เฉลย โจทย์ 21+4 ข้อ

Bug

[Time : O(N*M)] [Memory : O(N*M)]

[เรื่องที่ใช้ในการแก้ปัญหา : Dynamic Programming]

- Recursive

ไม่ทันเวลา ได้ประมาณเคสแรกเคสเดียว

- Dynamic Programming

จากโจทย์จะเห็นว่ากบสามารถเดินลงมายังใบบัวได้ 3 ทิศทาง แต่เราลองปรับการคิดใหม่เป็น ใบบัวด้านล่างนั้นมี 3 วิธีที่กบจะกระโดดมายังตัวเอง เช่นสมมุติใบบัวอยู่ตำแหน่ง (2,2) กบที่อยู่ตำแหน่ง (1,1) (2,1) (3,1) จะกระโดดมากินแมลงที่ใบตัวเองได้ และจำนวนแมลงที่กบกินได้มากที่สุดนั้นสามารถสร้างเป็นสมการ dynamic programming ได้ดังนี้

$$Dp_{ij} = V_{ij} \text{ เมื่อ } i = 1$$

$$DP_{ij} = \max(DP_{(i-1)(j-1)}, DP_{(i-1)j}, DP_{(i+1)j})$$

โดยกำหนดให้ DP_{ij} หมายถึงจำนวนที่กบกินแมลงได้มากที่สุดจากบรรทัดแรกมายังช่อง (j,i) และ V_{ij} คือจำนวนแมลงในช่อง (j,i) วิธีนี้จะทำให้ได้คะแนนเต็มในข้อนี้

Clock

[Time : O(1)] [Memory : O(1)]

[เรื่องที่ใช้ในการแก้ปัญหาข้อนี้ : if-else พื้นฐาน]

ไอเดียของข้อนี้คือหาว่าใน 1 นาทีนั้นเข็มสั้นหมุนไปได้กี่องศาและเข็มยาวหมุนได้กี่องศา

เข็มสั้นจะหมุนได้ 0.5 องศาต่อนาที และ เข็มยาวจะหมุนได้ 6 องศาต่อนาที

นำมุมทั้ง 2 มาลบกัน แล้วใส่ค่าสัมบูรณ์ดักไว้ เพราะมันจะมีกรณีที่เข็มสั้นนำเข็มยาว และเข็มยาวนำเข็มสั้นด้วย เวลาเอามาลบกันจะทำให้ติดลบ

หลังจากนั้นเช็คมุมที่ได้นั้นว่าเกิน 180 องศาหรือเปล่า ถ้าเกินแสดงว่าเป็นมุมป้าน แต่โจทย์ต้องการมุมที่น้อยกว่า (มุมตรงหรือมุมแหลม) ก็ให้เอา 360 ลบด้วยค่าเดิม ก็จะได้ค่าที่กลับกันที่น้อยกว่า 180 องศา

เสร็จแล้วคูณด้วย 2 เป็นอันเสร็จสิ้น

Color Drop

[Time : O(N*M)] [Memory : O(N*M)]

[เรื่องที่ใช้แก้ปัญหาข้อนี้ : BFS, Queue]

- O(K*k^2)

อัลกอรึทึ่มแรกคือวนลูป K ตัวเพียวๆแล้วก็สร้างหยดน้ำขนาด k x k วิธีนี้จะไม่ได้คะแนนเต็ม

- O(N*M)

สร้าง Queue ที่มีขนาด K ใส่พิกัดที่มีหยดน้ำลงไปตามลำดับ แล้วค่อยๆ pop ข้อมูลใน queue ทีละตัวออกมาโดยตัวที่ pop ออกมาก็ไปสร้างหยดน้ำในพิกัด x y ตามข้อมูลที่ pop ออกมา แล้ว push ข้อมูลลงไปใน queue ใหม่โดยมีเงื่อนไขว่า พิกัดที่ push เข้าไปใหม่นั้นเป็น 4 ทิศทางรอบด้านจากพิกัดเดิม แล้วขนาดความกว้างของหยดน้ำนั้นยังไม่เกิน k อีกทั้งหยดน้ำที่หยดลงไปนั้นต้องเป็นตัวอักษรที่สามารถหยดลงไปได้ มีค่ามากกว่าตัวอักษรเดิมหรือพิกัดตรงนั้นยังไม่เคยมีหยดน้ำหยดอยู่ แล้วค่อย push ลงไป

