Bit Shift Registers

Gospodin Malnar izradio je novu verziju svog procesora i ima pravu poslasticu za ovogodišnje olimpijce.

Procesor može pristupiti m različitih b-bitnih segmenata memorije (gdje $m = 100$ and $b = 2000$) koje nazivamo **registrima**, a oni su numerirani od 0 do $m - 1$.

Označit ćemo registre s $r[0], r[1], \ldots, r[m-1]$. Svaki registar je niz od b bitova, numeriranih od 0 (najdesniji bit) do $b - 1$ (najlijeviji bit). Za svaki i $(0 \leq i \leq m - 1)$ i svaki j $(0 \leq j \leq b - 1)$, označit ćemo j-ti bit registra i s $r[i][j]$.

Za neki niz bitova $d_0, d_1, \ldots, d_{l-1}$ (arbitrarne duljine l), **cjelobrojna vrijednost** tog niza jednaka je $2^0 \cdot d_0 + 2^1 \cdot d_1 + \ldots + 2^{l-1} \cdot d_{l-1}$. Kažemo da je **cjelobrojna vrijednost pohranjena u registru** i upravo cjelobrojna vrijednost niza njegovih bitova, tj. iznosi $2^0 \cdot r[i][0] + 2^1 \cdot r[i][1] + \ldots + 2^{b-1} \cdot r[i][b-1]$.

Procesor ima 9 tipova **instrukcija** koje se mogu koristiti za promjenu stanja registara. Svaka instrukcija operira na jednom ili više registara i svoj izlaz sprema u jednom od registara. U daljnjem tekstu, koristit ćemo notaciju $x := y$ za operaciju promjene vrijednosti x tako da postane jdnaka y. Operacije koje izvodi svaki tip instrukcije opisane su u sljedećim odlomcima:

- $move(t, y)$: Kopira niz bitova iz registra y u registar t. Odnosno, za svaki j $(0 \leq j \leq b - 1)$, postavlja $r[t][j] := r[y][j]$.
- $store(t, v)$: Pohranjuje vrijednost v u registar t, gdje je v niz od b bitova. Odnosno, za svaki j, postavlja $r[t][j] := v[j]$.
- $and(t, x, y)$: Pohranjuje u registar t bitovni-AND registara x i y. Odnosno, za svaki j $(0 \leq j \leq b - 1)$, postavlja $r[t][j] := 1$ ako su **oba bita** $r[x][j]$ i $r[y][j]$ jednaka 1. Inače, postavlja $r[t][j] := 0$.
- $or(t, x, y)$: Pohranjuje u registar t bitovni-OR registara x i y. Odnosno, za svaki j $(0 \leq j \leq b - 1)$, postavlja $r[t][j] := 1$ ako je **barem jedan od bitova** $r[x][j]$ i $r[y][j]$ jednak 1. Inače, postavlja $r[t][j] := 0$.
- $xor(t, x, y)$: Pohranjuje u registar t bitovni-XOR registara x i y. Odnosno, za svaki j $(0 \leq j \leq b - 1)$, postavlja $r[t][j] := 1$ ako je **točno jedan od bitova** $r[x][j]$ i $r[y][j]$ jednak 1. Inače, postavlja $r[t][j] := 0$.
- $not(t, x)$: Pohranjuje u retgistar t bitovni-NOT registra x. Odnosno, za svaki j $(0 \leq j \leq b - 1)$, postavlja $r[t][j] := 1 - r[x][j]$.

- $left(t, x, p)$: Pohranjuje u registar t bitovni posmak ulijevo registra x za p mjesta. Rezultat posmaka ulijevo regista x za p mjesta je niz v koji se sastoji od b bitova. Odnosno, za svaki j $(0 \leq j \leq b-1)$, $v[j] = r[x][j-p]$ ako je $j \geq p$, a inače $v[j] = 0$. Za svaki j $(0 \leq j \leq b-1)$, instrukcija postavlja $r[t][j] := v[j]$.

- $right(t, x, p)$: Pohranjuje u registar t bitovni posmak udesno registra x za p mjesta. Rezultat posmaka udesno regista x za p mjesta je niz v koji se sastoji od b bitova. Odnosno, za svaki j $(0 \leq j \leq b-1)$, $v[j] = r[x][j+p]$ ako je $j \leq b-1-p$, a inače $v[j] = 0$. Za svaki j $(0 \leq j \leq b-1)$, instrukcija postavlja $r[t][j] := v[j]$.

