

## Raspetljavanje belaja

Faruk Kudra je programer, te veliki kapacitet, koji se bavi efikasnim strukturama podataka. Jednoga dana, u kolaboraciji sa profesorima iz Njemačke, je izumio novu strukturu podataka koja može spremiti skup *nenegativnih*  $n$ -bitnih cijelih brojeva, pri čemu je  $n$  stepen broja dva, tj.  $n = 2^b$  za nenegativni cijeli broj  $b$ .

Struktura je na početku prazna. Program koji koristi strukturu mora slijediti sljedeća pravila:

- Program može dodavati  $n$ -bitne elemente u strukturu, jedaan po jedan, koristeći funkciju `add_element(x)`. Pokuša li dodati element koji već postoji u strukturi, ne događa se ništa.
- Nakon dodavanja posljednjeg elementa program treba pozvati funkciju `compile_set()` tačno jednom.
- Program potom može više puta pozvati funkciju `check_element(x)` da bi provjerio sadrži li struktura element  $x$ .

Kada je Faruk prvi put implementirao ovu strukturu, potkrao mu se bug u funkciji `compile_set()`. Bug uzrokuje permutiranje, tj. promjenu poretka bitova svakog elementa skupa na isti način. Faruk vas moli da otkrijete permutaciju bitova uzrokovanu ovim bugom.

Formalno, *permutacija* je niz  $p = [p_0, \dots, p_{n-1}]$  u kojemu se svaki broj od  $0$  do  $n - 1$  javlja tačno jednom. Posmatrajmo element skupa čiji je binarni zapis  $a_0 \dots a_{n-1}$  (pri tome je  $a_0$  najznačajniji bit). Pozivom funkcije `compile_set()` ovaj se element zamjenjuje elementom  $a_{p_0}, a_{p_1} \dots a_{p_{n-1}}$ .

Ista permutacija  $p$  koristi se za promjenu poretka cifara svakog elementa. Bilo koja permutacija je moguća, pa čak i  $p_i = i$  za svaki  $0 \leq i \leq n - 1$ .

Na primjer, neka je  $n = 4$ ,  $p = [2, 1, 3, 0]$ , te smo u skup ubacili elemente čiji su binarni zapisi `0000`, `1100` i `0111`. Pozivom funkcije `compile_set` ovi se elementi pretvaraju u `0000`, `0101` i `1110`, redom.

Vaš je zadatak napisati program koji pronalazi permutaciju  $p$  uz pomoć interakcije sa strukturuom podataka. Program treba (sljedećim redom):

1. odabrati skup  $n$ -bitih cijelih brojeva,
2. ubaciti te brojeve u strukturu podataka,
3. pozvati funkciju `compile_set` da bi uzrokovao bug,
4. provjeriti prisutnost nekih elemenata u izmijenjenom skupu,
5. koristeći te informacije odrediti i vratiti permutaciju  $p$ .

Primijetite da funkciju `compile_set` smijete pozvati samo jednom.

Dodatno, postoje i ograničenja na broj poziva funkcija. Program smije

- pozvati `add_element` najviše  $w$  puta ( $w$  kao "writes"),
- pozvati `check_element` najviše  $r$  puta ( $r$  kao "reads").

## Implementacijski detalji

Implementirajte funkciju:

- `int[] restore_permutation(int n, int w, int r)`
  - $n$ : broj bitova u binarnom zapisu svakog elementa skupa (kao i dužina permutacije  $p$ ).
  - $w$ : maksimalni dozvoljeni broj poziva `add_element`.
  - $r$ : maksimalni dozvoljeni broj poziva `check_element`.
  - funkcija treba vratiti rekonstruisanu permutaciju  $p$ .

U C jeziku, potpis funkcije malo je drugačiji:

- `void restore_permutation(int n, int w, int r, int* result)`
  - $n, w$  i  $r$  znače isto kao gore.
    - funkcija treba vratiti permutaciju  $p$  spremajući je u zadati niz `result`: za svaki  $i$  treba upisati  $p_i$  u `result[i]`.

## Bibliotečne funkcije(metode)

Za interakciju sa strukturom podataka koristite sljedeće funkcije(metode):

- `void add_element(string x)`

Ova funkcija u skup dodaje element opisan sa  $x$ .

  - $x$ : string znakova '0' i '1', binarni zapis cijelog broja koji se dodaje u skup. Dužina stringa  $x$  mora biti  $n$ .
- `void compile_set()`

Ova funkcija mora se pozvati tačno jednom. Nakon nje ne smijete pozivati `add_element()`, a prije nje ne smijete pozivati `check_element()`.
- `boolean check_element(string x)`

Ova funkcija provjerava je li element  $x$  u izmijenjenom skupu.

  - $x$ : string znakova '0' i '1', binarni zapis cijelog broja koji se provjerava. Dužina stringa  $x$  mora biti  $n$ .
  - vraća `true` ako je element  $x$  u izmijenjenom skupu, a `false` inače.

Ako vaš program prekrši bilo koje od gornjih ograničenja, rezultat bodovanja bit će "Wrong Answer".

Za sve stringove, prvi znak odgovara najznačajnijem bitu odgovarajućeg cijelog broja.

Grader će odrediti permutaciju  $p$  prije nego što pozove funkciju `restore_permutation`.

Za implementacijske detalje koristite zadate template datoteke.

## Primjer

Grader poziva:

- `restore_permutation(4, 16, 16)`. Imamo  $n = 4$  te program smije pozvati

najviše 16 "unosa" i 16 "provjera".

Program poziva:

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

S vrijednostima koje je vratio `check_element()` konzistentna je samo permutacija  $p = [2, 1, 3, 0]$ . Dakle, `restore_permutation` treba vratiti  $[2, 1, 3, 0]$ .

### Podzadaci

1. (20 bodova)  $n = 8$ ,  $w = 256$ ,  $r = 256$ ,  $p_i \neq i$  za najviše dva indeksa  $i$  ( $0 \leq i \leq n - 1$ ),
2. (18 bodova)  $n = 32$ ,  $w = 320$ ,  $r = 1024$ ,
3. (11 bodova)  $n = 32$ ,  $w = 1024$ ,  $r = 320$ ,
4. (21 bod)  $n = 128$ ,  $w = 1792$ ,  $r = 1792$ ,
5. (30 bodova)  $n = 128$ ,  $w = 896$ ,  $r = 896$ .

### Priloženi grader

Priloženi grader učitava ulaz u sljedećem obliku:

- red 1: cijeli brojevi  $n$ ,  $w$ ,  $r$ ,
- red 2:  $n$  cijelih brojeva, elementi permutacije  $p$ .