

# Bit Shift Registers

Christopher inginerul lucrează la un nou tip de procesor pentru computer.

Procesorul are acces la  $m$  celule de memorie distincte a câte  $b$  biți fiecare (unde  $m = 100$  și  $b = 2000$ ). Acestea sunt denumite **registri** și sunt numerotate de la  $0$  la  $m - 1$ . Denotăm regiștri cu  $r[0], r[1], \dots, r[m - 1]$ . Fiecare registru este un array de  $b$  biți, numerotat de la  $0$  (cel mai din dreapta bit) la  $b - 1$  (cel mai din stânga bit). Pentru fiecare  $i$  ( $0 \leq i \leq m - 1$ ) și fiecare  $j$  ( $0 \leq j \leq b - 1$ ), denotăm al  $j$ -lea bit al registrului  $i$  cu  $r[i][j]$ .

Pentru orice secvență de biți  $d_0, d_1, \dots, d_{l-1}$  (de lungime arbitrară  $l$ ), **valoarea întreagă** a secvenței este egală cu  $2^0 \cdot d_0 + 2^1 \cdot d_1 + \dots + 2^{l-1} \cdot d_{l-1}$ . Spunem că **valoarea întreagă salvată în registru**  $i$  este valoarea întreagă a secvenței de biți ale acestuia, adică este  $2^0 \cdot r[i][0] + 2^1 \cdot r[i][1] + \dots + 2^{b-1} \cdot r[i][b - 1]$ .

Procesorul are 9 tipuri de **instrucțiuni** care pot fi folosite pentru a modifica biții din regiștri. Fiecare instrucțiune operează pe unul sau mai mulți regiștri și salvează rezultatul în unul dintre regiștri. În cele ce urmează folosim  $x := y$  pentru a denota operația care modifică valoarea lui  $x$  astfel încât ea să devină egală cu  $y$ . Operațiile efectuate de fiecare tip de instrucțiune sunt descrise mai jos.

- $move(t, y)$ : Copiază array-ul de biți din registrul  $y$  în registrul  $t$ . Pentru fiecare  $j$  ( $0 \leq j \leq b - 1$ ), setează  $r[t][j] := r[y][j]$ .
- $store(t, v)$ : Setează registrul  $t$  să fie egal cu  $v$ , unde  $v$  este un array de  $b$  biți. Pentru fiecare  $j$  ( $0 \leq j \leq b - 1$ ), setează  $r[t][j] := v[j]$ .
- $and(t, x, y)$ : Calculează operația pe biți AND a regiștrilor  $x$  și  $y$ , și salvează rezultatul în registrul  $t$ . Pentru fiecare  $j$  ( $0 \leq j \leq b - 1$ ), setează  $r[t][j] := 1$  dacă **ambii** biți  $r[x][j]$  și  $r[y][j]$  sunt 1, și setează  $r[t][j] := 0$  altfel.
- $or(t, x, y)$ : Calculează operația pe biți OR a regiștrilor  $x$  și  $y$ , și salvează rezultatul în registrul  $t$ . Pentru fiecare  $j$  ( $0 \leq j \leq b - 1$ ), setează  $r[t][j] := 1$  dacă **cel puțin unul** dintre biții  $r[x][j]$  și  $r[y][j]$  este 1, și setează  $r[t][j] := 0$  altfel.
- $xor(t, x, y)$ : Calculează operația pe biți XOR a regiștrilor  $x$  și  $y$ , și salvează rezultatul în registrul  $t$ . Pentru fiecare  $j$  ( $0 \leq j \leq b - 1$ ), setează  $r[t][j] := 1$  dacă **exact unul** dintre biții  $r[x][j]$  și  $r[y][j]$  este 1, și setează  $r[t][j] := 0$  altfel.
- $not(t, x)$ : Calculează operația pe biți NOT a registrului  $x$ , și salvează rezultatul în registrul  $t$ . Pentru fiecare  $j$  ( $0 \leq j \leq b - 1$ ), setează  $r[t][j] := 1 - r[x][j]$ .
- $left(t, x, p)$ : Deplasează la stânga biții registrului  $x$  cu  $p$  poziții, și salvează rezultatul în registrul  $t$ . Rezultatul deplasării la stânga a biților din registrul  $x$  cu  $p$  poziții este un array  $v$

alcătuit din  $b$  biți. Pentru fiecare  $j$  ( $0 \leq j \leq b-1$ ),  $v[j] = r[x][j-p]$  dacă  $j \geq p$ , și  $v[j] = 0$  altfel. Pentru fiecare  $j$  ( $0 \leq j \leq b-1$ ), setează  $r[t][j] := v[j]$ .

