# MalwareX

2 seconds; 256 megabytes

Bob กำลังจะส่ง Alice ออกเดินทางไปทำภารกิจในอวกาศ Alice จะต้องใช้ระบบการสื่อสารพิเศษเพื่อรายงานข้อมูลต่าง ๆ กลับไปยังหัวหน้างาน Bob ที่อาศัยอยู่บนโลก

ระหว่างภารกิจ Alice จะถือเครื่องส่งข้อความ (sender) ที่สามารถส่งข้อความที่มีลักษณะเป็นบิตสตริง ความยาวเท่ากับ $N$ บิต ไปยังเครื่องรับข้อความ (receiver) ที่ Bob จะถือ โดยสำหรับแต่ละข้อความที่ Alice ส่ง เครื่องส่งข้อความจะแสดงผลข้อความเดียวกันนั้นที่เพิ่งถูกส่งออกไปล่าสุดเพื่อเป็นการยืนยันให้ Alice ทราบ

โชคร้าย ทีมพัฒนาระบบได้ค้นพบมัลแวร์ที่ล่องลอยอยู่ในอวกาศ ชื่อว่า MalwareX ซึ่งสามารถทำให้เครื่องส่งข้อความทำงานผิดแปลกไปจากเดิม กล่าวคือ ก่อนข้อความจะถูกส่ง <u>บิตต่าง ๆ จะถูกเรียงสับเปลี่ยน</u> และ <u>จะมีบิต $M$ บิตถูกแทรกเข้ามา</u> โดยที่ $M < N$ (ดังนั้นเครื่องรับข้อความจะได้รับข้อความยาว $N + M$ บิต)

การศึกษาเพิ่มเติมแสดงให้เห็นว่า MalwareX ประกอบไปด้วย **(1)** permutation[1] $P$ ความยาว $N$ ( $P : P_0 ... P_{N-1}$ ) และ **(2)** ลำดับ $L$ ที่ประกอบไปด้วยจำนวนเต็มไม่ติดลบ $N + 1$ จำนวน ( $L : L_0 ... L_N$ ) โดยที่ $L_0 + ... + L_N = M$ ทั้งนี้ ทั้ง permutation $P$ และลำดับ $L$ จะมีค่าคงที่ (จะไม่เปลี่ยนไปหลังจากมีการส่งข้อความแต่ละครั้ง) เมื่อ Alice พยายามส่งข้อความ $x : x_0 ... x_{N-1}$ จะมีสิ่งเกิดขึ้นตามลำดับต่อไปนี้

1. MalwareX เรียงสับเปลี่ยน $x$ โดยสร้าง $y = x_{P_0} x_{P_1} ... x_{P_{N-1}}$
2. MalwareX แทรกบิตเข้าไปใน $y$ ทั้งสิ้น $M$ บิต โดยที่
   a. สำหรับ $1 \le i \le N - 1$ แทรกบิต $L_i$ บิต ระหว่างตำแหน่ง $i - 1$ และตำแหน่ง $i$
   b. แทรกบิต $L_0$ บิตไปยังด้านหน้า (ก่อนตำแหน่ง $0$)
   c. แทรกบิต $L_N$ บิตไปยังด้านหลัง (หลังตำแหน่ง $N - 1$)

   <u>**บิตที่ถูกแทรกอาจไม่ได้มาจากการสุ่ม**</u> เรียกผลลัพธ์จากกระบวนการการแทรกบิตนี้ว่า $z$
3. เครื่องรับข้อความที่ Bob ถืออยู่ ได้รับข้อความ $z$
4. เครื่องส่งข้อความที่ Alice ถืออยู่ แสดงผล $z$ เพื่อเป็นการยืนยัน

