International Olympiad in Informatics 2016
12-19th August 2016
Kazan, Russia
day2_2

messy
Country: CHN

Unscrambling a Messy Bug 解读Bug

伊尔沙特是一位软件工程师，他的工作是设计高效的数据结构。有一天，他发明了一个新的数据结构。这个数据结构可以存储一个 n 位非负整数集合， n 是2的整数次幂，即 $n = 2^b$ ， b 是非负整数。

这个数据结构初始为空。使用该数据结构的程序必须要遵守下列规则：

- 程序可以添加一些元素到这个数据结构中，每次利用函数 `add_element(x)` 添加一个元素，每个元素是一个 n 位整数。如果程序要添加的元素已经在数据结构中，则什么事情也不会发生。
- 当添加完最后一个元素以后，程序应该调用一次函数`complie_set( )`，而且只能调用一次。
- 最后，程序可以调用函数 `check_element(x)` 来检查元素 x 是否在数据结构中。这个函数可以调用多次。

当伊尔沙特第一次实现该数据结构时，他在写函数 `compile_set()` 时出现一个bug。这个bug将集合中每个元素的二进制位以相同的方式重新排序。伊尔沙特希望你能帮助他找到由于该bug导致的重排列。

考虑一个序列 $p = [p_0, \ldots, p_{n-1}]$ ，该序列中 0 到 $n-1$ 这n个数字每个数字恰好出现一次。我们称该序列为一个排列。考虑集合中的一个元素，该元素的二进制表达为 $a_0, \ldots, a_{n-1}$ （a_0 是最高位）。当函数`compile_set()`被调用时，这个元素将被元素 $a_{p_0}, a_{p_1}, \ldots, a_{p_{n-1}}$ 替代。

同样的排列 p 会被用于每个元素的二进制位的重排列。这个排列 p 可以是任意一个排列，包括 $p_i = i$， $0 \le i \le n-1$。

例如，假设 $n = 4$, $p = [2, 1, 3, 0]$, 你已经插入的整数所对应的二进制表示为 `0000`, `1100` 和 `0111`。调用函数 `compile_set` 会将三个元素分别变成 `0000`, `0101` 和 `1110`。

你的任务是写一个程序，该程序通过和数据结构的交互来找到排列 p。该程序应该（按照下列顺序）：

1. 选择一个 n 位整数的集合，
2. 将这些整数插入到数据结构中，
3. 调用函数 `compile_set` 来激活bug，
4. 检查某些元素是否在修改以后的集合当中，
5. 利用该信息来判断和返回排列 p。

注意你的程序只能调用函数 `compile_set` 一次。

而且，你的程序调用库函数的次数是有限制的。具体的，你的程序可以

- 调用 `add_element` 最多 w 次（ w 表示“写”）
- 调用 `check_element` 最多调用 r 次（ r 表示“读”）。

实现细节

你应该实现一个函数（方法）：

- `int[] restore_permutation(int n, int w, int r)`
  - `n`: 集合中每个元素的二进制表示的位数 (也是排列 p 的长度）。
  - `w`: 你的程序调用函数 `add_element` 的最大次数。
  - `r`: 你的程序调用函数 `check_element` 的最大次数。
  - 函数应该返回恢复的排列 p。

C语言的函数原型略有不同：

- `void restore_permutation(int n, int w, int r, int* result)`
  - `n`, `w` 和 `r` 同上。
  - 函数返回排列 p 的方式是将 p 存储到提供的数组`result`中：对每个 i，函数将 p_i 存到`result[i]`中。

库函数

为了和数据结构进行交互，你的程序应该使用下列三个函数（方法）

- `void add_element(string x)`
  该函数将 `x` 所描述的元素添加到集合中。
  - `x`: 一个由 `'0'` 和 `'1'` 构成的字符串，它是要添加到集合中的元素的二进制表示。`x`的长度必须是 n。

- `void compile_set()`
  该函数必须调用一次且只能调用一次。在调用该函数之后，你的程序不能再调用函数 `add_element()`。在调用该函数之前，你的程序也不能调用函数 `check_element()`。
- `boolean check_element(string x)`
  该函数检查元素 `x` 是否在修改以后的集合当中。
  - `x`: 一个由 `'0'` 和 `'1'` 构成的字符串，它是要检查的元素的二进制表示。`x` 的长度必须是 n。
  - 如果元素 `x` 在修改后的集合中，则返回 `true`，否则返回 `false`。

注意：如果你的程序违反上述的任何一条限制，其评分输出将是 "Wrong Answer"。

对于所有的字符串，第一个字符都表示所对应整数的最高位。

评测程序在调用函数 `restore_permutation` 之前已经确定了排列 p。

请使用提供的模板文件，以获得关于你所使用的编程语言的实现细节。

例子

评测程序执行下列函数调用：

- `restore_permutation(4, 16, 16)`. 我们有 $n = 4$ 而且程序最多执行 16 次"写"和 16 次"读"操作。

程序执行下列函数调用：

- `add_element("0001")`
- `add_element("0011")`
- `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("0101")` returns `false`
- `check_element("1001")` returns `false`
- `check_element("0110")` returns `false`
- `check_element("1010")` returns `true`
- `check_element("1100")` returns `false`

只有一个排列和函数 `check_element()` 返回的**值**一致：
排列 $p = [2, 1, 3, 0]$。因此，`restore_permutation` **应该**返回 `[2, 1, 3, 0]`.

子任务

1. (20分) $n = 8$，$w = 256$，$r = 256$，最多有**两**个下**标** i **满**足 $p_i \neq i$（$0 \leq i \leq n - 1$），
2. (18分) $n = 32$，$w = 320$，$r = 1024$，
3. (11分) $n = 32$，$w = 1024$，$r = 320$，
4. (21分) $n = 128$，$w = 1792$，$r = 1792$，
5. (30分) $n = 128$，$w = 896$，$r = 896$。

样例评测程序

样例**评测**程序按照以下格式**读**入**输**入：

- 第一行: 整数 n，w，r，
- 第二行: n 个整数表示排列 p 的元素。