ثبات‌های شیفت بیت

مهندس کریستوفر روی یک نوع جدیدی از پردازنده کامپیوتر کار می‌کند.

پردازنده به m خانه حافظه b بیتی دسترسی دارد (که $m = 100$ و $b = 2000$)، که **ثبات** نامیده می‌شوند، و از 0 تا $m - 1$ شماره گذاری شده‌اند. ما ثبات‌ها را با $r[0], r[1], \ldots, r[m-1]$ مشخص می‌کنیم. هر ثبات یک آرایه از b بیت است، که از 0 (راست‌ترین بیت) تا $b - 1$ (چپ‌ترین بیت) شماره گذاری شده‌اند. برای هر i $(0 \leq i \leq m - 1)$ و هر j $(0 \leq j \leq b - 1)$، ما j-امین بیت ثبات i را با $r[i][j]$ مشخص می‌کنیم.

برای هر دنباله از بیت‌ها مثل $d_0, d_1, \ldots, d_{l-1}$ (به طول دلخواه l)، **مقدار صحیح** دنباله برابر با $2^0 \cdot d_0 + 2^1 \cdot d_1 + \ldots + 2^{l-1} \cdot d_{l-1}$ است. ما می‌گوییم که **مقدار صحیح ذخیره شده در ثبات** i، مقدار صحیح دنباله بیت‌های آن است. یعنی، برابر با $2^0 \cdot r[i][0] + 2^1 \cdot r[i][1] + \ldots + 2^{b-1} \cdot r[i][b-1]$ است.

پردازنده 9 نوع **دستور** دارد که برای تغییر بیت‌های ثبات‌ها می‌توانند استفاده شوند. هر دستور روی یک یا چند ثبات کار می‌کند و خروجی را در یکی از ثبات‌ها ذخیره می‌کند. در ادامه، ما از $x := y$ برای مشخص کردن عمل تغییر مقدار x به صورتی که مساوی y شود، استفاده می‌کنیم. عملیات‌های اجرا شده با هر نوع دستور در پایین توضیح داده شده‌اند.

- $move(t, y)$: کپی کردن آرایه بیت‌ها از ثبات y به ثبات t. برای هر j $(0 \leq j \leq b - 1)$، قرار می‌دهیم $r[t][j] := r[y][j]$.
- $store(t, v)$: ثبات t را برابر با v قرار می‌دهیم، که v یک آرایه از b بیت است. برای هر j $(0 \leq j \leq b - 1)$ قرار می‌دهیم $r[t][j] := v[j]$.
- $and(t, x, y)$: AND-بیتی ثبات‌های x و y را می‌گیرد، و حاصل را در ثبات t ذخیره می‌کند. برای هر j $(0 \leq j \leq b - 1)$، قرار می‌دهیم $r[t][j] := 1$ اگر **هر دو** $r[x][j]$ و $r[y][j]$ برابر 1 باشند، و در غیر این صورت قرار می‌دهیم $r[t][j] := 0$.
- $or(t, x, y)$: OR-بیتی ثبات‌های x و y را می‌گیرد، و حاصل را در ثبات t ذخیره می‌کند. برای هر j $(0 \leq j \leq b - 1)$، قرار می‌دهیم $r[t][j] := 1$ اگر **حداقل یکی** از $r[x][j]$ و $r[y][j]$ برابر 1 باشند، و در غیر این صورت قرار می‌دهیم $r[t][j] := 0$.
- $xor(t, x, y)$: XOR-بیتی ثبات‌های x و y را می‌گیرد، و حاصل را در ثبات t ذخیره می‌کند. برای هر j $(0 \leq j \leq b - 1)$، قرار می‌دهیم $r[t][j] := 1$ اگر **دقیقاً یکی** از $r[x][j]$ و $r[y][j]$ برابر 1 باشند، و در غیر این صورت قرار می‌دهیم $r[t][j] := 0$.
- $not(t, x)$: NOT-بیتی ثبات x را می‌گیرد، و حاصل را در ثبات t ذخیره می‌کند. برای هر j $(0 \leq j \leq b - 1)$، قرار می‌دهیم $r[t][j] := 1 - r[x][j]$.
- $left(t, x, p)$: تمام بیت‌ها در ثبات x را به چپ p تا شیفت می‌دهیم، و حاصل را در ثبات t ذخیره می‌کنیم. حاصل شیفت دادن p تا به چپ بیت‌ها در ثبات x، آرایه v شامل b بیت است. برای هر j $(0 \leq j \leq b - 1)$، $v[j] = r[x][j - p]$ اگر $j \geq p$، و در غیر این صورت $v[j] = 0$. برای هر j $(0 \leq j \leq b - 1)$، قرار می‌دهیم $r[t][j] := v[j]$.

