Регістри двійкового зсуву

Інженер Христина працює над новим типом комп'ютерного процесора.

Процесор має доступ до m різних b-бітних комірок пам'яті (де $m = 100$ та $b = 2000$), які називаються **регістри**, та пронумеровані від 0 до $m - 1$. Позначимо ці регістри $r[0], r[1], \ldots, r[m-1]$. Кожен регістр є масивом з b біт, пронумерованих від 0 (самий правий біт) до $b - 1$ (самий лівий біт). Для кожного i $(0 \le i \le m - 1)$ та кожного j $(0 \le j \le b - 1)$, позначимо j-й біт регістра i як $r[i][j]$.

Для кожної послідовності бітів $d_0, d_1, \ldots, d_{l-1}$ (довільної довжини l), **ціле значення** послідовності дорівнює $2^0 \cdot d_0 + 2^1 \cdot d_1 + \ldots + 2^{l-1} \cdot d_{l-1}$. Ми кажемо що **ціле значення, записане у регістрі** i є цілим значенням послідовності його бітів, тобто $2^0 \cdot r[i][0] + 2^1 \cdot r[i][1] + \ldots + 2^{b-1} \cdot r[i][b-1]$.

У процесора є 9 типів **інструкцій**, що можна використовувати для зміни бітів у регістрах. Кожна з інструкцій працює з одним або більше регістрами та записує результат у один з регістрів. У подальшому ми використовуємо $x := y$ щоб позначати операцію зміни значення x так, що воно стає рівним y. Операції, що виконуються кожним з типів інструкцій, описано нижче.

- $move(t, y)$: Скопіювати масив бітів з регістру y у регістр t. Для кожного j $(0 \le j \le b - 1)$, встановити $r[t][j] := r[y][j]$.
- $store(t, v)$: Встановити регістр t рівним v, де v є масивом з b бітів. Для кожного j $(0 \le j \le b - 1)$, встановити $r[t][j] := v[j]$.
- $and(t, x, y)$: Обчислити побітове AND регістрів x та y, та записати результат до регістру t. Для кожного j $(0 \le j \le b - 1)$, встановити $r[t][j] := 1$ якщо **обидва** $r[x][j]$ та $r[y][j]$ є 1, інакше встановити $r[t][j] := 0$.
- $or(t, x, y)$: Обчислити побітове OR регістрів x та y, та записати результат у регістр t. Для кожного j $(0 \le j \le b - 1)$, встановити $r[t][j] := 1$ якщо **принаймні один** з $r[x][j]$ та $r[y][j]$ є 1, інакше встановити $r[t][j] := 0$.
- $xor(t, x, y)$: Обчислити побітовий XOR регістрів x та y, та записати результат у регістр t. Для кожного j $(0 \le j \le b - 1)$, встановити $r[t][j] := 1$ якщо **рівно один** з $r[x][j]$ та $r[y][j]$ є 1, інакше встановити $r[t][j] := 0$.
- $not(t, x)$: Обчислити побітове NOT регістра x, та записати результат у регістр t. Для кожного j $(0 \le j \le b - 1)$, встановити $r[t][j] := 1 - r[x][j]$.
- $left(t, x, p)$: Зсунути біти регістра x вліво на p, та записати результат у регістр t. Результатом зсуву бітів у регістрі x вліво на p є масив v що складається з b бітів. Для

кожного j $(0 \le j \le b-1)$, $v[j] = r[x][j-p]$ якщо $j \ge p$, інакше $v[j] = 0$. Для кожного j $(0 \le j \le b-1)$, встановимо $r[t][j] := v[j]$.

- $right(t, x, p)$: Зсунути біти регістра x вправо на p, та записати результат у регістр t. Результатом зсуву бітів у регістрі x вправо на p є масив v що складається з b бітів. Для кожного j $(0 \le j \le b-1)$, $v[j] = r[x][j+p]$ якщо $j \le b-1-p$, інакше $v[j] = 0$. Для кожного j $(0 \le j \le b-1)$, встановимо $r[t][j] := v[j]$.

- $add(t, x, y)$: Додати цілі значення, що зберігаються у регістрах x та y,та записати результат у регістр t. Ця операція виконується за модудем 2^b. Формально, нехай X буде цілим числом, що зберігається у регістрі x, а Y буде цілим числом, що зберігається у регістрі y перед операцією. Нехай T буде цілим числом, що зберігається у регістрі t після операції. Якщо $X + Y < 2^b$, встановити біти t так, що $T = X + Y$. Інакше, встановити біти t так, що $T = X + Y - 2^b$.

Христина хоче розв'язувати два типи задач, використовуючи новий процесор. Тип задачі позначається цілим s. Для обох типів задач, ви маєте створити **програму**, тобто послідовність інструкцій, що визначені вище.

