Регистры битового сдвига

Инженер Кристофер работает над новым типом процессора.

Процессор имеет доступ к m различным b-битным ячейкам памяти (где $m = 100$ и $b = 2000$), которые называются **регистрами** и пронумерованы числами от 0 до $m - 1$. Мы будем обозначать регистры как $r[0], r[1], \ldots, r[m-1]$. Каждый регистр представляет собой массив из b битов, пронумерованных от 0 (самый правый бит) до $b - 1$ (самый левый бит). Для каждого i $(0 \leq i \leq m - 1)$ и каждого j $(0 \leq j \leq b - 1)$ обозначим j-й бит регистра i как $r[i][j]$.

Для любой последовательности битов $d_0, d_1, \ldots, d_{l-1}$ (произвольной длины l) определим **целочисленное значение** этой последовательности как $2^0 \cdot d_0 + 2^1 \cdot d_1 + \ldots + 2^{l-1} \cdot d_{l-1}$. Определим также **целочисленное значение регистра** i как целочисленное значение последовательности бит этого регистра, то есть значение $2^0 \cdot r[i][0] + 2^1 \cdot r[i][1] + \ldots + 2^{b-1} \cdot r[i][b-1]$.

Процессор имеет 9 типов **инструкций**, которые могут быть использованы для изменения значений регистров. Каждая инструкция использует один или несколько регистров и записывает результат в один из регистров. В дальнейшем мы будем использовать обозначение $x := y$ для операции присваивания биту x значения y. Операции, выполняемые каждой из инструкций, представлены ниже:

- $move(t, y)$: скопировать биты регистра y в регистр t. Для каждого j $(0 \leq j \leq b - 1)$ выполняется $r[t][j] := r[y][j]$.
- $store(t, v)$: присвоить биты регистра t значению v, где v — некоторый массив из b бит. Для каждого j $(0 \leq j \leq b - 1)$ выполняется присваивание $r[t][j] := v[j]$.
- $and(t, x, y)$: вычислить побитовое И регистров x и y и записать результат в регистр t. Для каждого j $(0 \leq j \leq b - 1)$ выполняется присваивание $r[t][j] := 1$ если **оба** значения $r[x][j]$ и $r[y][j]$ равны 1, и $r[t][j] := 0$ в противном случае.
- $or(t, x, y)$: вычислить побитовое ИЛИ регистров x и y и записать результат в регистр t. Для каждого j $(0 \leq j \leq b - 1)$ выполняется присваивание $r[t][j] := 1$ если **хотя бы одно** из значений $r[x][j]$ и $r[y][j]$ равны 1, и $r[t][j] := 0$ в противном случае.
- $xor(t, x, y)$: вычислить побитовое исключающее ИЛИ регистров x и y и записать результат в регистр t. Для каждого j $(0 \leq j \leq b - 1)$ выполняется присваивание $r[t][j] := 1$ если **ровно одно** из значений $r[x][j]$ и $r[y][j]$ равно 1, и $r[t][j] := 0$ в противном случае.

- $not(t, x)$: вычислить побитовое отрицание регистра x и записать результат в регистр в t. Для каждого j $(0 \leq j \leq b-1)$ выполняется присваивание $r[t][j] := 1 - r[x][j]$.

- $left(t, x, p)$: сдвинуть все биты из регистра x влево на p и записать результат в регистр t. Результат сдвига битов из регистра x на p влево представляет собой массив v из b бит. Для каждого j $(0 \leq j \leq b-1)$ значение $v[j] = r[x][j-p]$, если $j \geq p$, и $v[j] = 0$ в противном случае. Для каждого j $(0 \leq j \leq b-1)$ выполняется присваивание $r[t][j] := v[j]$.

- $right(t, x, p)$: сдвинуть все биты из регистра x вправо на p и записать результат в регистр t. Результат сдвига битов из регистра x вправо на p представляет собой массив v из b битов. Для каждого j $(0 \leq j \leq b-1)$ значение $v[j] = r[x][j+p]$, если $j \leq b-1-p$, и $v[j] = 0$ в противном случае. Для каждого j $(0 \leq j \leq b-1)$ выполняется присваивание $r[t][j] := v[j]$.

- $add(t, x, y)$: вычислить сумму целочисленных значений регистров x и y и записать результат в регистр t. Сложение выполняется по модулю 2^b. Более формально, пусть X обозначает целочисленное значение регистра x, а Y — целочисленное значение регистра y перед выполнением операции. Пусть T обозначает целочисленное значение регистра t после выполнения операции. Если $X + Y < 2^b$, то биты регистра t выставляются таким образом, чтобы $T = X + Y$. В противном случае биты регистра t выставляются таким образом, что $T = X + Y - 2^b$.

Кристофер хотел бы научиться решать два типа задач с помощью нового процессора. Тип задачи обозначается числом s. Для обоих типов задач вам необходимо предоставить **программу**, которая представляет собой последовательность операций, описанных выше.

