Бит Жылдыруу Регистрлери

Инженер Кристофер жаңы типтеги компьютердик процессордун үстүндө иштеп жатат.

Процессор m ар кандай b бит болгон эс уячага кире алат (бул жерде $m = 100$ жана $b = 2000$), алар **регистрлер** деп аталып, 0дөн $(m-1)$ге чейин номерленген. Регистрлерди $r[0], r[1], \ldots, r[m-1]$ менен белгилейбиз. Ар бир регистр 0дөн (оң жактагы бит) менен $(b-1)$ге (сол жактагы бит) чейинки b биттер массивин түзөт. Ар бир i $(0 \leq i \leq m-1)$ жана ар бир j $(0 \leq j \leq b-1)$ үчүн i регистрдин j-чи битин $r[i][j]$ менен белгилейбиз.

$d_0, d_1, \ldots, d_{l-1}$ (каалаган узундугу l) биттердин удаалаштыгы үчүн, удаалаштыктын **бүтүн мааниси** $2^0 \cdot d_0 + 2^1 \cdot d_1 + \ldots + 2^{l-1} \cdot d_{l-1}$ барабар болот. i-**Регистрде сакталган бүтүн мааниси** деп анын биттеринин удаалаштыгынын бүтүн мааниси, башкача айтканда, $2^0 \cdot r[i][0] + 2^1 \cdot r[i][1] + \ldots + 2^{b-1} \cdot r[i][b-1]$.

Процессордо **буйруктардын** 9 түрү бар, аларды регистрлердеги биттерди өзгөртүү үчүн колдонсо болот. Ар бир буйрук бир же бир нече регистрде иштейт жана чыгарылганды регистрлердин биринде сактайт. Төмөндө $x := y$ колдонуп, x маанисин yке барабар кылып өзгөртөбүз. Буйруктун ар бир түрү боюнча аткарылган операциялар төмөндө баяндалган.

- $move(t, y)$(жылдыр): t-регистрге y-регистрдеги биттер массивин көчүрүңүз. Ар бир j $(0 \leq j \leq b-1)$ үчүн $r[t][j] := r[y][j]$ болсун.
- $store(t, v)$(сакта): t-регистрди vга барабар кылып коюңуз, мында v - b биттер массиви. Ар бир j $(0 \leq j \leq b-1)$ үчүн $r[t][j] := v[j]$ коюңуз.
- $and(t, x, y)$(жана): x- жана y-регистрлердин бит-ANDин аткарып, натыйжасын t-регистрге сактаңыз. Ар бир j $(0 \leq j \leq b-1)$ үчүн $r[t][j] := 1$, эгерде **экөө тең** $r[x][j]$ жана $r[y][j]$ 1 болсо, болбосо $r[t][j] := 0$ деп коюлган.
- $or(t, x, y)$(же): x- жана y-регистрлердин бит-ORун аткарып, натыйжасын t-регистрге сактаңыз. Ар бир j $(0 \leq j \leq b-1)$ үчүн $r[t][j] := 1$ болсун, эгерде $r[x][j]$ жана $r[y][j]$ 1 **жок дегенде бир** 1 болсо,болбосо $[t][j] := 0$ болсун.
- $xor(t, x, y)$(барабардыгы жок же): x- жана y-регистрлердин бит-XORун аткарып, натыйжасын t-регистрге сактаңыз. Ар бир j $(0 \leq j \leq b-1)$ үчүн $r[t][j] := 1$ болсун, эгерде $r[x][j]$ жана $r[y][j]$ **так бири** 1ге барабар болсо, болбосо $r[t][j] := 0$ болсун.
- $not(t, x)$(эмес): x-регистрдин бит-NOTун аткарып, натыйжасын t-регистрге сактаңыз. Ар бир j $(0 \leq j \leq b-1)$ үчүн $r[t][j] := 1 - r[x][j]$.
- $left(t, x, p)$(сол): x-регистрдеги баардык биттерди p аралыкка солго жылдырып, натыйжасын t-регистрге сактаңыз. x-регистрдеги биттерди солго p аралыкка жылдыруунун натыйжасы b биттен турган v массиви болот. Ар бир j $(0 \leq j \leq b-1)$

үчүн $v[j] = r[x][j-p]$ болсун, эгерде $j \geq p$, болбосо $v[j] = 0$ болсун. Ар бир j $(0 \leq j \leq b-1)$ үчүн $r[t][j] := v[j]$ коюңуз.

- $right(t, x, p)$(оң): x-регистрдеги баардык биттерди p аралыкка оңго жылдырып, натыйжасын t-регистрге сактаңыз. X регистрдеги биттерди оңго p аралыкка жылдыруунун натыйжасы b биттерден турган v массиви болот. Ар бир j $(0 \leq j \leq b-1)$, $v[j] = r[x][j+p]$ болсун, эгерде $j \leq b-1-p$ болсо, болбосо $v[j] = 0$ болсун. Ар бир j $(0 \leq j \leq b-1)$ үчүн $r[t][j] := v[j]$ коюңуз.

- $add(t, x, y)$(кош-менен-модул): x- жана y-регистрлерге сакталган бүтүн маанини кошуп,натыйжаны t-регистрге сактаңыз. Кошумча кошуу 2^b модулу менен жүргүзүлөт. Формалдуу түрдө, операцияга чейин x-регистрде сакталган бүтүн сан X болсун, ал эми y-регистрде сакталган бүтүн сан Y болсун. T операциядан кийин t-регистрге сакталган бүтүн сан болсун. Эгерде $X + Y < 2^b$ болсо, анда t биттерин $T = X + Y$ кылып коюңуз. Болбосо, $T = X + Y - 2^b$ деп t биттерин койгула.

