หิมะ (snow)

รัฐมินนิโซตาเป็นรัฐที่อยู่ทางตอนเหนือของอเมริกา ทำให้ช่วงฤดูหนาวจะมีอากาศที่เย็นจัดและหิมะตกอยู่เป็นประจำ รัฐมินนิโซตาเมืองจำนวน N เมืองหมายเลข 0 ถึง $N-1$ และถนนจำนวน M ถนนหมายเลข 0 ถึง $M-1$ โดยถนนที่ i สามารถใช้ในการเดินทางจากเมือง $U[i]$ ไปเมือง $V[i]$ และยังสามารถใช้ในการเดินทางจากเมือง $V[i]$ ไปเมือง $U[i]$ ได้เช่นกัน (ใช้ในการเดินทางได้สองทิศทาง) รับประกันว่ามี**ถนนอย่างมากแค่เส้นเดียวที่เชื่อมระหว่างสองคู่เมืองใดๆ** และรับประกันว่า**ทุกคู่เมืองสามารถเดินทางไปหากันได้ด้วยถนนที่มีอยู่**

รัฐมินนิโซตาประกอบด้วยเมืองสำคัญทั้งหมด**สองเมือง** แต่ผู้ว่าการรัฐไม่รู้ว่าเมืองทั้งสองคือเมืองอะไร

ปกติแล้วในช่วงฤดูหนาว หากเราโรยเกลือบนถนนจะทำให้ถนนสามารถใช้งานได้ (เกลือช่วยให้หิมะไม่แข็งเป็นน้ำแข็ง) เพื่อให้ผู้ว่าการสามารถหาได้ว่าสองเมืองสำคัญคือเมืองอะไร ในแต่ละวันเขาสามารถเลือกโรยเกลือบนถนนแต่ละถนนหรือไม่โรยก็ได้ โดย**ถนนที่ถูกโรยเกลือ**จะสามารถ**ใช้ในเดินทางได้** แต่สำหรับ**ถนนที่ไม่ได้โรยเกลือ** หิมะจับกันเป็นน้ำแข็งและ**ไม่สามารถ**ใช้ในการเดินทางได้

ในแต่ละวันหลังจากที่ผู้ว่าการเลือกว่าจะโรยเกลือบนถนนแล้ว เขาจะสามารถรู้ได้ว่าจะสามารถเดินทางระหว่างสองเมืองสำคัญได้หรือไม่ กล่าวคือถ้าสามารถเดินทางระหว่างสองเมืองสำคัญได้แปลว่ามี**เส้นทางระหว่างสองเมืองสำคัญที่ประกอบไปด้วยถนนที่ถูกโรยเกลือในวันนั้นเท่านั้น**

หมายเหตุ: การโรยเกลือในแต่ละวันไม่มีความเกี่ยวข้องกัน เมื่อจบวันเกลือจะสลายและไม่สามารถละลายน้ำแข็งในวันถัดไปได้

เขาต้องการรู้ว่าสองเมืองสำคัญคือเมืองใด แต่เนื่องจากเขามีเวลาจำกัดเขาจึงต้องการรู้ให้เร็วที่สุดเท่าที่จะเป็นไปได้

รายละเอียดการเขียนโปรแกรม

```
vector<int> find_pair(int N, int M, vector<int> U, vector<int> V)
```

- ฟังก์ชันนี้จะถูกเรียกเรียกอย่างมาก 5 ครั้ง
- ฟังก์ชันนี้จะต้องทำการคืนค่าเป็น vector ที่ประกอบไปด้วยจำนวนเต็มสองจำนวนแทนเลขของเมืองสำคัญทั้งสอง
- การเรียกแต่ละครั้งให้พิจารณาเป็นปัญหาที่ไม่ขึ้นต่อกัน

```
bool road_salt(vector<bool> X)
```

- vector<bool> X : X แทนการโรยเกลือในวันนั้น
  - หาก **X[i] = false** แปลว่าเราจะไม่โรยเกลือบนถนนเส้นนั้นและถนนเส้นนั้นจะใช้งาน**ไม่ได้**สำหรับวันนั้น
  - หาก **X[i] = true** แปลว่าเราจะโรยเกลือบนถนนเส้นนั้นและถนนเส้นนั้นจะใช้งาน**ได้**สำหรับวันนั้น

- การคืนค่าของฟังก์ชัน road_salt
  - หาก**สามารถ**เดินทางระหว่างสองเมืองสำคัญได้จะ **return true**
  - หาก**ไม่สามารถ**เดินทางระหว่างสองเมืองสำคัญได้จะ **return false**

ขอบเขต

- $3 \leq N \leq 100\ 000$
- $M \leq 200\ 000$
- $0 \leq U[i], V[i] < N$ เมื่อ $0 \leq i < M$
- ทุกคู่เมืองสามารถเดินทางไปหากันได้ด้วยถนนที่มีอยู่

ปัญหาย่อย

1. (3 คะแนน) เมืองเชื่อมกันเป็นเส้นตรง และ $N \leq 50$ (มีถนน $N-1$ ถนน และถนนที่ i เชื่อมระหว่างเมือง $i+1$ กับเมือง i)
2. (26 คะแนน) เมืองเชื่อมกันเป็นเส้นตรง (มีถนน $N-1$ ถนน และถนนที่ i เชื่อมระหว่างเมือง $i+1$ กับเมือง i)
3. (4 คะแนน) เมืองเชื่อมกันเป็นวงกลม $M = N$ (ถนนที่ $i\ (0 \leq i \leq N-1)$ เชื่อมระหว่างเมือง $i+1$ กับเมือง i และถนนที่ N เชื่อมระหว่างเมือง $N-1$ กับ เมือง 0)
4. (5 คะแนน) เมืองทุกคู่มีถนนเชื่อมกัน (complete graph) $N \leq 50$
5. (9 คะแนน) เมืองทุกคู่มีถนนเชื่อมกัน (complete graph) $N \leq 300$
6. (4 คะแนน) $N \leq 10$
7. (8 คะแนน) สองเมืองสำคัญสามารถเดินทางไปหากันได้ โดยใช้ถนนแค่เส้นเดียว
8. (10 คะแนน) เมืองที่ 0 เป็นเมืองสำคัญ
9. (29 คะแนน) $M = N - 1$ (กราฟมีลักษณะเป็นต้นไม้)
10. (2 คะแนน) ไม่มีเงื่อนไขเพิ่มเติม

