# 位元移位暫存器

工程師 Christopher 正利用一種新型的計算機處理器 (computer processor) 進行工作。

此處理器能夠存取 $m$ 個不同的 $b$位元 (bit) 記憶單元 (其中 $m = 100$ 且 $b = 2000$)，稱為 **暫存器** (registers)，由 $0$ 編號至 $m - 1$。我們以 $r[0], r[1], \ldots, r[m-1]$ 來表示這些暫存器。每個暫存器為 $b$ 個位元的陣列，由 $0$ (最右邊的位元) 編號至 $b - 1$ (最左邊的位元)。對 $i$ $(0 \leq i \leq m-1)$ 與 $j$ $(0 \leq j \leq b-1)$，我們以 $r[i][j]$ 表示暫存器 $i$ 的第 $j$ 個位元。

對任何位元序列 $d_0, d_1, \ldots, d_{l-1}$ (任意長度 $l$)，此序列的 **整數值** 等於 $2^0 \cdot d_0 + 2^1 \cdot d_1 + \ldots + 2^{l-1} \cdot d_{l-1}$。我們稱 **儲存於暫存器 $i$ 的整數值** 為其位元序列的整數值，即 $2^0 \cdot r[i][0] + 2^1 \cdot r[i][1] + \ldots + 2^{b-1} \cdot r[i][b-1]$。

此處理器有 $9$ 種 **指令** (instructions) 可用來修訂暫存器中的位元。每個指令可作用於一或多個暫存器並將結果儲存於某個暫存器。以下，我們用 $x := y$ 來表示將 $x$ 的值改變為 $y$ 的操作。每種指令可進行的操作描述如下。

- $move(t, y)$: 複製暫存器 $y$ 的位元陣列至暫存器 $t$。對每個 $j$ $(0 \leq j \leq b-1)$ 設 $r[t][j] := r[y][j]$。
- $store(t, v)$: 將暫存器 $t$ 設為 $v$，其中 $v$ 為一位元陣列。對每個 $j$ $(0 \leq j \leq b-1)$，設 $r[t][j] := v[j]$。
- $and(t, x, y)$: 對暫存器 $x$ 和 $y$ 做 bitwise-AND，並將結果存於暫存器 $t$。對每個 $j$ $(0 \leq j \leq b-1)$，若 $r[x][j]$ 與 $r[y][j]$ **皆為** $1$，則設 $r[t][j] := 1$，否則設 $r[t][j] := 0$。
- $or(t, x, y)$: 對暫存器 $x$ 和 $y$ 做 bitwise-OR，並將結果存於暫存器 $t$。對每個 $j$ $(0 \leq j \leq b-1)$，若 $r[x][j]$ 與 $r[y][j]$ **至少一者**為 $1$，則設 $r[t][j] := 1$，否則設 $r[t][j] := 0$。
- $xor(t, x, y)$: 對暫存器 $x$ 和 $y$ 做 bitwise-XOR，並將結果存於暫存器 $t$。對每個 $j$ $(0 \leq j \leq b-1)$，若 $r[x][j]$ 與 $r[y][j]$ **恰有一者**為 $1$，則設 $r[t][j] := 1$，否則設 $r[t][j] := 0$。
- $not(t, x)$: 對暫存器 $x$ 做 bitwise-NOT，並將結果存於暫存器 $t$。對每個 $j$ $(0 \leq j \leq b-1)$，設 $r[t][j] := 1 - r[x][j]$。
- $left(t, x, p)$: 將暫存器 $x$ 中的所有位元向左移 $p$ 位，並將結果存於暫存器 $t$。將暫存器 $x$ 的位元左移 $p$ 位的結果為一 $b$ 個位元的陣列 $v$。對每個 $j$ $(0 \leq j \leq b-1)$，若 $j \geq p$，則 $v[j] = r[x][j-p]$，否則 $v[j] = 0$。對每個 $j$ $(0 \leq j \leq b-1)$，設 $r[t][j] := v[j]$。
- $right(t, x, p)$: 將暫存器 $x$ 中的所有位元向右移 $p$ 位，並將結果存於暫存器 $t$。將暫存器 $x$ 的位元右移 $p$ 位的結果為一 $b$ 個位元的陣列 $v$。對每個 $j$ $(0 \leq j \leq b-1)$，若

$j \le b-1-p$，則 $v[j] = r[x][j+p]$，否則 $v[j] = 0$。對每個 $j$ $(0 \le j \le b-1)$，設 $r[t][j] := v[j]$。

- $add(t, x, y)$: 將暫存器 $x$ 與暫存器 $y$ 的整數值相加，並將結果存於暫存器 $t$。此加法為模 $2^b$ 之結果。正式地說，令 $X$ 為操作前儲存於暫存器 $x$ 的整數值，而 $Y$ 為操作前儲存於暫存器 $y$ 的整數值。令 $T$ 為操作後儲存於暫存器 $t$ 的整數值。若 $X + Y < 2^b$，則設置 $t$ 的位元使其滿足 $T = X + Y$。否則，設置 $t$ 的位元使其滿足 $T = X + Y - 2^b$。

