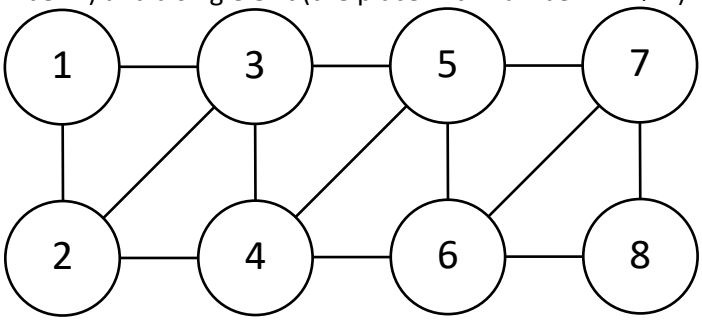
Vasi finally decided to visit Ruse and take a walk in its the park. From her tour guide, she knows that the park has a rectangular shape and its long sides are N units long. On each of those two sides, there are $N + 1$ interesting places (cafes, tennis courts, fountains and so on), so $2N + 2$ interesting places in total. They are numbered connected with alleys, like it is shown in the figure below ($N = 3$ in this case). The park has a single entrance (the place with number 1) and a single exit (the place with number $2N + 2$).



All alleys in the park are one-way and when taking a walk one must respect each alley's direction. The alleys are oriented in such a way that **there aren't any cycles**. Also, the entrance is the only place **where all alleys are out-going**, and the exit is the only place **where all alleys are in-going**. For all other places the following rule holds: **there are both in-going and out-going alleys**.

We will say that a particular alley has a positive direction, if it goes from a place with a lower number to a place with a higher one, otherwise we will say it has a negative direction. Vasi is aware that the two alleys from the entrance and the two alleys to the exit have a positive direction (since the entrance has number 1 and the exit – number $2N + 2$), but she doesn't know anything about the directions of the other alleys and she needs them to plan her walk in the park.

For the enthusiasts of mathematical and informatical problems at the park's entrance there is a computer with a special program on it. It answers questions of the following weird type: an arbitrary list of alleys, represented as the numbers of the places they connect, is given to the program and it returns the result of the XOR operation on their directions (where the positive direction will be represented with a 1 and the negative direction with a 0). *Let us remind you, that the XOR of two single-bit numbers is 1, if they are different, and 0, if they are the same. In case there are more than two operands, we first perform XOR on the first two, then to the result of that and the third operand and so on.* **The program is made in such a way that, if the given list only contains a single alley, it will just return its direction.**

Vasi want to find out the directions of all alleys with as few questions to the program as possible.

The jury has copied the program from the park's entrance and now it is uploaded to the grading system. Help Vasi by writing a function run, which is going to be compiled with the jury's program and will communicate with it by asking questions of the type described above and stating the directions it has found. After it is done running, your function will need to have correctly identified and stated the directions of all alleys in the park.

Implementation details

Your function `run` needs to have the following prototype:

```
void run(int n);
```

It will get called only once and will receive as an argument the whole positive number N – the length of the long side of the park (the total number of interesting places is $2N + 2$).

To communicate with the jury's program you can use the following two functions:

```
bool get_xor(const std::vector<std::pair<int, int>>& edges);
void state_direction(const std::pair<int, int>& edge, bool direction);
```

On each call of `get_xor` the result of the XOR operation of the directions of the alleys in the vector `edges`, is going to be returned. Each alley is represented as a pair of the numbers of the two places it connects (irrespective of order). Please note, that the complexity of the function `get_xor` (for both time and space) is linear on the number of alleys in the question.

Your function must call `state_direction` for each alley (again represented in the same way) with the found direction, which is stated with the parameter `direction`. If at any moment your program calls one of the two functions with an invalid alley or `state_direction` with the wrong direction, you will receive zero points for that test. You will also receive zero points for a test, if there are alleys, for which your function didn't call `state_direction`.

Send a file called **park.cpp** to the grading system. In it, besides the function `run`, you can have any sort of helping functions, classes, variables and so on. However, it must not contain a `main` function and it must include the header `park.h` with an instruction to the pre-processor `#include "park.h"` in its beginning.

Constraints

$1 \leq N \leq 100\,000$

Grading

Your solution will receive points, different from 0, if for each test it successfully performs the task in the given time limit and the number of questions asked by calling the function `get_xor` for each test is not above $3N$. The amount of points received will be as follows (Q – number of calls of `get_xor` for a single test; $\lceil x \rceil$ – the smallest whole number $\geq x$):

15 points : $Q \leq 3N$ for all tests, but $Q > 2N$ even for a single test.

45 points: $Q \leq 2N$ for all tests, but $Q > \left\lceil \frac{3}{2}N \right\rceil$ even for a single test.

70 points: $Q \leq \left\lceil \frac{3}{2}N \right\rceil$ for all tests, but $Q > \left\lceil \frac{16}{11}N \right\rceil$ even for a single test.

100 points: $Q \leq \left\lceil \frac{16}{11}N \right\rceil$ for all tests.

Local testing

You are given the file `Lgrader.cpp`, which you can compile together with your program to test it. When started it will ask you about N and after that about the directions of all alleys one after another (0 or 1). It will print out the communication which is happening. You can modify this file however you want.

Testing on the system and adaptive tests

On some particular tests (unknown to you) the jury's program will adapt to the questions your program asks, meaning the directions of the alleys in the park will turn out different depending on what queries are sent. If you test your program on the grading system, you will not have access to the adaptive option. The format of the input file must be the following: the natural number N, followed by $4N + 1$ zeroes and ones – the directions of the alleys one after another (1-2, 1-3, 2-3, 2-4, …).

Example communication

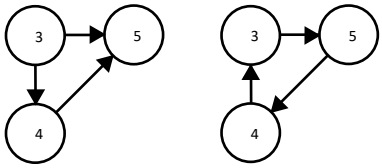
№	Actions of `run`	Actions and answers of the jury
1.		`run(2)`
2.	`state_direction({1,2}, 1)`	
3.	`state_direction({1,3}, 1)`	
4.	`state_direction({4,6}, 1)`	
5.	`state_direction({5,6}, 1)`	
6.	`get_xor({{3,4},{3,5},{4,5}})`	`1`
7.	`get_xor({{3,5}})`	`1`
8.	`state_direction({3,4}, 1)`	
9.	`state_direction({3,5}, 1)`	
10.	`state_direction({4,5}, 1)`	
11.	`get_xor({{2,3},{2,4}})`	`0`
12.	`state_direction({2,3}, 1)`	
13.	`state_direction({2,4}, 1)`	
14.	The function terminates.	

Explanation of the example communication

1. The jury's program calls `run` with $N = 2$.

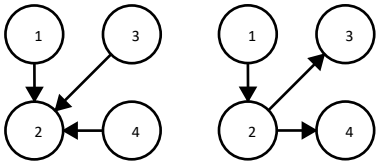
2-5. `Run` states the directions of the alleys from the entrance and to the exit.

6-7. `Run` asks about two XORs and gets 1 for both. Let's examine the two possibilities for the alleys:



8-10. The second option contains a cycle, but there aren't any cycles in the park, so `run` states the directions from the first option.

11. `Run` asks for the XOR of the directions of two alleys and gets 0. Therefore, their directions are the same:



12-13. In the first options all alleys from/to place 2 go in it, but there is only one such place in the park (the exit) and this is not it, so `run` states the directions from the second option.

14. `Run` terminates. 3 queries were used.