Входные данные к программе представляют собой n чисел $a[0], a[1], \ldots, a[n-1]$, каждое из которых состоит из k-бит, то есть, $a[i] < 2^k$ ($0 \leq i \leq n-1$). Перед выполнением программы все числа из входных данных содержаться последовательно в регистре 0, а именно, для каждого i $(0 \leq i \leq n-1)$ целочисленное значение последовательности из k бит $r[0][i \cdot k], r[0][i \cdot k + 1], \ldots, r[0][(i+1) \cdot k - 1]$ равно $a[i]$. Обратите внимание, что $n \cdot k \leq b$. Все остальные биты в регистре 0 (биты с индексами между $n \cdot k$ и $b-1$, включительно) и все биты в других регистрах изначально равны 0.

Выполнение программы состоит из выполнения инструкций этой программы по порядку. После выполнения последней инструкции **выходные данные** определяются итоговыми значениями битов регистра 0. Более конкретно, результатом работы программы является массив из n чисел $c[0], c[1], \ldots, c[n-1]$, где для каждого i ($0 \leq i \leq n-1$) значение $c[i]$ это целочисленное значение последовательности бит от $i \cdot k$ до $(i+1) \cdot k - 1$ регистра 0. Обратите внимание, что после выполнения программы все остальные биты регистра 0 (с индексами хотя бы $n \cdot k$) и все остальные биты других регистров могут иметь произвольные значения.

- Первая задача $(s = 0)$ состоит в нахождении минимального числа среди $a[0], a[1], \ldots, a[n-1]$. Более конкретно, $c[0]$ должно быть равно минимуму среди $a[0], a[1], \ldots, a[n-1]$. Значения $c[1], c[2], \ldots, c[n-1]$ могут быть произвольными.

- Вторая задача $(s = 1)$ состоит в сортировке чисел $a[0], a[1], \ldots, a[n-1]$ по неубыванию. Более конкретно, для каждого i ($0 \leq i \leq n-1$), $c[i]$ должно быть равно $(1+i)$-му числу в порядке сортировки $a[0], a[1], \ldots, a[n-1]$ (то есть, $c[0]$ должно быть равно наименьшему из чисел).

Помогите Кристоферу написать программы для решения обеих задач, состоящие из не более, чем q инструкций.

Детали реализации

Вам необходимо реализовать функцию:

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

- s: тип задачи.
- n: количество чисел во входных данных.
- k: количество бит в каждом из чисел из входных данных.
- q: максимальное доступное число инструкций.
- Функция будет вызвана ровно один раз и должна построить последовательность инструкций для выполнения нужной задачи.

Для построения последовательности инструкций функция может вызывать любую из перечисленных ниже функций:

```
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)
```

- Каждый из вызовов добавляет в список инструкций $move(t, y)$ $store(t, v)$, $and(t, x, y)$, $or(t, x, y)$, $xor(t, x, y)$, $not(t, x)$, $left(t, x, p)$, $right(t, x, p)$ или $add(t, x, y)$, соответственно.
- Для всех инструкций t, x, y должны быть хотя бы 0 и не больше $m-1$.
- Для всех инструкций t, x, y не обязательно должны быть попарно различными.
- Для инструкций $left$ и $right$ значение p должно быть хотя бы 0 и не более b.
- Для инструкций $store$ длина массива v должна быть равна b.

Вы можете также использовать следующую функцию для облегчения тестирования своего решения:

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

- Любой вызов этой функции будет проигнорирован при проверке вашей программы проверяющей системой.
- В доступном вам грейдере эта функция добавляет инструкцию $print(t)$ в программу.
- Когда грейдер встречает инструкцию $print(t)$, он печатает n k-битных чисел, которые образованы первыми $n \cdot k$ битами регистра t (более подробное описание находится в секции "Пример грейдера").
- t должно удовлетворять неравенствам $0 \leq t \leq m-1$.
- Любой вызов этой функции не учитывается при вычислении общего числа инструкций.

Функция `construct_instructions` должна завершить работу после добавления последней инструкции. После этого программа будет запущена на некотором количестве тестовых примеров, каждый из которых состоит из n k-битных чисел $a[0], a[1], \ldots, a[n-1]$. Решение считается прошедшим тест, если результирующий массив $c[0], c[1], \ldots, c[n-1]$ удовлетворяет следующим условиям:

- если $s = 0$, то $c[0]$ должно быть равно наименьшему из чисел $a[0], a[1], \ldots, a[n-1]$.
- если $s = 1$, то для всех i ($0 \leq i \leq n-1$) $c[i]$ должно быть равно $(1+i)$-му числу в порядке сортировки $a[0], a[1], \ldots, a[n-1]$.

Результатом выполнения вашей программы может являться одно из следующих сообщений:

- `Invalid index`: некорректный (возможно, отрицательный) номер регистра был использован в качестве t, x или y для одного из вызова функций.
- `Value to store is not b bits long`: длина массива v, переданного функции `append_store`, не совпадает с b.
- `Invalid shift value`: значение p, переданное функциям `append_left` или `append_right`, не находится между 0 и b, включительно.
- `Too many instructions`: ваша функция попыталась добавить более q инструкций.

