ดาวอังคาร (Mars)

เป็นที่ทราบกันทั่วไปว่าพวกฟาโรห์เป็นคนกลุ่มแรกที่ออกนอกโลกไปสัมผัสอวกาศ พวกเขาได้ส่งยานอวกาศลำแรกไปเยือนดาวเคราะห์ Thutmus I (ที่รู้จักกันในปัจจุบันคือดาวอังคาร) พื้นผิวของดาวดังกล่าวสามารถแทนได้เป็นตารางกริดของช่องจตุรัสขนาด $(2n+1) \times (2n+1)$ ที่แต่ละช่องเป็นพื้นดินหรือพื้นน้ำ สถานะของช่องที่แถวที่ i คอลัมน์ j (สำหรับ $0 \le i, j \le 2 \cdot n$) ระบุด้วย $s[i][j] =$ `'1'` ถ้าเป็นพื้นดิน และ $s[i][j] =$ `'0'` ถ้าเป็นน้ำ

เราจะเรียกช่องพื้นดินสองช่องว่าเชื่อมต่อกันถ้ามีเส้นทางที่ประกอบด้วยช่องที่เป็นพื้นดินระหว่างช่องทั้งสองโดยที่ทุกช่องที่มีลำดับต่อกันในเส้นทางมีการใช้ขอบร่วมกัน จะนิยามให้เกาะบนดาวเคราะห์ดังกล่าวเป็นเซตของช่องที่มีไม่สามารถเพิ่มได้อีก (maximal) ที่ทุก ๆ คู่ของช่องในเซตนั้นเชื่อมต่อกัน

เป้าหมายของยานอวกาศคือการนับจำนวนเกาะบนดาวเคราะห์ดังกล่าว อย่างไรก็ตาม งานดังกล่าวไม่สามารถกระทำได้โดยง่ายเพราะความโบราณของคอมพิวเตอร์ที่มีบนยาน คอมพิวเตอร์มีหน่วยความจำ h ที่เก็บข้อมูลในรูปของอาร์เรย์สองมิติขนาด $(2n+1) \times (2n+1)$ ที่แต่ละช่องสามารถเก็บค่าเป็นสตริงฐานสองความยาว 100 ตัวอักษร โดยที่แต่ละตัวอักษรจะเป็น `'0'` (รหัส ASCII 48) หรือ `'1'` (รหัส ASCII 49) เมื่อเริ่มต้นบิตแรกของแต่ละช่องในหน่วยความจำจะเก็บสถานะของแต่ละช่องในตารางกริด นั่นคือ $h[i][j][0] = s[i][j]$ (สำหรับทุก ๆ ค่า $0 \le i, j \le 2 \cdot n$) สำหรับบิตอื่นจะมีค่าเริ่มต้นเป็น `'0'` (รหัส ASCII 48)

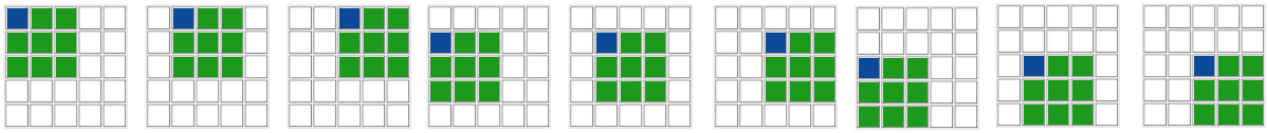
ในการประมวลผลข้อมูลในหน่วยความจำ คอมพิวเตอร์จะสามารถเข้าถึงข้อมูลได้ทีละส่วนย่อยขนาด 3×3 ของหน่วยความจำและสามารถเขียนค่าคงไปในช่องบนซ้ายของส่วนย่อยนั้น กล่าวอย่างเป็นทางการก็คือคอมพิวเตอร์จะสามารถอ่านค่าที่ตำแหน่ง $h[i..i+2][j..j+2]$ $(0 \le i, j \le 2 \cdot (n-1))$ และเขียนค่าลงไปที่ $h[i][j]$ กระบวนการดังกล่าวจะเรียกว่าเป็น **การประมวลผลช่อง** (i, j)

เพื่อที่จะจัดการกับข้อจำกัดของคอมพิวเตอร์ เหล่าฟาโรห์จึงได้คิดค้นกระบวนการดังต่อไปนี้:

