Bit Shift Regiszterek

Christopher, a mérnök egy új típusú processzoron dolgozik.

A processzorhoz tartozik m különböző b-bites memóriacella (ahol $m = 100$ és $b = 2000$), amelyeket **regisztereknek** hívunk, és 0-tól $m - 1$-ig vannak sorszámozva. A regisztereket jelölje $r[0], r[1], \ldots, r[m-1]$. Minden regiszter egy b bitből álló tömb, amelyet 0-tól (jobb szélső bit) $b - 1$-ig (bal szélső bit) sorszámozunk. Az i. regiszter j. bitjét $r[i][j]$-vel jelöljük $(0 \leq i \leq m - 1$ és $0 \leq j \leq b - 1)$.

Bármely $d_0, d_1, \ldots, d_{l-1}$ bitsorozatra (a hossza l, ami bármennyi lehet), a sorozat **számértéke** $2^0 \cdot d_0 + 2^1 \cdot d_1 + \ldots + 2^{l-1} \cdot d_{l-1}$. Azt mondjuk, hogy az i. **regiszterben tárolt számérték** az ott lévő bitsorozat számértéke, tehát $2^0 \cdot r[i][0] + 2^1 \cdot r[i][1] + \ldots + 2^{b-1} \cdot r[i][b-1]$.

A processzornak 9-féle **utasítása** van, amelyeket a regiszterekben tárolt bitek módosítására lehet használni. Minden utasítás egy vagy több regiszteren dolgozik, és a kimenetét az egyik regiszterbe írja. Az alábbiakban $x := y$ jelöli azt a műveletet, amely az x értékét egyenlővé teszi y-nal. Az utasítások által végrehajtott műveletek az alábbiak:

- $move(t, y)$: Az y. regiszterben tárolt bitek tömbjét átmásolja a t. regiszterbe. Minden j-re $(0 \leq j \leq b - 1)$ $r[t][j] := r[y][j]$.
- $store(t, v)$: A t. regisztert átállítja v-re, ahol v egy b bites tömb. Minden j-re $(0 \leq j \leq b - 1)$ $r[t][j] := v[j]$.
- $and(t, x, y)$: Az x. és y. regiszter tartalmán bitenkénti ÉS műveletet hajt végre, és az eredményt a t. regiszterben tárolja. Minden j-re $(0 \leq j \leq b - 1)$ $r[t][j] := 1$ ha $r[x][j]$ és $r[y][j]$ **mindkettő** 1-es, egyébként $r[t][j] := 0$.
- $or(t, x, y)$: Az x. és y. regiszter tartalmán bitenkénti VAGY műveletet hajt végre, és az eredményt a t. regiszterben tárolja. Minden j-re $(0 \leq j \leq b - 1)$, $r[t][j] := 1$ ha $r[x][j]$ és $r[y][j]$ közül **legalább az egyik** 1-es, egyébként $r[t][j] := 0$.
- $xor(t, x, y)$: Az x. és y. regiszter tartalmán bitenkénti KIZÁRÓ VAGY (XOR) műveletet hajt végre, és az eredményt a t. regiszterben tárolja. Minden j-re $(0 \leq j \leq b - 1)$ $r[t][j] := 1$ ha $r[x][j]$ és $r[y][j]$ közül **pontosan az egyik** 1-es, egyébként $r[t][j] := 0$.
- $not(t, x)$: Az x. regiszter tartalmát bitenként NEGÁLJA, és az eredményt a t. regiszterben tárolja. Minden j-re $(0 \leq j \leq b - 1)$ $r[t][j] := 1 - r[x][j]$.
- $left(t, x, p)$: Az x. regiszter minden bitjét p hellyel balra tolja, és az eredményt a t. regiszterben tárolja. Az x. regiszter tartalmának p hellyel balra tolása egy b-bites v tömböt ad eredményül. Minden j-re $(0 \leq j \leq b - 1)$ $v[j] = r[x][j - p]$ ha $j \geq p$, egyébként $v[j] = 0$. Minden j-re $(0 \leq j \leq b - 1)$ $r[t][j] := v[j]$.

