**АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА**

**ПРЕСТРУКТУРИРАНЕ**

Условието задава следната постановка. Имаме дърво, към което се прилагат два вида заявки. Първата е нестандартна – сменя се бащата на даден връх (с това и цялото поддърво на върха все едно „се мести“) на друг. Втората е стандартна – най-близък общ предшественик.

Първата подзадача е за 10 точки. Очакваното решение е наивно – бащата на всеки връх се държи в масив и смяната става константно. За това обаче *LCA* e с линейна сложност по дълбочината. В авторовата имплементация се държат дълбочините на върховете (без промяна) и се обхожда целият път на по-плиткия връх (макар променящи се дълбочините все са някакъв евристичен ориентир), след което се обхожда пътя на другия връх до срещане на общ връх. Така сложността е .

Втората подзадача е за 25 точки. Това е улеснена версия на задачата. Условието за ъпдейт операциите казва, че всеки връх по пътя нагоре ще има максимум една промяна и така можем да използваме стандартния алгоритъм за *LCA* с *binary lifting* в оригиналното дърво, като разгледаме случаи – дали двата върха са с непроменен път до корена, дали и двата са в поддървото на преместен връх или са в различни поддървета на преместени върхове. Първите случаи са стандартни, а при вторият случай трябва да се намерят кои са тези преместени върхове и да се намери най-близкият общ предшественик на техните бащи в оригиналното дърво. Въпросът е как да намерим дали по пътя от върха към корена има връх със сменен баща. Можем да използваме следната идея, която ще е полезна и за по-нататъшното решаване. Обхождаме в началото върховете с един *DFS* и си записваме в един ред кога влизаме и кога излизаме от връх. Тази поредица я записваме в сегментно дърво. Понеже ъпдейтите са хубави (не се мести връх, който има прикачени някакви нови върхове към поддървото си), можем при смяна на бащата на *х* да ъпдейтнем всички върхове от сегментното, чийто интервали се съдържат в интервала на влизане и излизане за *х*. Така като имаме връх, можем да видим в сегментното за неговата позиция кой е най-тесният интервал, който го съдържа. Съответно върхът, от който произлиза интервала ще е търсеният връх със сменен баща по пътя. Сложността и за двете заявки в случая е логаритмична и общо .

Третата подзадача е за 30 точки. Тя е дадена за алтернативни решения на цялата задача с евентуално коренова декомпозиция на заявките (ако може?) и решение с *heavy-light decomposition*, който след известен брой заявки се преструктурира, като стане много разбалансиран. Сложностите тук биха могли да бъдат и .

Четвъртата подзадача е за 35 точки. Има два стандартни алгоритъма за намиране на *LCA*. Първият го разглеждахме, а вторият използва идеята, за която загатнахме (само че при обхождането при връщането във всеки връх също добавяме срещането му в редицата) с пазене на съответните дълбочини и намирането на най-близък общ предшественик става със заявка за намиране на върха с минимална дълбочина в интервала между върховете. Оказва се, че този вариант можем да го направим и динамичен за задачата. Заявката, при която на връх *x* се сменя бащата на *y* е еквивалентна на това да преместим поддървото му в тази част на редицата, където е интервалът на *y*. Хубавото в това обхождане е, че поддървото на връх е последователен сегмент от масива. Така ако си държим масива в *treap* можем да извършваме тази заявка за логаритмично време, защото просто трябва да отрежем нужния интервал и да го вмъкнем на необходимото място. Важни детайли са следните. Като вмъкваме, трябва да добавим ново срещане на *y* веднага след сегмента, за да си запазим свойството за намиране на *LCA*. Също при закачане на поддървото се сменят дълбочините на всички върхове но с константа (нагоре или надолу), така че трябва да пазим в *treap* лейзи заявки за това. Освен това при това местене на сегмента, изгубваме индексите, които пазим за интервала на даден връх в подредбата. Този проблем можем да отстраним така – пазим за всеки номер на връх, указател към върха в *treap*, където се намира. Сега ако пазим и за всеки връх бащата можем по указател към върха да се качим нагоре и да видим индекса на върха в сегашната наредба с логаритмична сложност. Така описаното решение работи с логаритмична сложност, но има голяма константа. Все пак крайната сложност е .

Този подход може да се прилага и при доста голям вид заявки. Ако не ни трябваше да намираме *LCA*, можеше да ползваме редицата от втората подзадача, защото тя е по-удобна за местене – не се добавят нови елементи и освен това за по-бърза имплементация, в авторовото решение е пропуснато махането на излишен елемент след преместване на сегмента (номера на върхът, в който сме се връщали след приключване на поддървото на дадения връх), което нямаше да има нужда с другата редица.

*Автор: Илиян Йорданов*