Bitu nobīdes reģistri

Inženieris Kristofers strādā pie jauna veida datoru procesora.

Procesoram ir piekļuve m dažādām b bitu šūnām (kur $m = 100$ un $b = 2000$), kuras sauc par **reģistriem** un kas ir numurētas no 0 līdz $m - 1$. Apzīmēsim reģistrus kā $r[0], r[1], \ldots, r[m-1]$. Katrs reģistrs ir b bitu masīvs, kas ir numurēti no 0 (pats labējais bits) līdz $b - 1$ (pats kreisākais bits). Katram i $(0 \leq i \leq m - 1)$ un katram j $(0 \leq j \leq b - 1)$ apzīmēsim i-tā reģistra j-to bitu kā $r[i][j]$.

Jebkurai bitu virknei $d_0, d_1, \ldots, d_{l-1}$ (patvaļīgā garumā l) šīs virknes **veselo skaitļu vērtība** ir vienāda ar $2^0 \cdot d_0 + 2^1 \cdot d_1 + \ldots + 2^{l-1} \cdot d_{l-1}$. Teiksim, ka **reģistrā i saglabātā veselo skaitļu vērtība** ir tā bitu veselo skaitļu vērtība, t.i., tā ir $2^0 \cdot r[i][0] + 2^1 \cdot r[i][1] + \ldots + 2^{b-1} \cdot r[i][b-1]$.

Procesoram ir 9 veidu **instrukcijas**, kuras var izmantot, lai reģistros mainītu bitus. Katra instrukcija operē ar vienu vai vairākiem reģistriem un saglabā izvadu vienā no reģistriem. Turpmāk ar $x := y$ apzīmēsim operāciju, kas nomaina x vērtību, lai tā kļūst vienāda ar y. Katra instrukcijas veida veiktās operācijas ir aprakstītas tālāk.

- $move(t, y)$: Kopē bitu masīvu no reģistra y uz reģistru t. Katram j $(0 \leq j \leq b - 1)$ uzstāda $r[t][j] := r[y][j]$.
- $store(t, v)$: Uzstāda reģistru t vienādu ar v, kur v ir b bitu masīvs. Katram j $(0 \leq j \leq b - 1)$ uzstāda $r[t][j] := v[j]$.
- $and(t, x, y)$: Aprēķina reģistru x un y bināro UN operāciju un saglabā rezultātu reģistrā t. Katram j $(0 \leq j \leq b - 1)$ uzstāda $r[t][j] := 1$, ja **abi** $r[x][j]$ un $r[y][j]$ ir 1, un uzstāda $r[t][j] := 0$ citādi.
- $or(t, x, y)$: Aprēķina reģistru x un y bināro VAI operāciju un saglabā rezultātu reģistrā t. Katram j $(0 \leq j \leq b - 1)$ uzstāda $r[t][j] := 1$, ja **vismaz viens** no $r[x][j]$ un $r[y][j]$ ir 1, un uzstāda $r[t][j] := 0$ citādi.
- $xor(t, x, y)$: Aprēķina reģistru x un y bināro IZSLĒDZOŠAIS VAI operāciju un saglabā rezultātu reģistrā t. Katram j $(0 \leq j \leq b - 1)$ uzstāda $r[t][j] := 1$, ja **tieši viens** no $r[x][j]$ un $r[y][j]$ ir 1, un uzstāda $r[t][j] := 0$ citādi.
- $not(t, x)$: Aprēķina reģistra x bināro NE un saglabā rezultātu reģistrā t. Katram j $(0 \leq j \leq b - 1)$ uzstāda $r[t][j] := 1 - r[x][j]$.
- $left(t, x, p)$: Nobīda visus bitus reģistrā x pa kreisi par p pozīcijām, un saglabā rezultātu reģistrā t. Reģistra x visu bitu nobīdes pa kreisi par p pozīcijām rezultāts ir masīvs v no b bitiem. Katram j $(0 \leq j \leq b - 1)$ izpildās $v[j] = r[x][j - p]$, ja $j \geq p$, un $v[j] = 0$ citādi. Katram j $(0 \leq j \leq b - 1)$ operācija uzstāda $r[t][j] := v[j]$.

