Bit Shift Registers

Кристофър работи върху нов тип процесор.

Процесорът има достъп до m различни b-битови клетки от паметта (където $m = 100$ и $b = 2000$), които наричаме **регистри**, и те са номерирани от 0 to $m - 1$. Означаваме регистрите с $r[0], r[1], \ldots, r[m-1]$. Всеки регистър е масив от b бита, номерирани от 0 (за най-десния бит) до $b - 1$ (за най-левия бит). За всяко i $(0 \leq i \leq m - 1)$ и всяко j $(0 \leq j \leq b - 1)$, означаваме j-тия бит на регистъра i с $r[i][j]$.

За всяка редица от битове $d_0, d_1, \ldots, d_{l-1}$ (с произволна дължина l), **целочислената стойност** на редицата е равна на $2^0 \cdot d_0 + 2^1 \cdot d_1 + \ldots + 2^{l-1} \cdot d_{l-1}$. **Целочислена стойност, която се съхранява в регистър** i дефинираме като цялото число, чиято стойност е $2^0 \cdot r[i][0] + 2^1 \cdot r[i][1] + \ldots + 2^{b-1} \cdot r[i][b-1]$.

Процесорът има 9 типа **инструкции**, с които може да се модифицират битовете на регистрите. Всяка инструкция действа върху един или повече регистри и записва резултата в един от регистрите. Използваме записа $x := y$, за да означим действието за промяна на стойността x, така че тя да стане равна на y. Действията на инструкции са описани по-долу:

- $move(t, y)$: Копира масива от битове на регистъра y в регистъра t. За всяко j $(0 \leq j \leq b - 1)$, тази инструкция записва $r[t][j] := r[y][j]$.
- $store(t, v)$: Записва в регистър t стойността v, където v е масив от b бита. За всяко j $(0 \leq j \leq b - 1)$, тази инструкция записва $r[t][j] := v[j]$.
- $and(t, x, y)$: Взема побитово И на регистрите x и y и го записва в регистър t. За всяко j $(0 \leq j \leq b - 1)$, записва $r[t][j] := 1$ когато **и двете** $r[x][j]$ и $r[y][j]$ са 1, и записва $r[t][j] := 0$ в противен случай.
- $or(t, x, y)$: Взема побитово ИЛИ на регистрите x и y и го записва в регистър t. За всяко j $(0 \leq j \leq b - 1)$, записва $r[t][j] := 1$ когато **поне един от** $r[x][j]$ и $r[y][j]$ е 1, и записва $r[t][j] := 0$ в противен случай.
- $xor(t, x, y)$: Взема побитово ИЗКЛЮЧВАЩО ИЛИ на регистрите x и y и го записва в регистър t. За всяко j $(0 \leq j \leq b - 1)$, записва $r[t][j] := 1$ когато **точно един от** $r[x][j]$ и $r[y][j]$ е 1, и записва $r[t][j] := 0$ в противен случай.
- $not(t, x)$: Взема побитово НЕ на регистър x и го записва в регистър t. За всяко j $(0 \leq j \leq b - 1)$, записва $r[t][j] := 1 - r[x][j]$.
- $left(t, x, p)$: Измества всичките битове в регистъра x наляво на p позиции и записва резултата в регистър t. Резултатът от изместването на битовете от регистъра x наляво на p позиции е масив v състоящ се от b бита. За всяко j $(0 \leq j \leq b - 1)$,

$v[j] = r[x][j-p]$ ако $j \geq p$, и $v[j] = 0$ в противен случай. За всяко j $(0 \leq j \leq b-1)$, записва $r[t][j] := v[j]$.

- $right(t, x, p)$: Измества всичките битове в регистъра x надясно на p позиции и записва резултата в регистър t. Резултатът от изместването на битовете от регистъра x надясно на p позиции е масив v състоящ се от b бита. За всяко j $(0 \leq j \leq b-1)$, $v[j] = r[x][j-p]$ ако $j \leq b-1-p$, и $v[j] = 0$ в противен случай. За всяко j $(0 \leq j \leq b-1)$, записва $r[t][j] := v[j]$.

- $add(t, x, y)$: Събира целочислените стойности, намиращи се в регистър x и в регистър y и записва резултата в регистър t. Събирането се извършва по модул 2^b. Т.е. ако X е целочислената стойност, записана в регистър x и Y е целочислената стойност, записана в регистър y, то за целочислената стойност T, която ще е записана след изпълнението на инструкцията в регистър t, ще е изпълнено: Ако $X + Y < 2^b$, то се слагат битовете на t, така че $T = X + Y$. В противен случай, се слагат битовете на t, така че $T = X + Y - 2^b$.

