מאדים

זו עובדה ידועה שהפרעונים היו הראשונים להגיע לחלל החיצון. הם שיגרו את החללית הראשונה שנחתה על הפלנטה תחותמס 1 (המוכרת יותר כיום בשם מאדים). פני השטח של הפלנטה מתוארים כטבלה בגודל $(2n+1)\times(2n+1)$ של תאים ריבועיים כאשר בכל תא יש יבשה או מים. תכולת התא הממוקם בשורה ה-i ובעמודה ה-j $(0 \le i,j \le 2\cdot n)$ היא יבשה אם $s[i][j] =$ '1', ומים אם $s[i][j] =$ '0'.

נאמר ששני תאי יבשה מחוברים אם יש ביניהם מסלול של תאי יבשה כך שכל שני תאים עוקבים במסלול סמוכים בצלע אחד לשני. נגדיר אי על הפלנטה להיות קבוצה מקסימלית של תאי יבשה, כך שכל שני תאי יבשה בקבוצה מחוברים אחד לשני.

המשימה של החללית הייתה לספור את כמות האיים בפלנטה. אבל, המשימה לא הייתה קלה בגלל המחשב העתיק של החללית. למחשב היה זיכרון h שאיחסן מידע בצורת מערך דו מימדי בגודל $(2n+1)\times(2n+1)$ כאשר כל איבר במערך יכול לאחסן מחרוזת בינארית באורך 100 כשכל תו הוא או `'0'` (ערך ASCII 48) או `'1'` (ערך ASCII 49). תחילה, הביט הראשון של כל תא זכרון מאחסן את המצב של כל תא בטבלה, כלומר $h[i][j][0] = s[i][j]$ (לכל $0 \le i,j \le 2\cdot n$). כל שאר הביטים ב-h הם בהתחלה `'0'` (ערך ASCII 48).

על מנת לעבד את המידע המאוחסן בזיכרון, ביכולת המחשב לגשת לחלק בזיכרון בגודל 3×3 ולשכתב את הערך בתא השמאלי העליון בחלק זה. פורמלית, המחשב יכול לגשת לערכים ב-$h[i..i+2][j..j+2]$ $(0 \le i,j \le 2\cdot(n-1))$ ולשכתב את הערך בתא $h[i][j]$. תהליך זה יקרא בהמשך **עיבוד תא** (i,j).

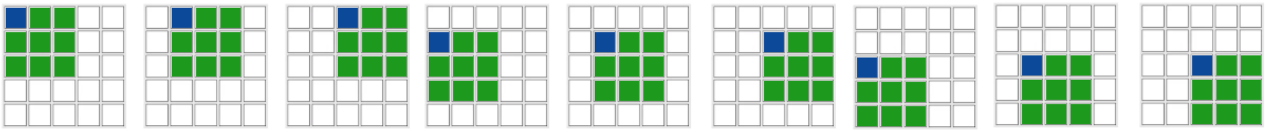
כדי להתמודד עם מגבלותיו של המחשב, הפרעונים פיתחו את השיטה הבאה:

- המחשב יעבד את המידע ב-n שלבים.
- בשלב k $(0 \le k \le n-1)$, נסמן $m = 2\cdot(n-k-1)$, המחשב יבצע עיבוד תא (i,j) עבור כל $0 \le i,j \le m$, בסדר עולה של i, ועבור כל i, בסדר עולה של j. במילים אחרות, המחשב יעבד את התאים בסדר הבא (משמאל לימין):

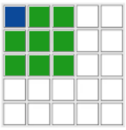
  $$(0,0),(0,1),\cdots,(0,m),(1,0),(1,1),\cdots,(1,m),\cdots,(m,0),(m,1),\cdots,(m,m).$$
- בשלב האחרון $(k=n-1)$, המחשב יעבד רק את תא $(0,0)$, ולאחר העיבוד, הערך שכתוב ב-$h[0][0]$ צריך להיות שווה לכמות האיים בפלנטה בבסיס בינארי, כשסיביות הערך הזוטרה (least significant bit) במספר מאוחסנת בתו הראשון במחרוזת.