Alice กำลังจะออกเดินทางไปทำภารกิจแล้ว ทีมพัฒนาระบบไม่มีเวลาที่จะออกแบบระบบการสื่อสารใหม่ได้ทันเวลา Alice และ Bob จึงตกลงกันว่า เมื่อ Alice เดินทางไปอยู่ในอวกาศแล้ว ให้ Alice ศึกษาหาลำดับ $L$ มาให้ได้ แล้วส่งลำดับนี้ให้ Bob ผ่านทางเครื่องส่งข้อความที่ติด MalwareX นี้เอง

สมมติว่า Alice ทราบลำดับ $L$ แล้ว หน้าที่ของคุณคือออกแบบมาตรการการติดต่อระหว่าง Alice และ Bob เพื่อให้ Bob ได้ทราบถึงลำดับ $L$ เช่นกัน นั่นคือ คุณจะต้องเขียนฟังก์ชันสองฟังก์ชัน ฟังก์ชันหนึ่งจะรับบทบาทเป็น Alice และอีกฟังก์ชันหนึ่งจะรับบทบาทเป็น Bob เกรดเดอร์จะเรียกฟังก์ชันแรกพร้อมกับ $N$, $M$, และลำดับ $L$ เพื่อให้ส่งข้อความแทน Alice หลังจากนั้น เกรดเดอร์จะเรียกฟังก์ชันที่สองพร้อมกับ $N$, $M$, และข้อความที่ Bob ได้รับตามลำดับที่ Alice ส่ง เพื่อให้ตอบว่าลำดับ $L$ เป็นอะไร คะแนนของคุณจะขึ้นอยู่กับจำนวนข้อความที่ Alice ส่ง

[1] permutation ความยาว **N** คือลำดับความยาว **N** ที่จำนวนเต็มตั้งแต่ **0** ถึง **N-1** ปรากฏในนั้นครั้งเดียวพอดี

## รายละเอียดการเขียนโปรแกรม

คุณต้องเขียนฟังก์ชันสองฟังก์ชันสำหรับมาตรการการติดต่อระหว่าง Alice และ Bob

ฟังก์ชันหนึ่ง จะถูกเรียกเพื่อให้ Alice ส่งข้อความ:

```
alice(int N, int M, int[] L)
```

- ฟังก์ชันนี้จะถูกเรียกเพียงหนึ่งครั้งในแต่ละ**กรณีทดสอบย่อย**
- $N$ : จำนวนบิตที่เครื่องส่งข้อความส่ง และ MalwareX เรียงสับเปลี่ยน
- $M$ : จำนวนบิตที่ MalwareX แทรกเข้ามาในแต่ละข้อความ
- $L$ : อาเรย์ความยาว $N+1$ ระบุจำนวนบิตที่ MalwareX แทรกเข้ามาในแต่ละตำแหน่ง
- ฟังก์ชัน `alice` สามารถเรียกฟังก์ชัน `sendMessage` เพื่อทำการส่งข้อความได้
- ฟังก์ชันนี้ไม่ควรส่งค่าใด ๆ คืน

ฟังก์ชันสอง จะถูกเรียกเพื่อให้ Bob สรุปค่าของอาเรย์ $L$

```
int[] bob(int N, int M, int Q, string S[])
```

- ฟังก์ชันนี้จะถูกเรียกเพียงหนึ่งครั้งในแต่ละ**กรณีทดสอบย่อย** หลังจากฟังก์ชัน `alice` ทำงานเสร็จสิ้นในกรณีทดสอบย่อยนั้น
- $N$ : จำนวนบิตที่เครื่องส่งข้อความส่งและ MalwareX เรียงสับเปลี่ยน
- $M$ : จำนวนบิตที่ MalwareX แทรกเข้ามาในแต่ละข้อความ
- $Q$ : จำนวนข้อความที่ถูกส่งมาจาก Alice
- $S$ : อาเรย์ความยาว $Q$ ระบุข้อความทั้งหมดที่ได้รับจาก Alice เรียงตามลำดับการส่ง แต่ละข้อความเป็นบิตสตริงที่มีความยาวเท่ากับ $N+M$ บิต
- ฟังก์ชันนี้ต้องคืนค่าอาเรย์ $L$

