Registros de desplazamiento de bits

Christopher, el ingeniero, está trabajando en un nuevo tipo de procesador de computadora.

El procesador tiene acceso a m diferentes celdas de memoria de b-bit (donde $m = 100$ y $b = 2000$), que se denominan **registros** y están numeradas desde 0 hasta $m - 1$. Denotamos los registros por $r[0], r[1], \ldots, r[m-1]$. Cada registro es una matriz de b bits, numerados desde 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 $d_0, d_1, \ldots, d_{l-1}$ (de longitud arbitraria l), el **valor entero** de la secuencia es igual a $2^0 \cdot d_0 + 2^1 \cdot d_1 + \ldots + 2^{l-1} \cdot d_{l-1}$.

Decimos que el **valor entero almacenado en un registro** i es el valor entero de la secuencia de sus bits, es decir, es $2^0 \cdot r[i][0] + 2^1 \cdot r[i][1] + \ldots + 2^{b-1} \cdot r[i][b-1]$.

El procesador tiene 9 tipos de **instrucciones** que se pueden usar para modificar los bits en los registros. Cada instrucción opera en uno o más registros y almacena la salida en uno de los registros. A continuación, usamos $x := y$ para denotar una operación de cambiar el valor de x de modo que sea igual a y. Las operaciones realizadas por cada tipo de instrucción se describen a continuación.

- $move(t, y)$: copia la matriz de bits en el registro y para registrar t. Para cada j $(0\ leq j\ leq b - 1)$, establezca $r[t][j] := r[y][j]$.
- $store(t, v)$: Establezca el registro t para que sea igual a v, donde v es una matriz de b bits. Para cada j $(0 \leq j \leq b - 1)$, establezca $r[t][j] := v[j]$.
- $and(t, x, y)$: tome el AND bit a bit de los registros x y y, y almacene el resultado en el registro t. Para cada j $(0 \leq j \leq b - 1)$, establezca $r[t][j] := 1$ si **tanto** $r[x][j]$ como $r[y][j]$ son 1, y establecer $r[t][j] := 0$ en caso contrario.
- $or(t, x, y)$: tome el OR bit a bit de los registros x y y, y almacene el resultado en el registro t. Para cada j $(0 \leq j \leq b - 1)$, establezca $r[t][j] := 1$ si **al menos uno** de $r[x][j]$ y $r[y][j]$ son 1, y establecer $r[t][j] := 0$ en caso contrario.
- $xor(t, x, y)$: Tome el XOR bit a bit de los registros x y y, y almacene el resultado en el registro t. Para cada j $(0 \leq j \leq b - 1)$, establezca $r[t][j] := 1$ si **exactamente uno** de $r[x][j]$ y $r[y][j]$es 1, y establecer $r[t][j] := 0$ en caso contrario.
- $not(t, x)$: Tome el bit a bit-NOT del registro x y almacene el resultado en el registro t. Para cada j $(0 \leq j \leq b - 1)$, establezca $r[t][j] := 1 - r[x][j]$.
- $left(t, x, p)$: Desplaza todos los bits del registro x hacia la izquierda en p y almacena el resultado en el registro t. El resultado de desplazar los bits del registro x a la izquierda p es

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

- $right(t, x, p)$: Desplaza todos los bits del registro x hacia la derecha en p y almacena el resultado en el registro t. El resultado de desplazar los bits del registro x a la derecha 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 $v[j] = 0$ de lo contrario. Para cada j $(0 \leq j \leq b-1)$, establezca $r[t][j] := v[j]$.

- $add(t, x, y)$: 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 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 $X + Y < 2^b$, establezca los bits de t, de modo que $T = X + Y$. De lo contrario, establezca los bits de t, de modo que $T = X + Y - 2^b$.

A Christopher le gustaría que resolviera dos tipos de tareas utilizando el nuevo procesador. El tipo de tarea se indica con un número entero s. Para ambos tipos de tareas, debe producir un **programa** , que es una secuencia de instrucciones definidas anteriormente.

La **entrada** al programa consta de n enteros $a[0], a[1], \ldots, a[n-1]$, cada uno con k -bits, es decir, $a[i] < 2^k$ ($0 \leq i \leq n-1$). Antes de que se ejecute el programa, todos los números de entrada se almacenan secuencialmente en el registro 0, de modo que para cada i $(0 \leq i \leq n-1)$ el valor entero de la secuencia de k bits $r[0][i \cdot k], r[0][i \cdot k + 1], \ldots, r[0][(i+1) \cdot k - 1]$ es igual a $a[i]$. Tenga en cuenta que $n \cdot k \leq b$. Todos los demás bits del registro 0 (es decir, aquellos con índices entre $n \cdot k$ y $b-1$, inclusive) y todos los bits de todos los demás registros se inicializan en 0.

Ejecutar un programa consiste en ejecutar sus instrucciones en orden. Después de que se ejecuta la última instrucción, la **salida** del programa se calcula basándose en el valor final de los bits en el registro 0. Específicamente, la salida es una secuencia de n enteros $c[0], c[1], \ldots, c[n-1]$, donde para cada i ($0 \leq i \leq n-1$), $c[i]$ es el valor entero de una secuencia que consta de los bits $i \cdot k$ a $(i+1) \cdot k - 1$ del registro 0. Ten en cuenta que después de ejecutar el programa, los bits restantes del registro 0 (con índices de 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 número entero más pequeño entre los enteros de entrada $a[0], a[1], \ldots, a[n-1]$. Específicamente, $c[0]$ debe ser el mínimo de $a[0], a[1], \ldots, a[n-1]$. Los valores de $c[1], c[2], \ldots, c[n-1]$ pueden ser arbitrarios.

