Студент: Александр Никулин
Дата: 8 ноября 2020 г.

Задача 1. *Найдите за $\mathcal{O}\left(V^2 + E\right)$ второе по весу остовное дерево в неографе.*

Решение. На практике мы уже доказали, что второе по весу остовное дерево отличается от первого только одним ребром. Можно попытаться решить совсем наивно, убирая ребра по одному из графа и строя минимальные остовные деревья, после чего выбрать минимальное. Однако, чтобы решить какое ребро выгоднее взять нам не надо пытаться достроить дерево заново (или пробовать для каждого ребра все оставшиеся ребра). Пусть у нас ребро кандидат e_{new} между вершинами (u, v). При его добавлении в дерево образуется цикл т.к. появится второй путь между этими вершинами. Чтобы избавиться от него нам нужно убрать какое-то ребро из цикла. Кажется оптимальным убирать максимальное ребро, т.к. в таком случае мы получим дерево с наименьшим весом. Таким образом нам надо найти ребро, которое даст наименьшую разницу с максимальным ребром в образовавшемся цикле. Можно предложить такой алгоритм:

1. Построим минимальное остовное дерево с помощью алгоритма Прима – $\mathcal{O}(V^2)$ или $\mathcal{O}(E + V \log V)$, если на Фибоначчиевой куче.
2. Далее, т.к. нам надо быстро узнавать максимальное ребро, заполним массив $max[u, v] = max(path(u, v))$, найдя для всех пар вершин максимальное ребро на пути между ними. Т.к. граф неориентированный и матрица будет симметрична, достаточно найти только в одну сторону. Для этого хватит поиска в глубину, который на дереве работает за $\mathcal{O}(V + V - 1) = \mathcal{O}(V)$, из каждой вершины, так что $\mathcal{O}(V^2)$.
3. Из всех ребер e, не принадлежащих минимальному остовному дерево найдем найдем такое, что $|max[u, v] - w_e(u, v)|$ будет минимальным. За $\mathcal{O}(E)$.

Т.к. таким образом мы найдем минимальное остовное дерево, которое отличается от оптимального на одно ребро, то оно обязано быть вторым минимальным. Итоговая асимптотика алогоритма $\mathcal{O}(V^2 + V^2 + E) = \mathcal{O}(V^2 + E)$.

Задача 2. *Пусть дан связный взвешенный неорграф, будем рассматривать его ребра в порядке невозрастания веса и удалять текущее ребро, если связность графа при этом не нарушается. Докажите, что этот алгоритм находит минимальный остов, или придумайте контрпример.*

Решение. Нужно доказать, что в результате получится остовное дерево, которое будет минимальным.

Первые пункты доказываются просто. В результате будет дерево, т.к. на каждой итерации цикла мы удаляем ребро, если оно не нарушает связность. Пусть у нас существует цикл, тогда все его максимальные ребра будут удалены (останется только одно минимальное), а значит в итоговом графе циклов не будет. Дерево будет остовным т.к. оно имеет только одну компоненту связности (опять же, по условию в цикле), а значит оно покрывает все вершины.

Для второго пункта, нужно прежде доказать, что никакое минимальное остовное дерево не содержит максимальное ребро в цикле. Можно сослаться на лемму о безопасном ребре (если более формально, то от обратного). Пусть у нас существует разрез по максимальному ребру. Тогда существует ребро с меньшим весом (безопасное ребро), которым можно заменить и получить меньшний общий вес. Тогда и получится, что из такого цикла максимальное ребро не попадает в минимальное остовное дерево.

Во время работы алгоритма мы убираем ребра в порядке невозрастания, а т.к. при каждом удалении остается связность, мы удаляем ребра с максимальным весом в цикле. Поэтому, если на какой-то итерации минимальное остовное дерево является подграфом нашего графа, на следующей итерации оно все еще останется подграфом. Т.к. в конце единственным остовным подграфом будет весь граф - то он и будет минимальным остовным.

