A. K-th Largest Value, 1 second, 256 megabytes, standard input, standard output

You are given an array a consisting of n integers. **Initially all elements of a are either 0 or 1**. You need to process q queries of two kinds:

- 1 x : Assign to a_x the value $1 - a_x$.
- 2 k : Print the k-th largest value of the array.

As a reminder, k-th largest value of the array b is defined as following:

- Sort the array in the non-increasing order, return k-th element from it.

For example, the second largest element in array $[0, 1, 0, 1]$ is 1, as after sorting in non-increasing order it becomes $[1, 1, 0, 0]$, and the second element in this array is equal to 1.

Input

The first line contains two integers n and q ($1 \le n, q \le 10^5$) — the length of the given array and the number of queries.

The second line contains n integers $a_1, a_2, a_3, \dots, a_n$ ($0 \le a_i \le 1$) — elements of the initial array.

Each of the following q lines contains two integers. The first integer is t ($1 \le t \le 2$) — the type of query.

- If $t = 1$ the second integer is x ($1 \le x \le n$) — the position of the modified number. You have to assign to a_x the value $1 - a_x$.
- If $t = 2$ the second integer is k ($1 \le k \le n$) — you need to print the k-th largest value of the array.

It's guaranteed that there will be **at least one** query of the second type (satisfying $t = 2$).

Output

For each query of the second type, print a single integer — the answer to the query.

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

output
```
1
0
1
0
```

Initially $a = [1, 1, 0, 1, 0]$.

The first operation is printing the third largest value, which is 1.

The second operation is assigning a_2 the value 0, a becomes $[1, 0, 0, 1, 0]$.

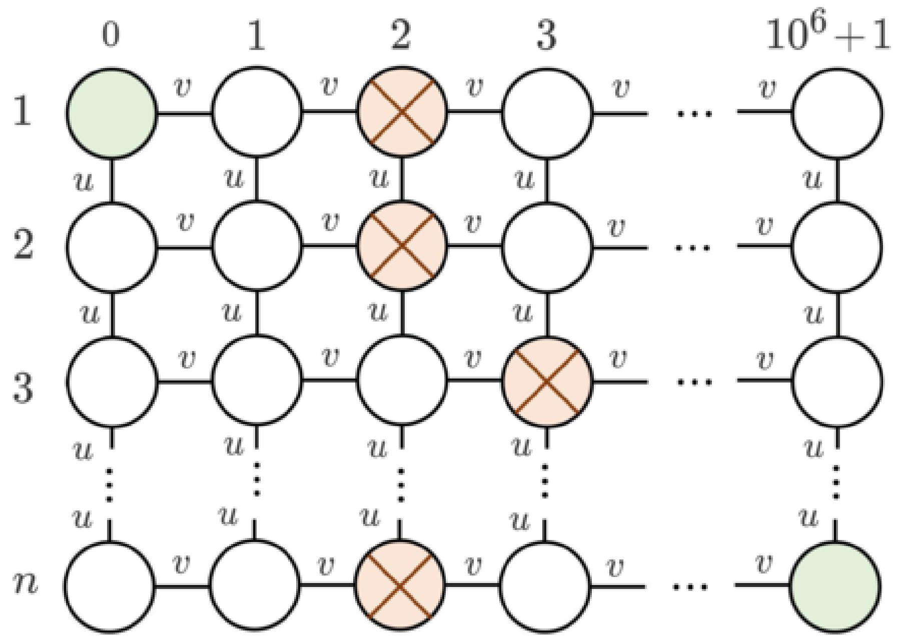
The third operation is printing the third largest value, it is 0.

The fourth operation is printing the first largest value, it is 1.

The last operation is printing the fifth largest value, it is 0.

B. Minimal Cost, 2 seconds, 256 megabytes, standard input, standard output

There is a graph of n rows and $10^6 + 2$ columns, where rows are numbered from 1 to n and columns from 0 to $10^6 + 1$:



Let's denote the node in the row i and column j by (i, j).

