**АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА**

**Lowest Hash**

Lowest Hash е относително лесна задача за A група, съответно очакванията ми са 8-10 (ако не и повече) човека да я направят за 100 точки (разбира се, в зависимост дали е дадена на самия пролетен турнир или на контролата).

Ниските ограничения на дължина на стринговете (едва до N ≤ 28) трябваше веднага да направят впечатление на състезателите и да ги насочи към това да търсят решение, което се нуждае от такива малки ограничения. Двете най-стандартни неща са:

* [Динамично оптимиране по битова маска](http://www.informatika.bg/lectures/dynamic-programming-part3) (при ограничение около 20)
* [Meet-in-the-middle](http://www.informatika.bg/lectures/meet-in-the-middle) (при ограничение около 40)

В случая, ограниченията бяха някъде по средата (pun intended), но малко по-съобразителните биха забелязали, че това не е и стандартната постановка нито на битова маска, нито на meet-in-the-middle, където всеки елемент можем или да вземем или да не вземем - тоест имаме по две възможности на елемент. В случая имахме по три възможности, тоест около 328 = 22,876,792,454,961 възможни резултата. Това е много, но не *абсурдно* много. Meet-in-the-middle звучи като точното нещото, което да ни трябва, тъй като разделяйки задачата на две, получаваме 314 = 4,782,969 резултата за всяка от двете части, което е напълно резонно в meet-in-the-middle решение.

Разбира се, трябва и да можем относително бързо да комбинираме резултат от едната половина с оптималния от другата. При приблизително 5,000,000 елемента от всяка страна, трябва това комбиниране да става за не повече от O(log) сложност, за да можем да имаме достатъчно малък брой операции за дадените ограничения по време. Дори с логаритмично комбиниране на резултати нещата са на границата - при 5,000,000 елемента, за всеки от които правим логаритъм операции получаваме 5000000 \* log(5000000) = 5000000 \* 23 = 115,000,000, което е кажи-речи максимума, който можем да си позволим. Както ще видим по-нататък, в тази задача константата не е ниска, тъй като трябва да сортираме елементите от едното множество (веднъж 500000 \* log(5000000)) и после за всеки елемент от другата страна да направим двоично търсене или upper\_bound – което е още толкова. При това оперираме с голямо парче памет (10,000,000 long long-а, което са 80 мегабайта и не е дори близо до това да се събере в кеша на процесора).

Сега да видим и как ще направим самото решение. Първо, ще разделим стринговете на две части (които ще наричаме "лява" – с първите N/2 + N%2 символа на низовете и "дясна" – с останалите). За всяка част ще генерираме всички възможни получени стрингове и ще намерим техните хешове (реално не ни трябва да генерираме самите стрингове, а само хешовете им, което забързва значително решението и изисква много по-малко памет). Генерирането можем да направим с проста рекурсия, която на всяка стъпка решава коя от трите възможни букви да ползва и я добавя към хеша.

След като имаме генерирани (хешответе на) стринговете, трябва да видим как можем да съчетаем начало на стринг от лявата страна с край на стринг от дясната. Всеки от хешовете от лявата страна можем да умножим по магическо число, което бихме направили в хеширащия алгоритъм, добавяйки буквите от дясната страна. Тъй като дължината на стринговете е фиксирана (и, съответно, константна както за стринговете в лявата страна, така и за тези в дясната), можем да умножим всеки хеш от лявата по 127N/2, като така симулираме добавянето на останалите символи, без експлицитно да ги добавяме. След като направим това, комбинирането на хеш от лявата страна с хеш от дясната става като просто ги съберем.

Забележете, че тъй като самите хешове от лявата страна могат да бъдат до 1000000000000037 (250), то не можем просто да умножим всяко число по 127N/2 – биха ни били нужни числа от порядъка на 2100 за междинните сметки, което не се поддържа от стандартните типове (ограничени до 264). Вместо това можем да се възползваме от това, че стринговете са относително къси, тоест N/2 може да достигне едва до 14 – не пречи да извършим умножението итеративно, като така не нарушаваме сложността на алгоритъма си (тъй като в случая 14 < log(5000000)). При по-големи ограничения алтернативно можеше да ползваме [бавно умножение](http://www.informatika.bg/lectures/fast-exponentiation), но то не беше нужно за тази задача.

Добре, имайки хеш от лявата страна, как трябва да го комбинираме с хеш от дясната страна, така че резултатът да е възможно най-малък? Нека хешът от лявата страна е H. Тогава търсим най-малкият хеш, по-голям или равен на MOD – H от дясната. Това може да стане с функцията upper\_bound или просто с двоично търсене, което сами си имплементираме. Тук трябва да внимаваме, тъй като е възможно да няма хеш от дясната страна, който да е по-голям или равен на MOD – H, в който случай трябва да вземем най-малкия елемент от дясната страна.

За да направим двоично търсене, трябва да сортираме всички хешове от дясно. Също така за всеки елемент от лявата страна трябва да намерим оптималния му match отдясно. Така имаме О(3N/2 \* log(3N/2)) за сортирането и още O(3N/2 \* log(3N/2)) за двоичните търсения. Алтернативно, можем да сортираме и двете страни и да ползваме два pointer-а, постигайки същата сложност.

Трябва да обърнем внимание на частния случай, в който N = 1, тоест една от двете групи (лявата или дясната, в зависимост от имплементацията) ще е празна. Това е тривиално, но все пак нещо, което не трябва да пропускаме.

Задачата предоставяше и няколко варианта да хванем частични точки. Един вариант беше просто да brute force-ваме възможните стрингове, докато имаме време. С такова решение бихме хванали около 30 точки. Друг вариант е да започваме с random стринг (генериран от A, B, и C) и да правим прост hill climbing – да сменяме буква по буква от него, докато хешът намалява. В зависимост от имплементацията, това можеше да донесе между 40 и 50 точки на състезателите. Динамично с map също беше вариант, макар и лош такъв (носи 25 точки).

*Автор: Александър Георгиев*