Christopher 想請你利用此新型處理器解決兩類型的任務。任務類型以一個整數 $s$ 表示。對這兩類任務，你需要產生一個 **程式** (program)，即一連串上述指令的序列。

此程式的 **輸入** (input) 為 $n$ 個整數 $a[0], a[1], \ldots, a[n-1]$，每個皆為 $k$ 位元，即 $a[i] < 2^k$ ( $0 \le i \le n-1$)。 在此程式執行前，所有的輸入數字被依序儲存於暫存器 $0$ 使得對每個 $i$ $(0 \le i \le n-1)$ 此 $k$ 位元序列 $r[0][i \cdot k], r[0][i \cdot k + 1], \ldots, r[0][(i+1) \cdot k - 1]$ 的整數值等於 $a[i]$。注意 $n \cdot k \le b$。 暫存器 $0$ 中所有其他位元 (即註標 (index) 介於 $n \cdot k$ 及 $b-1$ 間，包含此二值) 以及所有其他暫存器的所有位元都被初始化為 $0$。

執行一個程式即為依序執行其中的指令。 在最後一個指令被執行後，此程式之 **輸出** (output) 會基於暫存器 $0$ 中最後的位元值計算。 具體來說，此輸出為一 $n$ 個整數的序列 $c[0], c[1], \ldots, c[n-1]$，其中對每個 $i$ ( $0 \le i \le n-1$)， $c[i]$ 為暫存器 $0$ 中位元 $i \cdot k$ 至位元 $(i+1) \cdot k - 1$ 之位元序列的整數值。 注意在執行完此程式後，暫存器 $0$ 中的其他位元 (註標至少 $n \cdot k$) 以及其他所有暫存器的所有位元可能為任意值。

- 第一類任務 $(s = 0)$ 為找出輸入整數 $a[0], a[1], \ldots, a[n-1]$ 中最小者。 具體來說， $c[0]$ 必須為 $a[0], a[1], \ldots, a[n-1]$ 之最小者。 $c[1], c[2], \ldots, c[n-1]$ 可為任意值。
- 第二類任務 $(s = 1)$ 為將輸入整數 $a[0], a[1], \ldots, a[n-1]$ 以非遞減順序 (nondecreasing order) 排序。具體來說，對每個 $i$ ( $0 \le i \le n-1$)， $c[i]$ 應等於 $a[0], a[1], \ldots, a[n-1]$ 中第 $1+i$ 小的整數 (即 $c[0]$ 為輸入整數中最小的)。

請提供 Christopher 一些程式來解決這些任務，每個程式都由至多 $q$ 個指令構成。

## 實作細節

你應實作下列函式：

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

- $s$: 任務類型。
- $n$: 輸入中的整數個數。
- $k$: 每個輸入整數的位元數。
- $q$: 允許使用的指令數上界。
- 此函式被呼叫恰好一次且應建構一指令序列來完成要求的任務。

此函式應呼叫一或多個下列的函式來建構指令序列：

```
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)
```

- 上述函式分別添加一 $move(t, y)$、$store(t, v)$ 、$and(t, x, y)$、 $or(t, x, y)$、$xor(t, x, y)$、$not(t, x)$、$left(t, x, p)$、$right(t, x, p)$ 與 $add(t, x, y)$ 指令至欲建構的程式中。
- 對所有相關的指令，$t$、 $x$、 $y$ 必至少為 $0$ 且至多為 $m - 1$。
- 對所有相關的指令，$t$、 $x$、 $y$ 不一定兩兩相異。
- 對 $left$ 與 $right$ 指令， $p$ 必至少為 $0$ 且至多為 $b$。
- 對 $store$ 指令， $v$ 的長度必為 $b$。

你也可以呼叫下列函式來幫助你測試你的解：

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

- 在評測你的程式時，任何對於此函式的呼叫都將被忽略。
- 在範例評分程式中，此函式在程式中添加了一個 $print(t)$ 操作。
- 當範例評分程式在執行程式時遇到一個 $print(t)$ 操作時，它將印出 $n$ 個由暫存器 $t$ 前 $n \cdot k$ 個位元所定義的 $k$ 位元整數 (細節請參考 "範例評分程式" 一節)
- $t$ 必滿足 $0 \leq t \leq m - 1$。
- 對此函式的任何呼叫將不被計入建構的指令數中。

在添加最後一個指令後，`construct_instructions` 應回傳 (return)。

此程式將以若干筆測試資料評測，每筆測試資料會給定 $n$ 個 $k$ 位元整數 $a[0], a[1], \ldots, a[n-1]$ 做為輸入。 對於給定的輸入，若程式的輸出 $c[0], c[1], \ldots, c[n-1]$ 滿足下列條件，則你的解通過該筆測試資料：

- 若 $s = 0$， $c[0]$ 應為 $a[0], a[1], \ldots, a[n-1]$ 之最小值。
- 若 $s = 1$，對每個 $i$ ( $0 \leq i \leq n-1$)， $c[i]$ 應是 $a[0], a[1], \ldots, a[n-1]$ 中第 $1 + i$ 小的整數。

