Анализ на задача lazy

Тагове: greedy алгоритъм, низове

Задачата иска за въведен низ S да се намери лексикографски минималния вариант на S след извършването на точно едно изтриване на елемент от S и запазването на реда на останалите.

Ще спомена решението с "груба сила", което е да обиколим всеки символ и да създадем низа, който се получава ако игнорираме този символ, единствено за да отбележим една възможна опитимизация при него: ако в думата съществуват две еднакви съседни букви, например $...zz...$, няма смисъл да разглеждаме изтриванията на двете – независимио коя от двете букви изтрием, ще получим един и същ низ.

Това решение има *сложност* $O(n^2)$[1] – за всяка буква (от които има n на брой), ще се опитаме да я изтрием, съответно трябва да реконструираме целия низ без тази буква в паметта (което ще отнеме $n-1$ операции, независимо как го правим) и да сравним новия низ от $n-1$ букви със стария най-добър. Казваме, че времевата сложността на алгоритъма е квадратична или както записахме в началото на абзаца $O(n^2)$ – това е груба оценка на броя операции, реалистично не знаем (и не е съществено важно в общия случай) колко операции извършва процесора на компютъра ни, заради това се задоволяваме да следим колко "прости" операции имаме – долепяне на нова буква към края на дума (което ще казваме, че отнема $O(1)$ време, или с други думи – константно), сравнение на два низа (забележете, че както е описано в условието, пък и в реалността, сравнението на два низа отнема време пропорционално на дължината им, тоест в случая ще кажем $O(n)$) и така нататък.

Истински впечатляващото е как да решим задачата за линейно време (или записано $O(n)$ време) – това означава, че обхождайки низа *веднъж* ние ще можем да извлечем информация, която да ни каже коя буква трябва да изтрием (и какъв остатъчен низ да получим).

Интуитивно можем да се убедим, че е доста лоша идея да изтрием буква S_i, за която следващата буква S_{i+1} е по-голяма в азбучния ред. Визираме следния случай: нека $S =$ `dfbgg` (което е примерният тест от условието). Ако изтрием първият символ `d`, ще останем с `fbgg`, но без да правим каквито и да е допълнителни заключения, веднага виждаме, че влошихме ситуацията – можеш на първата позиция да оставим `d`, да изтрием от втората позиция `f` и да получим `dbgg`, което е лексикографски по-малък низ от `fbgg` (по случайност е и самият отговор в случая, въпреки че това не трябва да ни дава лошата идея, че отговорът винаги е да изтрием първият или вторият символ). Все пак резонно (и вярно) е състезател да заключи, че след като позициите i, такива че $S_i < S_{i+1}$ не са добри опции, ще се интересуваме единствено от позициите, за които $S_i > S_{i+1}$, а понеже сравнението е лексикографско ще се интересуваме конкретно от първата такава позиция.

Горният абзац предлага добра интуиция за решението и заедно с формата на състезанието, в който почти няма наказание за грешен събмит, печелившата стратегия за състезател видял тази идея е да я напише бързо и да я прати на системата. Все пак изложените твърдения дотук не представляват *доказателство*, което да ни убеждава че в абсолютно всеки случай правилният ход е да намерим първата позиция $S_i > S_{i+1}$ и да изтрием нея (най-малкото такава впрочем може да няма – тогава трием последната буква). Нека поправим грешката си и да докажем твърдението:

Твърдение 1. *За да получим лексикографски минимален низ чрез едно изтриване от S, трябва да изтрием първата позиция за която $S_i > S_{i+1}$ или ако не съществува такава – последният символ на низа.*

[1] Wikipedia страницата за Big O notation

Доказателство. Нека първо разкараме частният случай, където не съществува i, такова че $S_i > S_{i+1}$. Тогава би следвало, че за всяко i всъщност $S_i \leq S_{i+1}$ – нагледно низът ни има формата `aaa..abb.bbb.ee..ee...z..zz`, като естествено някои интервали от равни букви може изцяло да липсват. Тогава ако изтрием буква, която е равна по стойност на S_0, може просто да сме изтрили S_0, така че ще разгледаме просто този случай. Да кажем преди триенето на S_0 низът се е състоял в началото си от k копия на S_0 (и след това други, по-големи от S_0 букви). Ако сме изтрили S_0 ще останем само с $k-1$ копия на буквата – ще означим този вариант на S със S^1. Така този низ тривиално няма как да е най-малък, защото ако изтрием вместо това някоя по-късна буква ще запазим k-те копия на буквата (нека такъв вариант наречем S^2) и по-точно k-тата буква в S^2 ще бъде S_0, а в S_1 ще бъде следващата, по-голямо буква.

Аргументът се повтаря и в случай, че изтрием не S_0, а някоя по-късно срещаща се буква в S, заради това няма за го повтаряме. В остатъка от доказателството ще приемем, че съществува поне едно i, такова че $S_i > S_{i+1}$.

За да не се повтаряме още сумати пъти, нека кръстим с i_0 първата позиция за която следващата съдържа по-малка буква, с други суми $S_{i_0} > S_{i_0+1}$ и за всяко $j < i_0$ е изпълнено, че $S_j \leq S_{j+1}$.

Да допуснем, че сме сгрешили – не е i_0 отговорът, а някой друг индекс i.

Нека първо $i_0 < i$, тоест има някакъв следващ символ, който е по-добра оферта за изтриване. Ако изтрием позиция i_0, на позиция i_0 ще седи символа S_{i_0+1}, а ако изтрием позиция i на позиция i_0 все още ще седи S_{i_0}. Но ако се върнем на дефиницията на i_0, това е точно такъв индекс, за който е вярно, че $S_{i_0} > S_{i_0+1}$ – значи няма как да твърдим, че триейки i сме получили лексикографски по-добър отговор. Така този случай е невъзможен.

Нека сега разгледаме случая, в който $i < i_0$. Понеже казахме, че i_0 е първата позиция, за която $S_{i_0} > S_{i_0} + 1$, следва че $S_i \leq S_{i+1}$. Подобна на горния случай, ще разгледаме какво се случава с позиция i при двете опции – да изтрием i или да изтрием i_0. Ако изтрием S_i, на позиция i ще дойде S_{i+1}, което е в най-добрия случай равно на S_i, но може и да е по-голямо. Ако изтрием $i_0 > i$, обаче символа на позиция i ще е точно S_i. Следователно няма как триейки позиция по-ранна от i_0 да получим по-добър в лексикографски смисъл низ. □

Въпреки че удвоихме дължината на анализа, именно горното доказателство е стилът разсъждение, което би следвало да убеди състезател, че алгоритъмът работи в 100% от случаите (а не само в 10-те представени в системата). Забележете, че предвид че $n \leq 200\,000$ и низът се състои от 26-те букви на латинската азбука, по груби сметки има $27^{200\,000}$ възможни входа на задачата (което е число с $300\,000$ цифри).

В краткосрочен план, това доказателство е полезно на състезателя, защото след достатъчно тренировка, той (или тя) ще бъде способен бързо да нахвърля еквивалент на него, който дори да не е толкова формален, да го убеди във верността на твърденията му.

В дългосрочен план, запознаването с такива доказателства ще въведе състезателя в по-дълбокия свят на формалната математика и по-конкретно формалната верификация, която става все по-важна в професионалната сфера на софтуерното инженерство – именно подобни методи могат да ни уверят, че ключови софтуерни компоненти като операционна система, банково приложение или интернет браузар нямат грешки, които могат да доведат до сериозни пробиви в сигурността.

Автор: Иван Лупов