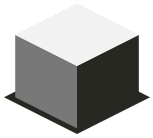
CCC beta #6: Valentine

Editorial

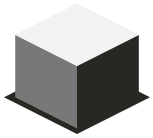


1. Divisor Present by op01

สำหรับข้อนี้ เราไม่สามารถไล่วนลูปตรง ๆ ตั้งแต่ 1 จนถึง N ได้ เพราะจะใช้เวลานานเกิน คือ $O(N)$ โดยวิธีการทำข้อนี้มีได้หลากหลาย วิธีที่ง่ายที่สุดที่หลายคนใช้คือ ไล่ตั้งแต่ 1 ไปจนถึง $\sqrt{N}$ โดยเราจะรู้ได้ทันทีว่า ถ้า i สามารถหาร N ลงตัวได้ แล้ว $\frac{N}{i}$ ก็จะสามารถหาร N ลงตัวได้เช่นกัน ดังนั้นเราสามารถหาจำนวนทั้งหมดที่หาร N ลงตัวได้ในเวลา $O(\sqrt{N})$ แล้วเราเอาจำนวนทั้งหมดนั้นมาเรียงจากน้อยไปมาก ซึ่งจำนวนทั้งหมดที่หาร N ลงตัวนั้น มีไม่เกิน $2\sqrt{N}$ ตัว (มาจากแนวคิดข้างต้น) ดังนั้น วิธีนี้ใช้เวลาในการทำงาน $O(\sqrt{N}\ log(\sqrt{N}))$

สิ่งที่ต้องระวัง คือ เมื่อ N เป็นจำนวนที่สามารถหา $\sqrt{N}$ แล้วได้เป็นจำนวนเต็ม เช่น 9 ซึ่งเมื่อเราหา $i = 3$ แล้ว เราจะหา $\frac{N}{i} = 3$ ได้อีกตัว ซึ่งจะซ้ำกัน

มีอีกวิธีคือเราไล่ print ค่า i เมื่อไล่ i จาก 1 ไปจนถึง $\sqrt{N}$ และเมื่อเสร็จแล้ว ให้ไล่ print ค่า $\frac{N}{i}$ เมื่อไล่ i จาก $\sqrt{N} - 1$ (เพื่อกันกรณีที่เป็นจำนวนที่หา $\sqrt{N}$ ได้เป็นจำนวนเต็ม) กลับมาจนถึง 1 เราก็จะได้ลำดับตัวประกอบที่เรียงกันแล้วในเวลา $2\sqrt{N}$ หรือก็คือ $O(\sqrt{N})$ นั่นเอง



2. Keeper of The Rose by PanTA

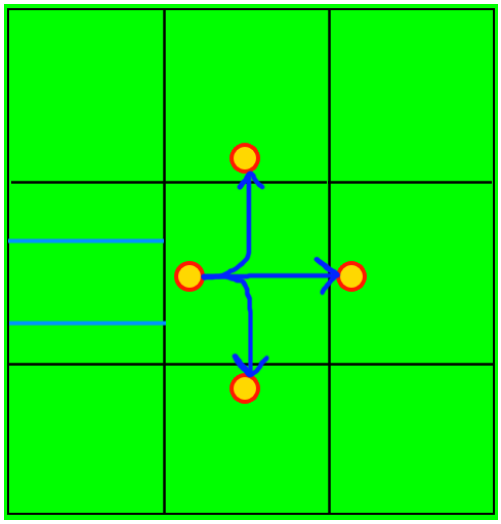
ข้อนี้เราสามารถทำตรง ๆ ได้ โดยจำเวลาที่เก็บดอกกุหลาบครั้งล่าสุดไว้ พอจะมาเก็บดอกนั้น ๆ ก็เช็คเวลาเก็บครั้งก่อนกับเวลาปัจจุบันว่าห่างกันพอจะให้ดอกกุหลาบพร้อมเก็บหรือไม่ ถ้าทำได้ให้ทำการเก็บดอกกุหลาบแล้วอัพเดทเวลาเก็บครั้งก่อนให้เป็นเวลาปัจจุบัน ถ้าไม่ได้ก็ไม่ทำการเก็บดอกกุหลาบ

โดยรวมจะใช้เวลาในการทำงาน $O(M)$

3. Choco Pipe by PalmPTSJ

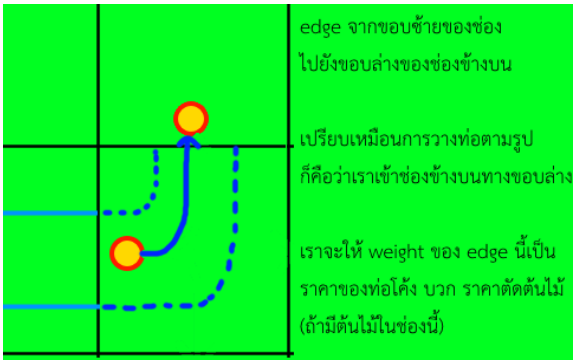
สำหรับข้อนี้ ก่อนอื่นต้องขออภัยผู้แข่งหลายๆคน ที่อธิบายโจทย์และให้ตัวอย่างไม่ดีพอ ทำให้มีหลายคนสับสนระหว่างความยาวและความสูงของสนามหญ้า และทำให้หลายคนอาจต้องเสียคะแนน ข้อนี้ใช้การรับเป็น W, H แทนความยาว (ในแนวนอน) และความสูง (ในแนวตั้ง) ตามลำดับ ซึ่งในอนาคต จะพยายามใช้เป็น R, C คือ จำนวนแถวและหลัก ซึ่งเป็นที่นิยมใช้มากกว่า และจะมีเคสตัวอย่างที่ดีกว่านี้

