Adamkovo kódenie

Adamko (tiež známy ako Kvetináč) veľmi rád programuje v JavaScripte. Jedného dňa potreboval dátovú štruktúru, ktorá je schopná ukladať n-bitové binárne reťazce, pričom n je mocnina 2. No a nová verzia JS priniesla úžasnú funkciu `compile_set()`, ktorú sa Adamko rozhodol okamžite využiť.

Adamkov program funguje nasledovne:

- Na začiatku je dátová štruktúra prázdna.
- Program postupne, jeden po druhom, pridáva do štruktúry n-bitové reťazce použitím funkcie `add_element(x)`. Ak sa program pokúsi pridať do štruktúry reťazec, ktorý sa tam už nachádza, nič sa nestane.
- Po tom, ako sa do štruktúry vloží posledný reťazec, zavolá Adamko funkciu `compile_set()`, ktorá celú štruktúru "zoptimalizuje".
- Následne môže Adamkov program použiť funckiu `check_element(x)`, ktorá zistí, či sa v štruktúre nachádza reťazec x. Túto funkciu môže samozrejme zavolať aj viackrát.

Problém je ale v tom, že v Adamkovej verzii JavaScriptu je vo funkcii `compile_set()` bug. Táto funkcia totiž preusporiada bity každého reťazca permutáciou p. Presnejšie, existuje jedna konkrétna permutácia $p = [p_0, \ldots, p_{n-1}]$, teda postupnosť čísel, v ktorej sa každé číslo medzi 0 a $n-1$ vyskytuje práve raz. Majme ľubovoľný reťazec $a = (a_0, \ldots, a_{n-1})$, ktorý sme vložili do dátovej štrukúry. Použitie funkcie `compile_set()` reťazec a zmení na reťazec $a' = (a_{p_0}, a_{p_1}, \ldots, a_{p_{n-1}})$.

Každý reťazec, ktorý sa nachádza v štruktúre, sa spermutuje tou istou permutáciou p. Permutácia p môže byť ľubovoľná. Prípustná je teda aj možnosť, že p je identita: teda že pre všetky i je $p_i = i$.

Napríklad, ak $n = 4$, $p = [2, 1, 3, 0]$ a do štruktúry sme vložili reťazce `0000`, `1100` a `0111`, po zavolaní `compile_set()` tam budú reťazce `0000`, `0101` a `1110`.

Adamko sa však tejto dátovej štruktúry vzdať nechce, lebo je efektívna. Však predsa stačí, aby vedel ako vyzerá permutácia p. A ako jeho dobrí kamaráti mu to určite radi zistíte.

Vašou úlohou je napísať program, ktorý nájde permutáciu p pomocou interakcie s implementáciou spomínanej dátovej štruktúry. Váš program by mal fungovať nasledovne (a v danom poradí):

1. vyberie si množinu n-bitových reťazcov,
2. vloží tieto reťazce do štruktúry pomocou funkcie `add_element(x)`,
3. zavolá funkciu `compile_set()`, ktorá každý uložený reťazec spermutuje tou

istou permutáciou p,
4. pomocou funkcie `check_element(x)` overí prítomnosť alebo neprítomnosť nejakých reťazcov v množine spermutovaných reťazcov,
5. zo zistených informácií určí permutáciu p a vráti ju ako výstup.

Váš program by mal zavolať funkciu `compile_set()` **práve raz**.

Naviac má váš program obmedzenia na to, koľkokrát môže zavolať príslušné funkcie.

- funkciu `add_element()` môže zavolať najviac w krát,
- funkciu `check_element()` môže zavolať najviac r krát.
  (Názvy premenných w a r sú odvodené od slov "write" a "read".)

Implementačné detaily

Vašou úlohou je naprogramovať funkciu:

- `int[] restore_permutation(int n, int w, int r)`
  - `n`: počet bitov každého reťazca v dátovej štruktúre (a tiež dĺžka p),
  - `w`: maximálny počet zavolaní funkcie `add_element()`, ktoré môže váš program urobiť,
  - `r`: maximálny počet zavolaní funkcie `check_element()`, ktoré môže váš program urobiť.
  - Funkcia by mala vrátiť permutáciu p ako pole dĺžky n, pričom na i-tej pozícii je hodnota p_i.

Pre jazyk C sa deklarácia funkcie trochu líši:

- `void restore_permutation(int n, int w, int r, int* result)`
  - `n`, `w` a `r` majú rovnaký význam ako je uvedené vyššie.
  - Funkcia by mala uložiť vypočítanú permutáciu p do poľa `result`: na pozícii `result[i]` by mal byť hodnota p_i.

Knižničné funkcie

Na vkladanie reťazcov do štruktúry a na zisťovanie príslušnosti reťazca v štruktúre použite nasledovné funkcie:

- `void add_element(string x)`
  Táto funkcia vloží do štruktúry reťazec `x`.
  - `x`: reťazec dĺžky n obsahujúci znaky `'0'` a `'1'`.

- `void compile_set()`
  Táto funkcia musí byť zavolaná práve raz. Váš program **nemôže** použiť funkciu `add_element()` po zavolaní tejto funckcie. Takisto **nemôže** použiť funkciu `check_element()` pred volaním tejto funkcie.
- `boolean check_element(string x)`
  Táto funkcia overí, či sa reťazec `x` nachádza v upravenej štruktúre.
  - `x`: reťazec dĺžky n obsahujúci znaky `'0'` a `'1'`.
  - funkcia vráti `true` ak sa reťazec `x` nachádza v štruktúre a vráti `false` v opačnom prípade.

Ak váš program poruší ktorékoľvek z vyššie uvedených obmedzení, bude hodnotený odpoveďou "Wrong Answer".

Pre lepšie pochopenie konkrétnej implementácie vo vami zvolenom jazyku nahliadnite do priložených ukážkových súborov.

Príklad

Testovač zavolá funkciu `restore_permutation()` s nasledovnými parametrami: `restore_permutation(4, 16, 16)`. Máme teda $n = 4$ a program môže do štruktúry vložiť najviac 16 reťazcov a tiež sa môže opýtať na najviac 16 reťazcov.

Predpokladajme, že váš program spravil nasledujúce volania knižničných funkcií:

- `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`

Iba jediná permutácia $p = [2, 1, 3, 0]$ zodpovedá hodnotám, ktoré vracali volania funkcie `check_element()`. Preto by funkcia `restore_permutation()` mala vrátiť pole `[2, 1, 3, 0]`.

Podúlohy

V každom testovacom vstupe platí, že testovač si zafixuje konkrétnu permutáciu p a až potom zavolá vašu funkciu `restore_permutation()`.

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

Ukážkový grader

Ukážkový grader, ktorý máte k dispozícii číta vstup v nasledovnom formáte:

- riadok 1: čísla n, w, r,
- riadok 2: n čísel udávajúcich permutáciu p.