

Unscrambling a Messy Bug

Ilshat es un ingeniero de software trabajando en estructuras de datos eficientes. Un día él inventó una nueva estructura de datos. Esta estructura puede almacenar un conjunto de enteros *no negativos* de n bits. Donde n es una potencia de dos. Esto es, $n = 2^b$ para algún entero no negativo b .

La estructura de datos está inicialmente vacía. Un programa que usa la estructura de datos tiene que seguir las siguientes reglas:

- El programa puede añadir elementos que son enteros de n bits a la estructura de datos, uno a la vez, usando la función `add_element(x)`. Si el programa trata de añadir un elemento que ya está presente en la estructura de datos, no pasa nada.
- Después de añadir el último elemento el programa debe llamar la función `compile_set()` exactamente una vez.
- Finalmente, el programa puede llamar a la función `check_element(x)` para verificar si el elemento x está presente en la estructura de datos. Esta función puede ser usada varias veces.

Cuando Ilshat implementó por primera vez esta estructura de datos, el cometió un error en la función `compile_set()`. El error reordena los dígitos binarios de cada elemento en el conjunto de la misma manera. Ilshat quiere que usted encuentre el reordenamiento exacto de dígitos causado por el error.

Formalmente, considere una secuencia $p = [p_0, \dots, p_{n-1}]$ en la cual cada número desde 0 a $n - 1$ aparece exactamente una vez. Llamamos a tal secuencia una *permutación*. Considere un elemento del conjunto, cuyos dígitos en binario son a_0, \dots, a_{n-1} (con a_0 siendo el dígito más significativo). Cuando se llama a la función `compile_set()`, este elemento es reemplazado por el elemento $a_{p_0}, a_{p_1}, \dots, a_{p_{n-1}}$.

La misma permutación p es usada para reordenar los dígitos de cada elemento almacenado en la estructura de datos. La permutación puede ser arbitraria, incluyendo la posibilidad que $p_i = i$ para cada $0 \leq i \leq n - 1$.

Por ejemplo, suponga que $n = 4$, $p = [2, 1, 3, 0]$, y que usted ha insertado en el conjunto enteros cuya representación binaria es `0000`, `1100` y `0111`. Llamar la función `compile_set` cambia esos elementos a `0000`, `0101` y `1110`, respectivamente.

Su tarea es escribir un programa que encuentre la permutación p interactuando con la estructura de datos. Debería (en el siguiente orden):

1. elegir un conjunto de enteros de n bits,
2. insertar esos enteros en la estructura de datos,

3. llamar a la función `compile_set` para disparar el error,
4. verificar la presencia de algunos elementos en el conjunto modificado,
5. usar esa información para determinar y # Decifrando un Error Enredado

Ilshat es un ingeniero de software trabajando en estructuras de datos eficientes. Un día él inventó una nueva estructura de datos. Esta estructura puede almacenar un conjunto de enteros *no negativos* de n bits. Donde n es una potencia de dos. Esto es, $n = 2^b$ para algún entero no negativo b .

La estructura de datos está inicialmente vacía. Un programa usando la estructura de datos tiene que seguir las siguientes reglas:

- El programa puede añadir elementos que son enteros de n bits a la estructura de datos, a uno a la vez, usando la función `add_element(x)`. Si el programa trata de añadir un elemento que ya está presente en la estructura de datos, no pasa nada.
- Después de añadir el último elemento el programa de llamar la función `compile_set()` exactamente una vez.
- Finalmente, el programa puede llamar la función `check_element(x)` para verificar si el elemento x está presente en la estructura de datos. Esta función puede ser usada varias veces.

Cuando Ilshat implementó por primera vez esta estructura de datos, el cometió un error en la función `compile_set()`. El error reordena los dígitos binarios de cada elemento en el conjunto de la misma manera. Ilshat quiere que usted encuentre el reordenamiento exacto de dígitos causado por el error.

Formalmente, considere una secuencia $p = [p_0, \dots, p_{n-1}]$ en la cual cada número desde 0 a $n - 1$ aparece exactamente una vez. Llamamos a tal secuencia una *permutación*. Considere un elemento del conjunto, cuyos dígitos en binario son a_0, \dots, a_{n-1} (con a_0 siendo el dígitos mas significativo). Cuando se llama a la función `compile_set()`, este elemento es reemplazado por el elemento $a_{p_0}, a_{p_1}, \dots, a_{p_{n-1}}$.