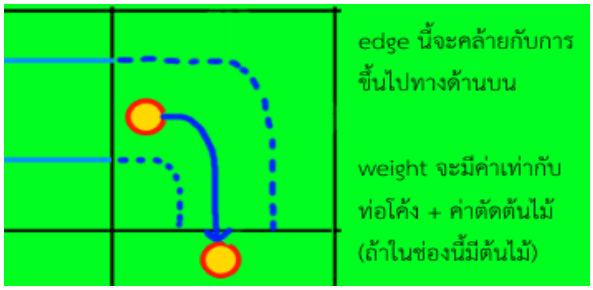
ข้อนี้ เราสามารถเปลี่ยนโจทย์ ให้กลายเป็นการหา Shortest Path ได้ โดยการมองขอบแต่ละด้านของทุกช่องให้เป็น node และทำการใส่ edge เพื่อเชื่อมระหว่างด้านนั้นกับด้านที่สัมพันธ์กันในอีกช่อง ดังรูปข้างล่าง

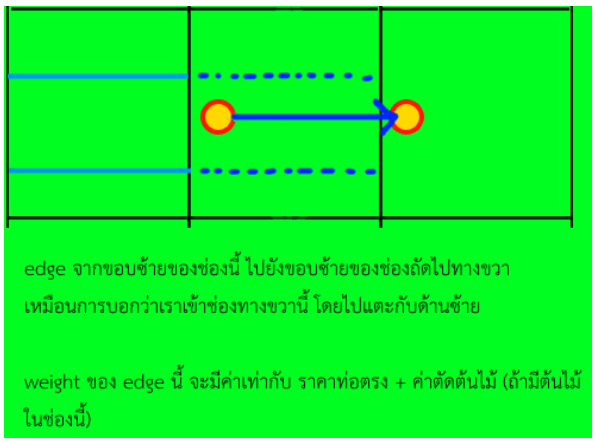


โดย node ที่อยู่ด้านซ้ายของช่อง หมายความว่าตอนนี้มีท่อที่มาแตะกับขอบทางด้านซ้าย ซึ่งจะมีทิศไปต่อได้อีก 3 ทิศ คือ ขึ้นข้างบน ไปแตะขอบล่าง / ลงข้างล่าง ไปแตะขอบบน / ไปทางขวาต่อ ไปแตะขอบซ้าย

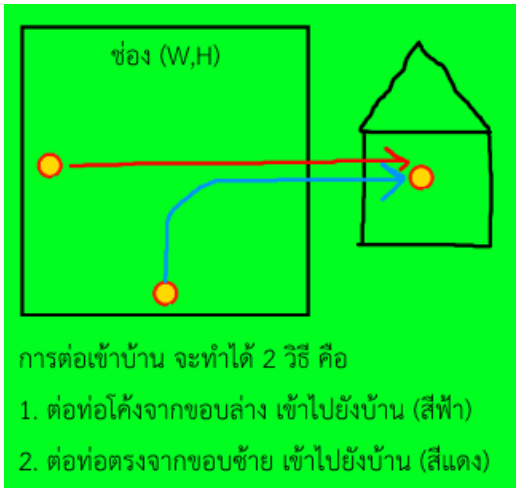
โดยสังเกตว่า node ที่อยู่ด้านซ้ายนี้ จะไม่สามารถต่อไปทิศทางซ้ายได้ (เพราะจะไปทับกับท่อเดิม) หมายความว่า แต่ละ node จะสามารถไปต่อได้ 3 ทิศทาง คือทุกทิศทางที่ไม่ใช่ทิศทางเดิมที่มายัง node นี้



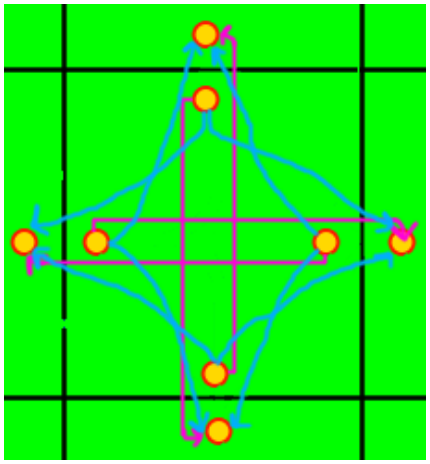




และสำหรับช่องสุดท้ายที่จะเข้าไปที่บ้าน จะใช้ edge ตามรูปข้างล่างนี้



ดังนั้น โจทย์ข้อนี้ก็คือการหา shortest path จาก node ทั้งหมด $4WH+1$ โหนด และมี edge ทั้งหมดไม่เกิน $3(4WH)$ เส้น เราสามารถใช้ Dijkstra's algorithm เพื่อหา shortest path โดยใช้ heap ปกติได้ ซึ่งจะทำงานในเวลา $O((E+V)log(V))$

