Codeforces Round 1020 (Div. 3)

A. Dr. TC

1 second, 256 megabytes

In order to test his patients' intelligence, Dr. TC created the following test.

First, he creates a binary string* s having n characters. Then, he creates n binary strings $a_1, a_2, \ldots, a_n$. It is known that a_i is created by first copying s, then flipping the i'th character (1 becomes 0 and vice versa). After creating all n strings, he arranges them into a grid where the i'th row is a_i.

For example,

- If $s = \texttt{101}$, $a = [\texttt{001}, \texttt{111}, \texttt{100}]$.
- If $s = \texttt{0000}$, $a = [\texttt{1000}, \texttt{0100}, \texttt{0010}, \texttt{0001}]$.

The patient needs to count the number of 1s written on the board in less than a second. Can you pass the test?

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

Input

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

The first line of each test case contains a single integer n ($1 \le n \le 10$) — the length of the binary string s.

The second line of each test case contains a single binary string s of size n.

Output

For each test case, output a single integer, the number of 1s on the board.

input
5 3 101 1 1 5 00000 2 11 3 010

output
5 0 5 2 4

The first example is explained in the statement.

For the second example, the only string written on the board will be the string 0; therefore, the answer is 0.

In the third example, the following strings will be written on the board: $[\texttt{10000}, \texttt{01000}, \texttt{00100}, \texttt{00010}, \texttt{00001}]$; so there are five 1s written on the board.

B. St. Chroma

2 seconds, 256 megabytes

Given a permutation* p of length n that contains every integer from 0 to $n-1$ and a strip of n cells, St. Chroma will paint the i-th cell of the strip in the color $\text{MEX}(p_1, p_2, \ldots, p_i)^\dagger$.

For example, suppose $p = [1, 0, 3, 2]$. Then, St. Chroma will paint the cells of the strip in the following way: $[0, 2, 2, 4]$.

You have been given two integers n and x. Because St. Chroma loves color x, construct a permutation p such that the number of cells in the strip that are painted color x is **maximized**.

*A permutation of length n is a sequence of n elements that contains every integer from 0 to $n-1$ exactly once. For example, $[0, 3, 1, 2]$ is a permutation, but $[1, 2, 0, 1]$ isn't since 1 appears twice, and $[1, 3, 2]$ isn't since 0 does not appear at all.

†The MEX of a sequence is defined as the first non-negative integer that does not appear in it. For example, $\text{MEX}(1, 3, 0, 2) = 4$, and $\text{MEX}(3, 1, 2) = 0$.

Input

The first line of the input contains a single integer t ($1 \le t \le 4000$) — the number of test cases.

The only line of each test case contains two integers n and x ($1 \le n \le 2 \cdot 10^5$, $0 \le x \le n$) — the number of cells and the color you want to maximize.

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

Output

Output a permutation p of length n such that the number of cells in the strip that are painted color x is **maximized**. If there exist multiple such permutations, output any of them.

input
7 4 2 4 0 5 0 1 1 3 3 1 0 4 3

output
1 0 3 2 2 3 1 0 3 2 4 1 0 0 0 2 1 0 1 2 0 3

The first example is explained in the statement. It can be shown that 2 is the maximum amount of cells that can be painted in color 2. Note that another correct answer would be the permutation $[0, 1, 3, 2]$.

In the second example, the permutation gives the coloring $[0, 0, 0, 4]$, so 3 cells are painted in color 0, which can be shown to be maximum.

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

C. Cherry Bomb

2 seconds, 256 megabytes

Two integer arrays a and b of size n are **complementary** if there exists an integer x such that $a_i + b_i = x$ over all $1 \le i \le n$. For example, the arrays $a = [2, 1, 4]$ and $b = [3, 4, 1]$ are complementary, since $a_i + b_i = 5$ over all $1 \le i \le 3$. However, the arrays $a = [1, 3]$ and $b = [2, 1]$ are not complementary.

