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

messy
Country: IDN

Mengembalikan sebuah Kesalahan Kacau

Ilshat adalah seorang software engineer yang bekerja dalam data struktur yang efisien. Suatu hari ia menemukan sebuah data struktur baru. Data struktur ini dapat menyimpan sebuah himpunan bilangan integer *non-negatif* n-bit, yang mana n adalah sebuah bilangan dua pangkat. Dengan kata lain, $n = 2^b$ untuk suatu bilangan integer non-negatif b.

Data struktur tersebut mula-mula kosong. Sebuah program menggunakan data struktur tersebut harus mengikuti aturan-aturan berikut:

- Program dapat menambahkan bilangan integer n-bit ke dalam data struktur, satu persatu, dengan menggunakan fungsi `add_element(x)`. Jika program mencoba menambahkan sebuah elemen yang telah ada di dalam data struktur, tidak akan ada yang terjadi.
- Setelah menambahkan elemen terakhir program harus memanggil fungsi `compile_set()` tepat sekali.
- Pada akhirnya, program boleh memanggil fungsi `check_element(x)` untuk menguji apakah elemen x ada dalam data struktur. Fungsi ini dapat digunakan beberapa kali.

Ketika Ilshat mengimplementasi data struktur ini untuk pertama kali, dia melakukan sebuah kesalahan dalam fungsi `compile_set()`. Kesalahannya menyusun ulang digit binary untuk tiap elemen pada himpunan dengan cara yang sama. Ilshat menginginkan Anda untuk mencari susunan ulang digit yang tepat yang disebabkan oleh kesalahan tersebut.

Secara resmi, pertimbangkan sebuah urutan $p = [p_0, \ldots, p_{n-1}]$ yang mana setiap angka dari 0 sampai $n-1$ muncul tepat sekali. Kita menyebut urutan tersebut adalah sebuah *permutasi*. Pertimbangkan sebuah elemen dari himpunan, dimana digit-digitnya dalam binary adalah $a_0, \ldots, a_{n-1}$ (dengan a_0 adalah bit yang paling signifikan). Ketika fungsi `compile_set()` dipanggil, elemen ini diganti dengan elemen $a_{p_0}, a_{p_1}, \ldots, a_{p_{n-1}}$.

Permutasi p yang sama digunakan untuk menyusun ulang digit-digit dari semua elemen. Semua permutasi mungkin, termasuk kemungkinan yang mana $p_i = i$ untuk setiap $0 \le i \le n-1$.

Sebagai contoh, anggap bahwa $n = 4$, $p = [2, 1, 3, 0]$, dan Anda telah menambahkan ke himpunan bilangan integer yang memiliki representasi binary `0000`, `1100` dan `0111`. Memanggil fungsi `compile_set` mengganti elemen-elemen ini menjadi `0000`, `0101` and `1110`, secara berurutan.

Tugas Anda adalah menuliskan sebuah program yang mencari permutasi p dengan berinteraksi dengan data struktur. Program tersebut harus (dengan urutan berikut):

1. memilih sebuah himpunan bilangan integer n-bit,
2. menambahkan bilangan-bilangan integer tersebut ke data struktur,
3. memanggil fungsi `compile_set` untuk mencetuskan kesalahan,
4. menguji adanya beberapa elemen dari himpunan yang telah diubah,
5. menggunakan informasi tersebut untuk menentukan dan mengembalikan permutasi p.

Perhatikan bahwa program Anda boleh memanggil fungsi `compile_set` hanya sekali.

Sebagai tambahan, terdapat sebuah batasan dari banyaknya panggilan program Anda ke fungsi library. Yaitu, program Anda dapat

- memanggil `add_element` paling banyak w kali (w adalah untuk "writes"),
- memanggil `check_element` paling banyak r kali (r adalah untuk "reads").

Rincian implementasi

Anda harus mengimplementasikan fungsi (method):

- `int[] restore_permutation(int n, int w, int r)`
  - `n`: banyaknya bit pada representasi binary dari tiap elemen pada himpunan (dan juga panjang dari p).
  - `w`: banyaknya operasi `add_element` maksimum yang program Anda dapat lakukan.
  - `r`: banyaknya operasi `check_element` maksimum yang program Anda dapat lakukan.
  - fungsi harus mengembalikan permutasi p.

Fungsi library

Untuk berinteraksi dengan data struktur, program Anda harus menggunakan tiga fungsi (methods) berikut ini:

- `void add_element(string x)`
  Fungsi ini menambahkan elemen yang dideskripsikan oleh `x` ke dalam himpunan.
  - `x`: sebuah string berisi karakter `'0'` dan `'1'` memberikan representasi binary dari sebuah bilangan integer yang akan ditambahkan pada himpunan. Panjang dari `x` harus n.

- `void compile_set()`
  Fungsi ini harus dipanggil tepat sekali. Program Anda tidak dapat memanggil `add_element()` setelah pemanggilan ini. Program Anda tidak dapat memanggil `check_element()` sebelum pemanggilan ini.
- `boolean check_element(string x)`
  Fungsi ini menguji apakah elemen `x` ada dalam himpunan yang telah diubah.
  - `x`: sebuah string berisi karakter `'0'` dan `'1'` memberikan representasi binary dari elemen yang akan diuji. Panjang dari `x` harus n.
  - mengembalikan `true` jika elemen `x` ada dalam himpunan yang telah diubah, dan `false` jika tidak.

Perhatikan bahwa jika program Anda melanggar salah satu dari batasan di atas, hasil grading akan "Wrong Answer".

Untuk setiap string, karakter pertama memberikan bit paling signifikan untuk bilangan integer yang sesuai.

Grader menetapkan permutasi p sebelum fungsi `restore_permutation` dipanggil.

Gunakan file template yang sudah disediakan untuk implementasi rinci dari bahasa pemrograman yang Anda pakai.

Contoh

Grader melakukan pemanggilan fungsi berikut:

- `restore_permutation(4, 16, 16)`. Kita memiliki $n = 4$ dan program dapat melakukan paling banyak 16 "writes" dan 16 "reads".

Program melakukan pemanggilan fungsi berikut:

- `add_element("0001")`
- `add_element("0011")`
- `add_element("0100")`
- `compile_set()`
- `check_element("0001")` mengembalikan `false`
- `check_element("0010")` mengembalikan `true`
- `check_element("0100")` mengembalikan `true`
- `check_element("1000")` mengembalikan `false`
- `check_element("0011")` mengembalikan `false`
- `check_element("0101")` mengembalikan `false`
- `check_element("1001")` mengembalikan `false`
- `check_element("0110")` mengembalikan `false`
- `check_element("1010")` mengembalikan `true`
- `check_element("1100")` mengembalikan `false`

Hanya satu permutasi yang konsisten dengan nilai-nilai yang dikembalikan oleh `check_element()`: permutasi $p = [2, 1, 3, 0]$. Jadi, `restore_permutation` harus mengembalikan `[2, 1, 3, 0]`.

Subtasks

1. (20 poin) $n = 8$, $w = 256$, $r = 256$, $p_i \neq i$ untuk paling banyak 2 indeks i ($0 \leq i \leq n - 1$),
2. (18 poin) $n = 32$, $w = 320$, $r = 1024$,
3. (11 poin) $n = 32$, $w = 1024$, $r = 320$,
4. (21 poin) $n = 128$, $w = 1792$, $r = 1792$,
5. (30 poin) $n = 128$, $w = 896$, $r = 896$.

Grader

Grader membaca masukan dengan format berikut:

- baris 1: bilangan integer n, w, r,
- baris 2: n bilangan integer memberikan elemen-elemen dari p.