- $right(t, x, p)$ : Deplasează la dreapta biții registrului  $x$  cu  $p$  poziții, și salvează rezultatul în registrul  $t$ . Rezultatul deplasării la dreapta a biților din registrul  $x$  cu  $p$  poziții este un array  $v$  alcătuit din  $b$  biți. Pentru fiecare  $j$  ( $0 \leq j \leq b-1$ ),  $v[j] = r[x][j+p]$  dacă  $j \leq b-1-p$ , și  $v[j] = 0$  altfel. Pentru fiecare  $j$  ( $0 \leq j \leq b-1$ ), setează  $r[t][j] := v[j]$ .
- $add(t, x, y)$ : Însumează valorile întregi salvate în regiștri  $x$  și  $y$ , și salvează rezultatul în registrul  $t$ . Adunarea este efectuată modulo  $2^b$ . Formal, fie  $X$  valoarea întreagă salvată în registrul  $x$ , și  $Y$  valoarea întreagă salvată în registrul  $y$ , înainte de operație. Fie  $T$  valoarea întreagă salvată în registrul  $t$ , după operație. Dacă  $X + Y < 2^b$ , setează biții lui  $t$  astfel încât  $T = X + Y$ . Altfel, setează biții lui  $t$  astfel încât  $T = X + Y - 2^b$ .

Christopher ar vrea să rezolve două tipuri de task-uri folosind noul procesor. Tipul unui task este denotat cu un număr întreg  $s$ . Pentru ambele tipuri de task-uri, voi trebuie să produceți un **program**, adică o secvență de instrucțiuni definite mai sus.

**Input-ul** programului este alcătuit din  $n$  numere întregi  $a[0], a[1], \dots, a[n-1]$ , fiecare având  $k$  biți, adică  $a[i] < 2^k$  ( $0 \leq i \leq n-1$ ). Înainte ca programul să fie executat, toate numerele din input sunt salvate secvențial în registrul 0 astfel încât pentru fiecare  $i$  ( $0 \leq i \leq n-1$ ) valoarea întreagă a secvenței de  $k$  biți  $r[0][i \cdot k], r[0][i \cdot k + 1], \dots, r[0][(i+1) \cdot k - 1]$  este egală cu  $a[i]$ . Notați că  $n \cdot k \leq b$ . Toți ceilalți biți din registrul 0 (adică cei cu indicii între  $n \cdot k$  și  $b-1$ , inclusiv) și toți biții din toți ceilalți regiștri sunt inițializați la 0.

Rularea unui program constă în executarea instrucțiunilor sale în ordine. După ce ultima instrucțiune este executată, **output-ul** programului este calculat în funcție de valorile finale ale biților registrului 0. Specific, output-ul este o secvență de  $n$  numere întregi  $c[0], c[1], \dots, c[n-1]$ , unde pentru fiecare  $i$  ( $0 \leq i \leq n-1$ ),  $c[i]$  este valoarea întreagă a secvenței alcătuite din biții de la  $i \cdot k$  la  $(i+1) \cdot k - 1$  din registrul 0. Notați că după ce programul este rulat, biții rămași din registrul 0 (cei cu indici cel puțin  $n \cdot k$ ) și toți ceilalți regiștri pot avea valori arbitrare.

- Primul task ( $s = 0$ ) este să găsiți cel mai mic număr dintre numerele din input  $a[0], a[1], \dots, a[n-1]$ . Specific,  $c[0]$  trebuie să fie minimul dintre  $a[0], a[1], \dots, a[n-1]$ . Valorile lui  $c[1], c[2], \dots, c[n-1]$  pot fi arbitrare.
- Al doilea task ( $s = 1$ ) este să sortați numerele din input  $a[0], a[1], \dots, a[n-1]$  în ordine crescătoare. Specific, pentru fiecare  $i$  ( $0 \leq i \leq n-1$ ),  $c[i]$  trebuie să fie egal cu al  $1+i$ -lea cel mai mic număr dintre  $a[0], a[1], \dots, a[n-1]$  (adică  $c[0]$  este cel mai mic număr dintre numerele din input).

Furnizați-i lui Christopher programe, fiecare fiind alcătuit din cel mult  $q$  instrucțiuni, care să rezolve aceste task-uri.

