Bit Shift Register

Christopher, sang engineer, sedang mengerjakan sebuah prosesor komputer tipe baru.

Prosesor tersebut memiliki akses ke m b-bit sel memory berbeda (dengan $m = 100$ dan $b = 2000$), yang umumnya disebut dengan **register**, dan dinomori dari 0 sampai $m - 1$. Kita merepresentasikan register-register tersebut dengan $r[0], r[1], \ldots, r[m-1]$. Tiap register merupakan sebuah array dengan b buah bit, dinomori dari 0 (bit terkanan) sampai $b - 1$ (bit terkiri). Untuk setiap i $(0 \le i \le m - 1)$ dan setiap j $(0 \le j \le b - 1)$, kita merepresentasikan bit ke-j dari register i dengan $r[i][j]$.

Untuk setiap sekuens dari bit $d_0, d_1, \ldots, d_{l-1}$ (dengan panjang l berapapun), **nilai bilangan bulat** dari sekuens tersebut adalah $2^0 \cdot d_0 + 2^1 \cdot d_1 + \ldots + 2^{l-1} \cdot d_{l-1}$. Kita mengatakan **nilai bilangan bulat di register** i sebagai nilai bilangan bulat dari sekuens bit di register tersebut, dengan kata lain, nilainya adalah $2^0 \cdot r[i][0] + 2^1 \cdot r[i][1] + \ldots + 2^{b-1} \cdot r[i][b-1]$.

Prosesor tersebut memiliki 9 buah tipe **instruksi** yang dapat digunakan untuk memodifikasi bit-bit pada register. Tiap instruksi beroperasi pada satu atau lebih register dan menyimpan keluaran di salah satu register. Kita menggunakan $x := y$ untuk merepresentasikan operasi yang merubah nilai dari x menjadi sama dengan nilai y. Operasi-operasi yang dilakukan oleh tiap tipe instruksi dideskripsikan di bawah:

- $move(t, y)$: Menyalin bit array di register y ke register t. Untuk setiap j $(0 \le j \le b - 1)$, set $r[t][j] := r[y][j]$.
- $store(t, v)$: Set register t menjadi sama dengan v, dimana v merupakan sebuah array berisi b bit. Untuk setiap j $(0 \le j \le b - 1)$, set $r[t][j] := v[j]$.
- $and(t, x, y)$: Ambil bitwise-AND dari register x dan y, lalu simpan hasilnya di register t. Untuk setiap j $(0 \le j \le b - 1)$, set $r[t][j] := 1$ apabila **kedua** $r[x][j]$ dan $r[y][j]$ bernilai 1, dan set $r[t][j] := 0$ jika tidak.
- $or(t, x, y)$: Ambil bitwise-OR dari register x dan y, lalu simpan hasilnya di register t. Untuk setiap j $(0 \le j \le b - 1)$, set $r[t][j] := 1$ apabila **paling tidak salah satu** dari $r[x][j]$ dan $r[y][j]$ bernilai 1, dan set $r[t][j] := 0$ jika tidak.
- $xor(t, x, y)$: Ambil bitwise-XOR dari register x dan y, lalu simpan hasilnya di register t. Untuk setiap j $(0 \le j \le b - 1)$, set $r[t][j] := 1$ apabila **tepat satu** dari $r[x][j]$ dan $r[y][j]$ bernilai 1, dan set $r[t][j] := 0$ jika tidak.
- $not(t, x)$: Ambil bitwise-NOT dari register x, lalu simpan hasilnya di register t. Untuk setiap j $(0 \le j \le b - 1)$, set $r[t][j] := 1 - r[x][j]$.

- $left(t, x, p)$: Geser semua bit di register x ke kiri sebesar p, dan simpan hasilnya di register t. Hasil dari penggeseran bit di register x ke kiri sebesar p merupakan sebuah array v yang terdiri dari b bit. Untuk setiap j $(0 \le j \le b-1)$, $v[j] = r[x][j-p]$ apabila $j \ge p$, dan $v[j] = 0$ jika tidak. Untuk setiap j $(0 \le j \le b-1)$, set $r[t][j] := v[j]$.