Cow the Nerd thinks everybody is interested in math, so he gave Cherry Bomb two integer arrays a and b. It is known that a and b both contain n **non-negative** integers not greater then k.

Unfortunately, Cherry Bomb has lost some elements in b. Lost elements in b are denoted with -1. Help Cherry Bomb count the number of possible arrays b such that:

- a and b are **complementary**.
- All lost elements are replaced with non-negative integers no more than k.

Input

The first line of the input contains a single 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 k ($1 \le n \le 2 \cdot 10^5$, $0 \le k \le 10^9$) — the size of the arrays and the maximum possible value of their elements.

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

The third line contains n integers $b_1, b_2, \ldots, b_n$ ($-1 \le b_i \le k$). If $b_i = -1$, then this element is missing.

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

Output

Output exactly one integer, the number of ways Cherry Bomb can fill in the missing elements from b such that a and b are complementary.

input

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

output

```
1
0
0
7
0
1
0
```

In the first example, the only way to fill in the missing elements in b such that a and b are complementary is if $b = [2, 0, 1]$.

In the second example, there is no way to fill in the missing elements of b such that a and b are complementary.

In the fourth example, some b arrays that are complementary to a are: $[4, 2, 3, 0, 1]$, $[7, 5, 6, 3, 4]$, and $[9, 7, 8, 5, 6]$.

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

D. Flower Boy

2 seconds, 256 megabytes

Flower boy has a garden of n flowers that can be represented as an integer sequence $a_1, a_2, \ldots, a_n$, where a_i is the beauty of the i-th flower from the left.

Igor wants to collect exactly m flowers. To do so, he will walk the garden **from left to right** and choose whether to collect the flower at his current position. The i-th flower among ones he collects must have a beauty of **at least** b_i.

Igor noticed that it might be impossible to collect m flowers that satisfy his beauty requirements, so **before** he starts collecting flowers, he can pick any integer k and use his magic wand to grow a new flower with beauty k and place it **anywhere** in the garden (between two flowers, before the first flower, or after the last flower). Because his magic abilities are limited, he may do this **at most once**.

Output the **minimum** integer k Igor must pick when he performs the aforementioned operation to ensure that he can collect m flowers. If he can collect m flowers without using the operation, output 0. If it is impossible to collect m flowers despite using the operation, output -1.

Input

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

The first line of each test case contains exactly two integers n and m ($1 \le m \le n \le 2 \cdot 10^5$) — the number of flowers in the garden and the number of flowers Igor wants to collect, respectively.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$) — where a_i is the beauty of the i-th flower from the left in the garden.

The third line of each test case contains m integers $b_1, b_2, \ldots, b_m$ ($1 \le b_i \le 10^9$) — where b_i is the minimum beauty the i-th flower must have that Igor will collect.

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

Output

For each test case, on a separate line, output the minimum integer k Igor must pick when he performs the aforementioned operation to ensure that he can collect m flowers. If he can collect m flowers without using the operation, output 0. If it is impossible to collect m flowers despite using the operation, output -1.

input

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

output

```
6
3
7
0
-1
-1
7
```

In the first test case, suppose Igor grows a flower of beauty 6 and places it between the third and the fourth flowers. Then, the garden will look like the following: $[3, 5, 2, 6, 3, 3, 5, 8, 1, 2]$. Then, he can select the second, fourth, sixth, seventh, and eighth flowers with beauties $[5, 6, 3, 5, 8]$.

In the third test case, he can grow a flower of beauty 7 and place it before the first flower. The garden will look like the following: $[7, 4, 3, 5, 4, 3]$. Now, he can choose the first, second, and fourth flowers.

In the fourth test case, Igor does not need to use the operation, so the answer is 0.

In the sixth test case, no matter how Igor performs the operation, he cannot collect 3 flowers such that the i-th flower he collects has a beauty of at least b_i, therefore the answer is -1.

