Домашняя работа

Котов Артем, МОиАД2020

8 ноября 2020 г.

Содержание

Task 1: Второе по минимальности MST

Условие: Найдите за $\mathcal{O}(V^2 + E)$ второе по весу остовное дерево в неографе.

Решение.

Давайте запускать Прима, но в процессе которого будем запоминать самое тяжелое ребро между парами любых ребер, например, мы можем позволить себе при добавлении нового ребра делать следующее: пусть оно соединяет вершины v и u, где $v \in S$, а $u \in \bar{S}$, где S текущее дерево. Пересчитаем для всех $v' \in S$ следующий массивчик $d[v'][u] = max(d[v'][v], (vu))$, то есть похоже на какую-то простенькую динамику на каждом шаге добавления новой вершины u к вершине v. Все равно у нас это у нас мажорировано самим поиском MST, поэтому такая добавочка во внешний цикл `while not q.empty()` (здесь q — очередь с приоритетом на массиве для классического Прима) не попортит сложности.

Update: Рассмотрим как Прим добавляет вершины: начинаем с какой-то вершины, пусть v, добавляем в дерево эту одну вершину (т.е. сейчас $S = v$, а $\bar{S} = G \setminus S = G \setminus v$), затем рассматриваем все ребра, которые соединяют вершину v с вершинами из $\bar{S}$, выбираем минимальное по весу ребро (пусть (vu)) и добавляем эту вершинку и это ребро в наше искомое MST.

Но теперь есть дополнительное действие, которое мы делаем, у нас есть массив, который хранит самое тяжелое ребро на пути $v \to u$: $d[v][u]$. То есть в этом примере на этом шаге мы бы просто записали $d[v][u] = w_{(vu)}$, на следующих итерациях классического Прима мы присоединяем какую-то вершину $u \in \bar{S}$ к какой-то вершине $v \in S$, при этом нам надо записать максимальное ребро на пути от каждой вершины, которые уже есть в дерево, до новодобавленной, но при этом мы уже знаем самое тяжелое ребро на пути от каждой вершины дерева до вершины v, то есть до той, к которой мы добавляем (это мы получили на каком-то из предыдущих шагов итерации).

Для этого мы просто пробегаемся по всем вершинами из текущего дерева, то есть по всем $v' \in S$ и смотрим на самое тяжелое ребро на пути $v' \to u$, причем для этого (так как дерево) достаточно сравнить самое тяжелое ребро на пути $v' \to v$ с новодобавленным ребром и выбрать из них наибольшее. Ну и да, правильнее сказать, что мы храним пару (вес, ребро) в этом массиве.

Зачем нам вообще этот массив? Чтобы быстро можно было отвечать на вопрос “какое самое тяжелое ребро на пути в MST”, потому что последующая процедура поиска второго MST как раз будет разрывать искусственно сделанный цикл по этому ребру, потому что если бы мы разорвали не по этому, то мы бы получили заведомо более тяжелое дерево.

Теперь, когда у нас есть в руках MST и такой массив самых тяжелых ребер между вершинами будем делать следующее: пробежимся по всем ребрам, которые не принадлежат построенному MST, их всего $E - V + 1$ штука, для каждого ребра $(uv) \notin T$ будем добавлять такое ребро к T и удаляться самое тяжелое ребро на пути $u \leftarrow v$ (это мы делаем очень быст-

ро за счет предподсчитанного массива самых тяжелых ребер), берем среди всех таких новых деревьев минимальное по весу. По сложности у нас был улучшенный Прим за $\mathcal{O}(V^2)$, еще у нас была пробежка по всем ребрам не из MST $\mathcal{O}(E - V)$, в итоге будет $\mathcal{O}(V^2 + E)$.

■

Task 2: Краскал наоборот

Условие: 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.

Будем доказывать корректность. Во-первых, заметим, что получающееся в конце процедуры мы получаем связанный подграф, так как при удалении проверяем связность. Во-вторых, надо понять, что в конце мы действительно получим дерево, ну для этого достаточно понять, что в конце в конечном подграфе не могут остаться циклы, так как в процессе обхода мы бы удалили наибольшее ребро из цикла, следовательно, разорвали бы его. Наконец, докажем, что получающееся дерево минимально. Рассмотрим простое утверждение: в любой момент времени работы алгоритма минимальное остовное дерево (т.е. множество ребер) является подмножеством текущего множества невыброшенных ребер. Будем работать по индукции:

- База: В начале очевидно, что когда мы еще не выбросили ни одного ребра, то среди всех ребер есть множество, которое принадлежит множеству ребер MST (тут надо воспользоваться тем, что исходный граф связный, следовательно, остовное дерево существует)

- Предположение: пусть утверждение верно для некоторого множества G, обозначим соответствующее MST как T. Надо показать, что после удаления ребра (то есть следующий момент алгоритма) существует остовное дерево T', являющееся подмножеством G.