- $add(t, x, y)$: Pohranjuje u registar t rezultat zbrajanja vrijednosti pohranjenih u registrima x i y. Zbrajanje se sprovodi modulo 2^b. Formalno, neka je X cjelobrojna vrijednost pohranjena u registru x, a Y neka bude cjelobrojna vrijednost pohranjena u registru y prije operacije. Neka je T cjelobrojna vrijednost pohranjena u registru t nakon operacije. Ako je $X + Y < 2^b$, operacija postavlja bitove registra t tako da vrijedi $T = X + Y$. Inače, operacija postavlja bitove registra t tako da vrjedi $T = X + Y - 2^b$.

Gospodin Malnar od olimpijaca traži da riješe dva tipa problema koriseći njegov novi procesor. Tip problema označen je cijelim brojem s. Za oba tipa problema, vaš je zadatak producirati **program**, odnosno slijed instrukcija opisanih u gornjim odlomcima.

Ulaz u vaš program sastoji se od n cijelih brojeva $a[0], a[1], \ldots, a[n-1]$, svaki od kojih se sastoji od k bitova, tj. $a[i] < 2^k$ ($0 \leq i \leq n-1$). Prije no što se program pokreće, svi brojevi iz ulaza bit će redom pohranjeni u registar 0, i to tako da za svaki i $(0 \leq i \leq n-1)$ cjelobrojna vrijednost niza od k bitova $r[0][i \cdot k], r[0][i \cdot k + 1], \ldots, r[0][(i+1) \cdot k - 1]$ bude jednaka $a[i]$. Primijetite da nužno vrijedi $n \cdot k \leq b$. Svi ostali bitovi u registru 0 (oni s indeksima između $n \cdot k$ i $b-1$ uključivo) i svi bitovi svih ostalih registara inicijalizirani su na 0.

Izvođenje programa u procesoru svodi se na slijedno izvođenje njegovih instrukcija. Nakon izvođenja posljenje instrukcije, **izlaz** programa izračunava se temeljem vrijednosti koj je u tom trenutku pohranjena u registru 0.

Preciznije, izlaz je niz od n cijelih brojeva $c[0], c[1], \ldots, c[n-1]$, gdje za svaki i ($0 \leq i \leq n-1$), $c[i]$ je cjelobrojna vrijednost podniza bitova od $i \cdot k$-og do $(i+1) \cdot k - 1$-og bita registra 0. Primijetite da nakon pokretanja programa, ostali bitovi registra 0 (s indeksima barem $n \cdot k$), kao i bitovi svih ostalih registara, mogu biti proizvoljni.

- Prvi problem $(s = 0)$ je pronaći najmanji broj među ulaznim cijelim brojevima $a[0], a[1], \ldots, a[n-1]$. Preciznije, $c[0]$ mora biti najmanji broj među $a[0], a[1], \ldots, a[n-1]$. Vrijednosti preostalih brojeva $c[1], c[2], \ldots, c[n-1]$ mogu biti proizvoljne.

- Drugi problem $(s = 1)$ je sortirati ulazne cijele brojeve $a[0], a[1], \ldots, a[n-1]$ u neopadajućem poretku. Preciznije, za svaki i ($0 \leq i \leq n-1$), $c[i]$ treba biti jednak $1+i$-tom najmanjem cijelom broju među $a[0], a[1], \ldots, a[n-1]$.

Riješite probleme gospodina Malnara tako da napišete programe od kojih se svaki sastoji od najviše q instrukcija.

Implementacijski detalji

Potrebno je implementirati sljedeću proceduru:

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

- s: tip problema .
- n: duljina ulaznog niza.
- k: broj bitova svakog od ulaznih brojeva.
- q: maksimalan broj instrukcija.
- Ova procedura bit će pozvana točno jednom i treba konstruirati slijed instrukcija koje obavljaju traženi zadatak.