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

E. Wolf

4 seconds, 256 megabytes

Wolf has found n sheep with tastiness values $p_1, p_2, \ldots, p_n$ where p is a permutation*. Wolf wants to perform binary search on p to find the sheep with tastiness of k, but p may not necessarily be sorted. The success of binary search on the range $[l, r]$ for k is represented as $f(l, r, k)$, which is defined as follows:

If $l > r$, then $f(l, r, k)$ fails. Otherwise, let $m = \lfloor \frac{l+r}{2} \rfloor$, and:

- If $p_m = k$, then $f(l, r, k)$ is **successful**,
- If $p_m < k$, then $f(l, r, k) = f(m + 1, r, k)$,
- If $p_m > k$, then $f(l, r, k) = f(l, m - 1, k)$.

Cow the Nerd decides to help Wolf out. Cow the Nerd is given q queries, each consisting of three integers l, r, and k. Before the search begins, Cow the Nerd may choose a non-negative integer d, and d indices $1 \leq i_1 < i_2 < \ldots < i_d \leq n$ where $p_{i_j} \neq k$ over all $1 \leq j \leq d$. Then, he may re-order the elements $p_{i_1}, p_{i_2}, \ldots, p_{i_d}$ however he likes.

For each query, output the **minimum** integer d that Cow the Nerd must choose so that $f(l, r, k)$ can be **successful**, or report that it is impossible. Note that the queries are independent and the reordering is not actually performed.

*A permutation of length n is an array that contains every integer from 1 to n exactly once.

Input

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

The first line of each test contains three integers n and q $(1 \leq n, q \leq 2 \cdot 10^5)$ — the length of p and the number of queries respectively.

The second line contains n integers $p_1, p_2, \ldots, p_n$ — the tastiness of the i-th sheep. It is guaranteed that every integer from 1 to n appears exactly once in p.

The following q lines contain three integers l, r, and k $(1 \leq l \leq r \leq n, 1 \leq k \leq n)$ — the range that the binary search will be performed on and the integer being searched for each query.

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

Output

For each query, output the minimum integer d that Cow the Nerd must choose so that $f(l, r, k)$ is successful on a new line. If it is impossible, output -1.

input

```
8
5 3
1 2 3 4 5
1 5 4
1 3 4
3 4 4
7 4
3 1 5 2 7 6 4
3 4 2
2 3 5
1 5 6
1 7 3
2 1
2 1
1 2 1
1 1
1
1 1 1
7 1
3 4 2 5 7 1 6
1 7 1
16 1
16 10 12 6 13 9 14 3 8 11 15 2 7 1 5 4
1 16 4
16 1
14 1 3 15 4 5 6 16 7 8 9 10 11 12 13 2
1 16 14
13 1
12 13 10 9 8 4 11 5 7 6 2 1 3
1 13 2
```

output

```
0 -1 0
2 0 -1 4
-1
0
-1
-1
-1
-1
```

In the first example, second query: Since 4 does not exist in the first three elements, it is impossible to find it when searching for it in that range.

In the second example, on the first query, you may choose the indices 2, 3, and swap them so $p = [3, 5, 1, 2, 7, 6, 4]$. Then, $f(3, 4, 2)$ will work as follows:

1. Let $m = \lfloor \frac{3+4}{2} \rfloor = 3$. Because $p_3 = 1 < 2$, then $f(3, 4, 2) = f(4, 4, 2)$.
2. Let $m = \lfloor \frac{4+4}{2} \rfloor = 4$. Because $p_4 = 2 = k$, then $f(4, 4, 2)$ is successful. Therefore, $f(3, 4, 2)$ is also successful.

The total indices chosen were 2, so the final cost is 2, which can be shown that is minimum. Note that for this query we can't choose index 4, since $p_4 = k = 2$.

In the last query of the second example, we can choose indices $2, 3, 4, 5$ and re-arrange them so $p = [3, 5, 2, 7, 1, 6, 4]$. Then, $f(1, 7, 3)$ is successful.

F. Goblin

2 seconds, 256 megabytes

Dr. TC has a new patient called Goblin. He wants to test Goblin's intelligence, but he has gotten bored of his standard test. So, he decided to make it a bit harder.

