แก้ขยุ้มบั๊กเขยิง

อิลแซ็ทเป็นวิศวกรซอฟต์แวร์ เขาทำงานเกี่ยวกับโครงสร้างข้อมูลที่มีประสิทธิภาพ วันหนึ่งเขาคิดค้นโครงสร้างข้อมูลชนิดใหม่ขึ้นมา โครงสร้างข้อมูลนี้สามารถจัดเก็บเซ็ตของจำนวนเต็ม *ไม่ติดลบ*ขนาด n บิต โดยที่ n เป็นผลจากการจับเลขสองมายกกำลัง กล่าวคือ $n = 2^b$ สำหรับจำนวนเต็มไม่ติดลบ b บางตัว

ตอนเริ่มต้นโครงสร้างข้อมูลนี้ว่างเปล่า โปรแกรมที่ใช้โครงสร้างข้อมูลนี้จะต้องทำตามกฎต่อไปนี้

- โปรแกรมสามารถเพิ่มสมาชิกที่เป็นจำนวนเต็มขนาด n บิตเข้าสู่โครงสร้างข้อมูล ทีละตัว โดยใช้ฟังก์ชัน `add_element(x)` หากโปรแกรมพยายามเพิ่มสมาชิกที่มีอยู่แล้วในโครงสร้างข้อมูล จะไม่มีอะไรเกิดขึ้น
- หลังจากที่เพิ่มสมาชิกตัวสุดท้ายเสร็จแล้ว โปรแกรมจะต้องเรียกฟังก์ชัน `compile_set()` หนึ่งครั้งพอดี
- ในตอนท้ายสุด โปรแกรมสามารถเรียกฟังก์ชัน `check_element(x)` เพื่อตรวจสอบว่า x อยู่ในโครงสร้างข้อมูลนี้หรือไม่ ฟังก์ชันนี้อาจถูกเรียกหลายครั้ง

ตอนแรกที่อิลแซ็ทลงมือเขียนโครงสร้างข้อมูลนี้ ฟังก์ชัน `compile_set()` ดันมีบั๊ก บั๊กนี้สลับสับเปลี่ยนลำดับของตัวเลขหลักต่างๆ ในฐานสองของสมาชิกแต่ละตัวในเซ็ต โดยสลับในลักษณะเดียวกัน อิลแซ็ทอยากให้คุณช่วยค้นหาการสลับสับเปลี่ยนที่เป็นผลพวงจากบั๊กดังกล่าว

กล่าวอย่างเป็นทางการ เมื่อพิจารณาลำดับ $p = [p_0, \ldots, p_{n-1}]$ โดยที่ทุกจำนวนตั้งแต่ 0 ถึง $n-1$ ปรากฏหนึ่งครั้งพอดี เราเรียกลำดับนั้นว่า *การเรียงสับเปลี่ยน* เมื่อพิจารณาสมาชิกตัวหนึ่งของเซ็ตซึ่งเขียนในฐานสองได้เป็น $a_0, \ldots, a_{n-1}$ (โดย a_0 เป็นหลักที่มีค่าสูงสุด) เมื่อฟังก์ชัน `compile_set()` ถูกเรียก สมาชิกตัวนี้จะถูกแทนที่ด้วย $a_{p_0}, a_{p_1}, \ldots, a_{p_{n-1}}$

สมาชิกทุกตัวถูกสลับบิตด้วยการเรียงสับเปลี่ยน p เดียวกัน การเรียงสับเปลี่ยนนั้นอาจจะเป็นอะไรก็ได้ รวมถึงกรณีที่ $p_i = i$ สำหรับแต่ละ $0 \le i \le n-1$ ก็อาจจะเป็นไปได้

ตัวอย่าง สมมติว่า $n = 4$, $p = [2, 1, 3, 0]$ และคุณเพิ่งจะเพิ่มตัวเลขฐานสองเหล่านี้เข้าไปในเซ็ต `0000`, `1100` และ `0111` การเรียกฟังก์ชัน `compile_set` จะเปลี่ยนสมาชิกเหล่านั้นเป็น `0000`, `0101` และ `1110` ตามลำดับ

ภารกิจของคุณคือการเขียนโปรแกรมเพื่อค้นหาการเรียงสับเปลี่ยน p โดยโต้ตอบกับโครงสร้างข้อมูล โปรแกรมคุณจะต้องทำสิ่งต่อไปนี้ ในลำดับต่อไปนี้

1. เลือกเซ็ตของจำนวนเต็มขนาด n บิต
2. เพิ่มจำนวนเต็มเหล่านั้นเข้าสู่โครงสร้างข้อมูลนี้
3. เรียกฟังก์ชัน `compile_set` เพื่อกระตุ้นให้เกิดบั๊ก
4. ตรวจสอบสมาชิกภาพของตัวเลขบางตัวในเซ็ตที่ถูกดัดแปลงนี้
5. ใช้ข้อมูลดังกล่าวในการบ่งชี้ และคืนค่า การเรียงสับเปลี่ยน p

ให้สังเกตว่า โปรแกรมของคุณจะเรียกฟังก์ชัน `compile_set` ได้เพียงหนึ่งครั้งเท่านั้น

นอกจากนี้ยังมีข้อจำกัดเกี่ยวกับจำนวนครั้งที่โปรแกรมของคุณสามารถเรียกฟังก์ชันของไลบรารี่ นั่นคือ โปรแกรมมีสิทธิ์

- เรียก `add_element` ได้ไม่เกิน w ครั้ง (w ย่อมาจาก "writes" หรือเขียน)
- เรียก `check_element` ได้ไม่เกิน r ครั้ง (r ย่อมาจาก "reads" หรืออ่าน)

