Registrový počítač

Ing. Kryštof pracuje na návrhu nového druhu procesoru.Tento procesor má m b-bitových **registrů** (paměťových buněk), kde $m = 100$ a $b = 2000$, očíslovaných od 0 do $m - 1$. Hodnoty těchto registrů označme $r[0], r[1], \ldots, r[m-1]$. Každý registr je pole b bitů očíslovaných od 0 (nejpravější bit) do $b - 1$ (nejlevější bit). Pro každé i $(0 \leq i \leq m - 1)$ a každé j $(0 \leq j \leq b - 1)$ označme hodnotu j-tého bitu registru i jako $r[i][j]$.

Číselná hodnota posloupnosti bitů $d_0, d_1, \ldots, d_{l-1}$ (libovolné délky l), je rovna $2^0 \cdot d_0 + 2^1 \cdot d_1 + \ldots + 2^{l-1} \cdot d_{l-1}$. **Číselná hodnota registru** i je rovna číselné hodnotě posloupnosti jeho bitů, tedy $2^0 \cdot r[i][0] + 2^1 \cdot r[i][1] + \ldots + 2^{b-1} \cdot r[i][b-1]$.

Procesor má 9 **instrukcí** měnících hodnoty registrů. Každá z nich bere jako vstup hodnotu jednoho nebo více registrů a výstup ukládá do jednoho z registrů. V následujícím popisu $u := v$ označuje operaci změny hodnoty bitu u na hodnotu v.

- $move(t, y)$: Zkopíruje hodnotu registru y do registru t. Pro každé j $(0 \leq j \leq b - 1)$ přiřadí $r[t][j] := r[y][j]$.
- $store(t, v)$: Nastaví hodnotu registru t na v, kde v je pole b bitů. Pro každé j $(0 \leq j \leq b - 1)$ přiřadí $r[t][j] := v[j]$.
- $and(t, x, y)$: Uloží do registru t bitový AND hodnot v registrech x a y. Pro každé j $(0 \leq j \leq b - 1)$ přiřadí $r[t][j] := 1$ jestliže bity $r[x][j]$ a $r[y][j]$ jsou **oba** 1 a $r[t][j] := 0$ jinak.
- $or(t, x, y)$: Uloží do registru t bitový OR hodnot v registrech x a y. Pro každé j $(0 \leq j \leq b - 1)$ přiřadí $r[t][j] := 1$ jestliže **alespoň jeden** z bitů $r[x][j]$ a $r[y][j]$ je 1 a $r[t][j] := 0$ jinak.
- $xor(t, x, y)$: Uloží do registru t bitový XOR hodnot v registrech x a y. Pro každé j $(0 \leq j \leq b - 1)$ přiřadí $r[t][j] := 1$ jestliže **právě jeden** z bitů $r[x][j]$ a $r[y][j]$ je 1 a $r[t][j] := 0$ jinak.
- $not(t, x)$: Uloží do registru t bitový NOT hodnoty v registru x. Pro každé j $(0 \leq j \leq b - 1)$ přiřadí $r[t][j] := 1 - r[x][j]$.
- $left(t, x, p)$: Uloží do registru t výsledek posunutí hodnoty registru x o p bitů doleva. Nejprve spočítá výsledek v tohoto posunutí, což je b-bitové pole v takové, že pro každé j $(0 \leq j \leq b - 1)$ platí $v[j] = r[x][j - p]$ jestliže $j \geq p$ a $v[j] = 0$ jinak. Poté pro každé j $(0 \leq j \leq b - 1)$ přiřadí $r[t][j] := v[j]$.
- $right(t, x, p)$: Uloží do registru t výsledek posunutí hodnoty registru x o p bitů doprava. Nejprve spočítá výsledek v tohoto posunutí, což je b-bitové pole v takové, že pro každé j

$(0 \leq j \leq b-1)$ platí $v[j] = r[x][j+p]$ jestliže $j \leq b-1-p$ a $v[j] := 0$ jinak. Poté pro každé j $(0 \leq j \leq b-1)$ přiřadí $r[t][j] := v[j]$.

- $add(t, x, y)$: Uloží do registru t zbytek součtu číselných hodnot registrů x a y po dělení 2^b. Formálně, nechť X je číselná hodnota registru x a Y je číselná hodnota registru y. Nechť $T = X + Y$ jestliže $X + Y < 2^b$ a $T = X + Y - 2^b$ jinak. Do registru t jsou přiřazeny bity tak, aby jeho číselná hodnota byla rovna T.

Kryštof po vás chce, abyste na jeho procesoru řešily dva typy úloh; typ úlohy je označen celým číslem s. Pro daný typ úlohy vytvořte **program**, což je posloupnost instrukcí definovaných výše.

