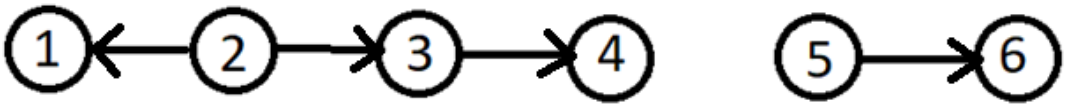
Анализ

Задачата се свежда до следното по-просто условие: Даден е неориентиран мулти граф. Искаме да отговорим на **Q** заявки от следния вид:

„ Поставяме пионка в даден връх и ориентираме всички ребра на графа по такъв начин, че местейки пионката неограничен брой пъти по ориентираните ребра, тя да посети възможно най-много на брой различни върхове. Колко е този най-голям брой? “

Тъй като броя на заявките може е до **N** една възможност е да решим задачата за всеки връх и след това да отговаряме на заявките с константна сложност.

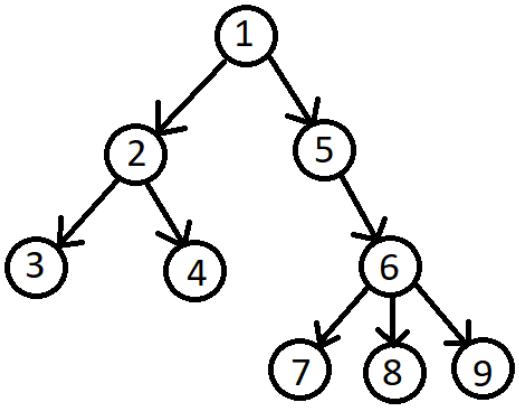
Първа подзадача. В този случай графа е съставен само от пръчки. За всеки връх ще е оптимално или да тръгнем „наляво“, или „надясно“:



Тогава отговора за връх X ако тръгнем наляво е броя на върховете вляво от X (и X самия) и аналогично в другата посока. Ако пуснем DFS от един от двата „крайни“ върха във всяка компонента и номерираме върховете по реда на посещаване съответно с num[X], то отговора за даден връх намиращ се в компонента с K върха е: max(num[X] , K-num[X]+1).

На примера горе имаме следните стойности: num[1]=1 num[2]=2 num[3]=3 num[4]=4 num[5]=1 num[6]=2. Сложност – **O(N + Q)**.

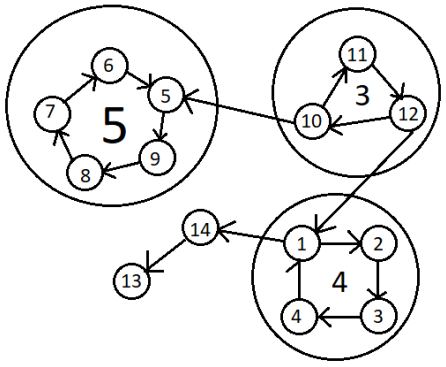
Втора подзадача. Тук графа е гора от дървета. Това не усложнява много нещата в сравнение с предишната подзадача предвид ограниченията. Отново за всеки връх ще е оптимално да тръгнем в някоя посока и няма опасност да се върнем обратно.



Искаме да намерим отговора за връх 1. Разглеждаме дървото на компонентата на връх 1 с корен 1. Можем да намерим отговора като видим колко най-много „надолу“ може да слезнем. Нека down[X] е отговора на въпроса за връх X. Тогава down[листо на дървото]=1. За всички останали върхове изчисляваме по формулата: down[X]=max(down[X_1], down[X_2], … down[$X_к$])+1, където $X_1, X_2 ... X_к$ са децата на X, а K е броят им. На примера вляво тези стойности са съответно: down[1]=4 down[2]=2 down[3]=1 down[4]=1 down[5]=3 down[6]=2 down[7]=1 down[8]=1 down[9]=1

За всеки връх X намираме тези стойности ако корена е X и съответно записваме down[X] като отговора за върха. Сложност – **O(N^2 + Q)**.

Трета подзадача. Осмисляйки малко що за граф е това имаме следната картина:



Той е съставен от „цикълчета“, някои от които са свързани един с друг. Такъв граф може да се разгледа също като частен случай на графа кактус, като тук всеки два прости цикъла нямат общи върхове. Може да забележим, че ако във всеки от тях ориентираме ребрата в една посока, то независимо къде „влизаме“ може да посетим всички върхове и да „излезнем“ от- където искаме. Така на всяко цикълче съпоставяме връх с тежест броя на върховете в него и задачата се свежда до предишната подзадача, като down[листо на дървото]=тежестта на листото и down[X]= max(down[X_1], down[X_2], down[$X_к$])+тежестта на X.

Ако искаме да намерим например отговора за 10 от картината тези стойности ще са: down[{5,6,7,8,9}]=5 down[{13}]=1 down[{14}]=2 down[{1,2,3,4}]=6 down[{10,11,12}]=9 => отг 9

Остава само проблема как да намерим цикълчетата. Фактът, че върховете, от които са съставени, са номерирани с последователни числа помага следния алгоритъм да работи:

Обхождаме числата от **1** до **N**. Ако върха с текущото число P не е посетен започваме да се движим по ребрата докато има такова от P до P+1 и не сме стигнали до връх с номер T, такъв че P е негов съсед (T≠P+1 или между P и T има повече от едно ребро). Ако не стигнем до такъв връх, значи P е като 13 и 14 от картината (не участва в цикъл). Ако стигнем, върховете P, P+1 ... T сформират нов цикъл. Сложност – **O(N^2 + Q)**.

