# IOI Training Camp 2011 – Final 3, 22 June, 2011

## Problem 1 Balance Flip [RunC]

Alice and Bob are being held captive by the serial killer Red John. Their only hope for survival is to successfully play the following game proposed by Red John.

Red John has a sequence of $N$ coins numbered $1, 2, \ldots, N$, some of which are initially heads and some of which are tails. Only Alice can see Red John's coins: Bob cannot. Red John's game consists of performing an arbitrary sequence of actions of the following form.

- Flip coin $i$, $1 \leq i \leq N$.
- Ask Bob to *balance* coins $i, i+1, \ldots, j$, where $i \leq j$.

  When this question is posed to Bob, all coins in the sequence belong to a single collection, numbered 1. To balance the coins from $i$ to $j$, Bob has to partition them into two collections, 1 and 2, such that both collections have the same number of heads. To achieve this, Bob is allowed to perform a sequence of operations of the following form.

  - *Flip a sequence of coins within the segment $i$ to $j$ from heads to tails and vice versa.*
  - *Switch a sequence of coins within the segment $i$ to $j$ from collection 1 to collection 2 and vice versa.*

  Red John mentally works out the effect of Bob's operations. If he is satisfied that they would result in an equal number of heads in collections 1 and 2, he makes another move.

Since Bob can't see the coins, Red John allows Alice to send Bob some information about them. Before each challenge from Red John to Bob to balance coins $i$ to $j$, Red John tells Alice the values $i$ and $j$. Alice then sends Bob some information before Bob starts balancing the coins. Alice's information to Bob is sent through Red John. To ensure that Alice and Bob are worthy of survival, Red John limits the amount of information Alice can send Bob.

Your task is to implement program components `alice.cpp` and `bob.cpp`, representing Alice and Bob, respectively. They interact through `grader.cpp`, which represents Red John.

1. Initially, `grader.cpp` calls Alice's function `alice_init(N,S)` and Bob's function `bob_init(N)`. `S` is an array which gives the initial configuration of the coins: for $1 \leq i \leq N$, `S[i]` is `true` if coin $i$ is heads and `false` if coin $i$ is tails.
2. When Red John flips coin $i$, `grader.cpp` updates `alice.cpp` about the new configuration of the coins by calling the function `update(i)`.
3. A balance challenge to Bob proceeds as follows.
   - `grader.cpp` first calls `alice_balance(i,j)`, where $1 \leq i \leq j \leq N$.
   - Within the function `alice_balance`, Alice builds up the message to be sent to Bob as a sequence of bits by repeated calls to `send(b)`, where `b` is `true` or `false`.
   - Let $k \geq 0$ be the number of bits that Alice sent Red John to transmit to Bob. After the call to `alice_balance` ends, `grader.cpp` calls `bob_balance(i,j,msg,k)`. `msg` is a boolean array, starting with index 0, containing the $k$ bits Alice has sent to Red John, in the same order that she generated them.

- When `bob_balance(i,j,msg,k)` is called, all coins from $i$ to $j$ inclusive are put into collection 1. Bob then calls the following functions in any sequence to partition these coins into two balanced collections.
  - `flip(a,b)`, where $i \leq a \leq b \leq j$, to flip all the coins from $a$ to $b$ inclusive—this changes each coin in the interval from heads to tails and vice versa.
  - `switch_coll(a,b)`, where $i \leq a \leq b \leq j$, to toggle the collection to which each of the coins from $a$ to $b$ inclusive belongs—this moves each coin in the interval from collection 1 to collection 2 and vice versa

  The function `bob_balance` returns when Bob believes he has achieved the balanced partition that Red John requires.
- The grader checks that Bob's instructions, if executed, would result in two collections with an equal number of heads. Note that the actual configuration of the coins is not disturbed by Bob's `flip` and `switch_coll` operations.

## Interfaces

`grader.cpp` provides the following functions:

- `void send(bool m);` (Bob is not allowed to call this function)
- `void flip(int a, int b);` (Alice is not allowed to call this function)
- `void switch_coll(int a, int b);` (Alice is not allowed to call this function)

`alice.cpp` should provide the following functions:

- `void alice_init(int N, bool S[]);`
- `void update(int i);`
- `void alice_balance(int i, int j);`

`bob.cpp` should provide the following functions:

- `void bob_init(int N);`
- `void bob_balance(int i, int j, bool msg[], int k);`

## Test Data

*op* denotes the total number of update and balance operations.

- *Subtask 1 (10 marks)* : $1 \leq N \leq 1000$, $1 \leq op \leq 1000$, at most $10^6$ calls to `send` are allowed.
- *Subtask 2 (20 marks)* : $1 \leq N \leq 10000$, $1 \leq op \leq 1000$, at most 20000 calls to `send` are allowed.
- *Subtask 3 (70 marks)* : $1 \leq N \leq 10^5$, $1 \leq op \leq 10^5$, at most $3 \times 10^6$ calls to `send` are allowed.

## Limits

The time limit for this problem is 3 seconds. The memory limit is 128MB.
*The grader used for the evaluation will implement* `flip` *and* `switch_coll` *in* $O(\log N)$ *time.*