Кристофър иска да реше два типа задачи чрез новия процесор. Типът на задачата се означава с цялото число s. За двата типа задачи, вие трябва да напишете **програма**, която е последователност от инструкции, от вида на описаните по-горе.

Входът на програмата (**input**) се състои от n цели числа $a[0], a[1], \ldots, a[n-1]$, всяко имащо k бита, т.е. $a[i] < 2^k$ ($0 \leq i \leq n-1$). Преди изпълнението на програмата всички числа от входа са записани последователно в регистър 0, така че за всяко i $(0 \leq i \leq n-1)$ стойността на последователността от k бита $r[0][i \cdot k], r[0][i \cdot k + 1], \ldots, r[0][(i+1) \cdot k - 1]$ е равна на $a[i]$. Да отбележим, че $n \cdot k \leq b$. Всичките други битове в регистъра 0 (т.е. тези, чиито индекси са между $n \cdot k$ и $b-1$, включително) и всички битове в другите регистри са инициализирани с 0.

Изпълнението на програмата се състои в изпълнението на инструкциите им в техния ред. След изпълнението на последната инструкция, изходът на програмата (**output**) представлява това, което се намира в регистър 0. Т.е. изходът е последователност от n цели числа $c[0], c[1], \ldots, c[n-1]$, където за всяко i ($0 \leq i \leq n-1$), $c[i]$ е целочислената стойност на редицата от битове $i \cdot k$ до $(i+1) \cdot k - 1$ в регистър 0. Да отбележим, че след изпълнението на програмата, останалите битове на регистър 0 (тези с индекси, равни най-малко на $n \cdot k$), както и всички битове на останалите регистри може да бъдат произволни.

- Първата задача $(s = 0)$ е да се намери най-малкото цяло число измежду входните числа $a[0], a[1], \ldots, a[n-1]$. Т.е. $c[0]$ трябва да е минимумът на $a[0], a[1], \ldots, a[n-1]$. Стойности на $c[1], c[2], \ldots, c[n-1]$ може да са произволни.

- Втората задача $(s = 1)$ е да се сортират числата от входа $a[0], a[1], \ldots, a[n-1]$ в ненамаляващ ред, т.е. за всяко i ($0 \leq i \leq n-1$), $c[i]$ трябва да е равно на $(1+i)$-тото най-малко число измежду $a[0], a[1], \ldots, a[n-1]$ (и $c[0]$ е най-малкото число измежду входните числа).

Напишете програма за Кристофър, състояща се от най-много q инструкции за всяка от задачите.

Детайли по реализацията

Трябва да имплементирате следната функция:

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

- s: тип задача.
- n: брой на числата във входа
- k: брой на битовете за всяко входно число
- q: максимален брой на инструкциите
- Функцията се извиква точно веднъж и трябва да конструира последователността от инструкции, за да бъде решена съответната задача.

Функцията може да извиква една или повече от следните функции, за да конструира редицата от инструкции.

```
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)
```

- Всяка от тези функции добавя по една инструкция, съответно $move(t, y)$ $store(t, v)$, $and(t, x, y)$, $or(t, x, y)$, $xor(t, x, y)$, $not(t, x)$, $left(t, x, p)$, $right(t, x, p)$ или $add(t, x, y)$ към програмата, която се контуира.
- За всяка валидна инструкция, t, x, y трябва да са най-малко равни на 0 и най-много на $m - 1$.
- За всички валидни инструкция, не е задължително t, x, y да са различни две по две числа.
- За инструкциитв $left$ и $right$, p трябва да е най-малко равно на 0 и най-много на b.
- За инструкцията $store$, дължината на v трябва да е b.

Вие може също да извиквате следната функция, за да си помогнете при тестването на вашето решение:

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

- Всяко извикване на тази функция ще бъде игнорирано при оценяването на вашето решение.
- В примерния грейдер тази функция добавя инструкция $print(t)$.

- Когато примерният грейдер срещне $print(t)$, той отпечатва n на брой k-битови цели числа, образувани от първите $n \cdot k$ бита на регистър t (Виж "Примерен грейдер" за подробности).
- t трябва да удовлетворява $0 \leq t \leq m - 1$.
- Всяко извикване на тази функция няма да се отчита за броя на конструираните инструкции.

След добавяне на последната инструкция, функцията `construct_instructions` трябва да завърши с return. Програмата се оценява с няколко тестови примери, всеки задаващ вход, състоящ се от n на брой k-битови цели числа $a[0], a[1], \ldots, a[n-1]$. Вашето решение минава успешно даден тест, когато изходът на програмата $c[0], c[1], \ldots, c[n-1]$ за входа в теста удовлетворява следните условия:

- Ако $s = 0$, то $c[0]$ трябва да най-малката стойност измежду $a[0], a[1], \ldots, a[n-1]$.
- Ако $s = 1$, то за всяко i ($0 \leq i \leq n-1$), $c[i]$ трябва да е $(1+i)$-тото най-малко число измежду $a[0], a[1], \ldots, a[n-1]$.

