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

messy
Country: ISR

Unscrambling a Messy Bug

אייל (Ilshat) הוא מהנדס תוכנה שעובד על מבני נתונים יעילים. יום אחד הוא המציא מבנה נתונים חדש. מבנה זה יכול לאחסן קבוצה של מספרים אי-שליליים בעלי n ביטים, כאשר n הוא חזקה של שתיים. כלומר, $n = 2^b$ עבור מספר שלם אי-שלילי b.

בהתחלה מבנה הנתונים ריק. תוכנית שמשתמשת במבנה הנתונים צריכה לציית לכללים הבאים:

- התוכנית יכולה להוסיף איברים שהם מספרים שלמים בעלי n ביטים לתוך מבנה הנתונים, מספר אחד בכל פעם, בעזרת הפונקציה add_element(x). אם התוכנית מנסה להוסיף איבר שכבר קיים במבנה הנתונים, לא קורה כלום.
- לאחר הוספת האיבר האחרון, התוכנית צריכה לקרוא לפונקציה compile_set() פעם אחת בדיוק.
- לבסוף, לתוכנית מותר לקרוא לפונקציה check_element(x) על מנת לבדוק האם האיבר x קיים במבנה הנתונים. ניתן להשתמש בפונקציה זו מספר פעמים.

כאשר אייל מימש את מבנה הנתונים לראשונה, היה לו באג בפונקציה compile_set(). הבאג משנה את סדר הספרות הבינאריות של כל אחד מאיברי הקבוצה באותו אופן. אייל מבקש מכם למצוא את שינוי הסדר הנגרם בעקבות הבאג.

פורמלית, נתבונן בסדרה $p = [p_0, \ldots, p_{n-1}]$ שבה כל מספר בין 0 ל-$n-1$ מופיע בדיוק פעם אחת. נקרא לסדרה כזו פרמוטציה. נתבונן באיבר מהקבוצה, שספרותיו הבינאריות הן $a_0, \ldots, a_{n-1}$ (כאשר a_0 הוא ה-most significant bit). כאשר קוראים לפונקציה compile_set(), איבר זה יוחלף באיבר $a_{p_0}, a_{p_1}, \ldots, a_{p_{n-1}}$.

הספרות הבינאריות של כל אחד מאיברי הקבוצה משתנות לפי אותה פרמוטציה p. כל פרמוטציה היא אפשרית, כולל האפשרות שמתקיים $p_i = i$ לכל $0 \leq i \leq n-1$.

לדוגמה, נניח שמתקיים $n = 4$, $p = [2, 1, 3, 0]$, והכנסתם לקבוצה מספרים שהייצוגים הבינאריים שלהם הם 0000, 1100 ו-0111. קריאה לפונקציה compile_set משנה את האיברים ל-0000, 0101 ו-1110, בהתאמה.

משימתכם היא לכתוב תוכנית שמוצאת את הפרמוטציה p על ידי אינטרקציה עם מבנה הנתונים. התוכנית צריכה (בסדר הבא):

1. לבחור קבוצה של מספרים שלמים בעלי n ביטים,
2. להכניס את המספרים לתוך מבנה הנתונים,

3. לקרוא לפונקציה compile_set כדי לגרום לבאג,
4. לבדוק את הימצאותם של איברים כלשהם בקבוצה החדשה,
5. להשתמש במידע זה כדי לגלות ולהחזיר את הפרמוטציה p.

שימו לב שלתוכנית שלכם מותר לקרוא לפונקציה compile_set פעם אחת בלבד.

בנוסף, יש מגבלה על מספר הפעמים שהתוכנית שלכם יכולה לקרוא לפונקציות הספרייה. ספציפית, לתוכנית מותר:

- לקרוא ל-add_element לכל היותר w פעמים (האות w מסמנת את המילה "writes").
- לקרוא ל-check_element לכל היותר r פעמים (האות r מסמנת את המילה "reads").

