Posuvné registre (Bit Shift Registers)

Kristínka vyrába nový procesor.

Jej procesor má m buniek pamäte. Budeme ich volať **registre** a očíslujeme si ich od 0 po $m-1$. Každý register má práve b bitov. Platí $m = 100$ a $b = 2000$.

Registre budeme značiť $r[0], r[1], \ldots, r[m-1]$. Na každý register sa môžeme dívať ako na pole b bitov, očíslovaných od 0 (najpravejší bit) po $b-1$ (najľavejší bit). Pre každé i $(0 \leq i \leq m-1)$ a každé j $(0 \leq j \leq b-1)$ budeme pod značením $r[i][j]$ chápať bit číslo j v registri číslo i.

Číselná hodnota postupnosti bitov $d_0, d_1, \ldots, d_{l-1}$ je hodnota $2^0 \cdot d_0 + 2^1 \cdot d_1 + \ldots + 2^{l-1} \cdot d_{l-1}$. (Číselnú hodnotu má postupnosť bitov ľubovoľnej dĺžky l.)

Vravíme, že **v registri i je uložená hodnota h**, ak platí že h je číselná hodnota postupnosti bitov v registri. Inými slovami, hodnota v registri i je $2^0 \cdot r[i][0] + 2^1 \cdot r[i][1] + \ldots + 2^{b-1} \cdot r[i][b-1]$.

Kristínkin procesor má 9 typov **inštrukcií**, ktoré vedia meniť bity registrov. Každá inštrukcia má nejaké registre ako vstupy a jeden register ako výstup. Zápisom $x := y$ budeme označovať operáciu, pri ktorej zmeníme hodnotu x na y. V nasledujúcom zozname uvádzame, čo robia jednotlivé inštrukcie.

- $move(t, y)$: Skopíruj bity registra y do registra t. Formálne, pre každé j $(0 \leq j \leq b-1)$ sprav $r[t][j] := r[y][j]$.
- $store(t, v)$: Ulož do registra t bity z poľa v. Pre tento príkaz musí v byť pole b bitov. Formálne, pre každé j $(0 \leq j \leq b-1)$, sprav $r[t][j] := v[j]$.
- $and(t, x, y)$: Vypočítaj bitový AND registrov x a y. Výsledok ulož do registra t. Formálne, pre každé j $(0 \leq j \leq b-1)$, ak sú **oba bity** $r[x][j]$ a $r[y][j]$ rovné 1, sprav $r[t][j] := 1$, inak sprav $r[t][j] := 0$.
- $or(t, x, y)$: To isté, ale robíme bitový OR. Teda priradenie $r[t][j] := 1$ spravíme ak sa **aspoň jeden** z bitov $r[x][j]$ a $r[y][j]$ rovná 1; inak spravíme priradenie $r[t][j] := 0$.
- $xor(t, x, y)$: To isté, ale robíme bitový XOR. Teda priradenie $r[t][j] := 1$ spravíme ak sa **práve jeden** z bitov $r[x][j]$ a $r[y][j]$ rovná 1; inak spravíme priradenie $r[t][j] := 0$.
- $not(t, x)$: Do registra t ulož bitový NOT registra x. Formálne, pre každé j $(0 \leq j \leq b-1)$, sprav $r[t][j] := 1 - r[x][j]$.
- $left(t, x, p)$: Posuň bity registra x doľava o p pozícií. Tým dostaneš pole v tvorené b bitmi. Jeho obsah je definovaný nasledovne: pre každé j $(0 \leq j \leq b-1)$, ak $j \geq p$ tak

$v[j] = r[x][j-p]$, inak $v[j] = 0$. Toto pole v následne uložíme do registra t (ako v operácii $store$).

- $right(t, x, p)$: Posuň bity registra x doprava o p pozícií. Tým dostaneš pole v tvorené b bitmi. Jeho obsah je definovaný nasledovne: pre každé j $(0 \leq j \leq b-1)$, ak $j \leq b-1-p$ tak $v[j] = r[x][j+p]$, inak $v[j] = 0$. Toto pole v následne uložíme do registra t (ako v operácii $store$).

