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

**messy**
Country: ARM

# Վերացնելով խառնաշփոթը

Ծրագրավորող Իլշատը աշխատում է էֆեկտիվ տվյալների կառուցվածքների վրա: Նա հեղինակել է նոր տվյալների կառուցվածք: Վերջինիս մեջ կարելի է պահել *ոչ բացասական*, $n$ բիթանոց ամբողջ թվերի բազմություն, որտեղ $n$-ը հանդիսանում է 2-ի աստիճան: Այսինքն $n = 2^b$ որևէ ոչ բացասական ամբողջ $b$ թվի համար.

Տվյալների կառուցվածքը սկզբում դատարկ է: Այն կարող է օգտագործվել ըստ հետևյալ կանոնների՝

- Ավելացնել մեկ $n$ բիթանոց ոչ բացասական ամբողջ թիվ, կանչելով `add_element(x)`: Եթե ավելացվում է էլեմենտ, որն արդեն առկա է բազմությունում, ապա ոչինչ տեղի չի ունենում: Կարող է կանչվել մի քանի անգամ:
- Վերջին ավելացումից հետո կատարել `compile_set()` կանչը ճիշտ մեկ անգամ:
- Կանչել `check_element(x)`՝ $x$ էլեմենտի առկայությունը ստուգելու համար: Կարող է կանչվել մի քանի անգամ:

Իլշատը սխալ էր թույլ տվել `compile_set()` ֆունկցիայի իրականացման մեջ: Առաջացած սխալի պատճառով ընթացիկ բազմության բոլոր թվերի 2-ական համակարգի թվանշանները վերադասավորվել էին (բոլորը նույն կերպ): Այժմ, Իլշատը ցանկանում է գտնել, թե ինչ տեղափոխության են ենթարկվել թվանշանները:

Ավելի ֆորմալ, դիտարկենք $p = [p_0, \ldots, p_{n-1}]$ հաջորդականությունը, որտեղ $0$-ից $n-1$ թվերը հանդիպում են ճիշտ մեկ անգամ: Այս հաջորդականությանը կանվանենք *«տեղափոխություն»*:
Դիտարկենք մեր բազմության որևէ էլեմենտ: Դիցուք $a_0, \ldots, a_{n-1}$-ը հանդիսանում է այդ թվի 2-ական ներկայացումը (որտեղ $a_0$-ն ամենաավագ բիթն է): `compile_set()` կանչից հետո նրա 2-ական ներկայացումը կլինի $a_{p_0}, a_{p_1}, \ldots, a_{p_{n-1}}$:

Բերված $p$ տեղափոխությունը օգտագործվում է բազմության բոլոր էլեմենտների թվանշանները վերադասավորելու համար: Հնարավոր է ցանկացած տեղափոխություն, նույնիսկ այն դեպքը, երբ $p_i = i$ ցանկացած $0 \le i \le n-1$ համար:

Օրինակ՝ $n = 4$, $p = [2, 1, 3, 0]$, և բազմությանը ավելացվել են `0000`, `1100`, `0111` 2-ական ներկայացում ունեցող էլեմենտները: `compile_set` ֆունկցիայի կանչից հետո նրանք կդառնան համապատասխանաբար `0000`, `0101` և `1110`:

Ձեր խնդիրն է գրել ծրագիր, որը կգտնի $p$ տեղափոխությունը: Այն պետք է հետևի նշված քայլերին (նույն հաջորդականությամբ)՝

1. ընտրել $n$ բիթանոց թվերի բազմություն,
2. ավելացնել դրանք տվյալների կառուցվածքի մեջ,
3. կանչել `compile_set` ֆունկցիան (որի ժամանակ ծրագրային սխալի հետևանքով տեղի կունենա «թվանշանների վերադասավորումը»),
4. ստուգել որոշ էլեմենտների առկայությունը ձևափոխված բազմությունում,
5. օգտագործել ստացված ինֆորմացիան՝ $p$ տեղափոխությունը գտնելու և վերադարձնելու համար:

Ձեր ծրագիրը պետք է կանչի `compile_set` ֆունկցիան *ճիշտ մեկ անգամ*:

Դուք կարող եք՝

- կանչել `add_element` առավելագույնը $w$ անգամ,
- կանչել `check_element` առավելագույնը $r$ անգամ:

## Իրականացման մանրամասները

Դուք պետք է իրականացնեք հետևյալ ֆունկցիան՝

- `int[] restore_permutation(int n, int w, int r)`
  - `n` -- բիթերի քանակը (ինչպես նաև $p$ տեղափոխության երկարությունը).
  - `w` -- `add_element()` կանչերի առավելագույն թույլատրելի քանակը,
  - `r` -- `check_element()` կանչերի առավելագույն թույլատրելի քանակը,
  - ֆունկցիան պետք է վերադարձնի գտնված $p$ տեղափոխությունը:

## Գրադարանային ֆունկցիաները

Տվյալների կառուցվածքը օգտագործելու համար դուք կարող եք օգտագործել հետևյալ երեք ֆունկցիաները՝

- `void add_element(string x)`
  Ավելացնել `x` էլեմենտը բազմությանը:
  - `x` -- ավելացվող `x` էլեմենտի 2-ական ներկացայումը որպես տող՝ բաղկացած `'0'` և `'1'` սիմվոլներից (ճիշտ $n$ սիմվոլից):
- `void compile_set()`
  Վերոհիշյալ ֆունկցիան պետք է կանչվի ճիշտ մեկ անգամ:
  Դրանից առաջ չի կարող կանչվել `check_element()`, իսկ դրանից հետո չի կարող կանչվել `add_element()`:
- `boolean check_element(string x)`
  Ստուգել `x` էլեմենտի առկայությունը բազմությունում:
  - `x` -- ստուգվող `x` էլեմենտի 2-ական ներկացայումը որպես տող՝ բաղկացած `'0'` և `'1'` սիմվոլներից (ճիշտ $n$ սիմվոլից):
  - վերադարձնել `true` եթե `x` էլեմենտը պատկանում է բազմությանը, կամ `false`՝ հակառակ դեպքում:

Ծրագրի կողմից վերը նշված սահմանափակումների խախտման դեպքում արդյունքը կլինի "Wrong Answer":

Բերված բոլոր տողերում առաջին սիմվոլը ներկայացնում է 2-ական

ներկայացման ամենաավագ բիթը։

Գրեյդերը ֆիքսում է $p$ տեղափոխությունը մինչև `restore_permutation` ֆունկցիայի կանչը։

## Օրինակ

Գրեյդերը կատարում է հետևյալ կանչը՝

- `restore_permutation(4, 16, 16)` -- այսինքն՝ $n = 4$ և ձեր ծրագիրը կարող է կատարել ամենաշատը $16$ "write" և $16$ "read".
  Ծրագիր կատարում է հետևյալ կանչերը՝
- `add_element("0001")`
- `add_element("0011")`
- `add_element("0100")`
- `compile_set()`
- `check_element("0001")` կվերադարձնի `false`
- `check_element("0010")` կվերադարձնի `true`
- `check_element("0100")` կվերադարձնի `true`
- `check_element("1000")` կվերադարձնի `false`
- `check_element("0011")` կվերադարձնի `false`
- `check_element("0101")` կվերադարձնի `false`
- `check_element("1001")` կվերադարձնի `false`
- `check_element("0110")` կվերադարձնի `false`
- `check_element("1010")` կվերադարձնի `true`
- `check_element("1100")` կվերադարձնի `false`

Դատելով `check_element()` կանչերի արդյունքներից՝ գոյություն ունի տեղափոխության միայն մեկ հնարավոր տարբերակ, այն է՝ $p = [2, 1, 3, 0]$։ Ստացվեց, որ `restore_permutation` ֆունկցիան այս դեպքում պետք է վերադարձնի `[2, 1, 3, 0]`։

## Ենթախնդիրներ

1. (20 միավոր) $n = 8$, $w = 256$, $r = 256$, $p_i \neq i$ ամենաշատը երկու $i$-երի համար ($0 \leq i \leq n - 1$),
2. (18 միավոր) $n = 32$, $w = 320$, $r = 1024$,
3. (11 միավոր) $n = 32$, $w = 1024$, $r = 320$,
4. (21 միավոր) $n = 128$, $w = 1792$, $r = 1792$,
5. (30 միավոր) $n = 128$, $w = 896$, $r = 896$.

## Գրեյդերի օրինակ

Գրեյդերի օրինակը ստանում է մուտքը հետևյալ ֆորմատով՝

- տող 1: երեք ամբողջ թվեր՝ $n$, $w$, $r$,
- տող 2: $n$ ամբողջ թվեր՝ $p_0, \ldots, p_{n-1}$։