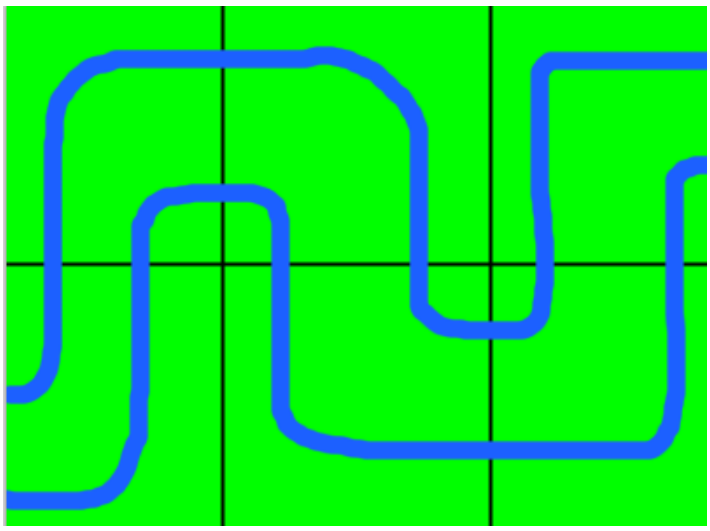


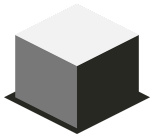
(ตัวอย่าง edge ทั้งหมดจาก 4 node ใน 1 ช่อง สีฟ้าคือท่อโค้ง ส่วนสีชมพูคือท่อตรง)

เคสดักข้อนี้ สำหรับเคสข้างล่างนั้นจะตอบ 6 เพราะเราสามารถโค้งวนกลับไปมาได้ ส่วนหลายคนจะตอบ 202 คือต่อท่อตรงไปแล้วโค้งตรงสุดท้าย คำตอบที่จริงคือตามรูปข้างล่าง

```
Input
100  1  0
3  2
...
...

Output
6
```





4. Love Letter by PalmPTSJ

ชุดทดสอบที่ 1 (10 คะแนน / N,Q ≤ 10)

สำหรับชุดทดสอบนี้ เราสามารถทำ bruteforce ได้ โดยสำหรับแต่ละคำถาม เราสามารถเลือก 4 ตัวมาจากตัวที่อยู่ในระยะ L_i ถึง R_i มา แล้วเช็คว่าทั้ง 4 ตัวนั้น จากซ้ายไปขวาเป็นตัว L, O, V, E หรือไม่ ซึ่งจะใช้เวลาในการทำงาน $O(QN^4)$

ชุดทดสอบที่ 2 (30 คะแนน / Q=1)

สำหรับชุดทดสอบนี้ เราใช้การสังเกตที่ว่า สมมติเราคำนวณมาแล้วตั้งแต่ L_i ถึง x แล้วจะเพิ่มตัว O เข้าไป เราจะได้ว่า O รวมจนถึงตอนนี้ จะมีค่าเท่ากับ O รวมก่อนหน้า บวก 1 และเราจะได้ว่ามีตัว LO รวมเป็น LO ก่อนหน้า แล้วบวกกับ จำนวน L ทั้งหมดก่อนหน้า (เพราะ L ทั้งหมดก่อนหน้าจะมาจับกับตัว O ตัวนี้ได้ เกิดเป็น LO เพิ่มขึ้นมา) จากการสังเกตนี้เราจะได้ว่า

ถ้าเพิ่มตัว ‘L’ : count[L] = count[L]+1

ถ้าเพิ่มตัว ‘O’: count[O] = count[O]+1

count[LO] = count[LO] + count[L]

ถ้าเพิ่มตัว ‘V’: count[V] = count[V]+1

count[OV] = count[OV] + count[O]

count[LOV] = count[LOV] + count[LO]

ถ้าเพิ่มตัว ‘E’: count[E] = count[E]+1

count[VE] = count[VE] + count[V]

count[OVE] = count[OVE] + count[OV]

count[LOVE] = count[LOVE] + count[LOV]

ดังนั้น สำหรับแต่ละคำถาม เราสามารถไล่เพิ่มตัวอักษรทีละตัว ตั้งแต่ L_i ถึง R_i แล้วสุดท้ายคำตอบก็คือ count[LOVE] ใช้เวลาในการทำงาน $O(QN)$

ชุดทดสอบที่ 3 (200 คะแนน / N,Q ≤ 100,000)

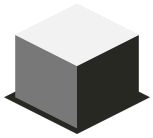
สำหรับชุดทดสอบนี้ เราสามารถใช้ข้อสังเกตคล้ายๆด้านบนได้ โดยเริ่มจากการทำ quicksum หาว่า count[L] ตั้งแต่ 1 ถึง i ใดๆเป็นเท่าใด และทำการหาแบบนี้สำหรับทุกตัว คือ L, O, V, E, LO, OV, VE, LOV, OVE, LOVE ไว้ ซึ่งสามารถทำได้ในเวลา $O(N)$ และใช้หน่วยความจำ $10N$ หรือ $O(N)$ ต่อไปนี้จะเรียก qs[i][c] ว่าเป็นจำนวนตัว c ทั้งหมด ตั้งแต่ช่วง 1 ถึง i ใดๆ

