אוגרי הזזת ביטים

כריסטופר המהנדס עובד על סוג חדש של מעבד מחשבים.

למעבד יש גישה ל-m תאי זיכרון שונים של b-ביט (כש-$m = 100$ ו-$b = 2000$), הנקראים **רגיסטרים**, וממוספרים מ-0 עד $m - 1$. אנו מסמנים את הרגיסטרים על ידי $r[0], r[1], \ldots, r[m-1]$. כל רגיסטר הוא מערך של b ביטים, הממוספרים מ-0 (הביט הימני ביותר) ל-$b - 1$ (הביט השמאלי ביותר). לכל i $(0 \leq i \leq m - 1)$ ולכל j $(0 \leq j \leq b - 1)$, אנו מסמנים את הביט ה-j של רגיסטר i על ידי $r[i][j]$.

לכל רצף של ביטים $d_0, d_1, \ldots, d_{l-1}$ (באורך שרירותי l), **הערך השלם** של הרצף שווה ל-$2^0 \cdot d_0 + 2^1 \cdot d_1 + \ldots + 2^{l-1} \cdot d_{l-1}$. נגיד ש**הערך השלם המאוחסן ברגיסטר** i הוא הערך השלם של רצף הביטים בו, כלומר, הוא $2^0 \cdot r[i][0] + 2^1 \cdot r[i][1] + \ldots + 2^{b-1} \cdot r[i][b-1]$.

למעבד 9 סוגים של **פקודות** שניתן להשתמש בהן על מנת לשנות את הביטים ברגיסטרים. כל פקודה פועלת על רגיסטר אחד או יותר ומאכסנת את הפלט באחד מהרגיסטרים. מכאן ואילך, אנו משתמשים ב-$x := y$ כדי לסמן את הפעולה של שינוי הערך של x כך שהוא נהיה שווה ל-y. הפעולות המבוצעות על ידי כל סוג של פקודה מתוארות מטה.

- $move(t, y)$: העתקת המערך של הביטים ברגיסטר y לרגיסטר t. לכל j $(0 \leq j \leq b - 1)$, הצבת $r[t][j] := r[y][j]$.
- $store(t, v)$: קביעת הרגיסטר t להיות שווה ל-v, כאשר v הוא מערך של b ביטים. לכל j $(0 \leq j \leq b - 1)$, הצבת $r[t][j] := v[j]$.
- $and(t, x, y)$: לקיחת ה-bitwise-AND של רגיסטרים x ו-y, ואחסון התוצאה ברגיסטר t. לכל j $(0 \leq j \leq b - 1)$, הצבת $r[t][j] := 1$ אם **גם** $r[x][j]$ **וגם** $r[y][j]$ הם 1, והצבת $r[t][j] := 0$ אחרת.
- $or(t, x, y)$: לקיחת ה-bitwise-OR של רגיסטרים x ו-y, ואחסון התוצאה ברגיסטר t. לכל j $(0 \leq j \leq b - 1)$, הצבת $r[t][j] := 1$ אם **לפחות אחד** מבין $r[x][j]$ ו-$r[y][j]$ הוא 1, והצבת $r[t][j] := 0$ אחרת.
- $xor(t, x, y)$: לקיחת ה-bitwise-XOR של רגיסטרים x ו-y, ואחסון התוצאה ברגיסטר t. לכל j $(0 \leq j \leq b - 1)$, הצבת $r[t][j] := 1$ אם **בדיוק אחד** מבין $r[x][j]$ ו-$r[y][j]$ הוא 1, והצבת $r[t][j] := 0$ אחרת.
- $not(t, x)$: לקיחת ה-bitwise-NOT של רגיסטר x, ואחסון התוצאה ברגיסטר t. לכל j $(0 \leq j \leq b - 1)$, הצבת $r[t][j] := 1 - r[x][j]$.
- $left(t, x, p)$: הזזת כל הביטים ברגיסטר x p מקומות שמאלה, ואחסון התוצאה ברגיסטר t. התוצאה של הזזת הביטים ברגיסטר x p מקומות שמאלה היא מערך v המורכב מ-b ביטים. לכל j $(0 \leq j \leq b - 1)$, אם $j \geq p$ אז $v[j] = r[x][j - p]$, ו-$v[j] = 0$ אחרת. לכל j $(0 \leq j \leq b - 1)$, הצב $r[t][j] := v[j]$.
- $right(t, x, p)$: הזזת כל הביטים ברגיסטר x p מקומות ימינה, ואחסון התוצאה ברגיסטר t. התוצאה של הזזת הביטים ברגיסטר x p מקומות ימינה היא מערך v המורכב מ-b ביטים. לכל j $(0 \leq j \leq b - 1)$,

$v[j] = r[x][j+p]$ אם $j \leq b-1-p$, ו-$v[j] = 0$ אחרת. לכל j $(0 \leq j \leq b-1)$, הצב $r[t][j] := v[j]$.