- $right(t, x, p)$: Geser semua bit di register x ke kanan sebesar p, dan simpan hasilnya di register t. Hasil dari penggeseran bit di register x ke kanan sebesar p merupakan sebuah array v yang terdiri dari b bit. Untuk setiap j $(0 \le j \le b-1)$, $v[j] = r[x][j+p]$ apabila $j \le b-1-p$, dan $v[j] = 0$ jika tidak. Untuk setiap j $(0 \le j \le b-1)$, set $r[t][j] := v[j]$.

- $add(t, x, y)$: Jumlahkan nilai bilangan bulat dari register x dan register y, lalu simpan hasilnya di register t. Penjumlahan tersebut dilakukan dengan modulo 2^b. Formalnya, misalkan X sebagai nilai bilangan bulat yang disimpan di register x, dan Y sebagai nilai bilangan bulat yang disimpan di register y sebelum operasi. Misalkan T merupakan nilai bilangan bulat yang disimpan di register t setelah operasi. Apabila $X + Y < 2^b$, set bit-bit dari t, sedemikian sehingga $T = X + Y$. Jika tidak, set bit-bit dari t, sedemikian sehingga $T = X + Y - 2^b$.

Christopher ingin Anda menyelesaikan dua jenis tugas menggunakan prosesor baru tersebut. Tugas tersebut direpresentasikan dengan sebuah bilangan bulat s. Untuk kedua jenis tugas, Anda perlu membuat sebuah **program**, yakni serangkaian instruksi yang telah kita definisikan di atas.

Masukan dari program terdiri dari n buah bilangan bulat $a[0], a[1], \ldots, a[n-1]$, masing-masing terdiri dari k buah bit, dengan kata lain, $a[i] < 2^k$ ($0 \le i \le n-1$). Sebelum program dijalankan, semua angka masukan disimpan secara berurutan pada register 0, sehingga untuk setiap i $(0 \le i \le n-1)$ nilai bilangan bulat pada rangkaian k buah bit $r[0][i \cdot k], r[0][i \cdot k + 1], \ldots, r[0][(i+1) \cdot k - 1]$ sama dengan $a[i]$. Perhatikan bahwa $n \cdot k \le b$. Bit lainnya pada register 0 (yaitu, bit yang memiliki indeks di antara $n \cdot k$ dan $b-1$, inklusif) dan semua bit pada register lainnya diinisialisasi dengan angka 0.

Menjalankan sebuah program merupakan menjalankan instruksinya secara berurutan. Setelah instruksi terakhir dijalankan, **keluaran** dari program dihitung berdasarkan nilai akhir dari bit yang terdapat pada register 0. Jelasnya, keluaran program terdiri dari dari n buah bilangan bulat $c[0], c[1], \ldots, c[n-1]$, di mana untuk setiap i ($0 \le i \le n-1$), $c[i]$ adalah nilai bilangan bulat dari rangkaian bits $i \cdot k$ sampai $(i+1) \cdot k - 1$ pada register 0. Perhatikan bahwa seteleah menjalankan program, bit lainnya pada register 0 (dengan indeks sekurang-kurangnya $n \cdot k$) dan semua bit pada register lainnya boleh bernilai berapapun.

- Tugas yang pertama $(s = 0)$ adalah untuk mencari bilangan bulat terkecil di antara bilangan bulat masukan $a[0], a[1], \ldots, a[n-1]$. Jelasnya, $c[0]$ harus merupakan nilai terkecil dari $a[0], a[1], \ldots, a[n-1]$. Nilai dari $c[1], c[2], \ldots, c[n-1]$ boleh bernilai berapapun.

- Tugas yang kedua $(s = 1)$ adalah untuk mengurutkan bilangan bulat $a[0], a[1], \ldots, a[n-1]$ dengan urutan tidak menurun. Jelasnya, untuk setiap i ($0 \le i \le n-1$), $c[i]$ harus merupakan nilai terkecil ke- $1+i$ di antara $a[0], a[1], \ldots, a[n-1]$ (dengan kata lain, $c[0]$ merupakan bilangan bulat terkecil di antara bilangan bulat masukan).

Berikanlan Christopher program-program yang masing-masing terdiri dari paling banyak q buah instruksi, yang dapat menyelesaikan tugas-tugas ini.