ฟังก์ชัน `alice` สามารถเรียกฟังก์ชัน `sendMessage` เพื่อทำการส่งข้อความได้

```
string sendMessage(string s)
```

- เรียกใช้งานฟังก์ชันนี้จาก `alice` เพื่อส่งข้อความผ่านเครื่องส่งข้อความ
- $s$ : บิตสตริงความยาวเท่ากับ $N$ บิต ระบุข้อความที่ Alice ต้องการส่ง
- ฟังก์ชันนี้จะทำการสร้างข้อความหลังจากการเรียงสับเปลี่ยนและการแทรกบิตโดย MalwareX ตาม permutation $P$ และลำดับ $L$ ที่กำหนด **<u>บิตที่ถูกแทรกอาจไม่ได้มาจากการสุ่ม</u>**
- ฟังก์ชันนี้คืนบิตสตริงความยาวเท่ากับ $N+M$ บิต
- ฟังก์ชันนี้จะถูกเรียกได้ไม่เกิน $15$ ครั้ง ในแต่ละ**กรณีทดสอบย่อย**

## ตัวอย่าง

สมมติว่า $N = 4$, $M = 2$, และ ลำดับ $L$ (ที่ Alice ทราบแต่ Bob ไม่ทราบ) มีค่าเป็น `[0, 1, 1, 0, 0]` นอกจากนี้ สมมติว่า permutation $P$ (ที่ทั้ง Alice และ Bob ไม่ทราบ) มีค่าเป็น `[0, 3, 2, 1]`

**<u>อันดับแรก</u>** เกรดเดอร์จะเรียก `alice(4, 2, [0, 1, 1, 0, 0])` เพื่อทดสอบโปรแกรมของคุณ `alice` สามารถเรียก `sendMessage` เพื่อส่งข้อความ สมมติว่ามีการโต้ตอบกันดังต่อไปนี้

| การเรียกครั้งที่ | ข้อความที่ `alice` พยายามส่ง (s) | ข้อความที่ bob จะได้รับ (`sendMessage(s)`) |
|---|---|---|
| 1 | `"0000"` | `"010000"` |
| 2 | `"0000"` | `"000100"` |
| 3 | `"0000"` | `"000000"` |
| 4 | `"1111"` | `"101011"` |
| 5 | `"0001"` | `"011000"` |

สังเกตว่า ถึงแม้ว่า `alice` จะพยายามส่งข้อความเดียวกัน (ในการเรียกครั้งที่ 1 ถึง 3) ข้อความที่ bob จะได้รับอาจต่างกัน และขอเน้นอีกครั้งว่า**<u>บิตที่ถูกแทรกอาจไม่ได้มาจากการสุ่ม</u>**

สังเกตเช่นกันว่า `sendMessage` จะคืนค่าเป็นข้อความที่ bob จะได้รับ ดังนั้น `alice` สามารถใช้ข้อมูลนี้ในการตัดสินใจข้อความที่จะส่งในอนาคตได้

หลังจากส่งข้อความ 5 ข้อความนี้แล้ว `alice` มั่นใจว่า bob จะมีข้อมูลเพียงพอจะทราบได้ว่าลำดับ $L$ เป็นอะไร จึงหยุดการทำงาน (ด้วยคำสั่ง `return;`)

**<u>อันดับต่อมา</u>** เกรดเดอร์จะเรียก `bob(4, 2, 5, ["010000", "000100", "000000", "101011", "011000"])` ซึ่งจะต้องคืนค่า `[0, 1, 1, 0, 0]`

## ข้อจำกัด

- $2 \le N \le 128$
- $1 \le M \le N - 1$
- จำนวนเต็มตั้งแต่ $0$ ถึง $N - 1$ ปรากฏใน $P$ ครั้งเดียวพอดี
- $L_i \ge 0$ (ในแต่ละ $0 \le i \le N$) และ $L_0 + ... + L_N = M$

