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

messy
Country: LVA

Nejaukas kļūdas atklāšana

Ilšats ir programmētājs un ir izdomājis jaunu efektīvu datu struktūru. Tajā var glabāt *nenegatīvu* n-bitu veselu skaitļu kopu, kur n ir divnieka pakāpe. Tas ir, $n = 2^b$ kādam nenegatīvam veselam skaitlim b.

Sākotnēji datu struktūra ir tukša. Programmai, kas lieto datu struktūru, ir jāievēro šādi noteikumi:

- Programma var datu struktūrai pa vienam pievienot n-bitu veselus skaitļus, lietojot funkciju `add_element(x)`. Ja programma mēģina pievienot tādu elementu, kāds jau datu struktūrā ir, nekas nenotiek.
- Pēc pēdēja elementa pievienošanas, programmai tieši vienu reizi ir jāizsauc funkcija `compile_set()`.
- Beidzot, programma var izsaukt funkciju `check_element(x)` lai pārbaudītu vai elements x is atrodams datu struktūrā. Šī funkcija var tikt izsaukta vairākkārt.

Kad Ilšats implementēja šo datu struktūru, viņs ielaida kļūdu funkcijā `compile_set()`. Kļūda vienā un tajā pašā veidā pārkārto kopas visu elementu bināros ciparus visiem skaitļiem. Ilšats grib uzzināt precīzu ciparu pārkārtošanas kārtību, ko izraisa ielaistā kļūda.

Formāli, aplūkosim virkni $p = [p_0, \ldots, p_{n-1}]$ kurā katrs skaitlis no 0 līdz $n-1$ atrodams tieši vienu reizi. Mēs sauksim tādu secību par *permutāciju*. Aplūkosim kopas elementu, kura binārie cipari ir $a_0, \ldots, a_{n-1}$ (ar a_0 tiek apzīmēts pats nozīmīgākais bits). Kad ir izsaukta funkcija `compile_set()`, šis elements tiek nomainīts ar elementu $a_{p_0}, a_{p_1}, \ldots, a_{p_{n-1}}$.

Tā pati permutācija p tiek lietota lai pārkārtotu visu skaitļu ciparus. Jebkura permutācija ir iespējama, ieskaitot permutāciju kur $p_i = i$ katram $0 \leq i \leq n-1$.

Piemēram, ja $n = 4$, $p = [2, 1, 3, 0]$, un jūs kopā ievietojat veselus skaitļus kuru binārā reprezentācija ir `0000`, `1100` un `0111`, tad funkcijas `compile_set` izsaukšana maina tos attiecīgi uz `0000`, `0101` un `1110`.

Jūsu uzdevums ir uzrakstīt programmu, kas atrod permutāciju p sadarbojoties ar datu struktūru. Programmai nepieciešams (tieši šādā secībā):

1. izvēlēties n-bitu veselu skaitļu kopu,
2. ievietot šos skaitļus datu struktūrā,
3. izsaukt funkciju `compile_set` lai izraisītu kļūdu,
4. pārbaudīt dažu elementu eksistēšanu izmainītajā kopā,
5. izmantojot šo informāciju, noskaidrot un atgriezt permutāciju p.

Ievērojiet, ka jūsu programma drīkst izsaukt funkciju `compile_set` tikai vienreiz.

Papildus, jūsu programmai eksistē ierobežojumi attiecībā uz bibliotēkas funkciju izsaukšanas reižu skaitu. Precīzāk, tā var

- izsaukt `add_element` ne vairāk kā w reizes (w nāk no vārda "write"),
- izsaukt `check_element` ne vairāk kā r reizes (r nāk no vārda "read").

Implementācijas detaļas

Jums ir jāimplementē viena funkcija (metode):

- `int[] restore_permutation(int n, int w, int r)`
  - `n`: bitu skaits katra kopas elementa binārā reprezentācijā (un arī p garums).
  - `w`: maksimāls skaits `add_element` operāciju, ko ir atļauts izdarīt jūsu programmai.
  - `r`: maksimāls skaits `check_element` operāciju, ko ir atļauts izdarīt jūsu programmai.
  - funkcijai ir jāatgriež atklātā permutācija p.

C valodai funkcijas signatūra ir mazliet atšķirīga:

- `void restore_permutation(int n, int w, int r, int* result)`
  - `n`, `w` and `r` ir tāda pati jēga kā augstāk.
  - funkcijai ir jāatgriež atklātā permutācija p saglābājot to dotaja masīvā `result`: katram i, tai jāsaglabā p_i vērtība elementā `result[i]`.

Bibliotēkas funkcijas

Lai sadarbotos ar datu struktūru, jūsu programmai ir jāizmanto šādas trīs funkcijas (metodes):

- `void add_element(string x)`
  Funkcija pievieno kopai elementu, kura apraksts ir `x`.
  - `x`: simbolu virkne no `'0'` un `'1'`, kas apraksta kopai pievienojamā veselā skaitļa bināro reprezentāciju. `x` garumam ir jābūt n.
- `void compile_set()`
  Šī funkcija jāizsauc tieši vienreiz. Jūsu programma nevar izsaukt `add_element()` pēc šī izsaukuma vai `check_element()` pirms šī izsaukuma.
- `boolean check_element(string x)`
  Funkcija pārbauda vai elements `x` eksistē izmainītajā kopā.
  - `x`: simbolu virkne no `'0'` un `'1'`, kas apraksta pārbaudāmā veselā skaitļa bināro reprezentāciju. `x` garumam ir jābūt n.
  - atgriež `true` ja elements ir izmainītajā kopā, un `false`, ja elementa nav.

Ievērojiet, ka, ja jūsu programma pārkāps jebkurus no augstākminētajiem ierobežojumiem, tad verdikts būs "Wrong Answer".

Visām simbolu virknēm pirmais simbols dod pašu nozīmīgāko attiecīgā veselā skaitļa bitu.

Vērtētājs noteic permutāciju p pirms funkcijas `restore_permutation` izsaukuma.

Implementācijas detaļām lūdzu izmantojiet piedāvātos šablona failus jūsu izmantotajā programmēšanas valodā.

Piemērs

Vērtētājs veic šādu funkcijas izsaukumu:

- `restore_permutation(4, 16, 16)`. Ar šo $n = 4$ un programma var veikt ne vairāk kā 16 `add_element` izsaukumu un 16 `check_element` izsaukumu.

Programma veic šādus funkciju izsaukumus:

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

Tikai viena permutācija atbilst `check_element()` atgrieztajām vērtībam: permutācija $p = [2, 1, 3, 0]$. Tātad, `restore_permutation` ir jāatgriež `[2, 1, 3, 0]`.

Apakšuzdevumi

1. (20 punkti) $n = 8$, $w = 256$, $r = 256$, $p_i \neq i$ ne vairāk kā 2 indeksiem i ($0 \leq i \leq n - 1$),
2. (18 punkti) $n = 32$, $w = 320$, $r = 1024$,
3. (11 punkti) $n = 32$, $w = 1024$, $r = 320$,
4. (21 punkts) $n = 128$, $w = 1792$, $r = 1792$,
5. (30 punkti) $n = 128$, $w = 896$, $r = 896$.

Piemēru vērtētājs

Piemēru vērtētājs lasa ievaddatus sekojošā formātā:

- 1. rinda: veselie skaitļi n, w, r,
- 2. rinda: n veselie skaitļi, kuri definē masīvu p.