เฉลย

การแข่งขันคณิตศาสตร์และวิทยาศาสตร์ระหว่างโรงเรียนครั้งที่ 18

(18th Triam Udom Mathematics and Science Olympiad)

วิชาคอมพิวเตอร์ รอบที่ 1

วันที่ 9 มกราคม 2563 เวลา 09:00 น. - 12:00 น.

ID โจทย์	ชื่อโจทย์	Time	Memory	คะแนนชุดทดสอบย่อย	รวม (คะแนน)
A-Tulympic	TU Lympics	1 s	256 MB	30 70	100
B-interdimensional-thief	ขโมยของข้ามมิติ	1 s	256 MB	37 63	100
C-zombieland	Zombie Land	1 s	256 MB	15 35 50	100
D-ezproblem	โจทย์สุดง่าย(มั้ง)	1 s	256 MB	25 75	100
E-sneakey-getaway	เนียนให้ผ่าน	1 s	256 MB	30 70	100
F-abgift	ab gift	1 s	256 MB	20 80	100

TU Lympics (100 คะแนน)

1 seconds, 256 megabytes

โจทย์ข้อนี้เป็นการทดสอบความสามารถในการ implement อัลกอริทึม เพราะฉะนั้น สามารถเขียนโค้ดตามที่โจทย์สั่งได้เลย

- รับข้อมูลคะแนนมาเก็บไว้ใส่ array
- สร้าง array ไว้สำหรับเก็บคะแนนของผู้เข้าแข่งขันแต่ละคน (เริ่มต้นที่ 0 คะแนน)
- สำหรับการแข่งขันแต่ละรอบ ให้สร้าง array ที่มีเลข 1 ถึง N ขึ้นมา แทนหมายเลขผู้เข้าแข่งขันในรอบนั้น แล้วเอามาเรียงลำดับจากน้อยไปมาก โดยให้เปรียบเทียบจากคะแนนของหมายเลขนั้นก่อน ถ้าคะแนนเท่ากันจึงเปรียบเทียบตัวเลขผู้เข้าแข่งขัน (หรืออาจจะเก็บเป็น pair ของคะแนนและหมายเลข แล้วใช้ฟังก์ชันเรียงลำดับข้อมูลก็ได้)
- ดำเนินการเพิ่มคะแนนให้ผู้เข้าแข่งขันตามลำดับที่เรียงไว้
- เมื่อทำครบทุกรอบแล้ว ให้เรียงลำดับคะแนนจากมากไปน้อย (วิธีคล้ายกับเรียงลำดับเวลาในขั้นตอนที่แล้ว) แล้วตอบว่าเพื่อนได้อันดับที่เท่าไร

หากใช้วิธีการเรียงข้อมูลที่ไม่มีประสิทธิภาพ เช่น Bubble Sort, Insertion Sort, Selection Sort จะได้เพียงชุดทดสอบกลุ่มที่ 1 เท่านั้น จะได้คะแนนเต็มก็ต่อเมื่อเขียนฟังก์ชันเรียงข้อมูลใน $O(n \log n)$ หรือใช้ฟังก์ชัน sort ที่มีในภาษา C++ อยู่แล้ว

ขโมยของข้ามมิติ (100 คะแนน)

1 seconds, 256 megabytes

โจทย์ข้อนี้เป็นโจทย์ breadth first search หาเส้นทางสั้นสุดในกราฟตารางสองมิติพื้นฐาน โดยเงื่อนไขเส้นทางข้อที่ 2 (กรณี $d_{x,y} \leq 0$) จะต้องแจกแจงอย่างระมัดระวัง นั่นคือแจกแจงเฉพาะ node (x', y') ที่มีระยะห่างจาก (x, y) ไป $|d_{x,y}|$ พอดีเท่านั้น

อนึ่ง ถึงแม้ว่า Time Complexity จะเป็น $O(N^3)$ (เพราะกรณีที่ $d_{x,y} \leq 0$ อาจจะชี้ไปได้ถึง $O(N)$ node) ในทางปฏิบัติ จำนวนเส้นจากแต่ละ node ไม่ได้มีมากครบถึง $N = 1250$

Zombie Land (100 คะแนน)

