

Registros de desplazamiento de bits

El ingeniero Christopher está trabajando en un nuevo tipo de procesador de ordenador.

El procesador tiene acceso a m diferentes celdas de memoria de b bits (donde $m = 100$ y $b = 2000$), que se llaman **registros**, y están numerados de 0 a $m - 1$. Denotamos los registros por $r[0], r[1], \dots, r[m - 1]$. Cada registro es una matriz de b bits, numerados de 0 (el bit más a la derecha) a $b - 1$ (el bit más a la izquierda). Para cada i ($0 \leq i \leq m - 1$) y cada j ($0 \leq j \leq b - 1$), denotamos el j -ésimo bit del registro i por $r[i][j]$.

Para cualquier secuencia de bits d_0, d_1, \dots, d_{l-1} (de longitud arbitraria l), el **valor entero** de la secuencia es igual a $2^0 \cdot d_0 + 2^1 \cdot d_1 + \dots + 2^{l-1} \cdot d_{l-1}$. Decimos que el **valor entero almacenado en un registro** i es el valor entero de la secuencia de sus bits, es decir, es $2^0 \cdot r[i][0] + 2^1 \cdot r[i][1] + \dots + 2^{b-1} \cdot r[i][b - 1]$.

El procesador tiene 9 tipos de **instrucciones** que pueden utilizarse para modificar los bits de los registros. Cada instrucción opera sobre uno o más registros y almacena la salida en uno de los registros. En lo que sigue, utilizamos $x := y$ para denotar una operación de cambio del valor de x de forma que se haga igual a y . Las operaciones realizadas por cada tipo de instrucción se describen a continuación.

- $move(t, y)$: Copia la matriz de bits del registro y al registro t . Para cada j ($0 \leq j \leq b - 1$), establece $r[t][j] := r[y][j]$.
- $store(t, v)$: Establecer el registro t para que sea igual a v , donde v es una matriz de b bits. Para cada j ($0 \leq j \leq b - 1$), establecer $r[t][j] := v[j]$.
- $and(t, x, y)$: Tomar la suma de los registros x e y , y almacenar el resultado en el registro t . Para cada j ($0 \leq j \leq b - 1$), se establece $r[t][j] := 1$ si **tanto** $r[x][j]$ como $r[y][j]$ son 1, y se establece $r[t][j] := 0$ en caso contrario.
- $or(t, x, y)$: Toma el bitwise-OR de los registros x e y , y almacena el resultado en el registro t . Para cada j ($0 \leq j \leq b - 1$), establecer $r[t][j] := 1$ si **al menos uno** de $r[x][j]$ y $r[y][j]$ son 1, y establecer $r[t][j] := 0$ en caso contrario.
- $xor(t, x, y)$: Tomar el bitwise-XOR de los registros x e y , y almacenar el resultado en el registro t . Para cada j ($0 \leq j \leq b - 1$), establecer $r[t][j] := 1$ si **exactamente uno** de $r[x][j]$ y $r[y][j]$ es 1, y establecer $r[t][j] := 0$ en caso contrario.
- $not(t, x)$: Tomar el bitwise-NOT del registro x , y almacenar el resultado en el registro t . Para cada j ($0 \leq j \leq b - 1$), establecer $r[t][j] := 1 - r[x][j]$.
- $left(t, x, p)$: Desplazar todos los bits del registro x hacia la izquierda en p , y almacenar el resultado en el registro t . El resultado de desplazar los bits del registro x hacia la izquierda en

p es una matriz v que consta de b bits. Para cada j ($0 \leq j \leq b-1$), $v[j] = r[x][j-p]$ si $j \geq p$, y $v[j] = 0$ en caso contrario. Para cada j ($0 \leq j \leq b-1$), establecer $r[t][j] := v[j]$.

- $right(t, x, p)$: Desplazar todos los bits del registro x hacia la derecha en p , y almacenar el resultado en el registro t . El resultado de desplazar los bits del registro x a la derecha en p es una matriz v que consta de b bits. Para cada j ($0 \leq j \leq b-1$), $v[j] = r[x][j+p]$ si $j \leq b-1-p$, y $v[j] = 0$ en caso contrario. Para cada j ($0 \leq j \leq b-1$), establecer $r[t][j] := v[j]$.
- $add(t, x, y)$: Suma los valores enteros almacenados en el registro x y el registro y , y almacena el resultado en el registro t . La suma se realiza en módulo 2^b . Formalmente, sea X el valor entero almacenado en el registro x , y Y el valor entero almacenado en el registro y antes de la operación. Sea T el valor entero almacenado en el registro t después de la operación. Si $X + Y < 2^b$, se fijan los bits de t , de forma que $T = X + Y$. En caso contrario, fijar los bits de t , de forma que $T = X + Y - 2^b$.

Christopher quiere que resuelvas dos tipos de tareas utilizando el nuevo procesador. El tipo de tarea se indica con un número entero s . Para ambos tipos de tareas, tienes que producir un **programa**, es decir, una secuencia de instrucciones definida anteriormente.