Detail Implementasi

Anda perlu mengimplementasikan fungsi berikut:

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

- s: tipe dari tugas.
- n: banyaknya bilangan bulat di masukan.
- k: banyaknya bit di tiap bilangan bulat pada masukan.
- q: maksimum banyaknya instruksi yang dibolehkan.
- Fungsi ini dipanggil tepat sekali dan harus mengkonstruksi sebuah rangkaian instruksi untuk menyelesaikan task yang dibutuhkan.

Fungsi ini harus memanggil satu atau lebih dari fungsi-fungsi berikut untuk mengkonstruksi sebuah rangkaian instruksi:

```
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)
```

- Tiap fungsi masing-masing menambahkan sebuah instruksi $move(t, y)$, $store(t, v)$, $and(t, x, y)$, $or(t, x, y)$, $xor(t, x, y)$, $not(t, x)$, $left(t, x, p)$, $right(t, x, p)$ atau $add(t, x, y)$ ke program.
- Untuk semua instruksi yang relevan, t, x, y harus paling sedikit bernilai 0 dan paling banyak bernilai $m - 1$.
- Untuk semua instruksi yang relevan, t, x, y tidak harus berbeda untuk tiap pasang.
- Untuk instruksi $left$ dan $right$, p harus paling sedikit bernilai 0 dan paling banyak bernilai b.
- Untuk instruksi $store$, panjang dari v harus bernilai b.

Anda dapat juga memanggil fungsi berikut untuk membantu Anda dalam mengetes solusi Anda:

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

- Pemanggilan apapun ke fungsi ini akan diabaikan pada saat proses penilaian solusi Anda.
- Pada contoh *grader*, fungsi ini menambahkan sebuah operasi $print(t)$ ke program.
- Ketika contoh *grader* mendapati sebuah operasi $print(t)$ selama eksekusi sebuah program, dia mencetak n k-bit bilangan bulat yang dibentuk dari $n \cdot k$ bit pertama dari register t (lihat bagian "Contoh *Grader*" untuk lebih jelasnya).
- t harus memenuhi $0 \leq t \leq m - 1$.

- Panggilan apapun ke fungsi ini tidak menambah banyaknya instruksi yang dikonstruksi.

Setelah menambahkan instruksi terakhir, `construct_instructions` harus *return*. Program tersebut kemudian dievaluasi dengan beberapa kasus uji, masing-masing mengspesifikasikan sebuah masukan yang terdiri dari n k-bit bilangan bulat $a[0], a[1], \ldots, a[n-1]$. Solusi Anda lulus dari sebuah kasus uji apabila keluaran dari program $c[0], c[1], \ldots, c[n-1]$ untuk masukan yang diberikan memenuhi konsidi berikut:

- Apabila $s = 0$, $c[0]$ harus merupakan nilai terkecil dari $a[0], a[1], \ldots, a[n-1]$.
- Apabila $s = 1$, untuk setiap i ($0 \leq i \leq n-1$), $c[i]$ harus merupakan $1+i$-th bilangan terkecil dari $a[0], a[1], \ldots, a[n-1]$.

Penilaian dari solusi Anda dapat mengakibatkan di salah satu dari pesan error berikut:

- `Invalid index`: indeks register yang salah (mungkin negatif) diberikan sebagai parameter t, x or y untuk beberapa pemanggilan pada salah satu fungsi.
- `Value to store is not b bits long`: panjang dari v yang diberikan ke `append_store` tidak sama dengan b.
- `Invalid shift value`: nilai dari p yang diberikan ke `append_left` atau `append_right` tidak berada diantara 0 dan b inklusif.
- `Too many instructions`: fungsi Anda mencoba untuk menambahkan lebih dari q instruksi.

Contoh

Contoh 1

Misalkan $s = 0$, $n = 2$, $k = 1$, $q = 1000$. Terdapat dua bilangan bulat $a[0]$ dan $a[1]$, masing-masing memiliki $k = 1$ bit. Sebelum program dieksekusi, $r[0][0] = a[0]$ dan $r[0][1] = a[1]$. Semua bit lain pada prosesor diset menjadi 0. Setelah menjalankan instruksi ini, kita perlu menghasilkan $c[0] = r[0][0] = \min(a[0], a[1])$, yang merupakan nilai minimum dari $a[0]$ dan $a[1]$.