- $add(t, x, y)$: סכימת הערכים השלמים המאוחסנים ברגיסטר x וברגיסטר y, ואחסון התוצאה ברגיסטר t. הסכימה מתבצעת מודולו 2^b. פורמלית, יהי X הערך השלם המאוחסן ברגיסטר x, ו-Y הערך השלם המאוחסן ברגיסטר y לפני הפעולה. יהי T הערך השלם המאוחסן ברגיסטר t אחרי הפעולה. אם $X + Y < 2^b$, קבע את הביטים של t, כך ש-$T = X + Y$. אחרת, קבע את הביטים של t, כך ש-$T = X + Y - 2^b$.

כריסטופר מעוניין שתפתרו שני סוגים של משימות באמצעות המעבד החדש. סוג המשימה מסומן על ידי מספר שלם s. עבור שני סוגי המשימות, עליכם להפיק **תוכנית**, כלומר רצף של פקודות שהוגדרו למעלה.

ה**קלט** לתוכנית מורכב מ-n מספרים שלמים $a[0], a[1], \ldots, a[n-1]$, בכל אחד k -ביטים, כלומר, $a[i] < 2^k$ $(0 \leq i \leq n-1)$. לפני שהתוכנית מורצת, כל המספרים בקלט מאוחסנים ברצף ברגיסטר 0, כך שלכל i $(0 \leq i \leq n-1)$ הערך השלם של הרצף של k הביטים $r[0][i \cdot k], r[0][i \cdot k + 1], \ldots, r[0][(i+1) \cdot k - 1]$ שווה ל-$a[i]$. שימו לב ש-$n \cdot k \leq b$. כל שאר הביטים ברגיסטר 0 (כלומר, בעלי האינדקסים בין $n \cdot k$ ל-$b-1$, כולל) וכל הביטים בכל שאר הרגיסטרים מאותחלים ל-0.

הרצת תוכנית בנויה מהרצת הפקודות שלה לפי הסדר. אחרי שהפקודה האחרונה מתבצעת, ה**פלט** של התוכנית מחושב לפי הערך הסופי של הביטים ברגיסטר 0. ספציפית, הפלט הוא רצף של n מספרים שלמים $c[0], c[1], \ldots, c[n-1]$, כאשר לכל i $(0 \leq i \leq n-1)$, $c[i]$ הוא הערך השלם של רצף הבנוי מהביטים $i \cdot k$ עד $(i+1) \cdot k - 1$ של הרגיסטר 0. שימו לב שלאחר הרצת התוכנית שאר הביטים של הרגיסטר 0 (בעלי אינדקסים גדולים שווים ל-$n \cdot k$) וכל הביטים של כל שאר הרגיסטרים יכולים להיות שרירותיים.

- המשימה הראשונה $(s = 0)$ היא למצוא את המספר הקטן ביותר מבין המספרים בקלט $a[0], a[1], \ldots, a[n-1]$. ספציפית, על $c[0]$ להיות המינימום של $a[0], a[1], \ldots, a[n-1]$. הערכים של $c[1], c[2], \ldots, c[n-1]$ יכולים להיות שרירותיים.
- המשימה השניה $(s = 1)$ היא למיין את המספרים בקלט $a[0], a[1], \ldots, a[n-1]$ בסדר לא יורד. ספציפית, לכל i $(0 \leq i \leq n-1)$, על $c[i]$ להיות שווה למספר ה-$1+i$ הקטן ביותר מבין $a[0], a[1], \ldots, a[n-1]$ (למשל, $c[0]$ הוא המספר הקטן ביותר מבין המספרים בקלט).

ספקו לכריסטופר את התוכניות, הבנויות מלכל היותר q פעולות כל אחת, שיכולות לפתור את המשימות האלו.

פרטי מימוש

עליכם לממש את הפונקציה הבאה:

```
void construct_instructions(int s, int n, int k, int q)
```

- s: סוג המשימה.
- n: מספר המספרים השלמים בקלט.
- k: מספר הביטים בכל שלם בקלט.
- q: מספר הפקודות המירבי המותר.
- פונקציה זו נקראת בדיוק פעם אחת וצריכה לבנות רצף של פקודות כדי לבצע את המשימה המבוקשת.