วิธีนี้ไม่ได้ O(N*M) แบบเป๊ะๆ อาจมีเลื่อมล้ำนิดหน่อย แต่ยังอยู่ในข้อจำกัดที่รับได้เพราะ k เพียงแค่ 30 ซึ่ง worst case ก็ไม่ถึง O(N*M*k) เพราะฉะนั้นถ้าใช้วิธีนี้จะได้คะแนนเต็ม

การ implement โค้ดเพิ่มเติมดูใน source code

Connect

[Time : O(N+Q)] [Memory : O(N)]

[เรื่องที่ใช้แก้ปัญหาข้อนี้ : Graph, Recursive]

สร้าง Graph ตาม input ที่ให้มาก่อน แล้วหลักจากนั้นก็ Recursive จาก node ทุก node ใน Graph โดยมีข้อแม้ว่าเราจะไม่ Recursive node ซ้ำตัวเดิม

โดยขั้นตอนในการ Recursive นั้นเราจะมาร์คเลขให้ node ทุกตัวที่เชื่อมต่อเข้าหากันได้เป็นเลขเดียวกัน (เป็นเลขอะไรก็ได้แต่ต้องเหมือนกัน) และสำหรับ Graph อื่นๆที่ไม่ได้เชื่อมโยงกันเลยก็ให้มาร์คเป็นเลขอื่นโดยห้ามซ้ำกับเลขของ Graph เดิมที่เคย Recursive ไปเลย

จากจุดนี้ทำให้เราได้ข้อมูลมาว่า node ที่โดนมาร์คเป็นเลขเดียวกัน แสดงว่าอยู่ในกราฟเดียวกันหรือพูดอีกนัยหนึ่งคือสามารถเชื่อมต่อกันได้

และ node ที่มีเลขมาร์คไม่เหมือนกันก็แสดงว่าอยู่คนละกราฟและไม่สามารถเชื่อมต่อกันได้

สำหรับขั้นตอนการ Recursive และมาร์คเลขใช้เวลาเพียงแค่ O(N)

ส่วนขั้นตอนสำหรับตอบคำถามใช้เวลาเพียง O(Q) เพียงแค่เช็คตอบว่าเป็นเลขเดียวกันหรือเปล่า

สรุปรวมข้อนี้ใช้ O(N+Q)

Cut The Tree

[Time : O((N+Q) log N)] [Memory : O(N)]

[เรื่องที่ใช้แก้ปัญหาข้อนี้ : Binary Search]

ขั้นตอนแรกเรา sort ความสูงต้นไม้ N ต้นก่อน ซึ่งกระบวนการนี้จะใช้เวลา O(N log N)

ต่อมาถึงเวลาของคำถาม เราสามารถแบ่งต้นไม้ N ต้นเป็นสองพวกเท่าๆกัน พวกแรกคือต้นไม้ที่ต่ำกว่าครึ่งหนึ่งของความสูงสูงสุดในกลุ่มต้นไม้นี้ และพวกสองคือต้นไม่ที่สูงกว่าครึ่งหนึ่งของความสูงสูงสุดในกลุ่มต้นไม้นี้ เราเช็คดูว่า Q นั้นอยู่ในกลุ่มไหน ถ้าอยู่ในกลุ่มน้อยกว่าก็ให้ Q ไปเช็คในกลุ่มน้อยกว่าโดยใช้กระบวนการเดียวกันต่อ ถ้าอยู่กลุ่มมากกว่าก็ให้ Q ไปเช็คในกลุ่มมากกว่า และก็นับรวมจำนวนต้นไม้ฝั่งน้อยกว่าด้วย ซึ่งวิธีนี้เป็นวิธีที่เรียกว่า Binary Search และเวลาทั้งหมดที่ใช้มันจะเป็น O(Q log N)