- $right(t, x, p)$: تمام بیت‌ها در ثبات x را به راست p تا شیفت می‌دهیم، و حاصل را در ثبات t ذخیره می‌کنیم. حاصل شیفت دادن p تا به راست بیت‌ها در ثبات x ، آرایه v شامل b بیت است. برای هر j $(0 \le j \le b-1)$ ، $v[j] = r[x][j+p]$ اگر $j \le b-1-p$ ، و در غیر این صورت $v[j] = 0$. برای هر j $(0 \le j \le b-1)$ ، قرار می‌دهیم $r[t][j] := v[j]$.

- $add(t, x, y)$: مقدار صحیح ذخیره شده در ثبات x و ثبات y را جمع می‌کنیم، و حاصل را در ثبات t ذخیره می‌کنیم. عملیات جمع به پیمانه 2^b انجام می‌شود. به صورت دقیق‌تر، فرض کنید X مقدار صحیح ذخیره شده در ثبات x ، و Y مقدار صحیح ذخیره شده در y قبل عملیات باشد. فرض کنید T مقدار صحیح ذخیره شده در t بعد عملیات باشد. اگر $X + Y < 2^b$ ، بیت‌های t را به صورتی قرار می‌دهیم که $T = X + Y$ باشد. در غیر این صورت، بیت‌های t راه به صورتی قرار می‌دهیم که $T = X + Y - 2^b$ باشد.

کریستوفر دوست دارد که شما با استفاده از پردازنده جدید دو نوع مسئله را حل کنید. نوع مسئله با یک عدد صحیح s مشخص می‌شود. برای هر نوع از مسائل، شما نیاز به اجرا یک **برنامه** دارید، که دنباله‌ای از دستورات تعریف شده در بالا است.

ورودی برنامه شامل n عدد صحیح $a[0], a[1], \ldots, a[n-1]$ است، هر کدام k -بیت دارند، یعنی، $a[i] < 2^k$ ($0 \le i \le n-1$). قبل از آن که برنامه اجرا شود، تمام اعداد ورودی در ثبات 0 به ترتیب ذخیره می‌شوند، به صورتی که برای هر i $(0 \le i \le n-1)$ مقدار صحیح دنباله از k بیت $r[0][i \cdot k], r[0][i \cdot k + 1], \ldots, r[0][(i+1) \cdot k - 1]$ برابر با $a[i]$ باشد. توجه کنید که $n \cdot k \le b$. سایر بیت‌ها در ثبات 0 (یعنی آن‌هایی که اندیسشان بین $n \cdot k$ and $b - 1$ ، شامل دو سر) و تمام بیت‌ها در سایر ثبات‌ها در ابتدا 0 هستند.

اجرا یک برنامه عبارت است از اجرای به ترتیب دستورات آن. بعد از این که آخرین دستور اجرا شد، **خروجی** برنامه بر اساس مقدار نهایی ذخیره شده در بیت‌های ثبات 0 محاسبه می‌شود. به صورت دقیق‌تر، خروجی یک دنباله از n عدد صحیح $c[0], c[1], \ldots, c[n-1]$ است، که برای هر i $(0 \le i \le n-1)$، $c[i]$ مقدار صحیح یک دنباله از بیت‌های $i \cdot k$ تا $(i+1) \cdot k - 1$ ثبات 0 است. توجه کنید که بعد از اجرای برنامه، بیت‌های باقی‌مانده از ثبات 0 (با اندیس حداقل $n \cdot k$) و تمام بیت‌های سایر ثبات‌ها می‌توانند دلخواه باشند.

