messy
Country: GEO

შეცდომის გასწორება

ილშატი პროგრამისტია და მუშაობს ეფექტური მონაცემთა სტრუქტურების შექმნაზე. ერთ დღეს მან მოიფიქრა მონაცემთა ახალი სტრუქტურა. ამ მონაცემთა სტრუქტურას შეუძლია შეინახოს *არაუარყოფითი* n -ბიტიანი მთელი რიცხვები, სადაც n არის ორის ხარისხები. ანუ, $n = 2^b$ რაიმე არაუარყოფითი b რიცხვისათვის.

თავიდან მონაცემთა სტრუქტურა ცარიელია. პროგრამამ, რომელიც იყენებს ამ მონაცემთა სტრუქტურას, უნდა დაიცვას შემდეგი წესები:

- პროგრამას უნდა შეეძლოს მონაცემთა სტრუქტურაში n -ბიტიანი მთელი რიცხვების დამატება, ერთმანეთის მიყოლებით ფუნქცია add_element(x)-ის გამოყენებით. თუ პროგრამა შეეცდება დაამატოს ისეთი ელემენტი, რომელიც უკვე არის სტრუქტურაში, მაშინ იგი არ დაემატება.
- ბოლო ელემენტის დამატების შემდეგ პროგრამამ უნდა გამოიძახოს ფუნქცია compile_set() ზუსტად ერთხელ.
- და ბოლოს, პროგრამას შეუძლია გამოიძახოს ფუნქცია `check_element(x)` იმისათვის, რომ შეამოწმოს, არის თუ არა x ელემენტი მონაცემთა სტრუქტურაში. ეს ფუნქცია შეიძლება გამოძახებული იქნეს ბევრჯერ.

როცა ილშატმა პირველად გაუკეთა რეალიზაცია ამ ფუნქციას, მან დაუშვა შეცდომა ფუნქცია `compile_set()`-ში. ამ ფუნქციის შესრულების შემდეგ შეცდომის გამო მონაცემთა სტრუქტურაში თითოეული რიცხვის ორობითი თანრიგები გადაადგილდება ერთიდაიგივე წესით. ილშატს სურს, რომ იპოვოთ ზუსტი ცვლილებები რიცხვებში, რაც მოხდა შეცდომის გამო.

ფორმალურად, განვიხილოთ მიმდევრობა $P = [p_0, \ldots, p_{n-1}]$, რომელშიც შედის ყველა რიცხვი 0 -დან $(n-1)$ -ის ჩათვლით მხოლოდ ერთხელ. ვუწოდოთ ასეთ მიმდევრობას *გადანაცვლება*. განვიხილოთ ელემენტი, რომლის ორობითი წარმოდგენაც არის $a_0, \ldots, a_{n-1}$ (სადაც a_0 არის პირველი ციფრი). როცა ფუნქცია `compile_set()` გამოიძახება, მაშინ ეს ელემენტი შეიცვლება ელემენტით $a_{p_0}, a_{p_1}, \ldots, a_{p_{n-1}}$.

ყველა ელემენტის ორობითი თანრიგების შესაცვლელად იგივე p გადანაცვლება გამოიყენება. გადანაცვლება შეიძლება იყოს ნებისმიერი, რომელშიც შედის $p_i = i$, სადაც $0 \le i \le n-1$.

მაგალითად, დავუშვათ $n = 4$, $p = [2, 1, 3, 0]$ და გვაქვს ორობითი რიცხვების სიმრავლე `0000`, `1100` და `0111`. ფუნქცია `compile_set`-ის გამოძახების შემდეგ ელემენტები შეიცვლებიან და გახდებიან `0000`, `0101` და `1110`, შესაბამისად.

თქვენი ამოცანაა დაწეროთ პროგრამა, რომელიც მონაცემთა სტრუქტურასთან

ურთიერთქმედებით იპოვის გადანაცვლება p-ს. მან უნდა (შემდეგი მიმდევრობით):
1.შეარჩიოს n-ბიტიანი მთელი რიცხვები,

1. ჩასვას ეს რიცხვები მონაცემთა სტრუქტურაში,
2. გამოიძახოს ფუნქცია `compile_set`, რათა მოხდეს აღწერილი შეცდომა,
3. შეამოწმოს, არის თუ არა ზოგიერთი ელემენტი მოდიფიცირებულ სტრუქტურაში,
4. გამოიყენოს ეს ინფორმაცია, რომ დაადგინოს და დაგვიბრუნოს გადანაცვლება p.

მიაქციეთ ყურადღება, რომ თქვენმა პროგრამამ უნდა გამოიძახოს ფუნქცია `compile_set` მხოლოდ ერთხელ.