你的解被評分後可能產生下列錯誤訊息之一：

- `Invalid index`: an incorrect (possibly negative) register index was provided as parameter $t$, $x$ or $y$ for some call of one of the procedures. (此函式中某些呼叫傳遞了錯誤的 (可能是負值) 暫存器註標至參數 $t$ 、$x$ 或 $y$。)
- `Value to store is not b bits long`: the length of $v$ given to `append_store` is not equal to $b$. (傳遞至 `append_store` 之 $v$ 的長度不等於 $b$。 )

- `Invalid shift value`: the value of $p$ given to `append_left` or `append_right` is not between $0$ and $b$ inclusive. (傳遞至 `append_left` 或 `append_right` 之 $p$ 值並不界於 $0$ 與 $b$ 之間 (包含 $0$ 與 $b$)。)
- `Too many instructions`: your procedure attempted to append more than $q$ instructions. (你的函式試圖添加超過 $q$ 個指令。)

# 範例

## 範例 1

假設 $s = 0$、$n = 2$、$k = 1$、 $q = 1000$。有兩個輸入整數 $a[0]$ 和 $a[1]$，每個都是 $k = 1$ 位元。程式執行前， $r[0][0] = a[0]$ 且 $r[0][1] = a[1]$。此處理器中其他所有位元都被設為 $0$。在執行完程式中所有指令後，需得 $c[0] = r[0][0] = \min(a[0], a[1])$，即 $a[0]$ 與 $a[1]$ 之最小值。

此程式僅有 4 種可能的輸入：

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

注意對這 4 個 cases 的任何一個， $\min(a[0], a[1])$ 等同於 $a[0]$ 和 $a[1]$ 做 bitwise-AND。因此，以下的呼叫為一個可能的程式建構方式：

1. `append_move(1, 0)`，添加一個指令將 $r[0]$ 複製至 $r[1]$。
2. `append_right(1, 1, 1)`，添加一個指令將 $r[1]$ 所有位元右移 $1$ 位，然後將結果存回 $r[1]$。因為每個整數都為 $1$ 位元，這使得 $r[1][0]$ 等於 $a[1]$。
3. `append_and(0, 0, 1)`，添加一個指令對 $r[0]$ 和 $r[1]$ 進行 bitwise-AND 運算，然後將結果存回 $r[0]$。在此指令被執行後， $r[0][0]$會被設為 $r[0][0]$ 與 $r[1][0]$ bitwise-AND 之結果，即為 $a[0]$ 與 $a[1]$ bitwise-AND，如所求。

## 範例 2

假設 $s = 1$、 $n = 2$、 $k = 1$、 $q = 1000$。如同前面的範例，此程式僅有 4 種可能的輸入。 對這 4 個 cases， $\min(a[0], a[1])$ 為 $a[0]$ 與 $a[1]$ bitwise-AND 的結果，而 $\max(a[0], a[1])$ 為 $a[0]$ 與 $a[1]$ bitwise-OR 的結果。以下的呼叫為一個可能的程式建構方式：

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)`

在執行完這些指令後， $c[0] = r[0][0]$ 包含 $\min(a[0], a[1])$，且 $c[1] = r[0][1]$ 包含 $\max(a[0], a[1])$，即將輸入排序完成後的結果。

## 條件限制

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

## 子任務

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

## 範例評分程式

此範例評分程式以下列格式讀取輸入：

- line $1$ : $s\ n\ k\ q$

接下來有若干行，每行為一筆測試資料。 每筆測試資料格式如下：

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

表示測試資料中的輸入為 $n$ 個整數 $a[0], a[1], ..., a[n-1]$。 所有的測試資料後會緊接著一行，為單獨一個 $-1$。

此範例評分程式首先呼叫 `construct_instructions(s, n, k, q)`。若此呼叫違反某些問題敘述中的限制，此範例評分程式會輸出 "實作細節" 一節末所列的錯誤訊息，而後終止 (exit)。否則，此範例評分程式先依序印出 `construct_instructions(s, n, k, q)` 添加的指令。對 $store$ 指令， $v$ 由註標 $0$ 至註標 $b - 1$ 印出。

接著，此範例評分程式依序處理測試資料。對每筆測試資料，被建構的程式會以此測試資料的內容為輸入被執行。

對每個 $print(t)$ 操作，令 $d[0], d[1], \ldots, d[n-1]$ 為一整數序列，使得對每個 $i$ ($0 \le i \le n - 1$)， $d[i]$ 為暫存器 $t$ 中位元序列 $i \cdot k$ 至 $(i + 1) \cdot k - 1$ 的整數值 (當此操作被執行時)。 此評分程式以下列格式輸出此序列：`register` $t$: $d[0]\ d[1]\ \ldots\ d[n-1]$。

一旦所有指令皆被執行，此範例評分程式印出此程式的輸出。

若 $s = 0$，此範例評分程式對於每筆測式資料的輸出格式如下：

- $c[0]$.

若 $s = 1$， 此範例評分程式對於每筆測式資料的輸出格式如下：

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

執行完所有測試資料後，此評分程式印出 `number of instructions:` $X$，其中 $X$ 為你的程式中的指令數。