เกรดเดอร์อาจเลือกแทรกบิตขึ้นอยู่กับข้อความที่ส่งในครั้งนั้น ๆ และ/หรือ ในครั้งก่อน ๆ

**<u>กรณีทดสอบหนึ่งอาจมีหลายกรณีทดสอบย่อย</u>** แต่ในแต่ละกรณีทดสอบ ผลรวม $N + M$ จะไม่เกิน $10^6$

## ปัญหาย่อย

1. (8 คะแนน) $N + M \leq 16$
2. (3 คะแนน) $L_0 = M$ หรือ $L_N = M$
3. (6 คะแนน) $L_i = M$ สำหรับบาง $0 \leq i \leq N$
4. (27 คะแนน) $2M \leq N$
5. (28 คะแนน) $P_i = i$ สำหรับทุก $0 \leq i \leq N - 1$
6. (28 คะแนน) ไม่มีข้อกำหนดเพิ่มเติม

ถ้าโปรแกรมของคุณได้รับผลการตรวจเป็น **Accepted** และ ในแต่ละกรณีทดสอบย่อย ฟังก์ชัน `sendMessage` ถูกเรียกใช้งานมากที่สุด $Q$ ครั้ง คะแนน $P$ ในแต่ละกรณีทดสอบย่อยจะขึ้นอยู่กับปัญหาย่อยซึ่งถูกคำนวณไว้ดังนี้

- ปัญหาย่อย 1. ถ้า $Q < N + M$ แล้ว $P = 8$ นอกจากนั้น $P = 0$
- ปัญหาย่อย 2. ถ้า $Q < 2 + \log_2 N$ แล้ว $P = 3$ นอกจากนั้น $P = 0$
- ปัญหาย่อย 3. ถ้า $Q < 2 + \log_2 N$ แล้ว $P = 6$ นอกจากนั้น $P = 0$
- ปัญหาย่อย 4. ถ้า $Q < 2 + \log_2 N$ แล้ว $P = 27$ นอกจากนั้น $P = 0$
- ปัญหาย่อย 5. ถ้า $Q < 2 + \log_2 N$ แล้ว $P = 28$ นอกจากนั้น $P = 0$
- ปัญหาย่อย 6. ถ้า $Q < 2 + \log_2 N$ แล้ว $P = 28$ นอกจากนั้น $P = 0$

## เกรดเดอร์ตัวอย่าง

เกรดเดอร์ตัวอย่างอ่านข้อมูลเข้าในรูปแบบต่อไปนี้

- บรรทัดที่ 1: $N\ M$
- บรรทัดที่ 2: $L_0\ L_1\ \ldots\ L_N$
- บรรทัดที่ 3: $P_0\ P_1\ \ldots\ P_{N-1}$

ในฟังก์ชั่น `sendMessage` แต่ละบิตที่ถูกแทรกมาโดย MalwareX จะได้รับการสุ่ม (แบบ psuedo-random) เป็น `0` หรือ `1` ด้วยโอกาสที่เท่า ๆ กัน ซึ่งในเกรดเดอร์จริงอาจไม่ได้เป็นแบบนี้

ถ้าโปรแกรมของคุณได้รับผลการตรวจเป็น **Accepted** เกรดเดอร์จะพิมพ์ `Accepted: Q` โดย `Q` คือจำนวนครั้งที่ฟังก์ชัน `sendMessage` ถูกเรียกใช้

ถ้าโปรแกรมของคุณได้รับผลการตรวจเป็น **Wrong Answer** เกรดเดอร์ตัวอย่างจะพิมพ์ `Wrong Answer: MSG` โดย `MSG` จะเป็นหนึ่งในข้อความต่อไปนี้:

- `illegal call`: เรียกใช้งานฟังก์ชัน `sendMessage` ไม่ถูกต้องตามเงื่อนไขที่กำหนด
- `incorrect`: คำตอบที่ฟังก์ชัน `bob` ส่งคืนไม่ถูกต้อง
- `too many calls`: `sendMessage` ถูกเรียกใช้เกิน $15$ ครั้ง