Исследование странного бага

Ильшат работает программистом и занимается разработкой высокопроизводительных структур данных. Однажды он изобрел новую структуру данных. Эта структура данных хранит множество *неотрицательных* n-битных целых чисел, где n — степень числа 2. А именно, $n = 2^b$ для некоторого неотрицательного целого числа b.

Изначально структура данных пуста. Программа, использующая структуру данных, должна соблюдать следующие правила:

- Программа может добавлять в структуру данных элементы, которые являются n-битными целыми числами, используя функцию `add_element(x)`. Если программа пытается добавить в структуру данных число, которое в ней уже есть, то ничего не происходит.
- После добавления элементов программа должна вызывать функцию `compile_set()` ровно один раз.
- После этого программа может вызывать функцию `check_element(x)`, чтобы проверить, есть ли элемент x в структуре данных. Эту функцию можно вызывать несколько раз.

Когда Ильшат в первый раз реализовал эту структуру данных, он сделал баг в функции `compile_set()`. В результате бага после вызова этой функции двоичные цифры каждого элемента в структуре данных оказываются переставлены, причем цифры всех элементов переставляются одним и тем же образом. Ильшат хочет выяснить, как именно переставляются цифры.

Формально, рассмотрим последовательность $p_0, \ldots, p_{n-1}$, в которой каждое число от 0 до $n-1$ встречается ровно один раз. Будем называть такую последовательность *перестановкой*. Рассмотрим элемент в структуре данных, двоичная запись которого состоит из цифр $a_0, \ldots, a_{n-1}$ (здесь a_0 — старший бит). Когда вызывается функция `compile_set()`, этот элемент заменяется на элемент с двоичной записью $a_{p_0}, a_{p_1}, \ldots, a_{p_{n-1}}$.

Одна и та же перестановка p используется для изменения порядка двоичных цифр каждого из элементов. Перестановка может быть любой, в частности тождественной, то есть возможно, что $p_i = i$ для всех $0 \le i \le n-1$.

Например, пусть $n = 4$, $p = [2, 1, 3, 0]$, и программа добавила в структуру данных элементы с двоичными представлениями `0000`, `1100` и `0111`. Вызов функции `compile_set` заменит эти элементы на элементы `0000`, `0101` и `1110`, соответственно.

Требуется написать программу, которая находит перестановку p, взаимодействуя со структурой данных. Она должна выполнить по порядку следующие действия:

1. выбрать множество n-битных неотрицательных целых чисел,
2. добавить эти элементы в структуру данных,
3. вызвать функцию `compile_set`, чтобы активировать баг,
4. проверить наличие некоторых элементов в получившейся структуре данных,
5. используя полученную информацию, определить и вернуть перестановку p.

Обратите внимание, что разрешается вызвать функцию `compile_set` ровно один раз.

Также существуют ограничения на количество запросов к структуре данных, которые разрешается сделать вашей программе. А именно, она может:

- вызвать функцию `add_element` не более w раз (w от английского слова "write"),
- вызвать `check_element` не более r раз (r от английского слова "read").

Детали реализации

Требуется реализовать одну функцию (метод):

- `int[] restore_permutation(int n, int w, int r)`
  - `n`: число бит в двоичной записи каждого элемента в структуре данных (а также длина искомой перестановки p).
  - `w`: максимальное количество вызовов функции `add_element`, которое разрешается сделать.
  - `r`: максимальное количество вызовов функции `check_element`, которое разрешается сделать.
  - функция должна вернуть найденную перестановку p.

В языке C сигнатура функции немного отличается:

- `void restore_permutation(int n, int w, int r, int* result)`
  - `n`, `w` и `r` означают от же самое;
  - функция должна вернуть найденную перестановку p, занеся ее в массив `result`: для каждого i требуется поместить значение p_i в `result[i]`.

Функции библиотеки

Для взаимодействия со структурой данных, ваша программа должна использовать следующие три функции (метода):

- `void add_element(string x)`
  Добавить элемент, заданный строкой `x`, в структуру данных.
  - `x`: строка из `'0'` и `'1'`, задающая двоичное представление числа, который необходимо добавить в структуру данных. Длина строки `x` должна быть равна n.

- `void compile_set()`
  Эта функция должна быть вызвана ровно один раз. Ваша программа не

может вызывать функцию `add_element()` после этого вызова. Ваша программа не может вызывать `check_element()` до этого вызова.
- `boolean check_element(string x)`
  Эта функция проверяет, есть ли элемент, заданный строкой `x` в полученной после вызова `compile_set()` структуре данных.
  - `x`: строка из `'0'` и `'1'`, задающая двоичное представление числа, наличие которого в структуре данных необходимо проверить. Длина строки `x` должна быть равна n.
  - возвращает `true`, если элемент `x` есть в полученной структуре данных, либо `false` в противном случае.

Обратите внимание, что если ваша программа нарушит одно из перечисленных ограничений, то результат проверки будет "Wrong Answer".

Для всех строк, задающих двоичные числа, первый символ соответствует старшему биту числа.

Используйте приведенный шаблон решения для получения деталей реализации на выбранном вами языке программирования.

Пример

Пусть сделан вызов функции:
- `restore_permutation(4, 16, 16)`.

В этом случае $n = 4$ и программа может сделать не более 16 добавлений и не более 16 проверок, что элемент есть в структуре данных.

Пусть программа делает следующие вызовы библиотечных функций:
- `add_element("0001")`
- `add_element("0011")`
- `add_element("0100")`
- `compile_set()`
- `check_element("0001")` возвращает `false`
- `check_element("0010")` возвращает `true`
- `check_element("0100")` возвращает `true`
- `check_element("1000")` возвращает `false`
- `check_element("0011")` возвращает `false`
- `check_element("0101")` возвращает `false`
- `check_element("1001")` возвращает `false`
- `check_element("0110")` возвращает `false`
- `check_element("1010")` возвращает `true`
- `check_element("1100")` возвращает `false`

Существует единственная перестановка, при которой все сделанные вызовы `check_element()` могли вернуть соответствующие результаты: перестановка $p = [2, 1, 3, 0]$. Следовательно, `restore_permutation` должна вернуть массив `[2, 1, 3, 0]`.

Подзадачи

1. (20 баллов) $n = 8$, $w = 256$, $r = 256$, $p_i \neq i$ для не более чем 2 индексов i ($0 \leq i \leq n - 1$),
2. (18 баллов) $n = 32$, $w = 320$, $r = 1024$,
3. (11 баллов) $n = 32$, $w = 1024$, $r = 320$,
4. (21 балл) $n = 128$, $w = 1792$, $r = 1792$,
5. (30 баллов) $n = 128$, $w = 896$, $r = 896$.

Пример проверяющего модуля

Пример проверяющего модуля принимает данные в следующем формате:

- Строка 1: целые числа n, w, r.
- Строка 2: n целых чисел, задающих перестановку p.