Bit Shift Registers

Christopher là một kĩ sư đang làm việc trên một loại vi xử lý mới.

Bộ vi xử lý có m ô nhớ khác nhau, mỗi ô có độ dài b-bit (trong đó $m = 100$ và $b = 2000$), được gọi là **các thanh ghi**, và chúng được đánh số từ 0 đến $m - 1$. Ta kí hiệu các thanh ghi là $r[0], r[1], \ldots, r[m-1]$. Mỗi thanh ghi là một mảng b bit, được đánh số từ 0 (bit phải nhất) đến $b - 1$ (bit trái nhất). Với mỗi i $(0 \le i \le m - 1)$ và mỗi j $(0 \le j \le b - 1)$, chúng ta kí hiệu bit thứ j của thanh ghi i là $r[i][j]$.

Với một dãy bit $d_0, d_1, \ldots, d_{l-1}$ (độ dài l) bất kì, **giá trị số nguyên** của dãy bằng $2^0 \cdot d_0 + 2^1 \cdot d_1 + \ldots + 2^{l-1} \cdot d_{l-1}$. Chúng ta nói rằng **giá trị số nguyên được lưu trong một thanh ghi** i là giá trị số nguyên của dãy bit của nó, cụ thể, giá trị đó là $2^0 \cdot r[i][0] + 2^1 \cdot r[i][1] + \ldots + 2^{b-1} \cdot r[i][b-1]$.

Bộ vi xử lý có 9 loại **lệnh thực thi** dùng để thay đổi giá trị các bit trong các thanh ghi. Mỗi lệnh thực thi xử lý dữ liệu trên một hay nhiều thanh ghi và ghi kết quả vào một trong các thanh ghi. Trong bài này, chúng ta kí hiệu $x := y$ là phép thay đổi giá trị của x để giá trị của x bằng y. Các lệnh thực thi được mô tả như sau:

- $move(t, y)$: Sao chép dãy bit ở thanh ghi y vào thanh ghi t. Với mỗi j $(0 \le j \le b - 1)$, gán $r[t][j] := r[y][j]$.
- $store(t, v)$: Thanh ghi t được gán bằng v, trong đó v là một dãy b bit. Với mỗi j $(0 \le j \le b - 1)$, gán $r[t][j] := v[j]$.
- $and(t, x, y)$: Lấy giá trị phép AND của thanh ghi x và y, và ghi kết quả vào thanh ghi t. Với mỗi j $(0 \le j \le b - 1)$, gán $r[t][j] := 1$ nếu **cả hai giá trị** $r[x][j]$ và $r[y][j]$ đều bằng 1, và gán $r[t][j] := 0$ trong các trường hợp khác.
- $or(t, x, y)$: Lấy giá trị phép OR của thanh ghi x và y, và ghi kết quả vào thanh ghi t. Với mỗi j $(0 \le j \le b - 1)$, gán $r[t][j] := 1$ nếu **ít nhất một trong hai** giá trị $r[x][j]$ hoặc $r[y][j]$ bằng 1, và gán $r[t][j] := 0$ trong các trường hợp khác.
- $xor(t, x, y)$: Lấy giá trị phép XOR của thanh ghi x và y, và ghi kết quả vào thanh ghi t. Với mỗi j $(0 \le j \le b - 1)$, gán $r[t][j] := 1$ nếu **đúng một trong hai** giá trị $r[x][j]$ hoặc $r[y][j]$ bằng 1, và gán $r[t][j] := 0$ trong các trường hợp khác.
- $not(t, x)$: Lấy giá trị phép NOT của thanh ghi x, và ghi kết quả vào thanh ghi t. Với mỗi j $(0 \le j \le b - 1)$, gán $r[t][j] := 1 - r[x][j]$.
- $left(t, x, p)$: Dịch chuyển tất cả các bit của thanh ghi x sang trái p bước, và ghi kết quả vào thanh ghi t. Kết quả của việc dịch bit ở thanh ghi x sang trái p bước là dãy v gồm b bit. Với

mỗi j $(0 \le j \le b-1)$, $v[j] = r[x][j-p]$ nếu $j \ge p$, và $v[j] = 0$ trong các trường hợp khác. Với mỗi j $(0 \le j \le b-1)$, gán $r[t][j] := v[j]$.