- $right(t, x, p)$: Nobīda visus bitus reģistrā x pa labi par p pozīcijām un saglabā rezultātu reģistrā t. Reģistra x visu bitu nobīdes pa labi par p pozīcijām rezultāts ir masīvs v no b bitiem. Katram j $(0 \leq j \leq b-1)$ izpildās $v[j] = r[x][j+p]$, ja $j \leq b-1-p$, un $v[j] = 0$ citādi. Katram j $(0 \leq j \leq b-1)$ operācija uzstāda $r[t][j] := v[j]$.
- $add(t, x, y)$: Saskaita reģistrā x un reģistrā x saglabātās veselās vērtības un saglabā rezultātu reģistrā t. Saskaitīšanu veic pēc moduļa 2^b. Formāli, lai X ir reģistrā x saglabātā veselo skaitļu vērtība un Y ir reģistrā y saglabātā vērtība pirms operācijas. Apzīmēsim reģistrā t pēc operācijas saglabāto veselo skaitļu vērtību kā T. Ja $X + Y < 2^b$, tad operācija uzstāda reģistra t bitus tā, lai $T = X + Y$. Citādi operācija uzstāda reģistra t bitus tā, lai $T = X + Y - 2^b$.

Kristofers gribētu, lai jūs atrisinātu divu veidu uzdevumus, izmantojot jauno procesoru. Uzdevuma veidu apzīmē ar veselu skaitli s. Abiem uzdevumu veidiem jums ir jāsagatavo **programma**, tas ir, augstāk definēto instrukciju virkne.

Programmas **ievads** sastāv no n veseliem skaitļiem $a[0], a[1], \ldots, a[n-1]$, katram saturot k bitus, t.i., $a[i] < 2^k$ ($0 \leq i \leq n-1$). Pirms programmas izpildes visi ievada skaitļi tiek secīgi saglabāti reģistrā 0 tā, lai katram i $(0 \leq i \leq n-1)$ k bitu virknes $r[0][i \cdot k], r[0][i \cdot k + 1], \ldots, r[0][(i+1) \cdot k - 1]$ veselo skaitļu vērtība ir vienāda ar $a[i]$. Ievērojiet, ka $n \cdot k \leq b$. Visi citi biti reģistrā 0 (t.i., biti ar indeksiem no $n \cdot k$ līdz $b-1$ ieskaitot) un visi biti visos citos reģistros tiek uzstādīti uz 0.

Programmas izpildi veido tās instrukciju secīga izpilde. Pēc pēdējās instrukcijas izpildes tiek aprēķināts programmas **izvads**, balstoties uz reģistra 0 bitu gala vērtībām. Precīzāk, izvads ir n veselu skaitļu virkne $c[0], c[1], \ldots, c[n-1]$, kur katram i ($0 \leq i \leq n-1$) $c[i]$ ir veselo skaitļu vērtība, kas sastāv no reģistra 0 bitiem $i \cdot k$ līdz $(i+1) \cdot k - 1$. Ievērojiet, ka pēc programmas izpildes reģistra 0 atlikušajiem bitiem (ar indeksiem no $n \cdot k$) un visu citu reģistru visiem bitiem var būt patvaļīgas vērtības.

- Pirmais uzdevums $(s = 0)$ ir atrast mazāko veselo skaitli no ievadītajiem veselajiem skaitļiem $a[0], a[1], \ldots, a[n-1]$. Precīzāk, $c[0]$ jābūt vismazākajam skaitlim no $a[0], a[1], \ldots, a[n-1]$. $c[1], c[2], \ldots, c[n-1]$ vērtības var būt patvaļīgas.
- Otrais uzdevums $(s = 1)$ ir sakārtot ievada skaitļus $a[0], a[1], \ldots, a[n-1]$ nedilstošā secībā. Precīzāk, katram i ($0 \leq i \leq n-1$) $c[i]$ ir jābūt vienādam ar $(1+i)$-to mazāko veselo skaitli no $a[0], a[1], \ldots, a[n-1]$ (t.i., $c[0]$ būs vismazākais veselais skaitlis no ievadītajiem veselajiem skaitļiem).

Sagatavojiet Kristoferam programmas, kas var atrisināt šos uzdevumus un kur katrā no tām ir ne vairāk kā q instrukcijas.