- $add(t, x, y)$: Sčítaj hodnoty v registroch x a y, výsledok ulož do registra t. Toto sčítanie sa počíta modulo 2^b. (Teda ak sú v registroch x a y hodnoty X a Y také, že $X + Y < 2^b$, chceme bity registra t nastaviť tak, aby mal hodnotu $X + Y$. V opačnom prípade chceme register t nastaviť tak, aby mal hodnotu $X + Y - 2^b$.

A to už je všetko. Kristínka by chcela na tomto procesore riešiť dva typy úloh. Typ úlohy označuje číslo s. Pre každý typ úlohy musíš ty napísať **program** pre Kristínkin procesor -- postupnosť vyššie uvedených inštrukcii.

Vstupom programu je n nezáporných k-bitových čísel $a[0], a[1], \ldots, a[n-1]$. Každé z týchto čísel má k bitov, teda platí $a[i] < 2^k$ ($0 \leq i \leq n-1$).

Na začiatku behu programu je celý tento vstup uložený v registri 0, a to nasledovne: pre každé i $(0 \leq i \leq n-1)$ platí, že postupnosť k bitov $r[0][i \cdot k], r[0][i \cdot k + 1], \ldots, r[0][(i+1) \cdot k - 1]$ má hodnotu $a[i]$. Všimnite si, že nutne $n \cdot k \leq b$.

Na začiatku behu programu sú všetky ostatné bity registra 0 (t.j. tie s indexami od $n \cdot k$ po $b-1$ vrátane) a aj všetky bity ostatných registrov inicializované na 0.

Beh programu spočíva vo vykonaní všetkých jeho inštrukcií v predpísanom poradí. Po vykonaní poslednej inštrukcie sa pozrieme na bity v registri 0 a z nich spočítame **výstup** programu. Výstupom programu je postupnosť n čísel $c[0], c[1], \ldots, c[n-1]$ získaná nasledovne: pre každé i ($0 \leq i \leq n-1$), $c[i]$ je hodnotou postupnosti bitov s indexmi $i \cdot k$ až $(i+1) \cdot k - 1$ v registri 0. Ostatné bity registra 0 ani ostatné registre nemajú vplyv na výstup programu.

- Prvou podúlohou $(s = 0)$ je nájsť minimum postupnosti $a[0], a[1], \ldots, a[n-1]$. Hodnota $c[0]$ vo výstupe musí byť rovná najmenšej zo vstupných hodnôt $a[0], a[1], \ldots, a[n-1]$. Hodnoty $c[1], c[2], \ldots, c[n-1]$ môžu byť ľubovoľné.

- Druhou podúlohou $(s = 1)$ je vstupnú postupnosť $a[0], a[1], \ldots, a[n-1]$ usporiadať do neklesajúceho poradia. Hodnoty $c[0],c[1],\dots,c[n-1]$ majú teda predstavovať tú istú sadu hodnôt ako $a[0], a[1], \ldots, a[n-1]$, len musia byť usporiadané od najmenšej po najväčšiu. Všimnite si, že aj teraz je $c[0]$ rovné minimu postupnosti a.

Pre každú podúlohu vyrobte Kristínke program, ktorý túto podúlohu rieši a má nanajvýš q inštrukcií.

Detaily implementácie

Tvojou úlohou je implementovať nasledujúcu funkciu:

```
void construct_instructions(int s, int n, int k, int q)
```

- s: číslo podúlohy.
- n: počet čísel vo vstupnej postupnosti.
- k: počet bitov v každom čísle vstupnej postupnosti.
- q: maximálny povolený počet inštrukcií v programe.
- Túto funkciu grader zavolá práve raz. Funkcia musí vyrobiť a odovzdať príslušný program.