  Если мы удаляем ребро e, которое не принадлежит исходному T, то $T = T'$ и все хорошо в плане включения множеств, т.к. для T все выполнено. Если же мы удаляем ребро e, которое принадлежит T, то у такого дерева нарушается связность, следовательно, оно разбивается на две компоненты T_1 и T_2, но исходное G остается связанным, так как мы проверяем связность перед удалением, т.е. в таком случае у нас должен существовать путь T_1 и T_2 этими компонентами, который не проходит по удаленному ребру. Но тогда значит, что до удаления у нас существовал здесь цикл в G, но не в T, т.к. иначе бы T не было бы деревом (я надеюсь, что мы еще не запутались в множествах), т.е. есть

некоторое отличное от удаленного ребра e ребро e', которое принадлежит G, но не принадлежит T, тогда после удаления ребра e и добавления ребра e', то мы получим T' — как бы новое MST (но мы покажем, что либо $T' = T$, либо такое невозможно).

Почему можем так сделать? Во-первых, T' — остовное дерево, так как мы связали две компоненты одним ребром (по лемме о разрезе циклов не возникло), а множество вершин, которое T' покрывает, такое же, как и у T, которое было остовным. Во-вторых, T' — минимально: рассмотрим отношение между $w(T)$ и $w(T')$. У нас возможны в целом 3 варианта:

1) $w(e') < w(e)$, но тогда $w(T') < w(T)$, но T — MST, следовательно, такого быть не множеств.
2) $w(e') > w(e)$, но тогда, так как мы проходим в порядке невозрастания веса ребер, мы бы наткнулись на e' раньше, чем на e и удалили бы его, так как связность не нарушилась бы (напомню, что e и e' принадлежат одному циклу в исходном графе), но при этом мы уже показали, что e' есть в G, следовательно, такое тоже невозможно.
3) $w(e') = w(e)$, но тогда $w(T') = w(T)$, следовательно, T' тоже MST.

Окончательно, имеем некоторое множество ребер, которое образует остовное дерево в исходном графе, и дополнительно является минимальным по весу, то есть это MST. ■

Task 3: Насколько сжатие путей быстрое?

Условие: Доказать, что СНМ с одной эвристикой сжатия пути, к которому поступают следующие запросы «сперва только join-ы вида $p[\texttt{root}] = \texttt{x}$ (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.

Пусть у нам пришло m_1 запросов на `join`, тогда за $\mathcal{O}(m_1)$ мы выполним формальное подвешивание поддеревьев к каким-то вершинами из запроса (банально, это `Node` с указателями) в итоге получим какой-то лес, в общем случае, но давайте не умаляя общности считать, что получили одно дерево, так как запросы `get` работают в рамках одного дерева.

Представим, что пришел запрос на элемент x, который находится на глубине h_x. Тогда для него мы ответим за $\mathcal{O}(h_x)$, после чего для всех вершин-предков (т.е. предков, предков предков и т. д. до корня) мы сможем отвечать моментально (т.к. их глубина стала равна 1), а глубина всех вершин, что глубже x, уменьшилась на h_x. Рассмотрим самый грустный вариант, когда у нас есть бамбук: самым тяжелым ответом будет `get` последнего элемента ($\mathcal{O}(n)$), зато все остальные будут за $\mathcal{O}(1)$ после этого. Если какой-то промежуточный, то мы бы ответили

за $\mathcal{O}(h_x)$ для него, но тогда самым тяжелым после этого стал бы запрос `get` на последний элемент, у которого глубина уже $\mathcal{O}(n - h_x)$, в итоге мы бы потратили $\mathcal{O}(h_x + n - h_x) = \mathcal{O}(n)$, на такую обработку, и можем считать, что на сам формальный ответ на запрос тратим $\mathcal{O}(1)$. Стоит еще заметить, что глубина любой вершины не может быть больше n, а после обработки вершины с глубиной h_x, не глубина любой вершины не больше $n - h_x$ (банально, не хватит вершин, чтобы так глубоко можно было бы уйти).