Realizācijas detaļas

Jums ir jārealizē tāda procedūra:

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

- s: uzdevuma veids.
- n: veselo skaitļu skaits ievadā.
- k: bitu skaits katrā ievada veselajā skaitlī.
- q: lielākais atļautais instrukciju skaits.
- Šī procedūra tiek izsaukta tieši vienreiz, un tai ir jāsagatavo instrukciju virkne, kas izpilda vajadzīgo uzdevumu.

Šai procedūrai būtu jāizsauc viena vai vairākas tādas procedūras, lai izveidotu instrukciju virkni:

```
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)
```

- Katra procedūra pievieno $move(t, y)$ $store(t, v)$, $and(t, x, y)$, $or(t, x, y)$, $xor(t, x, y)$, $not(t, x)$, $left(t, x, p)$, $right(t, x, p)$ vai $add(t, x, y)$ instrukciju programmai, attiecīgi.
- Visām instrukcijām, uz kurām tas attiecas, t, x, y ir jābūt vismaz 0 un ne lielākiem par $m - 1$.
- Visām instrukcijām, uz kurām tas attiecas, t, x, y ne obligāti ir pa pāriem atšķirīgi.
- $left$ un $right$ instrukcijām p ir jābūt vismaz 0 un ne lielākam par b.
- $store$ instrukcijām v garumam jābūt b.

Jūs varat arī izsaukt tādu procedūru atkļūdošanas nolūkos:

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

- Jebkurš šīs procedūras izsaukums netiks ņemts vērtā jūsu risinājuma vērtēšanas laikā.
- Paraugvērtētājā šī procedūra pievieno $print(t)$ operāciju programmai.
- Kad paraugvērtētājs sastop $print(t)$ operāciju programmas izpildes laikā, tas izdrukā n k bitu veselos skaitļus, kurus veido pirmie $n \cdot k$ reģistra t biti (skat. detaļas sekcijā "Paraugvērtētājs").
- t ir jāapmierina $0 \leq t \leq m - 1$.
- Jebkurš šīs procedūras izsaukums nepalielina izveidoto instrukciju skaitu.

Pēc pēdējās instrukcijas pievienošanas procedūrai `construct_instructions` ir jābeidzas. Tad programmu izpilda uz kāda testpiemēru skaita, kas katrs satur ievadu no n k bitu veselajiem skaitļiem $a[0], a[1], \ldots, a[n-1]$. Jūsu risinājums veiksmīgi iziet doto testpiemēru, ja programmas izvads $c[0], c[1], \ldots, c[n-1]$ dotajam ievadam apmierina tādus nosacījumus:

- Ja $s = 0$, $c[0]$ ir jābūt vismazākajai vērtībai no $a[0], a[1], \ldots, a[n-1]$.
- Ja $s = 1$, katram i ($0 \leq i \leq n-1$) $c[i]$ ir jābūt $(1+i)$-jām mazākajam veselajam skaitlim no $a[0], a[1], \ldots, a[n-1]$.

Jūsu risinājuma vērtēšana var beigties ar vieno no tādiem kļūdas paziņojumiem:

- `Invalid index`: nepareiz (iespējams, negatīvs) reģistra indekss tika padots kā t, x vai y parametrs kādam procedūras izsaukumam.
- `Value to store is not b bits long`: procedūrai `append_store` padotajam v garums nav vienāds ar b.
- `Invalid shift value`: procedūrām `append_left` vai `append_right` padotā p vertība nav starp 0 un b ieskaitot.
- `Too many instructions`: jūsu procedūra mēģināja pievienot vairāk nekā q instrukcijas.

Piemēri

1. piemērs

Aplūkosim gadījumu, kad $s = 0,\ n = 2,\ k = 1,\ q = 1000$. Ir divi ievada veselie skaitļi $a[0]$ un $a[1]$, katrs pa $k = 1$ bitam. Pirms programmas izpildes $r[0][0] = a[0]$ un $r[0][1] = a[1]$. Visi pārējie procesora biti ir uzstādīti uz 0. Pēc visu programmas instrukciju izpildes mums ir jāiegūst $c[0] = r[0][0] = \min(a[0], a[1])$, kas ir $a[0]$ un $a[1]$ mazākā vērtība.

