Unscrambling a Messy Bug

Nikola je programer koji se bavi strukturama podataka. Jednog dana mu je na pamet pala nova struktura koja može držati skup *nenegativnih* n-bitnih celih brojeva, pri čemu je n stepen broja dva, tj. $n = 2^b$ za neki nenegativan ceo broj b.

Kao i većina struktura, i ova je na početku prazna. Program koji koristi ovu strukturu mora poštovati sledeća pravila:

- Program može dodavati n-bitne brojeve u strukturu, jedan po jedan, koristeći funkciju `add_element(x)`. Ukoliko se dodaje element koji već postoji u strukturi, ne događa se ništa.
- Nakon dodavanja poslednjeg elementa program treba pozvati funkciju `compile_set()` tačno jednom.
- Program potom može više puta pozvati funkciju `check_element(x)` da bi proverio da li struktura sadrži element x.

Po običaju, Nikoli se potkrao bag prilikom implementacije funkcije `compile_set()`. Bag uzrokuje permutovanje, tj. promenu redosleda bitova svakog broja u strukturi na isti način. Kako Nikola ne bi išao peške skroz do Srbije zbog ove greške, on vas moli da mu pomognete da otkrije permutaciju bitova uzrokovanu ovim bagom.

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

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

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

Vaš zadatak je da napišete program koji pronalazi permutaciju p uz pomoć interakcije sa strukturom podataka. Program treba (u sledećem redosledu):

1. odabrati skup n-bitnih celih brojeva,
2. ubaciti te brojeve u strukturu podataka,
3. pozvati funkciju `compile_set` da bi uzrokovao bag,
4. proveriti prisustvo nekih elemenata u izmenjenom skupu,
5. koristeći te informacije odrediti i vratiti permutaciju p.

Primetite da funkciju `compile_set` smete pozvati samo jednom.

Dodatno, postoje i ograničenja na broj poziva funkcija. Program sme
- pozvati `add_element` najviše w puta (w kao "writes"),
- pozvati `check_element` najviše r puta (r kao "reads").

Detalji implementacije

Potrebno je da implementirate sledeću 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 funkcije `add_element`.
  - `r`: maksimalni dozvoljeni broj poziva funkcije `check_element`.
  - funkcija treba vratiti rekonstruiranu permutaciju p.

U programskom jeziku C, potpis funkcije je malo 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 uz pomoć niza `result`: za svaki i treba upisati p_i u `result[i]`.

Funkcije biblioteke

Za interakciju sa strukturom podataka koristite sledeće funkcije (metode):
- `void add_element(string x)`
  Ova funkcija u skup dodaje element opisan sa `x`.
  - `x`: string znakova `'0'` i `'1'`, binarni zapis celog broja koji se dodaje u skup. Dužina stringa `x` mora biti n.
- `void compile_set()`
  Ova funkcija se mora pozvati tačno jednom. Nakon poziva ove funkcije ne smete pozivati `add_element()`, a pre poziva ne smete pozivati `check_element()`.
- `boolean check_element(string x)`
  Ova funkcija proverava da li je element `x` u izmenjenom skupu.
  - `x`: string znakova `'0'` i `'1'`, binarni zapis celog broja koji se proverava. Dužina stringa `x` mora biti n.
  - vraća `true` ako je element `x` u izmenjenom skupu, a `false` inače.

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

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

Grejder će fiksirati permutaciju p pre nego što pozove funkciju `restore_permutation`.

Koristite date templejt-fajlove za bolji uvid u detalje implementacije za vaš programski jezik.

Primer

Grejder poziva:

- `restore_permutation(4, 16, 16)`. Ovde imamo $n = 4$ i program sme pozvati najviše 16 "unosa" i 16 "provera".

Program poziva:
- `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`

Samo jedna permutacija je konzistentna sa vrednostima koje je vratila funkcija `check_element()` i to je permutacija $p = [2, 1, 3, 0]$. Dakle, `restore_permutation` treba da vrati `[2, 1, 3, 0]`.

Podzadaci

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

Opis priloženog grejdera

Priloženi grejder učitava ulaz u sledećem formatu:
- linija 1: celi brojevi n, w, r,
- linija 2: n celih brojeva, elementi permutacije p.