Hanya terdapat 4 nilai masukan yang mungkin:

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

Perhatikan bahwa untuk semua kasus, $\min(a[0], a[1])$ bernilai sama dengan bitwise-AND dari $a[0]$ dan $a[1]$. Oleh karena itu, solusi yang mungkin adalah untuk mengkonstruksi sebuah program dengan melakukan pemanggilan berikut:

1. `append_move(1, 0)`, yang menambahkan sebuah instruksi untuk menyalin $r[0]$ ke $r[1]$.
2. `append_right(1, 1, 1)`, yang menambahkan sebuah instruksi yang mengambil semua bit di $r[1]$, menggeser bit-bit tersebut ke kanan sebesar 1 bit, lalu menyimpan hasilnya kembali ke $r[1]$. Karena tiap bilangan bulat memiliki panjang 1-bit, hal ini mengakibatkan $r[1][0]$ bernilai sama dengan $a[1]$.

3. `append_and(0, 0, 1)`, yang menambahkan sebuah instruksi untuk mengambil bitwise-AND dari $r[0]$ dan $r[1]$, lalu menyimpan hasilnya di $r[0]$. Setelah instruksi ini dieksekusi, $r[0][0]$ diset menjadi bitwise-AND dari $r[0][0]$ dan $r[1][0]$, yang bernilai sama dengan bitwise-AND dari $a[0]$ dan $a[1]$, seperti yang diharapkan.

Contoh 2

Misalkan $s = 1,\ n = 2,\ k = 1,\ q = 1000$. Seperti pada contoh sebelumnya, hanya terdapat 4 kemungkinan masukan pada program. Untuk semua kasus, $\min(a[0], a[1])$ adalah bitwise-AND dari $a[0]$ dan $a[1]$, dan $\max(a[0], a[1])$ adalah bitwise-OR dari $a[0]$ dan $a[1]$. Salah satu kemungkinan solusi adalah dengan cara melakukan pemanggilan berikut:

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)`

Setelah menjalankan instruksi ini, $c[0] = r[0][0]$ mengandung $\min(a[0], a[1])$, dan $c[1] = r[0][1]$ mengandung $\max(a[0], a[1])$, yang mengurutkan masukan.

Batasan

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

Subsoal

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

Contoh *Grader*

Contoh *grader* membaca masukan dengan format berikut:

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

Diikuti dengan beberapa buah baris, masing-masing mendeskripsikan satu buah kasus uji. Setiap kasus uji disediakan dalam format berikut:

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

dan mendeskripsikan sebuah kasus uji yang masukannya terdiri dari n buah bilangan bulat $a[0], a[1], ..., a[n-1]$. Deskrispsi dari setiap kasus uji diikuti oleh sebuah baris yang hanya mengandung -1

Contoh *grader* awalnya memanggil `construct_instructions(s, n, k, q)`. Jika pemanggilan ini melanggar beberapa batasan yang dituliskan pada deskripsi soal, contoh *grader* akan mencetak salah satu pesan error yang terdapat pada akhir bagian "Detail Implementasi" dan berhenti. Jika tidak, contoh *grader* akan mencetak setiap instruksi ditambahi dengan `construct_instructions(s, n, k, q)` secara berurutan. Untuk instruksi $store$, v dicetak dari indeks 0 sampai indeks $b-1$.

Lalu, contoh *grader* memproses kasus uji secara berurutan. Untuk setiap kasus uji, contoh *grader* menjalankan program yang telah dibangun terhadap masukan dari kasus uji.

Untuk setiap operasi $print(t)$, andaikan $d[0], d[1], \ldots, d[n-1]$ adalah rangkaian bilangan bulat, sehingga untuk setiap i ($0 \leq i \leq n-1$), $d[i]$ adalah nilai bilangan bulat dari rangkaian bit $i \cdot k$ sampai $(i+1) \cdot k - 1$ pada register t (ketika operasi dijalankan). *Grader* mencetak rangkaian ini dengan format berikut: `register` t: $d[0]$ $d[1]$ $\ldots$ $d[n-1]$.

Setelah semua instruksi telah dijalankan, contoh *grader* mencetak keluaran dari program.

If $s = 0$, keluaran dari contoh *grader* untuk setiap kasus uji adalah dalam format berikut:

- $c[0]$.

If $s = 1$, keluaran dari contoh *grader* untuk setiap kasus uji adalah dalam format berikut:

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

Setelah menjalankan semua kasus uji, *grader* mencetak `number of instructions:` X dimana X adalah jumlah instruksi pada program anda.