移位寄存器

工程師克里斯托弗正在研究一種新型計算機處理器。

處理器可以訪問 m 個不同的 b-位元的存儲單元 (其中 $m = 100$ 和 $b = 2000$), 它們被稱為**寄存器**, 編號從 0 到 $m - 1$。 我們用 $r[0], r[1], \ldots, r[m-1]$ 來表示這些寄存器。 每個寄存器都是一個 b 位的數組, 編號從 0 (最右邊的位)到 $b - 1$ (最左邊的位)。 對於每個 i $(0 \leq i \leq m-1)$ 和每個 j $(0 \leq j \leq b-1)$, 我們用 $r[i][j]$ 來表示寄存器 i 的第 j 位。

對於任何位元序列 $d_0, d_1, \ldots, d_{l-1}$ (任意長度 l), 該序列的**整數值**等於 $2^0 \cdot d_0 + 2^1 \cdot d_1 + \ldots + 2^{l-1} \cdot d_{l-1}$。 我們說寄存器中存儲的**整數值** i 是其位元序列的整數值, 即 $2^0 \cdot r[i][0] + 2^1 \cdot r[i][1] + \ldots + 2^{b-1} \cdot r[i][b-1]$。

處理器有 9 類型的**指令**, 可用於修改寄存器中的位元。每條指令對一個或多個寄存器進行操作, 並將輸出存儲在其中一個寄存器中。在下文中, 我們使用 $x := y$ 表示更改 x 的值使其等於 y 的操作。每種指令執行的操作如下所述。

- $move(t, y)$ ：將寄存器 y 中的位元數組複製到寄存器 t。對於每個 j $(0 \leq j \leq b-1)$, 使 $r[t][j] := r[y][j]$。
- $store(t, v)$ ：設置寄存器 t 使其等於 v, 其中 v 是 b 位元的數組。對於每個 j $(0 \leq j \leq b-1)$, 使 $r[t][j] := v[j]$。
- $and(t, x, y)$ ：取寄存器 x 和 y 的按位進行"和"運算, 並將結果存入寄存器 t。對於每個 j $(0 \leq j \leq b-1)$, 設置 $r[t][j] := 1$ 如果 $r[x][j]$ **和** $r[y][j]$ **都是** 1, 否則設置 $r[t][j] := 0$。
- $or(t, x, y)$ ：取寄存器 x 和 y 的按位進行"或"運算, 結果存入寄存器 t。對於每個 j $(0 \leq j \leq b-1)$, 設置 $r[t][j] := 1$ 若**至少有一個** $r[x][j]$ 或 $r[y][j]$ 是 1, 否則設置 $r[t][j] := 0$。
- $xor(t, x, y)$ ：取寄存器 x 和 y 的按位進行"異或"運算, 並將結果存儲在寄存器 t 中。對於每個 j $(0 \leq j \leq b-1)$, 設置 $r[t][j] := 1$ 如果 $r[x][j]$ 及 $r[y][j]$ 中**正好只有一個人** 為 1, 否則設置 $r[t][j] := 0$。
- $not(t, x)$ ：取寄存器 x 的按位進行"非"運算, 並將結果存入寄存器 t。對於每個 j $(0 \leq j \leq b-1)$, 設置 $r[t][j] := 1 - r[x][j]$。
- $left(t, x, p)$ ：將寄存器 x 中的所有位元左移 p 個位, 並將結果存入寄存器 t。將寄存器 x 中的位向左移動 p 的結果是一個由 b 位組成的數組 v。對於每個 j $(0 \leq j \leq b-1)$, 如果 $j \geq p$, $v[j] = r[x][jp]$, 並且 $v[j] = 0$ 除此以外。對於每個 j $(0 \leq j \leq b-1)$, 設置 $r[t][j] := v[j]$。
- $right(t, x, p)$ ：將寄存器 x 中的所有位右移 p 個位, 並將結果存入寄存器 t。將寄存器 x 中的位右移 p 的結果是一個由 b 位組成的數組 v。對於每個 j $(0 \leq j \leq b-1)$,

$v[j] = r[x][j+p]$ 如果 $j \leq b-1-p$, 以及 $v[j] = 0$ 否則。對於每個 j $(0 \leq j \leq b-1)$, 設置 $r[t][j] := v[j]$。

- $add(t, x, y)$ ：將寄存器 x 和寄存器 y 中存儲的整數值相加, 並將結果存儲在寄存器 t 中。加法以 2^b 為模進行。 具體來說, 設 X 為寄存器 x 中存儲的整數值, Y 為操作前寄存器 y 中存儲的整數值。設 T 為運算後存儲在寄存器 t 中的整數值。如果 $X + Y < 2^b$, 則設置 t 的位元數組, 使得 $T = X + Y$。 否則, 設置 t 的位元數組, 使得 $T = X + Y - 2^b$。

