C. Hundred-Mile Marathon

Time Limit: `5s`
Memory Limit: `1GB`

This is an interactive problem and only supports the C++ language.

In previous competitions, "Lightning Flea" and "Quantum Flea" ended in a dramatic tie, with both teams crossing the finish line simultaneously, making it impossible for high-speed cameras to determine the winner. To definitively decide the strongest team, the Flea King has decided to host an unprecedented "Hundred Mile Marathon"—a race that will span the entire city network of the Flea Kingdom!

The Flea Kingdom has n cities, numbered from 1 to n, with city 1 being the capital, Flealia. The connections between these cities form a tree.

Now, the Flea Kingdom plans to host a "Hundred-Mile Marathon" which can be described by two parameters (x, d), where x denotes the city where the marathon starts, and d represents the scale of the marathon. The ending point is always the capital.

For a marathon (x, d), all cities on the shortest path from city x to the capital will generate noise. Any city within a distance $\leq d$ from any noisy city (including itself) will have all its residents come to watch.

Each city has a certain number of fleas residing in it. Volt wants to know how many fleas will come to watch. However, this data is a national secret, so Volt cannot access it directly.

Specifically, the number of fleas in each city is encrypted and stored in an `info` type, which you can use addition operations.

Currently, the marathon plans are not fully finalized. You need to perform at most M_1 addition operations of the info type for preprocessing. Then, for each plan, compute the number of fleas that will come to watch using at most M_2 addition operations of the info type.

Implementation Details

You should include the header file `match.h` . You can add the following code at the beginning of your program:

```
#include "match.h"
```

The header file includes the following:

1. Defines the `info` data type corresponding to the encrypted information. Each `info` consumes 8 bytes of memory.
2. Defines `empty_info`, which is the encrypted result of 0.
3. Encapsulates addition for the `info` type. Specifically, you can use the following operator:
   `info operator + (info a, info b);`
4. Defines and implements a function `isempty(info a)` that determines whether an info decrypts to 0. This function returns true if and only if the decrypted value of `a` is 0.

You do not need to, and should not, implement the main function. Instead, you need to implement the following functions:

```
void init(int n, vector<int> fa, vector<info> a, int task_id)
```

`n` denotes the number of nodes in the tree.

`fa` is an array of length $n-1$, corresponding to the parents of nodes 2 to n.

`a` is an array of length n, corresponding to the encrypted number of fleas in cities 1 to n.

`task_id` indicates the Subtask number.

For a test point, `init` is called exactly once.

```
info query(int x, int d)
```

This function queries for the total number of fleas that will come to watch for a marathon. The parameters' meanings are detailed in the problem description.

Note: Uninitialized `info` variables do not have `empty_info` as their default value.

The header file implements the main function, meaning you can directly compile and run your program after including `match.h`. Additionally, your final program should not access standard input or output but is allowed to access stderr.

The final test uses a different implementation of the interactive library. Therefore, participants' solutions should not rely on the specific implementation of the interactive library and should not depend on the specific implementation of the info type in `match.h`.

Differences Between Provided and Evaluation `match.h` :

In the provided grader, uninitialized info variables default to `empty_info` , but this is **not** the case in the actual grader.

In the provided grader, the `info` type size is 4 bytes, whereas in the actual grader, it is 8 bytes.

During actual testing, if at least one of the two `info` operands is `empty_info` , the addition operation is not counted towards the operation limit.

Input Format

The first line contains six integers: $n, Q, id, seed, M_1, M_2$, representing the number of nodes in the tree, the number of queries, the Subtask number (0 for samples), the random seed (used for encryption in evaluation), and the two operation limits. **When using the provided `match.h` , ensure that $seed = 0$.**

The second line contains $n-1$ integers: $fa_2, fa_3, \ldots, fa_n$, representing the parents of nodes 2 to n. Ensure that $fa_i < i$.

The third line contains n integers: $a_1, a_2, \ldots, a_n$, representing the number of fleas in cities 1 to n. **When using the provided `match.h` , ensure that $a_i \in [0, 10^4]$.**

The next Q lines each contain two integers x, d, describing a query.

Output Format

If the number of preprocessing operations exceeds M_1, the program will output `wrong 1` and exit.

The provided interactive library will output $Q+1$ lines:

Lines 1 to Q each contain two integers, representing the answer to a query and the number of addition operations used. If the number of addition operations exceeds M_2, the program will output `wrong 2` and exit.

Line $Q+1$ outputs two numbers: the number of addition operations during preprocessing and the maximum number of addition operations in a single query.

Note: The evaluation interactive library will not output line $Q+1$.

Sample 0

Sample Input

```
5 4 0 0 100000 10
1 1 2 2
1 10 100 1000 10000
1 1
3 0
2 1
4 0
```

Sample Output

```
111
101
11111
1011
```

Note that line $Q + 1$ is not provided here.

Sample 1~5

See the download attachments.

Constraints

For all data, ensure that $1 \leq n \leq 3 \times 10^5$, $1 \leq Q \leq \min(3 \times 10^5, \lfloor 10^6 / M_2 \rfloor)$, $1 \leq x \leq n$, and $0 \leq d \leq n$.

Subtask	Points	$n \leq$	M_1	M_2	Special Properties
1	5	20	10^7	1	None
2	20	10^5	10^7	100	None
3	10	10^5	10^7	1	A
4	10	10^5	10^7	10	B
5	20	3×10^5	10^7	1	B

Subtask	Points	$n \leq$	M_1	M_2	Special Properties
6	15	3×10^5	10^7	3	None
7	20	3×10^5	7×10^6	1	None

Special Properties:

A: Ensures that the parent of node i is randomly selected from 1 to $i - 1$.

B: Ensures that $d \geq 100$.