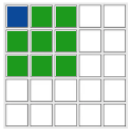
- คอมพิวเตอร์จะประมวลผลหน่วยความจำเป็นรอบ จำนวน n รอบ
- ในรอบที่ k $(0 \le k \le n-1)$ กำหนดให้ $m = 2 \cdot (n-k-1)$ คอมพิวเตอร์จะประมวลผลช่อง (i, j) สำหรับทุก ๆ ค่า $0 \le i, j \le m$ ตามลำดับที่เพิ่มขึ้นของ i และสำหรับแต่ละค่า i ตามลำดับที่เพิ่มขึ้นของ j กล่าวอีกทางหนึ่งก็คือคอมพิวเตอร์จะประมวลผลช่องในลำดับดังนี้ : $(0,0), (0,1), \cdots, (0,m)$, $(1,0), (1,1), \cdots, (1,m), \cdots, (m,0), (m,1), \cdots, (m,m)$.
- ในรอบสุดท้าย ($k = n-1$) คอมพิวเตอร์จะประมวลผลเฉพาะช่อง $(0,0)$ หลังจากนั้นค่าที่เขียนที่ $h[0][0]$ จะต้องเท่ากับจำนวนของเกาะบนดาวเคราะห์นี้ ในรูปของสตริงฐานสองที่หลักที่มีค่านัยสำคัญน้อยที่สุดจะอยู่ที่อักษรตัวแรกของสตริง

แผนภาพด้านล่างแสดงการที่คอมพิวเตอร์ประมวลผลหน่วยความจำที่มีขนาด 5×5 ($n = 2$) ช่องสีน้ำเงินจะแสดงช่องที่จะถูกเขียนทับ ส่วนช่องที่มีสีแสดงส่วนย่อยของอาร์เรย์ที่กำลังถูกประมวลผล

ในรอบที่ 0 คอมพิวเตอร์จะประมวลผลอาร์เรย์ย่อยตามลำดับต่อไปนี้:



ในรอบที่ 1 คอมพิวเตอร์จะประมวลผลอาร์เรย์ย่อยแค่ช่องเดียว:



งานของคุณคือการเขียนฟังก์ชันที่ทำให้คอมพิวเตอร์สามารถนับจำนวนเกาะบนดาวเคราะห์ Thutmus I

รายละเอียดการเขียนโปรแกรม

คุณจะต้องเขียนฟังก์ชันต่อไปนี้:

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

- a: อาร์เรย์ขนาด 3×3 ที่ระบุส่วนย่อยของอาร์เรย์ที่กำลังประมวลผล นั่นคือ $a = h[i..i+2][j..j+2]$ โดยที่แต่ละค่าในอาร์เรย์เป็นสตริงที่มีความยาวเท่ากับ 100 ตัวอักษรพอดี และทุกตัวอักษรจะมีค่าเป็น `'0'` (รหัส ASCII 48) หรือ `'1'` (รหัส ASCII 49)
- i, j: หมายเลขแถวและคอลัมน์ของช่องที่คอมพิวเตอร์กำลังประมวลผล
- k: หมายเลขรอบ
- n: จำนวนรอบทั้งหมด และพื้นผิวดาวเคราะห์จะประกอบไปด้วย $(2n+1) \times (2n+1)$ ช่อง
- ฟังก์ชันนี้จะต้องคืนค่าสตริงฐานสองที่มีความยาวเท่ากับ 100 ตัวอักษร ค่าที่คืนจะถูกเก็บในหน่วยความจำของคอมพิวเตอร์ช่องที่ $h[i][j]$
- การเรียกในครั้งสุดท้ายจะเกิดเมื่อ $k = n - 1$ ในการเรียกครั้งนี้ฟังก์ชันจะต้องคืนจำนวนเกาะบนดาวเคราะห์ดังกล่าวในรูปของสตริงที่หลักที่มีนัยยะสำคัญน้อยที่สุดจะอยู่ที่ตัวอักษรที่มีดัชนีเป็น 0 (อักษรแรกบนสตริง) และหลักที่สำคัญน้อยที่สุดรองลงไปเป็นตัวอักษรที่ดัชนีเท่ากับ 1 ต่อไปเรื่อย ๆ
- ฟังก์ชันจะต้องเป็นอิสระจากตัวแปร static และตัวแปร global ทั้งหมด ค่าที่คืนกลับมาจะต้องขึ้นกับค่าในพารามิเตอร์ที่ส่งให้เท่านั้น

แต่ละกรณีทดสอบประกอบด้วยสถานการณ์ T แบบที่เป็นอิสระต่อกัน (นั่นคืออยู่บนพื้นผิวที่แตกต่างกันของดาวเคราะห์) พฤติกรรมของฟังก์ชันของคุณสำหรับแต่ละสถานการณ์จะต้องไม่ขึ้นกับลำดับของสถานการณ์ เนื่องจากช่องที่จะได้รับการประมวลผลของแต่ละสถานการณ์อาจจะไม่ได้เกิดขึ้นในลำดับต่อกัน อย่างไรก็ตามรับประกันว่าในแต่ละสถานการณ์ การเรียกฟังก์ชัน `process` จะเกิดในลำดับตามที่ระบุไว้ในรายละเอียดข้างต้น

นอกจากนี้ สำหรับแต่ละกรณีทดสอบ อาจจะมีการเรียกใช้โปรแกรมของคุณพร้อม ๆ กันหลายการเรียกใช้ (instance) ขีดจำกัดของหน่วยความจำและเวลาการทำงานจะคิดสำหรับทุกการเรียกใช้รวมกัน ความพยายามที่จะส่งข้อมูลข้ามไปมาระหว่างหลาย ๆ การเรียกใช้จะถูกจัดว่าเป็นความพยายามที่จะโกงและจะทำให้คุณถูกเพิกถอนสิทธิ์การแข่งขัน (disqualification)