La **entrada** al programa consiste en n enteros $a[0], a[1], \dots, a[n-1]$, cada uno con k -bits, es decir, $a[i] < 2^k$ ($0 \leq i \leq n-1$).

Antes de ejecutar el programa, todos los números de entrada se almacenan secuencialmente en el registro 0, de manera que para cada i ($0 \leq i \leq n-1$) el valor entero de la secuencia de k bits $r[0][i \cdot k], r[0][i \cdot k + 1], \dots, r[0][(i+1) \cdot k - 1]$ es igual a $a[i]$. Tenga en cuenta que $n \cdot k \leq b$.

Todos los demás bits del registro 0 (es decir, los que tienen índices entre $n \cdot k$ y $b-1$, inclusive) y todos los bits de los demás registros se inicializan a 0.

La ejecución de un programa consiste en ejecutar sus instrucciones en orden. Una vez ejecutada la última instrucción, la **salida** del programa se calcula en base al valor final de los bits del registro 0. En concreto, la salida es una secuencia de n enteros $c[0], c[1], \dots, c[n-1]$, donde para cada i ($0 \leq i \leq n-1$), $c[i]$ es el valor entero de una secuencia formada por los bits $i \cdot k$ a $(i+1) \cdot k - 1$ del registro 0. Tenga en cuenta que después de ejecutar el programa los bits restantes del registro 0 (con índices al menos $n \cdot k$) y todos los bits de todos los demás registros pueden ser arbitrarios.

- La primera tarea ($s = 0$) es encontrar el menor número entero entre los enteros de entrada $a[0], a[1], \dots, a[n-1]$. En concreto, $c[0]$ debe ser el mínimo de $a[0], a[1], \dots, a[n-1]$. Los valores de $c[1], c[2], \dots, c[n-1]$ pueden ser arbitrarios.
- La segunda tarea ($s = 1$) es ordenar los enteros de entrada $a[0], a[1], \dots, a[n-1]$ en orden no decreciente. Específicamente, para cada i ($0 \leq i \leq n-1$), $c[i]$ debe ser igual al i -ésimo menor número entero entre $a[0], a[1], \dots, a[n-1]$ (es decir, $c[0]$ es el menor número entero entre los enteros de entrada).

Proporcione a Christopher con programas, que consten como máximo de q instrucciones cada uno, que puedan resolver estas tareas.

Detalles de la implementación

Usted deberá implementar los siguientes procedimientos:

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

- s : tipo de tarea.
- n : número de enteros en la entrada.
- k : número de bits en cada entero de entrada.
- q : número máximo de instrucciones permitidas.
- Este procedimiento se llama exactamente una vez y debe construir una secuencia de instrucciones para realizar la tarea requerida.

Este procedimiento debe llamar a uno o más de los siguientes procedimientos para construir una secuencia de instrucciones:

```
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)
```

- Cada procedimiento añade una instrucción $move(t, y)$, $store(t, v)$, $and(t, x, y)$, $or(t, x, y)$, $xor(t, x, y)$, $not(t, x)$, $left(t, x, p)$, $right(t, x, p)$ o $add(t, x, y)$ al programa, respectivamente.
- Para todas las instrucciones relevantes, t , x , y deben ser como mínimo 0 y como máximo $m - 1$.
- Para todas las instrucciones relevantes, t , x , y no son necesariamente distintas por pares.
- Para las instrucciones $left$ y $right$, p debe ser al menos 0 y como máximo b .
- Para las instrucciones $store$, la longitud de v debe ser b .

También puede llamar al siguiente procedimiento para ayudarle a probar su solución:

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

- Cualquier llamada a este procedimiento será ignorada durante la calificación de su solución.
- En el calificador de ejemplo, este procedimiento añade una operación $print(t)$ al programa.
- Cuando el calificador de ejemplo encuentra una operación $print(t)$ durante la ejecución de un programa, imprime n enteros de k bits formados por los primeros $n \cdot k$ bits del registro t (vea

la sección "Calificador de ejemplo" para más detalles).

- t debe satisfacer $0 \leq t \leq m - 1$.
- Cualquier llamada a este procedimiento no se suma al número de instrucciones construidas.

Después de añadir la última instrucción, `construct_instructions` debe retornar. El programa se evalúa entonces en un cierto número de casos de prueba, cada uno especificando una entrada que consiste en n k bits enteros $a[0], a[1], \dots, a[n - 1]$. Su solución pasa un caso de prueba dado si la salida del programa $c[0], c[1], \dots, c[n - 1]$ para la entrada proporcionada satisface las siguientes condiciones:

- Si $s = 0$, $c[0]$ debe ser el valor más pequeño entre $a[0], a[1], \dots, a[n - 1]$.
- Si $s = 1$, para cada i ($0 \leq i \leq n - 1$), $c[i]$ debe ser el $1 + i$ -ésimo número entero más pequeño entre $a[0], a[1], \dots, a[n - 1]$.

