เรจิสเตอร์แบบเลื่อนบิต

คริสโตเฟอร์ซึ่งเป็นวิศวกรกำลังออกแบบโปรเซสเซอร์แบบใหม่อยู่

โปรเซสเซอร์นี้สามารถเข้าถึงหน่วยความจำได้ m ตำแหน่ง และแต่ละตำแหน่งมีขนาด b บิต (โดยที่ $m = 100$ และ $b = 2000$) หน่วยความจำเหล่านี้ถูกเรียกว่า**เรจิสเตอร์**ซึ่งถูกกำกับด้วยหมายเลข 0 ถึง $m - 1$ ให้ $r[0], r[1], \ldots, r[m-1]$ ระบุถึงเรจิสเตอร์เหล่านี้ เรจิสเตอร์แต่ละตัวเป็นอาร์เรย์ของ b บิต กำกับด้วยหมายเลข 0 (บิตขวาสุด) ถึง $b - 1$ (บิตซ้ายสุด) สำหรับแต่ละ i (โดยที่ $0 \le i \le m - 1$) และแต่ละ j (โดยที่ $0 \le j \le b - 1$) เราจะระบุถึงบิตที่ j ของเรจิสเตอร์ตัวที่ i ด้วย $r[i][j]$

สำหรับลำดับของบิต $d_0, d_1, \ldots, d_{l-1}$ ใด ๆ (ซึ่งมีความยาวเป็น l) **ค่าจำนวนเต็ม**ของลำดับดังกล่าวคือ $2^0 \cdot d_0 + 2^1 \cdot d_1 + \ldots + 2^{l-1} \cdot d_{l-1}$ กำหนดให้**ค่าจำนวนเต็มที่เก็บในเรจิสเตอร์** i คือค่าจำนวนเต็มของลำดับของบิตในเรจิสเตอร์นั้น ซึ่งคือ $2^0 \cdot r[i][0] + 2^1 \cdot r[i][1] + \ldots + 2^{b-1} \cdot r[i][b-1]$

โปรเซสเซอร์นี้มี**คำสั่ง**อยู่ 9 คำสั่งซึ่งสามารถใช้ในการเปลี่ยนแปลงบิตในเรจิสเตอร์เหล่านี้ได้ แต่ละคำสั่งจะดำเนินการกับเรจิสเตอร์หนึ่งตัวหรือมากกว่าและจะเก็บผลลัพธ์ไว้ในเรจิสเตอร์หนึ่งตัว ต่อไปนี้เราจะใช้ $x := y$ เพื่อระบุถึงทำงานที่เป็นการเปลี่ยนค่าของ x ให้มีค่าเท่ากับ y การทำงานของคำสั่งแต่ละคำสั่งเป็นดังต่อไปนี้