- $right(t, x, p)$: Dịch chuyển tất cả các bit của thanh ghi x sang phải p bước, và ghi kết quả vào thanh ghi t. Kết quả của việc dịch bit ở thanh ghi x sang phải p bước là dãy v gồm b bit. Với mỗi j $(0 \le j \le b-1)$, $v[j] = r[x][j+p]$ nếu $j \le b-1-p$, và $v[j] = 0$ trong các trường hợp khác. Với mỗi j $(0 \le j \le b-1)$, gán $r[t][j] := v[j]$.

- $add(t, x, y)$: Cộng giá trị số nguyên lưu trong thanh ghi x và thanh ghi y, và ghi kết quả vào thanh ghi t. Phép cộng được thực hiện với việc lấy phần dư trong phép chia cho 2^b. Cụ thể, gọi X là các giá trị số nguyên được lưu trong thanh ghi x, và Y là giá trị số nguyên được lưu trong thanh ghi y trước khi thực hiện phép cộng. Gọi T là giá trị số nguyên lưu trong thanh ghi t sau khi thực hiện phép cộng. Nếu $X + Y < 2^b$, đặt dãy bit của thanh ghi t sao cho $T = X + Y$. Ngược lại, đặt dãy bit của thanh ghi t, sao cho $T = X + Y - 2^b$.

Christiopher muốn bạn giải hai loại bài toán bằng cách sử dụng bộ vi xử lý mới. Loại bài toán được đánh số một bởi số nguyên s. Với cả hai loại bài toán, bạn cần xây dựng một **chương trình**, là một dãy các lệnh thực thi được định nghĩa ở trên.

Dữ liệu đầu vào của chương trình gồm n số nguyên $a[0], a[1], \ldots, a[n-1]$, mỗi số gồm k-bit, có nghĩa là $a[i] < 2^k$ ($0 \le i \le n-1$). Trước khi chương trình được thực thi, tất cả các số nguyên trong dữ liệu đầu vào được lưu lần lượt vào thanh ghi 0, sao cho với mỗi i $(0 \le i \le n-1)$ giá trị số nguyên của dãy k bit $r[0][i \cdot k], r[0][i \cdot k + 1], \ldots, r[0][(i+1) \cdot k - 1]$ bằng với $a[i]$. Lưu ý rằng $n \cdot k \le b$. Các bit còn lại trong thanh ghi 0 (có nghĩa là các bit trong khoảng $n \cdot k$ và $b-1$, bao gồm cả hai đầu mút) và tất cả các bit ở trong các thanh ghi khác được khởi tạo bằng 0.

Việc chạy một chương trình là thực thi các lệnh theo thứ tự. Sau khi lệnh cuối cùng được thực thi, **kết quả đầu ra** của chương trình được tính dựa vào giá trị cuối cùng của các bit trong thanh ghi 0. Cụ thể, kết quả là một dãy n số nguyên $c[0], c[1], \ldots, c[n-1]$, trong đó với mỗi i ($0 \le i \le n-1$), $c[i]$ là giá trị số nguyên của dãy bit từ vị trí $i \cdot k$ đến vị trí $(i+1) \cdot k - 1$ của thanh ghi 0. Lưu ý rằng sau khi chạy chương trình, các bit còn lại của thanh ghi 0 (các bit từ vị trí $n \cdot k$ trở đi) và tất cả các bit ở các thanh ghi khác có thể nhận giá trị bất kỳ.

- Loại bài toán đầu tiên $(s = 0)$ là tìm số nguyên nhỏ nhất trong các số $a[0], a[1], \ldots, a[n-1]$. Cụ thể, $c[0]$ phải là giá trị nhỏ nhất của $a[0], a[1], \ldots, a[n-1]$. Các giá trị của $c[1], c[2], \ldots, c[n-1]$ có thể là bất kì.

- Loại bài toán thứ hai $(s = 1)$ là sắp xếp dãy số nguyên $a[0], a[1], \ldots, a[n-1]$ theo thứ tự không giảm. Cụ thể, với mỗi i ($0 \le i \le n-1$), $c[i]$ cần bằng số nguyên nhỏ thứ $1+i$ trong các số $a[0], a[1], \ldots, a[n-1]$ (có nghĩa là $c[0]$ là số nguyên nhỏ nhất trong các kết quả đầu vào).

Hãy giúp Christopher với các chương trình, mỗi chương trình gồm không quá q lệnh thực thi, để có thể giải được hai loại bài toán này.

