Agrain
(решение)

Задачата "Agrain" е по-лесна версия на задачата "Ragain", с която трябваше да се борят състезателите в А група. И двете задачи съдържат като подстринг "rain" и "again", ако се чудите откъде идват странните имена. Всъщност и двете задачи са по-сложнен вариант на задачата *Watering*, която дадох преди доста време на друго състезание. За съжаление на тези от вас, които случайно може да са я виждали, решението, което се изискваше там, ще ви донесе малък брой точки.

Нека първо видим как можем да проверим дали саксията е напоена, ако имаме K вече паднали капки. Ще сортираме координатите на капките, след което ще проверим за всяка двойка съседни (в сортирания ред по X) дали има такива, с разстояние помежду си по-голямо от D. Не бива да забравяме да проверим и разстоянието от най-лявата капка до лявата страна на саксията, и от най-дясната до нейния десен край. Това можем да направим по-лесно, ако преди сортирането добавим "фиктивни" капки, които са паднали в позиция 0 и позиция L и проверим само разстоянията между сортираните числа.

Така, сега да разгледаме подходите, които са възможни тук.

Първият и може би най-очевиден е след падането на всяка капка да проверим дали има "ненапоен" интервал. Този подход би бил O(N * N * logN), което би било за съжаление доста бавно (имаме N сортирания, всяко със сложност O(N * logN)). Това решение би взело около 15-20 точки.

Ако сме малко по-съобразителни, ще видим, че вместо да ползваме сортирането от стандартната библиотека, можем ние сами да си пазим точките сортирани, като това би ни премахнало един логаритъм. По-точно, ако до сега сме разгледали K-1 точки (и ги имаме сортирани в масив), то разглеждайки K-тата, можем просто да я "вмъкнем" сред вече сортираните на предходната стъпка. Това, всъщност, си е чисто сортиране чрез вмъкване (insertion sort)! Но все пак чрез него забързваме леко най-тривиалното решение до O(N * N). Това решение би взело около 20-30 точки.

Следващата стъпка в разсъжденията ни е да съобразим, че не е нужно да разглеждаме точките една по една. До някой момент саксията ще е ненапоена, след това ще падне капка, която премахва и последния ненапоен интервал, и оттам нататък тя ще си е все напоена. Често, когато имаме такава постановка, можем да ползваме двоично търсене. Такъв е случаят и сега – с двоичното търсене ще фиксираме някакво число K, което ни казва първите колко капки да вземем, и ще проверяваме дали с тях саксията би била напоена. Ако все още не е, продължаваме двоичното търсене с по-голямо K; ако пък е, продължаваме с по-малко такова. Това решение има O(logN) стъпки на двоичното търсене и вътре в него O(N * logN) за проверка дали фиксираните капки напояват саксията. Това прави общо O(N * logN * logN) и би взело около 40-50 точки.

От тук нататък почват решенията, които изискват по-нестандартни идеи.

Първото по-нестандартно решение е следното. Представете си, че сме малко човече, застанало в началото (краят с координата 0) на саксията. Гледаме кога ще падне капка на разстояние по-малко или равно на D "напред" от нас (тоест надясно от текущата ни позиция). Когато падне такава капка, ние "скачаме" там, където е паднала тя. Интервала, който сме прескочили, няма как да бъде "ненапоен", тъй като

е с размер <= D. Капките, които падат "зад" нас, реално не ни интересуват, тъй като те просто допълнително напояват вече напоен(и) под-интервал(и) на саксията. Продължаваме да скачаме така все по-надясно и по-надясно, докато по някое време се озовем на разстояние <= D от десния край на саксията, в който момент скачаме там и казваме, че няма нужда да падат повече капки.
Тук има един момент, който не уточнихме – какво става, ако сме в дадена позиция P и падне капка в X > P + D? Очевидно не можем да скочим там *сега*, но реално тя ще е напоила някакъв интервал по-нататък и трябва да отчетем това по някое време. Такива капки ще слагаме в приоритетна опашка (пирамида, heap) и ще си "пазим" за по-нататък. Когато падне капка, която е достатъчно близо и скочим на нея, ще гледаме най-близката капка, която вече е паднала (върха на пирамидата), и ще проверяваме дали тя не е паднала достатъчно близко за да можем да скочим върху нея. Това потенциално може да се случи с повече от една капка, като продължаваме да вадим от приоритетната опашка, докато можем. По-формално, алгоритъмът е следния:
При падането на *i*-тата капка с координати X_i докато сме в позиция P:

1) Ако тя е зад текущата ни позиция P (тоест X_i <= P) просто я игнорираме.
2) Ако тя е твърде далеч (тоест X_i > P + D) я слагаме в приоритетната опашка.
3) Ако не е нито назад, нито твърде напред, можем да скочим там.
    a. Обновяваме текущата си позиция P = X_i.
    b. Почваме да вадим вече падналите капки с най-малко X от приоритетната опашка, като потенциално променяме позицията си още няколко пъти (докато капките в приоритетната опашка свършат, или най-близката е на разстояние по-голямо от D).

