messy
Country: EST

Segane viga

Ilshat on tarkvarainsener, kes tegeleb andmestruktuuridega. Ühel päeval leiutas ta uue andmestruktuuri. See andmestruktuur hoiab *mittenegatiivseid* n-bitiseid täisarve, kus n on kahe aste. Teisisõnu, $n = 2^b$, kus b on mingi mittenegatiivne täisarv.

Andmestruktuur on esialgu tühi ja programm, mis seda andmestruktuuri kasutab, peab järgima järgmisi reegleid:

- Programm võib ükshaaval lisada n-bitiseid täisarve struktuuri funktsiooniga `add_element(x)`. Kui programm üritab lisada täisarvu, mis on andmestruktuuris juba olemas, ei juhtu midagi.
- Kui programm on kõik elemendid struktuuri lisanud, peab see täpselt üks kord kutsuma funktsiooni `compile_set()`.
- Lõpuks võib programm kasutada funktsiooni `check_element(x)`, et kontrollida, kas element x on andmestruktuuris. Seda funktsiooni võib kasutada korduvalt.

Ilshat tegi andmestruktuuri esimeses realisatsioonis funktsioonis `compile_set()` vea, mille tulemusel see funktsioon järjestab iga täisarvu bitid ümber, kusjuures kõigi arvude bitid järjestatakse ümber ühtemoodi. Ilshat soovib et Sa tuvastaksid, kuidas see viga arvude bitte ümber järjestab.

Formaalsemalt, vaatleme jada $p = [p_0, \ldots, p_{n-1}]$, kus iga täisarv $0 \ldots n-1$ esineb täpselt üks kord. Sellist jada nimetatakse *permutatsiooniks*. Vaatleme andmestruktuuri mingit elementi, mille bitid on $a_0, \ldots, a_{n-1}$ (kus a_0 on kõrgeim bitt). Funktsioon `compile_set()` asendab selle elemendi uue elemendiga $a_{p_0}, a_{p_1}, \ldots, a_{p_{n-1}}$.

Sama permutatsiooni p rakendatakse andmestruktuuri kõigi täisarvude bittidele. Permutatsioon võib olla ükskõik milline, ka selline, kus iga $0 \le i \le n-1$ korral $p_i = i$.

Olgu näiteks $n = 4$, $p = [2, 1, 3, 0]$ ja olgu andmestruktuuris arvud, mille kahendkujud on `0000`, `1100` ja `0111`. Funktsioon `compile_set` muudab need vastavalt arvudeks `0000`, `0101` ja `1110`.

Sinu ülesanne on kirjutada programm, mis üritab andmestruktuuriga suheldes tuvastada permutatsiooni p. Programm peaks tegema järgmist (antud järjekorras):

1. Vali mingi hulk n-bitiseid täisarve.
2. Lisa need täisarvud andmestruktuuri.
3. Käivita funktsioon `compile_set`, et viga esile kutsuda.
4. Kontrolli, kas mingid elemendid esinevad andmestruktuuris.

5. Kasuta saadud informatsiooni, et tuvastada permutatsioon p.

Pane tähele, et funktsiooni `compile_set` võib välja kutsuda ainult üks kord.

Lisaks võib ka teise teegi funktsioone kutsuda piiratud arv kordi, täpsemalt:
- funktsiooni `add_element` ülimalt w korda (w tähendab "writes"),
- funktsiooni `check_element` ülimalt r korda (r tähendab "reads").

Realisatsioon

Sinu lahendus peab realiseerima järgmise funktsiooni:
- `int[] restore_permutation(int n, int w, int r)`
  - `n`: bittide arv andmestruktuuri elementides (ja lisaks permutatsiooni p pikkus);
  - `w`: maksimaalne lubatud `add_element` kutsete arv;
  - `r`: maksimaalne lubatud `check_element` kutsete arv;
  - funktsioon peab tagastama leitud permutatsiooni p.

C keeles on funktsiooni liides natuke teistsugune:
- `void restore_permutation(int n, int w, int r, int* result)`
  - `n`, `w` ja `r` tähendavad, sama mis enne.
  - funktsioon peab väljastama permutatsiooni p funktsioonile antud massiivi `result`: p_i väärtus salvestada massiivi elementi `result[i]`.

Teegi funktsioonid

Andmestruktuuriga suhtlemiseks võib kasutada järgmisi funktsioone:
- `void add_element(string x)`
  See funktsioon lisab andmestruktuuri täisarvu, mida `x` kirjeldab.
  - `x`: märkidest `'0'` ja `'1'` koosnev sõne pikkusega n, mis esitab lisatava arvu kahendkuju.

- `void compile_set()`
  Seda funktsiooni tuleb kutsuda täpselt üks kord. Kõik `add_element()` kutsed tuleb teha enne ja kõik `check_element()` kutsed pärast seda funktsiooni.
- `boolean check_element(string x)`
  See funktsioon kontrollib, kas `x` esineb andmestruktuuris.
  - `x`: märkidest `'0'` ja `'1'` koosnev sõne pikkusega n, mis esitab kontrollitava arvu kahendkuju;
  - tagastab `true`, kui `x` on andmestruktuuris, ja `false`, kui mitte.

Kui programm eirab antud piiranguid, on hindamise tulemus "Wrong Answer".

Sõnedes vastab esimene märk arvu kõrgeimale bitile.

Hindamissüsteem valib permutatsiooni p enne funktsiooni `restore_permutation` kutsumist.

Vaata ka näitekoodi failides olevat keelespetsiifilist lisainfot.

Näide

Hindamissüsteem teeb järgmise kutse:

- `restore_permutation(4, 16, 16)`. Siin $n = 4$ ja Sinu programm võib teha ülimalt 16 kirjutamist ("writes") ning 16 lugemist ("reads").

Programm esitab järgmised kutsed:

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

Kõigi `check_element()` kutsete tulemustega on kooskõlas ainult üks permutatsioon, nimelt $p = [2, 1, 3, 0]$, seega peaks `restore_permutation` tagastama `[2, 1, 3, 0]`.

Alamülesanded

1. (20 punkti) $n = 8$; $w = 256$; $r = 256$; $p_i \neq i$ ülimalt 2 indeksi i ($0 \leq i \leq n - 1$) korral.
2. (18 punkti) $n = 32$; $w = 320$; $r = 1024$.
3. (11 punkti) $n = 32$; $w = 1024$; $r = 320$.
4. (21 punkti) $n = 128$; $w = 1792$; $r = 1792$.
5. (30 punkti) $n = 128$; $w = 896$; $r = 896$.

Näitekood

Näitekood loeb sisendi järgmisel kujul:

- Rida 1: täisarvud n, w, r.
- Rida 2: n täisarvu, mis kirjeldavad permutatsiooni p.