- $right(t, x, p)$: Az x. regiszter minden bitjét p hellyel jobbra tolja, és az eredményt a t. regiszterben tárolja. Az x. regiszter tartalmának p hellyel jobbra tolása egy b-bites v tömböt ad eredményül. Minden j-re $(0 \leq j \leq b-1)$, $v[j] = r[x][j+p]$ ha $j \leq b-1-p$, egyébként $v[j] = 0$. Minden j-re $(0 \leq j \leq b-1)$ $r[t][j] := v[j]$.
- $add(t, x, y)$: Az x. és y. regiszterben tárolt számértékeket összeadja, és az eredményt a t. regiszterben tárolja. Az összeadást modulo 2^b végzi. Formálisan, legyen X az x. regiszter tartalmának számértéke, és Y az y. regiszter tartalmának számértéke a művelet előtt. Jelölje T a t. regiszter tartalmának számértékét a művelet után. Ha $X + Y < 2^b$, a t. regiszter bitjeit úgy állítja be, hogy $T = X + Y$. Egyébként, a t. regiszter bitjeit úgy állítja be, hogy $T = X + Y - 2^b$.

Christpoher kétféle feladatot szeretne megoldani az új processzor használatával. A feladat típusát az s egész szám jelöli. Mindkét típusú feladatra elő kell állítanod egy **programot**, amely a fent definiált műveletek egy sorozata.

A program **bemenete** n egész számból áll: $a[0], a[1], \ldots, a[n-1]$, amelyek mindegyike k-bites, tehát $a[i] < 2^k$ ($0 \leq i \leq n-1$). A program végrehajtása előtt a számok a 0. regiszterben vannak egymás után írva úgy, hogy minden i-re $(0 \leq i \leq n-1)$ a k bit hosszúságú $r[0][i \cdot k], r[0][i \cdot k + 1], \ldots, r[0][(i+1) \cdot k - 1]$ bitsorozat számértéke $a[i]$. Vedd figyelembe, hogy $n \cdot k \leq b$. A 0. regiszterben a többi bit (amelyek sorszáma $n \cdot k$ és $b-1$ közötti, a széleket is beleértve) és a többi regiszterben az összes bit kezdeti értéke 0.

Egy program futtatása a műveleteinek sorban történő végrehajtását jelenti. Miután az utolsó művelet is végrehajtódott, a program **kimenete** a 0. regiszter végső állapota alapján számítódik. Konkrétan, a kimenet egy n egész számból álló sorozat: $c[0], c[1], \ldots, c[n-1]$, ahol minden i-re ($0 \leq i \leq n-1$) $c[i]$ a 0. regiszter $i \cdot k$. bitjétől az $(i+1) \cdot k - 1$. bitjéig terjedő bitsorozat számértéke. Megjegyezzük, hogy a program futtatása után a 0. regiszterben a többi bit (amelyek sorszáma legalább $n \cdot k$) és a többi regiszterben az összes bit tetszőleges lehet.

- Az első feladatban $(s = 0)$ a legkisebb számot kell megkeresni a bemenetben lévő $a[0], a[1], \ldots, a[n-1]$ egész számok között. Konkrétan, a $c[0]$ az $a[0], a[1], \ldots, a[n-1]$ számok minimuma kell legyen. A $c[1], c[2], \ldots, c[n-1]$ értékek tetszőlegesek lehetnek.
- A második feladatban $(s = 1)$ nemcsökkenő sorrendbe kell rendezni az $a[0], a[1], \ldots, a[n-1]$ egész számokat. Konkrétan, minden i-re ($0 \leq i \leq n-1$), $c[i]$ legyen az $1+i$. legkisebb szám az $a[0], a[1], \ldots, a[n-1]$ számok közül (tehát $c[0]$ a bemeneti számok közül a legkisebb).

Adj meg egy-egy legfeljebb q utasításból álló programot a feladatok megoldására.

