IOI Training Camp 2013 – Final 2

1 And how!

A boolean variable x can take two values, *True* and *False*, which we write as `tt` and `ff`, respectively. The function *and*, written $\wedge$, takes two boolean variables x and y as input and evaluates to `tt` if both inputs are `tt` and evaluates to `ff` otherwise. This is summarized by the following *truth table* that describes the behaviour of $x \wedge y$ for all possible combinations of x and y.

x	y	$x \wedge y$
ff	ff	ff
ff	tt	ff
tt	ff	ff
tt	tt	tt

It is easy to see $(x \wedge y) \wedge z = x \wedge (y \wedge z) = x \wedge y \wedge z$ evaluates to `tt` if and only if all of x, y and z are `tt`. In general, $x_0 \wedge x_1 \wedge \cdots \wedge x_n$ evaluates to `tt` if and only if all of $x_0, x_1, \ldots, x_n$ are `tt`.

You are given a hidden function $f(x_0, x_1, x_2, \ldots, x_{N-1})$ that computes the *and* of some subset of N boolean variables $x_0, x_1, \ldots, x_{N-1}$. In other words, $f(x_0, x_1, \ldots, x_{N-1}) = x_{j_1} \wedge x_{j_2} \wedge \cdots \wedge x_{j_K}$ for some $0 \leq j_1 < j_2 < \cdots < j_K \leq N-1$. Your goal is to identify the set of variables $x_{j_1}, x_{j_2}, \ldots, x_{j_K}$ that define f.

For this, you have access to a function *query()*. To use this function, you fix the value of each x_i as `tt` or `ff`. Let v_i denote the value assigned to x_i. The function *query*$(v_0, v_1, \ldots, v_{N-1})$ then reports whether $f(x_0, x_1, \ldots, x_{N-1})$ evaluates to `tt` or `ff` for the given assignment of values to the variables.

For example, suppose $N = 4$ and $f(x_0, x_1, x_2, x_3) = x_1 \wedge x_3$. Here are some possible calls to *query* with the corresponding answers.

- *query*(`tt`, `ff`, `tt`, `tt`) = `ff`, because $v_1 \neq$ `tt`.
- *query*(`tt`, `ff`, `tt`, `ff`) = `ff`, because $v_1 \neq$ `tt` and $v_3 \neq$ `tt`.
- *query*(`tt`, `tt`, `ff`, `tt`) = `tt`, because $v_1 = v_3 =$ `tt`.
- *query*(`ff`, `tt`, `ff`, `tt`) = `tt`, because $v_1 = v_3 =$ `tt`.

Your task

You have to write a function

```
void solve(int N, bool *ans)
```

that calls a function

```
bool query(const bool *q)
```

multiple times to compute the exact subset of variables that define the hidden function f.

- Your solution is returned by `solve()` through the array `ans[0..N-1]`, where `ans[i]` should be `true` if your computation reports variable x_i to be included in the definition of f, and `false` otherwise.

- You call `query` by passing a boolean array `q[0..N-1]`, where `q[0]`, `q[1]`, ..., `q[N-1]` are the values you choose for x_0, x_1, ..., x_{N-1}. The call to `query` returns the result of evaluating f with these values.
- Note that `ans` and `q` are declared as global arrays in the code provided to you (see below).

You will be provided two files.

- `grader.cpp`: You should not edit this file. This files reads a description of f from the input, calls `solve()` once and compares the array `ans` returned by `solve()` with f.

  `grader.cpp` also implements the function `query()` that reports the value of f for a given choice of inputs.
- `dummy_solution.cpp`: This is the file in which you write your code for `solve()`. There are some function headers, etc., that you should not modify or delete. The place where you have to insert your code is clearly marked.

Compiling and testing your code

To compile your code use the following command.

```
g++ grader.cpp dummy_solution.cpp -o grader
```

To provide test inputs to the code, see the section **Input format** below.

Submissions

You should submit your modified version of `dummy_solution.cpp` to the online judge, which will compile it with `grader.cpp`[1] and evaluate your implementation of `solve()`. You get full marks if your code is able to exactly identify the correct subset $x_{i_1}, x_{i_2}, \ldots, x_{i_K}$ for each candidate function f in less than or equal to 300 queries.

Input format

All input is done by `grader.cpp` so you do not need to write any code for input. However, to test your program, you can supply `grader.cpp` with the definition of a function f in the following format.

- The first contains two integers N and K, the total number of variables and the number of variables used in the definition of f, respectively.
- The second line contains K distinct integers in the range $0..N-1$, corresponding to the K variables from $x_0, x_1, \ldots, x_{N-1}$ that define f.

Output format

All output is done by `grader.cpp` so you do not need to write any code for output. Your implementation of `solve()` has to compute and return the array `ans[0..N-1]` as described above.

[1]Note: The exact implementation of `grader.cpp` in the online judge may vary from the code given to you, but the two will be functionally equivalent.

Test Data

- Subtask 1 (10 marks) : $N = 300$, $1 \leq K \leq 16$
- Subtask 2 (15 marks) : $N = 900$, $1 \leq K \leq 9$
- Subtask 3 (20 marks) : $N \leq 10^4$, $1 \leq K \leq 10$
- Subtask 4 (30 marks) : $N \leq 10^4$, $1 \leq K \leq 20$
- Subtask 5 (25 marks) : $N \leq 25,000$, $1 \leq K \leq 25$

Sample Input

$f(x_0, x_1, x_2, x_3) = x_1 \wedge x_3$

```
4 2
1 3
```

Sample Solution

```
ans[0] = false
ans[1] = true
ans[2] = false
ans[3] = true
```

Sample Interaction

```
q[0] = true; q[1] = false; q[2] = true; q[3] = true;
query(q) returns false

q[0] = true; q[1] = false; q[2] = true; q[3] = false;
query(q) returns false

q[0] = true; q[1] = true; q[2] = false; q[3] = true;
query(q) returns true

q[0] = false; q[1] = true; q[2] = false; q[3] = true;
query(q) returns true
```

Limits

- *Time limit:* 3 s
- *Memory limit:* 64 MB