ПРОЛЕТНИ СЪСТЕЗАНИЯ ПО ИНФОРМАТИКА

Велико Търново 28-30 април 2023

Група В 9-10 клас

Анализ на задача Разписание

Задачата накратко е: Дадено ни е дърво. Трябва да обработваме бързо заявки за махане на ребро и проверка дали два върха са от едно и също дърво в гората.

Първа подзадача. На всяко ребро съпоставяме номер и при махане на ребро просто отбелязваме номера като маркиран. При въпрос посредством **DFS** проверяваме дали има път между двата върха, минаващ само по немаркираните ребра.

Сложност – $O(Q \times N)$.

Втора подзадача. Когато става въпрос за пръчка, можем да си представим, че върховете са наредени в редица. Така задачата се свежда до това да „ъпдейтваме“ ребро като махнато и въпрос дали между два върха има махнато ребро. Лесно можем да отговаряме бързо на такива заявки със сегментно дърво.

Сложност – $O(Q \times \log(N))$.

Трета подзадача. Ще надградим идеята от втора подзадача. Възможно е да приложим директно същата, комбинирана с Heavy-light decomposition, но има и по-прост метод.

Да отбележим, че премахването на ребро е като да „откачим“ поддървото му от останалата част. Логично е да търсим начин да „ъпдейтнем“ информацията за върховете в това поддърво, затова ще намерим ойлеров тур на дървото (още познат като **DFS** order). В него всяко

поддърво представлява последователен интервал от върхове, а те удобно са наредени в редица.

Нека коренуваме дървото във връх 1. За всеки връх ще пазим колко ребра по пътя от него до корена са премахнати - rem[u].

При заявка за премахване, трябва да увеличим стойностите на съответния интервал от върхове в ойлеровия тур. Това можем да направим отново със сегментно.

При въпрос за u и v е достатъчно да проверим дали rem[u]=rem[lca(u,v)] и rem[v]=rem[lca(u,v)], където lca(u,v) е най-близкият им общ предшественик. Така си гарантираме, че нито едно ребро по двата пътя към lca(u,v) не е премахнато. За константно намиране на lca(u,v) можем да използваме sparse table (binary lifting също е опция, но няма да е достатъчно бърз за следващата подзадача).

Сложност – $O(Q \times \log(N))$.

Четвърта подзадача. Сегментното дърво работи оптимално, когато въпросите и ъпдейтите са балансирани. В случая обаче въпросите са 100 пъти повече, затова ще използваме коренова декомпозиция. За всяка група от $\sqrt{N}$ върха ще пазим допълнителна променлива toadd, която ще показва колко трябва да добавим към стойността на всяко rem в групата.

Така ъпдейта ще е по-бавен, но отговора на въпрос ще е просто rem+toadd (приемайки, че ползваме sparse table за lca(u,v)).

Сложност – $O(N \times \text{sqrt}(N) + Q)$.

Пета подзадача. Ще погледнем задачата под изцяло друг ъгъл. Ако вместо махане на ребра, добавяхме такива, е много ясно каква структура щяхме да използваме – Disjoint Set Union. В същността си, тя е просто **small**

to large. Същия метод може да бъде използван и при разделяне на някое дърво – ако винаги променяме информацията във върховете на по-малката от двете получени части, всеки връх ще бъде ъпдейтнат най-много **log(N)** пъти.

Остава само да уточним как определяме коя от двете части е по-малка. Тази задача не е така проста както при добавяне на ребра. Ще изхитрим проверката по следния начин: започваме да обхождаме двете части с **DFS**, но едновременно – един връх от едната част, един от другата. Това предполага, че **DFS**-то трябва да е итеративно (неприятно, знам). В момента, в който не останат непосетени върхове в някоя от двете части, сме обходили по-малката компонента + още толкова от по-голямата, което не е фатално за сложността.

Още една подробност са ребрата, които трием. Ако просто ги оставим и ги пропускаме, има опасност да получим квадратна сложност. Вместо това можем да ги трием постепенно при **DFS**-то, като при среща на махнато ребро го разменяме с последния съсед и pop_back()-ваме вектора.

Сложност – **O(N × log(N) + Q).**

Автор: Александър Гатев