- مسئله اول $(s = 0)$ پیدا کردن کوچک‌ترین عدد صحیح بین ورودی‌های صحیح $a[0], a[1], \ldots, a[n-1]$ است. به صورت دقیق‌تر، $c[0]$ باید کوچک‌ترین از $a[0], a[1], \ldots, a[n-1]$ باشد. مقدار $c[1], c[2], \ldots, c[n-1]$ می‌تواند دلخواه باشد.

- مسئله دوم $(s = 1)$ مرتب کردن اعداد صحیح ورودی $a[0], a[1], \ldots, a[n-1]$ به صورت غیر نزولی است. به صورت دقیق‌تر، برای هر i $(0 \le i \le n-1)$، $c[i]$ باید برابر با $i+1$ -امین کوچک‌ترین عدد صحیح بین $a[0], a[1], \ldots, a[n-1]$ باشد. (یعنی $c[0]$ کوچک‌ترین عدد صحیح بین ورودی‌های صحیح است).

به کریستوفر برنامه‌هایی، هر کدام شامل حداکثر q دستور، ارائه دهید که بتوانند این مسائل را حل کنند.

جزئیات پیاده‌سازی

شما باید تابع زیر را پیاده‌سازی کنید:

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

- s : نوع مسئله.
- n : تعداد اعداد صحیح در ورودی.

- k : تعداد بیت‌ها در هر ورودی صحیح.
- q : بیش‌ترین تعداد دستور مجاز.
- این تابع دقیقاً یک مرتبه فراخوانی می‌شود و باید دنباله‌ای از دستورات برای اجرای مسئله خواسته شده بسازد.

این تابع باید یک یا بیش‌تر از توابع زیر را فراخوانی کند تا یک دنباله از دستورات بسازد:

```
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)
```

- هر تابع یک دستور $move(t, y)$ ، $store(t, v)$ ، $and(t, x, y)$ ، $or(t, x, y)$ ، $xor(t, x, y)$ ، $not(t, x)$ ، $left(t, x, p)$ ، $right(t, x, p)$ یا $add(t, x, y)$ به برنامه اضافه می‌کند، به همین ترتیب.
- برای هر دستور مربوطه، t ، x ، y باید حداقل 0 و حداکثر $m - 1$ باشند.
- برای هر دستور مربوطه، t ، x ، y لازم نیست که دو به دو متفاوت باشند.
- برای دستورات $left$ و $right$ ، p باید حداقل 0 و حداکثر b باشد.
- برای دستورات $store$ ، طول v باید b باشد.

شما همچنین می‌توانید تابع زیر را برای کمک به تست کردن پاسختان فراخوانی کنید:

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

- هر فراخوانی از این تابع هنگام ارزیابی پاسختان نادیده گرفته می‌شود.
- در ارزیاب نمونه، این تابع یک دستور $print(t)$ به برنامه اضافه می‌کند.
- هنگامی که ارزیاب نمونه به یک دستور $print(t)$ در طول اجرای برنامه مواجه می‌شود، n عدد صحیح k-بیتی متشکل از اولین $n \cdot k$ بیت ثبات t چاپ می‌کند (بخش «ارزیاب نمونه» را برای جزئیات بیش‌تر ببینید).
- t باید $0 \leq t \leq m - 1$ را ارضا کند.
- هر فراخوانی این تابع به تعداد دستورات ساخته شده اضافه نمی‌کند.

بعد از اضافه کردن آخرین دستور، `construct_instructions` باید return را صدا کند. برنامه آن‌گاه روی تعدادی از تست کیس‌ها اجرا می‌شود، هر کدام مشخص می‌کنند یک ورودی شامل n عدد صحیح k-بیتی $a[0], a[1], \ldots, a[n-1]$ است. پاسخ شما در یک تست کیس داده شده قبول می‌شود اگر خروجی برنامه $c[0], c[1], \ldots, c[n-1]$ برای ورودی داده شده شرایط زیر را ارضا کند:

- اگر $s = 0$ ، $c[0]$ باید کوچک‌ترین مقدار بین $a[0], a[1], \ldots, a[n-1]$ باشد.
- اگر $s = 1$ ، برای هر i $(0 \leq i \leq n-1)$، $c[i]$ باید برابر با $i + 1$-امین کوچک‌ترین عدد صحیح بین $a[0], a[1], \ldots, a[n-1]$ باشد.