Задача 3. *Доказать, что СНМ с одной эвристикой сжатия пути, к которому поступают следующие запросы «сперва только join-ы вида $p[root] = x$ (где root – именно корень, а x – любая вершина*

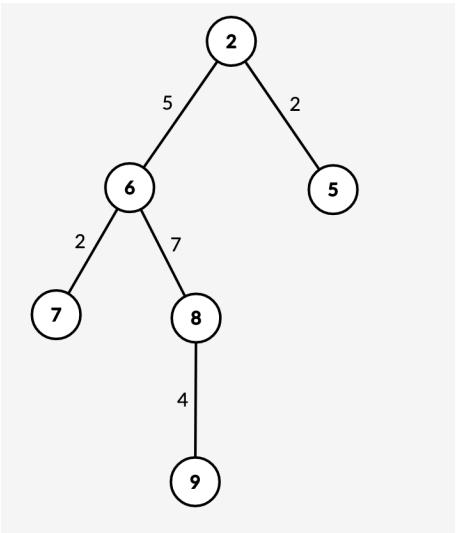
другого множества), затем только get-ы», работает за $\mathcal{O}(n+m)$ где n – число элементов, m – суммарное число запросов.

Решение. Рассмотрим два краевых случая. Пусть нам пришли только запросы вида *join* или только запросы *get*. Каждый из них выполняется за константное время т.к. это просто переприсваивание индекса или доступ по индексу, то есть $\mathcal{O}(m)$. Далее, т.к. все запросы *join* пришли в начале, после них каждая вершина будет переподвешена *get* не более одного раза (т.к. корневой элемент уже не изменится), а глубина любой вершины не может быть больше n. Пусть пришел запрос для элемента x_1, который на ходится на глубине h_1. *get* отработает за $\mathcal{O}(h_1)$, глубина вершин после x_1 уменьшится на h_1, а для всех вершин выше x_1 станет равной 1. В худшем случае (например, когда у нас бамбук), самый долгий запрос либо приходится на самую глубокую вершину, то есть за $\mathcal{O}(n)$, а все последующие за константу. Либо запрос приходит на какой-то из промежуточных, например на глубине h_k, тогда следующий наиболее долгий запрос может быть к элементу на глубине $n-h_k$, а итоговое время можно оценить как $\mathcal{O}(h_k+n-h_k)=\mathcal{O}(n)$. Итого: $\mathcal{O}(n+m)$

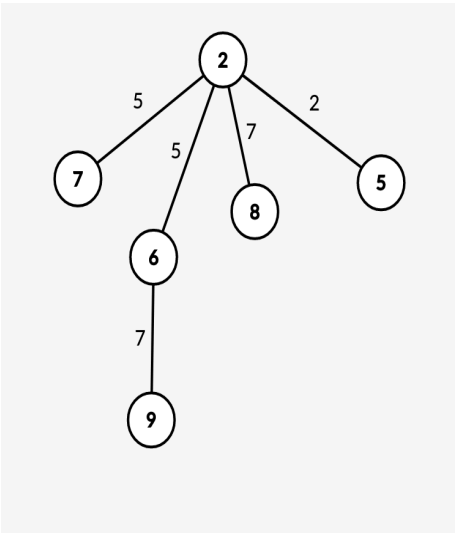
Задача 4. *Дано дерево из n вершин, с весами на рёбрах. Даны m вертикальных путей (путь $a \to b$ вертикальный, если $b-$ предок a). Посчитайте на каждом вертикальном пути максимум весов рёбер.* $n, m \leq 10^6$.

Решение. Мы можем воспользоваться тем, что все запросы даны заранее. Тогда их можно расположить так, что никакие запросы не будут перекрываться сверху вниз (когда предок в одном запросе расположен выше, чем в другом и этот запрос раньше). Если мы найдем максимальное ребро для первого запроса, после чего переподвесим вершины, это не помешает ответить на следующий запрос, где предок выше.

Алгоритм будет таким: Отсортируем запросы по b в порядке невозрастания (в предположении, что чем больше номер вершины – тем ниже она по дереву). После чего будем обрабатывать запросы последовательно, совершая для каждого сжатие путей. Например, путь у нас есть дерево как в (A) и список запросов (я их перевернул): $8 \to 9, 6 \to 9, 2 \to 8, 2 \to 7$. Тогда после выполнения всех запросов дерево станет как на (B).



(A) before



(B) after

В общем виде время работы для запросов можно оценить как $T(n) = c_1\mathcal{O}(n) + c_2\mathcal{O}(1)$. Каждый запрос сужает какой-то путь до одного ребра и все последующие запросы работают за константу. В худшем случае какой-то запрос работает максимум за $\mathcal{O}(N)$, например при запросе от листа к середине дерева (бамбука), в то время как все последующие запросы будут работать за константу (не обязательно равную 1, как показали примеры из чата). Также, $c_1+c_2 = m$, поэтому общая асимптотика $\mathcal{O}(n+m+m\log m)$. В целом, т.к. запросов ограниченное количество, мы можем отсортировать их с помощью radix sort за $\mathcal{O}(n \cdot 6)$.