**Анализ**

*Тагове: покриващи дървета, МПД, алгоритъм на Крускал, свързани компоненти, DSU, офлайн заявки, DSU с триене*

Ясно е, че постановката задава неориентиран, непретеглен граф. Главната пътна мрежа всъщност представлява покриващо дърво на графа. Цикличната замяна по същество образува цикъл с новодобавеното ребро, от което маха едно от старите ребра.

Първата подзадача е за 11 точки. Тук улеснение идва от това, че графът е пълен. Тогава при достатъчно голямо ***N*** отговорът е почти ясен. Ако въведеното покриващо дърво е звездовидно, т.е. един връх е свързан с всички други, то тогава със сигурност едно от тези ребра ще остане, но можем да сменим всички останали, така че крайният отговор ще е ***N***-2. В останалите случаи можем да направим покриващо дърво, което не включва нито едно ребро от покриващото дърво и цената му ще е ***N***-1.

Втората подзадача е за 20 точки. Трябва да направим някои основни наблюдения. Нека видим какво точно представлява цената за преобразуване от едно покриващо дърво в друго. Ясно е, че минимален брой смени ще направим, ако общите ребра на двете дървета не ги пипаме по възможност. Нека разглеждаме едно ребро, което не е в началното покриващо дърво. Тогава то затваря цикъл в началното дърво. Ако допуснем, че всички ребра от цикъла, участват в целевото покриващо дърво, получаваме цикъл в него, т.е. има поне едно, което не участва в целевото и е от началното. Заедно с него може да направим циклична замяна и да прогресираме с една стъпка. Но така можем да приложим абсолютно същите разсъждение за текущото покриващо дърво и целевото. По този начин извършвайки циклична замяна, добавяме едно нужно ребро и махаме едно излишно. Така ако двете дървета в началото имат *k* общи ребра, то е ясно, че ще ни трябват най-малко ***N***-1-*k* циклична замени за превръщане от едното дърво в другото. Тази формула ни помага и в контекста на най-скъпо дърво. Защото това означава, че то ще е такова, което има най-малко общи ребра с даденото. Тези разсъждения ни позволяват да направим една интуитивна грийди идея. Нека обхождаме последователно ребрата, които не се включват в даденото покриващо дърво. Тогава можем да пробваме да направим циклична замяна с текущото ребро само ако на пътят между неговите два върха в дървото има някое от оригиналните ребра. По този начин като минем през всички възможни ребра, които можем да добавим, ще сме извършили възможно най-много циклични замени и ще сме достигнали до най-скъпото покриващо дърво. За бързи промени по ребрата е най-удобно списъците със съседите да са реализирани чрез *unordered\_set*. Очакваната сложност за решение е .

Третата подзадача е за 30 точки. След като направихме ключовите разсъждения е време да оптимизираме процеса по намиране на покриващо дърво, което е най-скъпо. Можем да кажем, че търсим дърво, което има най-малко общи ребра с даденото. Една хубава идея е да направим графа претеглен – слагаме цена 0 на ребрата, които не са в покриващото дърво и цена 1 на ребрата, които са в него. След което търсим минимално покриващо дърво на този граф. Ясно е, че така ще получим покриващо дърво, в което има най-малък брой общи ребра с началното. Освен това неговата сума е точно броя общи ребра (тези, които са от оригиналното). Така можем да намерим и търсеният отговор. Тук най-удачно е да се ползва алгоритъма на Крускал, за да се възползваме от цените на ребрата. Друг вариант, който е по-ценен и за следващата подзадача е да забележим, че всъщност, когато първо обхождаме ребрата с цена 0, те са безплатни и ще добавим такива, колкото максимално да свържем върховете без да ползваме ребрата на даденото покриващо дърво. По друг начин казано, ако от началния граф махнем ребрата на даденото покриващо дърво, то в началото Крускал просто ще взема ребра, така че да направи покриващи дървета на свързаните компоненти, на които евентуално се разпада графа след махането на ребрата. Съответно ребрата, които ще се вземат с цена 1 е ясно точно колко ще са – колкото са свързаните компоненти минус 1, защото в крайна сметка трябва да свържем всички свързани компоненти в една компонента. Така всъщност за решение на тази подзадача е достатъчно да махнем ребрата на покриващото дърво и да преброим компонентите. Ако от ***N***-1 извадим техния брой ще получим отговора. Сложността е .

Четвъртата подзадача е за 39 точки. Вече имаме заявки, които променят структурата на началното дърво. Нека мислим в контекста на последните разсъждения. За да намерим отговора трябва да знаем само броя свързани компоненти, на които се разпада граф след премахване на ребрата от покриващото дърво. Това е точно задачата за *DSU*. Една заявка (циклична замяна) всъщност добавя едно ново ребро и маха едно старо ребро. Лошото е, че по принцип *DSU* не поддържа триене на ребра. В случая имаме едно улеснение – можем да обработваме заявките офлайн. Тогава има известен, не много просто алгоритъм за поддържане и на заявки за премахване на ребра. Идеята на кратко е следната. За всяко ребро намираме интервалите от време, в които е активно (всяка заявка е нов момент от време). Тази информация запазваме в сегментно дърво. За да намерим отговорите на заявките, трябва да обходим дървото. При влизане във всеки връх, добавяме новите активни ребра. Съответно, когато излизаме от всеки връх ще трябва да ги махаме. Но това по същество е просто операция по връщане на старо състояние (*undo* операция). Ако пазим извършваните промени в стек, то *DSU* може да поддържа тази операция. Всъщност този алгоритъм позволява всяка структура, която поддържа *undo* операция, да поддържа и триене при офлайн заявки. Едно от нещата, които трябва да променим на *DSU*-то заради поддържа на операция по връщане е да се откажем от амортизираната сложност. За тази цел най-лесно е да махнем компресията на пътя, така гарантирано имаме *log* сложност на всяка операция. Крайната сложност ще е .

*Идея: Руско Шиков  
Автор: Илиян Йорданов*