Вхід програми складається з n цілих $a[0], a[1], \ldots, a[n-1]$, кожне з яких має k-бітів, тобто $a[i] < 2^k$ ($0 \le i \le n-1$). Перед виконанням програми всі вхідні числа записуються послідовно у регістр 0, так що для кожного i $(0 \le i \le n-1)$ ціле значення послідовності з k бітів $r[0][i \cdot k], r[0][i \cdot k + 1], \ldots, r[0][(i+1) \cdot k - 1]$ дорівнює $a[i]$. Зауважте, що $n \cdot k \le b$. Усі інші біти у регістрі 0 (тобто з індексами між $n \cdot k$ та $b-1$, включно) та всі біти у всіх інших регістрах ініціалізуються 0.

Запуск програми полягає у послідовному виконанні інструкцій. Після виконання останньої інструкції **вивід** програми обчислюється базуючись на фінальному значнні бітів у регістрі 0. Більш докладно, вивід є послідовністю n цілих $c[0], c[1], \ldots, c[n-1]$, де для кожного i ($0 \le i \le n-1$), $c[i]$ є цілим значенням послідовності, що складається з бітів $i \cdot k$ до $(i+1) \cdot k - 1$ регістра 0. Зауважте, що після виконання програми решта бітів регістра 0 (з індексами починаючи з $n \cdot k$) та усі біти інших регістів можуть бути довільними.

- Першою задачою $(s = 0)$ є знайти найменше ціле серед вхідних цілих $a[0], a[1], \ldots, a[n-1]$. А саме, $c[0]$ має бути мінімумом $a[0], a[1], \ldots, a[n-1]$. Значення $c[1], c[2], \ldots, c[n-1]$ можуть бути довільними.

- Другою задачею $(s = 1)$ є відсортувати вхідні цілі $a[0], a[1], \ldots, a[n-1]$ у неспадаючому порядку. А саме, для кожного i ($0 \le i \le n-1$), $c[i]$ має бути рівним $1+i$ -ому найменшому цілому серед $a[0], a[1], \ldots, a[n-1]$ (тобто, $c[0]$ є найменшим серед усих вхідних цілих).

Надайте Христині програми, що складаються не більше ніж з q інструкцій кожна, які можуть розв'язувати ці задачі.

Деталі реалізації

Ви маєте реалізувати наступну процедуру:

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

- s: тип задачі.
- n: кількість цілих на вході.
- k: кількість бітів у кожному вхідному цілому.
- q: максимальна дозволена кількість інструкцій.
- Ця процедура викликається рівно один раз та має побудувати послідовність інструкцій, що виконує потрібну задачу.

Ця процедура має викликати одну або більше з наступних процедур щоб побудувати послідовність інструкцій:

```
void _move(int t, int y)
void _store(int t, bool[] v)
void _and(int t, int x, int y)
void _or(int t, int x, int y)
void _xor(int t, int x, int y)
void _not(int t, int x)
void _left(int t, int x, int p)
void _right(int t, int x, int p)
void _add(int t, int x, int y)
```

- Кожна процедура додає інструкцію $move(t, y)$ $store(t, v)$, $and(t, x, y)$, $or(t, x, y)$, $xor(t, x, y)$, $not(t, x)$, $left(t, x, p)$, $right(t, x, p)$ або $add(t, x, y)$ до програми, відповідно.
- Для усіх відповідних інструкцій t, x, y мають бути не менше 0 та не більше $m - 1$.
- Для усих відповідних інструкцій t, x, y не обов'язково попарно різні.
- Для інструкцій $left$ та $right$, p має не менше 0 та не більше b.
- Для інструкції $store$ довжина v має бути b.

Ви також можете викликати наступну процедуру щоб було зручніше тестувати ваш розв'язок:

```
void _print(int t)
```

- Усі виклики цієї процедури буде проігноровано підчас оцінки вашого розв'язку.
- У прикладі модуля перевірки ця процедура додасть до програми операцію $print(t)$.
- Коли приклад модуля перевірки зустрічає операцію $print(t)$ під час виконання програми, він друкує n k-бітних цілих, що утворено першими $n \cdot k$ бітами регістра t (дивіться секцію "Приклад модуля перевірки", щоб отримати деталі).
- t має задовольняти $0 \leq t \leq m - 1$.
- Виклик цієї процедури не враховується у кількість використаних інструкцій.

Після додавання останньої інструкції `construct_instructions` має завершитись. Після цього програма перевіряється на певній кількості тестів, кожен з яких задає вхід, що складається з n k

-бітних цілих $a[0], a[1], \ldots, a[n-1]$. Ваш розв'язок проходить тест, якщо вивід програми $c[0], c[1], \ldots, c[n-1]$ для заданих вхідних даних задовольняє наступні вимоги:

- Якщо $s = 0$, $c[0]$ має бути найменшим серед $a[0], a[1], \ldots, a[n-1]$.
- Якщо $s = 1$, для всіх i ($0 \leq i \leq n-1$), $c[i]$ має бути $1 + i$-м найменшим цілим серед $a[0], a[1], \ldots, a[n-1]$.

