Codeforces Round #520 (Div. 2)

A. A Prank

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

JATC and his friend Giraffe are currently in their room, solving some problems. Giraffe has written on the board an array a_1, a_2, ..., a_n of integers, such that $1 \leq a_1 < a_2 < \ldots < a_n \leq 10^3$, and then went to the bathroom.

JATC decided to prank his friend by erasing some **consecutive elements** in the array. Since he doesn't want for the prank to go too far, he will only erase in a way, such that Giraffe can still restore the array using the information from the remaining elements. Because Giraffe has created the array, he's **also aware** that it's an increasing array and all the elements are integers in the range $[1, 10^3]$.

JATC wonders what is the greatest number of elements he can erase?

Input
The first line of the input contains a single integer n ($1 \leq n \leq 100$) — the number of elements in the array.

The second line of the input contains n integers a_i ($1 \leq a_1 < a_2 < \cdots < a_n \leq 10^3$) — the array written by Giraffe.

Output
Print a single integer — the maximum number of consecutive elements in the array that JATC can erase.

If it is impossible to erase even a single element, print 0.

Examples

input
6 1 3 4 5 6 9
output
2

input
3 998 999 1000
output
2

input
5 1 2 3 4 5
output
4

Note
In the first example, JATC can erase the third and fourth elements, leaving the array $[1, 3, \_, \_, 6, 9]$. As you can see, there is only one way to fill in the blanks.

In the second example, JATC can erase the second and the third elements. The array will become $[998, \_, \_]$. Because all the elements are less than or equal to 1000, the array is still can be restored. Note, that he can't erase the first 2 elements.

In the third example, JATC can erase the first 4 elements. Since all the elements are greater than or equal to 1, Giraffe can still restore the array. Note, that he can't erase the last 4 elements.

B. Math

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

JATC's math teacher always gives the class some interesting math problems so that they don't get bored. Today the problem is as

follows. Given an integer n, you can perform the following operations zero or more times:

- `mul` x: multiplies n by x (where x is an arbitrary positive integer).
- `sqrt`: replaces n with $\sqrt{n}$ (to apply this operation, $\sqrt{n}$ must be an integer).

You can perform these operations as many times as you like. What is the minimum value of n, that can be achieved and what is the minimum number of operations, to achieve that minimum value?

Apparently, no one in the class knows the answer to this problem, maybe you can help them?

Input
The only line of the input contains a single integer n ($1 \leq n \leq 10^6$) — the initial number.

Output
Print two integers: the minimum integer n that can be achieved using the described operations and the minimum number of operations required.

Examples

input
20
output
10 2

input
5184
output
6 4

Note
In the first example, you can apply the operation `mul` 5 to get 100 and then `sqrt` to get 10.

In the second example, you can first apply `sqrt` to get 72, then `mul` 18 to get 1296 and finally two more `sqrt` and you get 6.

Note, that even if the initial value of n is less or equal 10^6, it can still become greater than 10^6 after applying one or more operations.

C. Banh-mi

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

JATC loves Banh-mi (a Vietnamese food). His affection for Banh-mi is so much that he always has it for breakfast. This morning, as usual, he buys a Banh-mi and decides to enjoy it in a special way.

First, he splits the Banh-mi into n parts, places them on a row and numbers them from 1 through n. For each part i, he defines the *deliciousness* of the part as $x_i \in \{0, 1\}$. JATC's going to eat those parts one by one. At each step, he chooses arbitrary remaining part and eats it. Suppose that part is the i-th part then his *enjoyment* of the Banh-mi will increase by x_i and the deliciousness of all the remaining parts will also increase by x_i. The initial enjoyment of JATC is equal to 0.

For example, suppose the deliciousness of 3 parts are $[0, 1, 0]$. If JATC eats the second part then his enjoyment will become 1 and the deliciousness of remaining parts will become $[1, \_, 1]$. Next, if he eats the first part then his enjoyment will become 2 and the remaining parts will become $[\_, \_, 2]$. After eating the last part, JATC's enjoyment will become 4.

However, JATC doesn't want to eat all the parts but to save some for later. He gives you q queries, each of them consisting of two integers l_i and r_i. For each query, you have to let him know what is the maximum enjoyment he can get if he eats all the parts with indices in the range $[l_i, r_i]$ in some order.

All the queries are independent of each other. Since the answer to the query could be very large, print it modulo $10^9 + 7$.

