IOI Training Camp 2011 – Test 5, 15 June, 2011

Problem 3 Dommasandra-Varthur Bit Sequence Challenge [RunC]

It is time again for the annual Dommasandra-Varthur Bit Sequence Challenge! After months of rigorous training, both towns have selected their team of participants for the challenge. During the event, each of the two teams will be given a sequence of N integers, each of which is 0 or 1. In Dommasandra's sequence, strictly more than half of the entries will be 1. In Varthur's sequence, strictly less than half of the entries will be 1. The aim of the Bit Sequence Challenge is for the two teams to identify a position x ($0 \leq x \leq N-1$) such that entry at position x in Dommasandra's sequence is different from the entry at position x in Varthur's sequence.

The two teams accomplish this task by sending each other messages. Each message is a sequence of bits (where a bit is 0 or 1). The quality of the road connecting Dommasandra and Varthur has worsened, so not only do messages take a long time to reach, but the messages might even be modified on the way! As a saving grace, you know that each message sent either arrives at its destination *intact* or with *exactly one bit flipped* (changed from 0 to 1 or vice versa). For instance, a message containing the sequence 0010010 might arrive as 0110010.

Due to increasing complications in bureacratic processes, each bit is sent in a separate file folder, with a Bit Transfer form filled in triplicate, wrapped in layers of red tape, and enclosed in a package with a rubber seal. This makes the whole package quite heavy, and the poor road can't really take all this weight. To avoid rendering the road completely unusable, the organisers have fixed a limit on the total number of bits that can be sent back and forth.

At any point, either of the teams can declare that they have identified a number x such that the two sequences differ at position x.

Your task is to implement two program components, `dom` and `var`, representing Dommasandra and Varthur respectively.

- Dommasandra must implement the functions `dom_start(N, arr)` and `dom_turn(size, msg)`.
- Varthur must implement the functions `var_start(N, arr)` and `var_turn(size, msg)`.

In the beginning, the grader will call `dom_start` exactly once, supplying N and the sequence of 0s and 1s for Dommasandra. The same is true for `var_start`. Then `dom_turn` and `var_turn` will be called alternately, starting with `dom_turn`.

Each call to `dom_turn` and `var_turn` will have as its arguments an integer `size`, and an array `msg` of `size` elements. `msg` will be the message sent by the other team (with possibly one bit flipped). The first call to `dom_turn` will be made with `size` set to 0.

Inside `dom_turn` and `var_turn`, the corresponding team can build up the sequence of bits they plan to send as a message in a step by step manner, by repeated calls to `add_to_message`. At the beginning of each invocation of `dom_turn` and `var_turn`, the message is empty, and `add_to_message(y)` appends the integer y (which must be either 0 or 1) to the message. Finally, to actually send the message, `dom_turn` and `var_turn` need to return the value -1. The organisers of the Bit Sequence Challenge require that every message sent must have at least one bit, that is, sending an empty message is not allowed.

To declare that the required position x has been found, `dom_turn` or `var_turn` must return x where $0 \leq x \leq N-1$. This will ensure that no further calls are made and the program will be terminated.

In some cases , Dommasandra and Varthur are guaranteed that all messages will reach their destination intact. `dom` and `var` can find out if this is the case by calling `error_possible()`, which returns `true` if there is a possibility of messages being modified, and `false` if messages are guaranteed to be delivered intact.

Limits

In all subtasks, $1 \leq N \leq 100000$.

Subtask 1 [30 points]

At most 300 calls to `add_to_message` in total are allowed. `error_possible()` returns `false`.

Subtask 2 [20 points]

At most 600 calls to `add_to_message` in total are allowed. `error_possible()` returns `true`.

Subtask 3 [20 points]

At most 400 calls to `add_to_message` in total are allowed. `error_possible()` returns `true`.

Subtask 4 [30 points]

At most 300 calls to `add_to_message` in total are allowed. `error_possible()` returns `true`.

Time and memory limits

The time limit for this task is 2 seconds. The memory limit is 64MB.

Shared variables, file access and network access are prohibited.