# RUSSIA - KAZAN

#### **International Olympiad in Informatics 2016**

12-19th August 2016 Kazan, Russia day2 2

**messy** Country: SWE

## **Unscrambling a Messy Bug**

Ilshat is a software engineer working on efficient data structures. One day he invented a new data structure. This data structure can store a set of *non-negative* n -bit integers, where n is a power of two. That is,  $n=2^b$  for some non-negative integer b.

The data structure is initially empty. A program using the data structure has to follow the following rules:

- The program can add elements that are n -bit integers into the data structure, one at a time, by using the function  $add_element(x)$ . If the program tries to add an element that is already present in the data structure, nothing happens.
- After adding the last element the program should call the function compile\_set() exactly once.
- Finally, the program may call the function check\_element(x) to check whether
  the element x is present in the data structure. This function may be used
  multiple times.

When Ilshat first implemented this data structure, he made a bug in the function compile\_set(). The bug reorders the binary digits of each element in the set in the same manner. Ilshat wants you to find the exact reordering of digits caused by the bug.

Formally, consider a sequence  $p_0,\ldots,p_{n-1}$  in which every number from 0 to n-1 appears exactly once. We call such a sequence a *permutation*. Consider an element of the set, whose digits in binary are  $a_0,\ldots,a_{n-1}$  (with  $a_0$  being the most significant bit). When the function compile\_set() is called, this element is replaced by the element  $a_{p_0},a_{p_1},\ldots,a_{p_{n-1}}$ .

The same permutation p is used to reorder the digits of every element. The permutation may be arbitrary, including the possibility that  $p_i=i$  for each  $0\leq i\leq n-1$ .

For example, suppose that n=4, p=[2,1,3,0], and you have inserted into the set integers whose binary representations are 0000, 1100 and 0111. Calling the function <code>compile\_set</code> changes these elements to 0000, 0101 and 1110, respectively.

Your task is to write a program that finds the permutation p by interacting with the

data structure. It should (in the following order):

- 1. choose a set of n -bit integers,
- 2. insert those integers into the data structure,
- 3. call the function compile\_set to trigger the bug,
- 4. check the presence of some elements in the modified set,
- 5. use that information to determine and return the permutation p.

Note that your program may call the function compile set only once.

In addition, there is a limit on the number of times your program calls the library functions. Namely, it may

- call add element at most w times (w is for "writes"),
- call check element at most r times (r is for "reads").

### Implementation details

You should implement one function (method):

- o int[] restore permutation(int n, int w, int r)
  - n: the number of bits in the binary representation of each element of the set (and also the length of p).
  - w: the maximum number of add\_element operations your program can perform.
  - r: the maximum number of check\_element operations your program can perform.
  - $\circ$  the function should return the restored permutation p.

In the C language, the function prototype is a bit different:

- void restore permutation(int n, int w, int r, int\* result)
  - on, w and r have the same meaning as above.
    - the function should return the restored permutation p by storing it into the provided array result: for each i, it should store the value  $p_i$  into result[i].

#### **Library functions**

In order to interact with the data structure, your program should use the following three functions (methods):

void add element(string x)

This function adds the element described by x to the set.

- x: a string of '0' and '1' characters giving the binary representation of an integer that should be added to the set. The length of x must be n.
- void compile\_set()
   This function must be called exactly once. Your program cannot call add\_element() after this call. Your program cannot call check\_element() before this call.
- boolean check element(string x)

This function checks whether the element x is in the modified set.

- x: a string of '0' and '1' characters giving the binary representation of the element that should be checked. The length of x must be n.
- returns true if element x is in the modified set, and false otherwise.

Note that if your program violates any of the above restrictions, its grading outcome will be "Wrong Answer".

For all the strings, the first character gives the most significant bit of the corresponding integer.

Please use the provided template files for details of implementation in your programming language.

#### **Example**

The grader makes the following function call:

 $\circ$  restore\_permutation(4, 16, 16). We have n=4 and the program can do at most 16 "writes" and 16 "reads".

The program makes the following function calls:

```
add_element("0001")
add_element("0100")
compile_set()
check_element("0001") returns false
check_element("0010") returns true
check_element("0100") returns true
check_element("1000") returns false
check_element("0011") returns false
check_element("0011") returns false
check_element("1001") returns false
check_element("1001") returns false
check_element("1001") returns false
check_element("1010") returns false
check_element("1010") returns false
check_element("1010") returns false
```

Only one permutation is consistent with these values returned by check\_element(): the permutation p = [2, 1, 3, 0]. Thus, restore\_permutation should return [2, 1, 3, 0].

#### Subtasks

```
1. (20 points) n=8, w=256, r=256, p_i\neq i for at most 2 indices i ( 0\leq i\leq n-1 ),
2. (18 points) n=32, w=320, r=1024,
3. (11 points) n=32, w=1024, r=320,
4. (21 points) n=128, w=1792, r=1792,
```

5. (30 points) n=128 , w=896 , r=896 .

## Sample grader

The sample grader reads the input in the following format:

- $\circ$  line 1: integers n, w, r,
- $\circ$  line 2: n integers giving the elements of p.