**Анализ**

Нека с G бележим графа образуван само от червени ребра, а с G’ – графа образуван само от сини ребра. Очевидно смяната на цвят на дадено ребро е всъщност премахване на реброто от единия от двата графа и добавянето му в другия.

Първа стъпка от решението е да намерим състоянието на началния граф. Това лесно се реализира с две обхождания на графа (само по червени и само по сини ребра) за O(N).

За достатъчно малки ограничения (N≤7) можем да пробваме всички възможни промени на ребра и да проверим дали получаваме граф, който е в желаното състояние. Това решение би имало сложност O(2M), където M е броя ребра, т.е. O(2N\*(N-1)/2).

За да можем да решим задачата за по-големи графи се нуждаем от няколко наблюдения.

**Решение за желано състояние (1, 1)**

Лесно е да съобразим, че решение съществува за всяко N≥4.

В този случай, да забележим, че ако началното ни състояние е (0, 1), то можем да си представям, че всички ребра в началния граф са с противоположния цвят, и задачата би имала същото оптимално решение, но началния граф би бил в състояние (1,0). Тоест случаите на трансформация (0, 1)->(1, 1) и (1, 0)->(1, 1) са еквивалентни. Нека тогава приемем, че началния граф е в състояние (1, 0).

Трябва да свържем G’ без да разделим G. Нека първоначално G’ има **k** свързани компоненти. Ясно е, че отговорът на задачата е поне **k-1**, тъй като едно променяне на ребро не би могло да намали компонентите с повече от 1. Нека разгледаме кога можем с една промяна да намалим броя компоненти.

**За k≥3**

Да вземем три свързани компоненти C1, C2 и C3 от G’. Да изберем по 1 връх от всяка компонента - **P**∈C1, **Q**∈C2 и **S**∈C3. Ребрата P-Q, Q-S и S-P са червени, тъй като P, Q и S са в различни компоненти в G’. Нека променим P-Q на синьо. Така с 1 промяна намаляме компонентите в G’ с 1, тъй като обединяваме C1 и C2. Проблем би имало само ако тази промяна раздели върховете P и Q в G, но това е невъзможно понеже ребрата P-S и S-Q все още са червени.

**За k=2**

Нека имаме само две свързани компоненти C1 и C2, като |C1|≤|C2|.

**За |C1|≥2**

Да вземем върхове P∈C1,Q∈C1, R∈C2 и S∈C2. Тогава можем да променим P-R в синьо, тъй като ребрата P-S, S-Q, Q-R остават червени, и съответно P и R остават свързани в G.

**За |C1|=1**

Да допуснем, че съществуват върхове P∈C1,Q∈C2 и R∈C2, такива че реброто Q-R е червено. Тогава няма проблем да обединим компонентите правейки реброто P-Q синьо, тъй като ребрата P-R и R-Q остават червени и съответно G остава свързан. Проблем има само когато не съществува такава тройка върхове P,Q,R. Тъй като |C1|=1 и N≥3, то |C2|≥2, съответно единствения проблем би бил Q-R да е синьо за всяка двойка Q∈C2 и R∈C2. Това би значело, че C2 e клик (пълен граф) в G’. Това е единствения случай, в който ни трябват 2 промени за да обединим компонентите. Нека наричаме такъв граф *сложен* (k=2, |C1|=1 и C2 е клик).

Резултата, който получаваме е, че можем да решим задачата за **k-1** промени прилагайки гореописаните обединения на компоненти, освен ако не се озовем в случая със сложен граф в който ни трябва една допълнителна промяна. Лесно можем да съобразим, че ако първоначално k≥3, то няма да достигнем до сложен граф. Това се дължи на факта, че при обединяването на две компонент, клик се получава само ако и двете компоненти се състоят от по 1 връх. Тъй като при k≥4 едната компонента от двете останали е обединение на повече от 2 компоненти, то тя очевидно не е клик. При k=3 за да имаме сложен граф, то трябва и трите компоненти в началото да се състоят от 1 връх, но тогава N=3 и знаем, че решение няма.

Достигаме до твърдението, че единствения начин да ни трябват **k** операции е ако първоначално G’ е сложен. Тогава очевидно оптималното решение е да направим 2 промени.

Алгоритъмът за този случай се свежда до:

* Ако G’ е сложен, решението е с 2 промени и може да бъде лесно конструирано.
* Ако G’ не е сложен, решението е с k-1 промени, където k е броя свързани компоненти в G’. Решението конструираме като на всяка стъпка взимаме две компоненти и ги обединяваме като избираме по един връх от всяка, така че след промяната на цвета на реброто между тях, G да остане свързан. След което променяме цвета на реброто с краища тези върхове.

**Решение за желано състояние различно от (1, 1)**

За да можем да решим този случай, трябва да докажем следното твърдение:

*Твърдение(\*)*: Не съществува граф в състояние (0, 0).

*Доказателство:* Да допуснем, че такъв граф съществува. Т.е. едновременно G и G’ не са свързани. Да вземем два произволни върха **P** и **Q**.

* Ако P-Q е червено, то има път от P до Q в G.
* Ако P-Q е синьо, то в G’ трябва да има поне един връх, до който не можем да стигнем нито от А, нито от B (в противен случай G’ би бил свързан). От това следва, че има връх S, такъв че P-S не е синьо и Q-S не е синьо. От това следва, че P-S, и Q-S са червени, т.е. има път P-S-Q от P до Q в G.

Доказахме, че за два произволни върха P и Q има път от P до Q в G, т.е G e свързан. Достигаме до противоречие, и следователно оригиналното допускане е невярно.

Така имаме директно решение за желано състояние (0, 0).

Нека *разделяне* на свързан граф се състои в премахването на минимален брой ребра, така че графът да стане несвързан. Да приемем, че имаме алгоритъм за разделяне на граф.

**Решение за желано състояние (0, 1) или (1, 0)**

Двата случая са еквивалентни, затова нека допуснем, че имаме желано състояние (0, 1). Това, което трябва да забележим, е че независимо дали началното състояние е (1, 0) или (1, 1), то е достатъчно просто да *разделим* G. От (\*) следва, че G’ ще бъде свързан и ще сме в състояние (0, 1).

Алгоритъмът за този случай се свежда до:

* Ако желаното състояние е (0, 0) то извеждаме -1
* Ако желаното състояние е (1, 0) то разделяме оптимално G
* Ако желаното състояние е (0, 1) то разделяме оптимално G’

**Алгоритъм за разделяне на граф**

Единствено остана да намерим ефикасен алгоритъм за разделяне на граф. Нека фиксираме връх **P**. След разделяне на графа ще имаме поне две компоненти, което значи, че ще съществува такова **Q**, че няма път между P и Q. Ако фиксираме случаен връх P и пробваме всяко Q, то можем да сведем задачата до разделяне на два върха един от друг, премахвайки минимален брой ребра. Това е сравнително известната задача за min-cut. От теоремата за min-cut-max-flow, минималния брой ребра, които трябва да премахнем за да разделим P от Q е равен на максималния поток от P до Q, като всички ребра имат капацитет 1. Точно кои ребра да премахнем можем да намерим за линейно време след като сме генерирали максималния поток.

Така сложността на този алгоритъм е O(N\*F), където F е времето за решаване на максимален поток.

Без оптимизации алгоритъмът на Ford-Fulkerson може да не е достатъчен за пълен брой точки, тъй като авторовото решение използва алгоритъма на Dinic, който е значително по-бърз.

*Автор: Енчо Мишинев*