La misma permutación p es usada para reordenar los dígitos de cada elemento almacenado en la estructura de datos. Cualquier permutación p es posible, incluyendo la posibilidad que $p_i = i$ para cada $0 \leq i \leq n - 1$.

Por ejemplo, suponga que $n = 4$, $p = [2, 1, 3, 0]$, y que usted ha insertado en el conjunto enteros cuya representación binaria es `0000`, `1100` y `0111`. Llamar la función `compile_set` cambia esos elementos a `0000`, `0101` y `1110`, respectivamente.

Su tarea es escribir un programa que encuentre la permutación p interactuando con la estructura de datos. Debería (en el siguiente orden):

1. elegir un conjunto de enteros de n bits,
2. insertar esos enteros en la estructura de datos,
3. llamar la función `compile_set` para disparar el error,
4. verificar la presencia de algunos elementos en el archivo modificado,
5. usar esa información para determinar y retornar la permutación p .

Note que su programa puede llamar la función `compile_set` únicamente una vez.

Adicionalmente, hay un límite al número de veces que su programa puede llamar a las funciones de librería. A saber, puede:

- llamar `add_element` a lo más w veces (w es de "writes"),
- llamar `check_element` a lo más r veces (r es de "reads").

Detalles de implementación

Usted debe implementar una función (método):

- `int[] restore_permutation(int n, int w, int r)`
 - n : el número de bits en la representación binaria de cada elemento en el conjunto (y también la longitud de p).
 - w : el número máximo de operaciones `add_element` que su programa puede ejecutar.
 - r : el número máximo de operaciones `check_element` que su programa puede ejecutar.
 - la función debe retornar la permutación p restaurada.

En el lenguaje C, el prototipo de la función es un poco diferente:

- `void restore_permutation(int n, int w, int r, int* result)`
 - n, w y r tienen el mismo significado que antes.

* la función debe retornar la permutación `\\(p\\)` restaurada

Funciones de librería

Con el propósito de interactuar con la estructura de datos, su programa debe usar las siguientes tres funciones (métodos):

- `void add_element(string x)`

Esta función añade el elemento descrito por x al conjunto.

 - x : es una cadena de '0' y '1' dando la representación binaria de un entero que debe ser añadido al conjunto. La longitud de x debe ser n .
- `void compile_set()`

Esta función debe ser llamada exactamente una vez. Su programa no puede llamar `add_element()` after this call. Su programa no puede llamar a `check_element()` antes de esta llamada.
- `boolean check_element(string x)`

Esta función verifica si el elemento x está en el conjunto modificado.

 - x : una cadena de '0' y '1' dando la representación binaria del elemento que debe ser verificado. La longitud de x debe ser n .
 - devuelve `true` si el elemento x está en el conjunto modificado, y `false` en otro caso.

Note que si su programa viola cualquiera de las instrucciones antes dadas, su calificación será "Wrong Answer".

Para todas las cadenas, el primer carácter da el bit más significativo del entero correspondiente.

El calificador fija la permutación p antes de que la función `restore_permutation` sea llamada.

Por favor use los archivos ejemplo proporcionados para detalles de implementación en su lenguaje de programación.

Ejemplo

El calificador hace el siguiente llamado de función:

- `restore_permutation(4, 16, 16)`. Tenemos $n = 4$ y el programa puede hacer a lo mas 16 "writes" y 16 "reads".

El programa hace los siguientes llamados de función:

- `add_element("0001")`
- `add_element("0011")`
- `add_element("0100")`
- `compile_set()`
- `check_element("0001")` devuelve `false`
- `check_element("0010")` devuelve `true`
- `check_element("0100")` devuelve `true`
- `check_element("1000")` devuelve `false`
- `check_element("0011")` devuelve `false`
- `check_element("0101")` devuelve `false`
- `check_element("1001")` devuelve `false`
- `check_element("0110")` devuelve `false`
- `check_element("1010")` devuelve `true`
- `check_element("1100")` devuelve `false`

Solamente una permutación es consistente con esos valores devueltos por `check_element()`: la permutación $p = [2, 1, 3, 0]$. Por lo tanto, `restore_permutation` debe retornar `[2, 1, 3, 0]`.

