Освещение дорог

Автор задачи: Тимофей Ижицкий, разработчик задачи: Алексей Михненко

Рассмотрим следующий жадный алгоритм поиска максимального паросочетания в дереве:

- Подвесим дерево за вершину 1 и запустим от неё `dfs`.
- В функции `dfs` сначала рекурсивно вызовем `dfs` от каждого её ребёнка u_i, который найдёт максимальное паросочетание в поддереве вершины u_i. Обозначим размер этого паросочетания за m_{u_i}.
- Пусть c_v равно количеству детей вершины v, которые не были заняты паросочетанием.
- Если $c_v = 0$, то не добавляем новые рёбра в паросочетание. Таким образом, размер паросочетания в поддереве вершины v равен $m_v = \sum m_{u_i}$.
- Если $c_v > 0$, то мы можем добавить одно дополнительное ребро в паросочетание. Добавим в паросочетание рёбро между v и любым его ребёнком, не занятым паросочетанием. В этом случае $m_v = 1 + \sum m_{u_i}$.

Корректность данного алгоритма можно доказать по индукции:

- Докажем, что `dfs`(v) находит максимальное паросочетание в поддереве вершины v. Более того, если существует максимальное паросочетание, в котором вершина v не занята, то он найдёт именно подобное паросочетание.
- База: если v — лист, то корректность алгоритма очевидна.
- Переход: если v не лист, то нетрудно видеть, что $m_v \leqslant 1 + \sum m_{u_i}$. Если все вершины u_i оказались насыщены паросочетанием, то $m_v = \sum m_{u_i}$, так как если мы возьмём любое ребро из v в ребёнка, то размер максимального паросочетания в поддереве этого ребёнка уменьшится на 1, так как иначе было нарушено свойство, что среди максимальных паросочетаний алгоритм находит то, в котором верхняя вершина не занята. В этом случае максимальное паросочетание уже найдено и вершина v им не насыщена, поэтому достаточно ничего не делать.

  Иначе мы можем добавить ребро в любого ненасыщенного ребёнка и увеличить максимальное паросочетание на 1. В этом случае вершина v обязана быть насыщенной, так как без вершины v размер максимального паросочетания равен $\sum m_{u_i}$. Таким образом, данный алгоритм действительно корректен.

В задаче требовалось «включать» и «выключать» рёбра в дереве и находить размер максимального паросочетания в связной по включённым рёбрам компоненте связности. Мы будем оптимизировать описанный выше алгоритм. Сначала запустим его явно и для всех вершин v найдём m_v и c_v, определённые выше. После запросов дерево может разбиться на независимые компоненты связности. В каждой такой компоненте связности выделим самую высокую (наиболее близкую к корню) вершину. Представим, что мы запустили жадный алгоритм в каждой компоненте связности от каждой выделенной вершины, который посчитает значения m_v и c_v. Далее мы будем поддерживать такие значения эффективно.

Разберёмся как меняются m_v и c_v при каждом из запросов:

1. Если ребро (v, u) «выключается», то без потери общности пусть v — предок вершины u. Тогда в поддереве вершины u ничего не изменилось. Вычтем m_u из всех m_p, где p — предок вершины v в новой компоненте связности вершины v. Теперь рассмотрим два случая:

   - Если жадный алгоритм может построить паросочетание, в которое не входит ребро (v, u), то в новой компоненте вершины v никакие значения больше не изменились. Чтобы проверить, можно ли построить такое паросочетание, достаточно проверить, что $c_u > 0$ или $c_v \neq 1$.

- Иначе, паросочетание в компоненте вершины v уменьшилось ещё на 1. То есть надо вычесть ещё 1 из всех m_p. В этом случае так же надо обновить значения c_p. Так как $c_v = 1 \implies u$ был единственным не насыщенным ребёнком до этого, поэтому надо вычесть 1 из c_v и добавить 1 к m_v. Теперь посмотрим на предка x вершины v (если его не существует или он лежит не в той же компоненте, что и вершина v, то ничего больше делать не надо). До этого мы считали в предке, что вершина v насыщена, а теперь это не так, поэтому надо увеличить c_x на 1. Если после этого $c_x = 1$, то это значит, что до этого вершина x не была соединена с ребёнком. В этом случае надо увеличить m_x на 1 и проделать для предка вершины x такой же алгоритм, как и для вершины v, так как для него получается ровно такой же случай: ребро в его ненасыщенного ребёнка больше нельзя использовать, поэтому это аналогично удалению такого ребра.

2. Если ребро (v, u) «включается», то пусть опять v — предок вершины u. В поддереве вершины u опять ничего не изменилось. Теперь ко всем m_p надо добавить m_u. Теперь заметим, что рассмотренный выше случай для предка x вершины полностью аналогичен добавлению нового ребёнка к вершине x. Поэтому здесь нужно применить такой же алгоритм, но с пропущенным первым шагом.

3. Чтобы найти максимальное паросочетание в компоненте связности вершины v достаточно подняться до самой высокой вершины r в этой компоненте и вывести m_r.

Данный способ наивно можно реализовать за $\mathcal{O}$(глубина вершины v) на запрос. Например в 4-й подгруппе глубина дерева равна $\mathcal{O}(\log n)$, поэтому такое решение будет работать за $\mathcal{O}(n \log n)$.

Чтобы реализовать данный способ в общем случае будем хранить значения m_v и c_v в Heavy-light декомпозиции. Тогда рассмотрим, что надо сделать при «выключении» ребра (v, u). Сначала надо вычесть m_u из всех m_p на каком-то вертикальном пути, что является тривиальным массовым запросом. Теперь надо от вершины v вверх по значениям c_i найти самую длинную последовательность $1, 0, 1, 0, \ldots$ На данном вертикальном пути надо заменить значения c_i на $0, 1, 0, 1, \ldots$. Так же на этом же пути надо вычесть 1 из каждого второго значения m_i.

В случае «включения» ребра, надо по значениям c найти максимальную последовательность $0, 1, 0, 1, \ldots$ и сделать аналогичные предыдущему случаю преобразования.

Чтобы искать наибольшую последовательность вида $1, 0, 1, 0, \ldots$ в HLD можно, например, для каждой чётности сохранить минимальное и максимальное значение. Тогда это можно релизовать обычным спуском по HLD. Чтобы после этого заменить эти значения на $0, 1, 0, 1, \ldots$ достаточно ко всем чётным индексам прибавить 1, а из всех нечётных вычесть. Это так же поддерживается в обычном дереве отрезков.

Таким образом, такое решение можно релизовать за $\mathcal{O}(n \log^2 n)$, что является достаточно эффективным.

В данной задаче так же существует решение за $\mathcal{O}(n \log n)$, использующее link-cut tree. Более того, оно позволяет решать задачу ещё в более общем случае, когда исходное дерево не дано и рёбра могут удаляться и добавляться произвольным образом так, чтобы не образовывалось циклов. Данное решение остаётся читателю в качестве упражнения.