Procedura treba pozvati jednu ili više sljedećih procedura kako bi konstruirala slijed instrukcija:

```
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)
```

- Svaka procedura dodaje instrukciju $move(t, y)$, $store(t, v)$, $and(t, x, y)$, $or(t, x, y)$, $xor(t, x, y)$, $not(t, x)$, $left(t, x, p)$, $right(t, x, p)$ ili $add(t, x, y)$ na kraj programa (tim redom).
- Za sve relevantne instrukcije, t, x, y moraju biti barem 0 i najviše $m - 1$.
- Za sve relevantne instrukcije, t, x, y nisu nužno međusobno različiti.
- Za instrukcije $left$ i $right$, p mora biti najmanje 0 i najviše b.
- Za $store$ instrukciju, duljina v mora biti jednaka b.

Također možete zvati sljedeću proceduru kako biste lakše testirali vaše rješenje:

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

- Svaki poziv ove procedure bit će ignoriran prilikom vrednovanja vašeg rješenja.
- U oglednom ocjenjivaču, ova procedura dodaje $print(t)$ operaciju u program.
- Kada ogledni ocjenjivač naiđe na $print(t)$ operaciju tijekom izvođenja programa, ispisuje n k-bitnih cijelh brojeva dobivenih od prvih $n \cdot k$ bitova registra t (vidi odlomak o oglednom ocjenjivaču za više detalja).
- t mora zadovoljavati $0 \leq t \leq m - 1$.
- Svaki poziv ove procedure ne povećava broj instrukcija završnog programa.

Nakon dodavanja posljednje instrukcije, `construct_instuction` treba završiti izvođenje. Program se tada evaluira na određenom broju testnih primjera, gdje svaki specificira ulaz koji se sastoji od n k-bitnih cijelih brojeva $a[0], a[1], \ldots, a[n-1]$. Vaše rješenje prolazi dani testni primjer ako izlaz vašeg programa $c[0], c[1], \ldots, c[n-1]$ za dani ulaz zadovoljava sljedeća svojstva:

- Ako je $s = 0$, $c[0]$ treba biti najmanja vrijednost među $a[0], a[1], \ldots, a[n-1]$.
- Ako je $s = 1$, za svaki i ($0 \leq i \leq n-1$), $c[i]$ treba biti $1 + i$-ta najmanja vrijednost među $a[0], a[1], \ldots, a[n-1]$.

Vrednovanje vašeg rješenja može rezultirati nekom od sljedećih poruka za pogrešku:

- `Invalid index`: netočan (možebitno negativan) indeks registra predan kao parametar t, x ili y za neki poziv neke od procedura.
- `Value to store is not b bits long`: Duljina parametra v predana instrukciji `_store` nije jednaka b.
- `Invalid shift value`: Vrijednost p predana instrukciji `_left` ili `_right` nije između 0 i b uključivo.
- `Too many instructions`: Vaša procedura pokušala je napraviti više od q instrukcija.

Ističe mi vrijeme za prijevod... ostatak teksta je na engleskom :(

Examples

Example 1

Suppose $s = 0$, $n = 2$, $k = 1$, $q = 1000$. There are two input integers $a[0]$ and $a[1]$, each having $k = 1$ bit. Before the program is executed, $r[0][0] = a[0]$ and $r[0][1] = a[1]$. All other bits in the processor are set to 0. After all the instructions in the program are executed, we need to have $c[0] = r[0][0] = \min(a[0], a[1])$, which is the minimum of $a[0]$ and $a[1]$.

There are only 4 possible inputs to the program:

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

We can notice that for all 4 cases, $\min(a[0], a[1])$ is equal to the bitwise-AND of $a[0]$ and $a[1]$. Therefore, a possible solution is to construct a program by making the following calls:

1. `_move(1, 0)`, which appends an instruction to copy $r[0]$ to $r[1]$.
2. `_right(1, 1, 1)`, which appends an instruction that takes all bits in $r[1]$, shifts them to the right by 1 bit, and then stores the result back in $r[1]$. Since each integer is 1-bit long, this results in $r[1][0]$ being equal to $a[1]$.
3. `_and(0, 0, 1)`, 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

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

Subtasks

1. (10 points) $s = 0, n = 2, k \le 2, q = 1000$
2. (11 points) $s = 0, n = 2, k \le 2, q = 20$
3. (12 points) $s = 0, q = 4000$
4. (25 points) $s = 0, q = 150$
5. (13 points) $s = 1, n \le 10, q = 4000$
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 ($0 \leq i \leq n-1$), $d[i]$ 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.