Input
The first line contains two integers n and q ($1 \leq n, q \leq 100\,000$).

The second line contains a string of n characters, each character is either '0' or '1'. The i-th character defines the deliciousness of the i-th part.

Each of the following q lines contains two integers l_i and r_i ($1 \leq l_i \leq r_i \leq n$) — the segment of the corresponding query.

Output
Print q lines, where i-th of them contains a single integer — the answer to the i-th query modulo $10^9 + 7$.

Examples

input

4 2 1011 1 4 3 4

output
14 3

input
3 2 111 1 2 3 3

output
3 1

Note
In the first example:

- For query 1: One of the best ways for JATC to eats those parts is in this order: 1, 4, 3, 2.
- For query 2: Both 3, 4 and 4, 3 ordering give the same answer.

In the second example, any order of eating parts leads to the same answer.

D. Fun with Integers

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

You are given a positive integer n greater or equal to 2. For every pair of integers a and b ($2 \leq |a|, |b| \leq n$), you can transform a into b if and only if there exists an integer x such that $1 < |x|$ and ($a \cdot x = b$ or $b \cdot x = a$), where $|x|$ denotes the absolute value of x.

After such a transformation, your score increases by $|x|$ points and you are **not allowed** to transform a into b nor b into a anymore.

Initially, you have a score of 0. You can start at any integer and transform it as many times as you like. What is the maximum score you can achieve?

Input
A single line contains a single integer n ($2 \leq n \leq 100\,000$) — the given integer described above.

Output
Print an only integer — the maximum score that can be achieved with the transformations. If it is not possible to perform even a single transformation for all possible starting integers, print 0.

Examples

input
4

output
8

input
6

output
28

input
2

output
0

Note
In the first example, the transformations are $2 \to 4 \to (-2) \to (-4) \to 2$.

In the third example, it is impossible to perform even a single transformation.

E. Company

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

The company X has n employees numbered from 1 through n. Each employee u has a direct boss p_u ($1 \le p_u \le n$), except for the employee 1 who has no boss. It is guaranteed, that values p_i form a tree. Employee u is said to be *in charge* of employee v if u is the direct boss of v or there is an employee w such that w is in charge of v and u is the direct boss of w. Also, any employee is considered to be in charge of himself.

In addition, for each employee u we define it's *level* $lv(u)$ as follow:

- $lv(1) = 0$
- $lv(u) = lv(p_u) + 1$ for $u \neq 1$

In the near future, there are q possible plans for the company to operate. The i-th plan consists of two integers l_i and r_i, meaning that all the employees in the range $[l_i, r_i]$, and only they, are involved in this plan. To operate the plan smoothly, there must be a project manager who is an employee in charge of **all** the involved employees. To be precise, if an employee u is chosen as the project manager for the i-th plan then for every employee $v \in [l_i, r_i]$, u must be in charge of v. Note, that u is not necessary in the range $[l_i, r_i]$. Also, u is always chosen in such a way that $lv(u)$ is as large as possible (the higher the level is, the lower the salary that the company has to pay the employee).

Before any plan is operated, the company has JATC take a look at their plans. After a glance, he tells the company that for every plan, it's possible to reduce the number of the involved employees **exactly** by one without affecting the plan. Being greedy, the company asks JATC which employee they should kick out of the plan so that the level of the project manager required is as large as possible. JATC has already figured out the answer and challenges you to do the same.

Input

The first line contains two integers n and q ($2 \le n \le 100\,000$, $1 \le q \le 100\,000$) — the number of employees and the number of plans, respectively.

The second line contains $n - 1$ integers $p_2, p_3, \ldots, p_n$ ($1 \le p_i \le n$) meaning p_i is the direct boss of employee i.

It is guaranteed, that values p_i form a directed tree with the root of 1.

Each of the following q lines contains two integers l_i and r_i ($1 \le l_i < r_i \le n$) — the range of the employees, involved in the corresponding plan.

Output

Print q lines, each containing two integers — the number of the employee which should be kicked from the corresponding plan and the maximum possible level of the project manager in that case.

