АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА
RAIN AGAIN

Задачата Ragain беше модификация на задача, която преди съм давал – името задачата Rain (съответно името Rain Again). Само че оригиналната задача беше в едномерно пространство и се очакваше решение O(N * log * log). Тук задачата беше в 2D и изискваното решение беше със сложност O(N * log) – съответно и нямаше нищо общо с решението на старата задача. Впрочем, задачата в едномерния си вариант има даже O(N) решение, което е впечатляващо (можете ли да го измислите?).

Така, сега към възможните решения. За разлика от повечето други мои задачи, тук дори най-простите решения не са толкова прости. Първо ще модифицираме начина, по който си представяме задачата в еквивалентна такава, която обаче се имплементира по-лесно (по-ефективно):

1. Вместо да падат капки (точки) ще падат правоъгълници с размер W на H и център координатите на точката.
2. Вместо да искаме да няма правоъгълник с размер W на H, който съдържа точка, ще искаме да няма непокрита точка от саксията от падащите правоъгълници.
3. За да е еквивалентно с оригиналната задача трябва и да "покрием" W/2 навътре от вертикалните стени и H/2 навътре от хоризонталните стени на саксията.

Ако помислите малко ще видите, че това е наистина еквивалентно. Може би по-близо до човешката интуция е ако саксията беше кръгла с радиус R и подинтервалите, които искаме да не съдържат точка бяха кръгове с радиус r. Така ако направим радиуса на саксията R-r, точките на кръгове с радиус r, а кръговете на точки, задачата се свежда до това да няма непокрита точка от саксията. Донякъде подобен трик се ползваше в задачите Spectre и MineSweeper.

След като сме променили задачата по този начин, най-лесното, което можем да направим, е да симулираме паданията на капките (вече правоъгълници) и да следим кога всяка точка от саксията става покрита. Тъй като ограниченията са относително големи, това ще работи само за около 7 от 25 теста, тоест ще хване около 30 точки.

Нека докато има непокрито пространство в саксията, вземем произволна точка от него и почнем да я движим надолу и надясно докато опрем в стените на вече сложени правоъгълници или краищата на саксията. Докато тази точка не стане покрита, то трябва да падат още капки (правоъгълници). Ако след като бъде покрита има все още непокрито пространство, отново прилагаме същата процедура, намирайки ъглова точка там.

Разбира се, проверката дали има празно пространство и намиране на ъглова точка не са особено тривиални. Затова ще си улесним живота и няма да го правим. Вместо това ще вземем *всички* такива потенциални точки и ще следим кога те стават покрити.
Всъщност тези точки не са толкова много. Те са или пресичане на два правоъгълинка, или на правоъгълник със стена на саксията, или долния ляв ъгъл на саксията. (Даже може да намалим двойно броя такива точки, като се досетим, че ни интересуват само горни и десни стени на правоъгълниците.). Отново ще прилагаме симулация, но този път при падане на капка (правоъгълник) ще гледаме кои от непокритите "специални" точки той покрива. В момента, в който не остане нито една непокрита "специална" точка знаем, че сме готови (това е последната капка, която ни е нужна).

Броят възможни точки е около O(N*N), а за всяка такава точка трябва да обходим всички N правоъгълника, тоест това решение е със сложност O(N^3). На практика, обаче, ако е имплементирано хубаво освен на много специфични тестове, то се държи като O(N^2) и би хванало близо 50 точки.

Окей, сега да разгледаме и по-бързите алгоритми. За тях дори можем да си позволим да разглеждаме капките като точки, а правоъгълника като правоъгълник – както е в условието на задачата.

По-находчивите състезатели веднага са се сетили да правят двоично търсене по отговора. Дори с него, обаче, не е лесно да проверим дали има или няма правоъгълник в размер W на H измежду точките. Е, има относително стандартен, макар и не особено прост алгоритъм за това – така наречената "помитаща линия" (sweep line) или "метод на метлата", както е по-известен в българската литература.
С две думи, решението е следното:

1. Двоичното търсене ни фиксира някакво число K.
2. Сортираме първите K точки.
3. Почваме да движим една вертикална линия по X (въпросната "помитаща линия").
4. Всяка точка (Xi, Yi) надясно от нея с Xi – W < X реално "пречи" да има празен правоъгълник между помитащата линия и тази точка (по-точно между координатите Yi-H и Yi+H по линията).
5. Ако никъде по линията не може да има правоъгълник, можем да преместим линията на следващия X.
6. Ако X + W > L вече няма как да има празен правоъгълник и казваме, че първите K точки са достатъчни саксията да е напоена.
7. Ако след всички K точки все още X + W <= L, то трябва да преместим лявата граница на двоичното да е K + 1.

Самата помитаща линия можем да реализираме по няколко начина, най-straight-forward от които е да ползваме индексно или интервално дърво. Това обаче би било излишно, тъй като може да се направи и с много по-малко код. Можем да ползваме map в кой Y колко точки има и брояч колко интервали с големина повече от H има по линията. За детайли погледнете авторското решение.

Това решение е O(N*log*log), тъй като имаме един log за двоичното търсене и O(N*log) вътре в него за помитащата линия. Самата помитаща линия е O(N*log), тъй като трябва да сортираме точките и после O(logN) на точка за поддържането на точките по линията. Все пак, това решение няма да хване пълния брой точки, а около 60-70 точки.

Как можем да направим решението дори по-бързо? С хитър (но не особено сложен) трик можем да се отървем от двоичното търсене. Вместо да фиксираме първите K точки и да ги сортираме, ще ги взимаме в реда, в който са ни дадени, и ако са твърде далеч от помитащата линия ще пъхаме в приоритетна опашка. Когато местим линията ще вадим от приоритетната опашка точките с най-малък X и да правим сходно нещо, което правихме в предишния подход. Тъй като всяка точка влиза и излиза най-много по веднъж от приоритетната опашка, то имаме O(N*logN). Самата помитаща линия е също O(N*logN). И тъй като вкарването и изкарването от приоритетната опашка е независимо от ъпдейтването на линията, цялото решение е O(N*logN).

Автор: Александър Георгиев