First, he creates a binary string* s having n characters. Then, he creates n binary strings $a_1, a_2, \ldots, a_n$. It is known that a_i is created by first copying s, then flipping the i-th character (1 becomes 0 and vice versa). After creating all n strings, he arranges them into an $n \times n$ grid g where $g_{i,j} = a_{i_j}$.

A set S of size k containing distinct integer pairs $\{(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)\}$ is considered good if:

- $1 \leq x_i, y_i \leq n$ for all $1 \leq i \leq k$.
- $g_{x_i, y_i} = 0$ for all $1 \leq i \leq k$.
- For any two integers i and j ($1 \leq i, j \leq k$), coordinate (x_i, y_i) is reachable from coordinate (x_j, y_j) by traveling through a sequence of adjacent cells (which share a side) that all have a value of 0.

Goblin's task is to find the maximum possible size of a good set S. Because Dr. TC is generous, this time he gave him two seconds to find the answer instead of one. Goblin is not known for his honesty, so he has asked you to help him cheat.

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

Input

The first line of the input consists of a single integer t ($1 \leq t \leq 10^3$) — the number of test cases.

The first line of each test contains a single integer n ($1 \leq n \leq 2 \cdot 10^5$) — the length of the binary string s.

The second line of each test contains a single binary string s of length n.

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 a single number, the maximum possible size of a good set of cells from the grid.

input
6 3 000 4 0010 7 1011001 4 0001 2 11 1 0

output
3 9 10 7 1 0

In the first example, the following grid has been written on the board:

$$100$$
$$010$$
$$001$$

The set that consists of cells $(1, 2)$ and $(1, 3)$ is good. The set that consists of cells $(1, 1)$ and $(1, 2)$ is not good, since the value of cell $(1, 1)$ is not 0. The set of cells $(1, 2)$, $(1, 3)$, $(2, 3)$ is good and has a maximum size of 3. Note that the set of cells $(2, 1)$, $(3, 1)$, and $(3, 2)$ is also good with a maximum size of 3.

In the second example, the following grid is written on the board:

$$1010$$
$$0110$$
$$0000$$
$$0011$$

And the maximum possible size of a good set is 9.

G1. Baudelaire (easy version)

3 seconds, 256 megabytes

This is the easy version of the problem. The only difference between the two versions is that in this version, it is guaranteed that every node is adjacent to node 1.

This problem is interactive.

Baudelaire is very rich, so he bought a tree of size n that is rooted at some arbitrary node. Additionally, every node has a value of 1 or -1. **In this version, every node is adjacent to node 1. However, please note that node 1 is not necessarily the root.**

Cow the Nerd saw the tree and fell in love with it. However, computer science doesn't pay him enough, so he can't afford to buy it. Baudelaire decided to play a game with Cow the Nerd, and if he won, he would gift him the tree.

Cow the Nerd does not know which node is the root, and he doesn't know the values of the nodes either. However, he can ask Baudelaire queries of two types:

- $1\ k\ u_1\ u_2\ \ldots\ u_k$: Let $f(u)$ be the sum of the values of all nodes in the path from the root of the tree to node u. Cow the Nerd may choose an integer k ($1 \leq k \leq n$) and k nodes $u_1, u_2, \ldots, u_k$, and he will receive the value $f(u_1) + f(u_2) + \ldots + f(u_k)$.
- $2\ u$: Baudelaire will toggle the value of node u. Specifically, if the value of u is 1 it will become -1, and vice versa.

Cow the Nerd wins if he guesses the value of every node correctly (the values of the final tree, **after** performing the queries) within $n + 200$ total queries. Can you help him win?

Input

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

The first line of each test case contains a single integer n ($2 \leq n \leq 10^3$), the size of the tree.

Each of the next $n - 1$ lines contains two integers u and v ($1 \leq u, v \leq n, u \neq v$), denoting an edge between nodes u and v in the tree. **In this version, it is guaranteed that either $u = 1$ or $v = 1$.**

It is guaranteed that the sum of n over all test cases does not exceed 10^3 and that each graph provided is a valid tree.

