# Permutation Game

อลิศ (Alice) กับบ๊อบ (Bob) เป็นเพื่อนกันมาตั้งแต่เด็ก ทั้งสองชอบเล่นเกมวัดความฉลาด วันนี้ทั้งสองจะเล่นเกมใหม่เกี่ยวกับกราฟ

ชุดเกมประกอบไปด้วยกราฟที่ **เชื่อมต่อกัน** (connected ที่มีจุดยอดจำนวน $m$ จุด โดยมีหมายเลขตั้งแต่ $0$ ถึง $m-1$ และมีเส้นเชื่อมจำนวน $e$ เส้น ซึ่งมีหมายเลขตั้งแต่ $0$ ถึง $e-1$ เส้นเชื่อมที่ $i$ เชื่อมจุดยอด $u[i]$ กับ $v[i]$

ชุดเกมยังประกอบด้วยการเรียงสับเปลี่ยน (permutation) $p[0], p[1], \ldots, p[n-1]$ ที่มีความยาว $n$ โดยที่ $m \leq n$ การเรียงสับเปลี่ยน (permutation) คืออาร์เรย์ที่จำนวนตั้งแต่ $0$ ถึง $n-1$ ปรากฏหนึ่งครั้งพอดี ตามบางลำดับ **คะแนน** ของการเรียงสับเปลี่ยน $p$ คือจำนวนดัชนี $i$ ที่ $p[i] = i$.

เกมจะเล่นไปไม่เกิน $10^{100}$ รอบ ในแต่ละรอบ เกมจะดำเนินไปดังนี้:

1. ถ้าอลิศตัดสินใจว่าจะจบเกม เกมจะจบลง
2. ไม่เช่นนั้น อลิศจะเลือก **ดัชนีที่แตกต่างกัน** $t[0], t[1], \ldots, t[m-1]$ สำหรับ $0 \leq t[i] < n$ สังเกตว่าเกม**ไม่**ได้ต้องการให้ $t[0] < t[1] < \ldots < t[m-1]$.
3. บ๊อบเลือกดัชนี $j$ ที่ $0 \leq j < e$ ของเส้นเชื่อมในกราฟ และสลับ $p[t[u[j]]]$ กับ $p[t[v[j]]]$.

อลิศต้องการจะทำให้คะแนนสุดท้ายของการเรียงสับเปลี่ยนมีค่ามากที่สุด ในขณะที่บ๊อบต้องการทำให้คะแนนสุดท้ายน้อยที่สุด

งานของคุณคือช่วยให้อลิศเล่นแข่งกับบ๊อบ โดยการเล่นของบ๊อบจะถูกดำเนินการโดยเกรดเดอร์

เราจะนิยามให้ *คะแนนที่ดีที่สุด* คือคะแนนสุดท้ายของการเรียงสับเปลี่ยนถ้าอลิศและบ๊อบเล่นเกมอย่างดีที่สุด

คุณจะต้องหาคะแนนที่ดีที่สุดดังกล่าวจากนั้นเล่นเกมกับบ๊อบเพื่อให้ได้คะแนน**อย่างน้อย**คะแนนที่ดีที่สุดนั้นหลังจากการเล่นบางตา

**สังเกตว่ากลยุทธ์ของอลิศจะต้องใช้ได้ ไม่ว่าบ๊อบจะเล่นอย่างไร รวมถึงในกรณีที่บ๊อบไม่ได้เล่นอย่างดีที่สุดด้วย**

## รายละเอียดการเขียนโปรแกรม

คุณจะต้องเขียนฟังก์ชันต่อไปนี้:

```
int Alice(int m, int e, std::vector<int> u, std::vector<int> v,
          int n, std::vector<int> p)
```

- $m$: จำนวนจุดยอดในกราฟ
- $e$: จำนวนเส้นเชื่อมในกราฟ

- $u$ และ $v$: อาเรย์ขนาด $e$ ที่ระบุเส้นเชื่อมในกราฟ
- $n$: ความยาวของการเรียงสับเปลี่ยน (permutation)
- $p$: อาเรย์ความยาว $n$ ที่ระบุการเรียงสับเปลี่ยน (permutation)
- ฟังก์ชันจะถูกเรียกครั้งเดียวเท่านั้น
- ฟังก์ชันจะต้องคืนจำนวนเต็มหนึ่งจำนวน – คะแนนที่ดีที่สุดของเกม

ในฟังก์ชันดังกล่าว คุณสามารถเรียกฟังก์ชันด้านล่างได้ :

```
int Bob(std::vector<int> t)
```

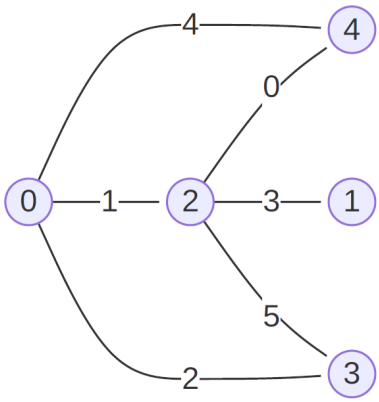
- $t$: อาเรย์ขนาด $m$ ที่มีดัชนีแตกต่างกัน โดยที่ $0 \le t[i] < n$ และ $t[i] \ne t[j]$ สำหรับ $i \ne j$
- ฟังก์ชันจะคืนจำนวนเต็มหนึ่งจำนวน $j$ ที่สอดคล้องกับเงื่อนไข $0 \le j < e$
- สามารถเรียกฟังก์ชันนี้ได้หลายครั้ง

## ตัวอย่าง

พิจารณาการเรียกต่อไปนี้:

```
Alice(5, 6, [4, 0, 3, 1, 4, 2], [2, 2, 0, 2, 0, 3],
      10, [8, 2, 7, 6, 1, 5, 0, 9, 3, 4])
```

กราฟแสดงดังรูปด้านล่าง:




และ $p$ จะมีค่าเริ่มต้นเป็น $[8, 2, 7, 6, 1, 5, 0, 9, 3, 4]$.