รายละเอียดการเขียนโปรแกรม

จงเขียนฟังก์ชันหนึ่งฟังก์ชัน

- `int[] restore_permutation(int n, int w, int r)`
  - `n` คือจำนวนบิตเมื่อเขียนสมาชิกของเซ็ตแต่ละตัวเป็นเลขฐานสอง และคือความยาวของ p ด้วยเช่นกัน
  - `w` คือจำนวนครั้งสูงสุดที่โปรแกรมของคุณสามารถเรียก `add_element`
  - `r` คือจำนวนครั้งสูงสุดที่โปรแกรมของคุณสามารถเรียก `check_element`
  - ฟังก์ชันนี้จะต้องคืนค่าการเรียงสับเปลี่ยน p ที่ถูกค้นพบ

สำหรับภาษาซี หัวฟังก์ชันมีความแตกต่างเล็กน้อย

- `void restore_permutation(int n, int w, int r, int* result)`
  - `n`, `w` และ `r` มีความหมายเหมือนกับที่กล่าวไว้ด้านบน
  - ฟังก์ชันนี้จะต้องคืนค่าการเรียงสับเปลี่ยน p โดยจัดเก็บไว้ในอาเรย์ `result`: สำหรับแต่ละ i มันควรเก็บค่าของ p_i ไว้ที่ `result[i]`

ฟังก์ชันของไลบรารี่

โปรแกรมของคุณสามารถโต้ตอบกับโครงสร้างข้อมูลได้ด้วยการใช้ฟังก์ชันสามฟังก์ชันต่อไปนี้

- `void add_element(string x)`
  ฟังก์ชันนี้เพิ่มสมาชิกที่กำหนดโดย `x` เข้าไปในเซ็ต
  - `x` คือสตริงของอักขระ `'0'` และ `'1'` ซึ่งเป็นวิธีเขียนเลขฐานสองของจำนวนเต็มที่จะเพิ่มเข้าสู่เซ็ต ความยาวของ `x` จะต้องเป็น n

- `void compile_set()`
  ฟังก์ชันนี้จะต้องถูกเรียกหนึ่งครั้งพอดี โปรแกรมของคุณไม่สามารถเรียก `add_element()` หลังจากนี้ โปรแกรมของคุณไม่สามารถเรียก `check_element()` ก่อนหน้านี้
- `boolean check_element(string x)`
  ฟังก์ชันนี้ตรวจสอบว่า `x` เป็นสมาชิกของเซ็ตที่ถูกดัดแปลงนี้หรือไม่
  - `x` คือสตริงของอักขระ `'0'` และ `'1'` ซึ่งเป็นวิธีเขียนเลขฐานสองของจำนวนเต็มที่ต้องการตรวจสอบ ความยาวของ `x` จะต้องเป็น n
  - คืนค่า `true` ถ้าตัวเลข `x` เป็นสมาชิกของเซ็ตที่ถูกดัดแปลง, และ `false` ในกรณีอื่น

โปรดทราบ ถ้าโปรแกรมของคุณฝ่าฝืนข้อจำกัดที่กล่าวไว้ด้านบน ผลของการตรวจจะเป็น "Wrong Answer" (คำตอบผิด)

สำหรับสตริงทุกตัว อักขระตัวแรกคือบิตที่มีค่าสูงสุดของจำนวนเต็ม

เกรดเดอร์จะกำหนดการเรียงสับเปลี่ยน p ให้ตายตัวก่อนที่ฟังก์ชัน `restore_permutation` จะถูกเรียก

สำหรับรายละเอียดการเขียนโปรแกรมในภาษาของคุณ โปรดดูไฟล์ต้นแบบที่ได้เตรียมไว้ให้

ตัวอย่าง

เกรดเดอร์เรียกฟังก์ชันดังต่อไปนี้

- `restore_permutation(4, 16, 16)` ให้ $n = 4$ และให้โปรแกรมสามารถ "write" (เขียน) ได้ 16 ครั้งเป็นอย่างมาก และ "read" (อ่าน) ได้ 16 ครั้งเป็นอย่างมาก

โปรแกรมเรียกฟังก์ชันดังต่อไปนี้

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

มีการเรียงสับเปลี่ยนเพียงแบบเดียวเท่านั้นที่สอดคล้องกับค่าที่คืนมาจาก `check_element()` นั่นคือ การเรียงสับเปลี่ยน $p = [2, 1, 3, 0]$ ดังนั้น `restore_permutation` ต้องคืนค่า `[2, 1, 3, 0]`

ปัญหาย่อย

1. (20 คะแนน) $n = 8$, $w = 256$, $r = 256$, มีตำแหน่งที่ $p_i \neq i$ ไม่เกิน 2 ตำแหน่งเท่านั้น ($0 \leq i \leq n - 1$)
2. (18 คะแนน) $n = 32$, $w = 320$, $r = 1024$
3. (11 คะแนน) $n = 32$, $w = 1024$, $r = 320$
4. (21 คะแนน) $n = 128$, $w = 1792$, $r = 1792$
5. (30 คะแนน) $n = 128$, $w = 896$, $r = 896$

เกรดเดอร์ตัวอย่าง

เกรดเดอร์ตัวอย่างอ่านข้อมูลนำเข้าในรูปแบบต่อไปนี้

- บรรทัดที่ 1: จำนวนเต็ม n, w, r
- บรรทัดที่ 2: จำนวนเต็ม n ตัวซึ่งระบุสมาชิกของ p