克里斯托弗希望您使用新處理器解決兩類任務。任務的類型由整數 s 表示。對於這兩種類型的任務, 您都需要生成一個**程序**, 即一個由上面定義的指令組成的序列。

程序的**輸入**由 n 個整數 $a[0], a[1], \ldots, a[n-1]$ 組成, 每個整數都有 k 位, 即 $a[i] < 2^k$ ($0 \leq i \leq n-1$)。 在程序執行之前, 所有輸入的數字都按順序存儲在寄存器 0 中, 這樣對於每個 i $(0 \leq i \leq n-1)$ k 位序列的整數值 $r[0][i \cdot k], r[0][i \cdot k + 1], \ldots, r[0][(i+1) \cdot k - 1]$ 等於 $a[i]$。注意 $n \cdot k \leq b$。 寄存器 0 中的所有其他位 (即索引在 $n \cdot k$ 和 $b-1$ 之間的那些位, 包含) 和所有其他寄存器中的所有位都初始化為 0。

運行一個程序就是按順序執行它的指令。 執行完最後一條指令後, 根據寄存器 0 中位的最終值來決定程序的**輸出**。 具體來說, 輸出是 n 個整數 $c[0], c[1], \ldots, c[n-1]$ 的序列, 其中對於每個 i ($0 \leq i \leq n-1$), $c[i]$ 是寄存器 0 的 $i \cdot k$ 到 $(i+1) \cdot k - 1$ 位組成的序列的整數值。 請注意, 在運行程序後, 寄存器 0 的剩餘位 (索引至少為 $n \cdot k$) 和所有其他寄存器的所有位都可以是任意的。

- 第一個任務 $(s = 0)$ 是在輸入整數 $a[0], a[1], \ldots, a[n-1]$ 中找到最小的整數。具體來說, $c[0]$ 必須是 $a[0], a[1], \ldots, a[n-1]$ 中的最小值。 $c[1], c[2], \ldots, c[n-1]$ 的值可以是任意的。
- 第二個任務 $(s = 1)$ 是按非遞減順序對輸入整數 $a[0], a[1], \ldots, a[n-1]$ 進行排序。具體來說, 對於每個 i ($0 \leq i \leq n-1$), $c[i]$ 應該等於 $a[0], a[1], \ldots, a[n-1]$ 中第 $1+i$ 個最小的整數 (即 $c[0]$ 是輸入整數中最小的整數)。

請為克里斯托弗提供程序, 每個程序最多包含 q 條指令, 可以解決這些任務。

實現細節

您應該編寫以下程序：

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

- s ：任務類型。
- n ：輸入中的整數數量。
- k ：每個輸入整數的位數。
- q ：允許的最大指令數。
- 這個函數只恰好被調用一次, 並且應該構造一個指令序列來執行所需的任務。

此函數應調用以下一個或多個函數來構造指令序列：

```
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)
```

- 每個函數相應增添一個 $move(t, y)$, $store(t, v)$, $and(t, x, y)$, $or(t, x, y)$, $xor(t, x, y)$, $not(t, x)$, $left(t, x, p)$, $right(t, x, p)$ 或 $add(t, x, y)$ 指令到指令序列的末端。
- 對於所有相關的指令, t, x, y 必須至小為 0, 最大為 $m - 1$。
- 對於所有相關的指令, t, x, y 不一定成對不同。
- 對於 $left$ 和 $right$ 指令, p 必須至少為 0 且至多為 b。
- 對於 $store$ 指令, v 的長度必須為 b。

您也可以調用以下函數來幫助您測試您的解決方案：

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

- 在實際對您的解決方案進行評分時, 系統將忽略對此函數的任何調用。
- 在樣例評分程式中, 此函數將 $print(t)$ 操作附加到程序中。
- 當樣本評分程式在程序執行函數中遇到 $print(t)$ 操作時, 它會打印出開頭 n 個 k 位記仔器的整數, 這些整數由寄存器 t 的前 $n \cdot k$ 位組成(參見詳情請參見“樣例評分程式”部分)。
- t 必須滿足 $0 \leq t \leq m - 1$。
- 這函數的任何調用都不會計算入指令的總數內。

在添加最後一條指令後, `construct_instructions` 應該返回。 然後在一定數量的測試用例上評估該程序, 每個測試用例指定一個由 n 個 k 位整數 $a[0], a[1], \ldots, a[n - 1]$ 組成的輸入。 如果對應測試用例的輸入, 你的程序所產生的輸出 $c[0], c[1], \ldots, c[n - 1]$ 能滿足以下條件, 則您的解決方案通過給定的測試用例：

- 如果 $s = 0$, $c[0]$ 應該是 $a[0], a[1], \ldots, a[n - 1]$ 中的最小值。
- 如果 $s = 1$, 對於每個 i ($0 \leq i \leq n - 1$), $c[i]$ 應該是 $a[0]$ 中第 $1 + i$ 個最小的整數, $a[1], \ldots, a[n - 1]$。