- $move(t, y)$: คัดลอกอาร์เรย์ของบิตในเรจิสเตอร์ y ไปยัง เรจิสเตอร์ t กล่าวคือจะตั้งค่า $r[t][j] := r[y][j]$ สำหรับแต่ละ j (โดยที่ $0 \le j \le b - 1$)
- $store(t, v)$: ตั้งค่าเรจิสเตอร์ t ให้เป็น v โดยที่ v เป็นอาร์เรย์ของ b บิต กล่าวคือจะตั้งค่า $r[t][j] := v[j]$ สำหรับแต่ละ j (โดยที่ $0 \le j \le b - 1$)
- $and(t, x, y)$: คำนวณผลการ AND รายบิตของเรจิสเตอร์ x และ y และเก็บผลลัพธ์ไว้ในเรจิสเตอร์ t กล่าวคือสำหรับแต่ละ j (โดยที่ $0 \le j \le b - 1$) ให้ตั้งค่า $r[t][j] := 1$ ถ้า **ทั้ง** $r[x][j]$ และ $r[y][j]$ เป็น 1 และตั้งค่า $r[t][j] := 0$ ในกรณีอื่น ๆ
- $or(t, x, y)$: คำนวณผลการ OR รายบิตของเรจิสเตอร์ x และ y แล้วเก็บผลลัพธ์ไว้ในเรจิสเตอร์ t กล่าวคือสำหรับแต่ละ j (โดยที่ $0 \le j \le b - 1$) ให้ตั้งค่า $r[t][j] := 1$ ถ้า **อย่างน้อยหนึ่งตัว** ใน $r[x][j]$ หรือ $r[y][j]$ เป็น 1 และตั้งค่า $r[t][j] := 0$ ในกรณีอื่น ๆ
- $xor(t, x, y)$: คำนวณผลการ XOR รายบิตของเรจิสเตอร์ x และ y แล้วเก็บผลลัพธ์ไว้ในเรจิสเตอร์ t กล่าวคือสำหรับแต่ละ j (โดยที่ $0 \le j \le b - 1$) ให้ตั้งค่า $r[t][j] := 1$ ถ้า **ตัวใดตัวหนึ่งเท่านั้น** ใน $r[x][j]$ หรือ $r[y][j]$ เป็น 1 และตั้งค่า $r[t][j] := 0$ ในกรณีอื่น ๆ
- $not(t, x)$: คำนวณผลการ NOT รายบิตของเรจิสเตอร์ x แล้วเก็บผลลัพธ์ไว้ในเรจิสเตอร์ t กล่าวคือสำหรับแต่ละ j (โดยที่ $0 \le j \le b - 1$) นั้น ให้ตั้งค่า $r[t][j] := 1 - r[x][j]$
- $left(t, x, p)$: เลื่อนบิตทั้งหมดในเรจิสเตอร์ x ไปทางซ้าย p ตำแหน่ง แล้วเก็บผลลัพธ์ไว้ในเรจิสเตอร์ t ผลของการเลื่อนบิตของเรจิสเตอร์ x ไปทางซ้ายไป p ตำแหน่ง เป็นอาร์เรย์ v ซึ่งประกอบด้วย b บิต

กล่าวคือสำหรับแต่ละ j (โดยที่ $0 \leq j \leq b-1$) นั้น $v[j] = r[x][j-p]$ ถ้า $j \geq p$ และ $v[j] = 0$ ในกรณีอื่น ๆ สำหรับแต่ละ j (โดยที่ $0 \leq j \leq b-1$) นั้น ให้ตั้งค่า $r[t][j] := v[j]$.

- $right(t, x, p)$: เลื่อนบิตทั้งหมดในเรจิสเตอร์ x ไปทางขวา p ตำแหน่ง แล้วเก็บผลลัพธ์ไว้ในเรจิสเตอร์ t ผลของการเลื่อนบิตของเรจิสเตอร์ x ไปทางขวา p ตำแหน่ง เป็นอาร์เรย์ v ซึ่งประกอบด้วย b บิต กล่าวคือสำหรับแต่ละ j (โดยที่ $0 \leq j \leq b-1$) นั้น $v[j] = r[x][j+p]$ ถ้า $j \leq b-1-p$ และ $v[j] = 0$ ในกรณีอื่น ๆ สำหรับแต่ละ j (โดยที่ $0 \leq j \leq b-1$) นั้น ให้ตั้งค่า $r[t][j] := v[j]$.

- $add(t, x, y)$: บวกค่าจำนวนเต็มของเรจิสเตอร์ x และเรจิสเตอร์ y แล้วเก็บผลลัพธ์ไว้ในเรจิสเตอร์ t การบวกนี้ทำแบบ modulo 2^b กล่าวคือ ให้ X เป็นค่าจำนวนเต็มที่เก็บอยู่ในเรจิสเตอร์ x และ ให้ Y เป็นค่าจำนวนเต็มที่เก็บอยู่ในเรจิสเตอร์ y ก่อนการทำงาน ให้ T เป็นค่าจำนวนเต็มที่เก็บในเรจิสเตอร์ t หลังการทำาน ถ้า $X + Y < 2^b$ ให้ตั้งค่าบิตของ t ให้ $T = X + Y$ ถ้าไม่เช่นนั้นให้ตั้งค่าบิตของ t ให้ $T = X + Y - 2^b$

คริสโตเฟอร์ต้องการให้คุณแก้ปัญหาของงานสองประเภทโดยใช้โปรเซสเซอร์ใหม่นี้ ประเภทของงานระบุด้วยจำนวนเต็ม s ในงานทั้งสองประเภทนี้คุณจะต้องสร้าง**โปรแกรม**ซึ่งเป็นลำดับของคำสั่งที่ระบุไว้ข้างต้นนี้