La calificación de su solución puede dar lugar a uno de los siguientes mensajes de error:

- `Invalid index`: se proporcionó un índice de registro incorrecto (posiblemente negativo) como parámetro t , x o y para alguna llamada de uno de los procedimientos.
- `Value to store is not b bits long`: la longitud de v dado a `append_store` no es igual a b .
- `Invalid shift value`: el valor de p dado a `append_left` o `append_right` no está entre 0 y b inclusivo.
- `Too many instructions`: su procedimiento intentó añadir más de q instrucciones.

Ejemplos

Ejemplo 1

Supongamos que $s = 0$, $n = 2$, $k = 1$, $q = 1000$. Hay dos enteros de entrada $a[0]$ y $a[1]$, cada uno con $k = 1$ bits. Antes de ejecutar el programa, $r[0][0] = a[0]$ y $r[0][1] = a[1]$. Todos los demás bits del procesador se ponen a 0 . Después de ejecutar todas las instrucciones del programa, necesitamos tener $c[0] = r[0][0] = \min(a[0], a[1])$, que es el mínimo de $a[0]$ y $a[1]$.

Sólo hay 4 entradas posibles para el programa:

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

Nosotros podemos notar que para los 4 casos, $\min(a[0], a[1])$ es igual al bitwise-AND de $a[0]$ y $a[1]$. Por lo tanto, una posible solución es construir un programa haciendo las siguientes llamadas:

1. `append_move(1, 0)`, el cual añade una instrucción para copiar $r[0]$ a $r[1]$.
2. `append_right(1, 1, 1)`, el cual añade una instrucción que toma todos los bits de $r[1]$, los desplaza a la derecha en 1 bit, y luego almacena el resultado de nuevo en $r[1]$. Como cada entero tiene una longitud de 1 bits, esto hace que $r[1][0]$ sea igual a $a[1]$.

3. `append_and(0, 0, 1)`, el cual añade una instrucción para tomar el bitwise-AND de $r[0]$ y $r[1]$, y almacenar el resultado en $r[0]$. Después de que se ejecute esta instrucción, $r[0][0]$ se establece como el bitwise-AND de $r[0][0]$ y $r[1][0]$, que es igual al bitwise-AND de $a[0]$ y $a[1]$, como se desea.

Ejemplo 2

Supongamos que $s = 1$, $n = 2$, $k = 1$, $q = 1000$. Como en el ejemplo anterior, sólo hay 4 entradas posibles para el programa. Para los 4 casos, $\min(a[0], a[1])$ es el bitwise-AND de $a[0]$ y $a[1]$, y $\max(a[0], a[1])$ es el bitwise-OR de $a[0]$ y $a[1]$. Una posible solución es hacer las siguientes llamadas:

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)`

Después de ejecutar estas instrucciones, $c[0] = r[0][0]$ contiene $\min(a[0], a[1])$, y $c[1] = r[0][1]$ contiene $\max(a[0], a[1])$, que ordena la entrada.

Restricciones

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

Subtareas

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

Ejemplo del calificador

El calificador de ejemplo lee la entrada en el siguiente formato:

- línea 1 : $s \ n \ k \ q$

A esto le sigue un cierto número de líneas, cada una de las cuales describe un único caso de prueba. Cada caso de prueba se proporciona en el siguiente formato:

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

y describe un caso de prueba cuya entrada consiste en n enteros $a[0], a[1], \dots, a[n - 1]$. La descripción de todos los casos de prueba va seguida de una sola línea que contiene únicamente -1 .

El calificador de ejemplo primero llama a `construct_instructions(s, n, k, q)`. Si esta llamada viola alguna restricción descrita en el enunciado del problema, el calificador de ejemplo imprime uno de los mensajes de error enumerados al final de la sección "Detalles de la implementación" y sale. De lo contrario, el calificador de ejemplo imprime primero cada instrucción anexada por `construct_instructions(s, n, k, q)` en orden. Para las instrucciones *store*, *v* se imprime desde el índice 0 hasta el índice $b - 1$.

A continuación, el calificador de ejemplo procesa los casos de prueba en orden. Para cada caso de prueba, ejecuta el programa construido en la entrada del caso de prueba.

Para cada operación `print(t)`, sea $d[0], d[1], \dots, d[n - 1]$ una secuencia de enteros, tal que para cada i ($0 \leq i \leq n - 1$), $d[i]$ es el valor entero de la secuencia de bits $i \cdot k$ a $(i + 1) \cdot k - 1$ del registro t (cuando se ejecuta la operación). El calificador imprime esta secuencia en el siguiente formato `register t : $d[0] \ d[1] \ \dots \ d[n - 1]$` .

Una vez que se han ejecutado todas las instrucciones, el calificador de ejemplo imprime la salida del programa.

Si $s = 0$, la salida del calificador de ejemplos para cada caso de prueba tiene el siguiente formato:

- $c[0]$.

Si $s = 1$, la salida del calificador de ejemplo para cada caso de prueba tiene el siguiente formato:

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

Después de ejecutar todos los casos de prueba, el calificador imprime `número de instrucciones: X` donde X es el número de instrucciones de su programa.