1 Обязательная часть

1. ***Разбиение на пути.***

   Решаем как уже разобранную задачу про неор графы. Единственное отличие - то, как мы дополняем до Эйлерова цикла. Если в связном неорграфе Эйлеров цикл тот, в котором нет четных вершин, то в орграфе Эйлеров цикл тот, в котором для любой вершины количества входящих и выходящих ребер равны. Заметим, что суммарные количества входящих и выходящих ребер графа по всем вершинам всегда равны. Тогда будем из каждой вершины, где не хватает до равенства выходящих, проводить ребра в те, которым не хватает входящих. В итоге получим Эйлеров граф. Если же изначально граф был несвязный, то из вершины одного графа, где не хватает выходящих, проведем ребро в то ребро другого, где не хватает входящих. И наоборот. Если же какой-то граф уже был Эйлеровым, то для обоих ребер используем одну и ту же вершину.

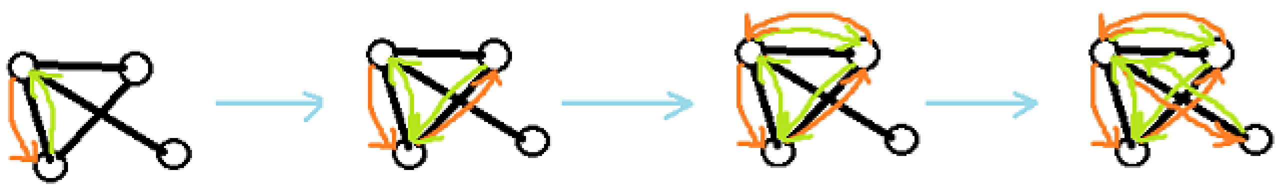
   Теперь у нас есть Эйлеров граф, повторим все действия из задачи про неор граф: найдем Эйлеров цикл в нем, удалим добавленный ранее ребра, цикл распадется на пути.

2. ***Минимизация максимального ребра на цикле.***

   Отсортируем ребра по весу. Бинпоиском будем искать минимальное такое ребро, которое было бы максимальным в цикле. Есть ли цикл проверяем дфсом, не забываем узнавать, правда ли оно максимальное в нем.

3. ***Разноцветные двери.***

   Комнаты - вершины. Двери - ребра. На каждую дверь, соединяющую комнаты A и B по два ребра: (A, B) и (B, A). С отдельными компонентами связности работаем по отдельности. Рассмотрим произвольную вершину. Возьмем одно из выходящих из нее ребер, покрасим его в оранжевый, а парное ему "обратное"в зеленый. И перейдем по этому ребру дальше, с тем же цветом, то есть первое рассматриваемое ребро следующей вершины тоже будем красить в оранжевый. Когда мы вернемся в эту вершину, то следующее исходящее из нее ребро покрасим в зеленый. И так будем чередовать. Выглядеть будет как на картинке ниже:

   Так как мы чередуем оранжевые и зеленые выходящие, то их будет либо равное (при четном количестве ребер), либо оранжевых будет на одну больше в каждой вершине.

4. ***Необходимые для достижимости ребра.***

   Мы умеем искать мосты. Ура, задача решена. Ищем все мосты, ищем какой-нибудь один любой путь, вытаскиваем все мосты на этом пути - задача решена.

5. ***Необходимые для достижимости вершины.***

   Точки сочленения мы вроде как тоже умеем искать... Занимаемся тем же самым, что и в предыдущей задаче.

6. ***Разбиение на паросочетания.***

   У нас имеется двудольный граф, в котором нет нечетных вершин (неорграф). А это значит, что у нас ураура Эйлеров граф, да еще и четной длины. Тогда возьмем наш Эйлеров цикл, возьмем какое-нибудь ребро на нем и будем, начиная с него красить по очереди ребра в два цвета. То есть первое ребро - 1, второе - 2, третье - 1, и тд. Получим две новый компоненты, причем в обеих компонентах степень каждой вершины равна $\frac{2^k}{2} = 2^{k-1}$. Тогда, чтобы получить совершенные паросочетания, нужно добиться такой ситуации, когда степень каждой вершины - 0. То есть k раз повторить эти действия (ведь каждый новый граф будет оставаться двудольным и эйлеровым). Отсюда и асимптотика $\mathcal{O}(kE)$.

7. ***Разбиение на две доли.***

   Бфсим. Для каждой компоненты связности по отдельности. Выбираем вершину. Говорим, что она цвета 1 (то есть в первой доли). Рассматриваем все смежные с ней, в которых еще не были. Говорим, что они цвета 2. Рассматриваем все смежные с ними, в которых еще не были - они опять цвета 1. И тд. То есть вершины на четной глубине - цвета 1, нечетной - цвета 2. По цветам разбиваем на две доли. Получается, что для любой вершины один из соседей обязательно будет в другой доле - та самая вершина, из которой мы пришли в нас во время бфса.

2 Дополнительная часть

1. ***Необходимые вершины в орграфе.***

   Запускаем дфс из вершины a. В каждой вершине будем хранить счетчик: сколько раз мы через нее проходили (и доходили) к вершине b. Изначально он 0. Также таскаем за собой flag, который будет показывать, дошли ли мы до вершины b. Если мы дошли до конца какого-то пути (то есть начали возвращаться в предыдущие вершины), но при этом мы не в вершине b - делаем $flag = 0$. Если в какой-то момент попали в вершину b — $flag = 1$ и заставляем алгоритм сделать шаг назад, а не идти дальше. И когда возвращаемся в вершину (после того как нашли b или конец пути) прибавляем к ней значение flag, показывая, что мы нашли путь.

   Храним переменную n - количество всех путей из a в b. Каждый раз, когда доходим до b - увеличиваем ее. В конце проходим по всем вершинам и сравниваем их счетчики с n. Если равны - значит все пути проходили через эту вершину, она необходима.