Bit Shift Registers

Kristofer je inženjer koji razvija novu vrstu procesora.

Procesor može pristupiti b-bitnim memorijskim ćelijama koje su međusobno različite i kojih ima m (gdje je $m = 100$ i $b = 2000$), koja nazivamo **registrima** i koje su označene brojevima od 0 do $m - 1$. Označimo registre sa $r[0], r[1], \ldots, r[m-1]$. Svaki registar je niz od b bita, označenih od 0 (krajni desni bit) do $b - 1$ (krajni lijevi bit). Za sve i $(0 \leq i \leq m - 1)$ i sve j $(0 \leq j \leq b - 1)$, označimo sa $r[i][j]$ bit j registra i.

Za svaki niz bita $d_0, d_1, \ldots, d_{l-1}$ (proizvoljne dužine l), **cjelobrojna vrijednost** niza je $2^0 \cdot d_0 + 2^1 \cdot d_1 + \ldots + 2^{l-1} \cdot d_{l-1}$. Kažemo da je **cjelobrojna vrijednost sačuvana u registru** i jednaka cjelobrojnoj vrijednosti niza bitova u registru, tj. $2^0 \cdot r[i][0] + 2^1 \cdot r[i][1] + \ldots + 2^{b-1} \cdot r[i][b-1]$.

Procesor ima 9 tipova **instrukcija** pomoću kojih može modifikovati sadržaj registara. Svaka instrukcija koristi jedan ili više registara i čuva rezultat u nekom od registara. U nastavku, koristićemo $x := y$ da označimo operaciju promjene vrijednosti x u y. Značenje svake od instrukcija je sljedeće:

- $move(t, y)$: Kopira niz bitova iz registra y u registar t. Za sve j $(0 \leq j \leq b - 1)$, $r[t][j] := r[y][j]$.
- $store(t, v)$: Registar t biće jednak v, gdje je v niz od b bita. Za sve j $(0 \leq j \leq b - 1)$, $r[t][j] := v[j]$.
- $and(t, x, y)$: Primjenjuje operaciju bitwise-AND nad registrima x i y i smješta rezultat u registar t. Za sve j $(0 \leq j \leq b - 1)$, $r[t][j] := 1$ ako su **oba** $r[x][j]$ i $r[y][j]$ jednaka 1 i $r[t][j] := 0$ u suprotnom.
- $or(t, x, y)$: Primjenjuje operaciju bitwise-OR nad registrima x i y i smješta rezultat u registar t. Za sve j $(0 \leq j \leq b - 1)$, $r[t][j] := 1$ ako je **bar jedan** od $r[x][j]$ i $r[y][j]$ jednak 1 i $r[t][j] := 0$ u suprotnom.
- $xor(t, x, y)$: Primjenjuje operaciju bitwise-XOR nad registrima x i y, i smješta rezultat u registar t. Za sve j $(0 \leq j \leq b - 1)$, $r[t][j] := 1$ ako je **tačno jedan** od $r[x][j]$ i $r[y][j]$ jednak 1 i $r[t][j] := 0$ u suprotnom.
- $not(t, x)$: Primjenjuje operaciju bitwise-NOT nad registrom x i smješta rezultat u registar t. Za sve j $(0 \leq j \leq b - 1)$, $r[t][j] := 1 - r[x][j]$.
- $left(t, x, p)$: Pomjera ulijevo bitove registra x za p pozicija i smješta rezultat u registar t. Rezultat pomjeranja bitova registra x ulijevo za p pozicija je niz v koji ima b bita. Za sve j $(0 \leq j \leq b - 1)$, $v[j] = r[x][j - p]$ ako je $j \geq p$ i $v[j] = 0$ u suprotnom. Za sve j $(0 \leq j \leq b - 1)$, set $r[t][j] := v[j]$.