הדיאגרמה להלן מראה איך המחשב מעבד זיכרון בגודל 5×5 $(n=2)$. התא הכחול הוא התא שהמחשב משכתב, והתאים הצבועים הם תת המערך שהמחשב מעבד כרגע.

במהלך שלב 0, המחשב יעבד את תתי המערכים הבאים בסדר הבא (משמאל לימין):



במהלך שלב 1, המחשב יעבד תת מערך יחיד:



המשימה שלכם היא לממש פונקציה שתאפשר למחשב לחשב את כמות האיים בפלנטה תחותמס 1, בהתחשב בדרך בה הוא פועל.

פרטי מימוש

עליכם לממש את הפונקציה הבאה:

```
string process(string[][] a, int i, int j, int k, int n)
```

- a: מערך בגודל 3×3 המייצג את תת המערך שמעבדים כרגע. ספציפית, $a = h[i..i+2][j..j+2]$, כשכל איבר ב-a הוא מחרוזת באורך 100 בדיוק וכל תו יהיה או `'0'` (ערך ASCII 48) או `'1'` (ערך ASCII 49).
- i, j: מספר השורה והעמודה של התא שהמחשב מעבד כרגע.
- k: מספר השלב הנוכחי.
- n: מספר השלבים הכולל, ומימדי פני השטח של הפלנטה הבנויה מ-$(2n+1) \times (2n+1)$ תאים.
- על הפונקציה להחזיר מחרוזת בינארית באורך 100. הערך המוחזר יהיה מאוחסן בזיכרון המחשב, בתא $h[i][j]$.
- הקריאה האחרונה לפונקציה זו תתרחש כאשר $k = n - 1$. בקריאה זו, על הפונקציה להחזיר את כמות האיים על הפלנטה בייצוג בינארי כשסיבית הערך הזוטרה (least significant bit) מיוצגת על ידי הערך באינדקס 0 (התו הראשון במחרוזת) והסיבית השניה מהסוף באינדקס 1 וכן הלאה.
- הפונקציה חייבת לא להיות תלויה באף משתנה סטטי או גלובלי, וערך ההחזרה שלה צריך להיות תלוי אך ורק בפרמטרים המועברים אליה.

כל טסטקייס כולל T תרחישים שונים (כלומר, פני שטח שונים של פלנטות). ההתנהגות של הפונקציה לכל תרחיש צריכה להיות בלתי תלויה בסדר של התרחישים, מכיוון שקריאות ל-`process` עבור כל תרחיש לא בהכרח יהיו רצופות. עם זאת, מובטח שעבור כל תרחיש, הקריאות ל-`process` יהיו בסדר המתואר בסטייטמנט.

בנוסף, לכל טסטקייס, מספר מופעים של התכנית שלכם יתחילו במקביל. מגבלות הזיכרון וזמן המעבד הן על כל מופעי התכנית יחדיו. כל ניסיון מכוון להעביר מידע מחוץ לרשת בין מופעים אלו נחשב כרמאות ועלול לגרום לפסילה מהתחרות.

בפרט, כל מידע שנשמר למשתנה סטטי או גלובלי במהלך קריאה ל-`process` לא מובטח להיות זמין בקריאות הבאות לפונקציה.

מגבלות

- $1 \le T \le 10$
- $1 \le n \le 20$
- $s[i][j]$ הוא או '`0`'(ערך ASCII 48) או '`1`'(ערך ASCII 49) (לכל $0 \le i, j \le 2 \cdot n$)
- האורך של $h[i][j]$ הוא 100 בדיוק (לכל $0 \le i, j \le 2 \cdot n$)
- כל תו של $h[i][j]$ הוא או `'0'` (ערך ASCII 48) או `'1'` (ערך ASCII 49) (לכל $0 \le i, j \le 2 \cdot n$)

עבור כל קריאה לפונקציה `process`:

- $0 \leq k \leq n-1$
- $0 \leq i,j \leq 2 \cdot (n-k-1)$

תתי משימות

1. (6 נקודות) $n \leq 2$
2. (8 נקודות) $n \leq 4$
3. (7 נקודות) $n \leq 6$
4. (8 נקודות) $n \leq 8$
5. (7 נקודות) $n \leq 10$
6. (8 נקודות) $n \leq 12$
7. (10 נקודות) $n \leq 14$
8. (24 נקודות) $n \leq 16$
9. (11 נקודות) $n \leq 18$
10. (11 נקודות) $n \leq 20$

דוגמאות

דוגמה 1

הביטו על המקרה שבו $n=1$ ו-s היא כדלקמן:

```
'1' '0' '0'
'1' '1' '0'
'0' '0' '1'
```

בדוגמה זו, פני השטח של הפלנטה מורכבים מ-3×3 תאים ומ-2 איים. יהיה רק שלב אחד של קריאות לפונקציה `process`.

