Очевидно задачата описва графи – станциите са върховете, а линиите са ребрата, които имат тегло – времето на изминаване (вече ще използваме само понятията върхове и ребра с тегла). Тези графи са специален вид – дървета, но с още едно допълнително ребро. Така те са с ***N*** върха и ***N*** ребра. Оказва се, че те също имат хубави свойства, които ще използваме за решение на задачата. Тестовете са подбрани, така че частични точки могат да се изкарат и без особени наблюдения върху този вид графи.

Първата подзадача е за 11 точки. Тук няма заявки от втория тип (такива, които променят графа). Нека текущата заявка е за върховете *x* и *y*. Достатъчно е да открием двата пътя между тях, като същевременно можем да намираме и максималните ребра в тях. Понеже ограниченията са малки за тази подзадача можем да използваме модификация на алгоритъма *DFS*. Нека тръгваме от върха *x* и ще пазим в масив *used* дали сме минали през връх. Но за да намерим повече от един път (в случая два) трябва да нулираме след като приключим обхождането на всички пътища от връх . Допълнително за забързване можем да се връщаме назад, когато стигнем крайния връх *y*. В най-лошия случай ще трябва да обходим приблизително целия граф два пъти, за да открием маршрутите. Сложността на описаното ще е . Всеки път, когато имаме заявка ще пускаме модифицирания *DFS*, и общата сложност ще е .

Втората подзадача е за 15 точки. Изисква се известно надграждане на предната идея. Правейки модифицирания *DFS* от някакъв стартов връх *x*, можем да открием двата пътя не само до *y*, но и до всеки друг връх. Ако това не е очевидно, трябва да си представим какво се случва. *DFS* създава някакво покриващо дърво на граф. Понеже ребрата са точно ***N***, то само едно ребро не участва в него. То създава и един цикъл. Всъщност двата прости пътя, които са от *x* до някакъв връх, се създават от минаването по два начина през цикъла. Затова с така модифицирания *DFS* ще можем да минем по двата различни начина през цикъла и да намерим двата пътя до всички върхове. Понеже в тази подзадача върховете са малко, то можем преди обработване на заявките да направим преизчисляване на двата пътя от всеки до всеки връх. Така ще можем да отговаряме на заявките константно! Сложността тук се получава .

Третата подзадача отново е за 16 точки. Ограниченията са същите като за предната подзадача, но вече имаме заявки за промяна. Това, което трябва да забележим, че промяната е само „козметична“. В действителност структурата на ребрата и теглата не се променят, а само номерата на върховете. Така ако на връх е сменен номерът, то за да гледаме пътищата от него, трябва да гледаме пътищата от оригиналния номер на върха, който е бил. Използвайки това наблюдение, виждаме, че за да поддържаме ъпдейт операции е достатъчно в един масив (нека се казва *orig*) да пазим за всеки номер на връх кой е бил оригиналният номер на това място. В началото за всеки връх задаваме *orig[x]=x*. При заявка за смяна на номерата на *x* и *y* е достатъчно да направим *swap(orig[x],orig[y])*. За заявките от първия тип трябва да гледаме в преизчисления масив за върховете с номера *orig[x]* и *orig[y]*. Сложността отново е .

Четвъртата подзадача е за 42 точки и е голямата стъпка за решаването на задачата. Вече трябва да разгледаме свойствата на тези особени графи, които имат един и същ брой върхове и ребра. Както казахме те са като дървета, но с едно допълнително ребро, т.е. при тях има само един цикъл. Той разделя останалите върхове в малки дръвчета (ако премахнем всички ребра на цикъла, то ще останат известен брой дръвчета). Това означава, че между два върха има или само 1 прост път, или само 2. Ако двата върха са в дръвче, тогава между тях има единствен прост път. Но заявките от първия тип са такива, че между върхове в тях има точно 2 пътя. От сега нататък ще приемаме, че това са върхове само от цикъла (за останалите просто намираме максималното ребро на единствения път от върха до корена на дръвчето и използваме и тази стойност при заявката). Нека представим цикъла като масив от елементи, които са теглата на ребрата. Задачата се свежда до това да можем да определяме максималното число за интервал в масив с малкото уточнение, че интервалът може да е за елементите вътре в него или извън него. Можем да направим динамично (*sparse table*) *dp1[x][y]*, което да казва колко е максималният елемент в интервала с дължина надясно от *x* (започвайки с *x*). Неговият стейт е очевидно с памет, като можем да го попълваме константно: за всяко *y* ≥ 1. Можем да сметнем и *dp2[x][y]*, което ще дава същата информация, но за интервала отляво. Така като търсим максималния елемент в даден интервал (за елементите вътре в него) е достатъчно да намерим достатъчно голяма степен , така че интервалите на *dp1[beg][pow]* и *dp2[end][pow]* да се застъпят. Следователно за всяка заявка от първия тип ще ни трябва време за работа. За заявките от втория тип можем да използваме идеята от предната подзадача, така че да я извършваме константно. Ограниченията позволяват дори и да не се сетим за тази идея, разменяйки стойности на динамичното да извършваме и този вид заявки за логаритмично време. Общата сложност сега е или при по-добрата реализация .

Петата подзадача е за 16 точки и при нея има само заявки от първи тип. Тя е дадена за състезателите, които имат проблем с ъпдейт заявките. Също за по-бързи имплементации на горната идея (например използването на динамично само в едната посока и предварително преизчисляване на търсените степени за всяка дължина на интервалите.). Очакваната сложност тук е .

*Автор: Илиян Йорданов*