Megvalósítás

A következő függvényt kell elkészítened:

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

- s: a feladat típusa.
- n: a bemenetben lévő számok darabszáma.
- k: az egyes bemeneti számok bitjeinek száma.
- q: az utasítások maximálisan megengedett száma.
- Ez a függvény pontosan egyszer lesz meghívva, és meg kell adnia egy utasítássorozatot, ami megoldja a kért feladatot.

A függvényen belül az alábbi függvényeket kell hívni az utasítássorozat megadásához:

```
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)
```

- A fentiek mindegyike rendre egy-egy $move(t, y)$ $store(t, v)$, $and(t, x, y)$, $or(t, x, y)$, $xor(t, x, y)$, $not(t, x)$, $left(t, x, p)$, $right(t, x, p)$ illetve $add(t, x, y)$ utasítást ad a programhoz.
- t, x, y legalább 0 és legfeljebb $m - 1$ lehet a releváns utasításokban.
- t, x, y nem feltétlenül páronként különbözőek a releváns utasításokban.
- A $left$ és a $right$ utasításban p legalább 0 és legfeljebb b lehet.
- A $store$ utasításban a v hossza pontosan b legyen.

A megoldásod tesztelésében az alábbi függvény hívása segíthet:

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

- A kiértékelés során ennek a függvénynek minden hívása figyelmen kívül lesz hagyva.
- A minta értékelőben ez egy $print(t)$ műveletet ad a programodhoz.
- Amikor a minta értékelő egy $print(t)$ művelethez ér a program végrehajtása közben, kiír n k-bites egész számot, amelyek a t. regiszter első $n \cdot k$ bitjét alkotják (részletek a "Minta értékelő" című részben).
- $0 \leq t \leq m - 1$ kell legyen.
- Ez a függvényhívás nem növeli a program utasításainak számát.

Az utolsó utasítás megadása után a `construct_instructions` fejeződjön be. A program ezt követően kiértékelődik néhány teszteseten, amelyekben a bemenet n darab k-bites egész számból áll: $a[0], a[1], \ldots, a[n-1]$. A megoldásod átmegy egy adott teszteseten, ha a program $c[0], c[1], \ldots, c[n-1]$ kimenete a megadott bemenet esetén teljesíti a következő feltételeket:

- Ha $s = 0$, a $c[0]$ az $a[0], a[1], \ldots, a[n-1]$ számok közül a legkisebb.

- Ha $s = 1$, minden i-re ($0 \leq i \leq n - 1$), $c[i]$ az $1 + i$. legkisebb szám az $a[0], a[1], \ldots, a[n - 1]$ számok közül.

A megoldásod kiértékelése az alábbi hibaüzenetek egyikét adhatja eredményül:

- `Invalid index`: egy helytelen (akár negatív) regiszter sorszám volt megadva valamelyik fenti függvény t, x vagy y paramétereként.
- `Value to store is not b bits long`: a `append_store` függvények adott v hossza nem b.
- `Invalid shift value`: a `append_left` or `append_right` fügvények adott p értéke nem 0 és b közötti (a széleket beleértve).
- `Too many instructions`: a megoldásod több, mint q utasítást adott meg.

Példák

1. példa

Tegyük fel, hogy $s = 0$, $n = 2$, $k = 1$, $q = 1000$. Két szám van a bemenetben, $a[0]$ és $a[1]$, mindkettő $k = 1$ bites. A program futtatása előtt $r[0][0] = a[0]$ és $r[0][1] = a[1]$. A processzorban minden más bit 0. A program utasításainak végrehajtása után $c[0] = r[0][0] = \min(a[0], a[1])$ kell legyen, ami az $a[0]$ és $a[1]$ értékek közül a kisebb.