על פונקציה זו לקרוא לאחת או יותר מהפונקציות הבאות על מנת לבנות רצף פקודות:

```
void append_move(int t, int y)
void append_store(int t, bool[] v)
void append_and(int t, int x, int y)
void append_or(int t, int x, int y)
void append_xor(int t, int x, int y)
void append_not(int t, int x)
void append_left(int t, int x, int p)
void append_right(int t, int x, int p)
void append_add(int t, int x, int y)
```

- כל פונקציה מוסיפה פקודת $move(t, y)$, $store(t, v)$, $and(t, x, y)$, $or(t, x, y)$, $xor(t, x, y)$, $not(t, x)$, $left(t, x, p)$, $right(t, x, p)$ או $add(t, x, y)$ לתוכנית, בהתאמה.
- לכל הפקודות הרלוונטיות, t , x , y חייבים להיות לפחות 0 ולכל היותר $m - 1$.
- לכל הפקודות הרלוונטיות, t , x , y לא בהכרח שונים זה מזה.
- עבור הפקודות $left$ ו-$right$, על p להיות לפחות 0 ולכל היותר b .
- עבור הפקודה $store$, האורך של v חייב להיות b .

אתם יכולים גם לקרוא לפונקציה הבאה כדי לעזור לכם בבדיקת הפתרון שלכם:

```
void append_print(int t)
```

- כל קריאה לפונקציה זו לא תעשה כלום במהלך הבדיקה של הפתרון שלכם.
- בגריידר לדוגמה, פונקציה זו מוסיפה פקודת $print(t)$ לתוכנית.
- כשהגריידר לדוגמה נתקל בפקודת $print(t)$ במהלך הרצת תוכנית, הוא מדפיס n מספרים שלמים של k -ביט הנוצרים על ידי $n \cdot k$ הביטים הראשונים של רגיסטר t (ראו את החלק "גריידר לדוגמה" לפירוט).
- t חייב לקיים $0 \leq t \leq m - 1$.
- כל קריאה לפונקציה זו לא נחשבת למספר הפקודות שנוספו.

אחרי הוספת הפקודה האחרונה, על `construct_instructions` לחזור. התוכנית לאחר מכן מורצת על מספר טסטקייסים, שכל אחד מציין קלט המורכב מ-n מספרים שלמים של k -ביט $a[0], a[1], \ldots, a[n-1]$. הפתרון שלכם עובר טסטקייס נתון אם הפלט של התוכנית $c[0], c[1], \ldots, c[n-1]$ עבור הקלט הנתון עומד בדרישות הבאות:

- אם $s = 0$, $c[0]$ צריך להיות הערך הקטן ביותר מבין $a[0], a[1], \ldots, a[n-1]$.
- אם $s = 1$, לכל i ($0 \leq i \leq n-1$), $c[i]$ צריך להיות המספר ה-$1 + i$ הקטן ביותר מבין $a[0], a[1], \ldots, a[n-1]$.

הבדיקה של הפתרון שלכם יכולה לתת כל אחת מהשגיאות הבאות:

- `Invalid index`: אינדקס שגוי לרגיסטר (יתכן ששלילי) ניתן כפרמטר t , x או y לקריאה כלשהי של אחת מהפונקציות.
- `Value to store is not b bits long`: האורך של v שניתן ל-`append_store` לא שווה ל-b .
- `Invalid shift value`: הערך של p שינתן ל-`append_left` או ל-`append_right` הוא לא בין 0 ל-b , כולל.
- `Too many instructions`: הפונקציה שלכם ניסתה להוסיף מעל q פקודות.

דוגמאות

דוגמה 1

נניח כי $s = 0$, $n = 2$, $k = 1$, $q = 1000$. ישנם שני שלמים $a[0]$ ו-$a[1]$, לכל אחד $k = 1$ ביט. לפני הרצת התוכנית, $r[0][0] = a[0]$ ו-$r[0][1] = a[1]$. כל שאר הביטים במעבד נקבעים ל-0 . אחרי שכל הפקודות בתוכנית מבוצעות, צריך שיתקיים $c[0] = r[0][0] = \min(a[0], a[1])$, שזה המינימום של $a[0]$ ו-$a[1]$.

ישנם רק 4 קלטים אפשריים לתוכנית:

- מקרה 1 : $a[0] = 0, a[1] = 0$
- מקרה 2 : $a[0] = 0, a[1] = 1$
- מקרה 3 : $a[0] = 1, a[1] = 0$
- מקרה 4 : $a[0] = 1, a[1] = 1$

אני יכולים לראות שלכל 4 המקרים, $\min(a[0], a[1])$ שווה ל-bitwise-AND של $a[0]$ ו-$a[1]$. לכן, פתרון אפשרי הוא לבנות תוכנית על ידי ביצוע הקריאות הבאות:

1. `append_move(1, 0)`, שמוסיפה פקודה להעתקת $r[0]$ ל-$r[1]$.
2. `append_right(1, 1, 1)`, שמוסיפה פקודה שלוקחת את כל הביטים ב-$r[1]$, מזיזה אותם ימינה בביט 1 , ואז מאחסנת את התוצאה בחזרה ב-$r[1]$. משום שכל שלם הוא באורך 1 -ביט, התוצאה של זה היא ש-$r[1][0]$ יהיה שווה ל-$a[1]$.
3. `append_and(0, 0, 1)`, שמוסיפה פקודה לקחת את ה-bitwise-AND של $r[0]$ ו-$r[1]$, ולאחסן את התוצאה ב-$r[0]$. אחרי שפקודה זו מבוצעת, $r[0][0]$ נקבע להיות ה-bitwise-AND של $r[0][0]$ ושל $r[1][0]$, ששווה ל-bitwise-AND של $a[0]$ ושל $a[1]$, כמו שרצינו.