หลังจากนั้น ลองสังเกตว่า ถ้าเราต้องการหาแค่ในช่วง L_i ถึง R_i จะต้องทำอย่างไร โดยจะสังเกตได้ว่า (ในที่นี้ ให้ a=L_i และ b=R_i)

- ถ้า a=1 จะได้คำตอบเป็น qs[b][‘LOVE’] ทันที
- หา ‘E’ ในช่วงนี้ได้จาก count[‘E’] = qs[b][‘E’] - qs[a-1][‘E’]
- count[‘VE’] = qs[b][‘VE’] - qs[a-1][‘VE’] - qs[a-1][‘V’]*count[‘E’] เพราะว่า เราต้องหัก VE ที่เกิดจาก V ข้างนอกช่วง มารวมกับ E ที่อยู่ในช่วงนี้
- count[‘OVE’] = qs[b][‘OVE’] - qs[a-1][‘OVE’] - qs[a-1][‘OV’]*count[‘E’] - qs[a-1][‘O’]*count[‘VE’] ในอันนี้จะคล้ายกับอันบน คือต้องหัก OVE ที่เกิดจาก OV นอกช่วงมารวมกับ E ที่อยู่ในช่วงนี้ และหักที่เกิดจาก O นอกช่วง มารวมกับ VE ในช่วงนี้
- count[‘LOVE’] = qs[b][‘LOVE’] - qs[a-1][‘LOVE’] - qs[a-1][‘LOV’]*count[‘E’] - qs[a-1][‘LO’]*count[‘VE’] - qs[a-1][‘L’]*count[‘OVE’]

เราจะได้ count[‘LOVE’] เป็นคำตอบของช่วงนี้ วิธีนี้จะใช้เวลา $O(Q)$

หมายเหตุ ข้อนี้อาจใช้วิธี Query square root decomposition หรือ MO’s Algorithm ได้ โดยใช้ข้อสังเกตจากการเพิ่ม/ลด ตัวอักษรแต่ละตัวออก แล้วเราจะสามารถจัดลำดับทุกคำถามใหม่ ให้สามารถตอบทุกคำถามพร้อมๆกันในเวลา $O((N+Q)\sqrt{N})$ ได้ ซึ่งผู้แต่งโจทย์ก็ใช้วิธีนี้ในการทำเฉลยตอนแรก สามารถอ่านเพิ่มเติมได้ที่ http://blog.anudeep2011.com/mos-algorithm/



5. Activity Selection by JETHO

เราสามารถหาเซตของคำตอบโดย ทำการเรียงลำดับทุกๆกิจกรรมจาก B น้อยไปหามากและกำหนดหมายเลขของแต่ละกิจกรรมที่เรียงจากค่า B มากไปน้อย เป็น 1 ถึง 2N-1 และหาผลรวม A ของกิจกรรมที่มีหมายเลขเป็น คู่ และ คี่ ถ้ากิจกรรมที่มีหมายเลขคี่มี ผลรวมของ A มากกว่า หรือเท่ากับครึ่งหนึ่งของทั้งหมด ให้เลือกเซ็ตของหมายเลขคี่เป็นคำตอบ (สำหรับกิจกรรมเลขคี่ จะมี N กิจกรรมพอดี) หรือ ถ้ากิจกรรมที่มีหมายเลขคู่มี ผลรวม A มากกว่าหรือเท่ากับครึ่งหนึ่งของทั้งหมด ให้เลือกเซ็ตของหมายเลขคู่ที่รวม หมายเลข 2N-1 (เป็นกิจกรรมที่มี B มากที่สุด) เป็นคำตอบ ซึ่งรวมแล้วจะใช้เวลาการทำงาน $O(N\ log(N))$ โดยเราสามารถพิสูจน์ได้ดังนี้

เมื่อเราเรียงลำดับกิจกรรมจาก B น้อยไปหามากโดย $A_i, B_i \geq 0$

$A_1, A_2, A_3, \dots, A_{2N-1}$

$B_1, B_2, B_3, \dots, B_{2N-1}$

โดย $B_1 \leq B_2 \leq B_3 \leq \dots \leq B_{2N-1}$

ถ้าผลรวมของ A ที่ i เป็นเลขคี่มากกว่าหรือเท่ากับครึ่งหนึ่งของทั้งหมด เราสามารถพิสูจน์ได้ว่า

$B_2 \leq B_3, B_4 \leq B_5, \dots, B_{2N-2} \leq B_{2N-1}$

$B_2 + B_4 + \dots + B_{2N-2} \leq B_3 + B_5 + \dots + B_{2N-1}$

$B_2 + B_4 + \dots + B_{2N-2} \leq B_1 + B_3 + B_5 + \dots + B_{2N-1}$

$B_1 + B_2 + B_3 + \dots + B_{2N-1} \leq 2\,(B_1 + B_3 + B_5 + \dots + B_{2N-1})$

$B_1 + B_3 + B_5 + \dots + B_{2N-1}$ **มากกว่าหรือเท่ากับครึ่งหนึ่ง** ของ ผลรวม B

ถ้าผลรวมของ A ที่ i เป็นเลขคี่น้อยครึ่งหนึ่งของทั้งหมด นั่นหมายความว่า ผลรวมของ A ที่ i เป็นเลขคู่มีค่ามากกว่าหรือเท่ากับครึ่งหนึ่งของทั้งหมด เราสามารถพิสูจน์ได้ว่า

