.
- Choose index 5, and remove p_5 from the array. The array becomes $p = [2, 3, 6, 4, 1]$.
- Choose index 3, and remove p_3 from the array. The array becomes $p = [2, 3, 4, 1]$.
- Choose index 3, and remove p_3 from the array. The array becomes $p = [2, 3, 1]$.
- Choose index 2, and remove p_2 from the array. The array becomes $p = [2, 1]$.

The maximum number of operations we can perform is 4. Any permutation with a score of 4 is valid.

C. Cool Partition

2 seconds, 256 megabytes

Yousef has an array a of size n. He wants to partition the array into one or more contiguous segments such that each element a_i belongs to exactly one segment.

A partition is called *cool* if, for every segment b_j, all elements in b_j also appear in b_{j+1} (if it exists). That is, every element in a segment must also be present in the segment following it.

For example, if $a = [1, 2, 2, 3, 1, 5]$, a *cool* partition Yousef can make is $b_1 = [1, 2]$, $b_2 = [2, 3, 1, 5]$. This is a *cool* partition because every element in b_1 (which are 1 and 2) also appears in b_2. In contrast, $b_1 = [1, 2, 2]$, $b_2 = [3, 1, 5]$ is not a *cool* partition, since 2 appears in b_1 but not in b_2.

Note that after partitioning the array, you do **not** change the order of the segments. Also, note that if an element appears several times in some segment b_j, it only needs to appear at least once in b_{j+1}.

Your task is to help Yousef by finding the maximum number of segments that make a *cool* partition.

Input

The first line of the input contains 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 size of the array.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le n$) — the elements of the array.

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

Output

For each test case, print one integer — the maximum number of segments that make a *cool* partition.

input

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

output

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

The first test case is explained in the statement. We can partition it into $b_1 = [1, 2]$, $b_2 = [2, 3, 1, 5]$. It can be shown there is no other partition with more segments.

In the second test case, we can partition the array into $b_1 = [1, 2]$, $b_2 = [1, 3, 2]$, $b_3 = [1, 3, 2]$. The maximum number of segments is 3.

In the third test case, the only partition we can make is $b_1 = [5, 4, 3, 2, 1]$. Any other partition will not satisfy the condition. Therefore, the answer is 1.

D. Retaliation

2 seconds, 256 megabytes

Yousef wants to explode an array $a_1, a_2, \ldots, a_n$. An array gets exploded when all of its elements become equal to zero.

In one operation, Yousef can do **exactly** one of the following:

1. For every index i in a, decrease a_i by i.
2. For every index i in a, decrease a_i by $n - i + 1$.

Your task is to help Yousef determine if it is possible to explode the array using any number of operations.

Input

The first line of the input 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 ($2 \le n \le 2 \cdot 10^5$) — the size of the array.

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

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

Output

For each test case, print "YES" if Yousef can explode the array, otherwise output "NO".

You can output the answer in any case (upper or lower). For example, the strings "yEs", "yes", "Yes", and "YES" will be recognized as positive responses.

input

```
6
4
3 6 6 3
5
21 18 15 12 9
10
2 6 10 2 5 5 1 2 4 10
7
10 2 16 12 8 20 4
2
52 101
2
10 2
```

output

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

In the second test case, we can do the following:

- Perform 1 operation of the first type. The array becomes $[20, 16, 12, 8, 4]$.
- Perform 4 operations of the second type. The array becomes $[0, 0, 0, 0, 0]$.

In the first, third, fourth, and sixth test cases, it can be proven that it is impossible to make all elements equal to zero using any number of operations.

E. Lost Soul

2 seconds, 256 megabytes

You are given two integer arrays a and b, each of length n.

You may perform the following operation any number of times:

- Choose an index i $(1 \le i \le n-1)$, and set $a_i := b_{i+1}$, **or** set $b_i := a_{i+1}$.

Before performing any operations, you are allowed to choose an index i $(1 \le i \le n)$ and remove both a_i and b_i from the arrays. This removal can be done **at most once**.

Let the number of matches between two arrays c and d of length m be the number of positions j $(1 \le j \le m)$ such that $c_j = d_j$.

Your task is to compute the maximum number of matches you can achieve.

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

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

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

The third line contains n integers $b_1, b_2, \ldots, b_n$ $(1 \le b_i \le n)$ — the elements of b.

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

Output
For each test case, print a single integer — the answer for the test case.

input

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

output

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

In the first test case, we can do the following:

- We will choose not to remove any index.
- Choose index 3, and set $a_3 := b_4$. The arrays become: $a = [1, 3, 2, 4]$, $b = [4, 3, 2, 2]$.
- Choose index 1, and set $a_1 := b_2$. The arrays become: $a = [3, 3, 2, 4]$, $b = [4, 3, 2, 2]$.
- Choose index 1, and set $b_1 := a_2$. The arrays become: $a = [3, 3, 2, 4]$, $b = [3, 3, 2, 2]$. Notice that you can perform $a_i := b_{i+1}$ and $b_i := a_{i+1}$ on the same index i.

The number of matches is 3. It can be shown that this is the maximum answer we can achieve.