ข้อมูลนำเข้า ของโปรแกรมประกอบด้วยจำนวนเต็ม n ตัวได้แก่ $a[0], a[1], \ldots, a[n-1]$ โดยที่แต่ละตัวมี k บิต กล่าวคือ $a[i] < 2^k$ (โดยที่ $0 \leq i \leq n-1$) ก่อนที่โปรแกรมนี้จะทำงาน ข้อมูลนำเข้าตามลำดับทั้งหมดจะถูกเก็บไว้ในเรจิสเตอร์ 0 โดยที่สำหรับ i (โดยที่ $0 \leq i \leq n-1$) ค่าจำนวนเต็มของลำดับของ k บิต $r[0][i \cdot k], r[0][i \cdot k + 1], \ldots, r[0][(i+1) \cdot k - 1]$ คือ $a[i]$ ให้ถือว่า $n \cdot k \leq b$ บิตอื่น ๆ ในเรจิสเตอร์ 0 (ซึ่งคือบิตที่มีหมายเลขอยู่ระหว่าง $n \cdot k$ และ $b-1$ รวมหัวท้าย) และบิตใด ๆ ในเรจิสเตอร์อื่น ๆ มีค่าเริ่มต้นเป็น 0

การทำงานของโปรแกรมประกอบด้วยการทำงานแต่ละคำสั่งของโปรแกรมตามลำดับ หลังจากการทำงานของคำสั่งสุดท้าย **ข้อมูลส่งออก** ของโปรแกรมจะถูกคำนวณจากค่าสุดท้ายที่อยู่ในเรจิสเตอร์ 0 กล่าวคือ ข้อมูลส่งออกเป็นลำดับของจำนวนเต็ม n ตัว $c[0], c[1], \ldots, c[n-1]$ โดยที่สำหรับแต่ละ i (โดยที่ $0 \leq i \leq n-1$) นั้น $c[i]$ คือค่าจำนวนเต็มของลำดับที่ประกอบด้วยบิต $i \cdot k$ ถึง $(i+1) \cdot k - 1$ ของเรจิสเตอร์ 0 กำหนดให้หลังจากการทำงานของโปรแกรมนั้น บิตอื่น ๆ ของเรจิสเตอร์ 0 (ซึ่งคือบิตที่กำกับด้วยหมายเลข $n \cdot k$ เป็นอย่างน้อย) และบิตใด ๆ ในเรจิสเตอร์อื่น ๆ สามารถเป็นค่าใด ๆ ก็ได้

- งานประเภทแรก $(s = 0)$ คือการหาค่าจำนวนเต็มน้อยสุดในบรรดาข้อมูลนำเข้าจำนวนเต็ม $a[0], a[1], \ldots, a[n-1]$ กล่าวคือ $c[0]$ จะต้องเป็นค่าที่น้อยที่สุดระหว่าง $a[0], a[1], \ldots, a[n-1]$ ค่าของ $c[1], c[2], \ldots, c[n-1]$ สามารถเป็นค่าใด ๆ ก็ได้

- งานประเภทที่ 2 $(s = 1)$ คือการเรียงข้อมูลจำนวนเต็ม $a[0], a[1], \ldots, a[n-1]$ ให้เป็นลำดับไม่ลด กล่าวคือ สำหรับแต่ละ i (โดยที่ $0 \leq i \leq n-1$) นั้น $c[i]$ ควรจะเป็นตัวเลขจำนวนเต็มที่มีค่าน้อยเป็นลำดับที่ $1 + i$ ในบรรดา $a[0], a[1], \ldots, a[n-1]$ (กล่าวอีกนัยหนึ่งคือ $c[0]$ เป็นจำนวนเต็มที่มีค่าน้อยสุดของข้อมูลนำเข้าจำนวนเต็มทั้งหมด)

สร้างโปรแกรมที่มีจำนวนคำสั่งไม่เกิน q คำสั่ง ให้กับคริสโตเฟอร์เพื่อทำงานข้างต้นนี้