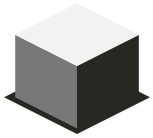
$B_1 \leq B_2, B_3 \leq B_4, \dots, B_{2N-3} \leq B_{2N-2}$

$B_1 + B_3 + \dots + B_{2N-3} \leq B_2 + B_4 + \dots + B_{2N-2}$

$B_1 + B_3 + \dots + B_{2N-3} \leq B_2 + B_4 + \dots + B_{2N-2} + B_{2N-1}$

$B_1 + B_2 + B_3 + \dots + B_{2N-1} \leq 2\,(B_2 + B_4 + \dots + B_{2N-2} + B_{2N-1})$

$B_2 + B_4 + \dots + B_{2N-2} + B_{2N-1}$ **มากกว่าหรือเท่ากับครึ่งหนึ่ง** ของผลรวม B



6. Valentine Party by JETHO

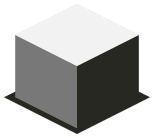
สำหรับโจทย์ข้อนี้เราสามารถใช้โครงสร้างข้อมูลแบบ Binary Index Tree (BIT) ซึ่งสามารถอัพเดทข้อมูลตัวที่ i และหาผลรวมของสมาชิกในช่วง $[A,\ B]$ ใด ๆ ได้ในเวลา $O(log(N))$ เมื่อ N คือจำนวนตัวของข้อมูล

โดยสิ่งที่เราทำนั้นคือการประยุกต์ใช้ BIT โดยเราจะมีข้อมูล N ตัว เขียนแทนด้วย H_i โดย ให้พลังชีวิตของสมาชิกคนที่ i มีค่าเท่ากับ $H_1 + H_2 + ... + H_i$ เมื่อบอสโจมตีสมาชิกในช่วง $[S,\ E]$ ด้วยพลังโจมตี D เราจะอัพเดทพลังชีวิตของผู้เล่นในช่วง โดยการลดค่า H_S เท่ากับ D และ เพิ่มค่า H_{E+1} เท่ากับ D โดย BIT สามารถอัพเดทค่า H_i และหาค่า $H_1 + H_2 + ... + H_i$ ได้ในเวลา $O(log(N))$ ดังนั้น เราสามารถอัพเดทพลังชีวิตของสมาชิกในช่วงต่าง ๆ และหาค่าพลังชีวิตของสมาชิกคนที่ i ได้ในเวลา $O(log(N))$

ประสิทธิภาพโดยรวมคือ $O(P\,Q\,log(N))$ เมื่อ P คือจำนวนครั้งการโจมตีของบอส , Q คือจำนวนคำถามของ EFF และ N คือจำนวนสมาชิก

ข้อมูลเพิ่มเติม Binary Index Tree :

https://www.topcoder.com/community/data-science/data-science-tutorials/binary-indexed-trees/

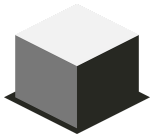


7. Lock On! by JETHO

สำหรับโจทย์ข้อนี้เราต้องหาช่วง $[S, E]$ ซึ่งเป็นอินเตอร์เซคชันของช่วงทุกช่วง หรือถ้าไม่มีช่วงดังกล่าวเมื่อ $S > E$ ในการหาระยะทางน้อยสุดที่ต้องเดินจากจุด P ไปยังช่วงนั้นมีค่าเท่ากับ

- $P - E$ เมื่อ $P > E$
- $S - P$ เมื่อ $P < S$
- 0 เมื่อ P อยู่ในช่วง $[S, E]$

โดยสามารถทำงานได้โดยใช้เวลาการทำงานเป็น $O(N)$

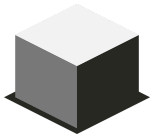


8. Tasty Chocolate by Nottyking

โจทย์ข้อนี้สามารถแก้ได้ด้วย Dynamic Programming (DP) โดยสังเกตว่าโจทย์ข้อนี้มีค่าความอร่อย (T) ไม่เกิน 3000 ดังนั้นเราสามารถเปลี่ยนช็อคโกแลตแต่ละชิ้นให้มีค่าความอร่อยตั้งแต่ 1 ถึง 3000 แล้วดูว่าเมื่อเปลี่ยนแล้วจำนวนครั้งของการเปลี่ยนช็อกโกแลตที่น้อยที่สุดเป็นเท่าไร โดยจะได้สมการ Dynamic Programming ดังนี้

$$DP[i][j] = \min \begin{cases} 1 + DP[i-1][j-1] : \text{เปลี่ยนช็อคโกแลต} \\ DP[i-1][j-1] : \text{ไม่เปลี่ยนช็อคโกแลต} \\ DP[i][j-1] \end{cases}$$

ซึ่งจะใช้เวลาการทำงานเป็น $O(NT)$



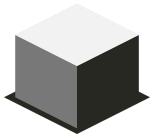
9. FFT's relation by bT33

จากการสังเกตความสัมพันธ์ เราสามารถพิสูจน์ได้ว่า

$$f(n) = fibonacci(n) + \sum_{i=2}^{n} b_i \cdot fibonacci(n-i+1)$$

โดยเริ่มแรก เราจะคำนวณ Preprocess ค่าฟีโบนัชชีเอาไว้ล่วงหน้า แล้วในแต่ละคำถามเราสามารถคำนวณตามสูตร โดยคำนวณเฉพาะ i ที่ $b_i \neq 0$ ทำให้ใช้เวลาการทำงานเป็น $O(k)$ ในแต่ละคำถาม