פרטי מימוש

עליכם לממש את הפונקציה הבאה (שיטה):

- int[] restore_permutation(int n, int w, int r)
  - n: מספר הביטים בייצוג הבינארי של כל איבר בקבוצה (וזהו גם האורך של p).
  - w: המספר המקסימלי של קריאות לפונקציה add_element שמותר לתוכניתכם לבצע.
  - r: המספר המקסימלי של קריאות לפונקציה check_element שמותר לתוכניתכם לבצע.
  - הפונקציה צריכה להחזיר את הפרמוטציה p.

בשפת C, החתימה של הפונקציה מעט שונה:

- void restore_permutation(int n, int w, int r, int* result)
  - n, w ו-r באותה משמעות כפי שתואר לעיל.
  - הפונקציה צריכה להחזיר את הפרמוטציה p על ידי כתיבתה למערך result: לכל i, יש לכתוב את הערך p_i לתוך result[i].

פונקציות ספרייה

על מנת לתקשר עם מבנה הנתונים, תוכניתכם צריכה להשתמש בשלוש הפונקציות הבאות (שיטות):

- void add_element(string x)

פונקציה זו מוסיפה את האיבר שמתואר על ידי x לקבוצה.

- x: מחרוזת של תווים $'0'$ ו-$'1'$ שמתארת את הייצוג הבינארי של המספר השלם שרוצים להוסיף לקבוצה. האורך של x חייב להיות n.

- void compile_set()

תוכניתכם חייבת לקרוא לפונקציה זו בדיוק פעם אחת. לתוכניתכם אסור לקרוא ל-add_element() לאחר קריאה לפונקציה זו. לתוכניתכם אסור לקרוא ל-check_element() לפני הקריאה לפונקציה זו.

- boolean check_element(string x)

פונקציה זו בודקת האם האיבר x קיים בקבוצה החדשה.

  - x: מחרוזת של תווים $'0'$ ו-$'1'$ שמתארת את הייצוג הבינארי של האיבר שאותו רוצים לבדוק. האורך של x חייב להיות n.
  - הפונקציה מחזירה true אם האיבר x נמצא בקבוצה החדשה, ואחרת false.

שימו לב שאם תוכניתכם חורגת מהמגבלות שתוארו לעיל, התוצאה תהיה "Wrong answer".

בכל המחרוזות, התו הראשון הוא ה-most significant bit של המספר.

הגריידר קובע את הפרמוטציה p לפני שהוא קורא ל-restore_permutation.

אנא השתמשו בקבצי ה-template לפרטי המימוש עבור שפת התכנות שלכם.

דוגמה

הגריידר מבצע את הקריאה הבאה:

- restore_premutation(4, 16, 16). מתקיים $n = 4$ והתוכנית יכולה לבצע לכל היותר 16 כתיבות ("writes") ו-16 קריאות ("reads").

התוכנית מבצעת את הקריאות הבאות:

- 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

קיימת רק פרמוטציה אחת שמתאימה לערכים שהוחזרו על ידי check_element(): הפרמוטציה היא $p = [2, 1, 3, 0]$. לכן, restore_permutation צריכה להחזיר $[2, 1, 3, 0]$.

תת משימות

1. (20 נקודות): $n = 8$, $w = 256$, $r = 256$, $p_i \neq i$ עבור לכל היותר שני ערכים של i ($0 \leq i \leq n - 1$).
2. (18 נקודות): $n = 32$, $w = 320$, $r = 1024$.
3. (11 נקודות): $n = 32$, $w = 1024$, $r = 320$.
4. (21 נקודות): $n = 128$, $w = 1792$, $r = 1792$.
5. (30 נקודות): $n = 128$, $w = 896$, $r = 896$.

גריידר לדוגמה

הגריידר קורא את הקלט בפורמט הבא:

- שורה 1: המספר n ואחריו w ואחריו r.
- שורה 2: n מספרים שלמים שמתארים את איברי p.