Кристофер сизден жаңы процессорду колдонуп, эки тапшырманы чечүүнү каалайт. Тапшырманын түрү s бүтүн сан менен белгиленет. Тапшырмалардын эки түрү үчүн сиз **программаны** (бул жогоруда аныкталган буйруктардын удаалаштыгы түрүндө) түзүүңүз керек.

Программага **киргизүү** $a[0], a[1], \ldots, a[n-1]$, n бүтүн сандан турат. Алардын ар биринде k бит бар, башкача айтканда, $a[i] < 2^k$ ($0 \leq i \leq n-1$). Программа аткарыла электе, бардык киргизилген сандар 0- регистрге удаа сакталат, мындайча, ар бир i $(0 \leq i \leq n-1)$ үчүн k бит катарынын бүтүн мааниси $r[0][i \cdot k]$, $r[0][i \cdot (k+1)]$, $\ldots, r[0][(i+1) \cdot k - 1]$ га барабар $a[i]$. $n \cdot k \leq b$ экендигин эске алыңыз. 0-регистриндеги бардык башка биттер (башкача айтканда, $n \cdot k \ldots b-1$ индекстери бар) жана башка бардык регистрлердеги биттер 0 менен башталган.

Программаны иштетүү анын буйруктарынын ирети менен аткаруудан турат. Акыркы буйрук аткарылгандан кийин, программанын **чыгышы** 0-регистрдеги биттердин акыркы маанисинин негизинде эсептелет. Тактап айтканда, чыгышы n бүтүн сандардын удаалаштыгы $c[0], c[1], \ldots, c[n-1]$ түрүндө болот, бул жерде ар бир i үчүн ($0 \leq i \leq n-1$)), $c[i]$ 0-регистрдин $i \cdot k$ дан $(i+1) \cdot k - 1$ биттеринен турган удаалаштыктын бүтүн маанисине барабар.

Эске алыңыз, программаны иштеткенден кийин 0-регистрдин калган биттери (жок дегенде $n \cdot k$ индекстери менен) жана башка бардык регистрлердин бардык биттери каалагандай болушу мүмкүн.

- $(s = 0)$ биринчи тапшырмасы $a[0], a[1], \ldots, a[n-1]$ сандарынын арасынан эң кичинекей бүтүн санды табуу. Тактап айтканда, $c[0]$ $a[0], a[1], \ldots, a[n-1]$ минималдуусу болушу керек. $c[1], c[2], \ldots, c[n-1]$ маанилери каалагандай болушу мүмкүн.

- $(s = 1)$ экинчи тапшырма $a[0], a[1], \ldots, a[n-1]$ киргизилген бүтүн сандарды барабар же өсүү ирети менен иреттөө. Тактап айтканда, ар бир i ($0 \leq i \leq n-1$) үчүн $c[i]$ $a[0], a[1] \ldots, a[n-1]$ аралыгындагы $1+i$ - кичинекей бүтүн санга барабар болушу керек. (б.а. $c[0]$ - киргизилген сандардын ичиндеги эң кичинекей бүтүн сан).

Кристоферге ушул тапшырмаларды чече турган, qдан көп эмес буйруктан турган программаларды бериңиз.

Implementation Details

You should implement the following procedure:

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

- s: type of task.
- n: number of integers in the input.
- k: number of bits in each input integer.
- q: maximum number of instructions allowed.
- This procedure is called exactly once and should construct a sequence of instructions to perform the required task.

This procedure should call one or more of the following procedures to construct a sequence of instructions:

```
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)
```

- Each procedure appends a $move(t, y)$ $store(t, v)$, $and(t, x, y)$, $or(t, x, y)$, $xor(t, x, y)$, $not(t, x)$, $left(t, x, p)$, $right(t, x, p)$ or $add(t, x, y)$ instruction to the program, respectively.
- For all relevant instructions, t, x, y must be at least 0 and at most $m - 1$.
- For all relevant instructions, t, x, y are not necessarily pairwise distinct.
- For $left$ and $right$ instructions, p must be at least 0 and at most b.
- For $store$ instructions, the length of v must be b.

You may also call the following procedure to help you in testing your solution:

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

- Any call to this procedure will be ignored during the grading of your solution.
- In the sample grader, this procedure appends a $print(t)$ operation to the program.
- When the sample grader encounters a $print(t)$ operation during the execution of a program, it prints n k-bit integers formed by the first $n \cdot k$ bits of register t (see "Sample Grader" section

for details).
- t must satisfy $0 \leq t \leq m-1$.
- Any call to this procedure does not add to the number of constructed instructions.

After appending the last instruction, `construct_instructions` should return. The program is then evaluated on some number of test cases, each specifying an input consisting of n k-bit integers $a[0], a[1], \ldots, a[n-1]$. Your solution passes a given test case if the output of the program $c[0], c[1], \ldots, c[n-1]$ for the provided input satisfies the following conditions:

- If $s = 0$, $c[0]$ should be the smallest value among $a[0], a[1], \ldots, a[n-1]$.
- If $s = 1$, for each i ($0 \leq i \leq n-1$), $c[i]$ should be the $1+i$-th smallest integer among $a[0], a[1], \ldots, a[n-1]$.

The grading of your solution may result in one of the following error messages:

- `Invalid index`: an incorrect (possibly negative) register index was provided as parameter t, x or y for some call of one of the procedures.
- `Value to store is not b bits long`: the length of v given to `_store` is not equal to b.
- `Invalid shift value`: the value of p given to `_left` or `_right` is not between 0 and b inclusive.
- `Too many instructions`: your procedure attempted to append more than q instructions.

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.