Четвърта подзадача. Тя е същността на решението за 100т просто в малко по-примитивен вариант. В сравнение с трета подзадача тук не е гарантирано, че има само „циклични компоненти“. Но дали това променя свойството, че ако ориентираме ребрата в тях по някакъв начин, то откъдето и да влезнем ще можем да посетим всички върхове и да излезнем откъдето искаме? Ами НЕ! За да го докажем обаче, вече няма да наричаме тези компоненти „циклични“, а ще използваме по-теоретичното понятие – двойно свързани компоненти (при премахването на кое да е ребро от тях, те остават свързани) Очаква се състезателите в А група да са запознати с техните свойства, затова направо преминаваме към основното твърдение:

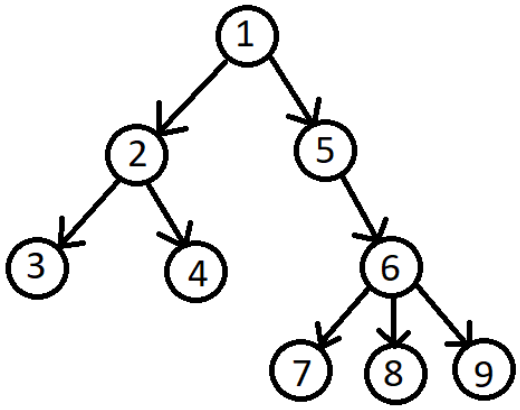
Теорема: Ребрата на всяка двойно свързана компонента могат да бъдат ориентирани по такъв начин, че тя да стане силно свързана.

Веднъж щом стане силно свързана и от всеки връх може да стигнем до всеки друг е лесно да се види, че гореописаното свойство е възможно. Но защо теоремата е вярна? Ще го покажем по-късно. Веднъж приемайки, че е вярна отново задачата се свежда до втора подзадача и съпоставяне на тежести на двойно свързаните компонентите както в трета подзадача.

Не може да намерим двойно свързаните компоненти с алгоритъма от трета подзадача затова използваме универсалния – за всяко ребро проверяваме дали разделя графа (дали е мост) и ако да значи то не участва в нито една двойно свързана компонента. Оттам, обхождайки останалите ребра, намираме компонентите. Сложност – **O(N^2 + Q)**.

Всъщност причината за последователното номериране в трета подзадача е именно, за да накара състезателите да се замислят за алгоритъма, който не работи и при двойно свързаните компоненти. Иначе просто ще решат и четвърта подзадача без много да му мислят.

Пета подзадача. Ясно е, че с квадратното динамично от втора подзадача не може да стигнем по-надалече. Затова се налага да го оптимизираме. Забелязваме, че отговора за даден връх е дължината на най-дългата пътека, която или слиза надолу – down[X], или първо се качва нагоре и по някое време слиза надолу – up[X]. Избираме произволен връх за корен на дървото и изчисляваме всички down[] стойности както във втора подзадача.



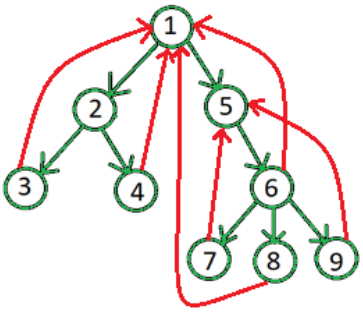
Сега остава да изчислим up[] стойностите. За всеки връх X имаме две възможности за тази стойност:
up[родителя на X]+1 – ако се изкачим повече от един връх нагоре
down[родителя на X]+1 – ако се изкачим до родителя и от него слезнем надолу. Трябва обаче да имаме предвид, че е възможно именно, тръгвайки от родителя на X към X, да се получава down[родителя на X]. Затова за всеки връх изчисляваме и sdown[X] – най-дългата пътечка, чийто втори връх е различен от този на down[X] (не непременно с различна дължина от down[X]).

Сега за up[X] взимаме по-голямата от тези стойности. Получаваме: up[X]=max(up[родителя на X], down[родителя на X])+1, когато X не участва в най-дългата пътека надолу от родителя му (down[X]+1≠down[родителя на X]) и up[X]=max(up[родителя на X], sdown[родителя на X])+1 в противен случай. В примера стойностите на sdown[] и up[] са съответно:
sdown[1]=3 sdown[2]=1 sdown[3]=0 sdown[4]=0 sdown[5]=0 sdown[6]=1 sdown[7]=0 sdown[8]=0 sdown[9]=0 up[1]=0 up[2]=5 up[3]=6 up[4]=6 up[5]=4 up[6]=5 up[7]=6 up[8]=6 up[9]=6

Окончателният отговор за X е max(up[X], down[X]). Сложност – **O(N + Q)**.

Шеста подзадача. Оптимизираме идеята на четвърта подзадача с умното динамично от пета подзадача и линейния алгоритъм за намиране на двойно свързани компоненти. Сложност – **O(N + Q)**.

Остава само да докажем теоремата от по-рано. Имайки предвид линейния алгоритъм за намиране на двойно свързани компоненти, може да направим следното наблюдение: ако ориентираме ребрата по реда им на обхождане (от текущия към непосетен връх оцветяваме реброто в зелено и към посетен връх в червено) получаваме ориентиран граф. Зелените ребра формират дърво с корен избрания връх.



От свойството на една двойно свързана компонента знаем, че от всеки връх (освен корена) може да се изкачим нагоре използвайки червените ребра „нагоре“ и зелените „надолу“. Това прави получения ориентиран граф силно свързан.

Автор: Александър Гатев