นอกจากนี้ จะไม่มีการรับประกันว่าข้อมูลที่ถูกเก็บไว้ในตัวแปรแบบ static หรือตัวแปร global ระหว่างการเรียกใช้ฟังก์ชัน `process` จะสามารถใช้ได้ในการเรียกใช้ฟังก์ชันนี้ในรอบถัด ๆ ไป

เงื่อนไข

- $1 \le T \le 10$
- $1 \le n \le 20$
- $s[i][j]$ จะมีค่าเป็น `'0'` (รหัส ASCII 48) หรือ `'1'` (รหัส ASCII 49) (สำหรับทุก ๆ $0 \le i, j \le 2 \cdot n$)
- ความยาวของ $h[i][j]$ จะเท่ากับ 100 (สำหรับทุก ๆ $0 \le i, j \le 2 \cdot n$)
- แต่ละตัวอักษรใน $h[i][j]$ จะเป็น `'0'` (ASCII 48) หรือ `'1'` (ASCII 49) (สำหรับทุก ๆ $0 \le i, j \le 2 \cdot n$)

สำหรับแต่ละการเรียกใช้ฟังก์ชัน `process`

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

ปัญหาย่อย

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

ตัวอย่าง

ตัวอย่าง 1

พิจารณาตัวอย่างที่ $n = 1$ และ s เป็นดังนี้:

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

ในตัวอย่างนี้ พื้นผิวของดาวเคราะห์ประกอบไปด้วยช่องจำนวน 3×3 ช่อง และมีเกาะจำนวน 2 เกาะ จะมีการเรียกฟังก์ชัน `process` จำนวนแค่ 1 รอบ

ในรอบที่ 0 เกรดเดอร์จะเรียกฟังก์ชัน `process` หนึ่งครั้งเท่านั้น:

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

สังเกตว่าเราแสดงแค่สามบิตแรกของทุกช่องใน h

ฟังก์ชันนี้ควรคืนค่า `"0100..."` (บิตที่ละไว้คือศูนย์) โดยที่ $\ldots0010$ ในฐานสองจะมีค่าเท่ากับ 2 ในเลขฐานสิบ สังเกตว่ามีศูนย์ตามมาต่ออีก 96 ตัว ซึ่งเราได้ละไว้และเขียนแทนด้วย `...`

ตัวอย่าง 2

พิจารณากรณีที่ $n = 2$ และ s เป็นดังนี้:

```
'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'
```

ในตัวอย่างนี้ พื้นผิวดาวเคราะห์ประกอบด้วยช่องจำนวน 5×5 ช่อง และมีเกาะจำนวน 4 เกาะ จะมีการเรียกใช้ฟังก์ชัน `process` จำนวน 2 รอบ

ในรอบที่ 0 เกรดเดอร์จะเรียกฟังก์ชัน `process` จำนวน 9 ครั้ง ดังนี้

```
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)
```

สมมติว่าการเรียกฟังก์ชันด้านบนคืนค่า `"011"`, `"000"`, `"000"`, `"111"`, `"111"`, `"011"`, `"110"`, `"010"`, `"111"` ตามลำดับ โดยที่บิตที่เราทิ้งไปมีค่าเป็นศูนย์ ดังนั้นเมื่อการทำงานรอบที่ 0 สิ้นสุดลง หน่วยความจำ h จะเก็บค่าดังนี้:

```
"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"
```

ในรอบที่ 1 เกรดเดอร์จะเรียกฟังก์ชัน `process` หนึ่งครั้งเท่านั้น:

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

ฟังก์ชันนี้ควรคืนค่า `"0010000..."` (บิตที่ละไว้คือศูนย์) โดยที่ $\ldots00100$ ในฐานสองจะมีค่าเท่ากับ 4 ในเลขฐานสิบ สังเกตว่ามีศูนย์ตามมาต่ออีก 93 ตัว ซึ่งเราได้ละไว้และเขียนแทนด้วย `...`

เกรดเดอร์ตัวอย่าง

เกรดเดอร์ตัวอย่างอ่านข้อมูลนำเข้าในรูปแบบต่อไปนี้:

- บรรทัดที่ 1: T

- ข้อมูลส่วนที่ i ($0 \le i \le T-1$): ข้อมูลส่วนที่แทนสถานการณ์ i.
  - บรรทัดที่ 1: n
  - บรรทัดที่ $2+j$ ($0 \le j \le 2 \cdot n$): $s[j][0]\ s[j][1]\ \ldots\ s[j][2 \cdot n]$

เกรดเดอร์ตัวอย่างจะพิมพ์ผลลัพธ์ในรูปแบบต่อไปนี้:

- บรรทัดที่ $1+i$ ($0 \le i \le T-1$): ค่าสุดท้ายที่คืนจากฟังก์ชัน `process` ในสถานการณ์ที่ i