Sửa lỗi hỗn độn (Unscrambling a Messy Bug)

Ilshat là kỹ sư phần mềm làm việc với các cấu trúc dữ liệu hiệu quả. Có lần anh thiết kế được một cấu trúc dữ liệu mới. Cấu trúc dữ liệu này có thể cất giữ một tập số nguyên n-bit *không âm*, trong đó n có dạng luỹ thừa của hai. Nghĩa là, $n = 2^b$ với một số nguyên không âm b nào đó.

Đầu tiên cấu trúc dữ liệu là rỗng. Chương trình sử dụng cấu trúc dữ liệu này cần phải tuân theo các qui tắc sau:

- Chương trình có thể bổ sung các phần tử là các số nguyên n-bit vào cấu trúc dữ liệu, mỗi lần một số, bằng việc sử dụng hàm `add_element(x)`. Nếu chương trình bổ sung một phần tử mà phần tử đó đã có mặt trong cấu trúc dữ liệu thì không có điều gì xảy ra.
- Sau khi bổ sung phần tử cuối cùng chương trình phải gọi hàm `compile_set()` đúng một lần.
- Cuối cùng, chương trình có thể gọi hàm `check_element(x)` để kiểm tra sự có mặt của phần tử x trong cấu trúc dữ liệu. Hàm này có thể gọi nhiều lần.

Khi lần đầu tiên cài đặt cấu trúc dữ liệu này, Ilshat đã mắc lỗi trong hàm `compile_set()`. Lỗi này đã sắp xếp lại thứ tự các bit nhị phân của mỗi phần tử trong tập theo cùng một cách. Ilshat muốn bạn tìm ra chính xác cách sắp xếp lại thứ tự các bít gây ra bởi lỗi này.

Một cách chính xác, xét dãy $p = [p_0, \ldots, p_{n-1}]$ trong đó mỗi số trong khoảng từ 0 đến $n-1$ xuất hiện đúng một lần. Ta gọi dãy như vậy là một *hoán vị*. Xét một phần tử của tập với các chữ số trong hệ đếm nhị phân là $a_0, \ldots, a_{n-1}$ (với a_0 là bit có trọng số lớn nhất). Khi hàm `compile_set()` được gọi thực hiện, phần tử này được thay thế bởi phần tử $a_{p_0}, a_{p_1}, \ldots, a_{p_{n-1}}$.

Cùng một hoán vị p được sử dụng để sắp xếp lại thứ tự các chữ số của mỗi phần tử. Hoán vị có thể là tuỳ ý, bao gồm cả khả năng $p_i = i$ với mỗi $0 \le i \le n-1$.

Chẳng hạn, giả sử $n = 4$, $p = [2, 1, 3, 0]$, và bạn đã nạp vào tập các số nguyên với biểu diễn nhị phân là `0000`, `1100` và `0111`. Việc gọi hàm `compile_set` sẽ biến đổi các phần tử này tương ứng thành `0000`, `0101` và `1110`.

Nhiệm vụ của bạn là viết chương trình để tìm ra hoán vị p bằng cách tương tác với cấu trúc dữ liệu. Chương trình phải thực hiện (theo thứ tự sau đây):

1. chọn tập các số nguyên n-bit,
2. bổ sung các số này vào cấu trúc dữ liệu,
3. gọi hàm `compile_set` để kích hoạt lỗi,
4. kiểm tra sự có mặt của một số phần tử trong tập đã được biến đổi,

5. sử dụng các thông tin thu được để xác định và trả về hoán vị p.

Chú ý là chương trình của bạn phải gọi hàm `compile_set` đúng một lần.

Ngoài ra, còn có hạn chế về số lần mà chương trình của bạn gọi các hàm thư viện. Cụ thể là

- gọi `add_element` nhiều nhất w lần (w thay thế cho "writes"),
- gọi `check_element` nhiều nhất r lần (r thay thế cho "reads").

Chi tiết cài đặt

Bạn cần cài đặt hàm (thủ tục):

- `int[] restore_permutation(int n, int w, int r)`
  - `n`: số lượng bit trong biểu diễn nhị phân của mỗi phần tử trong tập (và đồng thời cũng là độ dài của hoán vị p).
  - `w`: số lần lớn nhất thao tác `add_element` mà chương trình của bạn có thể thực hiện.
  - `r`: số lần lớn nhất thao tác `check_element` mà chương trình của bạn có thể thực hiện.
  - hàm phải trả lại hoán vị tìm được p.

Trong ngôn ngữ C, dạng hàm có khác một chút:

- `void restore_permutation(int n, int w, int r, int* result)`
  - `n`, `w` và `r` có ý nghĩa giống như ở trên.
  - hàm phải trả lại hoán vị tìm được p bằng cách cất nó vào mảng được cấp sẵn `result`: với mỗi i, hàm cất giá trị p_i vào `result[i]`.

Các hàm thư viện (Library functions)

Để tương tác với cấu trúc dữ liệu, chương trình của bạn phải sử dụng ba hàm sau đây:

- `void add_element(string x)`
  Hàm này bổ sung phần tử mô tả bởi `x` vào tập.
  - `x`: là xâu gồm các ký tự `'0'` và `'1'` cho biểu diễn của một số nguyên cần bổ sung vào tập. Độ dài của `x` phải là n.

- `void compile_set()`
  Hàm này phải được gọi thực hiện đúng một lần. Chương trình của bạn không thể gọi `add_element()` sau lệnh gọi này. Chương trình của bạn không thể gọi `check_element()` trước khi thực hiện lệnh gọi này.
- `boolean check_element(string x)`
  Hàm này kiểm tra phần tử `x` có mặt trong tập biến đổi hay không.
  - `x`: xâu gồm các ký tự `'0'` và `'1'` cho biểu diễn nhị phân của một phần tử cần kiểm tra. Độ dài của xâu `x` phải là n.
  - trả lại `true` nếu phần tử `x` có mặt trong tập biến đổi, và `false` nếu trái lại.

Chú ý là nếu chương trình của bạn vi phạm bất cứ hạn chế nào ở trên, kết quả chấm sẽ là "Wrong Answer".

Với tất cả các xâu, ký tự đầu tiên cho bít có trọng số lớn nhất của số nguyên tương ứng.

Trình chấm cố định một hoán vị p trước khi hàm `restore_permutation` được gọi.

Hãy sử dụng các file mẫu cho trước trong phần cài đặt theo ngôn ngữ bạn lựa chọn.

Ví dụ

Trình chấm thực hiện các lệnh gọi hàm sau:

- `restore_permutation(4, 16, 16)`. Ta có $n = 4$ và chương trình có thể thực hiện nhiều nhất là 16 "writes" và 16 "reads".

Chương trình thực hiện các lệnh gọi hàm sau:

- `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`

Có duy nhất một hoán vị thích hợp với các giá trị được trả lại bởi `check_element()` ở trên: hoán vị $p = [2, 1, 3, 0]$. Do đó, `restore_permutation` phải trả lại `[2, 1, 3, 0]`.

Subtasks

1. (20 points) $n = 8$, $w = 256$, $r = 256$, $p_i \neq i$ với nhiều nhất 2 chỉ số 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

Trình chấm mẫu sẽ đọc dữ liệu theo khuôn dạng:

- dòng 1: các số nguyên n, w, r,
- dòng 2: n các số nguyên cho các thành phần của p.