反解混亂的錯誤 (Unscrambling a Messy Bug)

Ilshat 是一個軟體工程師，他主要在研發各種有效率的資料結構。有一天他發明了一種新的資料結構。這個資料結構可以儲存一組非負 n 位元的整數，其中 n 為二的次方。也就是 $n = 2^b$，其中b為某個非負整數。

這個資料結構一開始並無存放任何資料。當程式使用此資料結構時，必須遵守下列規則:

- 程式可使用函式`add_element(x)`加入若干元素(每個元素為 n 位元的整數)到資料結構中，加入的方式為一次一個。如果程式嘗試加入一個已經存在於資料結構的元素，則沒有作用。
- 在加入最後一個元素到資料結構之後，程式應呼叫函式`compile_set()`恰好一次。
- 最後，程式可以呼叫函式`check_element(x)`去檢**查**元素 x 是否已出現在資料結構中。這個函式可以被使用數次。

當Ilshat 開始實作這個資料結構時，他製造一個錯誤(bug)在函式 `compile_set()`。這個錯誤可用相同的方法重新排列集合裡面每個元素的二進位元。Ilshat 要你去找出此錯誤所造成的確切的重新排列為何。

正式地來**說**，考慮一組序列 $p = [p_0, \ldots, p_{n-1}]$，其中 0 到 $n-1$ 在此序列各只出現一次。我們稱此序列為一個排列 (*permutation*)。考慮集合中的一個元素，其二進位元為 $a_0, \ldots, a_{n-1}$ (以 a_0 為最左位元)。當函式 `compile_set()`被呼叫，這個元素將被替換成元素 $a_{p_0}, a_{p_1}, \ldots, a_{p_{n-1}}$.

此相同的排列 p 被使用來重新排列每個元素的位元。任意的排列都可能，包含 $p_i = i$，其中 $0 \le i \le n-1$。

例如，假設 $n = 4$， $p = [2, 1, 3, 0]$，並且你已經加入一組整數，其二進位表示分別為`0000`, `1100` 和 `0111`。呼叫函式`compile_set`將分別改變這些元素為 `0000`, `0101` 和 `1110`。

你的任務是寫一個程式藉由讀取和寫入跟資料結構互動，去找出排列 p。此程式的執行順序應為:

1. 選擇若干個 n 位元整數,
2. 插入這些整數到資料結構中,
3. 呼叫函式 `compile_set` 去觸發錯誤(trigger the bug),
4. 檢**查**某些元素，是否出現在修改過的集合中,
5. 使用獲得的資訊去判斷並回傳排列 p。

注意你的程式只能呼叫函式 `compile_set`一次。

此外，你的程式呼叫函式庫的函式有次數限制。程式可以

- 呼叫函式 `add_element` 至多 w 次 (w 是 "寫入"),
- 呼叫函式 `check_element` 至多 r 次 (r 是 "讀取")。

實作細節

你應實作一個函式 (方法):

- `int[] restore_permutation(int n, int w, int r)`
  - `n`: 在資料結構中的元素其二元表示法的位元個數 (亦即是 p 的長度)。
  - `w`: 你的程式可以執行 `add_element` 運算的最多次數。
  - `r`: 你的程式可以執行 `check_element` 運算的最多次數。
  - 函式應回傳還原的排列 p.

在 C 語言中，函式樣板(prototype)有一些不同:

- `void restore_permutation(int n, int w, int r, int* result)`
  - `n`, `w` 和 `r` 與前述的定義相同。
  - 藉由儲存到所提供的陣列`result`，函式應回傳還原的排列 p: 對於每個 i，函式應該儲存數**值** p_i 到 `result[i]`。

函式庫函式

為了藉由讀和寫與資料結構互動，你的程式應使用下列三個函式(方法):

- `void add_element(string x)`
  這個函式加入元素`x` 到資料結構中。
  - `x`: 代表要被加入的整數其二進位表示法。`x` 的長度必需是 n。

- `void compile_set()`
  這個函式必需被呼叫恰好一次。在此函式呼叫之後，你的程式不能呼叫函式 `add_element()`。 在此函式呼叫之前，你的程式不能再呼叫函式`check_element()`。
- `boolean check_element(string x)`
  這個函式去檢**查**是否元素`x`出現在修改的資料結構中。
  - `x`: 代表要被檢**查**的整數其二進位表示法。`x` 的長度必需是 n。
  - 假如元素 `x` 在修改的資料結構中則回傳 `true`，否則回傳 `false`。

注意:如果你的程式違反上述限制，你的評分結果將是"Wrong Answer"。

對於所有的字串，第一個字元代表所對應的整數的最左位元。

在實作細節時，請使用所提供對應程式語言的樣板檔案(template files)。

範例

評分程式呼叫下列函式:

- `restore_permutation(4, 16, 16)`。我們知 $n = 4$ 並且程式可執行不超過 16 次"寫入" 和 16 次"讀取"。

你的程式要執行下列的函式呼叫:

- `add_element("0001")`
- `add_element("0011")`
- `add_element("0100")`
- `compile_set()`
- `check_element("0001")` returns `false`
- `check_element("0010")` returns `true`
- `check_element("0100")` returns `true`
- `check_element("1000")` returns `false`
- `check_element("0011")` returns `false`
- `check_element("0101")` returns `false`

- `check_element("1001")` returns `false`
- `check_element("0110")` returns `false`
- `check_element("1010")` returns `true`
- `check_element("1100")` returns `false`

只有一個排列與函式`check_element()`回傳的**值**一致: 排列 $p = [2, 1, 3, 0]$. 因此，`restore_permutation` 應回傳 `[2, 1, 3, 0]`。

子任務

1. (20 points) $n = 8$, $w = 256$, $r = 256$, $p_i \neq i$ for at most 2 indices i ($0 \leq i \leq n - 1$),
2. (18 points) $n = 32$, $w = 320$, $r = 1024$,
3. (11 points) $n = 32$, $w = 1024$, $r = 320$,
4. (21 points) $n = 128$, $w = 1792$, $r = 1792$,
5. (30 points) $n = 128$, $w = 896$, $r = 896$.

範例評分程式

範例評分程式用下列格式讀取下列資料:

- line 1: 整數 n , w , r ,
- line 2: n 個integers 代表排列 p 。