1 seconds, 256 megabytes

ขั้นแรก จะต้องหาให้ได้ก่อนว่ามีตึกใดเป็นตึกอันตรายบ้าง ตามโจทย์ ตึกอันตรายคือตึกใด ๆ ที่อยู่ในเส้นทางที่สั้นที่สุดจากตึก S ไปตึก E

ในกรณีที่มีเพียงเส้นทางเดียว การหาตึกอันตรายทำได้ไม่ยากนัก เพียงแค่ใช้ Dijkstra's Algorithm เริ่มต้นจากตึก S ไปยังตึก E พร้อมเก็บข้อมูล parent node ไว้ ทำให้สามารถ trace เส้นทางจากตึก E ย้อนกลับมาหาตึก S ได้

ในกรณีที่มีหลายเส้นทาง จะต้องใช้ข้อสังเกตเพิ่มเติมคือ ตึก x จะเป็นตึกอันตรายได้ก็ต่อเมื่อ ระยะทางสั้นสุดจากตึก S ไปตึก x บวกกับระยะทางสั้นสุดจากตึก E ไปตึก x เท่ากับระยะทางสั้นสุดจากตึก S ไปตึก E นั่นคือ $dist(S,x)+dist(E,x)=dist(S,E)$ ดังนั้น ให้ใช้ Dijkstra's Algorithm หาเส้นทางสั้นสุดจากตึก S ไปยังทุกตึก และจากจุด E ไปยังทุกตึก แล้วใช้เงื่อนไขดังกล่าวเพื่อตรวจสอบว่าแต่ละตึกเป็นตึกอันตรายหรือไม่

เมื่อระบุตึกอันตรายทั้งหมดได้แล้ว ให้ใช้ Dijkstra's Algorithm โดยเริ่มต้นจากตึกอันตรายทุกตึกพร้อมกัน (เซต distance เป็น 0 แล้วนำใส่ priority queue ทุกตึกเลย) ไปยังตึกอื่น ๆ ทั้งหมด จะทำให้ได้ระยะทางสั้นสุดจากตึกอันตรายใด ๆ ไปยังตึกอื่น สามารถนำมาตอบคำถามทั้งหมด Q คำถามได้ทันที

โจทย์สุดง่าย(มั้ง) (100 คะแนน)

1 seconds, 256 megabytes

สำหรับโจทย์ประเภทนี้ ควรเริ่มโดยการลองแจกแจงกรณี $n = 1, 2, 3, \ldots$ ก่อนเพื่อดูว่ามีแบบรูปใด ๆ หน้าสัมประสิทธิ์หรือไม่ เนื่องจากเราสนใจเพียงแค่สัมประสิทธิ์ของ x^2 เท่านั้น เราสามารถคูณพหุนามได้โดยข้ามพจน์ $x^3, x^4, \ldots$ ได้เลย

$$\begin{aligned}
f(0,a) &= x - a \\
f(1,a) &= x^2 + (-2a)x + a \\
f(2,a) &= \cdots + (2a + 4a^2)x^2 + (-4a^2)x + a \\
f(3,a) &= \cdots + (4a^2 + 8a^3 + 16a^4)x^2 + (-8a^3)x + a \\
f(4,a) &= \cdots + (8a^3 + 16a^4 + 32a^5 + 64a^6)x^2 + (-16a^4)x + a
\end{aligned}$$

สังเกตว่า $f(n,a)$ จะมีสัมประสิทธิ์หน้า x^2 ทั้งหมด n พจน์บวกกัน โดยเริ่มจาก $(2a)^{n-1}$ ไปถึง $(2a)^{2n-2}$ นั่นคือ $\sum_{i=n-1}^{2n-2}(2a)^i$

จากสูตรอนุกรมเรขาคณิต $1 + r^1 + r^2 + \cdots + r^{n-1} = \frac{r^n - 1}{r-1}$

$$\begin{aligned}
\sum_{i=n-1}^{2n-2}(2a)^i &= \sum_{i=0}^{2n-2}(2a)^i - \sum_{i=0}^{n-2}(2a)^i \\
&= \frac{(2a)^{2n-1}}{2a-1} - \frac{(2a)^{n-1}}{2a-1} \\
&= \left((2a)^{2n-1} - (2a)^{n-1}\right)(2a-1)^{-1}
\end{aligned}$$