รายละเอียดการเขียนโปรแกรม

คุณจะต้องเขียนฟังก์ชันต่อไปนี้:

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

- s: ประเภทของงาน
- n: จำนวนของจำนวนเต็มในข้อมูลนำเข้า
- k: จำนวนบิตของแต่ละจำนวนเต็มที่นำเข้า
- q: จำนวนของคำสั่งมากที่สุดที่อนุญาต
- ฟังก์ชันนี้จะถูกเรียกใช้เพียงครั้งเดียวและต้องสร้างลำดับของคำสั่งที่ทำงานตามที่ต้องการ

ฟังก์ชันนี้สามารถเรียกฟังก์ชันด้านล่างหนึ่งหรือหลายครั้งเพื่อสร้างลำดับของคำสั่ง:

```
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)
```

- แต่ละฟังก์ชันจะเพิ่มคำสั่งต่อไปนี้ต่อท้ายโปรแกรม $move(t, y)$, $store(t, v)$, $and(t, x, y)$, $or(t, x, y)$, $xor(t, x, y)$, $not(t, x)$, $left(t, x, p)$, $right(t, x, p)$ หรือ $add(t, x, y)$ ตามลำดับ
- สำหรับทุกคำสั่งที่เกี่ยวข้อง t, x, y จะต้องมีค่าอย่างน้อย 0 และไม่เกิน $m - 1$
- สำหรับทุกคำสั่งที่เกี่ยวข้อง t, x, y ไม่จำเป็นต้องมีค่าที่แตกต่างกัน
- สำหรับคำสั่ง $left$ และ $right$ p ต้องมีค่าอย่างน้อย 0 และไม่เกิน b
- สำหรับคำสั่ง $store$ ความยาวของ v จะต้องเท่ากับ b

คุณอาจจะเรียกฟังก์ชันด้านล่างเพื่อช่วยในการทดสอบคำตอบของคุณ

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

- การเรียกฟังก์ชันดังกล่าวในระหว่างตรวจโปรแกรมของคุณจะไม่เกิดผลอะไรต่อการตรวจ
- ในเกรดเดอร์ตัวอย่าง ฟังก์ชันนี้จะเพิ่มคำสั่ง $print(t)$ ต่อท้ายโปรแกรม
- เมื่อเกรดเดอร์ตัวอย่างพบคำสั่ง $print(t)$ ระหว่างการทำงาน จะพิมพ์จำนวนเต็มขนาด k บิต จำนวน n จำนวน ที่เกิดจาก $n \cdot k$ บิตแรกของเรจิสเตอร์ t (ดูส่วน "เกรดเดอร์ตัวอย่าง" สำหรับรายละเอียด)
- t จะต้องสอดคล้องกับเงื่อนไข: $0 \leq t \leq m - 1$
- การเรียกฟังก์ชันนี้ จะไม่เพิ่มจำนวนของคำสั่งที่มีการสร้างขึ้น

หลังจากการเพิ่มคำสั่งสุดท้ายแล้ว `construct_instructions` จะต้องจบการทำงาน โปรแกรมจะถูกเรียกให้ทำงานกับกรณีทดสอบต่าง ๆ ที่ระบุข้อมูลนำเข้าที่ประกอบด้วยจำนวนเต็ม k บิตจำนวน n จำนวน $a[0], a[1], \ldots, a[n-1]$ โปรแกรมของคุณจะผ่านการทดสอบในกรณีทดสอบใด ๆ ถ้าผลลัพธ์ของโปรแกรม $c[0], c[1], \ldots, c[n-1]$ สำหรับข้อมูลนำเข้าใด ๆ ผ่านเงื่อนไขต่อไปนี้:

- ถ้า $s = 0$ $c[0]$ จะต้องมีค่าน้อยที่สุดในบรรดา $a[0], a[1], \ldots, a[n-1]$

- ถ้า $s = 1$ สำหรับแต่ละค่า i (เมื่อ $0 \leq i \leq n-1$) $c[i]$ จะต้องเป็นจำนวนเต็มที่น้อยที่สุดลำดับที่ $1+i$ ของรายการ $a[0], a[1], \ldots, a[n-1]$