Резултатът от оценяването на вашето решение може да бъде едно от следните съобщения:

- `Invalid index`: некоректен (възможно е да има отрицателен индекс на регистър, подаден като параметър t, x or y за някое извикване на една от функциите.
- `Value to store is not b bits long`: дължината на v подадена на `_store` не е равна на b.
- `Invalid shift value`: стойността на p подадена на `_left` или `_right` не е между 0 и b включително.
- `Too many instructions`: вашата функция се опитва да добави повече от q инструкции.

Примери

Пример 1

Нека $s = 0,\ n = 2,\ k = 1,\ q = 1000$. Има две числа във входа $a[0]$ и $a[1]$, всяко с $k = 1$ бита. Преди изпълнението на програмата, $r[0][0] = a[0]$ и $r[0][1] = a[1]$. Всички други битове са 0. След изпълнението на всички инструкции, трябва да имаме $c[0] = r[0][0] = \min(a[0], a[1])$, което е минимум на $a[0]$ и $a[1]$.

Има само 4 възможни входа на програмата:

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

Отбелязваме, че за всичките 4 случая $\min(a[0], a[1])$ е равно на побитово И на $a[0]$ и $a[1]$. Така, възможно решение е да се конструира програма чрез следните извиквания:

1. `append_move(1, 0)`, което добавя инструкция за копиране на $r[0]$ към $r[1]$.

2. `append_right(1, 1, 1)`, което добавя инструкция, която взема всички битове на $r[1]$, измества ги надясно с 1, и запазва резултата в $r[1]$. Понеже всяко число е дълго 1 бит, резултатът в $r[1][0]$ е равен на $a[1]$.
3. `append_and(0, 0, 1)`, което добавя инструкция да направи побитово И на $r[0]$ и $r[1]$, и да запази резултата в $r[0]$. Тогава $r[0][0]$ става равно на побитово И на $r[0][0]$ и $r[1][0]$, което е равно на побитово И на $a[0]$ и $a[1]$.

Пример 2

Нека $s = 1,\ n = 2,\ k = 1,\ q = 1000$. Както в предния пример, има само 4 възможни входа. За всичките $\min(a[0], a[1])$ е побитово И на $a[0]$ и $a[1]$, и $\max(a[0], a[1])$ е побитово ИЛИ на $a[0]$ и $a[1]$. Възможно решение е :

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)`

След изпълнение на инструкциите, $c[0] = r[0][0]$ съдържа $\min(a[0], a[1])$, и $c[1] = r[0][1]$ съдържа $\max(a[0], a[1])$, с което входът е сортиран

Ограничения

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

Подзадачи

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

Примерен грейдер

Примерният грейдер чете входа в следния формат

- ред 1 : s n k q

Което е следвано от няколко реда, всеки описващ по един тест Всеки тест е даден в следния формат:

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

и описва тест, чийто вход има n числа $a[0], a[1], ..., a[n-1]$.

Описанието на всички тестове завършва с един ред, съдържащ само -1.

Примерният грейдер първо извиква `construct_instructions(s, n, k, q)`. Ако това извикване нарушава някои ограничения, примерният грейдер отпечатва съобщение за грешка и завършва. В противен случай примерният грейдер оптечатва всяка инструкция от `construct_instructions(s, n, k, q)` по реда им. За инструкцията $store$, се отпечатват битовете на v от индекс 0 до индекс $b-1$.

След това, примерният грейдер обработва тестовете, За всеки тест примерният грейдер изпълняна конструираната програма.

За всяка операция $print(t)$, нека $d[0], d[1], \ldots, d[n-1]$ е редица от числа, такива че за всяко i ($0 \leq i \leq n-1$), $d[i]$ е стойността на редицата от битове $i \cdot k$ to $(i+1) \cdot k - 1$ на регистъра t след изпълнение на инструкцията. Грейдерът отпечатва редицата в следния формат: `register` t: $d[0]$ $d[1]$ $\ldots$ $d[n-1]$.

След като инструкуията е изпълнена, грейдерът отпечатва изходът на програмата.

Ако $s = 0$, изходът на грейдера за всеки тест е във формат:

- $c[0]$.

Ако $s = 1$, изходът на грейдера за всеки тест е във формат

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

След изпълнението за всички тестове, грейдерът отпечатва

`number of instructions:` X, където X е броя на инструкциите.