International Olympiad in Informatics 2016
12-19th August 2016
Kazan, Russia
day2_2

messy
Country: BRA

Corrigindo um erro confuso

O Ilshat é um engenheiro de *software* que trabalha em estruturas de dados eficientes. Um dia ele inventou uma nova estrutura de dados. Esta estrutura de dados consegue guardar um conjunto de inteiros *não-negativos* de n *bits*, onde n é uma potência de dois. Isto é, $n = 2^b$ para algum inteiro não negativo b.

A estrutura de dados está inicialmente vazia. O programa que usa a estrutura de dados tem de seguir as seguintes regras:

- O programa pode adicionar elementos que são inteiros de n *bits* à estrutura de dados, um de cada vez, usando a função `add_element(x)`. Se o programa tentar adicionar um elemento que já está presente na estrutura de dados, nada acontece.
- Depois de adicionar o último elemento, o programa deve chamar a função `compile_set()` exatamente uma vez.
- Finalmente, o programa pode chamar a função `check_element(x)` para verificar se o elemento x está presente na estrutura de dados. Esta função pode ser usada múltiplas vezes.

Quando o Ilshat implementou esta estrutura de dados pela primeira vez, ele tinha um erro na função `compile_set()`. O erro reordena da mesma forma os dígitos binários de cada elemento no conjunto. O Ilshat quer que encontres a reordenação de dígitos que o erro causou.

Formalmente, considere a sequência $p = [p_0, \ldots, p_{n-1}]$ onde cada número de 0 a $n-1$ aparece exatamente uma vez. Chamamos a uma tal sequência de *permutação*. Considere um elemento do conjunto cujos digitos em binário são $a_0, \ldots, a_{n-1}$ (sendo a_0 o *bit* mais significativo). Quando a função `compile_set()` é chamada, este elemento é substituido pelo elemento $a_{p_0}, a_{p_1}, \ldots, a_{p_{n-1}}$.

A mesma permutação p é usada para reordenar os dígitos de todos os elementos. Qualquer permutação é possível, incluindo a possibilidade de $p_i = i$ para cada $0 \leq i \leq n-1$.

Por exemplo, supondo que $n = 4$, $p = [2, 1, 3, 0]$, e foram inseridos no conjunto os inteiros cujas representações binárias são `0000`, `1100` e `0111`. Ao chamar a função `compile_set`, estes elementos são mudados para `0000`, `0101` e `1110`, respetivamente.

A sua tarefa é escrever um programa que encontra a permutação p ao interagir com a estrutura de dados. O programa deve (na ordem seguinte):

1. escolher um conjunto de inteiros de n *bits*,
2. inserir esses inteiros na estrutura de dados,

3. chamar a função `compile_set` para acionar o erro,
4. verificar a presença de alguns elementos no conjunto modificado,
5. usar essa informação para determinar e devolver a permutação p.

Note que o seu programa pode chamar a função `compile_set` apenas uma vez.

Adicionalmente, há um limite no número de vezes que o programa pode chamar as funções da biblioteca. Especificamente, o programa pode

- chamar `add_element` no máximo w vezes (w é para "writes" ("escritas")),
- chamar `check_element` no máximo r vezes (r é para "reads" ("leituras")).

Detalhes de implementação

Você deve implementar uma função (método):

- `int[] restore_permutation(int n, int w, int r)`
  - `n`: o número de *bits* na representação binária de cada elemento do conjunto (e também o comprimento de p).
  - `w`: o número máximo de operações `add_element` que o programa pode fazer.
  - `r`: o número máximo de operações `check_element` que o programa pode fazer.
  - a função deve retornar a permutação restaurada p.

Na linguagem C, o protótipo da função é um pouco diferente:

- `void restore_permutation(int n, int w, int r, int* result)`
  - `n`, `w` e `r` têm o mesmo significado que acima.
  - a função deve retornar a permutação restaurada p ao guardá-la no *array* providenciado `result`: para cada i, o programa deve guardar o valor de p_i em `result[i]`.

Funções da biblioteca

Para interagir com a estrutura de dados, o seu programa deve usar as seguintes três funções (métodos):

- `void add_element(string x)`
  Esta função adiciona um elemento descrito por `x` ao conjunto.
  - `x`: uma *string* de caracteres `'0'` e `'1'` que descrevem a representação binária do inteiro que deve ser adicionado ao conjunto. O comprimento de `x` tem de ser n.

- `void compile_set()`
  Esta função tem de ser chamada exatamente uma vez. O seu programa não pode chamar `add_element()` depois desta chamada. O seu programa não pode chamar `check_element()` antes desta chamada.
- `boolean check_element(string x)`
  Esta função verifica se o elemento `x` está no conjunto modificado.
  - `x`: uma *string* de caracteres `'0'` e `'1'` que descrevem a representação do elemento que deve ser verificado. O comprimento de `x` tem de ser n.
  - devolve `true` se o elemento `x` está no conjunto modificado e `false` caso contrário.

Note que se o seu programa violar alguma das restrições acima, o resultado da avaliação será "Wrong Answer".

Para todas as *strings*, o primeiro caracter corresponde ao *bit* mais significativo do inteiro correspondente.

O avaliador fixa a permutação p antes da função `restore_permutation` ser chamada.

Por favor use os arquivos modelo para obter detalhes de implementação na sua linguagem de programação.

Exemplo

O avaliador faz as seguintes chamadas de função:

- `restore_permutation(4, 16, 16)`. Temos que $n = 4$ e o programa pode fazer no máximo 16 "writes" ("escritas") e 16 "reads" ("leituras").

O programa faz as seguintes chamadas de função:

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

Apenas uma permutação é consistente com estes valores devolvidos por `check_element()`: a permutação $p = [2, 1, 3, 0]$. Assim, `restore_permutation` deve devolver `[2, 1, 3, 0]`.

Subtarefas

1. (20 pontos) $n = 8$, $w = 256$, $r = 256$, $p_i \neq i$ para no máximo 2 índices i ($0 \leq i \leq n - 1$),
2. (18 pontos) $n = 32$, $w = 320$, $r = 1024$,
3. (11 pontos) $n = 32$, $w = 1024$, $r = 320$,
4. (21 pontos) $n = 128$, $w = 1792$, $r = 1792$,
5. (30 pontos) $n = 128$, $w = 896$, $r = 896$.

Avaliador de exemplo

O avaliador de exemplo lê o *input* no seguinte formato:

- linha 1: inteiros n, w, r,

- linha 2: n inteiros representando os elementos de p.