Ir tikai 4 iespējamie programmas ievadi:

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

Mēs varam ievērot, ka visiem 4 gadījumiem $\min(a[0], a[1])$ ir vienāds ar bināro UN no $a[0]$ un $a[1]$. Tādējādi, iespējamais risinājums ir izveidot programmu, izmantojot tādu izsaukumus:

1. `append_move(1, 0)`, kas pievieno instrukciju $r[0]$ kopēšanai uz $r[1]$.
2. `append_right(1, 1, 1)`, kas pievieno instrukciju paņemt visus $r[1]$ bitus, nobīdīt tos pa labi uz 1 bitu, un saglabāt rezultātu atpakaļ $r[1]$. Tā kā katrs veselais skaitlis ir 1 bitu garš, rezultātā $r[1][0]$ kļūst vienāds ar $a[1]$.
3. `append_and(0, 0, 1)`, kas pievieno instrukciju aprēķināt $r[0]$ un $r[1]$ bināro UN un saglabāt rezultātu $r[0]$. Pēc šīs instrukcijas izpildes $r[0][0]$ ir uzstādīts uz bināro UN no $r[0][0]$ un $r[1][0]$, kas ir vienāds ar bināro UN no $a[0]$ un $a[1]$, kas arī bija vajadzīgs.

2. piemērs

Aplūkosim gadījumu, kad $s = 1,\ n = 2,\ k = 1,\ q = 1000$. Kā iepriekšējā piemērā, ir tikai 4 iespējamie programmas ievadi. Visiem četriem gadījumiem $\min(a[0], a[1])$ ir binārais UN no $a[0]$ un $a[1]$, un $\max(a[0], a[1])$ ir binārais VAI no $a[0]$ un $a[1]$. Iespējamais risinājums ir veikt tādus izsaukumus:

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)`

Pēc šo instrukciju izpildes $c[0] = r[0][0]$ satur $\min(a[0], a[1])$ un $c[1] = r[0][1]$ satur $\max(a[0], a[1])$, kas atbilst sakārtotam ievadam.

Ierobežojumi

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

Apakšuzdevumi

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

Paraugvērtētājs

Paraugvertētājs lasa ievadu tādā formātā:

- 1. rinda: $s\ n\ k\ q$

Pēc tam seko viena vai vairākas rindas, kas katra apraksta vienu testpiemēru. Katrs testpiemērs ir tādā formātā:

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

un apraksta ievadu, kas sastāv no n veselajiem skaitļiem $a[0], a[1], \ldots, a[n-1]$. Pēc visu testpiemēru apraksta seko viena rinda, kas satur tikai -1.

Paraugvērtētājs vispirms izsauc `construct_instructions(s, n, k, q)`. Ja šis izsaukums pārkāpj nosacījumos norādītos ierobežojumus, paraugvērtētājs izdrukā vienu no kļūdas paziņojumiem, kas ir aprakstīti sekcijas "Realizācijas detaļas" beigās, un iziet. Citādi paraugvērtētājs sākumā izdrukā katru instrukciju, kuru pievienoja procedūra `construct_instructions(s, n, k, q)`, pievienošanas secībā. $store$ instrukcijām tiek izdrukāts v no indeksa 0 līdz indeksam $b - 1$.

Pēc tam paraugvērtētājs secīgi apstrādā testpiemērus. Katram testpiemēram tas izpilda sagatavoto programmu uz testpiemēra ievada.

Katrai $print(t)$ operācijai definēsim $d[0], d[1], \ldots, d[n-1]$ kā veselu skaitļu virkni, tādu, ka katram i ($0 \leq i \leq n-1$) $d[i]$ ir reģistra t (ar ko tiek veikta operācija) bitu virknes $i \cdot k$ to $(i+1) \cdot k - 1$ veselā vērtība. Paraugvērtētājs izdrukā šo virkni tādā formātā: `register` t: $d[0]$ $d[1]$ $\ldots$ $d[n-1]$.

Pēc visu instrukciju izpildes paraugvērtētājs izdrukā programmas izvadu.

Ja $s = 0$, paraugvērtētāja izvads katram testpiemēram ir tādā formātā:

- $c[0]$.

Ja $s = 1$, paraugvērtētāja izvads katram testpiemēram ir tādā formātā:

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

Pēc visu testpiemēru izpildes paraugvērtētājs izdrukā `number of instructions:` X, kur X ir instrukciju skaits jūsu programmā.