Initially for each i the i-th row has exactly one obstacle — at node (i, a_i). You want to move some obstacles so that you can reach node $(n, 10^6 + 1)$ from node $(1, 0)$ by moving through edges of this graph (you can't pass through obstacles). Moving one obstacle to an adjacent by edge free node costs u or v coins, as below:

- If there is an obstacle in the node (i, j), you can use u coins to move it to $(i - 1, j)$ or $(i + 1, j)$, if such node exists and if there is no obstacle in that node currently.
- If there is an obstacle in the node (i, j), you can use v coins to move it to $(i, j - 1)$ or $(i, j + 1)$, if such node exists and if there is no obstacle in that node currently.
- Note that you **can't move obstacles outside the grid**. For example, you can't move an obstacle from $(1, 1)$ to $(0, 1)$.

Refer to the picture above for a better understanding.

Now you need to calculate the minimal number of coins you need to spend to be able to reach node $(n, 10^6 + 1)$ from node $(1, 0)$ by moving through edges of this graph without passing through obstacles.

Input

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

The first line of each test case contains three integers n, u and v ($2 \le n \le 100$, $1 \le u, v \le 10^9$) — the number of rows in the graph and the numbers of coins needed to move vertically and horizontally respectively.

The second line of each test case contains n integers $a_1, a_2, \dots, a_n$ ($1 \le a_i \le 10^6$) — where a_i represents that the obstacle in the i-th row is in node (i, a_i).

It's guaranteed that the sum of n over all test cases doesn't exceed $2 \cdot 10^4$.

Output

For each test case, output a single integer — the minimal number of coins you need to spend to be able to reach node $(n, 10^6 + 1)$ from node $(1, 0)$ by moving through edges of this graph without passing through obstacles.

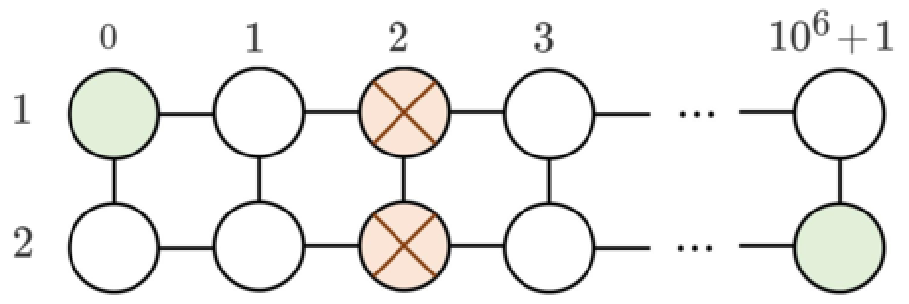
It can be shown that under the constraints of the problem there is always a way to make such a trip possible.

input
```
3
2 3 4
2 2
2 3 4
3 2
2 4 3
3 2
```

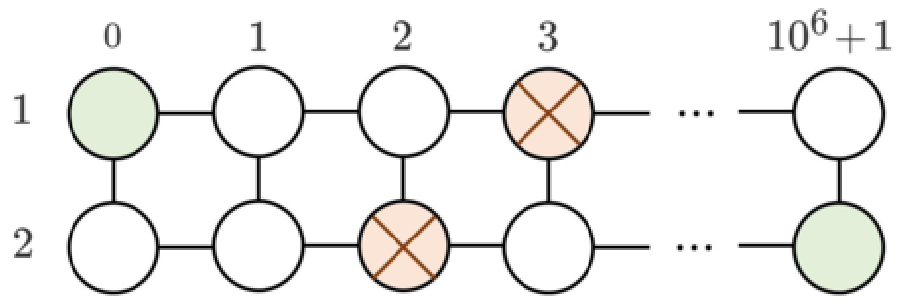
output

```
7
3
3
```

In the first sample, two obstacles are at $(1, 2)$ and $(2, 2)$. You can move the obstacle on $(2, 2)$ to $(2, 3)$, then to $(1, 3)$. The total cost is $u + v = 7$ coins.



In the second sample, two obstacles are at $(1, 3)$ and $(2, 2)$. You can move the obstacle on $(1, 3)$ to $(2, 3)$. The cost is $u = 3$ coins.



C. Pekora and Trampoline,

2 seconds, 256 megabytes, standard input, standard output

There is a trampoline park with n trampolines in a line. The i-th of which has strength S_i.

Pekora can jump on trampolines in multiple passes. She starts the pass by jumping on any trampoline of her choice.

