АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА
Graze

Някои от състезателите (по-скоро тези в Б група) може би биха се сетили, че 2010 отново имаше задача с Ели и нейните овчици (задачата Sheep). Интересен факт е, че и за нейното решение се изискваше същата основна идея, както и за решението на тази задача. Както и да е, повечето от състезателите, които са я решавали, вече трябва да са в по-горна група (да не говорим, че самото авторско решение беше грешно, но това е съвсем друга тема).

Задачата Graze изискваше от състезателите да групират множество точки с друго множество точки в едномерното пространство, така че най-голямото разстояние между точка от първото множество и нейният „партньор" от другото множество да е минимално. Това можеше да стане ефективно с поредица от прости алгоритми, водещи до кратък, лесен код, за който трябваше известна съобразителност и внимателност.

Нека първо разгледаме по-простата задача, където N = M и K = 1. Така всяка кошара ще бъде обитавана от точно една овчица, като няма да останат празни кошари. Как бихме групирали овцете с кошарите? Тук е малко по-интуитивно отколкото в оригиналната задача, да сортираме овцете, после да сортираме кошарите, и да сложим най-лявата овца в най-лявата кошара, втората овца във втората кошара и т.н., докато накрая сложим най-дясната овца в най-дясната кошара. Този алчен алгоритъм води до сравнително очевидно оптимално решение. Така в тази опростена задача забелязваме, че никога две овце не пресичат пътя си (най-много се „докосват", ако в началото са на една и съща позиция, или отиват в кошари на една и съща позиция).

Оказва се, че същото правило важи и за оригиналната задача. Нещо повече, първата стъпка, която трябва да направим, е същата – трябва да сортираме овцете и кошарите. Тъй като техния брой беше относително голям за „глупав" алгоритъм се изискваше това да стане със сложност O(N*logN + M*logM). Другият вариант беше състезателите да бъдат достсатъчно наблюдателни, че координатите са сравнително малки и можем да ползваме дори по-бърз и лесен алгоритъм – Counting Sort -- за O(N + M).

След тази стъпка, обаче, новата задача не може да бъде решена с простото Greedy, което ползвахме за опростения й вариант. Това е така, защото може най-лявата овца да не бъде в най-лявата кошара. Нека обаче някой ни е „подсказал", че отговорът на задачата е L. Можем ли да проверим дали това е възможно (тоест дали не са ни излъгали)? С малка модификация, ползвайки същия алчен алгоритъм, се оказва, че можем. Слагаме най-лявата овца в най-лявата кошара, която е на разстояние не

повече от L единици и не е пълна, после втората, третата и т.н. докато сложим всички овце. Ако по някое време някоя от овчиците не може да достигне непълна кошара на разстояние L или по-малко, то казваме, че са ни излъгали. Наистина, тъй като казахме, че овчиците няма да пресичат пътя си (това би водило до неоптимален отговор), тази малко по-сложна алчна стратегия води до разпределяне, което може да провери дали даден отговор е възможен или не.

Но как ни помага това? Все пак никой не ни казва какъв е отговорът. Тук прилагаме техниката „Двоично Търсене", която прави именно това. Пробва различни отговори, и в зависимост от това дали с дадено L разпределянето е възможно или не, намаля пространството на търсене на половина. Тук монотонността на функцията на отговора е очевидна – ако можем да вкараме овцете по кошари на разстояние по-малко или равно на L, то очевидно можем и на разстояние по-малко или равно на L+1. Обратно, ако не можем с L, то не можем и с L-1.

Така задачата се свежда до следните три стъпки:

1. Сортираме координатите на овцете и кошарите
2. Правим Binary Search по отговора (определяйки L)
3. Проверяваме дали това L е валидно (ползвайки алчния алгоритъм)

Така можем бързо да намерим желаното минимално разстояние (а съответно и време).

С каква сложност можем да имплементираме алчния алгоритъм? Единият вариант е за всяка овца да проверим всички кошари, водейки до O(N * M). Това, обаче, за дадените ограничения е доста бавно (би хванало около 50-70 точки). Друг вариант е за всяка овца да намираме съответстващата й кошара с балансирани двоични дървета (map), което би довело до O(N * logM). Това отново е малко бавно и не би хванало пълен брой точки (но пък е сравнително сложно за тази група). Най-бързият вариант е да пазим индекса на най-лявата кошара, която още не е пълна и да итерираме овчиците от ляво надясно. Така ако кошарата е твърде наляво, увеличаваме индекса (докато стане достатъчно близо). Ако кошарата вече е пълна, то отново го увеличаваме. Ако овчицата още не е сложена, а кошарата с текущия индекс е твърде НАДЯСНО, то няма решение с това L. Така постигаме линейно решение, тъй като разглеждаме всяка овца и всяка кошара по най-много веднъж.

Сложността на описания алгоритъм можем да сметнем, като определим сложностите на отделните му части. Сортирането е или O(N * logN + M * logM) или O(N + M). Greedy алгоритъмът е линеен, тоест O(N + M). Двоичното търсене е със сложност O(log(MAX_DIST)), където MAX_DIST е максималното разстояние между овца и кошара, което беше 1,000,000. И тъй като във всяка стъпка от двоичното търсене имаме инстанция на Greedy алгоритъма, то заедно за двата получаваме

O(log(MAX_DIST) * (N + M)). Така общата сложност на алгоритъма се доминира от двоичното търсене и алчния алгоритъм, тоест е O(log(MAX_DIST) * (N + M)).

Остана да проверим частния случай, когато отговорът е -1. Това става, когато броят места в кошарите е по-малък от броя овце. Тоест когато K * M < N. Но да ползваме тази формула може да се окаже подвеждащо. Някои състезатели в тази група със сигурност ще пропуснат факта, че 100000 * 100000 не се побира в 32 битов int (получава се overflow), а има тест, в който M * K овърфлоува и става отрицателно число... макар и да има място за всички овце! Така те ще изпечатат -1 за тест, в който има решение. Алтернатива би била примерно да проверяваме дали N / K <= M или N / M <= K. Тези два начина също не са съвсем коректни, тъй като използваме integer деление, и, например, 10 / 3 = 3, тоест при N = 10, M = 3, K = 3 може да кажем, че има място за всички овце, като всъщност няма. Най-сигурният начин беше просто да сложим дясната граница на двоичното търсене 1,000,001 (число с 1 по-голямо от най-големия възможен отговор) и ако никога не я променим то да установим, че няма отговор.

Автор: Александър Георгиев