АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА
Consistency

Задачата Consistency беше мотивирана от проблем от реалния живот (файловете в torrent-ите наистина биват проверявани за консистентност по подобен начин). Нещо повече, структурата, която ще ползваме, бива ползвана и на практика (макар и в малко по-различен вариант).

Нека първо помислим как бихме решили задачата ако нямахме промяна на символи. Ползвайки алгоритъма на Рабин-Карп, задачата можеше да бъде решена с константни отговори – тоест за O(N + M + Q). Ще преизчислим префиксен масив с всички хешове от началото до всеки индекс, след което ползвайки идеята на хеширане в някаква база по модул можем бързо да намираме хеша на произволен подстринг.

В тази задача обаче имаме и промени. Как можем да подходим тук?

Както и повечето задачи, в които се иска бърз ъпдейт на единичен елемент и бързо куери на интервал, и тук ще прибегнем до ползването на индексни дървета, или в случая малко по-мощното нещо Сегментни Дървета. Вместо сума или xor или произведение или каквото и друго сме свикнали да пазим в дървото, този път ще пазим хешове.

Оригиналното решение на проблема (което се ползва в реалния живот), разчита на структура данни, наречена Merkle Tree. Тя, реално, не е нищо друго от следното сегментно дърво: файлът бива разделен на малки блокове данни (обикновено 64K), които се хешират (примерно с SHA-1 или MD5), като хешовете се поставят в листата на дървото. Бащата на две листа съдържа хеша на конкатенацията на двата хеша, разглеждани като стрингове. Така ако имаме два последователни блока с хешове, съответно, "`0beec7b5ea3f0fdbc95d0dd47f3c5bc275da8a33`" и "`62cdb7020ff920e5aa642c3d4066950dd1f01f4d`", то баща им ще държи "`3eb4e0e693987b4bcb871a44e943b7223b5dd2e5`" - хеша на стринга "`0beec7b5ea3f0fdbc95d0dd47f3c5bc275da8a3362cdb7020ff920e5aa642c3d4066950dd1f01f4d`".

Това няма да работи съвсем за задачата, която ни интересува (да не говорим, че включва ползване на хеширащи алгоритми, които не са вградени в C++), но състезателите трябваше да комбинират знанията си за хеширане с тези за сегментни дървета и да достигнат до работещо решение.

В листата на сегментното дърво ще пазим хеша на един символ; в бащите им ще пазим хеша на стринга, съставен от двата символа, в бащата на баща им ще пазим хеша на стринга, съставен от двата символа, конкатенирани с още два и т.н. Ъпдейтът на дървото (промяна на символ) става със стандартната за сегментно дърво сложност от O(logN).

Когато търсим хеша на произволен под-стринг, трябва да "конструираме" хеша, като ползваме идеята на Рабин-Карп почти директно. Тъй като Рабин-Карп изисква да умножаваме по базата на степен дължината на суфикса, качвайки се нагоре в дървото (ако ползваме идеята от тази статия) ще трябва да

пазим и на каква височина сме (за да знаем колко символа съдържа дясното под-дърво на левия индекс). Допълнително, ще ни трябва и броят символи, покрити от левия индекс и броят символи, покрити от десния индекс (за да можем накрая, отново ползвайки идеята на Рабин-Карп, да обединим хешовете, които сме генерирали от левия индекс и от десния индекс). Все пак, тъй като единствено поддържаме три допълнителни брояча и извършваме по едно умножение, събиране, и взимане на модул, сложността не се променя и остава O(logN) за query.

Така цялото решение представлява:

1. Преизчисляваме степените на базата (базите, ако ползваме повече от един хеш) в масив за O(N)
2. Строим сегментно дърво за първия стринг за O(N) (но можем да си позволим и "глупаво" строене за O(N * logN))
3. Строим сегментно дърво за втория стринг за O(M) (но можем да си позволим и "глупаво" строене за O(M * logM))
4. За всяко куери:
    a. Ако е ъпдейт, обновяваме съответното дърво за O(logN) или O(logM)
    b. Ако е куери, правим търсене двете дървета за O(logN + logM).

Цялата сложност на решението е O(N + M + Q * (logN + logM)).

В авторското решение се ползва тройно хеширане с модули около 1,000,000,000, като така шансът за колизия става по-малък от $1/10^{13}$ (хеш спейсът ни е с размер 10^{27}). Реално, тъй като тестовете няма как да бъдат настроени за модула и базата, които състезателите са избрали, двоен, а дори единичен хеш биха имали голям шанс да хванат 100 точки на задачата, като, съответно, работят забележимо по-бързо.

Реално дори "наивното" решение, в което просто симулираме куеритата символ по символ работи доста бързо. Макар и да има специфични тестове срещу него (тест генераторът е близо 500 реда), само тестове с почти максимални ограничения могат да доведат до това то да е бавно (особено ако са вкарани някакви дори базови оптимизации). За него бяха предвидени малко повече от 50 точки.

Автор: Александър Георгиев