รวมเป็นเวลา O((N+Q) log N) ทำให้ได้คะแนนเต็มในข้อนี้

Delete [เฉลยยังไม่สมบูรณ์]

[Time : O(N log N)] [Memory : O(N)]

[เรื่องที่ใช้แก้ปัญหาข้อนี้ : Fenwick Tree]

- O(N^2)

ไล่วน for ลบทีละตัว N ครั้งจาก N ตัว วิธีนี้ไม่ทันเวลา

- O(N log N)

กรณีข้อมูลที่เรียงลำดับหรือไม่เรียงนั้นเราถือว่ามีค่าไม่ต่างกัน คีย์หลักของข้อนี้คือตำแหน่งของข้อมูลต่างหาก เช่น 1 2 3 กับ 3 1 2 ตำแหน่งของข้อมูลก็ยังเป็น 1 2 3 เหมือนเดิม เราลบตัวที่ 3 ตำแหน่งข้อมูลตัวที่ 3 ก็จะหายไปแค่นั้น

แต่สำหรับข้อนี้ทริคของมันอยู่ที่เมื่อตำแหน่งเกิดการเปลี่ยนแปลงขึ้น เช่น 1 2 3 เมื่อลบตัวที่ 2 ออกจะเป็น 1 3 4 สิ่งที่เปลี่ยนแปลงคือลำดับของข้อมูลที่อยู่หลังตำแหน่งที่ 2 จะบวกเพิ่มขึ้นทีละหนึ่งทั้งหมด แม้ว่าเราจะวนลูปบวกเลขไปเรื่อยๆจนถึง N ก็ยังใช้เวลา O(N^2) อยู่ดี

วิธีที่จะช่วยประหยัดเวลาข้อนี้คือใช้ Data Structure ที่เรียกว่า Fenwick Tree โดยสามารถอัพเดทตำแหน่งใดๆได้ภายใน log N และสามารถหาผลรวมได้ภายใน log N

เราก็เพิ่มค่าในตำแหน่งที่ต้องการไปอีก 1 เพียงแค่ช่องเดียว ใช้เวลา log N

สำหรับวิธีหาผลรวมก็ใช้หลักการหาผลรวมตั้ง 0 ถึง ตำแหน่งปัจจุบัน ใช้ว่า log N เช่นเดียวกัน

การ implememt เพิ่มเติมดูได้ใน source code

ปล. ข้อนี้ยังไม่สมบูรณ์แบบ เพราะว่ามีโอกาสที่การเปลี่ยนแปลงลำดับจะเกิดการชนกัน เช่น 12345 ลบตัวที่ 4 เป็น 1235 ลบตัวที่ 2 ออก แทนที่จะเป็น 135 แต่กลับเป็น 134 เพราะฉะนั้นวิธีแก้แบบโกงๆกรณีที่มันเกิดการชนกันก็ให้วน while เลื่อนอาร์เรย์ไปเรื่อยๆจนกระทั่งได้ตำแหน่งที่ไม่ชนกัน เช่น 134 ถ้าลบตัวที่ 3

จะเห็นตำแหน่งที่ 4 มันโดนดึงออกไปแล้ว ก็ไปใช้ตัวที่ 5 ถ้าตัวที่ 5 ใช้แล้วก็ดึงตัวที่ 6 ซึ่ง worst case สำหรับกรณีแบบนี้คือ O(N^N log N) ซึ่งพี่เจนเทสเคสไม่ได้เองที่ทำให้ใช้อัลกอนี้ผ่านหมด