## Detalii de Implementare

Trebuie să implementați următoarea procedură:

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

- $s$ : tipul task-ului.
- $n$ : numărul de numere întregi din input.
- $k$ : numărul de biți pentru fiecare număr întreg din input.
- $q$ : numărul maxim de instrucțiuni permise.
- Această procedură este apelată exact o dată și ar trebui să construiască o secvență de instrucțiuni pentru a efectua task-ul necesar.

Această procedură ar trebui să apeleze una sau mai multe proceduri de mai jos pentru a construi o secvență de instrucțiuni:

```
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)
```

- Fiecare procedură adaugă o instrucțiune  $move(t, y)$ ,  $store(t, v)$ ,  $and(t, x, y)$ ,  $or(t, x, y)$ ,  $xor(t, x, y)$ ,  $not(t, x)$ ,  $left(t, x, p)$ ,  $right(t, x, p)$  sau  $add(t, x, y)$  la program, respectiv.
- Pentru toate instrucțiunile relevante,  $t$ ,  $x$ ,  $y$  trebuie să fie de cel puțin 0 și cel mult  $m - 1$ .
- Pentru toate instrucțiunile relevante,  $t$ ,  $x$ ,  $y$  nu sunt neapărat distincte două câte două.
- Pentru instrucțiunile  $left$  și  $right$ ,  $p$  trebuie să fie cel puțin 0 și cel mult  $b$ .
- Pentru instrucțiunile  $store$ , lungimea lui  $v$  trebuie să fie  $b$ .

De asemenea, puteți apela următoarea procedură pentru a vă testa soluția:

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

- Orice apel a acestei proceduri va fi ignorat în timpul evaluării soluției dvs.
- În exemplul de grader, această procedură adaugă programului o operație  $print(t)$ .
- Când exemplul de grader întâlnește o operație  $print(t)$  în timpul execuției unui program, acesta tipărește  $n$  numere întregi de  $k$  biți formate din primii  $n \cdot k$  biți din registrul  $t$  (vezi Secțiunea „Exemplu de Grader” pentru detalii).
- $t$  trebuie să satisfacă condiția  $0 \leq t \leq m - 1$ .
- Orice apel al acestei proceduri nu se adaugă la numărul de instrucțiuni construite.

După adăugarea ultimei instrucțiuni, `construct_instructions` ar trebui să returneze. Programul este apoi evaluat pe un anumit număr de cazuri de testare, fiecare specificând un input constând din  $n$  numere de  $k$  biți  $a[0], a[1], \dots, a[n - 1]$ . Soluția dvs. trece un caz de testare dacă output-ul programului  $c[0], c[1], \dots, c[n - 1]$  pentru input-ul furnizat îndeplinește următoarele condiții:

- Dacă  $s = 0$ ,  $c[0]$  ar trebui să fie cea mai mică valoare dintre  $a[0], a[1], \dots, a[n-1]$ .
- Dacă  $s = 1$ , pentru fiecare  $i$  ( $0 \leq i \leq n-1$ ),  $c[i]$  ar trebui să fie al  $1+i$ -lea cel mai mic întreg dintre  $a[0], a[1], \dots, a[n-1]$ .

Evaluarea soluției dvs. poate duce la unul dintre următoarele mesaje de eroare:

- Invalid index: a fost transmis ca parametru  $t$ ,  $x$  sau  $y$  un index de registru incorect (posibil negativ) pentru un apel al uneia dintre proceduri.
- Value to store is not  $b$  bits long: lungimea lui  $v$  atribuită la `append_store` nu este egală cu  $b$ .
- Invalid shift value: valoarea lui  $p$  atribuită la `append_left` sau `append_right` nu este între 0 și  $b$  inclusiv.
- Too many instructions: procedura dvs. a încercat să adauge mai mult de  $q$  instrucțiuni.

## Exemple

### Exemplul 1

Presupunem că  $s = 0$ ,  $n = 2$ ,  $k = 1$ ,  $q = 1000$ . Există două numere întregi la intrare  $a[0]$  și  $a[1]$ , fiecare având  $k = 1$  biți. Înainte de executarea programului,  $r[0][0] = a[0]$  și  $r[0][1] = a[1]$ . Toți ceilalți biți din procesor sunt setați la 0. După executarea tuturor instrucțiunilor din program, trebuie să avem  $c[0] = r[0][0] = \min(a[0], a[1])$ , care este minimul dintre  $a[0]$  și  $a[1]$ .