Tvoja funkcia musí vyrobený program odovzdať tak, že postupne v správnom poradí zavolá nasledovné funkcie gradera:

```
void append_move(int t, int y)
void append_store(int t, bool[] v)
void append_and(int t, int x, int y)
void append_or(int t, int x, int y)
void append_xor(int t, int x, int y)
void append_not(int t, int x)
void append_left(int t, int x, int p)
void append_right(int t, int x, int p)
void append_add(int t, int x, int y)
```

- Každé volanie takejto funkcie pridá na koniec programu príslušnú inštrukciu.
- Parametre t, x, y musia vždy mať hodnotu medzi 0 a $m-1$ vrátane.
- Parametre t, x, y nemusia byť nutne navzájom rôzne (ani v rámci jednej inštrukcie).
- Pre inštrukcie $left$ a $right$, p musí byť nezáporné a nanajvýš rovné b.
- Pre inštrukciu $store$, pole v musí mať dĺžku b.

Pri testovaní svojho riešenia môžeš tiež zavolať túto funkciu gradera:

```
void append_print(int t)
```

- Počas skutočného testovania tvojho riešenia táto funkcia nebude nič robiť, jej volania budú ignorované.
- V ukážkovom graderi táto funkcia pridá na aktuálny koniec programu inštrukciu $print(t)$.
- Keď ukážkový grader stretne pri vykonávaní programu inštrukciu $print(t)$, vypíše postupnosť n celých k-bitových čísel, získaných z prvých $n \cdot k$ bitov registra t. (Viď popis v časti Ukážkový grader.)
- t musí spĺňať $0 \leq t \leq m-1$.
- Volania tejto funkcie nezvyšujú počet inštrukcií v zostrojovanom programe.

Po tom, ako tvoja funkcia `construct_instructions` úspešne vytvorila a odovzdala program, má skončiť. Tvoj program bude následne otestovaný na niekoľkých testoch. Každý test obsahuje konkrétny vstup $a[0], a[1], \ldots, a[n-1]$ tvorený n nezápornými k-bitovými číslami. Tvoj program úspešne vyrieši tento test, ak jeho výstupná postupnosť c spĺňa vyššie uvedené požiadavky pre príslušnú podúlohu.

Testovanie tvojho riešenia ti môže dať jednu z týchto chybových hlášok:

- `Invalid index`: ako parameter t, x alebo y tvoje riešenie skúsilo odovzdať hodnotu mimo povoleného rozsahu (možno zápornú).
- `Value to store is not b bits long`: pole v pre inštrukciu `append_store` nemá dĺžku b.
- `Invalid shift value`: hodnota p v `append_left` alebo `append_right` nie je medzi 0 a b vrátane.
- `Too many instructions`: Tvoja funkcia sa pokúsila do programu pridať viac ako q inštrukcií.

Príklady

Príklad 1

Predpokladajme, že $s = 0,\ n = 2,\ k = 1,\ q = 1000$. Na vstupe sú teda dve celé čísla $a[0]$ a $a[1]$, každé z nich má $k = 1$ bit. Pred začiatkom vykonávania programu platí $r[0][0] = a[0]$ a $r[0][1] = a[1]$. Všetky ostatné bity v procesore sú inicializované na nulu. Keďže $s = 0$, naším cieľom je dosiahnuť, aby na konci platilo $c[0] = r[0][0] = \min(a[0], a[1])$.

V tejto situácii sú len štyri možné vstupy:

- Možnosť 1: $a[0] = 0, a[1] = 0$
- Možnosť 2: $a[0] = 0, a[1] = 1$
- Možnosť 3: $a[0] = 1, a[1] = 0$
- Možnosť 4: $a[0] = 1, a[1] = 1$

Môžeme si všimnúť, že v týchto štyroch prípadoch vždy platí, že minimum $a[0]$ a $a[1]$ je rovné ich bitovému ANDu. Korektný program teda môžeme odovzdať nasledovne:

1. `append_move(1, 0)` pridá inštrukciu, ktorá skopíruje obsah $r[0]$ to $r[1]$.
2. `append_right(1, 1, 1)` pridá inštrukciu, ktorá zoberie bity v $r[1]$, posunie ich doprava o 1 bit a výsledok uloží naspäť do $r[1]$. Keďže $a[0]$ aj $a[1]$ majú len jeden bit, po tomto posune bude platiť, že v $r[1][0]$ je hodnota $a[1]$.
3. `append_and(0, 0, 1)` pridá inštrukciu, ktorá zoberie bitový AND registrov $r[0]$ a $r[1]$ a výsledok uloží do $r[0]$. Tým hodnotu $r[0][0]$ nastavíme na bitový AND doterajších hodnôt $r[0][0]$ a $r[1][0]$, čiže hodnôt $a[0]$ a $a[1]$.

Príklad 2

Predpokladajme, že $s = 1,\ n = 2,\ k = 1,\ q = 1000$. Máme teda rovnako veľký vstup, len tentokrát ho chceme celý usporiadať. Podobne ako v prvom príklade si všimneme, že $\max(a[0], a[1])$ vieme vypočítať ako ich bitový OR. Z toho dostávame, že jedným možným riešením je spraviť nasledovné volania:

1. `append_move(1,0)`
2. `append_right(1,1,1)`
3. `append_and(2,0,1)`

4. `append_or(3,0,1)`
5. `append_left(3,3,1)`
6. `append_or(0,2,3)`

Po týchto inštrukciách bude v $c[0] = r[0][0]$ minimum a v $c[1] = r[0][1]$ maximum zadaných dvoch vstupov, čím sme ich usporiadali.

Obmedzenia

- $m = 100$
- $b = 2000$
- $0 \leq s \leq 1$
- $2 \leq n \leq 100$
- $1 \leq k \leq 10$
- $q \leq 4000$
- $0 \leq a[i] \leq 2^k - 1$ (for all $0 \leq i \leq n - 1$)

Podúlohy

1. (10 points) $s = 0, n = 2, k \leq 2, q = 1000$
2. (11 points) $s = 0, n = 2, k \leq 2, q = 20$
3. (12 points) $s = 0, q = 4000$
4. (25 points) $s = 0, q = 150$
5. (13 points) $s = 1, n \leq 10, q = 4000$
6. (29 points) $s = 1, q = 4000$

Ukážkový grader

Ukážkový grader očakáva na začiatku toto:

- riadok $1:$ s n k q

Nasleduje nejaký počet riadkov a v každom popis jedného testu. Každý test má formát:

- $a[0]$ $a[1]$ $\ldots$ $a[n-1]$

Za posledným testom je riadok, v ktorom je jediné číslo -1.

Ukážkový grader najskôr zavolá tvoju funkciu `construct_instructions(s, n, k, q)`. Ak toto volanie poruší niektoré z obmedzení v zadaní, grader vypíše príslušnú chybu a skončí.

Inak najskôr vypíše všetky odovzdané inštrukcie v správnom poradí. Pre inštrukcie $store$ pole v vypíše od indexu 0 po index $n - 1$.

Potom bude ukážkový grader po jednom prechádzať cez testy. Na každom teste spustí tvoj program.

Pre každú operáciu $print(t)$: nech $d[0], d[1], \ldots, d[n-1]$ je postupnosť celých čísel taká, že pre každé i ($0 \leq i \leq n - 1$), $d[i]$ je hodnotou postupnosti bitov s indexami $i \cdot k$ až $(i + 1) \cdot k - 1$ v

registri t (v čase vykonania tejto inštrukcie). Grader vypíše túto postupnosť nasledovne: `register` t: $d[0]\ d[1]\ \ldots\ d[n-1]$.

Po poslednej inštrukcii grader vypíše výstup tvojho programu: pre $s = 0$ len $c[0]$, pre $s = 1$ vypíše

- $c[0]\ c[1]\ \ldots\ c[n-1]$.

Po poslednom teste grader vypíše `number of instructions:` X kde X je počet inštrukcií, ktoré použil tvoj program.