แต่พี่ก็คิดวิธีใหม่ได้แล้วโดยที่ worst case ยังเป็น O(N log N) อยู่ แต่ใช้เรื่อง tree ซึ่งต้องเขียนโค้ดใหม่ทั้งหมด เดี๋ยวพี่จะอัพเดท solution ให้ใหม่นะครัช หลังจากอัพเดท solution เสร็จพี่จะเพิ่มเทสเคสอีก 2 เทสเคสสำหรับกรณี worst case นะครัช

Easy Factorial

[Time : O(N)] [Memory : O(1)]

[เรื่องที่ใช้ในการแก้ปัญหา : for-loop พื้นฐาน]

ปัญหาข้อนี้คือถ้าเราหา N! ก่อน จะทำให้ค่าที่ได้เกิน long long และคำนวณผิดพลาด วิธีแก้คือดูว่า r กับ (n-r) อะไรมากกว่า แล้วก็ไล่คูณจาก n(n-1)(n-2)(n-3)....(n-r หรือ n-(n-r)) วิธีนี้จะผ่านได้ฉลุยพร้อมคะแนนเต็ม แล้วก็ไล่หาร r หรือ (n-r) ที่ยังไม่ได้หารให้เรียบร้อย

Expo 1

[Time : O(M)] [Memory : O(1)]

[เรื่องที่ใช้ : for-loop พื้นฐาน]

วน for M ครั้งคูณกันแทนการยกกำลังก็ได้คะแนนเต็มข้อนี้

Expo 2

[Time : O(M)] [Memory : O(1)]

[เรื่องที่ใช้ : for-loop พื้นฐาน, Modulo]

การวน for M ครั้งหาค่าก่อนที่ mod นั้นจะทำให้เลขเกิน int และให้คำตอบที่ผิดพลาด วิธีการแก้คือเพิ่มการ mod ทุกครั้งที่คูณผลลัพธ์เสร็จ ก็จะได้คะแนนเต็มในข้อนี้

Fast Typing

[Time : O(N*(100^2))] [Memory : O(N^100)]

[เรื่องที่ใช้ : for-loop 2 ชั้น]

วนลูป 2 ชั้นโดยสร้างตัวแปรเก็บไว้อาจจะชื่อว่าตัวแปร check = 1; แล้วไล่เทียบ string ตั้งแต่ต้นจนจบคำ ถ้าไม่เหมือนกันให้ check = 0; ซึ่งถ้า check เป็น 1 แสดงว่ามีคำซ่อนอยู่ในข้อความของผู้เข้าแข่งขัน แต่ถ้าเป็น 0 แสดงว่าไม่ได้ซ่อนไว้ แล้วก็นับคะแนนตามจำนวนคำทั้งหมดที่พบก็จะได้คะแนนในข้อนี้

Minesweeper

[Time : O(N*M*9)] [Memory : O(N*M)]

[เรื่องที่ใช้ : for-loop 4 ชั้น]

วนลูป 4 ชั้น ให้ 2 ชั้นแรกเป็นการวนในตาราง N x M และ 2 ชั้นถัดมาคือวนเช็คตาราง 3 x 3 รอบๆ ว่ามีระเบิดทั้งหมดกี่อัน เราสามารถตัดเงื่อนไขเช็คซับซ้อนกรณีที่เป็นหัวมุมหรือขอบได้ โดยการสร้างขอบตารางขึ้นมาและสร้างข้อมูลโดยให้อาร์เรย์เริ่มจาก 1 แทนที่เริ่มจาก 0 การ implement ดูได้ใน source code

Number Evolution

[Time : O(M!)] [Memory : O(M)]

[เรื่องที่ใช้ในการแก้ปัญหา : Recursive]

เราสามารถสร้างฟังก์ชั่น Recursive โดยให้พารามิเตอร์หลักมี 3 ตัวได้แก่ 1.จำนวนผลรวมของเลขแต่ละหลัก 2.เลขที่สร้างขึ้นมา 3.จำนวนหลัก โดยการ Recursive จะ Recursive ไปเรื่อยจนกว่าผลรวมจะเป็น 0 หรือจำนวนหลักจะเกิน M และเพิ่มเงื่อนไขในฟังก์ชั่นคือกรณีที่เป็นหลักแรกให้เริ่มสร้างตัวเลขจาก 1-9 ไม่ใช่ 0-9 การ implement ดูได้ใน source code

