АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА **COUNTM**

Ключовото наблюдение е, че ако фиксираме някоя точка за средна в *М-фигура*, то ще си разделим задачата на 2 подзадачи:

|  |  |
| --- | --- |
| *Фиг. 1* | * Да намерим броя на валидните леви части спрямо тази точка (лявата част спрямо фиксираната за централна т. *C* на *фиг. 1* е означена в **червено**);
* Да намерим броя на валидните десни части спрямо тази точка (дясната част спрямо фиксираната за централнат**.** *C* на *фиг. 1* е означено в **синьо**);

Забележете, че валидните леви и десни части не зависят една от друга; т.е. за всяка валидна лява част може да сложим всяка една валидна дясна част. Тогава отговорът за дадена централна точка ще е просто броят на валидни десни, умножен по броя на валидни леви части. |

Друго полезно наблюдение е, че няма нужда да решаваме поотделно двете подзадачи, тъй като лявата част може да се превърне в дясна като разгледаме симетричния образ на равнината спрямо оста *Оy*. Нека решаваме за дясната част:

Ще сортираме точките и ще ги обхождаме по намаляване на тяхната *x*-координата. Нека текущата точка от обхождането е *T*(*tx, ty*) и разглеждаме всички отсечки, образувани от точки, които вече са били обходени (и следователно са с по-голяма x-координата). Ако вземем хоризонталната права, на която лежи T, то ще имаме 6 възможни ориентации на тези отсечки:

|  |  |
| --- | --- |
| *Фиг. 2* | 1. Отсечка от вид /, която е изцяло над правата;
2. Отсечка от вид \, която е изцяло над правата;
3. Отсечка от вид /, която **пресича** правата;
4. Отсечка от вид \, която **пресича** правата;
5. Отсечка от вид /, която е изцяло под правата;
6. Отсечка от вид \, която е изцяло под правата;

Отсечките от видове (1), (3) и (5) са с неправилна ориентация на двойката точки, защото точката с по-голям *y* има и по-голяма *x-*координата. Отсечките от видове (2), (4) и (6) са с правилна ориентация, но от тях ни интересува единствено броят на вид (4) (в зелено на *фиг. 2*). |

Добре, но как да намерим този брой? Нека първо да определим кое е лесно за намиране:

1. Броят на всички отсечки можем да намерим комбинаторно. Това е броят начини, по които можем да изберем две точки от всички обходени до сега. Ако сме обходили *P* точки до текущата, то ще имаме общо $\frac{P\left(P-1\right)}{2}$ отсечки.
2. Броят на неправилно ориентираните отсечки също можем да намерим сравнително лесно. За целта след като обходим дадена точка, ще натрупваме в променлива *wrong* броя на всички точки, които са над нея. По този начин, преди обхождането на следващата точка в *wrong* ще имаме точно броя на неправилно ориентираните отсечки. Търсенето на броя точки над фиксирана точка може да се реализира със логаритмична сложност чрез съкратено индексно дърво (познато още като *Binary Indexed Tree* (*BIT*)[[1]](#one) или *Fenwick tree*) за сума. Разликата е, че когато вкарваме дадена стойност в дървото, няма да я вкараме на нейната нормална позиция *pos*, а ще я обърнем спрямо бройката на всички позиции, примерно *Q*. С други думи, ще обновим дървото на позиции *Q-pos* вместо на *pos*. Причината за това е, че на нас ни трябва сумата на всички отсечки от дадена позиция до края, докато съкратеното индексно поддържа заявки, търсещи сума от началото до дадена позиция. Ако не сте запознати с идеята, препоръчваме Ви да се разучите структурата и да решите задачите Timus 1028 [[2]](#two) и Timus 1090 [[3]](#three) .
3. Отсечките от вид (6) може да се пресметнат отново чрез съкратено индексно дърво, но малко по-усложнено. За една точка можем да намерим *по-долните* по начина, описан в **II**, но използвайки дървото в чист вид. Сега, с този брой ще обновим друго съкратено индексно дърво. По този начин, ще можем да правим заявки от вида „Колко на брой са всички отсечки от вида \, намиращи се под зададена *y*-координата?“, чиято сложност е отново логаритмична. Ако при заявката използваме *ty*, то получаваме точно отсечките от вид (6).

Ако от бройката намерена в **I** извадим намереното в **II** и **III**, то ще получим сбора на броя на отсечки от видове (2) и (4). Но задачата за намиране на отсечките от вид (2), които трябва да премахнем, е доста сложна. Какво ще правим? Ами, отново ще си преобразуваме задачата до нещо познато. ☺

Можем да намерим броя на всички отсечки над фиксирана *y*-координата като отново използваме съкратено индексно дърво. Така, ако над *ty* има K точки, то те образуват $\frac{K\left(K-1\right)}{2}$ отсечки. Така намерихме сумата на (1) и (2). Нека извадим тази сума от намереното до тук:

 (2) + (4) – [(1) + (2)] = (4) – (1)

Така вече не трябва да търсим точките от вид (2), а тези от вид (1). Как ще ги намерим? Търсим броя на отсечки от вида /, намиращи се *над* дадена y-координата. Това напомня подозрително на **III.** И наистина е същата задача! Разликата е, че когато вкарваме дадена стойност в дървото трябва да използваме описаното в **II** обръщане. Изваждаме намерения брой от полученото по-горе и ни остава само (4), което ни трябва. ☺

Нека сега изчислим и каква е сложността на полученото решение. В началото имаме сортиране, което може да направим оптимално за *O*(*N \* log N*), където N е броят на точките. Впоследствие за всяка точка извършваме много операции, които обаче са предимно заявки и обновявания на съкратени индексни дървета, затова дори и най-бавните от тях са със сложност *log N* (поради което и самото решение има сравнително голяма „скрита константа“ – работи за 0,75 сек. на максимален тест). Така получаваме O(n \* log n) сложност на решението.

*Кратък коментар:*

Голяма част от опитните състезатели още от самото начало се „нахвърлиха“ върху тази задача. Наистина, условието е кратко и ясно, а и самата задача напомня за множество други. За жалост, това доведе единствено до бързо написани грешни решения, които изпускат много от случаите (а пък някои решения някак си успяваха да броят и W-фигурите).

Най-сложната структура, използвана в задачата е съкратено индексно дърво, която е в конспекта за група B (9-10 клас). Но при имплементацията на решението има редица други малки трикове, които я усложняват.

Като упражнение, читателят може да се пробва да реши задачата в случая, когато може да има много точки с еднакви x или y координати. Разсъжденията са подобни, но трябва да се разгледат повече случаи. Примерно решение можете да намерите в допълнителните материали за задачата. Можете да погледнете и решението на Владислав Харалампиев, което е правилно като идея, но е около 3 пъти по-бавно от авторското.

*автор: Ясен Трифонов*

**Препратки:**

[1] <http://community.topcoder.com/tc?module=Static&d1=tutorials&d2=binaryIndexedTrees>

[2] <http://acm.timus.ru/problem.aspx?space=1&num=1028>

[3] <http://acm.timus.ru/problem.aspx?space=1&num=1090>