Chi tiết cài đặt

Bạn cần cài đặt hàm sau:

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

- s: loại bài toán.
- n: số lượng số nguyên trong dữ liệu đầu vào.
- k: số lượng bit của mỗi số nguyên đầu vào.
- q: số lệnh thực thi tối đa được phép sử dụng.
- Hàm này được gọi đúng một lần và cần phải tạo ra một dãy các lệnh thực thi để giải được bài toán.

Hàm này cần phải gọi một hay một số các hàm sau để tạo ra dãy các lệnh thực thi:

```
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)
```

- Mỗi lần gọi các hàm sẽ thêm một lệnh thực thi $move(t, y)$ $store(t, v)$, $and(t, x, y)$, $or(t, x, y)$, $xor(t, x, y)$, $not(t, x)$, $left(t, x, p)$, $right(t, x, p)$ hay $add(t, x, y)$ tương ứng vào cuối chương trình.
- Với tất cả các lệnh thực thi, t, x, y phải có giá trị nhỏ nhất là 0 và lớn nhất là $m - 1$.
- Với tất cả các lệnh thực thi, t, x, y có thể không cần đôi một khác nhau.
- Với các lệnh $left$ và $right$, p phải có giá trị nhỏ nhất là 0 và lớn nhất là b.
- Với các lệnh $store$, độ dài của v phải bằng b.

Bạn có thể gọi hàm sau để giúp kiểm thử lời giải:

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

- Các lời gọi hàm này sẽ được bỏ qua khi chấm bài.
- Trong trình chấm mẫu, hàm này thêm một lệnh $print(t)$ vào cuối chương trình.
- Khi trình chấm mẫu gặp lệnh $print(t)$ trong quá trình thực thi chương trình, trình chấm mẫu sẽ in ra dãy n số nguyên k-bit được tạo ra bởi $n \cdot k$ bit đầu tiên của thanh ghi t (xem chi trong phần "Trình chấm mẫu").
- t phải thỏa mãn $0 \leq t \leq m - 1$.
- Các lời gọi đến hàm này không làm tăng số lượng lệnh thực thi.

Sau khi thêm vào lệnh thực thi cuối cùng, hàm `construct_instructions` cần phải kết thúc. Sau đó chương trình sẽ được chấm dựa trên một số bộ dữ liệu, mỗi bộ dữ liệu gồm n số nguyên k-bit $a[0], a[1], \ldots, a[n-1]$. Chương trình của bạn được tính là qua được một bộ dữ liệu nếu kết quả đầu ra $c[0], c[1], \ldots, c[n-1]$ ứng với dữ liệu đầu vào thỏa mãn các điều kiện sau:

- Nếu $s = 0$, $c[0]$ bằng giá trị nhỏ nhất của $a[0], a[1], \ldots, a[n-1]$.
- Nếu $s = 1$, với mỗi i ($0 \leq i \leq n-1$), $c[i]$ bằng số nhỏ thứ $1 + i$ trong số các giá trị $a[0], a[1], \ldots, a[n-1]$.

Kết quả chương trình của bạn có thể nhận một trong các lỗi sau:

- `Invalid index`: một lời gọi hàm nào đó với chỉ số thanh ghi cho bởi tham số t, x hoặc y sai (có thể âm)
- `Value to store is not b bits long`: độ dài của v cho lời gọi hàm `append_store` không bằng b.
- `Invalid shift value`: giá trị p cho hàm `append_left` hoặc `append_right` không nằm trong khoảng 0 và b, bao gồm cả hai đầu mút.
- `Too many instructions`: chương trình của bạn sử dụng quá q lệnh thực thi.

Các ví dụ

Ví dụ 1

Xét $s = 0$, $n = 2$, $k = 1$, $q = 1000$. Có hai số nguyên $a[0]$ và $a[1]$, mỗi số gồm $k = 1$ bit. Trước khi chương trình được thực thi, $r[0][0] = a[0]$ và $r[0][1] = a[1]$. Tất cả các bit khác trong bộ vi xử lý được gán bằng 0. Sau khi tất cả các lệnh thực thi của chương trình được thực hiện, ta cần phải có $c[0] = r[0][0] = \min(a[0], a[1])$, là giá trị nhỏ hơn giữa $a[0]$ và $a[1]$.

