找Bug

伊爾沙特是一位軟體工程師，他的工作是設計高效的資料結構。有一天，他發明了一個新的資料結構。這個資料結構可以存儲一個由 n 位元的非負整數所組成的集合， n 是 2 的整數次冪，即 $n = 2^b$ ， b 是非負整數。

這個資料結構初始時為空的。使用該資料結構的程式必須要遵守下列規則：

- 程式可以利用函數 `add_element(x)` 一次添加一個元素到這個資料結構中，每個元素都是一個 n 位元整數。如果程式要添加的元素已經在資料結構中，什麼事情也不會發生。
- 當添加完最後一個元素以後，程式應該調用一次 (而且僅一次) 函數 `complie_set( )`。
- 最後，程式可以調用函數 `check_element(x)`來檢**查**元素 x 是否在資料結構中。這個函數可以調用多次。

當伊爾沙特第一次實現該資料結構時，他在寫函數 `compile_set()` 時出現一個bug。這個bug將集合中每個元素的二進位位元以相同的方式重新排列。伊爾沙特希望你能幫助他找到由於該bug導致的重排列。

考慮一個序列 $p = [p_0, \ldots, p_{n-1}]$ ，該序列中 0 到 $n-1$ 這 n 個數每個數恰好出現一次。我們稱該序列為一個排列。考慮集合中的一個元素，該元素的二進位表達為 $a_0, \ldots, a_{n-1}$ （ a_0 是最高位）。當函數 `compile_set()` 被調用時，這個元素將被元素 $a_{p_0}, a_{p_1}, \ldots, a_{p_{n-1}}$ 替代。

同樣的排列 p 會被用於每個元素的二進位位元的重排列。這個排列 p 可以是任意一個排列，包括 $p_i = i$， $0 \le i \le n-1$。

例如，假設 $n = 4$, $p = [2, 1, 3, 0]$, 你已經插入的整數所對應的二進位表示為 `0000`, `1100` 和 `0111`.調用函數 `compile_set` 會將三個元素分別變成 `0000`, `0101` 和 `1110`。

你的任務是寫一個程式，該程式通過和資料結構的交互來找到排列 p。該程式應該（按照下列順序）：

1. 選擇一個 n 位元整數的集合，
2. 插入這些整數到資料結構中，
3. 調用函數 `compile_set` 來**啟**動bug，
4. 檢**查**某些元素是否在修改以後的集合當中，
5. 利用該資訊來判斷和返回排列 p。

注意你的程式只能調用函數 `compile_set` 一次。

而且，你的程式調用庫函數的次數是有限制的。具體的，可以調用的庫函數及其調用次數限制為

- 庫函數 `add_element` 最多調用 w 次 （w 表示“寫”）
- 庫函數 `check_element` 最多調用 r 次 （r 表示“讀”）。

實現細節

你應該實現一個函數（方法）：

- `int[] restore_permutation(int n, int w, int r)`
  - `n`: 集合中每個元素的二進位表示的位元數 (也是排列 p 的長度）。
  - `w`: 你的程式調用函數 `add_element` 的最大次數。
  - `r`: 你的程式調用函數 `check_element` 的最大次數。
  - 函數應該返回恢復的排列 p。

C 語言的函數原型略有不同：

- `void restore_permutation(int n, int w, int r, int* result)`
  - `n`, `w` 和 `r` 同上。
  - 函數返回排列 p 的方式是將 p 存儲到提供的陣列 `result` 中：對每個 i，函數將 p_i 存到 `result[i]`中。

庫函數

為了和資料結構進行交互，你的程式應該使用下列三個函數（方法）

- `void add_element(string x)`
  該函數將元素 `x` 添加到集合中。
  - `x`: 一個由 `'0'` 和 `'1'` 構成的字串，它是要添加到集合中的元素的二進位表示。`x`的長度必須是 n。

- `void compile_set()`
  該函數必須調用一次且只能調用一次。在調用該函數之後，你的程式不能再調用函數 `add_element()`。在調用該函數之前，你的程式也不能調用函數 `check_element()`。
- `boolean check_element(string x)`
  該函數檢**查**元素`x`是否在修改以後的集合當中。
  - `x`: 一個由`'0'` 和 `'1'`構成的字串，它是要檢**查**的元素的二進位表示。`x`的長度必須是 n。
  - 如果元素 `x` 在修改後的集合中，則返回 `true`，否則返回`false`。

注意：如果你的程式違反上述的任何一條限制，其評分輸出將是 "Wrong Answer"。

對於所有的字串，第一個字元都表示所對應整數的最高位元。

評測程式在調用函數 `restore_permutation` 之前已經確定了排列 p。

請使用提供的範本檔，參考關於你所使用的程式設計語言的實現細節。

例子

評測程式執行下列函式呼叫：

- `restore_permutation(4, 16, 16)`. 我們有 $n = 4$ 而且程式最多執行 16 次"寫"和 16 次"讀"操作。

程式執行下列函式呼叫：

- `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`

只有一個排列和函數 `check_element()` 返回的**值**一致：
排列 $p = [2, 1, 3, 0]$。因此, `restore_permutation` 應該返回 `[2, 1, 3, 0]`.

子任務

1. (20分) $n = 8$, $w = 256$, $r = 256$, 最多有兩個下標 i 滿足 $p_i \neq i$ ($0 \leq i \leq n - 1$),
2. (18分) $n = 32$, $w = 320$, $r = 1024$,
3. (11分) $n = 32$, $w = 1024$, $r = 320$,
4. (21分) $n = 128$, $w = 1792$, $r = 1792$,
5. (30分) $n = 128$, $w = 896$, $r = 896$.

樣例測試程式

樣例測評程式按照以下格式讀入輸入：

- 第一行: 整數 n, w, r,
- 第二行: n 個整數表示排列 p 的元素。