解決方案的評分程式可能會輸出以下錯誤消息之一：

- `Invalid index`：對於其中一個函數的某些調用, 提供了不正確(可能為負)的寄存器索引作為參數 t, x 或 y。
- `Value to store is not b bits long`：提供給 `append_store` 的 v 的長度不等於 b。
- `invalid shift value`：提供給`append_left` 或`append_right` 的 p 值不在 0 和 b 之間。
- `Too many instructions`：您的程序試圖附加超過 q 條指令。

例子

樣例 1

假設 $s = 0$, $n = 2$, $k = 1$, $q = 1000$。有兩個輸入整數 $a[0]$ 和 $a[1]$, 每個都有 $k = 1$ 位。在程序執行之前, $r[0][0] = a[0]$ 和 $r[0][1] = a[1]$。處理器中的所有其他位都設置為 0。程序中所有指令執行完畢後, 我們需要有 $c[0] = r[0][0] = \min(a[0], a[1])$, 即 a 的最小值 $[0]$ 和 $a[1]$。

程序只有 4 個可能的輸入：

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

我們可以注意到, 對於所有 4 種情況, $\min(a[0], a[1])$ 等於 $a[0]$ 和 $a[1]$ 的按位"和"運算。因此, 一種可能的解決方案是通過進行以下調用來構建程序：

1. `append_move(1, 0)`, 附加一條指令將 $r[0]$ 複製到 $r[1]$。
2. `append_right(1, 1, 1)`, 附加一條指令, 取 $r[1]$ 中的所有位, 將它們向右移動 1 位, 然後將結果存回 $r[1]$。由於每個整數的長度為 1 位, 這導致 $r[1][0]$ 等於 $a[1]$。
3. `append_and(0, 0, 1)`, 附加一條指令對 $r[0]$ 和 $r[1]$ 進行按位"和"運算, 然後將結果存入 $r[0]$。這條指令執行後, $r[0][0]$ 被設置為 $r[0][0]$ 和 $r[1][0]$ 的按位"和", 等於按位"和"的運動, 該運算其實相當於 $a[0]$ 和 $a[1]$ 進行 "和" 運算。

樣例 2

假設 $s = 1$, $n = 2$, $k = 1$, $q = 1000$。與前面的樣例一樣, 程序只有 4 個可能的輸入。對於所有 4 種情況, $\min(a[0], a[1])$ 是 $a[0]$ 和 $a[1]$ 的按位"和", 而 $\max(a[0], a[1])$ 是 $a[0]$ 和 $a[1]$ 的按位"或"。一個可能的解決方案是進行以下調用：

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)`

執行完這些指令後, $c[0] = r[0][0]$ 包含 $\min(a[0], a[1])$, $c[1] = r[0][1]$ 包含 $\max(a[0], a[1])$, 即對輸入成功進行排序。

限制

- $m = 100$
- $b = 2000$
- $0 \le s \le 1$
- $2 \le n \le 100$
- $1 \le k \le 10$
- $q \le 4000$

- $0 \leq a[i] \leq 2^k - 1$ (對於所有 $0 \leq i \leq n - 1$)

子任務

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

樣例評分程式

樣例評分程式按以下格式讀取輸入：

- 第 1 行: s n k q

這之後有若干行, 每行描述一組測試數據。 每個測試數據都以以下格式提供：

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

它描述了一組測試數據, 其輸入由 n 個整數 $a[0], a[1], ..., a[n-1]$ 組成。 在所有測試數據的描述後跟著一行僅包含 -1。

樣例評分程式首先調用 `construct_instructions(s, n, k, q)`。如果此調用違反了問題陳述中描述的某些限制, 則樣例評分程式將打印出在 “實現細節” 部分末尾所列出的錯誤消息之一併退出。 否則, 樣本評分程式首先按順序打印出在 `construct_instructions(s, n, k, q)` 時所附加上的每條指令。對於 $store$ 指令, v 會從索引 0 打印到索引 $b - 1$。

然後, 樣例評分程式會按順序處理測試數據。 對於每個測試數據, 它在測試數據的輸入上運行所構建的程序。

對於每個 $print(t)$ 操作, 設 $d[0], d[1], \ldots, d[n-1]$ 是一個整數序列, 使得對於每個 i ($0 \leq i \leq n - 1$), $d[i]$ 是寄存器 t 的 $i \cdot k$ 到 $(i + 1) \cdot k - 1$ 位序列的整數值 (當操作完成後)。 評分程式以下列格式打印這個序列： `register` t: $d[0]$ $d[1]$ $\ldots$ $d[n-1]$。

執行完所有指令後, 樣例評分程式將打印程序的輸出。

如果 $s = 0$, 每個測試用例的樣本評分程式的輸出格式如下：

- $c[0]$。

如果 $s = 1$, 則每個測試用例的評分程式輸出格式如下：

- $c[0]$ $c[1]$ $\ldots$ $c[n-1]$。

執行完所有測試用例後, 評分程式打印 `number of instructions:` X, 其中 X 是程序中的指令數目。