การคิดเลขยกกำลังเมื่อ n มีค่ามากถึง 10^{18} สามารถทำได้ด้วยเทคนิค Binary Exponentiation

เนื่องจากโจทย์ต้องการให้เราตอบใน $10^9 + 7$ เราจะต้องใช้ Fermat's Little Theorem ช่วย นั่นคือ $(2a-1)^{-1} \equiv (2a-1)^{p-2} \mod p$ เราสามารถหาค่าของ $(2a-1)^{p-2}$ โดยใช้ Binary Exponentiation เช่นเดียวกันกับก่อนหน้านี้

เนียนให้ผ่าน (100 คะแนน)

1 seconds, 256 megabytes

เพื่อความสะดวก เราจะแปลงข้อมูลใหม่โดยนำข้อมูลทุกตัวลบด้วย 50% ของความจุผู้โดยสาร ดังนั้นเราจะได้ X_i เท่ากับจำนวนผู้โดยสารที่เกินมาจาก 50% (ถ้าติดลบก็คือขาด) ดังนั้น โจทย์ข้อนี้จึงกลายเป็น จงหาวิธีแบ่งข้อมูลออกเป็นส่วนที่ติดกันโดยแต่ละส่วนจะต้องมีผลรวมเกิน 0 ให้ได้จำนวนส่วนมากที่สุด โจทย์การแบ่งส่วนข้อมูล (Partitioning Problem) เป็นโจทย์พื้นฐานใน dynamic programming สามารถแก้ได้ดังนี้

นิยาม dp_i เป็นจำนวนส่วนที่มากที่สุดที่เป็นไปได้ เมื่อพิจารณาปัญหาย่อยที่มีข้อมูลเพียง i ตัวแรกเท่านั้น เราจะต้องเลือกว่าส่วนสุดท้ายควรเริ่มที่ตำแหน่งใด แล้วจึงแบ่งส่วนข้อมูลที่เหลือต่อแบบ recursive หากเลือกให้ข้อมูลช่วงสุดท้ายเริ่มต้นที่ตำแหน่ง $j+1$ (นั่นคือจะเหลือข้อมูลเพียง j ตัวเท่านั้นโดย $j<i$) เราจะได้คำตอบเป็น dp_j+1 โดยมีเงื่อนไขคือ ผลรวมของ X ตั้งแต่ตำแหน่งที่ $j+1$ ถึง i จะต้องมากกว่า 0

เราจะต้องเลือกว่าช่วงสุดท้ายควรเริ่มต้นที่ตำแหน่งใดดี หากเลือกตำแหน่ง j ($1 \le j < i$) ช่วงสุดท้าย (ช่วง j ถึง i) จะต้องมีผลรวมเกิน 0 ถ้าเลือกช่วงนี้เป็นช่วงสุดท้าย จำนวนช่วงทั้งหมดจะเท่ากับ $dp_{j-1}+1$ (ทั้งนี้ กรณีที่ช่วงสุดท้ายครอบคลุม array ทั้งหมด นั่นคือ $j=1$ ให้พิจารณา $dp_0=0$ เพราะจะได้คำตอบเป็น 1 ช่วงพอดี)

เพื่อความสะดวก กำหนดให้ $\sigma(i)=\sum_{j=1}^{i} X_j$ (prefix sum) เราสามารถเขียน recurrence formula ได้ดังนี้

$$dp_i = \max_{0 \le j < i, \sigma(i)-\sigma(j)>0} (dp_j+1)$$

(กำหนดให้ $dp_0=X_0=0$, กรณีที่ไม่มีค่าใดที่ตรงเงื่อนไข $\max$ กำหนดให้ $dp_i=-\infty$)

จากสูตรดังกล่าว สามารถคำนวณคำตอบได้ภายในเวลา $O(n^2)$ ทำให้ได้คะแนนในชุดทดสอบที่ 1 ไป