- $right(t, x, p)$: Pomjera nadesno bitove registra x za p pozicija i smješta rezultat u registar t. Rezultat pomjeranja bitova registra x nadesno za p pozicija je niz v koji ima b bita. Za sve j $(0 \leq j \leq b-1)$, $v[j] = r[x][j+p]$ if $j \leq b - 1 – p$ i $v[j] = 0$ inače. Za sve j $(0 \leq j \leq b-1)$, $r[t][j] := v[j]$.
- $add(t, x, y)$: Sabira cjelobrojne vrijednosti sačuvane u registrima x i y i smješta rezultat u registar t. Sabiranje se obavlja po modulu 2^b. Formalno, neka je X cjelobrojna vrijednost sačuvana u registru x i Y cjelobrojna vrijednost sačuvana u registru y prije izvršenja ove operacije. Neka je T cjelobrojna vrijednost sačuvana u registru t nakon operacije. Ako je $X + Y < 2^b$, postaviti bitove registra t tako da je $T = X + Y$. U suprotnom, postaviti bitove registra t tako da je $T = X + Y - 2^b$.

Kristofer želi da riješi dva tipa zadataka primjenom novog procesora. Tip zadatka je označen brojem s. Za oba tipa zadatka potrebno je kreirati **progam** tj. niz instrukcija koje smo ranije opisali.

Program ima **ulaz** koji sadrži n cijelih brojeva $a[0], a[1], \ldots, a[n-1]$, pri čemu svaki broj ima k bita, tj. $a[i] < 2^k$ ($0 \leq i \leq n-1$). Prije pokretanja programa, svi ulazni brojevi su smješteni redom u registar 0, tako da za sve i $(0 \leq i \leq n-1)$ cjelobrojna vrijednost niza od k bita $r[0][i \cdot k], r[0][i \cdot k + 1], \ldots, r[0][(i+1) \cdot k - 1]$ je jednako $a[i]$. Obratite pažnju da je $n \cdot k \leq b$. Svi ostali bitovi u registru su 0 (tj. oni sa indeksima $n \cdot k$ i $b - 1$, uključivo) a svi bitovi od drugim registrima su postavljeni na 0.

Pokretanje programa se sastoji izvršavanja instrukcija onim redom kako su napisane. Nakon izvršavanja posljednje instrukcije, **izlaz** programa se izračunava na osnovu konačnih vrijednosti bitova u registru 0. Naime, izlaz je niz od n cijelih brojeva $c[0], c[1], \ldots, c[n-1]$, gdje za sve i ($0 \leq i \leq n-1$), $c[i]$ predstavlja cjelobrojnu vrijednost niza bitova od $i \cdot k$ do $(i+1) \cdot k - 1$ registra 0. Obratite pažnju da nakon pokretanja programa, ostali bitovi registra 0 (sa indeksima najmanje $n \cdot k$) i svi bitovi u ostalim registrima mogu biti proizvoljni.

- Prvi zadatak $(s = 0)$ je određivanje najmanjeg cijelog broja u ulaznom nizu cijelih brojeva $a[0], a[1], \ldots, a[n-1]$. Preciznije, $c[0]$ treba da je minimum brojeva $a[0], a[1], \ldots, a[n-1]$. Vrijednosti $c[1], c[2], \ldots, c[n-1]$ mogu biti proizvoljne.
- Drugi zadatak $(s = 1)$ je sortiranje ulaznih cijelih brojeva $a[0], a[1], \ldots, a[n-1]$ u neopodajući poredak. Preciznije, za sve i ($0 \leq i \leq n-1$), $c[i]$ treba da je jednak $1 + i$-tom najmanjem broju među brojevima $a[0], a[1], \ldots, a[n-1]$ (tj. $c[0]$ je najmanji broj među ulaznim brojevima).

Pomozite Kristoferu da napiše programe za ova dva zadatka, pri čemu programi mogu sadržati najviše q instrukcija.

Detalji implementacije

Potrebno je implementirati sljedeću funkciju:

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

- s: tip zadatka.
- n: broj ulaznih cijelih brojeva.
- k: broj bitova u svakom ulaznom broju.
- q: najveći mogući broj instrukcija.
- Ova funkcija se poziva tačno jednom i treba da kreira niz instrukcija koje izvršavaju zadatak.