If at the moment Pekora jumps on trampoline i, the trampoline will launch her to position $i + S_i$, and S_i will become equal to $\max(S_i - 1, 1)$. In other words, S_i will decrease by 1, except of the case $S_i = 1$, when S_i will remain equal to 1.

If there is no trampoline in position $i + S_i$, then this pass is over. Otherwise, Pekora will continue the pass by jumping from the trampoline at position $i + S_i$ by the same rule as above.

Pekora can't stop jumping during the pass until she lands at the position larger than n (in which there is no trampoline). Poor Pekora!

Pekora is a naughty rabbit and wants to ruin the trampoline park by reducing all S_i to 1. What is the minimum number of passes she needs to reduce all S_i to 1?

Input

The first line contains a single integer t ($1 \le t \le 500$) — the number of test cases.

The first line of each test case contains a single integer n ($1 \le n \le 5000$) — the number of trampolines.

The second line of each test case contains n integers $S_1, S_2, \ldots, S_n$ ($1 \le S_i \le 10^9$), where S_i is the strength of the i-th trampoline.

It's guaranteed that the sum of n over all test cases doesn't exceed 5000.

Output

For each test case, output a single integer — the minimum number of passes Pekora needs to do to reduce all S_i to 1.

input

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

output

```
4
3
0
```

For the first test case, here is an optimal series of passes Pekora can take. (The bolded numbers are the positions that Pekora jumps into during these passes.)

- $[1, 4, \mathbf{2}, 2, \mathbf{2}, 2, \mathbf{2}]$
- $[1, \mathbf{4}, 1, 2, 1, \mathbf{2}, 1]$
- $[1, \mathbf{3}, 1, 2, \mathbf{1}, \mathbf{1}, \mathbf{1}]$
- $[1, \mathbf{2}, 1, \mathbf{2}, 1, \mathbf{1}, \mathbf{1}]$

For the second test case, the optimal series of passes is show below.

- $[\mathbf{2}, 3]$
- $[1, \mathbf{3}]$
- $[1, \mathbf{2}]$

For the third test case, all S_i are already equal to 1.

D. Zookeeper and The Infinite Zoo,

3 seconds, 256 megabytes, standard input, standard output

There is a new attraction in Singapore Zoo: The Infinite Zoo.

The Infinite Zoo can be represented by a graph with an infinite number of vertices labeled $1, 2, 3, \ldots$. There is a directed edge from vertex u to vertex $u + v$ if and only if $u \& v = v$, where $\&$ denotes the bitwise AND operation. There are no other edges in the graph.

Zookeeper has q queries. In the i-th query she will ask you if she can travel from vertex u_i to vertex v_i by going through directed edges.

Input

The first line contains an integer q ($1 \le q \le 10^5$) — the number of queries.

The i-th of the next q lines will contain two integers u_i, v_i ($1 \le u_i, v_i < 2^{30}$) — a query made by Zookeeper.

Output

For the i-th of the q queries, output "YES" in a single line if Zookeeper can travel from vertex u_i to vertex v_i. Otherwise, output "NO".

You can print your answer in any case. For example, if the answer is "YES", then the output "Yes" or "yeS" will also be considered as correct answer.