Subtareas

1. (20 puntos) $n = 8$, $w = 256$, $r = 256$, $p_i \neq i$ para a lo más 2 índices i ($0 \leq i \leq n - 1$),
2. (18 puntos) $n = 32$, $w = 320$, $r = 1024$,
3. (11 puntos) $n = 32$, $w = 1024$, $r = 320$,
4. (21 puntos) $n = 128$, $w = 1792$, $r = 1792$,
5. (30 puntos) $n = 128$, $w = 896$, $r = 896$.

Calificador ejemplo

El calificador ejemplo lee la entrada en el siguiente formato:

- línea 1: enteros n , w , r ,
- línea 2: n enteros dando los elementos de p .

la permutación p .

Note que su programa puede llamar la función `compile_set` únicamente una vez.

Adicionalmente, hay un límite en el número de veces que su programa llame a las funciones de librería. A saber, puede:

- llamar `add_element` a lo más w veces (w es de "writes"),
- llamar `check_element` a lo más r veces (r es de "reads").

Detalles de implementación

Usted debe implementar una función (método):

- `int[] restore_permutation(int n, int w, int r)`
 - n : el número de bits en la representación binaria de cada elemento en el conjunto (y también la longitud de p).
 - w : el número máximo de operaciones `add_element` que su programa puede ejecutar.
 - r : el número máximo de operaciones `check_element` que su programa puede ejecutar.
 - la función debe retornar la permutación p restaurada.

En el lenguaje C, el prototipo de la función es un poco diferente:

- `void restore_permutation(int n, int w, int r, int* result)`
 - n, w y r tienen el mismo significado que antes.

* la función debe retornar la permutación p restaurada

Funciones de librería

Con el propósito de interactuar con la estructura de datos, su programa debe usar las siguientes tres funciones (métodos):

- `void add_element(string x)`

Esta función añade el elemento descrito por x al conjunto.

 - x : es una cadena de '0' y '1' dando la representación binaria de un entero que debe ser añadido al conjunto. La longitud de x debe ser n .
- `void compile_set()`

Esta función debe ser llamada exactamente una vez. Su programa no puede llamar `add_element()` después de esta llamada. Su programa no puede llamar a `check_element()` antes de esta llamada.
- `boolean check_element(string x)`

Esta función verifica si el elemento x está en el conjunto modificado.

 - x : una cadena de '0' y '1' dando la representación binaria del elemento que debe ser verificado. La longitud de x debe ser n .
 - devuelve `true` si el elemento x está en el conjunto modificado, y `false` en otro caso.

Note que si su programa viola cualquiera de las instrucciones antes dadas, su calificación será "Wrong Answer".

Para todas las cadenas, el primer carácter da el bit mas significativo del entero correspondiente.

El calificador fija la permutación p antes de que la función

`restore_permutation` sea llamada.

Por favor use los archivos ejemplo proporcionados para detalles de implementación en su lenguaje de programación.

Ejemplo

El calificador hace el siguiente llamado de función:

- `restore_permutation(4, 16, 16)`. Tenemos $n = 4$ y el programa puede hacer a lo mas 16 "writes" y 16 "reads".

El programa hace los siguientes llamados de función:

- `add_element("0001")`
- `add_element("0011")`
- `add_element("0100")`
- `compile_set()`
- `check_element("0001")` devuelve `false`
- `check_element("0010")` devuelve `true`
- `check_element("0100")` devuelve `true`
- `check_element("1000")` devuelve `false`
- `check_element("0011")` devuelve `false`
- `check_element("0101")` devuelve `false`
- `check_element("1001")` devuelve `false`
- `check_element("0110")` devuelve `false`
- `check_element("1010")` devuelve `true`
- `check_element("1100")` devuelve `false`

Solamente una permutación es consistente con esos valores devueltos por `check_element()`: la permutación $p = [2, 1, 3, 0]$. Por lo tanto, `restore_permutation` debe retornar `[2, 1, 3, 0]`.

Subtareas

1. (20 puntos) $n = 8$, $w = 256$, $r = 256$, $p_i \neq i$ para lo más 2 índices i ($0 \leq i \leq n - 1$),
2. (18 puntos) $n = 32$, $w = 320$, $r = 1024$,
3. (11 puntos) $n = 32$, $w = 1024$, $r = 320$,
4. (21 puntos) $n = 128$, $w = 1792$, $r = 1792$,
5. (30 puntos) $n = 128$, $w = 896$, $r = 896$.

Ejemplo de "Grader"

El Grader de muestra lee las entradas con el siguiente formato:

- línea 1: enteros n , w , r ,
- línea 2: n enteros dando los elementos de p .