Замечание. *То есть в каком-то смысле если бы запросы приходили в порядке углубления элементов, а последний запрос был бы как раз на самый глубокий элемент, то мы бы потратили как раз $\mathcal{O}(n + m_2)$ времени на это. То есть так или иначе, если нам надо отвечать на самый глубокий элемент, то мы неизбежно тратим $\mathcal{O}(n)$, важно, что если мы не сразу к нему обращаемся, то легче (в плане суммарно легче) нам не становится, потому что мы так или иначе до него все равно доберемся, просто потратим кусочки работы не сразу на нем, а на другим элементах повыше (я честно пытался это написать понятно).*

Об этом еще можно думать в терминах количества переподвешенных вершин, т.к. мы каждую вершину мы можем переподвесить не более одного раза, допустим, при обработке вершины мы переподвесили k вершин (что на самом деле значит, что ее глубина k), осталось $n - k$ непереподвешенных вершин, но всего-то вершин n, следовательно на эти операции мы в худшем случае потратим $\mathcal{O}(n)$, а на формальный ответ можем считать, что тратим $\mathcal{O}(1)$, т.к. сжатие мы посчитали отдельно. В итоге получатся сложность $\mathcal{O}(n + m)$ ■

Task 4: Сжатие путей и максимум на пути в дереве

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

Решение.

Отсортируем все запросы по b в порядке уменьшение глубины b (за линию узнаем все глубины, т.е. $\mathcal{O}(n)$). Это будет стоить нам $\mathcal{O}(m \log m)$, для запросов с одинаковым b оставляем их в обычном порядке, в котором пришли, это не важно.

Теперь обрабатывает запросы в отсортированном порядке следующим образом: идем от вершины a к вершине b производя сжатие путей до b, причем, в процессе сжатия в качестве веса ребра указываем максимальный на пути от вершины до b, то есть каждая вершины v на пути $a \to b$ будет подвешены к b, вес ребра — наибольшее ребро на пути $v \to b$. В худшем, сложность будет $\mathcal{O}(n + m \log m)$.

Почему это работает? Рассмотрим путь $a \to b \to c \to d$. Ясно, что самое тяжелое ребро на пути $a \to d$ — это самое тяжелое из ребер на путях $a \to b$, $b \to c$ и $c \to d$, если сжать такие

пути, то это просто вопрос о том, какой из ребер $a \to b$, $b \to c$ и $c \to d$ самое тяжелое. А так как у нас запросы только вертикальные, то мы не можем иметь ситуации, когда путь в каком-то предке уходит в других детей, т.е. ветки детей остаются нетронутыми, разве что вместе со своим предком при сжатии куда-то поднимаются, но пути в плане самого тяжелого ребра не портятся. Так же это вообще никак не отражается на остальной части дерева (имеется в виду, что после каждого запроса максимум что меняется только поддерево на b).

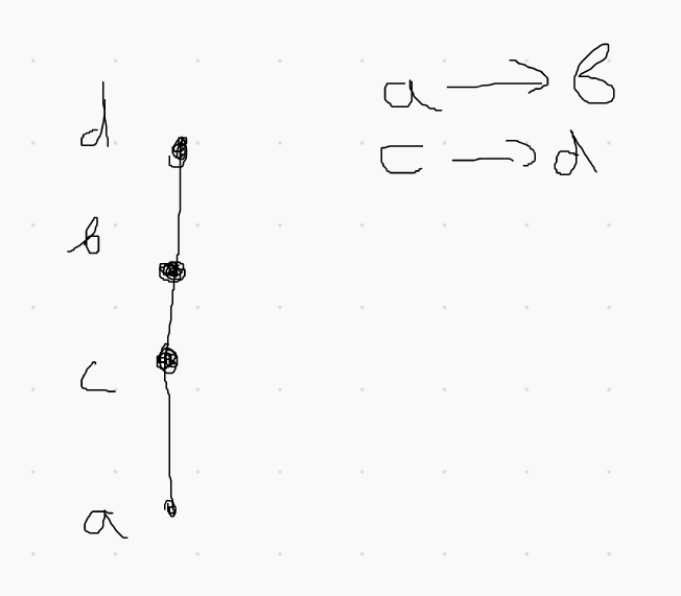

Рис. 1: Даже в такой ситуации, мы бы сначала сжали путь $a \to b$ и c бы просто подвесилась бы к b, то есть мы ничего не потеряли в плане самого тяжелого ребра.

■