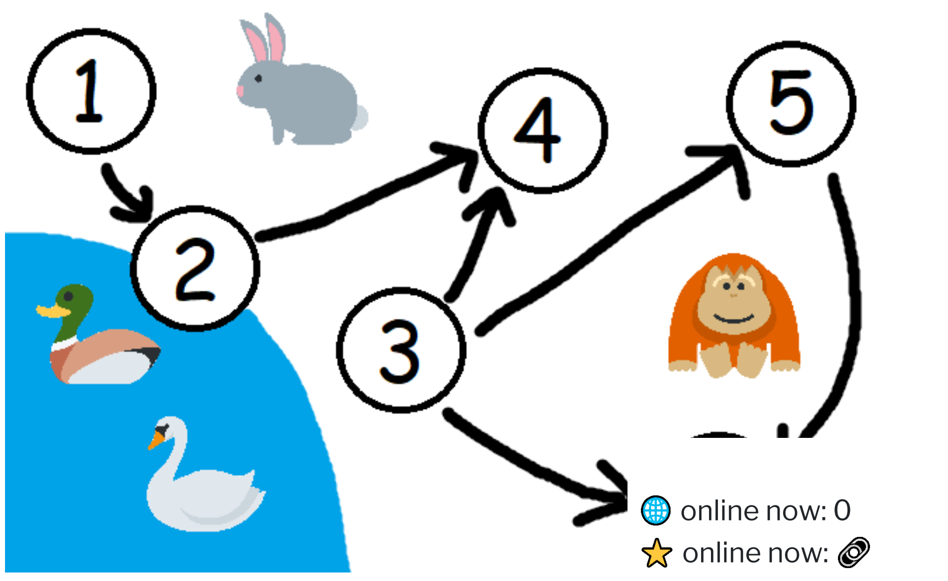
input

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

output

```
YES
YES
NO
NO
YES
```

The subgraph on vertices $1, 2, 3, 4, 5, 6$ is shown below.



online now: 0
online now:

E. Fib-tree, 1 second, 256 megabytes, standard input, standard output

Let F_k denote the k-th term of Fibonacci sequence, defined as below:

- $F_0 = F_1 = 1$
- for any integer $n \geq 0$, $F_{n+2} = F_{n+1} + F_n$

You are given a tree with n vertices. Recall that a tree is a connected undirected graph without cycles.

We call a tree a **Fib-tree**, if its number of vertices equals F_k for some k, and at least one of the following conditions holds:

- The tree consists of only 1 vertex;
- You can divide it into two Fib-trees by removing some edge of the tree.

Determine whether the given tree is a Fib-tree or not.

Input

The first line of the input contains a single integer n ($1 \leq n \leq 2 \cdot 10^5$) — the number of vertices in the tree.

Then $n - 1$ lines follow, each of which contains two integers u and v ($1 \leq u, v \leq n$, $u \neq v$), representing an edge between vertices u and v. It's guaranteed that given edges form a tree.

Output

Print "YES" if the given tree is a Fib-tree, or "NO" otherwise.

You can print your answer in any case. For example, if the answer is "YES", then the output "Yes" or "yeS" will also be considered as correct answer.

input
```
3
1 2
2 3
```
output
```
YES
```

input
```
5
1 2
1 3
1 4
1 5
```
output
```
NO
```

input
```
5
1 3
1 2
4 5
3 4
```
output
```
YES
```

In the first sample, we can cut the edge $(1, 2)$, and the tree will be split into 2 trees of sizes 1 and 2 correspondently. Any tree of size 2 is a Fib-tree, as it can be split into 2 trees of size 1.

In the second sample, no matter what edge we cut, the tree will be split into 2 trees of sizes 1 and 4. As 4 isn't F_k for any k, it's not Fib-tree.

In the third sample, here is one possible order of cutting the edges so that all the trees in the process are Fib-trees: $(1, 3), (1, 2), (4, 5), (3, 4)$.

F. Magnets, 1 second, 256 megabytes, standard input, standard output

This is an interactive problem.

Kochiya Sanae is playing with magnets. Realizing that some of those magnets are demagnetized, she is curious to find them out.

There are n magnets, which can be of the following 3 types:

- N
- S
- - — these magnets are demagnetized.

Note that **you don't know** the types of these magnets beforehand.

You have a machine which can measure the force between the magnets, and you can use it **at most** $n + \lfloor \log_2 n \rfloor$ times.

You can put some magnets to the left part of the machine and some to the right part of the machine, and launch the machine. Obviously, you can put one magnet to at most one side (you don't have to put all magnets). You can put the same magnet in different queries.

Then the machine will tell the force these magnets produce. Formally, let n_1, s_1 be the number of N and S magnets correspondently on the left and n_2, s_2 — on the right. Then the force between them would be $n_1 n_2 + s_1 s_2 - n_1 s_2 - n_2 s_1$. Please note that the force is a **signed** value.

However, when the **absolute** value of the force is **strictly larger than** n, the machine will crash into pieces.

You need to find **all** magnets of type - (all demagnetized ones), **without breaking the machine**.

Note that the interactor is not adaptive. The types of the magnets are fixed before the start of the interaction and do not change with queries.

It is guaranteed that there are **at least** 2 magnets whose type is not -, and **at least** 1 magnet of type -.