Interaction

To ask a query of type 1, output a line in the following format (without the quotes):

- "`? 1 k` u_1 u_2 ... u_k", ($1 \leq k, u_i \leq n$)

The jury will return a single integer, $f(u_1) + f(u_2) + \ldots + f(u_k)$.

To ask a query of type 2, output a line in the following format:

- "`? 2` u" ($1 \leq u \leq n$)

The jury will toggle the value of node u: if its value is 1 it will become -1 and vice versa.

When you have found the answer, output a single line in the following format:

- "! $v_1, v_2, \ldots, v_n$" ($v_i = 1$ or $v_i = -1$, and v_i is the value of node i after performing the queries)

After that, proceed to process the next test case or terminate the program if it was the last test case. Printing the answer does not count as a query.

The interactor is **not** adaptive, meaning that the values of the tree are known before the participant asks the queries.

If your program makes more than $n + 200$ queries, your program should immediately terminate to receive the verdict `Wrong Answer`. Otherwise, you can get an arbitrary verdict because your solution will continue to read from a closed stream.

After printing a query do not forget to output the end of line and flush the output. Otherwise, you may get the `Idleness Limit Exceeded` verdict. 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 the documentation for other languages.

Hacks

For hacks, use the following format.

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

The first line of each test case must contain exactly two integers n and $root$ ($2 \le n \le 10^3, 1 \le root \le n$) — the size of the tree and the root of the tree.

The second line of each test case must contain exactly n integers $a_1, a_2, \ldots, a_n$ ($|a_i| = 1$) — where a_i is the value of node i.

Each of the following $n - 1$ lines must contain exactly two integers u and v ($1 \le u, v \le n$) — denoting an edge of the tree between nodes u and v.

The sum of n over all test cases must not exceed 10^3 and every graph provided must be a valid tree. **For this version, every node must be adjacent to node 1.**

input

```
1
4
1 2
3 1
1 4

0


-6
```

output

```
? 1 3 1 2 3

? 2 2
? 1 3 1 2 3

! -1 -1 -1 1
```

In the example, the root of the tree is node 2, and the values of the nodes are $[-1, 1, -1, 1]$. Therefore,
$f(1) = 0, f(2) = 1, f(3) = -1, f(4) = 1$.

First, we query about the sum $f(1) + f(2) + f(3)$, so we receive the value 0. Then, we toggle the value of node 2, and the values are: $[-1, -1, -1, 1]$. Therefore,
$f(1) = -2, f(2) = -1, f(3) = -3, f(4) = -1$, and
$f(1) + f(2) + f(3) = -6$.

Finally, we figure out that the values of the nodes are $[-1, -1, -1, 1]$, so we answer that.

Note that this is just an explanation of how the queries work, and it is not supposed to use any specific strategy to solve the problem.

G2. Baudelaire (hard version)

3 seconds, 256 megabytes

This is the Hard Version of the problem. The only difference between the two versions is that in the Hard Version the tree may be of any shape.

This problem is interactive.

Baudelaire is very rich, so he bought a tree of size n, rooted at some arbitrary node. Additionally, every node has a value of 1 or -1.

Cow the Nerd saw the tree and fell in love with it. However, computer science doesn't pay him enough, so he can't afford to buy it. Baudelaire decided to play a game with Cow the Nerd, and if he won, he would gift him the tree.

Cow the Nerd does not know which node is the root, and he doesn't know the values of the nodes either. However, he can ask Baudelaire queries of two types:

- $1\ k\ u_1\ u_2\ \ldots\ u_k$: Let $f(u)$ be the sum of the values of all nodes in the path from the root of the tree to node u. Cow the Nerd may choose an integer k ($1 \le k \le n$) and k nodes $u_1, u_2, \ldots, u_k$, and he will receive the value $f(u_1) + f(u_2) + \ldots + f(u_k)$.
- $2\ u$: Baudelaire will toggle the value of node u. Specifically, if the value of u is 1, it will become -1, and vice versa.