Ova funkcija poziva jednu ili više od sljedećih funkcija da bi konstruisala traženi niz instrukcija:

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)

- Svaka od ovih funkcija nadovezuje instrukciju $move(t, y)$ $store(t, v)$, $and(t, x, y)$, $or(t, x, y)$, $xor(t, x, y)$, $not(t, x)$, $left(t, x, p)$, $right(t, x, p)$ i $add(t, x, y)$ na program.
- Za sve relevantne instrukcije, t, x, y moraju biti između 0 i $m - 1$ uključivo.
- Za sve relevantne instrukcije, t, x, y ne moraju biti po parovima različiti.
- Za instrukcije $left$ i $right$, p mora biti između 0 i b, uključivo.
- Za instrukciju $store$, dužina v mora biti b.

Možete pozivati i sljedeću funkciju, u cilju testiranja vašeg rješenja:

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

- Svaki poziv ove funkcije biće ignorisan prilikom ocjenivanja vašeg rješenja.
- U programu za ocjenjivanje (grader-u), ova funkcija nadovezuje operaciju $print(t)$ na program.
- Kada program za ocjenivanje naiđe na operaciju $print(t)$ tokom izvršavanja programa, štampa n k-bitnih cijelih brojeva formiranih od prvih $n \cdot k$ bitova registra t (za više detalja pogledajte sekciju "Program za ocjenivanje (Sample Grader)").
- t mora da zadovoljava uslov $0 \leq t \leq m - 1$.
- Poziv ove funkcije ne uvećava broj konstruisanih instrukcija.

Nakon nadovezivanja posljednje instrukcije, funkcija `construct_instructions` završava rad. Program se zatim ocjenjuje na više test-primjera, pri čemu svaki primjer zadaje ulaz koji se sastoji od n k-bitnih cijelih brojeva integers $a[0], a[1], \ldots, a[n-1]$. Vaše rješenje prolazi dati test-primjer ako, za dati ulaz, izlaz vašeg programa $c[0], c[1], \ldots, c[n-1]$ zadovolja sljedeće uslove:

- Ako $s = 0$, $c[0]$ treba da je najmanji od brojeva $a[0], a[1], \ldots, a[n-1]$.
- Ako je $s = 1$, za sve i ($0 \leq i \leq n - 1$), $c[i]$ treba da je $1 + i$-ti najmanji broj od brojeva $a[0], a[1], \ldots, a[n-1]$.

Pri ocjenjivanju vašeg rješenja moguće se sljedeće greške:

- `Invalid index`: neispravan indeks (moguće negativan) je predat kao parametar t, x ili y prilikom poziva neke od funkcija.
- `Value to store is not b bits long`: vrijednost v predata funkciji `append_store` nije jednaka b.
- `Invalid shift value`: vrijednost p predata funkcijama `append_left` ili `append_left` nije između 0 i b inkluzivno.

- `Too many instructions`: vaše rješenje je pokušalo da izvrši više od q instrukcija.

Primjeri

Primjer 1

Pretpostavimo da je $s = 0,\ n = 2,\ k = 1,\ q = 1000$. Postoje dva ulazna cijela broja $a[0]$ i $a[1]$, svaki sa po $k = 1$ bitom. Prije pokretanja programa, $r[0][0] = a[0]$ i $r[0][1] = a[1]$. Svi ostali bitovi u svim registrima su postavljeni na 0. Naon izvršenja svih instrukcija, mora važiti $c[0] = r[0][0] = \min(a[0], a[1])$, što je manji od brojeva $a[0]$ i $a[1]$.

