Unscrambling a Messy Bug

L'Ilshat és un enginyer de software que treballa en estructures de dades eficients. Un dia es va inventar una nova estructura de dades. Aquesta estructura pot desar un conjunt d'enters no negatius de n bits, on n és una potència de dos. És a dir, $n = 2^b$ per a algun enter no negatiu b.

Inicialment l'estructura està buida. Un programa que utilitzi aquesta estructura de dades ha de seguir les regles següents:

- El programa pot afegir elements que siguin enters de n bits, d'un en un, utilitzant la funció `add_element(x)`. Si el programa intenta afegir un element que ja és present a l'estructura de dades, no passa res.
- Després d'afegir l'útlim element el programa ha de cridar la funció `compile_set()` exactament una vegada.
- Finalment, el programa pot cridar la funció `check_element(x)` per mirar si un element x és present a l'estructura de dades. Aquesta funció es pot cridar més d'una vegada.

Quan l'Ilshat va implementar per primera vegada aquesta estructura de dades, va fer un error en la funció `compile_set()`. L'error reordena els digits binaris de cada element del conjunt d'una determinada manera. L'Ilshat vol que trobeu la manera exacta en què l'error reordena els dígits.

Formalment, considereu una seqüència $p = [p_0, \ldots, p_{n-1}]$ on cada nombre entre 0 i $n-1$ hi apareix exactament una vegada. D'aquesta seqüència en direm *permutació*. Considereu un element del conjunt, amb dígits en binari $a_0, \ldots, a_{n-1}$ (essent a_0 el bit més significatiu). Quan es crida la funció `compile_set()`, aquest element se substitueix per l'element $a_{p_0}, a_{p_1}, \ldots, a_{p_{n-1}}$.

Els dígits de tots els elements es reordenen amb la mateixa permutació p. Qualsevol permutació és possible, i fins i tot pot passar que $p_i = i$ per a cada $0 \leq i \leq n-1$.

Per exemple, suposeu que $n = 4$, $p = [2, 1, 3, 0]$, i heu inserit en el conjunt els enters amb representació binària `0000`, `1100` i `0111`. Quan es crida la funció `compile_set` aquests elements es canvien per `0000`, `0101` i `1110`, respectivament.

La teva tasca consisteix a escriure un programa que trobi la permutació p interactuant amb l'estructura de dades. Per això cal que faci el següent (en aquest ordre):

1. triar un conjunt d'enters de n bits,
2. inserir aquests enters en l'estructura de dades,
3. cridar la funció `compile_set`, perquè es produeixi l'error,

4. comprovar si el conjunt modificat conté certs elements,
5. fer servir aquesta informació per tal de calcular i retornar la permutació p.

Teniu en compte que el vostre programa només pot cridar la funció `compile_set` una única vegada.

A més a més, el nombre de vegades que el vostre programa pot cridar les funcions donades està limitat. En concret, el vostre programa pot

- cridar `add_element` com a molt w vegades (w ve de "writes", escriptures en anglès),
- cridar `check_element` com a molt r vegades (r ve de "reads", lectures en anglès).

Detalls d'implementació

Se us demana que implementeu una funció (mètode):

- `int[] restore_permutation(int n, int w, int r)`
  - `n`: el nombre de bits en la representació binària de cada element del conjunt (i també la mida de p).
  - `w`: el nombre màxim d'operacions del tipus `add_element` que pot fer el vostre programa.
  - `r`: el nombre màxim d'operacions del tipus `check_element` que pot fer el vostre programa.
  - la funció ha de retornar la permutació p.

En el llenguatge C, la capçalera de la funció és una mica diferent:

- `void restore_permutation(int n, int w, int r, int* result)`
  - `n`, `w` i `r` tenen el mateix significat que abans.
  - la funció ha de retornar la permutació p desant-la a l'array d'entrada `result`: per a cada i, ha de desar el valor p_i en `result[i]`.

Funcions proporcionades

Per tal de poder interactuar amb l'estructura de dades, el vostre programa ha d'utilitzar les tres següents funcions (mètodes):

- `void add_element(string x)`
  Aquesta funció afegeix al conjunt l'element que descriu `x`.
  - `x`: un string format per caràcters `'0'` i `'1'` que dóna la representació binària d'un enter que s'ha d'afegir al conjunt. La mida de `x` ha de ser n.

- `void compile_set()`
  Aquesta funció s'ha de cridar exactament una vegada. Després de fer-ho el vostre programa no podrà cridar més la funció `add_element()`. El vostre programa no pot cridar la funció `check_element()` abans de fer aquesta crida.
- `boolean check_element(string x)`
  Aquesta funció comprova si l'element `x` es troba en el conjunt modificat.
  - `x`: un string format per caràcters `'0'` i `'1'` que dóna la representació binària de l'enter que s'ha de comprovar. La mida de `x` ha de ser n.
  - retorna `true` si l'element `x` es troba en el conjunt modificat, i `false` si no.

Teniu en compte que si el vostre programa no satisfà alguna de les restriccions anteriors, el resultat de l'avaluació serà "Wrong Answer".

Per a tots els strings, el primer caràcter es correspon amb el bit més significatiu de l'enter corresponent.

El grader fixa la permutació p abans que es cridi la funció `restore_permutation`.

Si us plau, feu servir les plantilles donades per veure els detalls d'implementació en el vostre llenguatge de programació.

Exemple

El grader fa la següent crida:

- `restore_permutation(4, 16, 16)`. Tenim que $n = 4$ i el programa pot fer com a molt 16 "writes" i 16 "reads".

El programa fa les següents crides:

- `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`

Només hi ha una permutació que sigui consistent amb aquests valors que retorna `check_element()`: la permutació $p = [2, 1, 3, 0]$. Per tant, `restore_permutation` hauria de retornar `[2, 1, 3, 0]`.

Subtasques

1. (20 punts) $n = 8$, $w = 256$, $r = 256$, $p_i \neq i$ per com a molt 2 índexs i ($0 \leq i \leq n - 1$),
2. (18 punts) $n = 32$, $w = 320$, $r = 1024$,
3. (11 punts) $n = 32$, $w = 1024$, $r = 320$,
4. (21 punts) $n = 128$, $w = 1792$, $r = 1792$,
5. (30 punts) $n = 128$, $w = 896$, $r = 896$.

Grader de mostra

El grader de mostra llegeix l'entrada en el format següent:

- línia 1: enters n, w, r,
- línia 2: n enters que donen els elements de p.