Домашняя работа

Котов Артем, МОиАД2020

1 ноября 2020 г.

Содержание

Task 1: Дерево Штайнера **2**

Task 2: Расстояния в меняющемся графе **5**

Task 1: Дерево Штайнера

Условие: Дан взвешенный неориентированный граф $G = \langle V, E \rangle$. В графе есть подмножество вершин T, которые мы назовем терминалами. Минимальное дерево Штайнера — это связный подграф графа G минимального веса, содержащий все терминалы. Требуется найти такое дерево.

1) Пусть $|T| = 3$, решить за $\mathcal{O}(E \log V)$.
2) Пусть $|T| = 4$, решить за $\mathcal{O}(V^3)$.

Решение.

1) У нас из чатика добавилось условие, что веса неотрицательные, это важно нам, чтобы можно было запускать алгоритм Дейкстры. Запустим для каждой терминальной вершины алгоритм Дейкстры, на выходе мы будем знать расстояния от каждой терминальной вершины до любой другой вершины графа. Это стоило нам $\mathcal{O}(E \log V)$. Теперь, будем пробегаться по всем вершинам исходного графа (пусть текущая вершина обозначена за u) и смотреть на ST $= d[T_1][u] + d[T_2][u] + d[T_3][u]$, где $d[T_i][u]$ — расстояние от i-ого терминала до вершины u. Возьмем минимум из всех ST. Сам ST по определению будет равен весу дерева (плюс, возможно, несколько раз учтенные ребра), содержащего все терминалы, так как содержит все ребра дерева. Так что ясно, что если мы учли какие-то ребра несколько раз, то можем подвинуть вершину u на эти ребра, то есть найдем вершину-перекресток, на которой минимизируется ST (это делается за линию, так что сложность не ухудшили). Это будет либо какая-то нетерминальная вершина, либо она может совпасть с одним их терминалов, но ничего страшного в этом нет, это не мешает взять нам минимум ST.

   Замечание. *Под вершиной-перекрестком понимается такая вершина, сумма путей от которой до терминалов минимальна.*

   Естественно в процессе изначального запуска алгоритма Дейкстры нам надо поддерживать пути (например, через массив предков), чтобы в конце, когда мы нашли вершину-перекресток, смогли восстановить непосредственно сами пути от нее до терминальных вершин, а значит и искомое дерево дерево.

Update:

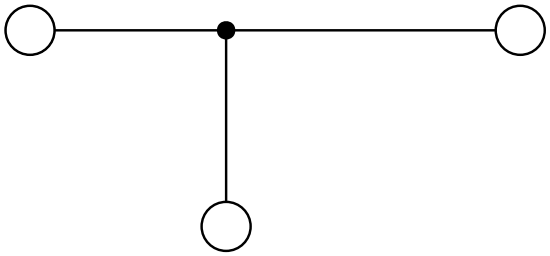

Рис. 1: Один из топологически различных вариантов соединения терминалов. Большие окружности — терминалы, черный кружочек — отличная от терминала вершина исходного графа. Черные соединяющие линии соответствует кратчайшим путям между вершинами. Второй вариант соединения соответствует случаю, когда черный кружочек совпадает с терминалом.

Замечание. *Почему рассматриваем только такие картинки? Рассмотрим в каком-то смысле "итеративную" процедуру подключения вершин в дерево: для одной вершины ничего делать не надо, для пары вершин — построить путь между ними, для трех вершин мы можем либо построить путь от одного из терминалов, либо построит путь к одной из вершин, которая уже участвует в пути между первыми двумя терминалами, либо взять вообще постороннюю новую вершины и перестроить пути до нее от терминалов.*

Ясно, что при условии положительности весов, деревья с листовыми вершинами, отличными от терминалов, заведомо не дают искомое минимальное, так как такую ветку можно просто обрубить и заведомо не увеличить (может быть так, что туда вели ребра нулевого веса), то есть искать ответ можно среди соединений, показанных на Рис. 1

Чтобы убедиться, что мы нашли минимальное дерево, рассмотрим следующие:

1. вес дерева — это сумма весов всех ребер его составляющих.
2. длина простого пути — это сумма весов ребер, по которым этот путь проходит.

Рассмотрим, для примера, произвольный набор из 4-ёх вершин, соединенный как на Рис. 1. Предположим, что такое дерево, получающееся на минимальных путях между этими вершинами, на самом деле не минимально. Тогда, существует какой-то другой, отличный от построенных путей, путь между какими-то вершинами в такой же топологии (это важно, так как, например, путь между терминалами может быть короче, но суммарное дерево может оказаться тяжелее, чем через перекресток), у которого длина меньше, чем у построенного, но тогда возможно лишь два варианта: либо это невозможно в силу того, что мы строили кратчайший путь, следовательно путей с меньшей длиной между вершинами нет, либо есть просто эквивалентный по длине другой путь,

но это картины не меняет. Раз так, то мы получили, что сумма ребер на каждом пути действительно минимально, значит и сумма всех ребер всех путей также минимальна для конкретных 4-ёх вершин.

Теперь к вопросу о том, почему мы выбираем минимальное покрывающее дерево для данных терминалов: тут все просто, мы фактически перебираем всевозможные деревья, как на Рис. 1 и вырожденные случаи тоже, когда перекресток совпадает с одним из терминалов. То есть в каком-то смысле рассматриваем все множество деревьев для этих терминалов (важно, что мы все еще работает в рамках картинки) и берем из него самое минимальное. Надеюсь, я правильно понял вопрос.

