A. Directional Increase

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

We have an array of length n. Initially, each element is equal to 0 and there is a pointer located on the first element.

We can do the following two kinds of operations any number of times (possibly zero) in any order:

1. If the pointer is not on the last element, increase the element the pointer is currently on by 1. Then move it to the next element.
2. If the pointer is not on the first element, decrease the element the pointer is currently on by 1. Then move it to the previous element.

But there is one additional rule. **After we are done, the pointer has to be on the first element.**

You are given an array a. Determine whether it's possible to obtain a after some operations or not.

Input

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

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

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($-10^9 \le a_i \le 10^9$) — 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" (without quotes) if it's possible to obtain a after some operations, 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).

Example

input

```
7
2
1 0
4
2 -1 -1 0
4
1 -4 3 0
4
1 -1 1 -1
5
1 2 3 4 -10
7
2 -1 1 -2 0 0 0
1
0
```

output

```
No
Yes
No
No
Yes
Yes
Yes
```

Note

In the first test case we can obtain the array after some operations, but the pointer won't be on the first element.

One way of obtaining the array in the second test case is shown below.

$\langle \underline{0}, 0, 0, 0\rangle \to \langle 1, \underline{0}, 0, 0\rangle \to \langle \underline{1}, -1, 0, 0\rangle \to \langle 2, \underline{-1}, 0, 0\rangle \to \langle 2, 0, \underline{0}, 0\rangle \to \langle 2, \underline{0}, -1, 0\rangle \to \langle \underline{2}, -1, -1, 0\rangle$

B. Fake Plastic Trees

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

We are given a rooted tree consisting of n vertices numbered from 1 to n. The root of the tree is the vertex 1 and the parent of the vertex v is p_v.

There is a number written on each vertex, initially all numbers are equal to 0. Let's denote the number written on the vertex v as a_v.

For each v, we want a_v to be between l_v and r_v $(l_v \leq a_v \leq r_v)$.

In a single operation we do the following:

- Choose some vertex v. Let $b_1, b_2, \ldots, b_k$ be vertices on the path from the vertex 1 to vertex v (meaning $b_1 = 1$, $b_k = v$ and $b_i = p_{b_{i+1}}$).
- Choose a non-decreasing array c of length k of nonnegative integers: $0 \leq c_1 \leq c_2 \leq \ldots \leq c_k$.
- For each i $(1 \leq i \leq k)$, increase a_{b_i} by c_i.

What's the minimum number of operations needed to achieve our goal?

Input

The first line contains an 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 $(2 \leq n \leq 2 \cdot 10^5)$ — the number of the vertices in the tree.

The second line of each test case contains $n-1$ integers, $p_2, p_3, \ldots, p_n$ $(1 \leq p_i < i)$, where p_i denotes the parent of the vertex i.

The i-th of the following n lines contains two integers l_i and r_i $(1 \leq l_i \leq r_i \leq 10^9)$.

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

Output

For each test case output the minimum number of operations needed.

Example

input

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

output

```
1
2
2
5
```

Note

In the first test case, we can achieve the goal with a single operation: choose $v = 2$ and $c = [1, 2]$, resulting in $a_1 = 1, a_2 = 2$.

In the second test case, we can achieve the goal with two operations: first, choose $v = 2$ and $c = [3, 3]$, resulting in $a_1 = 3, a_2 = 3, a_3 = 0$. Then, choose $v = 3, c = [2, 7]$, resulting in $a_1 = 5, a_2 = 3, a_3 = 7$.

C. Keshi in Search of AmShZ

time limit per test: 2 seconds

memory limit per test: 256 megabytes

input: standard input

output: standard output

AmShZ has traveled to Italy from Iran for the Thom Yorke concert. There are n cities in Italy indexed from 1 to n and m **directed** roads indexed from 1 to m. Initially, Keshi is located in the city 1 and wants to go to AmShZ's house in the city n. Since Keshi doesn't know the map of Italy, AmShZ helps him to see each other as soon as possible.

In the beginning of each day, AmShZ can send one of the following two messages to Keshi:

- AmShZ sends the index of one road to Keshi as a **blocked** road. Then Keshi will understand that he should never use that road and he will remain in his current city for the day.
- AmShZ tells Keshi to move. Then, Keshi will randomly choose one of the cities reachable from his current city and move there. (city B is reachable from city A if there's an out-going road from city A to city B which hasn't become **blocked** yet). If there are no such cities, Keshi will remain in his current city.
  Note that AmShZ always knows Keshi's current location.

AmShZ and Keshi want to find the smallest possible integer d for which they can make sure that they will see each other after at most d days. Help them find d.

Input

The first line of the input contains two integers n and m $(2 \le n \le 2 \cdot 10^5, 1 \le m \le 2 \cdot 10^5)$ — the number of cities and roads correspondingly.

The i-th line of the following m lines contains two integers v_i and u_i $(1 \le v_i, u_i \le n, v_i \ne u_i)$, denoting a **directed** road going from city v_i to city u_i.

It is guaranteed that there is at least one route from city 1 to city n. Note that there may be more than one road between a pair of cities in each direction.

Output

Output the smallest possible integer d to make sure that AmShZ and Keshi will see each other after at most d days.