หมายเหตุ: การคำนวณเพื่อหาคำตอบควรใช้ตัวแปรชนิด Integer 64 bit



10. Papercut by bT33

ในข้อนี้เราสามารถมองการตัดกระดาษให้เป็นลักษณะของ Planar Graph ได้ โดยให้จุดตัดทั้งหมดที่เกิดจากรอยตัดหรือเส้นขอบกระดาษ และจุดปลายของเส้นรอยตัดทั้งหมด เป็น Vertex และให้เส้นที่ลากระหว่างจุดที่เป็น Vertex ทั้งหมดเป็น Edge และเราสามารถคำนวณ Face ได้จากสูตรของออยเลอร์ $V - E + F - C = 1$ โดยคำตอบของโจทย์ข้อนี้จะเป็น $F - 1$ (ไม่นับ Face ที่อยู่นอกขอบกระดาษ)

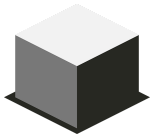
และเนื่องจากในข้อนี้การตัดจะขนานหรือตั้งฉากกันเสมอซึ่งเราอาจใช้วิธีการย่อตารางให้เหลือขนาด

เท่ากับ $O(N^2)$ แล้วใช้ Flood Fill นับจำนวน Component ได้

โดยทั้ง 2 วิธี จะใช้เวลาการทำงานเท่ากับ $O(N^2)$

เพิ่มเติมเกี่ยวกับสูตรของออยเลอร์

- V คือ จำนวน Vertex
- E คือ จำนวน Edge
- F คือ จำนวน Face (คำตอบที่ต้องการสำหรับข้อนี้คือ F-1)
- C คือ จำนวน Connected Component หรือ จำนวนส่วนของกราฟที่แยกกัน



11. Rose Buy by PeaTT

สำหรับโจทย์ข้อนี้จะมีความคล้ายคลึงกับโจทย์ Maximum Subarray แต่จะมีความซับซ้อนมากกว่า เริ่มต้นจะมองให้ง่ายในกรณีที่ K = 1 หากกำหนดให้ DP(j) เป็นผลรวมต่อเนื่องกันจากช่อง i ถึงช่อง j (i ≤ j) ที่สูงที่สุดในทุกวิธีที่เป็นไปได้โดยเราจะได้ว่า

$$DP(j) = Max(DP(j-1) + a_j,\ a_j)$$

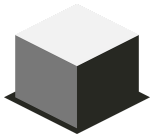
เมื่อมองในกรณีที่ K > 1 กำหนดให้ $DPK(j)$ เป็นผลรวมต่อเนื่องกันจากช่อง i ถึงช่อง j (i ≤ j) ที่สูงที่สุดโดยที่ j - i + 1 ≥ K นั่นคือ $DPK(j)$ จะหาคำตอบได้ในกรณีที่ j >= K เท่านั้น โดยเราจะได้ว่า

$$DPK(j) = Max(DPK(j-1) + a_j, Sum(j))$$

โดยกำหนดว่า $Sum(j)$ คือ ผลรวมต่อเนื่องของตัวเลขจากช่องที่ j - K + 1 ถึงช่อง j (รวมทั้งสิ้น K ช่อง) ซึ่งผลรวมดังกล่าวสามารถหาได้จากสมการว่า

$$Sum(j) = Sum(j-1) + a_j,\ a_{j-K})$$

ซึ่งทั้ง $DPK(j)$ และ $Sum(j)$ เราสามารถหาได้ในลูปเพียงชั้นเดียว ในข้อนี้จึงมีการดำเนินการเป็น $O(N)$ นั่นเอง



12. Hot Head by First4196

เนื่องจากโจทย์กำหนด “ทุก ๆ คู่ของสถานที่ท่องเที่ยวสามารถเดินทางหากันได้ โดยใช้เฉพาะถนนที่ปรับปรุงใหม่เท่านั้น” จึงได้ว่าถนนใหม่จะวางตัวอยู่ในลักษณะของ Tree เราจึงสามารถมองปัญหาเป็นการเลือก Path ย่อย ๆ จาก Tree แล้วนำ Path เหล่านั้นมาเชื่อมกัน ด้วยถนนเก่าภายนอก Tree ซึ่งจะมีอยู่เพียงพอเสมอ ยกเว้นกรณีที่มี Node หนึ่งมีถนนใหม่เชื่อมกับ Node อื่นทุก Node จะทำให้จำเป็นต้องเลือกถนนใหม่อย่างน้อย 1 เส้น