Input

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

Interaction

For each test case you should start by reading an integer n ($3 \leq n \leq 2000$) — the number of the magnets.

It is guaranteed that the total sum of all n over all test cases doesn't exceed 2000.

After that you can put some magnets into the machine and make a query from the statement.

You have to print each query in three lines:

- In the first line print "? l r" (without quotes) where l and r ($1 \leq l, r < n$, $l + r \leq n$) respectively denote the number of the magnets you put to left and right.
- In the second line print l integers $a_1, \dots, a_l$ ($1 \leq a_i \leq n$, $a_i \neq a_j$ if $i \neq j$) — the indices of the magnets you put to left.
- In the third line print r integers $b_1, \dots, b_r$ ($1 \leq b_i \leq n$, $b_i \neq b_j$ if $i \neq j$) — the indices of the magnets you put to right.
- The same magnet can't be put to both sides **in the same query**. Formally, you should guarantee that $a_i \neq b_j$ for any i and j. However, you may leave some magnets unused.

After printing a query do not forget to **output end of line and flush the output**. To do this, use:

- fflush(stdout) or cout.flush() in C++;
- System.out.flush() in Java;
- flush(output) in Pascal;
- stdout.flush() in Python;
- see documentation for other languages.

After this, you should read an integer F — the force these magnets produce.

Note that if your query is invalid(either the query limit exceeds, the machine crashes or the arguments are invalid), **the interactor will terminate immediately**. In this case terminate your program to receive verdict Wrong Answer instead of arbitrary verdicts.

If you are confident about your answer, use the f
report it:

- "! k A", where k is the number of magnets you found, and A is an array consisting of k different integers from 1 to n denoting the indices of the magnets of type - that you found. You may print elements of A in **arbitrary order**.
- After that, if this is the last test case, you have to terminate your program; otherwise you should immediately continue to deal with the next test case.

Note that the interactor is not adaptive. The types of the magnets are fixed before the start of interaction and do not change with queries.

Hacks

To hack a solution, use the following format:

The first line contains a single integer t ($1 \le t \le 100$) — the number of test cases in your hack.

Then follow the descriptions of the t test cases, each printed in two lines:

- The first line contains a single integer n ($3 \le n \le 2000$) — the number of magnets.
- The second line contains a string S of length n consisting of only N, S and -, denoting the magnets' types.
- Each of your test case should guarantee that there are **at least** 2 magnets whose type is not -, and **at least** 1 magnet of type -. Meanwhile, the total sum of n in all test cases should not exceed 2000.

input

```
1
4



0



1



0



0
```

output

```

? 1 1
3
4

? 1 2
1
2 3

? 1 1
1
4

? 1 1
1
3

! 2 3 4
```

The empty lines in the sample are just for you to better understand the interaction process. **You're not required to print them**.

In the sample, the types of the magnets are NN--.

At first, you put the third magnet on the left and the fourth one on the right. Both of them have type -, thus no force is produced.

Then you put the first magnet on the left and the second and third one on the right. The third magnet has type -, while the other two magnets are of type N, so the force produced is 1.

In the following two queries, the force is 0 since there is only a magnet with property - on the right.

Then we can determine that the magnets of type - are the third and the fourth one, so we should print ! 2 3 4 and exit.

G. Switch and Flip,

1 second, 256 megabytes, standard input, standard output

There are n coins labeled from 1 to n. Initially, coin c_i is on position i and is facing upwards (($c_1, c_2, \dots, c_n$) is a permutation of numbers from 1 to n). You can do some operations on these coins.

In one operation, you can do the following:

- Choose 2 distinct indices i and j.
- Then, swap the coins on positions i and j.
- Then, flip both coins on positions i and j. (If they are initially faced up, they will be faced down after the operation and vice versa)

Construct a sequence of at most $n+1$ operations such that after performing all these operations the coin i will be on position i at the end, **facing up**.

Note that you **do not need** to minimize the number of operations.

Input

The first line contains an integer n ($3 \le n \le 2 \cdot 10^5$) — the number of coins.

The second line contains n integers $c_1, c_2, \dots, c_n$ ($1 \le c_i \le n$, $c_i \ne c_j$ for $i \ne j$).