Există doar 4 cazuri posibile pentru input-ul programului:

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

Putem observa că pentru toate cele 4 cazuri,  $\min(a[0], a[1])$  este egal cu operația pe biți AND dintre  $a[0]$  și  $a[1]$ . Prin urmare, o posibilă soluție este de a construi un program prin efectuarea următoarelor apeluri:

1. `append_move (1, 0)`, care adaugă o instrucțiune de copiere a  $r[0]$  în  $r[1]$ .
2. `append_right (1, 1, 1)`, care adaugă o instrucțiune care ia toți biții din  $r[1]$ , îi deplasează la dreapta cu 1 bit, apoi stochează rezultatul înapoi în  $r[1]$ . Deoarece fiecare număr întreg are o lungime de 1-bit, aceasta rezultă în  $r[1][0]$  să fie egal cu  $a[1]$ .
3. `append_and (0, 0, 1)`, care adaugă o instrucțiune pentru a face operația pe biți AND dintre  $r[0]$  și  $r[1]$ , apoi stochează rezultatul în  $r[0]$ . După ce această instrucțiune este executată,  $r[0][0]$  este setat la operația pe biți AND dintre  $r[0][0]$  și  $r[1][0]$ , care este egal cu operația pe biți AND dintre  $a[0]$  și  $a[1]$ , așa cum ne-am dorit.

### Exemplul 2

Suppose  $s = 1$ ,  $n = 2$ ,  $k = 1$ ,  $q = 1000$ . Ca și în exemplul precedent, avem doar 4 cazuri posibile pentru input-ul programului. Pentru toate cele 4 cazuri,  $\min(a[0], a[1])$  este operația pe biți AND dintre  $a[0]$  și  $a[1]$ , și  $\max(a[0], a[1])$  este operația pe biți OR dintre  $a[0]$  și  $a[1]$ . O posibilă soluție este să efectuați următoarele apeluri:

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)`

După executarea acestor instrucțiuni,  $c[0] = r[0][0]$  conține  $\min(a[0], a[1])$ , și  $c[1] = r[0][1]$  conține  $\max(a[0], a[1])$ , care sortează input-ul.

## Restricții

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

## Subtasks

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

## Exemplul de Grader

Grader-ul citește input-ul după următorul format:

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

Aceasta este urmată de mai multe linii, fiecare descriind un caz de testare. Fiecare caz de testare este furnizat după următorul format:

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

și descrie un caz de testare al cărui input este alcătuit din  $n$  numere întregi  $a[0], a[1], \dots, a[n-1]$ . După descrierea tuturor cazurilor de testare urmează încă o linie care conține  $-1$ .

Grader-ul apelează mai întâi `construct_instructions(s, n, k, q)`. Dacă această apelare încalcă vreo constrângere descrisă în enunțul problemei, grader-ul printează unul dintre mesajele de eroare listate la sfârșitul secțiunii "Detalii de Implementare" și termină execuția. Altfel, grader-ul mai întâi printează fiecare instrucțiune adăugată de `construct_instructions(s, n, k, q)` în ordine. Pentru instrucțiunea `store, v` este printat de la indexul  $0$  la indexul  $b-1$ .

Apoi, grader-ul procesează cazurile de testare în ordine. Pentru fiecare caz de testare, execută programul construit de voi pe input-ul cazului de testare.

Pentru fiecare operație `print(t)`, fie  $d[0], d[1], \dots, d[n-1]$  secvența de numere astfel încât pentru fiecare  $i$  ( $0 \leq i \leq n-1$ ),  $d[i]$  este valoarea întreagă a secvenței de biți de la  $i \cdot k$  la  $(i+1) \cdot k - 1$  a registrului  $t$  (la momentul executării operației). Grader-ul printează această secvență după următorul format: `register t: d[0] d[1] ... d[n-1]`.

Odată ce toate instrucțiunile au fost executate, grader-ul printează output-ul programului.

Dacă  $s = 0$ , output-ul grader-ului pentru fiecare caz de testare este după următorul format:

- $c[0]$ .

Dacă  $s = 1$ , output-ul grader-ului pentru fiecare caz de testare este după următorul format:

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

După ce toate cazurile de testare sunt executate, grader-ul printează `number of instructions: X` unde  $X$  este numărul de instrucțiuni ale programului vostru.