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

messy
Country: CZE

Rozpletení zmatené struktury

Ilšat je softwarový inženýr pracující na efektivních datových strukturách. Jednoho dne vynalezl novou datovou strukturu. Do této datové struktury je možné uložit množinu *nezáporných* n-bitových celých čísel, kde n je mocnina dvou. To znamená, že $n = 2^b$ pro nějaké nezáporné celé číslo b.

Datová struktura je na začátku prázdná. Program používající datovou strukturu musí dodržovat následující pravidla:

- Program může přidávat prvky, které jsou n-bitovými celými čísly, do datové struktury jeden po druhém pomocí funkce `add_element(x)`. Pokud se program pokusí přidat prvek, který je již v datové struktuře obsažen, nic se nestane.
- Po přidání posledního prvku program musí právě jednou zavolat funkci `compile_set()`.
- Poté může program volat funkci `check_element(x)` pro zjištění, zda prvek x je v datové struktuře obsažen. Tuto funkci lze použít vícekrát.

Když Ilšat poprvé implementoval tuto datovou strukturu, udělal chybu ve funkci `compile_set()`. Chyba způsobí zpermutování bitů každého prvku uloženého ve struktuře - a to pro všechny prvky stejně. Ilšat nyní potřebuje zjistit, jakou permutaci bitů chyba způsobila.

Formálně, mějme posloupnost $p = [p_0, \ldots, p_{n-1}]$, kde se každé číslo od 0 do $n-1$ vyskytuje právě jednou. Takové posloupnosti říkáme permutace. Uvažujme prvek ze struktury, jehož bity jsou $a_0, \ldots, a_{n-1}$ (kde a_0 je nejvýznamnější bit). Po zavolání funkce `compile_set()` je tento prvek nahrazen prvkem $a_{p_0}, a_{p_1} \ldots, a_{p_{n-1}}$.

Ilšatova chybná struktura používá k prohození bitů každého prvku stejnou permutaci p. Permutace může být libovolná, včetně možnosti, že $p_i = i$ pro každé $0 \leq i \leq n-1$.

Předpokládejme například, že $n = 4$, $p = [2, 1, 3, 0]$, a do struktury vložíme čísla, jejichž binární reprezentace jsou `0000`, `1100` a `0111`. Volání funkce `compile_set()` změní tyto prvky na `0000`, `0101` a `1110`.

Vaším úkolem je napsat program, který nalezne permutaci p interakcí s Ilšatovou datovou strukturou. Program musí (v tomto pořadí):

1. zvolit množinu n-bitových celých čísel,
2. vložit tato čísla do struktury,
3. zavolat funkci `compile_set()`, což aktivuje chybu,
4. zkontrolovat přítomnost nějakých prvků v modifikované struktuře,
5. tyto informace použít k určení a vrácení hledané permutace p.

Všimněte si, že program může volat funkci `compile_set()` pouze jednou.

Počet možných volání knihovních funkcí je omezen:
- smíte zavolat `add_element` nejvýš w krát (w znamená "zápisy" (writes)),
- smíte zavolat `check_element` nejvýš r krát (r znamená "čtení" (reads)).

Implementační detaily

Vašim úkolem je implementovat jednu funkci (metodu):
- `int[] restore_permutation(int n, int w, int r)`
  - `n`: počet bitů v binární reprezentaci každého prvku struktury (a též délka permutace p).
  - `w`: maximální počet volání `add_element`, které váš program smí provést.
  - `r`: maximální počet volání `check_element`, které váš program smí provést.
  - Funkce musí vrátit zjištěnou permutaci p.

V jazyce C je signatura funkce odlišná:
- `void restore_permutation(int n, int w, int r, int* result)`
  - `n`, `w` a `r` mají stejný význam jako výše
  - Funkce musí vrátit zjištěnou permutaci p vložením do poskytnutého pole `result` takto: pro každé i uloží hodnotu p_i do `result[i]`.

Knihovní funkce

Za účelem interakce s datovou strukturou smí váš program používat následující funkce (metody):
- `void add_element(string x)`
  Tato funkce přidá prvek reprezentovaný řetězcem `x` do struktury.
  - `x`: řetězec znaků `'0'` a `'1'` reprezentující binární podobu celého čísla, které chceme do struktury přidat. Délka `x` musí být n.
- `void compile_set()`
  tato funkce smí být zavolána pouze jednou a poté se již nesmí volat `add_element`. Před zavoláním `compile_set` se nesmí volat `check_element`.
- `boolean check_element(string x)`
  Funkce prověří, zda je `x` v permutované struktuře.
  - `x`: řetězec znaků `'0'` a `'1'` reprezentující binární podobu celého čísla, které chceme ve struktuře hledat. Délka `x` musí být n.
  - vrací `true`, právě když je prvek `x` v permutované struktuře.

Poznámka: porušíte-li výše uvedená pravidla, získáte hodnocení "Wrong Answer".

U všech řetězců představuje první znak nejvýznamnější bit reprezentovaného čísla.

Vyhodnocovač zvolí permutaci p před voláním funkce `restore_permutation`.

Pro další detaily k implementaci ve zvoleném jazyce použijte prosím poskytnuté vzorové soubory.

Příklad

Vyhodnocovač provede následující volání funkce:

- `restore_permutation(4, 16, 16)`. V tomto případě máme $n = 4$ a program smí provést 16 "zápisů" a 16 "čtení".

Program provede následující volání funkce:

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

Těmto hodnotám vraceným z volání funkce `check_element` odpovídá pouze permutace $p = [2, 1, 3, 0]$. Tudíž `restore_permutation` musí vrátit `[2, 1, 3, 0]`.

Podúlohy

1. (20 bodů) $n = 8$, $w = 256$, $r = 256$, $p_i \neq i$ pro nejvýše dva indexy i ($0 \leq i \leq n - 1$),
2. (18 bodů) $n = 32$, $w = 320$, $r = 1024$,
3. (11 bodů) $n = 32$, $w = 1024$, $r = 320$,
4. (21 bodů) $n = 128$, $w = 1792$, $r = 1792$,
5. (30 bodů) $n = 128$, $w = 896$, $r = 896$.

Vzorový vyhodnocovač

Vzorový vyhodnocovač čte vstup v následujícím formátu:

- řádek 1: celá čísla n, w, r,
- řádek 2: n celých čísel představujících prvky permutace p.