- La segunda tarea $(s = 1)$ es ordenar los enteros de entrada $a[0], a[1], \ldots, a[n-1]$ en orden no decreciente. Específicamente, para cada i ($0 \leq i \leq n-1$), $c[i]$ debe ser igual al $1+i$-ésimo entero más pequeño entre $a[0], a[1], \ldots, a[n-1]$ (es decir, $c[0]$ es el número entero más pequeño entre los enteros de entrada).

Proporcione a Christopher programas, que constan de un máximo de q instrucciones cada uno, que puedan resolver estas tareas.

Detalles de Implementación

Debes implementar el siguiente procedimiento:

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

- s: tipo de tarea.
- n: numero de enteros en la entrada.
- k: numero de bits en cada entero de la entrada.
- q: maximo numero 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 agrega $mover(t, y)$ $tienda(t, v)$, $y(t, x, y)$, $o(t, x, y)$, $xor(t, x, y)$, $not(t, x)$, $left(t, x, p)$, $right(t, x, p)$ o $add(t, x, y)$ instrucción al programa, respectivamente.
- Para todas las instrucciones relevantes, t, x, y deben ser al menos 0 y como máximo $m - 1$.
- Para todas las instrucciones relevantes, t, x, y no son necesariamente distintos por pares.
- Para las instrucciones $left$ y $right$, p debe ser al menos 0 y como máximo b.
- Para las instrucciones de $store$, la longitud de v debe ser b.

También puedes llamar al siguiente procedimiento para que te ayude a probar tu solución:

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

- Cualquier llamada a este procedimiento será ignorada durante la calificación de su solución.
- Cuando el evaluador de muestra encuentra una operación $print(t)$ durante la ejecución de un programa, imprime números enteros de nk bits formados por los primeros n·k bits del registro t (consulte la sección "Evaluador de muestra" para obtener más detalles).
- En el evaluador de ejemplo este procedimiento agrega una operación $print(t)$ al programa. En la ejecución de un programa, imprime enteros de n k-bit formados por los primeros $n cdotk$ bits de registre t (consulte la sección "Evaluador de ejemplo" para obtener 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 agregar la última instrucción, `construct_instructions` debería regresar return.

A continuación, el programa se evalúa en algunos casos de prueba, cada uno de los cuales especifica una entrada que consta de n k-bits enteros $a[0], a[1], \ldots, a[n-1]$. Su solución pasa un caso de prueba dado si la salida del programa $c[0], c[1], \ldots, c[n-1]$ para la entrada proporcionada satisface las siguientes condiciones:

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

El evaluador de tu solución puede resultar en 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 dada en `_store` no es igual a b.
- `Invalid shift value`: el valor de p dado en `_left` o `_right` no está entre 0 y b inclusive.
- `Too many instructions`: tu procedimiento agregar más de q instrucciones.

Ejemplos

Ejemplo 1

Supongase que $s = 0$, $n = 2$, $k = 1$, $q = 1000$. Hay dos enteros de entrada $a[0]$ y $a[1]$, cada uno tiene $k = 1$ bit. Antes de ejecutar el programa, $r[0][0] = a[0]$ y $r[0][1] = a[1]$. Todos los otros bits en el procesador estan en 0. despues que todas las instrucciones del programa son ejecutadas, necesitamos que tenga $c[0] = r[0][0] = \min(a[0], a[1])$, que es el mínimo de $a[0]$ y $a[1]$.

Solo 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: $a[0] = 1, a[1] = 1$

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

1. `_move(1, 0)`, que agrega una instrucción para copiar $r[0]$ a $r[1]$.
2. `_right(1, 1, 1)`, que agrega una instrucción que toma todos los bits en $r[1]$, Los mueve a la derecha por 1 bit, y luego almacena el resultado en $r[1]$. Dado que cada entero es 1-bit de largo, esto da como resultado que $r[1][0]$ sea igual que $a[1]$.
3. `_and(0, 0, 1)`, que agrega una instrucción para tomar el bit a bit-AND de $r[0]$ y $r[1]$, luego guarda el resultado en $r[0]$. Después de que se ejecuta esta instrucción, $r[0][0]$ se establece

en el bit a bit-AND de $r[0][0]$ y $r[1][0]$, que es igual al AND bit a bit de $a[0]$ y $a[1]$, como se desea.

Ejemplo 2

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

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)`

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 de evaluador

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

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

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

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

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

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

Luego, el evaluador 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.

Por cada operación $print(t)$, deja $d[0], d[1], \ldots, d[n-1]$ una secuencia de enteros, tal que para cada i ($0 \le i \le n-1$), $d[i]$ es el valor entero de la secuencia de bits desde $i \cdot k$ hasta $(i+1) \cdot k - 1$ del registro t (cuando se ejecuta la operación). El evaluador imprime esta secuencia en el siguiente formato: `register` t: $d[0]\ d[1]\ \ldots\ d[n-1]$.

Una vez que se han ejecutado todas las instrucciones, el evaluador de ejemplo imprime el resultado del programa.

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

- $c[0]$.

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

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

Después de ejecutar todos los casos de prueba, el evaluador imprime `number of instructions:` X donde X es el numero de instrucciones de tu programa.