6730084521 Chatrphol O.

{211027 Data Structure Grader}

Dancing Ceremony (d68_q0 [1 Star])

(1 sec, 512mb)

ค่ำคืนเทศกาลงานกาล่าประจำปีของเมืองมีคู่เต้นเข้าประกวดทั้งหมด N คู่ แต่ละคู่ประกอบด้วยผู้นำและผู้ตาม ความสามารถในการเต้นรำของทั้งสองสามารถวัดได้เป็นตัวเลข a_i และ b_i แทนความสามารถในการเต้นรำของผู้นำและผู้ตามของคู่เต้นที่ i ตามลำดับ

กรรมการได้ตั้งคะแนน **"ดัชนีสมดุลจังหวะ"** ขึ้นมาโดยนิยามด้วย $a_i - b_i$ เพื่อวัดว่าคู่เต้นนั้นเด่นไปทางผู้นำมากน้อยเพียงได้ (บวกคือผู้นำเด่นกว่า ลบคือผู้ตามเด่นกว่า) หากคู่เต้นสองคู่มายืนเต้นข้างกันแล้วดัชนีสมดุลจังหวะมีผลรวมกันเป็นบวก จะถือว่าสองคู่นี้เป็น **perfect pair of couples** นั่นคือ สำหรับคู่เต้นที่ i และ j (โดยที่ i < j) จะเป็น perfect pair of couples ก็ต่อเมื่อ

$$(a_i - b_i) + (a_j - b_j) > 0$$

กรรมการต้องการทราบว่าในงานเลี้ยงฉลองคืนนี้ มีจำนวน perfect pair of couples ทั้งหมดกี่คู่

ข้อมูลนำเข้า

บรรทัดแรก มีจำนวนเต็ม N (1 <= N <= 200,000) แทนด้วยจำนวนคู่เต้นทั้งหมด

บรรทัดที่สอง มีจำนวนเต็ม N ตัว $a_1 \ldots a_N$ ($1 <= a_i <= 10^9$) แทนความสามารถในการเต้นรำของผู้นำในแต่ละคู่

บรรทัดที่สาม มีจำนวนเต็ม N ตัว $b_1 \ldots b_N$ ($1 <= b_i <= 10^9$) แทนความสามารถในการเต้นรำของผู้ตามในแต่ละคู่

ข้อมูลส่งออก

มีบรรทัดเดียว เป็นจำนวนเต็มที่แทนจำนวน perfect pair of couples ทั้งหมด

ชุดข้อมูลทดสอบ

- 20% N <= 20
- 40% N <= 5,000
- 20% ดัชนีสมดุลจังหวะของคู่ที่ i จะไม่มากไปกว่าคู่ที่ j เมื่อ i < j
- 20% ไม่มีข้อกำหนดพิเศษอื่นใด

ข้อแนะนำ

คำตอบอาจใหญ่เกินกว่าที่จะเก็บในข้อมูลประเภท int แนะนำให้ใช้ long long แทน

ตัวอย่าง

ข้อมูลนำเข้า	ข้อมูลส่งออก
4 3 7 1 4 3 5 4 2	3 // คู่ 1 – 2, คู่ 1 – 4 และ คู่ 2 – 4 เป็น perfect pair of couples
5 3 6 5 7 9 4 4 1 2 1	10 // ทุกคู่เป็น perfect pair of couples

(มีตัวอย่างต่อหน้าถัดไป)

ข้อมูลนำเข้า	ข้อมูลส่งออก
10 34 28 35 18 23 20 13 29 47 9 23 16 50 34 43 23 45 25 46 22	9
10 29 50 48 50 50 24 22 19 18 25 12 27 24 36 10 8 41 47 27 42	28

