АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА
Earrings

Задачата Earrings представляваше стандартна задача от университетските курсове по Изкуствен Интелект – а именно задачата за data clustering. В случая тя е по-специален подслучай, в който елементите трябва да бъдат разбити само в два клъстера (най-простият възможен случай). Както се оказва, за този случай съществува перфектно (оптимално) решение, което намира точен отговор.

Решението е следното. Прави се двоично търсене по отговора (разстоянието между двете най-отдалечени точки от едно и също множество). След като сме фиксирали това разстояние, за всеки две точки можем да кажем дали те могат или не могат да попаднат в едно и също множество. Наистина, ако разстоянието между тях е по-голямо от фиксираното от двоичното търсене, то те *трябва* да принадлежат на различни множества (клъстери).

Така за всяка двойка точки (P_i, P_j), разстоянието между които е по-голямо от фиксираното с двоичното търсене, можем да кажем:

- Ако P_i *е* в първото множество, то P_j *не е* в него.
- Ако P_i *не е* в първото множество, то P_j *е* в него.
- Ако P_j *е* в първото множество, то P_i *не е* в него.
- Ако P_j *не е* в първото множество, то P_i *е* в него.

По-опитните състезатели веднага ще намерят условия като горното за познати. За да приведем в малко по-"компютърен" език горните логически правила ще разбием всяка точка на две: да е в първото множество и да не е в първото множество. На практика това е просто булева стойност true или false, в зависимост дали точката принадлежи или не на първото множество. Така горните правила можем да запишем като:

- P_i -> $!P_j$
- $!P_i$ -> P_j
- P_j -> $!P_i$
- $!P_j$ -> P_i

Така цялата задача (след като сме фиксирали максималното разстояние с двоичното търсене) се свежда до това да проверим дали можем за всяка точка да сложим стойност true или false (което, както казахме, е еквивалентно на това да я разпределим в първото или във второто множество), така че всички условия от горния вид (създадени от точки, които са на по-голямо от фиксираното разстояние) да са изпълнени.

Това е точно задачата за Boolean Satisfiability, при това нейният полиномиално решим екземпляр с най-много две променливи във всеки член (също позната като 2-SAT).

Оттам нататък решението е за всяка стъпка на двоичното търсене да се строи графът от логическите преходи (всяко от горните 4 логически условия е насочено ребро в граф с 2*N върха, където има по един връх за всяка точка и за нейното отрицание). Даже в случая сме улеснени от това, че не се иска да намерим едно възможно разпределение (макар че и това е възможно), а само дали има такова разпределение. Това става относително просто, като се ползва алгоритъм за намиране на силно-свързаните компоненти в насочен граф. Ако една променлива (точка) и нейното отрицание попаднат в една и съща силно-свързана компонента (което на по-човешки език е еквивалентно на това една променлива да трябва да е както true, така и false едновременно), то решение няма. Ако никоя от променливите

не е в една и съща силно-свързана компонента с нейното отрицание, то решение има и продължаваме със следващата стъпка на двоичното търсене.

В случая можем през цялото време да ползваме integer числа (както за двоичното търсене, така и за разстоянията между точките – просто ще ползваме разстоянието на квадрат). Това не променя решението, само го прави по-бързо и по-точно (не се налага да се ползват floating point epsilon-и). Единственото място, където ползваме sqrt() за да намерим истинското разстояние е при печатане на отговора.

Нека изчислим каква е сложността на горното решение. Имаме O(logN) за двоичното търсене и допълнително O(M) за намиране на силно-свързаните компоненти (където M е броят ребра в графа). Но тъй като в най-лошия случай имаме ребро между (почти) всяка двойка точки, то M ~= N*N, откъдето идва сложността O(N^2) за намиране на силно-свързаните компоненти. Така цялата сложност е O(N^2 * logN).

В задачата беше изрично указано, че 35% от тестовете ще съдържат по-малко или равно на 20 точки, което позволяваше bruteforce решение. При него просто пробваме *всяка* възможна конфигурация от точки по множествата и виждаме коя от тях дава оптимален отговор. Това решение е със сложност O(2^N * N).

Ако има начин как по 2 точки да фиксираме едно от множествата, то би съществувало и решение със сложност O(N^3). Идеята му е да пробваме всяка двойка точки, които фиксират едно от множествата (което определя и второто множество) и да ползваме алгоритъма за най-далечна точка за да намерим максималните разстояния. Обикновено алгоритъмът за най-далечна точка е O(N*logN), но неговата най-голяма сложност идва от това, че трябва да сортираме точките. Вместо това в нашия случай бихме могли да сортираме точките предварително, като оттам нататък намирането на изпъкнала обвивка става с O(N). Допълнително, след това намиране на двете най-далечни точки по обвивката също става с O(N). Така след като сме фиксирали две точки можем да намерим с O(N) най-далечните точки във всяко от двете множества, водейки до O(N^3) цялостно решение.

Много заблуждаващо в тази задача е, че на пръв поглед ако фиксираме двете най-далечни точки от едно от множествата, то можем да намерим останалите точки от това множество по алчен начин – като просто вземем всички точки, които са на по-малко или равно от това разстояние от двете фиксирани. Това, обаче, е грешно – някои от взетите по алчен начин точки могат *помежду си* да са на по-голямо разстояние, отколкото двете фиксирани. Това е що-годе интуитивно решение (макар и грешно) и предполагам някои от състезателите ще се подведат по това. Все пак, тъй като решението само по себе си е хубаво, за тях са предвидени около 50-60 точки.

Автор: Александър Георгиев