Реално всяка капка или бива игнорирана, или бива вкарана (и изкарана) по веднъж от приоритетната опашка. Тъй като вкарването и изкарването от структурата данни "пирамида" става със сложност O(logN), то целият алгоритъм ще бъде със сложност O(N * logN). Това решение би взело около 70-80 точки.

Решение със сходна сложност може да бъде имплементирано и с балансирано двоично дърво (примерно set или map от стандартната библиотека), което обаче пази самите ненапоени интервали и ги разделя когато падне капка някъде вътре в тях.

И накрая, решението за 100 точки, изненадващо не ползва някакви сложни структури от данни, а просто масив (и известна доза съобразителност :))

Първо трябва да помислим над следното нещо – ако ни е дадено, че L = 1,000,000, D = 2, N = 100,000, възможно ли е отговорът да не е -1? Относително лесно се вижда, че отговорът е не - както и да са паднали капките, със сигурност ще има под-интервал с дължина над 2, в който да няма капка. Нека обобщим наблюдението: ако ни е дадено L и N, то колко най-малко трябва да е D за да има отговор (тоест той да не е -1)? Отговорът е що-годе очевиден: L / N.

Ще разделим саксията на части с размер D, които ще наричаме "бъкети". Знаем, че те няма да са твърде много – те ще са най-много N+1 (което следва от горното наблюдение - иначе няма да имаме отговор). Първият бъкет ще е под-интервалът (0, D), следващият ще е (D, 2 * D), следващият (2 * D, 3 * D) и т.н., като последният потенциално ще е по-къс под-интервал. Произволна капка с координати X_i попада в бъкет с индекс X_i / D (разделено без остатък). За всеки от тези бъкети ще пазим по две числа – минималната и максималната координата на капка, попаднала в него. За първия бъкет ще твърдим, че още в началото има паднала капка в 0, а за последния – в L (така имитираме краищата на саксията). Ако, например, сме имали саксия с дължина L = 12 и максимално разстояние D = 5, бихме я разделили на три бъкета, съответно (0, 5), (5, 10) и (10, 12). След падането на капки в координатите {5, 2, 7} първият бъкет ще има min = 0, max = 2, вторият ще има min = 5, max = 7, а третият min = 12, max = 12.

Сега ще ползваме идеята за "свързан списък", която би трябвало да познавате много добре, но малко неочаквано влиза в употреба тук. Ще "свързваме" два съседни бъкета когато разликата между максималното число на по-левия и минималното на по-десния стане по-малка или равна на D. В момента, в който всички бъкети станат свързани в едно, вече няма да има под-интервал с размер по-голям от D, в който да не е паднала капка.
Защо? Ако имаме бъкет, в който не е паднала капка, то където и да са падали капките наляво и надясно от него, неговата дължина е D, съответно все още има под-интервал с дължина по-голяма от D, в който да не е падала капка. От друга страна, ако даден бъкет е свързан както наляво, така и надясно, то целият е "напоен". Ако всички бъкети са свързани в едно, то съответно и цялата саксия е напоена.

В това решение обработваме капките в реда, в който ни ги дават. За всяка трябва да намерим бъкета B_i, в който принадлежи. Това става за O(1), чрез формулата $B_i = X_i$ / D, която споменахме по-рано. След това трябва да обновим минималното и максималното число за този бъкет (отново с O(1)). Накрая трябва потенциално да свържем бъкета (ако не е бил свързан) с този от ляво и/или този отдясно.
Тъй като не ползваме "свързаността" на бъкетите за нищо друго, освен да проверим дали всички бъкети са свързани в едно, можем да го правим "наужким" – само да броим колко бъкета вече са били свързани. Очакваме да имаме L / D свързвания общо, затова просто правим един брояч, който в началото е равен на L / D и го намаляме при всяко ново свързване. Когато броячът стане равен на 0 сме готови (цялата саксия е напоена).
Така за всяка капка дъжд имаме O(1) операции, което означава, че решихме цялата задача за O(N). Това решение върви значително по-бързо от другите, тъй като първо е асимптотично по-бързо, и второ, най-сложната структура данни, която ползва, е масив.

Автор: Александър Георгиев