ارزیابی پاسخ شما ممکن است به یکی از خطاهای زیر نتیجه دهد:

- `Invalid index`: یک ایندکس ثبات غلط (احتمالاً منفی) به عنوان پارامتر t ، x یا y برای یک فراخوانی از یکی از تابع‌ها داده شده است.

- `Value to store is not b bits long`: طول v داده شده به `_store` برابر با b نیست.
- `Invalid shift value`: مقدار p داده شده به `_left` یا `_right` بین 0 و b شامل این دوتا نیست.
- `Too many instructions`: تابع شما تلاش به اضافه کردن بیش از q دستور کرده است.

مثال‌ها

مثال ۱

فرض کنید $s = 0$ ، $n = 2$ ، $k = 1$ ، $q = 1000$. دو ورودی صحیح $a[0]$ و $a[1]$ وجود دارد، هر کدام $k = 1$ بیت دارند. قبل از اجرا برنامه، $r[0][0] = a[0]$ و $r[0][1] = a[1]$. سایر بیت‌ها در پردازنده 0 قرار داده شده‌اند. بعد از اجرا تمام دستورات برنامه، ما باید داشته باشیم $c[0] = r[0][0] = \min(a[0], a[1])$ ، که کوچک‌ترین $a[0]$ و $a[1]$ است.

تنها ۴ ورودی ممکن به برنامه وجود دارد:

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

ما می‌توانیم متوجه شویم که برای تمام ۴ حالت، $\min(a[0], a[1])$ برابر با AND-بیتی $a[0]$ و $a[1]$ است. از این رو، یک پاسخ ممکن، ساختن یک برنامه با فراخوانی تابع زیر است:

1. `_move(1, 0)`, که یک دستور برای کپی کردن $r[0]$ به $r[1]$ اضافه می‌کند.
2. `_right(1, 1, 1)`, که یک دستور اضافه می‌کند که تمام بیت‌های $r[1]$ را می‌گیرد، و آن‌ها را 1 بیت به راست شیفت می‌دهد، و سپس حاصل را در $r[1]$ ذخیره می‌کند. از آن جایی که هر عدد صحیح به طول 1 - بیت است، این عمل باعث می‌شود که $r[1][0]$ برابر با $a[1]$ شود.
3. `_and(0, 0, 1)`, که یک دستور اضافه می‌کند تا AND-بیتی $r[0]$ و $r[1]$ را بگیرد، سپس حاصل را در $r[0]$ ذخیره کند. بعد از اجرا این دستور، $r[0][0]$ برابر با AND-بیتی $r[0][0]$ و $r[1][0]$ قرار داده می‌شود، که برابر است با AND-بیتی $a[0]$ و $a[1]$ ، همان چیزی که می‌خواستیم.

مثال ۲

فرض کنید $s = 1$ ، $n = 2$ ، $k = 1$ ، $q = 1000$. همانند مثال قبلی، تنها ۴ ورودی ممکن برای این برنامه وجود دارد. برای تمام ۴ حالت، $\min(a[0], a[1])$ برابر با AND-بیتی $a[0]$ و $a[1]$ است، و $\max(a[0], a[1])$ برابر با OR-بیتی $a[0]$ و $a[1]$ است. یک پاسخ ممکن فراخوانی توابع زیر است:

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)`

بعد از اجرا این دستورات، $c[0] = r[0][0]$ شامل $\min(a[0], a[1])$ ، و $c[1] = r[0][1]$ شامل $\max(a[0], a[1])$ است.

محدودیت‌ها

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

زیرمسئله‌ها

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

ارزیاب نمونه

(این بخش را از نسخه انگلیسی صورت سوال بخوانید)