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) számozunk. 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 **értéke** $2^0 \cdot d_0 + 2^1 \cdot d_1 + \ldots + 2^{l-1} \cdot d_{l-1}$. Azt mondjuk, hogy az i. **regiszterben tárolt érték** az ott lévő bitsorozat értéke, 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 értékeket összeadja, és az eredményt a t. regiszterben tárolja. Az összeadást modulo 2^b végzi. Formálisan, legyen X az x. regiszter tartalmának értéke, és Y az y. regiszter tartalmának értéke a művelet előtt. Jelölje T a t. regiszter tartalmának értékét 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 $T = X + Y - 2^b$.

Christpher 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 értéke $a[i]$. 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 értéke. 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 _move(int t, int y)
void _store(int t, bool[] v)
void _and(int t, int x, int y)
void _or(int t, int x, int y)
void _xor(int t, int x, int y)
void _not(int t, int x)
void _left(int t, int x, int p)
void _right(int t, int x, int p)
void _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 műveletekben.
- t, x, y nem feltétlenül páronként különbözőek a releváns műveletekben.
- A $left$ és $right$ műveletekben p legalább 0 és legfeljebb b lehet.
- A $store$ műveletben a v hossza pontosan b legyen.

A megoldásod tesztelésében az alábbi függvény hívása segíthet:

```
void _print(int t)
```

- A kiértékelés során minden hívása ennek a függvénynek figyelmen kívül lesz hagyva.
- A minta értékelőben ez egy $print(t)$ műveletet ad a programohoz.
- 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 utasítása 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, amely valamelyik fenti függvény t, x vagy y paramétereként volt megadva.
- `Value to store is not b bits long`: a `_store` függvények adott v hossza nem b.
- `Invalid shift value`: a `_left` or `_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. `_move(1, 0)`: hozzáad egy utasítást, amely $r[0]$ tartalmát $r[1]$-be másolja.
2. `_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. `_and(0, 0, 1)`: hozzáad egy utasítást, amely bitenkénti ÉS műveletet végez az which appends an instruction to take the bitwise-AND of $r[0]$ and $r[1]$, then store the result in $r[0]$. After this instruction is executed, $r[0][0]$ is set to the bitwise-AND of $r[0][0]$ and $r[1][0]$, which is equal to the bitwise-AND of $a[0]$ and $a[1]$, as desired.

Example 2

Suppose $s = 1$, $n = 2$, $k = 1$, $q = 1000$. As with the earlier example, there are only 4 possible inputs to the program. For all 4 cases, $\min(a[0], a[1])$ is the bitwise-AND of $a[0]$ and $a[1]$, and $\max(a[0], a[1])$ is the bitwise-OR of $a[0]$ and $a[1]$. A possible solution is to make the following calls:

1. `_move(1,0)`
2. `_right(1,1,1)`

3. `_and(2,0,1)`
4. `_or(3,0,1)`
5. `_left(3,3,1)`
6. `_or(0,2,3)`

After executing these instructions, $c[0] = r[0][0]$ contains $\min(a[0], a[1])$, and $c[1] = r[0][1]$ contains $\max(a[0], a[1])$, which sorts the input.

Constraints

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

Subtasks

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) $s = 1, q = 4000$

Sample Grader

The sample grader reads the input in the following format:

- line 1 : $s\ n\ k\ q$

This is followed by t lines, each describing a single test case. Each test case is provided in the following format:

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

and describes a test case whose input consists of n integers $a[0], a[1], ..., a[n-1]$. The description of all test cases is followed by a single line containing solely -1.

The sample grader first calls `construct_instructions(s, n, k, q)`. If this call violates some constraint described in the problem statement, the sample grader prints one of the error messages listed at the end of the "Implementation Details" section and exits. Otherwise, the sample grader first prints each instruction appended by `construct_instructions(s, n, k, q)` in order. For $store$ instructions, v is printed from index 0 to index $b - 1$.

Then, the sample grader processes test cases in order. For each test case, it runs the constructed program on the input of the test case.

For each $print(t)$ operation, let $d[0], d[1], \ldots, d[n-1]$ be a sequence of integers, such that for each i ($0 \le i \le n-1$), $d[i]$ is the integer value of the sequence of bits $i \cdot k$ to $(i+1) \cdot k - 1$ of register t (when the operation is executed). The grader prints this sequence in the following format: `register` t: $d[0]\ d[1]\ \ldots\ d[n-1]$.

Once all instructions have been executed, the sample grader prints the output of the program.

If $s = 0$, the output of the sample grader for each test case is in the following format:

- $c[0]$.

If $s = 1$, the output of the grader for each test case is in the following format:

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

After executing all test cases, the grader prints `number of instructions:` X where X is the number of instructions in your program.