Permutation

[Time : O(N)] [Memory : O(N)]

[เรื่องที่ใช้ในการแก้ปัญหา : Math]

ข้อนี้เราเก็บข้อมูลตั้งแต่ 1! จนถึง (N-1)! แล้วก็นำตัวเลข K มาหารด้วย (N-1)! ก็จะได้ตัวเลขที่อยู่ในลำดับ permutation ซึ่งตัวเลขที่ได้มานี้เป็นเลขดิบ การจะหาเลขที่ถูกต้องนั้นมีเงื่อนไขว่า permutation จะไม่ใช้เลขซ้ำกันทุกๆหลัก เพราะฉะนั้นสำหรับหลักแต่ละหลักเมื่อได้เลข K/(N-1)! แล้ว ก็เอามาเช็คในอาร์เรย์อีกตัวที่ทำหน้าที่เก็บค่าไว้ว่าเลขนี้ใช้ไปหรือยัง ถ้ายังก็ดึงมาแสดงผลได้เลย แต่ถ้าใช้แล้วก็กระโดดไปยังเลขถัดไปเรื่อยๆที่ยังไม่ได้ใช้มาแสดงผล

แล้วก็เอาค่า K%(N-1)! แล้วทำกระบวนการข้างต้นเหมือนเดิมจนได้ permutation มาครบทุกตัว

การ implement ดูได้ใน source code

Prime

[Time : O(N*$(1000000000)^{1/2}$)] [Memory : O(1)]

[เรื่องที่ใช้ในการแก้ปัญหา : Math, for-loop]

การเช็คตัวเลข K ใดๆว่าเป็นจำนวนเฉพาะหรือไม่นั้น เราเพียงแค่วนลูปตั้งแต่ 2-รากที่สองของ K ว่ามีตัวเลขใดบ้างที่หารลงตัว หากมีตัวเลขที่หารลงตัวแสดงว่า K ไม่ใช่จำนวนเฉพาะ

Same Number

[Time : O(N)] [Memory : O(1)]

[เรื่องที่ใช้แก้ปัญหา : Bucket]

- O(N log N) solution

วิธีการคือเอาตัวเลขมาเรียงลำดับใหม่จากน้อยไปมากแล้วเทียบกันตัวต่อตัว ซึ่งวิธีนี้จะได้คะแนนรวมแค่ครึ่งเท่านั้น

- O(N) solution

สร้างอาร์เรย์ขนาด 10 ตัวมา แล้วค่อยๆยัดหลักตัวเลขลงไปอาร์เรย์ขนาด 10 ตัวนั้น โดยสมมุติค่าของเลขเป็น 5 ก็เพิ่มค่าอาร์เรย์ตำแหน่งที่ 5 เพิ่มไปอีก 1 และก็ทำอีกครั้งสำหรับเลขตัวที่สองแต่เปลี่ยนจากเพิ่มค่าเป็น 1 เป็นลดค่าลงไป 1 เมื่อเสร็จกระบวนการ ค่าที่ติดลบนั้นหมายถึงว่าจำนวนเลขที่มีอยู่ในเลขแรกแต่ไม่มีอยู่ในเลขที่สองก็นำค่าติดลบทั้งหมดมารวมกันก็จะได้คำตอบที่ต้องการ

Scoring

[Time : O(N)] [Memory : O(N)]

[เรื่องที่ใช้ : for-loop]

