АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА GATTACA

Gattaca беше задача, която съчетаваше две учудващо стандартни техники – поради това се очакваше да не бъде особено сложна за състезателите и да има поне 2-3 човека с пълен брой точки на нея.

Първото нещо, което състезателите трябваше да забележат, е че отговорът е "монотонен" – тоест ако има стринг, който изпълнява условията от задачата с дължина L, то то ще има и такъв с дължина L-1. Обратно, ако няма такъв с дължина L, то няма да има и с дължина L+1. Когато това е изпълнено, много често един възможен подход е ползването на двоично търсене. Тази задача не е изключение (в случая двоичното търсене е по дължината на отговора), като това наблюдение беше ключово за решаването ѝ (поне при някои от решенията).

След това най-простото, което състезателите можеха да направят, е да проверят за всеки подстринг в *S1* (с фиксираната от двоичното дължина) дали се среща поне *K* пъти в *S2*. Макар и линейните проверки да са *много* бавни, това решение би хванало доста точки – около 50. Това е донякъде учудващо, тъй като на теория сложността на това решение за една единствена стъпка на двоичното е O(N * M * L), където L е дължината на текущо-фиксирания подстринг. Тъй като тази дължина може да расте до min(N, M), на практика цялата сложност на това решение е O(logN * N * M * min(N, M)). Но тъй като освен при много специфични тестове отговорът остава сравнително малък, реално за повечето тестове това решение се държи като O(logN * N * M).

Една възможна оптимизация на горното "глупаво" решение е да забързаме сравненията на стрингове. За да направим това бихме могли да ползваме алгоритъма на Кнут-Морис-Прат и да постигнем чист O(logN * N * M) независимо от тестовете. (Всъщност не съм пробвал да пиша това решение, но изглежда като да е възможно.)

Друга възможна оптимизация на "глупавото" решение е да избегнем търсенето "всеки от *S1* с всеки от *S2*". Това можем да направим, като генерираме всички подстрингове с фиксираната дължина от двата стринга, сортираме ги, и после търсим колко пъти всеки подстринг от *S1* се среща като подстринг в *S2*. Търсенето след като подстринговете са сортирани можем да направим по-умно: логаритмично, ако ползваме ново двоично търсене или дори линейно, ако ползваме плъзгащ се прозорец.

На пръв поглед това решение има сложност O(N * logN + M * logM), тъй като имаме до O(N) подстринга от S1, до O(M) подстринга от S2, като сортиранията са съответно O(N*logN) и O(M*logM). Което, обаче, пропускаме, е че стринговете може да са дълги, съответно проверката кой от два стринга е по-малък може да отнеме до O(min(N, M)) време. Така това решение се оказва O(N * logN * N + M * logM * M). Нещо повече, ако пазим самите подстрингове, то е O(N^2 + M^2) откъм памет, което за дадените ограничения е далеч над това, което можем да си позволим. (Ако направим сортиранията малко по-умно можем да се справим поне с паметта.)

Все пак, с него можем да хванем всички тестове, където отговорът е сравнително къс. За целта трябва да направим една лесна, но не много често срещана модификация на двоичното търсене.

Първо да видим какъв е проблемът: ако имаме тест с *N* и *M* близки до 100000, то на първата стъпка от двоичното търсене бихме пробвали дължина L = 50000, при което бихме се нуждали от над 2 гигабайта памет само за да пазим генерираните подстрингове.

Затова вместо стандартното двоично търсене ще направим такова с увеличаваща се стъпка. Демек първо пробваме L = 1, ако стане с него пробваме L = 2, после L = 4 и т.н. докато стигнем L, за което няма подстринг с такава дължина. След това почваме двоично търсене между това L и L / 2.

Забележете, че ако отговорът е, да кажем, с дължина 10, то първото L при което не успеем ще е само 16 – така няма да се наложи никога да сортираме стрингове с голяма дължина! А запазихме сложността на двоичното O(log). Сложността му (ако отговорът е къс и сравнението на стрингове считаме за O(1) е O(logN * (N * logN + M * logM)).

Разбира се, когато отговорът е с голяма дължина това решение в този си вид няма как да работи.

Стигаме до истинското решение. Проблемът при горното решение е, че:

1) сравнението на дълги стрингове може да отнеме много време;
2) много на брой дълги стрингове изисква много памет.

Тук на помощ идва втората много стандартна техника – хеширане! Вместо да сортираме подстринговете, ще сортираме само техните хешове. Наистина, през повечето време ни интересува единствено дали два стринга са еднакви или различни – с което хешовете се справят чудесно. Допълнително, хешовете са с O(1) памет независимо колко дълги са стринговете, следователно се справяме както с 1), така и с 2).

За да генерираме самите хешове с дадена дължина, трябва да ползваме така наречените "търкалящи се хешове" (rolling hash). При тях чрез модулна аритметика премахваме първият символ от даден хеш и добавяме нов на последна позиция.

За да не се интересуваме от колизии пък можем да ползваме двойно или дори тройно хеширане (хеширания с различна база и модул) на подстринговете, което прави шансът за колизия астрономически малък. В смисъл - наистина. Ако се ползва тройно хеширане (както е в авторското решение), шансът слънчевата радиация да промени бит в паметта и затова да се получи грешен отговор е по-голям, от шанса да се получи колизия.

Така. Връщайки се на решението - тук отново ползваме или двоично търсене или плъзгащ се прозорец за да намерим кои от хешовете на *S1* се срещат поне *K* пъти из хешовете на *S2*. Сложността на това решение е O(logN * (N * logN + M * logM))[1].

Алтернативно май има по-бързо (но по-сложно) решение, базирано на суфиксен масив (suffix array). Все пак прецених, че текущото решение е достатъчно сложно/интересно за целите на състезанието.

[1] Всъщност не е точно така. Понеже трябва да намерим лексикографски най-малкият отговор, имаме допълнително сравнение (все пак) на стрингове, което е линейно. Ако има много кандидати за това и правим тези сравнения често, това решение също е с порядък по-голяма сложност. Ако оставим сравненията само за последната итерация на двоичното (демек когато вече сме намерили правилната дължина) се оказва, че не може да се получат много *различни* кандидати. Така трябва да *не* сравняваме лексикографски стринговете при

итерациите на двоичното, и след като сме намерили дължината на отговора да пуснем още веднъж алгоритъма. Наистина, поне аз не можах да измисля тест, при който да има много кандидати и все пак отговорът да не може да бъде по-голям.

Автор: Александър Георгиев