Примеры

Пример 1

Пусть $s = 0$, $n = 2$, $k = 1$, $q = 1000$. Входные данные состоят из двух чисел $a[0]$ и $a[1]$, каждое из которых состоит из $k = 1$ бит. Перед выполнением программы $r[0][0] = a[0]$ и $r[0][1] = a[1]$. Все остальные биты выставлены в 0. После выполнения всех инструкций программы должно выполняться $c[0] = r[0][0] = \min(a[0], a[1])$, что обозначает наименьшее среди чисел $a[0]$ и $a[1]$.

Существует 4 возможных варианта входных данных:

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

- вариант 4: $a[0] = 1, a[1] = 1$

Заметим, что для всех 4 случаев $\min(a[0], a[1])$ совпадает с побитовым И чисел $a[0]$ и $a[1]$. Таким образом, одна из возможных программ может содержать следующие 3 инструкции:

1. `append_move(1, 0)`, которая копирует значение $r[0]$ в $r[1]$.
2. `append_right(1, 1, 1)`, которая рассматривает биты $r[1]$, сдвигает их вправо на 1 бит, и записывает результат в $r[1]$. Так как каждое из чисел состоит из 1-го бита, в результате этой операции $r[1][0]$ совпадает с $a[1]$.
3. `append_and(0, 0, 1)`, которая вычисляет побитовое И чисел $r[0]$ и $r[1]$ и записывает результат в $r[0]$. После выполнения этой инструкции $r[0][0]$ будет содержать побитовое И значений $r[0][0]$ и $r[1][0]$, которое совпадает с побитовым И значений $a[0]$ и $a[1]$, как и требовалось.

Пример 2

Пусть $s = 1,\ n = 2,\ k = 1,\ q = 1000$. Как и в предыдущем примере, существует лишь 4 варианта входных данных. Для каждого из этих 4 вариантов $\min(a[0], a[1])$ совпадает со значением побитового И значений $a[0]$ и $a[1]$, а $\max(a[0], a[1])$ совпадает с побитовым ИЛИ значений $a[0]$ и $a[1]$. Одна из допустимых программ может содержать следующие инструкции:

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)`

После выполнения этих инструкций $c[0] = r[0][0]$ содержит $\min(a[0], a[1])$, а $c[1] = r[0][1]$ содержит $\max(a[0], a[1])$, что является достаточным для сортировки.

Ограничения

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

Подзадачи

1. (10 баллов) $s = 0, n = 2, k \le 2, q = 1000$
2. (11 баллов) $s = 0, n = 2, k \le 2, q = 20$
3. (12 баллов) $s = 0, q = 4000$

4. (25 баллов) $s = 0, q = 150$
5. (13 баллов) $s = 1, n \leq 10, q = 4000$
6. (29 баллов) $s = 1, q = 4000$

Пример грейдера

Пример грейдера считывает данные в следующем формате:

- строка 1 : $s\ n\ k\ q$

Далее следуют несколько строк, каждая из которых описывает тестовый пример. Каждый из тестовых примеров приведен в следующем формате

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

и описывает тестовый пример, в котором входные данные состоят из n чисел $a[0], a[1], ..., a[n-1]$.. Описание всех тестовых случаев заканчивается строкой, содержащей единственное число -1.

Грейдер вызывает функцию `construct_instructions(s, n, k, q)`. Если вызов функции приводит к нарушению каких-либо ограничений из условия задачи, грейдер выводит одно из сообщений об ошибке из параграфа "Детали реализации" и завершает работу. В противном случае грейдер выводит инструкции, добавленных в программу при вызове `construct_instructions(s, n, k, q)`, в порядке их добавления. Для инструкций $store$ массив v будет напечатан от индекса 0 до индекса $b-1$.

После этого грейдер обрабатывает все тестовые примеры. Для каждого из этих тестовых примеров, грейдер запускает построенную программу на этом примере.

Для каждой операции $print(t)$ пусть $d[0], d[1], \ldots, d[n-1]$ это последовательность чисел, в которой для каждого i ($0 \leq i \leq n-1$) значение $d[i]$ равно целочисленному значению последовательности бит от $i \cdot k$ до $(i+1) \cdot k - 1$ регистра t (в момент выполнения операции). Грейдер печатает эту информацию в следующем виде: `register` t: $d[0]\ d[1]\ \ldots\ d[n-1]$.

После выполнения всех инструкций, грейдер печатает результат работы вашей программы.

Если $s = 0$, то результат для каждого тестового примера печатается в следующем формате:

- $c[0]$.

Если $s = 1$, то результат для каждого тестового примера печатается в следующем формате:

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

После выполнения программы на каждом из тестовых примеров, грейдер выводит `number of instructions:` X, где X равно общему числу инструкций в вашей программе.