ამის გარდა არსებობს შეზღუდვა იმის შესახებ, თუ რამდენჯერ უნდა გამოიძახოთ ბიბლიოთეკებიდან ეს ფუნციები. კერძოდ, მას შეუძლია:

- გამოიძახოს `add_element` არაუმეტეს w-ჯერ (w არის "დამატებისათვის"),
- გამოიძახოს `check_element` არაუმეტეს r-ჯერ (r არის "შემოწმებისათვის").

იმპლემენტაციის დეტალები

თქვენ უნდა შექმნათ ერთი ფუნქცია (მეთოდი):

- `int[] restore_permutation(int n, int w, int r)`
  - `n`: ბიტების რაოდენობა ყოველი ელემენტის ორობით წარმოდგენაში (და აგრეთვე არის p-ს სიგრძეც).
  - `w`: მაქსიმალური რაოდენობა `add_element` ოპერაციისა, რომელიც თქვენს პროგრამას შეუძლია შეასრულოს.
  - `r`: მაქსიმალური რაოდენობა `check_element` ოპერაციისა, რომელიც თქვენს პროგრამას შეუძლია შეასრულოს.
  - ფუნქციამ უნდა დაადგინოს და დაგვიბრუნოს p მიმდევრობა.

In the C language, the function prototype is a bit different:

- `void restore_permutation(int n, int w, int r, int* result)`
  - `n`, `w` and `r` have the same meaning as above.
    - the function should return the restored permutation p by storing it into the provided array `result`: for each i, it should store the value p_i into `result[i]`.

Library functions

სტრუქტურასთან ურთიერთქმედებისათვის თქვენმა პროგრამამ უნდა გამოიყენოს შემდეგი სამი ფუნქცია (მეთოდი):

- `void add_element(string x)`
  ეს ფუნქცია ამატებს `x` ელემენტს სიმრავლეში.
  - `x`: `'0'`-ებისა და `'1'`-ებისაგან შედგენილი სტრიქონი. იგი წარმოადგენს იმ რიცხვის ორობით ჩანაწერს, რომელიც უნდა დაემატოს სტრუქტურას (სიმრავლეს). `x`-ის სიგრძე უნდა იყოს n.
- `void compile_set()`

ეს ფუნქცია გამოძახებული უნდა იქნეს ზუსტად ერთხელ. თქვენს პროგრამას არ შეუძლია გამოიძახოს `add_element()` ამ გამოძახების შემდეგ. თქვენს პროგრამას არ შეუძლია გამოიძახოს `check_element()` ამ გამოძახების წინ.

- `boolean check_element(string x)`
  ეს ფუნქცია ამოწმებს ელემენტი `x` არის თუ არა მოდიფიცირებულ სიმრავლეში.
  - `x`: `'0'`-ებისა და `'1'`-ებისაგან შედგენილი სტრიქონი. სიმბოლოები წარმოადგენენ `x` შესამოწმებელი რიცხვის ორობით წარმოდგენას, სიგრძე უნდა იყოს n.
  - აბრუნებს `true` თუ ელემენტი `x` არის მოდიფიცირებულ სიმრავლეში და `false` წინააღმდეგ შემთხვევაში.

მიაქციეთ ყურადღება, რომ თუკი თქვენი პროგრამა დაარღვევს ზემოთ აღწერილ რომელიმე წესს, შედეგი იქნება "Wrong Answer".

ყველა სტრინგში პირველი სიმბოლო გვაძლევს შესაბამისი რიცხვის უდიდეს ნიშნად ბიტს.

გრადერი აფიქსირებს p გადანაცვლებას ფუნქცია restore_permutation-ის გამოძახების შემდეგ.

Example

გრადერი აკეთებს შემდეგი ფუნქციის გამოძახებას:

- `restore_permutation(4, 16, 16)`. როცა $n = 4$, პროგრამას შეუძლია გამოიძახოს 16-ჯერ "დამატება" და 16 "შემოწმება".

პროგრამა აკეთებს ფუნქციათა შემდეგ გამოძახებებს:

- `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`

`check_element()`-ის მიერ დაბრუნებულ მნიშვნელობებს მხოლოდ ერთი გადანაცვლება შეესაბამება: ესაა გადანაცვლება $p = [2, 1, 3, 0]$. მაშასადამე, `restore_permutation`–მა უნდა დააბრუნოს `[2, 1, 3, 0]`.

Subtasks

1. (20 points) $n = 8$, $w = 256$, $r = 256$, $p_i \neq i$ არაუმეტეს ორი i ინდექსისათვის ($0 \leq i \leq n - 1$),
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

The sample grader reads the input in the following format:

- line 1: integers n, w, r,
- line 2: n integers giving the elements of p.