Cow the Nerd wins if he guesses the value of every node correctly (the values of the final tree, **after** performing the queries) within $n + 200$ total queries. Can you help him win?

Input

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

The first line of each test case contains a single integer n ($2 \le n \le 10^3$), the size of the tree.

Each of the next $n - 1$ lines contains two integers u and v ($1 \le u, v \le n, u \ne v$), denoting an edge between nodes u and v in the tree.

It is guaranteed that the sum of n over all test cases does not exceed 10^3 and that each graph provided is a valid tree.

Interaction

To ask a query of type 1, output a line in the following format (without the quotes):

- "`? 1 k` $u_1\ u_2\ \ldots\ u_k$", ($1 \le k, u_i \le n$)

The jury will return a single integer, $f(u_1) + f(u_2) + \ldots + f(u_k)$.

To ask a query of type 2, output a line in the following format:

- "`? 2` u" ($1 \le u \le n$)

The jury will toggle the value of node u: if its value is 1, it will become -1 and vice versa.

When you have found the answer, output a single line in the following format:

- "! $v_1, v_2, \ldots, v_n$" ($v_i = 1$ or $v_i = -1$, and v_i is the value of node i after performing the queries)

After that, proceed to process the next test case or terminate the program if it was the last test case. Printing the answer does not count as a query.

The interactor is **not** adaptive, meaning that the values of the tree are known before the participant asks the queries.

If your program makes more than $n + 200$ queries, your program should immediately terminate to receive the verdict `Wrong Answer`. Otherwise, you can get an arbitrary verdict because your solution will continue to read from a closed stream.

After printing a query do not forget to output the end of line and flush the output. Otherwise, you may get the `Idleness Limit Exceeded` verdict. 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 the documentation for other languages.

Hacks

For hacks, use the following format.

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

The first line of each test case must contain exactly two integers n and $root$ ($2 \le n \le 10^3, 1 \le root \le n$) — the size of the tree and the root of the tree.

The second line of each test case must contain exactly n integers $a_1, a_2, \ldots, a_n$ ($|a_i| = 1$) — where a_i is the value of node i.

Each of the following $n - 1$ lines must contain exactly two integers u and v ($1 \le u, v \le n$) — denoting an edge of the tree between nodes u and v.

The sum of n over all test cases must not exceed 10^3 and every graph provided must be a valid tree.

input

```
3
4
1 4
4 2
2 3

1

-1


-5

-5

2
1 2

2

7
1 2
2 7
7 3
7 4
7 5
7 6

-1
```

output

```
? 1 3 1 2 4

? 1 2 3 1

? 2 4
? 1 3 1 2 4

? 1 2 3 1

! -1 -1 -1 -1


? 1 1 1

! 1 1




? 1 1 1

! -1 1 1 1 1 1 -1
```

In the first example, the root of the tree is node 4 and the values are: $[-1, -1, -1, 1]$ (the i-th value is the value of node i).

Initially, $f(1) = 0, f(2) = 0, f(3) = -1, f(4) = 1$. Therefore, the answer to our first query is $f(1) + f(2) + f(4) = 1$, and of the second query is $f(3) + f(1) = -1$.

After toggling the value of node 4, the values are $[-1, -1, -1, -1]$. In addition, $f(1) = -2, f(2) = -2, f(3) = -3, f(4) = -1$. Therefore, $f(1) + f(2) + f(4) = -5$ and $f(3) + f(1) = -5$.

We answer that the final values of the nodes are $[-1, -1, -1, -1]$, which is correct. Notice that we report the values of the nodes **after** the changes, and not before.

In the second example, the root of the tree is 2 and the initial values are $[1, 1]$.

In the last example, the root of the tree is 1 and the initial values are $[-1, 1, 1, 1, 1, 1, -1]$.

Note that this is just an explanation of how the queries work, and it is not supposed to use any specific strategy to solve the problem.