Postoje samo 4 moguća ulaza za program:

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

Uočite da za sva 4 slučaja, $\min(a[0], a[1])$ se može dobiti kao bitwise-AND nad $a[0]$ i $a[1]$. Dakle, jedno moguće rješenje za konstrukciju programa je pomoću sljedećih poziva:

1. `append_move(1, 0)`, koji nadovezuje instrukciju koja kopira $r[0]$ u $r[1]$.
2. `append_right(1, 1, 1)`, koji nadovezuje instrukciju koja uzima sve bitove registra $r[1]$, pomjera ih udesno za 1 bit I sačuva vrijednost nazad u $r[1]$. Kako su svi ulazi dugački 1 bit, kao rezultat dobijamo da je $r[1][0]$ sada jednak $a[1]$.
3. `append_and(0, 0, 1)`, koji nadovezuje instrukciju koja primjenuje bitwise-AND nad $r[0]$ i $r[1]$ i zatim smjesti rezultat u $r[0]$. Nakon izvršavanje ove instrukcije, $r[0][0]$ će imati vrijednost bitwise-AND nad $r[0][0]$ i $r[1][0]$, što je u stvari bitwise-AND nad $a[0]$ i $a[1]$, što smo i željeli.

Primjer 2

Pretpostavimo da je $s = 1,\ n = 2,\ k = 1,\ q = 1000$. Kao i u prethodnom primjeru, postoje sano 4 moguća ulaza za naš program. U sva 4 slučaja, $\min(a[0], a[1])$ je bitwise-AND nad $a[0]$ i $a[1]$, a $\max(a[0], a[1])$ je bitwise-OR nad $a[0]$ i $a[1]$. Jedno moguće rješenje je pozivanje sljedećih funkcija:

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)`

Nakon izvršavanja ovih instrukcija, $c[0] = r[0][0]$ sadrži $\min(a[0], a[1])$ a $c[1] = r[0][1]$ sadrži $\max(a[0], a[1])$, čime je ulaz sortiran.

Ograničenja

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

Podzadaci

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

Program za ocjenjivanje (sample grader)

Program za ocjenjivanje (sample grader) čita ulaz u sledećem formatu:

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

Nakon toga slijedi nekoliko redova, koji opisuju po jedan test slučaj. Svaki test primjer je u sljedećem formatu:

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

i opisuje test slučaj čiji se ulaz sastoji od n cijelih brojeva $a[0], a[1], ..., a[n-1]$. Opis svih test slučajeva prati jedan red koji sadrži samo -1.

Program za ocjenjivanje (sample grader) prvo poziva `construct_instructions(s, n, k, q)`. Ako ovaj poziv krši neko ograničenje opisano u tekstu zadatka, grader ispisuje jednu od poruka o greškama navedenu na kraju odjeljka „Detalji implementacije“ i izlazi. U suprotnom, program za ocjenjivanje (sample grader) prvo štampa svaku naredbu dodatu u `construct_instructions(s, n, k, q)` redom. Za instrukcije $store,\ v$ se štampa od indeksa 0 do indeksa $b - 1$.

Zatim, program za ocjenjivanje (sample grader) obrađuje testove po redu. Za svaki test slučaj pokreće konstruisani program na ulazu test slučaja.

Za svaku operaciju $print(t)$, neka je $d[0], d[1], \ldots, d[n-1]$ niz cijelih brojeva, takav da sve i ($0 \leq i \leq n - 1$), $d[i]$ je cjelobrojna vrijednost niza bitova $i \cdot k$ do $(i + 1) \cdot k - 1$ registra t (kada je operacija izvršena). Program za ocjenjivanje(sample grader) štampa ovaj niz u sledećem formatu:: `register` t: $d[0]\ d[1]\ \ldots\ d[n-1]$.

Jednom kada su izvršene sve instrukcije, program za ocjenjivanje (sample grader) štampa izlaz programa.

Ako je $s = 0$, izlaz grejdera za svaki test primer je u sljedećem formatu:

- $c[0]$. Ako je $s = 1$, izlaz grejdera za svaki test primer je u sljedećem formatu::
- $c[0]$ $c[1]$ $\ldots$ $c[n-1]$.

Nakon izvršavanja svih test slučajeva, grejder štampa `number of instructions:` X gdje je X broj instrukcija u vašem programu.