In the second test case, we can do the following to achieve a maximum of 3:

- Let's choose to remove index 5. The arrays become: $a = [2, 1, 5, 3, 4]$, $b = [3, 2, 4, 5, 6]$.
- Choose index 4, and set $b_4 := a_5$. The arrays become: $a = [2, 1, 5, 3, 4]$, $b = [3, 2, 4, 4, 6]$.
- Choose index 3, and set $a_3 := b_4$. The arrays become: $a = [2, 1, 4, 3, 4]$, $b = [3, 2, 4, 4, 6]$.
- Choose index 2, and set $a_2 := b_3$. The arrays become: $a = [2, 4, 4, 3, 4]$, $b = [3, 2, 4, 4, 6]$.
- Choose index 1, and set $b_1 := a_2$. The arrays become: $a = [2, 4, 4, 3, 4]$, $b = [4, 2, 4, 4, 6]$.
- Choose index 2, and set $b_2 := a_3$. The arrays become: $a = [2, 4, 4, 3, 4]$, $b = [4, 4, 4, 4, 6]$.
- Choose index 1, and set $a_1 := b_2$. The arrays become: $a = [4, 4, 4, 3, 4]$, $b = [4, 4, 4, 4, 6]$.

In the third test case, it can be shown that we can not get any matches. Therefore, the answer is 0.

F. Wildflower

2 seconds, 256 megabytes

Yousef has a rooted tree* consisting of exactly n vertices, which is rooted at vertex 1. You would like to give Yousef an array a of length n, where each a_i $(1 \le i \le n)$ **can either be** 1 **or** 2.

Let s_u denote the sum of a_v where vertex v is in the subtree† of vertex u. Yousef considers the tree *special* if all the values in s are **pairwise distinct** (i.e., all subtree sums are unique).

Your task is to help Yousef count the number of different arrays a that result in the tree being *special*. Two arrays b and c are different if there exists an index i such that $b_i \neq c_i$.

As the result can be very large, you should print it modulo $10^9 + 7$.

*A tree is a connected undirected graph with $n - 1$ edges.

† The subtree of a vertex v is the set of all vertices that pass through v on a simple path to the root. Note that vertex v is also included in the set.

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

Each test case consists of several lines. The first line of the test case contains an integer n $(2 \le n \le 2 \cdot 10^5)$ — the number of vertices in the tree.

Then $n - 1$ lines follow, each of them contains two integers u and v $(1 \le u, v \le n, u \neq v)$ which describe a pair of vertices connected by an edge. It is guaranteed that the given graph is a tree and has no loops or multiple edges.

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

Output
For each test case, print one integer x — the number of different arrays a that result in the tree being *special*, modulo $10^9 + 7$.

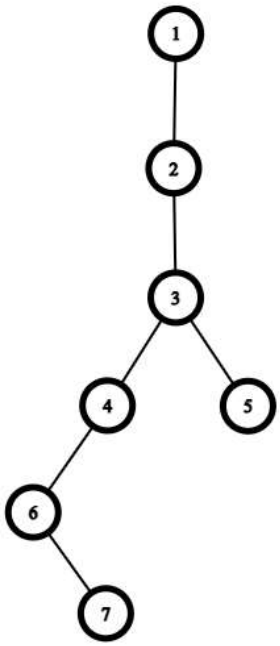
input

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

output

```
4
24
0
16
48
0
4
```

The tree given in the fifth test case:



G. Omg Graph

2 seconds, 256 megabytes

You are given an undirected connected weighted graph. Define the cost of a path of length k to be as follows:

- Let the weights of all the edges on the path be $w_1, \ldots, w_k$.
- The cost of the path is $(\min_{i=1}^{k} w_i) + (\max_{i=1}^{k} w_i)$, or in other words, the maximum edge weight + the minimum edge weight.

Across all paths from vertex 1 to n, report the cost of the path with minimum cost. Note that the path is not necessarily simple.

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 two integers n and m ($2 \le n \le 2 \cdot 10^5, n - 1 \le m \le \min(2 \cdot 10^5, \frac{n(n-1)}{2})$).

The next m lines each contain integers u, v and w ($1 \le u, v \le n, 1 \le w \le 10^9$) representing an edge from vertex u to v with weight w. It is guaranteed that the graph does not contain self-loops or multiple edges and the resulting graph is connected.

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

Output

For each test case, output a single integer, the minimum cost path from vertex 1 to n.

input

```
4
3 2
1 2 1
2 3 1
3 2
1 3 13
1 2 5
8 9
1 2 6
2 3 5
3 8 6
1 4 7
4 5 4
5 8 7
1 6 5
6 7 5
7 8 5
3 3
1 3 9
1 2 8
2 3 3
```

output

```
2
18
10
11
```

For the second test case, the optimal path is $1 \to 2 \to 1 \to 3$, the edge weights are $5, 5, 13$ so the cost is $\min(5, 5, 13) + \max(5, 5, 13) = 5 + 13 = 18$. It can be proven that there is no path with lower cost.

H. Incessant Rain

3 seconds, 192 megabytes

Note the unusual memory limit.

Silver Wolf gives you an array a of length n and q queries. In each query, she replaces an element in a. After each query, she asks you to output the maximum integer k such that there exists an integer x such that it is the k-majority of a subarray* of a.

An integer y is the k-majority of array b if y appears at least $\lfloor \frac{|b|+1}{2} \rfloor + k$ times in b, where $|b|$ represents the length of b. Note that b may not necessarily have a k-majority.

*An array b is a subarray of an array a if b can be obtained from a by the deletion of several (possibly, zero or all) elements from the beginning and several (possibly, zero or all) elements from the end.

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 two integers n and q ($1 \le n, q \le 3 \cdot 10^5$) — the length of a and the number of queries.

The following line contains n space-separated integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le n$).

The following q lines contain two integers i and x, denoting the query that replaces a_i with x ($1 \le i, x \le n$).

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

Output

For each test case, output the answer to all queries on a single new line, separated by a space.

input

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

output

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