กรณีที่เป็นตัวอักษรเดียวไม่ได้มีเครื่องหมายก้ามปูครอบก็คิดคะแนนตามปกติ สำหรับตัวอักษรที่อยู่ในก้ามปูก็ให้จำค่าเอาไว้ว่ามีตัวไหนที่อยู่ในก้ามปูบ้าง เมื่อถึงตัวอักษรปิดของก้ามปู ( ]) ก็บวกเลขที่ต่ำที่สุด ก็จะได้คะแนนเต็ม การ implement ดูใน source code

Sort Number

[Time : O(N log N)] [Memory : O(N)]

[เรื่องที่ใช้ในการแก้ปัญหา : Sort]

เราใช้ quick sort ในการแก้ปัญหาในข้อนี้ สำหรับคนที่ใช้ฟังก์ชั่นสำหรับรูปเช่น qsort หรือ sort ใน STL ก็เพิ่มเงื่อนไขบางอย่างโดยการให้เทียบค่าผลรวมหลักก่อน ถ้าต่างกันก็ return ค่าก่อนเลยถ้าเท่ากันก็ return ค่าเลขไปอีกที

Square

[Time : O(N^2 log N)] [Memory : O(N^2)]

[วิธีที่ใช้ในการแก้ปัญหาในข้อนี้ : Dynamic Programming, Binary Search, Math]

ข้อนี้เป็นข้อที่ค่อนข้างยากมากถึงแม้ว่าจะมีพื้นฐานดีแล้ว เพราะข้อนี้ใช้ 2 เรื่องรวมกันทั้ง Dynamic Programming และ Binary Search

อันดับแรกพูดถึงเรื่องของ Dynamic Programming เราจะเก็บค่าผลรวมตั้งแต่ช่องที่ (1,1) จนถึงช่องที่ (j,i) โดยใช้สมการดังนี้

DP[i][j] = V[i][j] เมื่อ i = 1 หรือ j = 1

DP[i][j] = DP[i-1][j] + DP[i][j-1] – DP[i-1][j-1] + V[i][j]

กำหนดให้ DP[i][j] คือผลรวมตั้งแต่ช่อง (1,1) ถึงช่อง (j,i) และ V[i][j] คือค่าที่อยู่ในช่อง (j,i)

สำหรับการหาค่าผลรวมของพื้นที่สี่เหลี่ยมขนาด k x k เริ่มต้นจากช่อง (j,i) จะใช้สมการ

V = DP[i+k-1][j+k-1] – DP[i+k-1][j-1] – DP[i-1][j+k-1] + DP[i-1][j-1]

เราจะไล่ไปที่ละช่องจนครบทุกช่องสำหรับการค่าผลรวมของพื้นที่สี่เหลี่ยมขนาด k x k โดยถ้าผลรวมเป็น 0 หรือเท่ากับ k x k แสดงว่าเป็นสี่เหลี่ยมที่มีเลข 1 หรือ 0 ท่วมทั้งสี่เหลี่ยม

ปัญหาคือโจทย์ไม่ได้กำหนดสี่เหลี่ยมขนาด k x k มาเพียงแต่ให้หาว่าขนาดสี่เหลี่ยมสูงสุดที่บรรจุอยู่เป็นเท่าไร ซึ่งการที่ไล่สี่เหลี่ยมขนาด 2 x 2 ไปจนถึง N x N แล้วใช้ Dynamic Programming ข้างต้นก็ยังไม่สามารถทันเวลาได้อยู่ดี

ข้อสังเกตุข้อนี้คือ ถ้าในตาราง N x N มีสี่เหลี่ยมขนาด K x K ซ่อน นั่นหมายถึง ตารางนี้ก็มีสี่เหลี่ยมขนาด (K-1) x (K-1) ซ่อนอยู่ด้วย ดังนั้นเราสามารถไล่ตัวเลขจาก K x K ลงมายัง 2 x 2 โดยใส่ break ได้ แต่ก็ยังไม่เพียงพอที่จะทันเวลาอยู่ดี

คีย์หลักคือการใช้ Binary Search โดยหาค่าที่อยู่ในช่วง 2-N ว่ามีสี่เหลี่ยมขนาดใดซ่อนอยู่