במהלך שלב 0, הגריידר יקרא לפונקציה `process` בדיוק פעם אחת:

```
process([["100","000","000"],["100","100","000"],["000","000","100"]],0,0,0,1)
```

שימו לב שרק 3 הביטים הראשונים של כל תא ב-h מוצגים.

על פונקציה זו להחזיר "`0100...`" (הביטים המושמטים כולם אפס), כאשר $\ldots0010$ בבסיס בינארי שווה ל-2 בבסיס עשרוני. שימו לב שיש 96 אפסים שהושמטו והוחלפו ב-`...`.

דוגמה 2

הביטו על המקרה שבו $n=2$ ו-s היא כדלהלן:

```
'1' '1' '0' '1' '1'
'1' '1' '0' '0' '0'
'1' '0' '1' '1' '1'
'0' '1' '0' '0' '0'
'0' '1' '1' '1' '1'
```

בדוגמה זו, פני השטח של הפלנטה מורכבים מ-5×5 תאים ומ-4 איים. יהיו 2 שלבים של קריאות לפונקציה `process`.

במהלך שלב 0, הגריידר יקרא לפונקציה `process` 9 פעמים:

```
process([["100","100","000"],["100","100","000"],["100","000","100"]],0,0,0,2)
process([["100","000","100"],["100","000","000"],["000","100","100"]],0,1,0,2)
process([["000","100","100"],["000","000","000"],["100","100","100"]],0,2,0,2)
process([["100","100","000"],["100","000","100"],["000","100","000"]],1,0,0,2)
process([["100","000","000"],["000","100","100"],["100","000","000"]],1,1,0,2)
process([["000","000","000"],["100","100","100"],["000","000","000"]],1,2,0,2)
process([["100","000","100"],["000","100","000"],["000","100","100"]],2,0,0,2)
process([["000","100","100"],["100","000","000"],["100","100","100"]],2,1,0,2)
process([["100","100","100"],["000","000","000"],["100","100","100"]],2,2,0,2)
```

נניח שהקריאות לעיל החזירו את הערכים `"011"`, `"000"`, `"000"`, `"111"`, `"111"`, `"011"`, `"110"`, `"010"`, `"111"` בהתאמה כשהביטים המושמטים כולם אפס. אז, לאחר ששלב 0 הסתיים, h יאחסן את הערכים הבאים:

```
"011", "000", "000", "100", "100"
"111", "111", "011", "000", "000"
"110", "010", "111", "100", "100"
"000", "100", "000", "000", "000"
"000", "100", "100", "100", "100"
```

במהלך שלב 1, הגריידר יקרא לפונקציה `process` פעם אחת:

```
process([["011","000","000"],["111","111","011"],["110","010","111"]],0,0,1,2)
```

לבסוף, על קריאה זו להחזיר `"0010000...."` (כשהביטים המושמטים כולם אפס), כש-$....0000100$ בבסיס בינארי זה 4 בעשרוני. שימו לב שיש 93 אפסים שהושמטו והוחלפו ב-`....`.

גריידר לדוגמה

הגריידר לדוגמה קורא את הקלט בפורמט הבא:

- שורה 1: T
- בלוק i $(0 \le i \le T-1)$: בלוק המתאר את תרחיש i.
    - שורה 1: n
    - שורה $2+j$ $(0 \le j \le 2 \cdot n)$: $s[j][0]\ s[j][1]\ \ldots\ s[j][2 \cdot n]$

הגריידר לדוגמה מדפיס את התוצאה בפורמט הבא:

- שורה $1+i$ $(0 \le i \le T-1)$: ערך ההחזרה האחרון של הפונקציה `process` עבור התרחיש ה-i בבסיס עשרוני.