Output

In the first line, output an integer q $(0 \le q \le n+1)$ — the number of operations you used.

In the following q lines, output two integers i and j $(1 \le i, j \le n, i \ne j)$ — the positions you chose for the current operation.

input

```
3
2 1 3
```

output

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

input

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

output

```
0
```

Let coin i facing upwards be denoted as i and coin i facing downwards be denoted as $-i$.

The series of moves performed in the first sample changes the coins as such:

- $[\ \ 2,\ \ 1,\ \ 3]$
- $[-3,\ \ 1, -2]$
- $[-3,\ \ 2, -1]$
- $[\ \ 1,\ \ 2,\ \ 3]$

In the second sample, the coins are already in their correct positions so there is no need to swap.

H. Yuezheng Ling and Dynamic Tree,

1.5 seconds, 256 megabytes, standard input, standard output

Yuezheng Ling gives Luo Tianyi a tree which has n nodes, rooted at 1.

Luo Tianyi will tell you that the parent of the i-th
for $2 \le i \le n$), and she will ask you to perform

1. She'll give you three integers l, r and x ($2 \le l \le r \le n$, $1 \le x \le 10^5$). You need to replace a_i with $\max(a_i - x, 1)$ for all i with $l \le i \le r$.
2. She'll give you two integers u, v ($1 \le u, v \le n$). You need to find the LCA of nodes u and v (their lowest common ancestor).

Input

The first line contains two integers n and q ($2 \le n, q \le 10^5$) — the number of nodes and the number of queries, respectively.

The second line contains $n - 1$ integers $a_2, a_3, \dots, a_n$ ($1 \le a_i < i$), where a_i is the parent of the node i.

Next q lines contain queries. For each query, the first integer of each line is t ($t = 1$ or 2) — the type of the query.

If $t = 1$, this represents the query of the first type. Then, three integers will follow: l, r, x ($2 \le l \le r \le n, 1 \le x \le 10^5$), meaning that you have to replace a_i with $\max(a_i - x, 1)$ for all i with $l \le i \le r$.

If $t = 2$, this represents the query of the second type. Then, two integers will follow: u and v ($1 \le u, v \le n$), and you have to find the LCA of u and v.

It's guaranteed that there is at least one query of the second type.

Output

For each query of the second type output answer on a new line.