Vstup programu se skládá z n nezáporných celých k-bitových čísel $a[0], a[1], \ldots, a[n-1]$, tedy $0 \leq a[i] < 2^k$ pro $0 \leq i \leq n-1$. Na začátku běhu programu jsou tato čísla uložna jedno po druhém v registru 0, pro každé i $(0 \leq i \leq n-1)$ je tedy číselná hodnota posloupnosti k bitů $r[0][i \cdot k], r[0][i \cdot k + 1], \ldots, r[0][(i+1) \cdot k - 1]$ rovna $a[i]$. Vždy platí $n \cdot k \leq b$. Všechny ostatní bity registru 0 (ty s indexy od $n \cdot k$ do $b-1$ včetně) a všechny bity ostatních registrů jsou nastaveny na 0.

Poté jsou postupně vyhodnoceny všechny instrukce v programu. Po vyhodnocenI poslední instrukce se **výstup** programu nachází v registru 0. Výstupem je posloupnost n nezáporných celých čísel $c[0], c[1], \ldots, c[n-1]$, kde pro každé i ($0 \leq i \leq n-1$) je $c[i]$ rovno číselné hodnotě posloupnosti bitů $i \cdot k, \ \ldots, \ (i+1) \cdot k - 1$ registru 0. Hodnoty ostatných bitů registru 0 (s indexy od $n \cdot k$ do $b-1$ včetně) a hodnoty ostatních registrů mohou být libovolné.

- První typ úlohy $(s = 0)$ je nalezení minima ze vstupních čísel $a[0], a[1], \ldots, a[n-1]$. Tedy $c[0]$ musí být minimum z $a[0], a[1], \ldots, a[n-1]$, zatímco hodnoty výstupů $c[1], c[2], \ldots, c[n-1]$ mohou být libovolné.
- Druhý typ úlohy $(s = 1)$ je setřídit vstupní čísla $a[0], a[1], \ldots, a[n-1]$ v neklesajícím pořadí. Pro každé i ($0 \leq i \leq n-1$) musí být $c[i]$ rovno $(1+i)$-tému nejmenšímu z čísel $a[0], a[1], \ldots, a[n-1]$ (speciálně $c[0]$ je rovno minimu ze vstupních čísel).

Vymyslete Kryštofovi programy řešící tyto dva typy úloh. Každý z programů může mít nejvýše q instrukcí.

Implementační detaily

Implementujte následující funkci:

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

- s: typ úlohy.
- n: počet čísel na vstupu.
- k: počet bitů každého ze vstupních čísel.
- q: největší povolený počet instrukcí.
- Tato funkce bude volána právě jednou a musí vytvořit program řešící požadovanou úlohu.