- กรณีที่สี่เหลี่ยมขนาด c ไม่ได้ซ่อนอยู่ แสดงว่าต้องมีสี่เหลี่ยมขนาดที่น้อยกว่า c ซ่อนอยู่
- กรณีที่สี่เหลี่ยมขนาด c ซ่อนอยู่ แสดงว่าอาจจะมีสี่เหลี่ยมขนาดที่มากกว่า c ซ่อนอยู่

นำ 2 เงื่อนไขนี้มาสร้างเป็น Binary Search ซึ่งใช้เวลา O(log N) ในการคำนวณ

และแต่ละครั้งของ Binary Search ก็จะทำการค้นหาสี่เหลี่ยมที่ซ่อนอยู่ใช้เวลา O((N-c)^2 + c^2) ประมาณ O(N^2) สรุปเวลารวมแล้ว O(N^2 log N) ทันในเวลา 2 วินาทีพอดี

Sum Fraction

[Time : O(1)] [Memory : O(1)]

[เรื่องที่ใช้ในการแก้ปัญหา : Math (หรม. ครน.)]

เรื่องนี้ต้องใช้เรื่อง หรม. และ ครน. เข้ามาช่วย เพราะว่าหากนำเลข a b c d e มาคูณกันเลยค่าที่ได้นั้นจะเกิน long long อย่างแน่นอน กระบวนการหา หรม. ก็ใช้วิธียูคลิดโดยใช้ Recursive เข้าช่วย (ดูได้ใน source code) ส่วนการหา ครน. สำหรับหลายตัวนั้น เราสามารถคำนวณได้ดังนี้

lcm = a*b/gcd(a,b)

lcm คือ ครน. และ gcd คือ หรม. ของ a,b ก็สามารถหา ครน. ของหลายตัวได้ เมื่อหา ครน. ได้เสร็จสรรพก็นำไปหาร a b c d e ทีละตัวบวกกันก็จะได้แค่ A และ ครน. ก็เป็นค่า B แทน

Teleport (เฉลยยังไม่สมบูรณ์)

[Time : O(N*M)] [Memory : O(N*M)]

[เรื่องที่ใช้แก้ปัญหา : Math, Bucket]

เงื่อนไขที่บอกว่าเราสามารถวาร์ปไปยังช่องที่ตัวเลขที่สูงกว่านั้นเป็นเพียงแค่ตัวหลอกที่ทำให้ข้อนี้ยากขึ้นเท่าตัว แต่ถ้าลองพิสูจน์ โดยพิจารณาว่า กำหนดให้ a b c ใดๆโดยที่ให้ค่าเป็นจำนวนเต็มและ a < b < c โดยสุ่มให้อยู่ช่องใดๆก็ได้นั้น จะเห็นได้ว่าระยะทางจาก a ไป c นั้นจะมีค่าน้อยกว่าหรือเท่ากับ a ไป b ไป c เสมอ ดังนั้นการหาระยะทางที่มากที่สุดระหว่าง 2 ตัวใดๆคือต้องไม่มีเลขมาคั่น 2 ตัวนั้นๆได้

และโจทย์ได้จำกัดเอาไว้ว่าในตารางมีตัวเลขครบทุกตัวโดยไม่ซ้ำกัน และตัวเลขทุกตัวนั้นน้อยกว่า N*M นั่นแสดงว่าในตารางนั้นมีตัวเลขตั้งแต่ 0 – N*M ครบหมดทุกตัว

จากบทพิสูจน์ข้างต้นทำให้เราสรุปได้ว่า การหาระยะทางที่มากที่สุดหาผลรวมของระยะห่างทุกๆ 2 ช่องที่มีตัวเลขต่างกันเพียงแค่ 1 เพราะถ้าต่างกัน 2 จะสามารถมีเลขมาแทรกได้ทำให้ได้ระยะทางมากกว่าเดิม

- O(N*M log N*M)