ดังนั้นเราจะได้วิธีการดังนี้

1. เริ่มจากเช็คกรณีพิเศษคือมี Node หนึ่ง มีถนนใหม่เชื่อมกับ Node อื่นทุก Node และถนนเหล่านั้นมี $T_i \geq X$ ทั้งหมดซึ่งสามารถตรวจสอบได้โดยการดูว่ามี Node ที่มี Degree เท่ากับ N-1 หรือไม่กรณีนี้จะได้คำตอบจะเป็น **$(N-2)X + (T_i$ ที่มีค่าน้อยที่สุด$)$**
2. หากไม่ตรงกับกรณีพิเศษจะได้ว่าถนนเก่าภายนอก Tree จะมีอยู่เพียงพอเสมอ ทำให้สามารถหาคำตอบได้ด้วย Dynamic Programming บน Tree โดยเลือก Node หนึ่งเป็น Root แล้ว Breadth-First Search หรือ Depth-First Search ลงใน Tree เพื่อคำนวณคำตอบจาก Leaf ขึ้นมาที่ Root โดยการเลือกถนนใหม่ 1 เส้นจะใช้เวลา T_i และการไม่เลือกจะใช้เวลา X เนื่องจากต้องใช้ถนนเก่า 1 เส้นมาทดแทน จะได้สมการ Dynamic Programming ดังที่แสดงในหน้าถัดไป และได้คำตอบคือ **dp[0][Root]**

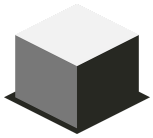
$$\mathbf{dp[1][i]} = \text{sum} + \min \begin{cases} 0 & ; \text{ไม่เปลี่ยนมาใช้ถนนใหม่} \\ \min 1 & ; \text{เปลี่ยนมาใช้ถนนใหม่ 1 เส้น} \end{cases}$$

$$\mathbf{dp[0][i]} = \text{sum} + \min \begin{cases} 0 & ; \text{ไม่เปลี่ยนมาใช้ถนนใหม่} \\ \min 1 & ; \text{เปลี่ยนมาใช้ถนนใหม่ 1 เส้น} \\ \min 1 + \min 2 & ; \text{เปลี่ยนมาใช้ถนนใหม่ 2 เส้น} \end{cases}$$

เมื่อกำหนดให้

- **dp[0][i]** แทน ผลรวมของเวลาที่น้อยที่สุดในที่ใช้ Subtree ที่มี Node i เป็น Root และ**ไม่สามารถ**เลือกถนนใหม่ที่เชื่อม Node i กับพ่อของ Node i ได้ **นั่นคือสามารถเลือกถนนใหม่ที่เชื่อมกับลูกได้อย่างมาก 2 เส้น**
- **dp[1][i]** แทน ผลรวมของเวลาที่น้อยที่สุดในที่ใช้ Subtree ที่มี Node i เป็น Root และ**สามารถ**เลือกถนนใหม่ที่เชื่อม Node i กับพ่อของ Node i ได้ **นั่นคือสามารถเลือกถนนใหม่ที่เชื่อมกับลูกได้อย่างมาก 1 เส้น**
- **sum** = ผลรวมของ **(X + dp[0][j])** สำหรับทุก Node j ที่เป็นลูกของ Node i
  โดย **sum** แทนผลรวมของเวลาที่ใช้ถ้าหากไม่เลือกถนนใหม่ที่เชื่อมกับลูกเลย
- **min1** = ค่าน้อยสุดของ **(T_{ij} + dp[1][j]) - (X + dp[0][j])** สำหรับทุก j ที่เป็นลูกของ i
  โดย **min1** แทนเวลาที่เปลี่ยนไปที่ดีที่สุดหากเปลี่ยนมาเลือกถนนใหม่เชื่อมกับลูกเป็นเส้นแรก
- **min2** = ค่าน้อยสุดอันดับ 2 ของ **(T_{ij} + dp[1][j]) - (X + dp[0][j])** สำหรับทุกลูก j ของ i
  โดย **min2** แทนเวลาที่เปลี่ยนไปที่ดีที่สุดหากเปลี่ยนมาเลือกถนนใหม่เชื่อมกับลูกเป็นเส้นที่ 2

โดยรวมแล้วข้อนี้จะใช้เวลาการทำงานเป็น $O(N)$



13. Dating Laundry by SaBuZa

สามารถสังเกตได้ว่าคำตอบที่ดีที่สุดจะมาจากการเลือกเสื้อที่ใช้เวลาซักน้อยที่สุดที่เลือกได้ ซึ่งสามารถพิสูจน์โดยใช้ข้อขัดแย้ง

Proof :

สมมติให้ d, d' แทนจำนวนวันที่ใช้ในการซักเสื้อ s, s' ตามลำดับ โดยที่ $d < d'$

ให้ c แทนวันที่พิจารณาปัจจุบัน

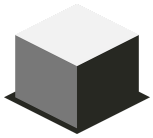
หากเลือกใส่เสื้อ s จะสามารถใส่เสื้อได้อีกครั้งในวันที่ $c + d + 1$

ในขณะที่หากเลือก s' จะสามารถใส่เสื้อได้อีกครั้งในวันที่ $c + d' + 1$

พิจารณากรณีที่ เลือก s' แล้วได้คำตอบที่ดีกว่า s จะเกิดขึ้นก็ต่อเมื่อ $c+d'+1 < c+d+1$

แต่จากที่เราสมมติให้ $d < d'$ เป็นจริง ได้ว่า $c + d + 1 < c + d' + 1$ ทำให้เกิดข้อขัดแย้ง

ดังนั้น การเลือกเสื้อตัวที่ใช้เวลาซักน้อยที่สุดเท่าที่จะเลือกได้ในขณะนั้น จะเป็นคำตอบที่ดีที่สุด