จากเงื่อนไขด้านบน เราสามารถพิสูจน์ได้ว่า คะแนนที่ดีที่สุดของการเรียงสับเปลี่ยนจะเท่ากับ $1$

สมมติว่า อลิศ เล่นเกมดังด้านล่าง $4$ ตา:

| ค่า $t$ ที่ส่งให้ Bob | ค่าที่คืนจาก Bob | คู่ดัชนีที่เกี่ยวข้องใน $p$ | $p$ ภายหลังการสลับโดย Bob |
|---|---|---|---|
| $[3,1,5,2,0]$ | $5$ | $5,2$ | $[8,2,5,6,1,7,0,9,3,4]$ |
| $[9,3,7,2,1]$ | $0$ | $1,7$ | $[8,9,5,6,1,7,0,2,3,4]$ |
| $[5,6,7,8,9]$ | $1$ | $5,7$ | $[8,9,5,6,1,2,0,7,3,4]$ |
| $[7,5,2,3,6]$ | $3$ | $5,2$ | $[8,9,2,6,1,5,0,7,3,4]$ |

สังเกตว่าการเล่นของอลิศและบ็อบข้างต้นไม่จำเป็นต้องเป็นการเล่นที่ดีที่สุด การเล่นที่แสดงด้านบนมีไว้เพื่อเป็นตัวอย่างของการเล่นเกมเท่านั้น สังเกตด้วยว่าอลิศสามารถจบเกมได้เลยทันทีเช่นกัน เนื่องจากคะแนนของการเรียงสับเปลี่ยนในตอนแรกนั้นมีค่าเท่ากับ $1$ แล้ว

หลังจากที่อลิศได้เล่นเกมตามด้านบนแล้ว คะแนนที่ได้จริงของการเรียงสับเปลี่ยนจะเท่ากับ $3$ ( $p[2]=2, p[5]=5, p[7]=7$).

สุดท้าย ฟังก์ชัน `Alice()` จะคืนค่า $1$ – คะแนนที่ดีที่สุดของการเรียงสับเปลี่ยน

**สังเกตว่า แม้ว่าอลิศจะสามารถทำคะแนนได้ $3$ โดยเล่นกับบ็อบ คุณจะได้ $0$ คะแนนในกรณีนี้ ถ้าค่าที่คืนจากฟังก์ชัน `Alice()` เท่ากับ $3$ แทนที่จะเป็น $1$**

## เงื่อนไข

- $2 \le m \le 400$
- $m-1 \le e \le 400$
- $0 \le u[i], v[i] < m$
- $m \le n \le 400$
- $0 \le p[i] < n$
- กราฟเชื่อมต่อกัน (connected) ไม่มี self-loops หรือเส้นเชื่อมหลายเส้นที่เชื่อมจุดยอดคู่เดียวกัน (นั่นคือไม่มี multiple edges)
- $p$ เป็นการเรียงสับเปลี่ยน (permutation) นั่นคือ $p[i] \ne p[j]$ สำหรับ $i \ne j$

## ปัญหาย่อย

1. (6 points) $m = 2$
2. (6 points) $e > m$
3. (10 points) $e = m - 1$
4. (24 points) $e = m = 3$
5. (24 points) $e = m = 4$
6. (30 points) $e = m$

สำหรับแต่ละปัญหาย่อย คุณสามารถได้คะแนนบางส่วนได้ ให้ $r$ เป็นอัตราส่วนมากที่สุดของ $\frac{k}{n}$ ในทุก ๆ กรณีทดสอบในปัญหาย่อยนั้น โดยที่ $k$ คือจำนวนรอบที่เล่นเกม (นั่นคือ จำนวนครั้งที่เรียกฟังก์ชัน `Bob()`) คะแนนของปัญหาย่อยนั้นที่คุณจะได้รับจะคูณด้วยจำนวนตามตารางด้านล่าง:

| เงื่อนไข | ตัวคูณ |
|---|---|
| $12 \le r$ | $0$ |
| $3 < r < 12$ | $1 - \log_{10}(r-2)$ |
| $r \le 3$ | $1$ |

กล่าวคือ ถ้าคุณแก้ปัญหาโดยเล่นเกมไม่เกิน $3n$ รอบ คุณจะได้คะแนนเต็มในปัญหาย่อยนั้น ถ้าคุณใช้มากกว่า $12n$ รอบ คุณจะได้คะแนน $0$ ในปัญหาย่อยนั้น (ซึ่งจะแสดงในระบบว่า `Output isn't correct`).

## เกรดเดอร์ตัวอย่าง

เกรดเดอร์ตัวอย่างจะอ่านข้อมูลนำเข้าในรูปแบบต่อไปนี้:

- line $1$: $m\ e$
- line $2+i$ $(0 \le i \le e-1)$: $u[i]\ v[i]$
- line $2+e$: $n$
- line $3+e$: $p[0]\ p[1]\ \ldots\ p[n-1]$

เกรดเดอร์ตัวอย่างจะพิมพ์ผลลัพธ์ในรูปแบบต่อไปนี้:

- line $1$: การเรียงสับเปลี่ยนสุดท้าย $p$
- line $2$: ค่าที่คืนจาก `Alice()`
- line $3$: คะแนนของการเรียงสับเปลี่ยนสุดท้าย
- line $4$: จำนวนรอบที่เล่น