АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА ЧАСТИЦИ

Да предположим, че частиците от всеки вид са номерирани с числата от *1* до *N*. Нека *x-частица* с номер *i* се изстрелва от точка върху оста *x,* отстояща на разстояние x_i от координатното начало и нейната скорост е означена с v_i. Съответно *y-частица* с номер *j* се изстрелва от точка върху оста *y,* отстояща на разстояние y_j от координатното начало и нейната скорост е означена с u_j.

X-частицата с номер *i* ще пресече траекторията на *y-частицата* с номер *j* в момент y_j/v_i от началото на експеримента. *Y-частицата* с номер *j* ще пресече траекторията на *x-частицата* с номер *i* в момент x_i/u_j от началото на експеримента. За да има сблъсък между тези две частици трябва тези моменти да съвпадат, т.е. да е изпълнено $y_j/v_i = x_i/u_j$ или $x_i*v_i = y_j*u_j$. Второто равенство е по-удобно за използване, тъй като се сравняват цели числа и няма опасност да възникне грешка от представянето на реални числа в аритметика с плаваща запетая.

И така: необходимо и достатъчно условие *x-частицата* с номер *i* и *y-частицата* с номер *j* да се сблъскат и да произведат единица енергия е да е изпълнено равенството $x_i*v_i = y_j*u_j$ и към момента на сблъсъка нито една от тях да не се е сблъскала вече с частица от другия вид. Ако си представим *x-частица* с номер *i*, която се изстрелва от точка x_i върху оста *x*, то тя ще се сблъска с най-близко летящата до оста *x y-частица* с номер *j*, която още не е изчезнала и за която е изпълнено условието $x_i*v_i = y_j*u_j$.

Това води към следната идея за решаване на задачата: сортират се в ненамаляващ ред отделно *x-частиците* и *y-частиците* по отдалечеността на точките, от които се изстрелват, от координатното начало, обхождат се *x-частиците* в сортирания им масив от *1* до *N* (т.е. от частиците, които се изстрелват най-близко до координатното начало към тези, които се изстрелват от най-отдалечени точки) и за *x-частица* с номер *i* се търси най-близко летящата до оста *x y-частица* (с най-малък индекс *j*), за която $x_i*v_i = y_j*u_j$ и *y-частицата* с номер *j* все още не се е сблъскала с друга *x-частица* (не е „изчезнала", произвеждайки енергия). Ако такава *y-частица* се намери, то към брояча на единиците произведена енергия се добавя *1* и съответната *y-частица* се маркира като „изчезнала". Такова решение е със сложност $O(N^2)$ и ще донесе 20 точки. Реализацията на това решение е в **particles_naive.cpp**.

Първото подобрение на тази идея е, запазвайки обхождането на *x-частиците* в реда от наивното решение, за *x-частица* с номер *i* да се ускори търсенето на подходяща *y-частица.* За целта се сменя сортирането на *y-частиците* - те се подреждат първо по произведението y_p*u_p, а при равни произведения - по отдалечеността на точката на изстрелване. Тогава при обхождането на *x-частиците* за всяка *x-частица* двоично се търси най-левият елемент в сортирания масив на *y-частиците*, за който $x_i*v_i = y_j*u_j$ (този най-ляв елемент съответства на *y-частицата* с такава стойност на произведението, която е най-близко до оста *x*). Ако има такъв, то се проверява:

- ако съответната *y-частица* все още не е маркирана като „изчезнала", то е намерена *y-частица*, която ще се сблъска със съответната *x-частица*, ще се добави единица енергия и *y-частицата* трябва да се маркира като изчезнала.
- ако съответната *y-частица* вече е маркирана като изчезнала, то се тръгва от нея към края на сортирания масив и се търси първият елемент след нея, за който все още $x_i*v_i = y_j*u_j$, но съответната *y-частица* не е маркирана като

изчезнала – ако има такъв, то ще се образува единица енергия (със съответното маркиране на *y-частицата).*

Такова решение в някои случаи ще има сложност, близка до *O(NlogN)*, но в други случаи сложността му ще се приближава до $O(N^2)$. То ще донесе 50-60 точки. Реализацията на това решение е в **practicles_1.cpp.**

Следващо подобрение на идеята, което във всички случаи води до решение със сложност *O(NlogN)* се получава, като се съобрази, че в сортирания масив на *y-частиците*, онези от тях, за които y_j*u_j е едно и също, „изчезват" (ако, разбира се има *x-частици*, с които да се сблъскат) последователно отляво надясно (спомнете си сортировката – първо се сортира по произведението, а при еднакви произведения, по разстоянието до оста *x*). Така че, ако в структурата на елементите на масива за *y-частиците* добавим поле, което е *0*, когато частицата вече е „изчезнала", и *1*, ако тя все още съществува, то този масив ще остава сортиран както по произведението y_j*u_j, така и, в рамките на едно и също произведение, по стойността на това поле. Това ни позволява двоичното търсене да го правим по критерий, формиран от двете полета – произведение y_j*u_j и полето, с което се маркира дали дадена *y-частица* е „изчезнала". В това решение, което е с обща сложност *O(NlogN)* има три части – сортиране на *x-частиците*(*O(NlogN)),* сортиране на *y-частиците*(*O(NlogN))* и обхождане на всички *x-частици* с двоично търсене за всяка на подходяща за „сблъсък" *y-частица* (*O(NlogN)).*
Такова решение ще донесе 70-80 точки. Реализацията на това решение е в **practicles_2.cpp.**

Последната стъпка в подобряване на решението е да се откажем от сортирането и обхождането на *x-частиците* по отдалечеността на точките им на изстрелване от координатното начало, а да ги подредим също като *y-частицате* - първо по произведението x_p*v_p, а при равни произведения - по отдалечеността на точката на изстрелване. Тогава, след сортирането на двата масива (сложност *O(NlogN)*), правим класическо асинхронно обхождане, като там където срещнем $x_i*v_i = y_j*u_j$ добавяме единица енергия без въобще да се занимаваме с отбелязването и проверката на това дали дадена частица е „изчезнала". Самата симетричност на това решение по отношение на *x-частиците* и *y-частиците* подсказва, че то трябва да е най-добро. То също е със сложност (*O(NlogN)),*но след сортирането, преброяването на двойките частици, които ще се сблъскат е със сложност *O(N).* Това решение носи 100 точки и реализацията му е в **practicles.cpp.**

Автор: Руско Шиков