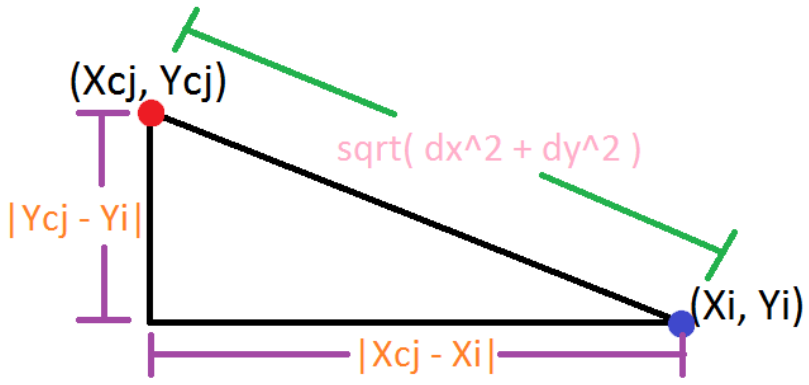
14. Surveillance by SaBuZa

จากโจทย์ถามว่า ด้วยตำแหน่งแต่ละตำแหน่ง X_{cj}, Y_{cj} ซึ่งใช้แทนตำแหน่งของโดรนในระนาบสองมิติ จะต้องใช้ระยะสังเกตขยายออกจากแต่ละจุด X_{cj}, Y_{cj} ให้น้อยที่สุดเท่าไร จึงจะสามารถครอบคลุมพิกัด X_i, Y_i ได้ครบทุกจุด หากแทนระยะสังเกตเป็นวงกลมรอบแต่ละจุด X_{cj}, Y_{cj} ด้วย R

ก่อนอื่น เนื่องจากโจทย์กำหนดว่าพิกัดอยู่บนระนาบสองมิติ เราสามารถหาระยะห่างระหว่างกล้องแต่ละตัว ถึงจุดแต่ละจุดได้ด้วยการใช้ Pythagoras's Theorem ให้ (X_{cj}, Y_{cj}) แทนพิกัดของกล้องแต่ละตัวที่เราสนใจ และ X_i, Y_i แทนจุดสังเกตที่เราสนใจ

รายละเอียด Pythagoras's Theorem (หากทราบแล้วสามารถข้ามไปได้)

กำหนดให้ $dx = |X_{cj} - X_i|$, $dy = |Y_{cj} - Y_i|$ และ sqrt แทน square-root จะสามารถวาดรูปคร่าว ๆ ได้ดังภาพตัวอย่างด้านล่าง



สามารถหาข้อมูลเพิ่มเติมเกี่ยวกับ Pythagoras's Theorem ได้ที่

https://en.wikipedia.org/wiki/Pythagorean_theorem

หลังจากที่เราสามารถหาระยะห่างระหว่างกล้องและจุดสังเกตใดๆได้แล้ว ปัญหาต่อไปคือ เราจะสามารถหาขนาดรัศมี R ที่น้อยที่สุดที่ทำให้จุดสังเกตทุกจุดอยู่ในขอบเขตกล้องทุกตัวได้อย่างไร

Solution 1 : Direct Approach

สังเกตว่า หากสุ่มเลือกพิจารณาจุดสังเกตใดๆมาจุดหนึ่ง เราสามารถหาระยะทางที่น้อยที่สุดจากจุดนั้นไปยังกล้องใดๆ ได้โดยการหาระยะห่างระหว่างจุดที่เราสนใจกับกล้องทุกๆตัว ด้วยการทำงานทั้งหมด C รอบ (C แทนจำนวนกล้อง) แล้วนำมาเปรียบเทียบกัน แน่นอนว่าเราสามารถหาระยะทางได้ด้วยวิธีนี้กับจุดทุกจุด

สำหรับทุกๆจุดสังเกต N จุด ในแต่ละจุดสังเกตทำงาน C รอบ ดังนั้นการทำงานทั้งหมดของโปรแกรมจะเท่ากับ $O(NC)$ โดย จากขอบเขตข้อมูลในโจทย์ $1 \leq N, C \leq 1000$ โปรแกรมจะทำงานอย่างมากที่สุด 1,000,000 ครั้ง ซึ่งทันเวลาแน่นอน

ขั้นสุดท้าย รัศมี R ที่น้อยที่สุดที่จะทำให้จุดสังเกตทุกจุดอยู่ในระยะกล้อง คือ ค่าระยะทางที่มากที่สุดที่เป็นไปได้ของจุดไปยังกล้องที่ใกล้ที่สุด

Solution 2 : Binary Search Approach

ในข้อนี้ ช่วงคำตอบจะอยู่ในช่วง $[0, maxDistance]$ โดย $maxDistance$ คือระยะมากที่สุดที่เป็นไปได้ เราจึงสามารถใช้ Binary Search ในการหาคำตอบ โดยใช้เวลาทั้งหมด $O(NC\ log(maxDistance))$

คำแนะนำ: จริง ๆ แล้ว ในข้อนี้คำตอบจะต้องตอบรากที่สองของเลขจำนวนเต็ม หากคิดค่ารากที่สองก่อนค่อยนำไป Binary Search อาจทำให้ค่าทศนิยมที่ได้มีความคลาดเคลื่อนไปมาก ควรใช้ระยะทางกำลังสองในการ Binary Search หลังจากได้คำตอบแล้วจึงค่อยนำมาคิดรากที่สองภายหลัง