messy
Country: MKD

Откривање на баг

Владе е софтверски инженер кој работи на ефикасни податочни структури. Тој измислил една нова податочна структура. Во оваа податочна структура може да се чува множество од *ненегативни* n-битни цели броеви, каде n е степен на два, т.е. $n = 2^b$ за некој ненегативен цел број b.

Една програма која ја користи оваа податочна структура мора да ги исполнува следните правила:

- На почетокот податочната структура е празна. Во програмата може да се додаваат елементи (n-битни ненегативни цели броеви) во податочната струкура. Користејќи ја функцијата `add_element(x)` елементите се додаваат еден по еден. Ако во програмата се направи обид да се додаде елемент кој веќе го има во податочната структура - нема да се случи ништо.
- По додавање на последниот елемент, во програмата треба да се повика функцијата `compile_set()` точно еднаш.
- На крајот, во програмата може да се повикува функцијата `check_element(x)` за да се провери дали елементот x го има во податочната структура. Оваа функција може да се користи повеќе пати.

Кога Владе прво ја имплементирал оваа податочна структура, тој направил баг во функцијата `compile_set()`. Со овој баг на ист начин е сменет редоследот на бинарните цифри на секој елемент во множеството (податочната структура). Владе сака да го најде точниот редослед на цифрите кој е променет со багот.

Формално, нека $p = [p_0, \ldots, p_{n-1}]$ е дадена низа во која секој број од 0 до $n-1$ се појавува точно еднаш. Оваа низа се нарекува *пермутација*. Нека $a_0, \ldots, a_{n-1}$ се бинарните цифри на еден елемент од множеството (a_0 е најзначајниот бит). Откако ќе се повика функцијата `compile_set()` овој елемент ќе се замени со елемент чии бинарни цифри се $a_{p_0}, a_{p_1}, \ldots, a_{p_{n-1}}$.

Истата пермутација p се користи за да се смени редоследот на секој елемент од множеството. Секоја пермутација е можна, вклучувајќи го и случајот кога $p_i = i$ за секое i, $0 \leq i \leq n-1$.

На пример, нека $n = 4$, $p = [2, 1, 3, 0]$, и нека во множеството се додадени целите броеви чии бинарни репрезентации се `0000`, `1100` и `0111`. По повикување на функцијата `compile_set` елементите ќе се променат во `0000`, `0101` и `1110`, соодветно.

Ваша задача е да напишете програма која ја наоѓа пермутацијата p преку интеракција со податочната структура. Во програмата треба да го направите

следното (во даденият редослед):

1. Да изберете множество од n-битни ненегативни цели броеви,
2. Да ги додадете избраните броеви во податочната структура,
3. Да ја повикате функција `compile_set` за да се изврши кодот со баг,
4. Да проверите дали некои ненегативни броеви ги има во модифицираното множество елементи,
5. Да ја искористите таа информација за да ја одредите и да ја вратите пермутацијата p.

Да забележиме дека вашата програма смее да ја повика функцијата `compile_set` точно еднаш.
Уште повеќе, постои ограничување на тоа колку пати програмата ги повикува функциите од библиотеката. Имено, во програмата може

- да се повика функцијата `add_element` најмногу w пати (најмногу w "запишувања"),
- да се повика функцијата `check_element` најмногу r пати (најмногу r "читања").

Детали за имплементација

Вие треба да имплементирате една функција (метод):

- `int[] restore_permutation(int n, int w, int r)`
  - `n`: бројот на битови во бинарната репрезентација на секој елемент од множеството (т.е. должината на p).
  - `w`: максималниот број на повици на функцијата `add_element` во вашата програма.
  - `r`: максималниот број на повици на функцијата `check_element` во вашата програма.
  - функцијата треба да ја врати најдената пермутација p.

Во јазикот C, потписот на функцијата е малку различен:

- `void restore_permutation(int n, int w, int r, int* result)`
  - `n`, `w` и `r` го имаат истото значење како погоре.
  - функцијата треба да ја зачува најдената пермутација p во низата `result`: за секое i, вредноста p_i треба да се зачува во `result[i]`.

Функции од библиотеката

За интеракција со податочната структура, вашата програма треба да ги користи следните три функции (методи):

- `void add_element(string x)`
  Оваа функција го додава елементот `x` во множеството.
  - `x`: стринг од знаците `'0'` и `'1'` со кои е дадена бинарната репрезентација на ненегативниот цел број кој треба да се додаде во множеството. Должината на `x` мора да е n.

- `void compile_set()`
  Оваа функција мора да се повика точно еднаш. По повикот на оваа функција, во вашата програма не може да се повика функцијата `add_element()`. Пред повикот на оваа функција, во вашата програма не може да се повика

функцијата `check_element()`.

- `boolean check_element(string x)`
  Оваа функција проверува дали елементот `x` го има во модифицираното множество.
  - `x`: стринг од знаците `'0'` и `'1'` со кои е дадена бинарната репрезентација на елементот кој треба да се провери. Должината на `x` мора да е n.
  - враќа `true` ако елементот `x` го има во модифицираното множество, а инаку враќа `false`.

Да забележиме дека ако вашата програма ги прекршува дадените ограничувања, излезот од оценувачот ќе биде "Wrong Answer".

Кај сите стрингови првиот знак е најзначајниот бит на соодветниот цел број.

Пред да се повика функцијата `restore_permutation` оценувачот ја фиксира пермутацијата p.

Ве молиме да ги користите дадените темплејт датотеки за детали околу имплементацијата во вашиот програмски јазик.

Пример

Оценувачот го прави следниот повик:

- `restore_permutation(4, 16, 16)`. Дадено е дека $n = 4$ и во програмата може да има најмногу 16 "запишувања" и 16 "читања".

Програмата ги прави следните функциски повици:

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

Само една пермутација е можна земајќи ги предвид дадените повици и резултати кои ги враќа функцијата `check_element()`, а тоа е пермутацијата $p = [2, 1, 3, 0]$. Оттука, `restore_permutation` треба да ја врати низата `[2, 1, 3, 0]`.

Подзадачи

1. (20 поени) $n = 8$, $w = 256$, $r = 256$, $p_i \neq i$ за најмногу 2 индекса i ($0 \leq i \leq n - 1$),
2. (18 поени) $n = 32$, $w = 320$, $r = 1024$,
3. (11 поени) $n = 32$, $w = 1024$, $r = 320$,
4. (21 поени) $n = 128$, $w = 1792$, $r = 1792$,
5. (30 поени) $n = 128$, $w = 896$, $r = 896$.

Оценувач

Оценувачот ги чита влезните податоци во следниот формат:

- линија 1: цели броеви n, w, r,
- линија 2: n цели броеви кои ги претставуваат елементите на p.