สำหรับชุดทดสอบที่ 2 ให้สังเกตเงื่อนไขว่า $\sigma(i)-\sigma(j)>0$ คือ $\sigma(i)>\sigma(j)$ ดังนั้น หากเรามอง $\sigma(i)$ เป็น array แล้ว recurrence formula นี้ก็คือการแก้โจทย์คลาสสิก Longest Increasing Subsequence นั่นเอง วิธี greedy และ binary search จะทำให้สามารถแก้โจทย์ข้อนี้ได้ใน $O(n \log n)$ และได้คะแนนเต็มในที่สุด

ab gift (100 คะแนน)

1 second, 256 megabytes

สมมุติว่าเลือกช่วงที่ l ถึง r เราจะสามารถจัดรูปคำตอบได้ดังนี้

$$\sum_{i=l}^{r}\left(a_i \cdot \frac{\prod_{j=l}^{r} b_j}{b_i}\right) = \left(\sum_{i=l}^{r}\frac{a_i}{b_i}\right)\left(\prod_{i=l}^{r} b_i\right)$$

หากเก็บ prefix sum โดย $\sigma(i) = \sum_{j=1}^{i}\frac{a_j}{b_j}$ และ prefix product โดย $\pi(i) = \prod_{j=1}^{i} b_j$ (ควรใช้ตัวแปรประเภท double ในการเก็บข้อมูล อย่าลืมว่าโจทย์ให้ $10000b_i$ มา ไม่ใช่ b_i) เราสามารถหาคำตอบได้โดยการทดลองทุกช่วง $l \le r$ ที่เป็นไปได้ คำนวณคำตอบ $(\sigma(r) - \sigma(l-1))\left(\frac{\pi(r)}{\pi(l-1)}\right)$ หาว่าช่วงใดได้คำตอบมากที่สุด ได้ Time Complexity เป็น $O(N^2)$ ทำให้ได้คะแนนในชุดทดสอบกลุ่มแรก

ชุดทดสอบกลุ่มที่สองจะต้องใช้การสังเกต สมมุติว่าตอนนี้เรามีช่วง l ถึง r อยู่แล้ว เราจะพิจารณาเพิ่มข้อมูลตัวที่ $r+1$ เข้ามาก็ต่อเมื่อมันทำให้คำตอบดีขึ้น เพื่อความสะดวกจะเขียนให้ $s = \sigma(r) - \sigma(l-1)$ (ผลรวมของ $\frac{a_i}{b_i}$ ในช่วงตอนนี้) และ $p = \frac{\pi(r)}{\pi(l-1)}$ (ผลคูณของ b_i ในช่วงตอนนี้)

$$\begin{aligned} sp &< \left(s + \frac{a_{r+1}}{b_{r+1}}\right)(p \cdot b_{r+1}) \\ s &< \left(s + \frac{a_{r+1}}{b_{r+1}}\right)(b_{r+1}) \\ s &< sb_{r+1} + a_{r+1} \\ s(1 - b_{r+1}) &< a_{r+1} \end{aligned}$$

จากข้อมูล $a_i + 10000b_i = 10000 \implies a_i = 10000(1 - b_i)$ เราสามารถเขียนได้ดังนี้

$$\begin{aligned} s(1 - b_{r+1}) &< 10000(1 - b_{r+1}) \\ s &< 10000 \end{aligned}$$

นั่นคือ เราควรขยายช่วงทางด้านขวาไปเรื่อย ๆ ตราบใดที่ผลรวมของ $\frac{a_i}{b_i}$ ในช่วงนั้นน้อยกว่า 10000 อยู่ ดังนั้น การเขียนโค้ดข้อนี้สามารถทำได้ด้วยวิธี two-pointer โดยเริ่มต้น ให้ $l = r = 1$ แล้วลองขยับ r เพิ่มขึ้นเรื่อย ๆ ให้ได้มากที่สุด เมื่อเสร็จแล้วจึงเพิ่มเป็น $l = 2$ แล้วขยับ r อีก, เพิ่มเป็น $l = 3$ แล้วขยับ r อีก, ... เช่นนี้ไปเรื่อย ๆ โดยระหว่างที่ทำให้จดคำตอบที่มากที่สุดไว้

วิธีนี้จะใช้ time complexity เพียง $O(N)$ เท่านั้น ทำให้ได้คะแนนเต็มในข้อนี้