การให้คะแนน

กำหนดให้ M แทนจำนวนครั้งที่คุณเรียกใช้ฟังก์ชัน `road_salt` ในแต่ละกรณีทดสอบย่อยและกำหนดให้ k คือค่ามากที่สุดของ M ทุกกรณีทดสอบในปัญหาย่อยนี้ คะแนนของคุณจะเป็นไปตามเงื่อนไขดังต่อไปนี้

สำหรับทุก subtask **คุณจะได้คะแนนก็ต่อเมื่อค่า** $k \leq 55$

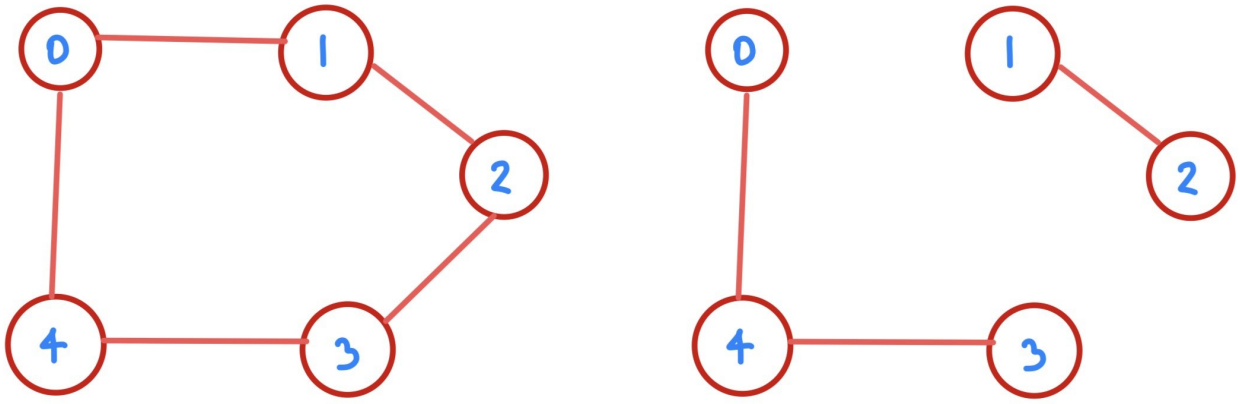
หากในแต่ละกรณีทดสอบย่อย คุณเรียกใช้ฟังก์ชัน `roat_salt` มากกว่า 55 ครั้งโปรแกรมจะจบการทำงานทันทีและเกรดเดอร์จะรายงาน wrong answer ออกมา

ตัวอย่าง

พิจารณาตัวอย่างดังต่อไปนี้ `find_pair(5, 5, [1, 2, 3, 4, 0], [0, 1, 2, 3, 4])` และให้เมืองสำคัญคือเมือง 2 และเมือง 4

ฟังก์ชันดังกล่าวอาจเรียกฟังก์ชัน `road_salt` ตามลำดับต่อไปนี้

ฟังก์ชัน	ค่าที่ได้
`road_salt([true, true, true, true, true])`	`true`
`road_salt([false, true, false, true, true])`	`false`
`road_salt([false, true, true, true, false])`	`true`
`road_salt([true, true, false, true, false])`	`false`



รูปทางได้ซ้ายคือกราฟหลังจากเรียก `road_salt([true, true, true, true, true])` ในขณะที่รูปทางด้านขวาคือกราฟหลังจากเรียก `road_salt([false, true, false, true, true])`

สมมุติว่าข้อมูลเพียงพอที่จะให้รู้ว่าเมืองสำคัฐคือเมืองสำคัญคือเมือง 2 และเมือง 4 ฟังกชัน `find_pair` ควรจะ return $[2, 4]$ หรือ $[4, 2]$ จึงจะเป็นคำตอบที่ถูกต้อง

เกรดเดอร์ตัวอย่าง

กำหนดให้ T คือจำนวนชุดทดสอบย่อย (จำนวนครั้งที่เราโปรแกรมเรียกฟังก์ชัน `find_pair` ในการทำงานของโปรแกรมหนึ่งรอบ)

กำหนดให้ X กับ Y เป็นหมายเลขของเมือง

- บรรทัดที่ 1: T

สำหรับแต่ละข้อมูลทดสอบย่อย

- บรรทัดที่ 1: $N\ M\ X\ Y$
- M บรรทัดถัดมา : $U[i]\ V[i]$

ข้อมูลส่งออกสำหรับเกรดเดอร์ตัวอย่าง แต่ละบรรทัดจะความถูกต้องของคำตอบกับจำนวนครั้งที่ทำการเรียกฟังก์ชัน `road_salt` สำหรับแต่ละชุดทดสอบย่อย

ในเกรดเดอร์ตัวอย่างสามารถทำการเรียกฟังก์ชัน `road_salt` กี่ครั้งก็ได้ แต่ในเกรดเดอร์จริงจะสำหรับแต่ละปัญหาย่อยจะเรียกได้ไม่เกิน 55 ครั้ง

ข้อจำกัด

- Time limit: 5 seconds
- Memory limit: 512 MB