Posloupnost instrukcí tvořících tento program funkce `construct_instructions` vrátí pomocí volání jedné nebo několika z následujících funkcí:

```
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 volání funkce $move(t, y)$ $store(t, v)$, $and(t, x, y)$, $or(t, x, y)$, $xor(t, x, y)$, $not(t, x)$, $left(t, x, p)$, $right(t, x, p)$ či $add(t, x, y)$ přidá odpovídající instrukci na konec programu.
- Čísla registrů t, x, y musí být mezi 0 a $m - 1$ včetně.
- V rámci jedné instrukce t, x a y **nemusí** být navzájem různé.
- Pro instrukce $left$ a $right$ musí p být mezi 0 a b včetně.
- Pro instrukce $store$ musí délka pole v být rovna b.

Vaše řešení také může (i vícekrát) volat následující funkci, která vám může pomoct při testování:

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

- Volání této funkce je ignorováno při ostrém vyhodnocování.
- V ukázkovém vyhodnocovači volání této funkce přidá na konec programu operaci $print(t)$.
- Když ukázkový vyhodnocovač při interpretaci programu vytvořeného vaším řešením narazí na operaci $print(t)$, vypíše n k-bitových nezáporných celých čísel tvořených prvními $n \cdot k$ bity registru t (viz též sekce "Ukázkový vyhodnocovač").
- t musí splňovat $0 \leq t \leq m - 1$.
- Volání této funkce se nezapočítává do počtu instrukcí programu.

Po přidání poslední instrukce do programu se funkce `construct_instructions` musí ukončit. Vytvořený program je poté otestován na několika testovacích vstupech, z nichž každý je tvořen n k-bitovými nezápornými celými čísly $a[0], a[1], \ldots, a[n-1]$. Vaše řešení bude pro takový testovací vstup považováno za správné, jestliže výstup $c[0], c[1], \ldots, c[n-1]$ vašeho programu pro tento vstup splňuje následující podmínky:

- Jestliže $s = 0$, $c[0]$ je rovno nejmenší z hodnot $a[0], a[1], \ldots, a[n-1]$.
- Jestliže $s = 1$, pro každé i ($0 \leq i \leq n-1$) je $c[i]$ rovno $(1+i)$-té nejmenší z hodnot $a[0], a[1], \ldots, a[n-1]$.

Výsledkem vyhodnocení mohou být následující chybové zprávy:

- `Invalid index`: Parametr t, x či y jedné z instrukcí, udávající číslo registru, je chybný (například záporný).
- `Value to store is not b bits long`: Délka pole v ve volání funkce `append_store` není rovna b.
- `Invalid shift value`: Hodnota parametru p volání funkce `append_left` či `append_right` není mezi 0 a b včetně.
- `Too many instructions`: Váš program obsahuje více než q instrukcí.

Příklady

Příklad 1

Nechť $s = 0$, $n = 2$, $k = 1$, $q = 1000$. Na vstupu jsou dvě celá čísla $a[0]$ a $a[1]$, obě tvořená $k = 1$ bitem. Na začátku vyhodnocování programu platí $r[0][0] = a[0]$ a $r[0][1] = a[1]$ a všechny ostatní bity mají hodnotu 0. Na konci vyhodnocování musí platit $c[0] = r[0][0] = \min(a[0], a[1])$, což je minimum z hodnot $a[0]$ a $a[1]$.

Program má tedy pouze 4 možné vstupy:

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

Pro všechny 4 případy platí, že $\min(a[0], a[1])$ je rovno bitovému ANDu hodnot $a[0]$ a $a[1]$. Jedno možné řešení tedy je vytvořit program zavoláním následující posloupnosti funkcí:

1. `append_move(1, 0)`, což přidá do programu instrukci, která zkopíruje $r[0]$ do registru 1.
2. `append_right(1, 1, 1)`, což přidá do programu instrukci, která bity $r[1]$ posune o 1 doprava a výsledek uloží zpět do registru 1. Jelikož vstupní čísla jsou 1-bitová, $r[1][0]$ tedy bude rovno $a[1]$.
3. `append_and(0, 0, 1)`, což přidá do programu instrukci, která do registru 0 uloží bitový AND hodnot $r[0]$ a $r[1]$. Tím je $r[0][0]$ nastaveno na bitový AND hodnot $r[0][0]$ a $r[1][0]$, čímž dostáváme požadovaný bitový AND vstupů $a[0]$ a $a[1]$.

Příklad 2

Nechť $s = 1$, $n = 2$, $k = 1$, $q = 1000$. Stejně jako v minulém přikladu jsou jen 4 možné vstupy a pro každý z nich je $\min(a[0], a[1])$ rovno bitovému AND hodnot $a[0]$ a $a[1]$ a $\max(a[0], a[1])$ je rovno bitovému OR hodnot $a[0]$ a $a[1]$. Možné řešení je tedy následující posloupnost volání funkcí:

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 provedení odpovídajících instrukcí je $c[0] = r[0][0]$ rovno $\min(a[0], a[1])$ a $c[1] = r[0][1]$ je rovno $\max(a[0], a[1])$, čímž je vstup setříděn v neklesajícím pořadí.

Omezení

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

Podúlohy

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

Ukázkový vyhodnocovač

Ukázkový vyhodnocovač načítá vstup v následujícím formátu:

- řádka 1 : $s\ n\ k\ q$

Poté následuje libovolný počet řádek, z nichž každá popisuje jeden testovací vstup. Každý testovací vstup je zadán v následujícím formátu

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

a popisuje vstup skládající se z n nezáporných celých čísel $a[0], a[1], \ldots, a[n-1]$. Vstup pro ukázkový vyhodnocovač je ukončen řádkou obsahující pouze číslo -1.

Ukázkový vyhodnocovač nejprve zavolá funkci `construct_instructions(s, n, k, q)`. Jestliže tato funkce poruší některé z omezení ze zadání, vzorový vyhodnocovač vypíše jednu z chybových zpráv popsaných v sekci "Implementační detaily" a skončí. Jinak vypíše popořadě všechny instrukce přidané do programu funkcí `construct_instructions(s, n, k, q)` . U instrukcí $store$ je pole v vypisováno v pořadí od indexu 0 do indexu $b - 1$.

Vyhodnocovač poté zpracuje po pořadě zadané testovací vstupy a pro každý z nich interpretuje vytvořený program.

Pro instrukci $print(t)$, nechť $d[0], d[1], \ldots, d[n-1]$ je posloupnost nezáporných celých čísel takových, že pro každé i ($0 \leq i \leq n-1$) je $d[i]$ rovno číselné hodnotě posloupnosti bitů registru t s indexy od $i \cdot k$ do $(i+1) \cdot k - 1$ v okamžiku provedení instrukce. Vyhodnocovač vypíše tuto posloupnost ve formátu: `register` t: $d[0]\ d[1]\ \ldots\ d[n-1]$.

Po provedení všech instrukcí programu vyhodnocovač vypíše jeho výsledek.

Jestliže $s = 0$, výstup pro každy testovací vstup má následující formát:

- $c[0]$.

Jestliže $s = 1$, výstup pro každý testovací vstup má následující formát:

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

Po vyhodnocení všech testovacích vstupů vyhodnocovač vypíše `number of instructions:` X, kde X je počet instrukcí v programu.