การตรวจคำตอบที่คุณส่งมา อาจจะทำให้ได้ข้อความแสดงความผิดพลาดต่อไปนี้

- `Invalid index`: มีการส่งดัชนีของเรจิสเตอร์ที่ผิดพลาด (อาจจะเป็นลบ) มาเป็นพารามิเตอร์ t, x หรือ y ในการเรียกบางฟังก์ชัน
- `Value to store is not b bits long`: ความยาวของ v ที่ส่งให้ `append_store` ไม่เท่ากับ b
- `Invalid shift value`: ค่าของ p ที่ส่งให้กับฟังก์ชัน `append_left` or `append_right` ไม่อยู่ระหว่าง 0 ถึง b (รวม 0 และ b ด้วย)
- `Too many instructions`: ฟังก์ชันของคุณพยายามที่จะเพิ่มคำสั่งไปมากกว่า q คำสั่ง

ตัวอย่าง

ตัวอย่างที่ 1

สมมติว่า $s = 0$, $n = 2$, $k = 1$, $q = 1000$ มีจำนวนเต็มที่นำเข้า $a[0]$ และ $a[1]$ ที่แต่ละจำนวนมี $k = 1$ บิต ก่อนที่โปรแกรมที่คุณส่งมาจะเริ่มทำงาน $r[0][0] = a[0]$ และ $r[0][1] = a[1]$ ส่วนบิตต่าง ๆ ที่เหลือจะมีค่าเป็น 0 หลังจากที่ทุกคำสั่งในโปรแกรมของคุณทำงานแล้ว เราต้องการให้ $c[0] = r[0][0] = \min(a[0], a[1])$ ซึ่งคือค่าที่น้อยที่สุดของ $a[0]$ และ $a[1]$

มีรูปแบบของข้อมูลนำเข้า 4 แบบ:

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

สังเกตว่าในทั้ง 4 กรณี $\min(a[0], a[1])$ มีค่าเท่ากับการ AND รายบิตของ $a[0]$ และ $a[1]$ ดังนั้นคำตอบหนึ่งที่เป็นไปได้ก็คือการสร้างโปรแกรมโดยเรียกดังนี้:

1. `append_move(1, 0)` ซึ่งจะเพิ่มคำสั่งสำหรับคัดลอก $r[0]$ ไปยัง $r[1]$
2. `append_right(1, 1, 1)` ซึ่งจะเพิ่มคำสั่งสำหรับนำทุกบิตของ $r[1]$ เลื่อนไปทางขวา 1 บิตและเก็บผลลัพธ์กลับไปที่ $r[1]$ เนื่องจากจำนวนเต็มมีความยาว 1 บิต คำสั่งดังกล่าวจะทำให้ $r[1][0]$ มีค่าเท่ากับ $a[1]$
3. `append_and(0, 0, 1)` ซึ่งจะเพิ่มคำสั่งที่คำนวนการ AND รายบิตของ $r[0]$ และ $r[1]$ แล้วเก็บผลลัพธ์ลงใน $r[0]$ หลังจากคำสั่งนี้ทำงาน $r[0][0]$ จะถูกทำให้มีค่าเท่ากับผลการ AND รายบิตของ $r[0][0]$ และ $r[1][0]$ ซึ่งมีค่าเท่ากับผลของการ AND รายบิตของ $a[0]$ และ $a[1]$ ตามต้องการ

ตัวอย่างที่ 2

สมมติว่า $s = 1$, $n = 2$, $k = 1$, $q = 1000$ เช่นเดียวกับในตัวอย่างข้างต้น มีข้อมูลนำเข้ามาในโปรแกรมทั้งสิ้น 4 รูปแบบ ในทุก ๆ รูปแบบ $\min(a[0], a[1])$ คือผลของการ AND รายบิตของ $a[0]$ และ $a[1]$ และ $\max(a[0], a[1])$ คือผลของการ OR รายบิตของ $a[0]$ และ $a[1]$ คำตอบที่เป็นไปได้รูปแบบหนึ่งจะเรียกฟังก์ชันดังนี้:

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)`

หลังการทำงานตามคำสั่งเหล่านี้ $c[0] = r[0][0]$ จะมีค่าเท่ากับ $\min(a[0], a[1])$ และ $c[1] = r[0][1]$ จะมีค่าเท่ากับ $\max(a[0], a[1])$ ซึ่งเรียงลำดับแล้ว

ข้อจำกัด

- $m = 100$
- $b = 2000$
- $0 \le s \le 1$
- $2 \le n \le 100$
- $1 \le k \le 10$
- $q \le 4000$
- $0 \le a[i] \le 2^k - 1$ (สำหรับทุก ๆ i ที่ $0 \le i \le n - 1$)

ปัญหาย่อย

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

เกรดเดอร์ตัวอย่าง

เกรดเดอร์ตัวอย่างอ่านข้อมูลในรูปแบบต่อไปนี้:

- บรรทัดที่ 1 : $s\ n\ k\ q$

หลังจากนั้นจะมีบรรทัดต่อมาจำนวนหนึ่ง แต่ละบรรทัดระบุกรณีทดสอบหนึ่งกรณี แต่ละกรณีทดสอบจะเขียนในรูปแบบดังนี้:

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

เป็นการระบุกรณีทดสอบที่ข้อมูลนำเข้าประกอบด้วยจำนวนเต็ม $a[0], a[1], ..., a[n-1]$ หลังจากระบุกรณีทดสอบครบแล้วจะต้องมีบรรทัดสุดท้ายที่ระบุ -1 เท่านั้น เพื่อจบข้อมูลนำเข้าสำหรับเกรดเดอร์

เกรดเดอร์ตัวอย่างจะเริ่มต้นโดยการเรียก `construct_instructions(s, n, k, q)` ถ้าการเรียกดังกล่าวนี้ละเมิดเงื่อนไขบางอย่างที่ระบุในโจทย์ เกรดเดอร์จะพิมพ์ข้อความแสดงความผิดพลาดในตอนท้ายของส่วน "ราย

ละเอียดการเขียนโปรแกรม" และจบการทำงาน ถ้าไม่มีความผิดพลาดใด ๆ เกรดเดอร์ตัวอย่างจะพิมพ์คำสั่งทั้งหมดที่ส่งมาจาก `construct_instructions(s, n, k, q)` ตามลำดับ สำหรับคำสั่ง $store$ ค่า v จะถูกพิมพ์เป็นดัชนี 0 ถึง $b-1$

จากนั้น เกรดเดอร์ตัวอย่างจะประมวลผลกรณีทดสอบตามลำดับ แต่ละกรณีทดสอบ เกรดเดอร์จะประมวลผลโปรแกรมไปตามลำดับของกรณีทดสอบ

สำหรับคำสั่ง $print(t)$ ให้ $d[0], d[1], \ldots, d[n-1]$ เป็นลำดับของจำนวนเต็ม โดยที่แต่ละ i (สำหรับ $0 \leq i \leq n-1$) $d[i]$ เป็นค่าจำนวนเต็มของลำดับของบิตที่ $i \cdot k$ ถึง $(i+1) \cdot k - 1$ ของเรจิสเตอร์ t (ในขณะที่คำสั่งนั้นกำลังทำงาน) เกรดเดอร์จะพิมพ์ลำดับดังกล่าวในรูปแบบนี้: `register` t:
$d[0]$ $d[1]$ $\ldots$ $d[n-1]$

เมื่อคำสั่งทั้งหมดทำงานเสร็จสิ้น เกรดเดอร์ตัวอย่างพิมพ์ผลลัพธ์ของโปรแกรม

ถ้า $s = 0$ ผลลัพธ์ของเกรดเดอร์ตัวอย่างสำหรับแต่ละกรณีทดสอบจะอยู่ในรูปแบบต่อไปนี้:

- $c[0]$

ถ้า $s = 1$ ผลลัพธ์ของเกรดเดอร์ตัวอย่างสำหรับแต่ละกรณีทดสอบจะอยู่ในรูปแบบต่อไปนี้::

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

หลักการทำงานทุกกรณีทดสอบ เกรดเดอร์พิมพ์ `number of instructions:` X โดยที่ X คือจำนวนคำสั่งในโปรแกรมของคุณ