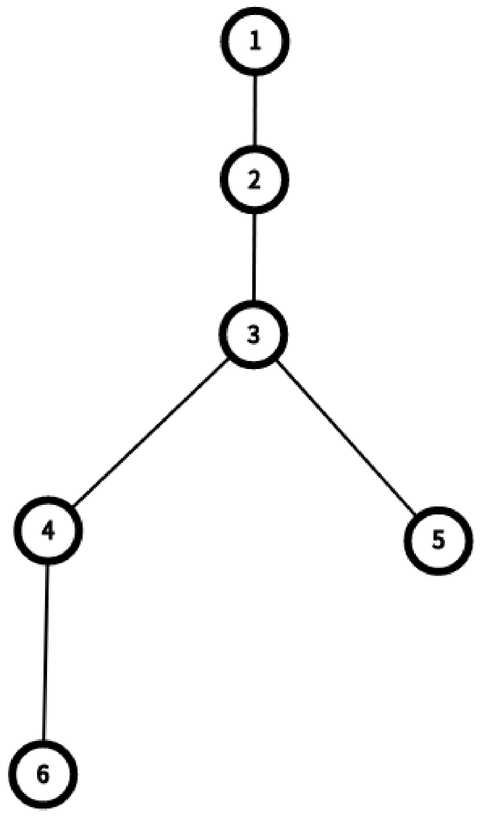
input

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

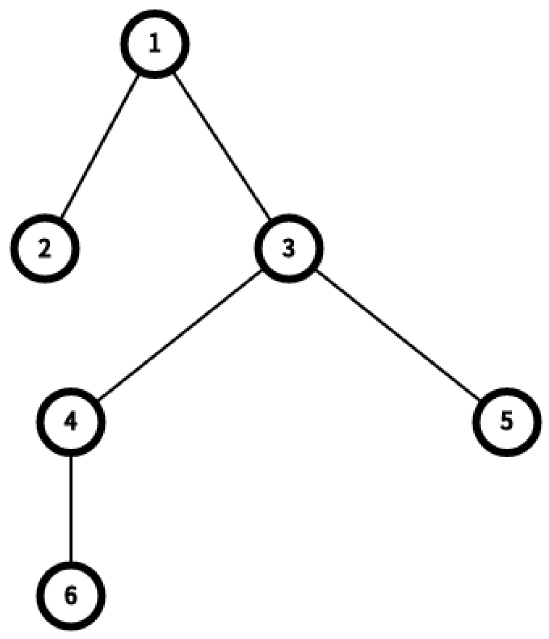
output

```
3
3
1
```

The tree in example is shown below.



After the query of the first type, the tree changes and is looking as shown below.



I. Ruler Of The Zoo, 3 seconds, 256 megabytes, standard input, standard output

After realizing that Zookeeper is just a duck, the animals have overthrown Zookeeper. They now have to decide a new ruler among themselves through a fighting tournament of the following format:

Initially, animal 0 is king, while everyone else queues up with animal 1 at the front of the queue and animal $n - 1$ at the back. The animal at the front of the queue will challenge the king to a fight, and the animal with greater strength will win the fight. The winner will become king, while the loser joins the back of the queue.

An animal who **wins 3 times consecutively** will be crowned ruler for the whole zoo. The strength of each animal depends on how many consecutive fights he won. Animal i has strength A_i with 0 consecutive win, B_i with 1 consecutive win, and C_i with 2 consecutive wins. Initially, everyone has 0 consecutive win.

For all animals, $A_i > B_i$ and $C_i > B_i$. Also, the values of A_i, B_i, C_i are **distinct** (all $3n$ values are pairwise different).

In other words, an animal who is not a king has strength A_i. A king usually has a strength of B_i or C_i. The exception is on the first turn, the first king (animal 0) has strength A_i.

Who is the new ruler, and after how many fights? Or will it end up that animals fight forever with no one ending up as ruler?

Input

The first line contains one integer n ($4 \le n \le 6000$) — number of the animals.

i-th of the next n lines contains 3 integers A_i, B_i and C_i ($0 \le A_i, B_i, C_i \le 10^9$).

It is guaranteed that $A_i > B_i$ and $C_i > B_i$, and that all values of A_i, B_i and C_i are distinct.

Output

Output two integers in a single line. The first is the index of the animal that will become ruler, and the second is the number of fights passed until some animal becomes the ruler.

If the animals will fight for infinitely long, output -1 -1 instead.