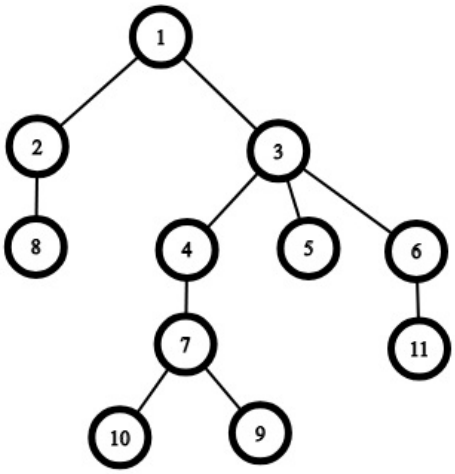
If there are more than one way to choose that employee, print any of them.

Example

input
11 5 1 1 3 3 3 4 2 7 7 6 4 6 4 8 1 11 9 11 8 11
output
4 1 8 1 1 0 11 3 8 1

Note

In the example:



In the first query, we can choose whether 4 or 5 or 6 and the project manager will be 3.
In the second query, if we choose any employee other than the employee 8, the project manager will be 1. If we choose 8, the project manager will be 3. Since $lv(3) = 1 > lv(1) = 0$, choosing 8 is the best strategy.

In the third query, no matter how we choose the employee, the project manager will always be 1.

In the fourth query, if we choose 9 or 10 then the project manager will be 3. If we choose 11 then the project manager will be 7. Since $lv(7) = 3 > lv(3) = 1$, we choose 11 as the answer.

F. Upgrading Cities

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

There are n cities in the kingdom X, numbered from 1 through n. People travel between cities by some **one-way** roads. As a passenger, JATC finds it weird that from any city u, he can't start a trip in it and then return back to it using the roads of the kingdom. That is, the kingdom can be viewed as an acyclic graph.

Being annoyed by the traveling system, JATC decides to meet the king and ask him to do something. In response, the king says that he will upgrade some cities to make it easier to travel. Because of the budget, the king will only upgrade those cities that are important or semi-important. A city u is called *important* if for every city $v \neq u$, there is either a path from u to v or a path from v to u. A city u is called *semi-important* if it is not important and we can destroy exactly one city $v \neq u$ so that u becomes important.

The king will start to act as soon as he finds out all those cities. Please help him to speed up the process.

Input
The first line of the input contains two integers n and m ($2 \leq n \leq 300\,000$, $1 \leq m \leq 300\,000$) — the number of cities and the number of one-way roads.

Next m lines describe the road system of the kingdom. Each of them contains two integers u_i and v_i ($1 \leq u_i, v_i \leq n$, $u_i \neq v_i$), denoting one-way road from u_i to v_i.

It is guaranteed, that the kingdoms' roads make an acyclic graph, which doesn't contain multiple edges and self-loops.

Output
Print a single integer — the number of cities that the king has to upgrade.

Examples

input
7 7 1 2 2 3 3 4 4 7 2 5 5 4 6 4
output
4

input
6 7 1 2 2 3

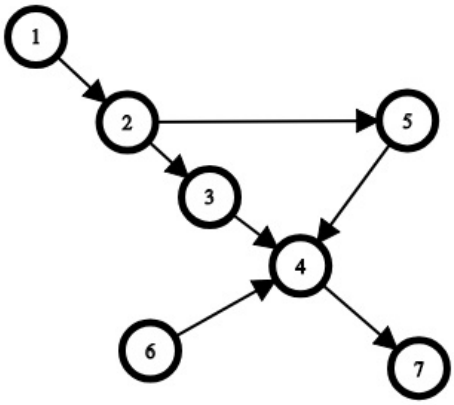
3 4
1 5
5 3
2 6
6 4

output

4

Note

In the first example:



- Starting at the city 1 we can reach all the other cities, except for the city 6. Also, from the city 6 we cannot reach the city 1. Therefore, if we destroy the city 6 then the city 1 will become important. So 1 is a semi-important city.
- For city 2, the set of cities that cannot reach 2 and cannot be reached by 2 is $\{6\}$. Therefore, destroying city 6 will make the city 2 important. So city 2 is also semi-important.
- For city 3, the set is $\{5, 6\}$. As you can see, destroying either city 5 or 6 will not make the city 3 important. Therefore, it is neither important nor semi-important.
- For city 4, the set is empty. So 4 is an important city.
- The set for city 5 is $\{3, 6\}$ and the set for city 6 is $\{3, 5\}$. Similarly to city 3, both of them are not important nor semi-important.
- The city 7 is important since we can reach it from all other cities.

So we have two important cities (4 and 7) and two semi-important cities (1 and 2).
In the second example, the important cities are 1 and 4. The semi-important cities are 2 and 3.