Csak 4-féle lehet a program bemenete:

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

Észrevehetjük, hogy mind a 4 esetben $\min(a[0], a[1])$ megegyezik az $a[0]$ és $a[1]$ számokra vett bitenkénti ÉS művelet eredményével. Így egy lehetséges megoldás az alábbi függvényhívásokkal megadott program:

1. `append_move(1, 0)`: hozzáad egy utasítást, amely $r[0]$ tartalmát $r[1]$-be másolja.
2. `append_right(1, 1, 1)`: hozzáad egy utasítást, amely veszi az $r[1]$ összes bitjét, jobbra tolja őket 1 hellyel, és az eredményt visszaírja $r[1]$-be. Mivel minden szám 1 bites, ennek eredményeképp $r[1][0]$ egyenlő lesz $a[1]$-gyel.
3. `append_and(0, 0, 1)`: hozzáad egy utasítást, amely bitenkénti ÉS műveletet végez az $r[0]$ és $r[1]$ regiszterekkel, és az eredményt $r[0]$-ba írja. Az utasítás végrehajtása után $r[0][0]$ az $r[0][0]$ és $r[1][0]$ bitekre vett ÉS művelet eredménye, amely megegyezik az $a[0]$ és $a[1]$ számokra vett bitenkénti ÉS eredményével, és ez a kívánt eredmény.

2. példa

Tegyük fel, hogy $s = 1$, $n = 2$, $k = 1$, $q = 1000$. Ahogy az előző példában is, csak 4-féle lehetséges bemenet van. Mind a 4 esetben $\min(a[0], a[1])$ az $a[0]$ és $a[1]$ számokra vett

bitenkénti ÉS eredménye, míg $\max(a[0], a[1])$ az $a[0]$ és $a[1]$ számokra vett bitenkénti VAGY eredménye. Egy lehetséges megoldás az alábbi függvényhívásokkal megadott program:

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)`

A fenti utasítások végrehajtása után $c[0] = r[0][0]$ értéke $\min(a[0], a[1])$ lesz, és $c[1] = r[0][1]$ értéke $\max(a[0], a[1])$ lesz, tehát a bemenet rendezve lesz.

Korlátok

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

Részfeladatok

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

Minta értékelő

A minta értékelő az alábbi formában olvassa a bemenetet:

- 1. sor: $s\ n\ k\ q$

Ezt néhány sor követi, mindegyik egy tesztesetet ír le. Minden teszteset az alábbi formában van megadva:

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

Ez egy olyan tesztesetet ír le, amelyben a bemenet n egész szám: $a[0], a[1], ..., a[n-1]$. Az összes teszteset leírását egy olyan sor zárja le, amelyben egyetlen -1 van.

A minta értékelő elsőként a `construct_instructions(s, n, k, q)` hívást hajtja végre. Ha ez a függvényhívás nem felel meg valamely feladatleírásbeli feltételnek, az értékelő kiírja a "Megvalósítás" rész végén lévő hibaüzenetek egyikét, és kilép. Egyébként kiírja a `construct_instructions(s, n, k, q)` által megadott utasításokat, sorrendben. A $store$ műveletek esetén a v-t a 0. indextől kezdve a $b-1$. indexig írja ki.

Ezután a minta értékelő sorban feldolgozza a teszteseteket. Minden tesztre a megadott programot futtatja a teszteset bemenetére.

Minden $print(t)$ műveletre legyen $d[0], d[1], \ldots, d[n-1]$ olyan egész számok sorozata, melyben minden i-re ($0 \leq i \leq n-1$), $d[i]$ a t. regiszter $i \cdot k$. bitjétől $(i+1) \cdot k - 1$. bitjéig terjedő részének számértéke (amikor a művelet végrehajtódik). Az értékelő a következő formátumban írja ki ezt a sorozatot: `register` t: $d[0]\ d[1]\ \ldots\ d[n-1]$.

Amint az összes utasítás végrehajtódott, az értékelő kiírja a program kimenetét.

Ha $s = 0$, akkor minden tesztesetre a kimenet formátuma:

- $c[0]$.

Ha $s = 1$, akkor minden tesztesetre a kimenet formátuma:

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

Miután minden tesztesetet lefuttatta, azt írja ki, hogy: `number of instructions:` X, ahol X a programod utasításainak száma.