input

```
4
5 1 2
10 8 11
9 0 3
7 4 6
```

output

```
-1 -1
```

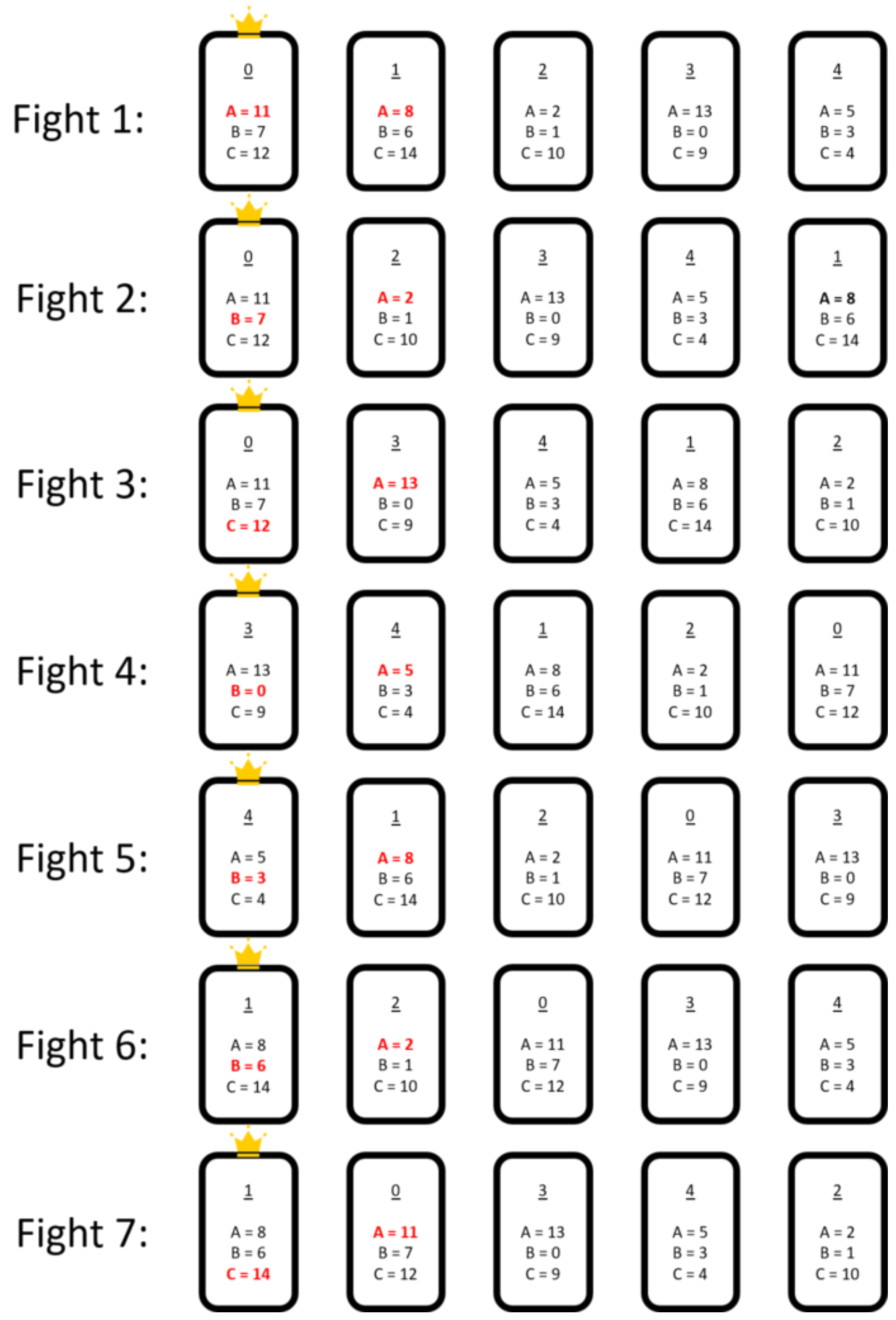
input

```
5
11 7 12
8 6 14
2 1 10
13 0 9
5 3 4
```

output

```
1 7
```

The following describes the sequence of events for the second sample. Note that in fight 1, the king (animal 0) has strength A_0. The tournament ends at fight 7 as animal 1 wins fight $5, 6$ and 7.


Fight 1:
0 A = 11 B = 7 C = 12
1 A = 8 B = 6 C = 14
2 A = 2 B = 1 C = 10
3 A = 13 B = 0 C = 9
4 A = 5 B = 3 C = 4
Fight 2:
0 A = 11 B = 7 C = 12
2 A = 2 B = 1 C = 10
3 A = 13 B = 0 C = 9
4 A = 5 B = 3 C = 4
1 A = 8 B = 6 C = 14
Fight 3:
0 A = 11 B = 7 C = 12
3 A = 13 B = 0 C = 9
4 A = 5 B = 3 C = 4
1 A = 8 B = 6 C = 14
2 A = 2 B = 1 C = 10
Fight 4:
3 A = 13 B = 0 C = 9
4 A = 5 B = 3 C = 4
1 A = 8 B = 6 C = 14
2 A = 2 B = 1 C = 10
0 A = 11 B = 7 C = 12
Fight 5:
4 A = 5 B = 3 C = 4
1 A = 8 B = 6 C = 14
2 A = 2 B = 1 C = 10
0 A = 11 B = 7 C = 12
3 A = 13 B = 0 C = 9
Fight 6:
1 A = 8 B = 6 C = 14
2 A = 2 B = 1 C = 10
0 A = 11 B = 7 C = 12
3 A = 13 B = 0 C = 9
4 A = 5 B = 3 C = 4
Fight 7:
1 A = 8 B = 6 C = 14
0 A = 11 B = 7 C = 12
3 A = 13 B = 0 C = 9
4 A = 5 B = 3 C = 4
2 A = 2 B = 1 C = 10