דוגמה 2

נניח כי $s = 1$, $n = 2$, $k = 1$, $q = 1000$. כמו בדוגמה הקודמת, יש רק 4 קלטים אפשריים לתוכנית. לכל 4 המקרים, $\min(a[0], a[1])$ הוא ה-bitwise-AND של $a[0]$ ו-$a[1]$, ו-$\max(a[0], a[1])$ הוא ה-bitwise-OR של $a[0]$ ו-$a[1]$. פתרון אפשרי הוא לבצע את הקריאות הבאות:

1. `append_move(1,0)`
2. `append_right(1,1,1)`
3. `append_and(2,0,1)`
4. `append_or(3,0,1)`
5. `append_left(3,3,1)`
6. `append_or(0,2,3)`

אחרי ביצוע פקודות אלו, $c[0] = r[0][0]$ מכילה את $\min(a[0], a[1])$, ו-$c[1] = r[0][1]$ מכיל את $\max(a[0], a[1])$, מה שממיין את הקלט.

מגבלות

- $m = 100$
- $b = 2000$
- $0 \leq s \leq 1$

- $2 \leq n \leq 100$
- $1 \leq k \leq 10$
- $q \leq 4000$
- $0 \leq a[i] \leq 2^k - 1$ (לכל $0 \leq i \leq n-1$)

תת משימות

1. (10 נקודות) $s = 0, n = 2, k \leq 2, q = 1000$
2. (11 נקודות) $s = 0, n = 2, k \leq 2, q = 20$
3. (12 נקודות) $s = 0, q = 4000$
4. (25 נקודות) $s = 0, q = 150$
5. (13 נקודות) $s = 1, n \leq 10, q = 4000$
6. (29 נקודות) $s = 1, q = 4000$

גריידר לדוגמה

הגריידר לדוגמה קורא את הקלט בפורמט הבא:

- שורה 1 : $s\ n\ k\ q$

לאחר מכן יש מספר כלשהו של שורות, כל אחת מתארת טסטקייס בודד. כל טסטקייס מיוצג בפורמט הבא:

- $a[0]\ a[1]\ \ldots\ a[n-1]$

ומתאר טסטקייס שהקלט שלו מורכב מ-n מספרים שלמים $a[0], a[1], ..., a[n-1]$. לאחר תיאור כל הטסטקייסים יש שורה בודדת המכילה רק -1.

הגריידר לדוגמה קורא תחילה ל-`construct_instructions(s, n, k, q)`. אם קריאה זו מפרה מגבלות מסוימות המתוארות בתיאור הבעיה, הגריידר לדוגמה מדפיס את אחת מהודעות השגיאה הרשומות בסוף החלק "פרטי מימוש" ויוצא. אחרת, הגריידר לדוגמה מדפיס תחילה כל אחת מהפקודות שהוספו על ידי `construct_instructions(s, n, k, q)` לפי הסדר. עבור פקודות $store$, הדפסת v מתבצעת מהאינדקס 0 לאינדקס $b-1$.

לאחר מכן, הגריידר לדוגמה מעבד את הטסטקייסים לפי הסדר. לכל טסטקייס, הוא מריץ את התוכנית שנבנתה על הקלט של הטסטקייס.

לכל פקודת $print(t)$, יהי $d[0], d[1], \ldots, d[n-1]$ רצף המספרים השלמים, כך שלכל i $(0 \leq i \leq n-1)$, $d[i]$ הוא הערך השלם של רצף הביטים $i \cdot k$ עד $(i+1) \cdot k - 1$ של רגיסטר t (כשהפקודה מתבצעת). הגריידר מדפיס רצף זה בפורמט הבא:

`register` t : $d[0]\ d[1]\ \ldots\ d[n-1]$

ברגע שכל הפקודות התבצעו, הגריידר לדוגמה מדפיס את הפלט של התוכנית.

אם $s = 0$, הפלט של הגריידר לדוגמה לכל טסטקייס הוא בפורמט הבא:

- $c[0]$.

אם $s = 1$, הפלט של הגריידר לדוגמה לכל טסטקייס הוא בפורמט הבא:

- $c[0]\ c[1]\ \ldots\ c[n-1]$.

לאחר ביצוע כל הטסטקייסים, הגריידר מדפיס X `:number of instructions` כאשר X הוא מספר הפקודות בתוכנית שלכם.