Оцінка ваших розвязків може дати в результаті одну з наступних помиок:

- `Invalid index`: неправильний (можливо від'ємний) індекс регістра було відправлено в якості параметра t, x, або y у якомусь із викликів процедури.
- `Value to store is not b bits long`: довжина v наданого процедурі `_store` не дорівнює b.
- `Invalid shift value`: величина p наданого `_left` або `_right` не в межах 0 та b включно.
- `Too many instructions`: ваша процедура перевищила обмеження q на кількість інструкццій.

Приклади

Приклад 1

Припустимо, що $s = 0$, $n = 2$, $k = 1$, $q = 1000$. Є 2 вхідних цілих $a[0]$ і $a[1]$, кожне з яких складається із $k = 1$ біт. До запуска програми, $r[0][0] = a[0]$ і $r[0][1] = a[1]$. Всі інші біти у процесорі дорівнюють 0. Після виконання всіх інструкцій програмою потрібно отримати $c[0] = r[0][0] = \min(a[0], a[1])$, що є мінімумом $a[0]$ та $a[1]$.

Можливо лише 4 варіанти вхідних даних:

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

Можемо помітити, що для всіх 4 випадків $\min(a[0], a[1])$ дорівнює побітовому AND чисел $a[0]$ та $a[1]$. Тому можливим розвязком буде побудова програми наступним чином:

1. `_move(1, 0)`, що додає інструкцію скопіювати $r[0]$ to $r[1]$.
2. `_right(1, 1, 1)`, що додає інструкцію, що бере всі біти у $r[1]$, зсуває їх вправо на 1 біт, і зберігає результат у $r[1]$. Оскільки кожне ціле має довжину 1 біт, в результаті маємо, що $r[1][0]$ дорівнюватиме $a[1]$.
3. `_and(0, 0, 1)`, що додає інструкцію зробити побітовий AND $r[0]$ та $r[1]$, і зберегти результат до $r[0]$. Після виконання цієї інструкції, $r[0][0]$ стає рівним побітовому AND з $r[0][0]$ та $r[1][0]$, що дорівнює побітовому AND з $a[0]$ та $a[1]$, як і було потрібно.

Приклад 2

Припустимо, що $s = 1,\ n = 2,\ k = 1,\ q = 1000$. Як і в попередньому прикладі, є рівно 4 варіанти вхідних даних. Для всіх 4 випадків, $\min(a[0], a[1])$ це побітовий AND $a[0]$ та $a[1]$, а $\max(a[0], a[1])$ це побітовий OR $a[0]$ та $a[1]$. Можливим варіантом розв'язку є наступна послідовність команд:

1. `_move(1,0)`
2. `_right(1,1,1)`
3. `_and(2,0,1)`
4. `_or(3,0,1)`
5. `_left(3,3,1)`
6. `_or(0,2,3)`

Після виконання цих інструкцій $c[0] = r[0][0]$ містить $\min(a[0], a[1])$, і $c[1] = r[0][1]$ містить $\max(a[0], a[1])$, що сортує вихідні дані.

Обмеження

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

Підзадачі

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

Приклад модуля перевірки

Модуль перевірки зчитує дані у наступному форматі:

- рядок $1:\ s\ n\ k\ q$

Потім йдуть t рядків, кожен з яких описує окремий тест. Кожен тест має наступний формат:

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

і описує тест, вхідні дані якого складаються з n цілих $a[0], a[1], \ldots, a[n-1]$. Після всіх тестів йде окремий рядок, що містить єдину -1.

Модуль перевірки спочатку виклакає `construct_instructions(s, n, k, q)`. Якщо цей виклик порушує певні обмеження, описані в умові, він виводить одне із описаних в деталях реалізаціїї повідомлень з помилкою та закінчує виконання. В іншому випадку, модуль перевірки спочатку виводить всі інструкції додані `construct_instructions(s, n, k, q)` по порядку. Для інструкцій $store,\ v$ виводиться з індекса 0 до індекса $b-1$.

Потім модуль перевірки по порядку виконує тестові випадки. Для всіх тестів він запускає сконструйовану програму на тестових даних.

Для кожної команди $print(t)$, нехай $d[0], d[1], \ldots, d[n-1]$ є така послідовність цілих, що для всіх i ($0 \le i \le n-1$), $d[i]$ є цілим значенням послідовності бітів від $i \cdot k$ до $(i+1) \cdot k - 1$ регістра t (коли операція виконана). Модуль перевірки виводить послідовність у наступному форматі: `register` t: $d[0]\ d[1]\ \ldots\ d[n-1]$.

Коли виконані всі інструкції програми, модуль перевірки друкує вивід програми.

Якщо $s = 0$, вивід модуля перевірки для кожного тесту має такий формат:

- $c[0]$.

Якщо $s = 1$, вивід модуля перевірки для кожного тесту має такий формат:

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

Після викнанні всіх тестів модуль перевірки друкує `number of instructions:` X де X кількість інструкцій у вашій програмі.