Examples

input

```
2 1
1 2
```

output

```
1
```

input

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

output

```
2
```

input

```
5 7
1 2
2 3
3 5
1 4
4 3
4 5
3 1
```

output

```
4
```

Note

In the first sample, it's enough for AmShZ to send the second type of message.

In the second sample, on the first day, AmShZ blocks the first road. So the only reachable city from city 1 will be city 4. Hence on the second day, AmShZ can tell Keshi to move and Keshi will arrive at AmShZ's house.

It's also possible for AmShZ to tell Keshi to move for two days.

D. Decinc Dividing

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

Let's call an array a of m integers $a_1, a_2, \ldots, a_m$ **Decinc** if a can be made increasing by removing a decreasing subsequence (possibly empty) from it.

- For example, if $a = [3, 2, 4, 1, 5]$, we can remove the decreasing subsequence $[a_1, a_4]$ from a and obtain $a = [2, 4, 5]$, which is increasing.

You are given a permutation p of numbers from 1 to n. Find the number of pairs of integers (l, r) with $1 \le l \le r \le n$ such that $p[l \ldots r]$ (the subarray of p from l to r) is a **Decinc** array.

Input

The first line contains a single integer n $(1 \le n \le 2 \cdot 10^5)$ — the size of p.

The second line contains n integers $p_1, p_2, \ldots, p_n$ $(1 \le p_i \le n$, all p_i are distinct) — elements of the permutation.

Output

Output the number of pairs of integers (l, r) such that $p[l \ldots r]$ (the subarray of p from l to r) is a **Decinc** array. $(1 \le l \le r \le n)$

Examples

input

```
3
2 3 1
```

output

```
6
```

input

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

output

```
19
```

input

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

output

```
39
```

Note

In the first sample, all subarrays are **Decinc**.

In the second sample, all subarrays except $p[1 \ldots 6]$ and $p[2 \ldots 6]$ are **Decinc**.

E. Outermost Maximums

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

Yeri has an array of $n + 2$ non-negative integers : $a_0, a_1, \ldots, a_n, a_{n+1}$.

We know that $a_0 = a_{n+1} = 0$.

She wants to make all the elements of a equal to zero in the minimum number of operations.

In one operation she can do one of the following:

- Choose the leftmost maximum element and change it to the maximum of the elements on its left.
- Choose the rightmost maximum element and change it to the maximum of the elements on its right.

Help her find the minimum number of operations needed to make all elements of a equal to zero.

Input

The first line contains a single integer n ($1 \le n \le 2 \cdot 10^5$).

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($0 \le a_i \le n$).

Output

Print a single integer — the minimum number of operations needed to make all elements of a equal to zero.

Examples

input

```
6
1 4 2 4 0 2
```

output

```
7
```

input

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

output

```
9
```

input

```
4
0 0 0 0
```

output

```
0
```

Note

In the first sample, you get $\langle 1, \underline{1}, 2, 4, 0, 2 \rangle$ by performing the first operation and $\langle 1, 4, 2, \underline{2}, 0, 2 \rangle$ by performing the second operation.

One way to achieve our goal is shown below. (The underlines show the last change.)
$\langle 1, 4, 2, 4, 0, 2 \rangle \to \langle 1, 4, 2, \underline{2}, 0, 2 \rangle \to \langle 1, \underline{1}, 2, 2, 0, 2 \rangle \to \langle 1, 1, 2, 2, 0, \underline{0} \rangle \to \langle 1, 1, 2, \underline{0}, 0, 0 \rangle \to \langle 1, 1, \underline{0}, 0, 0, 0 \rangle \to \langle \underline{0}, 1, 0, 0, 0, 0 \rangle \to \langle 0, \underline{0}, 0,$

In the third sample each element is already equal to zero so no operations are needed.

F. I Might Be Wrong

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

You are given a binary string S of length n indexed from 1 to n. You can perform the following operation any number of times (possibly zero):

- Choose two integers l and r ($1 \le l \le r \le n$). Let cnt_0 be the number of times `0` occurs in $S[l \ldots r]$ and cnt_1 be the number of times 1 occurs in $S[l \ldots r]$. You can pay $|cnt_0 - cnt_1| + 1$ coins and sort the $S[l \ldots r]$. (by $S[l \ldots r]$ we mean the substring of S starting at position l and ending at position r)

  For example if $S =$ `11001`, we can perform the operation on $S[2 \ldots 4]$, paying $|2-1|+1=2$ coins, and obtain $S =$ `10011` as a new string.

Find the minimum total number of coins required to sort S in increasing order.

Input

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

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

The second line of each test case contains a binary string S of n characters $S_1 S_2 \ldots S_n$. ($S_i =$ `0` or $S_i =$ 1 for each $1 \le i \le n$)

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

Output

For each test case, output the minimum total number of coins required to sort S in increasing order.

Example

input

```
7
1
1
2
10
3
101
4
1000
5
11010
6
110000
20
01000010001010011000
```

output

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

Note

In the first test case, S is already sorted.

In the second test case, it's enough to apply the operation with $l=1, r=2$.

In the third test case, it's enough to apply the operation with $l=1, r=2$.