Step

1) รับค่า N (pair) , vector A , vector B

2) sort vector

↳ เรียงจาก น.- ไป +

-5	-4	-2	.....	90
id 0	1	2		N

3) Logic เดิม ⇒ Loop ซ้อน Loop แน่นอน

i = 0 | 0 1 2 3 ..... N-1 | outer

ต้องคิด i = 0 → j = i+1

1 2 3 4 5 ... N-1

i+1

ต้องเพิ่ม sort แล้ว

↳ ใช้เวลา N(N-1) ⇒ $O(n^2-n)$

ได้แค่ 60/100 ~ ออกแบบโค้ดไม่ดีพอ

อยู่ใน vector ชื่อ dance_difference

Code

```
// check difference
vector <long long> difference ;
for (int pos = 0 ; pos < dancer_pair ; pos++) {
    difference.push_back( personA[pos] - personB[pos] ) ;
}

// nested for loop , check perfect pair
long long perfect_dance_couple = 0 ;
for (int i = 0 ; i < dancer_pair ; i++) {
    for (int j = i + 1 ; j < dancer_pair ; j++) {
        if (difference[i] > -difference[j]) {
            perfect_dance_couple++ ;
        }
    }
}

cout << perfect_dance_couple ;
}
```

Two-Pointers

Logic อันดาว ⇒ ใช้ O(N) แทน ⇒ ไม่เช็คซ้าย, ขวา (เช็คใจร่าง)

vector dance_difference

-5	-3	-2	-1	0	1	2	4	5	...	90

left_bound → , ← right_bound

Logic : คือ Scope ที่ต้องเช็คลงเรื่อยๆ

```
// sort difference vector
sort( dance_difference.begin() , dance_difference.end() ) ; // sort with iterator fi

// nested for loop , check perfect pair -> NOT WORK (60/100)
// using O(n log n) is safer : binary search or nomral search
long long perfect_dance_couple = 0 ;

// Logic : scope smaller for pair
long long left_bound = 0 ;
long long right_bound = dancer_pair - 1 ;

while (left_bound < right_bound) {
    // scope left down or scope right down
    if (dance_difference[left_bound] + dance_difference[right_bound] > 0) {
        perfect_dance_couple += (right_bound - left_bound) ;
        right_bound-- ;
    }
    else {
        left_bound++ ;
    }
}
```

กำหนด bound L,R ไว้ก่อน ต้องอ้างอิงเสมอ

เช็คว่า "ดีพอ" ?

ถ้าดีพอ distance : ระยะห่าง right - left

ขยับเข้ามา (จากขวา)

ขยับเข้ามา (จากซ้าย)

left_bound , right_bound

Ex มี 5 ตัว : -10 -5 -2 -1 3 4 8

distance

LB → LB (1) , RB

Ex ↳ -10 , 8 ~ ไม่เข้า cond. ~ else ~ ขยับ L เข้ามา

↳ -5 , 8 ~ เข้า cond. > 0 ⇒ ตัวเลขข้างใน ถ้าไกลกันเท่าไร > 0

∴ นับทุกตัวที่อยู่ใน distance ได้เลย ~ ขยับ R เข้ามา

อีก Logic : Binary Search O(N * log(N))

คล้าย Loop ซ้อน Loop แต่ ตัดที่ไม่จำเป็นออก

vector difference

-5	-3	2	1	3	7	8	10

LB → UB ⊕ } รวมกัน Log(N)

แทนเจอตน.ที่เข้าเงื่อนไข ⊕

```
// check difference
vector <long long> dance_difference ;
for (int pos = 0 ; pos < dancer_pair ; pos++) {
    dance_difference.push_back( personA[pos] - personB[pos] ) ;
}

// sort difference vector
sort( dance_difference.begin() , dance_difference.end() ) ; // sort with iterator first and end

// nested for loop , check perfect pair -> NOT WORK (60/100)
// using O(n log n) is safer : binary search or nomral search
long long perfect_dance_couple = 0 ;

for (int i = 0; i < dancer_pair; i++) {

    // create iterator for upper bound then calculate distance
    auto it = upper_bound(
        dance_difference.begin() + i + 1, // เริ่มค้นหาจากตำแหน่งถัดไป
        dance_difference.end(),           // ค้นหาไปจนสุดเวกเตอร์
        -dance_difference[i]              // ค่าที่ใช้เปรียบเทียบ -> check if dance[i] > -dance[j]
        // so find dance[j] by > -dnace[i]
    );

    // calculate difference to end
    perfect_dance_couple += distance(it, dance_difference.end());
}

cout << perfect_dance_couple ;
}
```

เทียบ ตัวแรก

รวมไปจนถึง end

LB , RB (end)

สร้าง iterator มาหาค่า Upper bound

i+1 ที่เข้าเงื่อนไข

นับ distance ไกลกัน ถึง end ที่เข้าเงื่อนไข เช่น