Chỉ có 4 khả năng của dữ liệu đầu vào:

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

Ta có thể nhận thấy rằng trong cả 4 trường hợp, $\min(a[0], a[1])$ bằng với giá trị của phép AND giữa $a[0]$ và $a[1]$. Vì vậy, một lời giải hợp lệ có thể gọi các hàm như sau :

1. `append_move(1, 0)`, thêm một lệnh thực thi sao chép thanh ghi $r[0]$ sang thanh ghi $r[1]$.
2. `append_right(1, 1, 1)`, thêm một lệnh thực thi lấy tất cả các bit của $r[1]$, dịch chúng sang bên phải 1 bit, và ghi kết quả trở lại $r[1]$. Vì mỗi số có độ dài 1-bit, kết quả ghi trong $r[1][0]$ bằng với giá trị của $a[1]$.
3. `append_and(0, 0, 1)`, thêm một lệnh thực thi lấy kết quả phép AND của $r[0]$ và $r[1]$, rồi ghi kết quả vào $r[0]$. Sau khi lệnh này được thực thi, $r[0][0]$ sẽ được nhận giá trị là kết quả của phép AND giữa $r[0][0]$ và $r[1][0]$, nó cũng bằng với kết quả của phép AND giữa $a[0]$ và $a[1]$ như chúng ta mong muốn.

Ví dụ 2

Xét $s = 1$, $n = 2$, $k = 1$, $q = 1000$. Giống như ví dụ trước, chỉ có 4 khả năng cho dữ liệu đầu vào. Với cả 4 trường hợp, $\min(a[0], a[1])$ là giá trị của phép AND giữa $a[0]$ và $a[1]$, và

$\max(a[0], a[1])$ là giá trị của phép OR giữa $a[0]$ và $a[1]$. Một lời giải hợp lệ có thể gọi các hàm như sau:

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)`

Sau khi thực thi những lệnh trên, $c[0] = r[0][0]$ chứa $\min(a[0], a[1])$, và $c[1] = r[0][1]$ chứa $\max(a[0], a[1])$, tương đương với sắp xếp dữ liệu đầu vào.

Các ràng buộc

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

Các subtask

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

Trình chấm mẫu

Trình chấm mẫu đọc dữ liệu theo định dạng sau:

- dòng 1 : $s\ n\ k\ q$

Trong một số dòng tiếp theo, mỗi dòng mô tả một bộ dữ liệu. Mỗi bộ dữ liệu được ghi theo định dạng sau:

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

mô tả một bộ dữ liệu gồm n số nguyên $a[0], a[1], \ldots, a[n-1]$. Các bộ dữ liệu được kết thúc với một dòng ghi duy nhất số -1.

Trình chấm mẫu đầu tiên sẽ gọi hàm `construct_instructions(s, n, k, q)`. Nếu lời gọi này vi phạm một số ràng buộc được cho trong bài toán, trình chấm mẫu sẽ in ra một trong số các lỗi được liệt kê ở phần cuối của phần "Chi tiết cài đặt" và kết thúc. Ngược lại, đầu tiên trình chấm mẫu tuần tự in ra mỗi lệnh thực thi được gọi bởi `construct_instructions(s, n, k, q)`. Với lệnh $store$, v được in ra theo thứ tự từ 0 đến $b-1$.

Sau đó, trình chấm mẫu tuần tự xử lý các bộ dữ liệu. Với mỗi bộ dữ liệu, chương trình được tạo ra sẽ được chạy với bộ dữ liệu đó.

Với mỗi lệnh $print(t)$, gọi $d[0], d[1], \ldots, d[n-1]$ là một dãy các số nguyên, sao cho với mỗi i ($0 \leq i \leq n-1$), $d[i]$ là giá trị số nguyên của dãy bit từ $i \cdot k$ đến $(i+1) \cdot k - 1$ của thanh ghi t (khi mà lệnh đã được thực thi). Trình chấm mẫu in dãy số này theo định dạng sau: `register` t: $d[0]\ d[1]\ \ldots\ d[n-1]$.

Khi mà tất cả các lệnh được thực thi, trình chấm mẫu in ra kết quả của chương trình.

Nếu $s = 0$, kết quả của trình chấm mẫu với mỗi bộ dữ liệu được in theo định dạng sau:

- $c[0]$.

Nếu $s = 1$, kết quả của trình chấm mẫu với mỗi bộ dữ liệu được in theo định dạng sau:

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

Sau khi xử lý xong tất cả các bộ dữ liệu, trình chấm mẫu in ra `number of instructions:` X trong đó X là số lệnh thực thi của chương trình của bạn.