นำตัวเลขทุกตัวมาเรียงจากน้อยไปมากโดยการ sort แล้วหาผลรวมทั้งหมด

- O(N*M)

สร้าง array ขนาด N*M เก็บพิกัดของตัวเลขทุกตัวเอาไว้ เพราะมีตัวเลขถึงแค่ N*M แล้ววน for ให้ผลรวมทั้งหมด

ปล. ข้อนี้พี่ค่อนข้างมั่นใจเรื่อง proof ระดับหนึ่ง แต่ก็เริ่มลังเลจนกระทั่งเจ็ตมาทักพี่ 555 เอาเป็นใครเก่ง Math เก่ง proof ช่วยพี่ทีว่าไอ้ที่พี่สรุป 3-4 บรรทัดย่อหน้าแรกมันถูกหรือเปล่า

Ten

[Time : O(K)] [Memory : O(1)]

[เรื่องที่ใช้ในการแก้ปัญหา : Bucket]

แปลงตัวอักษร A J K Q ให้เป็น 1 10 11 12 เพื่อให้ง่ายต่อการคำนวณ และก็นำเลขพวกนี้ไปโยงเข้ากับตำแหน่งอาร์เรย์ให้บวกค่าเพิ่มเก็บไว้โดยใช้หลักการ Bucket หลังจากนั้นก็คำนวณหาจำนวนคู่โดยการหา min(จำนวนไพ่ A,จำนวนไพ่ 9) + min(จำนวนไพ่ 2,จำนวนไพ่ 8)... สำหรับกรณี 5 J K Q ก็หาร 2 และปัดเศษออกก็จะได้คำตอบที่ต้องการ

Tree Number

[Time : O(N*Q)] [Memory : O(1)]

[เรื่องที่ใช้ในการแก้ปัญหา : Math, Recursive]

- Recursive

อัลกอนี้ปรึกษาเจ็ตได้เลยครัช 555 ตอนแรกที่คิดโจทย์ไม่ได้มี Recursive อยู่ในหัวเลยนะ หลักการคือคล้าย Binary Search โดยกำหนดเลขสูงสุดเป็น 2^K แล้วค่อยๆหักเลขออกจาก Q ไปเรื่อยๆเมื่อเป็น 0 ก็ให้ return ค่าจำนวนชั้นที่ Recursive ได้ก็จะได้คำตอบ

- Math

แปลงเป็นเลขฐาน 2 แล้วหาว่าเลข 1 ที่อยู่ขวาสุดในเลขฐาน 2 นั้นอยู่ในตำแหน่งไหน แล้วเอา N ตั้งลบก็จะได้คำตอบออกมา

ปล. สำหรับ source code ของพี่ใช้เรื่อง bitwise operator เล็กน้อยใครสงสัยถามได้นะ แต่สามารถเขียนวิธีธรรมดาได้โดยยังอ้างไอเดียเดิมได้อยู่

Turn

[Time : O(1)] [Memory : O(1)]

[เรื่องที่ใช้ : if-else]

นำ k หารเอาเศษด้วย 360 แล้วเช็คว่ามากกว่า 180 หรือไม่ ถ้ามากกว่าให้เอา 360 ตั้งลบ

Village

[Time : O((N*M)^K)] [Memory : O(N*M)]

[เรื่องที่ใช้ในการแก้ปัญหา : Recursive]

ข้อนี้เป็นการฝึก Recursive เพียวๆ อัลกอรึทึ่มจึงสั้นๆไม่มีอะไรมาก โดยการ Recursive เราจะทำการสร้างอาร์เรย์มาเพิ่มตัวหนึ่งแล้วมาร์คเลขเอาไว้ว่าช่องไหนที่สามารถทุบได้กับช่องไหนทุบไม่ได้อาจให้เป็นเลข 1 หรือ 0 ทุกครั้งที่มีการ Recursive ก็จะมาร์คเลขนี้เอาไว้ สำหรับการเขียน code ดูได้ที่ source code