---

2) Тут какая-то жесть: насчитаем с помощью алгоритма Флойда-Уоршелла насчитаем матрицу расстояний для исходного графа, это будет стоить нам как раз $\mathcal{O}(V^3)$. Затем попробуем провернуть такой же трюк, как и в предыдущем пункте, но ясно, что может быть не более двух вершин-перекрестков (так как терминальных вершин всего 4), который в оптимальном дереве, естественно, могут как полностью совпадать с терминалами, так и лишь только какая-то одна из них. Так как у нас в руках уже есть матрица расстояний, то мы быстро отвечаем на вопрос какое же расстояние между двумя вершинами в графе.

   Теперь, так же как и в предыдущем пункте переберем эти две вершины-перекрестка и найдем ту пару, на которой минимизируется суммарный вес ребер (то есть это будет какая-то комбинация минимальных путей между терминалами и перекрестками). По путям восстановим искомое дерево. Это будет стоить нам $\mathcal{O}(V^2)$, что грустно, но Флойд и Уоршелл и так все съели.

   Замечание. *Вообще, это по сути перебор, но так как алгоритм Флойда-Уоршелла уже задал сложность, то ничего страшного.*

Update: Почему не более двух перекрестков? Из замечания в предыдущем пункте про “итеративную” процедуру: можно продолжить присоединение теперь 4-ой вершины к дереву, опять же куча разных вариантов, из которых интересны, пожалуй, лишь два (в остальных есть совпадение перекрестка и терминала):

1. когда новый терминал присоединяется к тому же перекрестку, что и предыдущие три терминала (хотя этот случай тоже не очень интересный)

2. когда мы заводим еще один перекресток (на картинке проще):

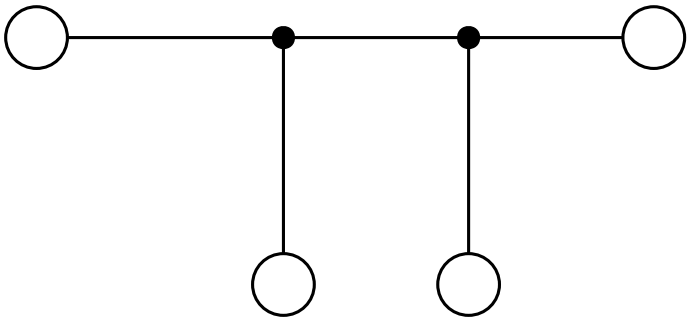

Рис. 2: Вариант дерева для четырех терминалов и двух перекрестков

Попытка добавить большее количество перекрестков приведет просто к тому, что на самом деле это будет не перекресток, а вершина какого-то пути, но и в целом, я надеюсь, что из “итеративной” процедуру присоединения новых терминалов видно, что мы как бы выпускаем лучик из терминала. Этот лучик может врезаться в терминал, тогда не очень интересно; может врезаться в перекресток, а может врезаться, например, в путь, тогда образуется новый перекресток.

Стоит отметить, что случай, когда у нас новый перекресток образуется не на пути, а где-то в вакууме через аналогию с лучиком не очень хорошо понимается, только если Вы не знаете случайно КЭД и что лучик может умереть, ни с чем не столкнувшись :)

К слову еще одно замечание, которое не прозрачно из предыдущего: в случае вырождения (то есть когда перекрестки либо совпадают друг с другом, либо с терминалом(ами)) мы перебираем еще всевозможные соединения (опять же в смысле соединения путями) этих вершин, то есть в каком-то смысле перебираем такие деревья.

■

Task 2: Расстояния в меняющемся графе

Условие: Нужно научиться на запрос «уменьшился вес ребра» за $\mathcal{O}(V^2)$ пересчитывать матрицу расстояний. Считайте, что в графе не было и не появилось отрицательных циклов.

Решение.

Рассмотрим пути, который проходили через ребро $e = (cd)$ до изменения веса ребра (считаем, что матрица расстояний у нас уже была насчитана). Для таких путей верно, что $d[a][b] = d[a][c] + w_e + d[d][b]$.

Теперь пусть изменился вес ребра e (даже если оно нам в запросе не дается, то за $\mathcal{O}(E) \subseteq \mathcal{O}(V^2)$, мы его найдем). Рассмотрим всевозможные (переберем) вершины a и b и “заставим” пройти путь через ребро e, то есть сравним изначальное $d[a][b]$ и $d[a][c] + w_e + d[d][b]$ и $d[a][d] + w_e + d[c][b]$, то есть обновляем значение расстояния в матрице в следующем виде: $d[a][b] =$

$\min(d[a][b], d[a][c] + w_e + d[d][b], d[a][d] + w_e + d[c][b])$. Это будет стоить нам $\mathcal{O}(V^2)$ как перебор всех начальных и конечных вершин.

Почему это работает? Если раньше оптимальный путь между двумя вершинами не проходил по измененному ребру, то после изменения, потенциально, новый оптимальный путь может проходить через него, поэтому проверим, стал ли потенциальный новый путь меньше, чем исходно оптимальный. Если же новый путь не оптимальнее старого, то ничего менять для таких вершин не надо, так как это изменение не затронуло этот путь (еще стоит отдельно отметить, что изменение этого ребра не затронуло пути от a до c и т.п.). По-хорошему, может случится ситуация, когда вообще все пути начали проходить через это ребро, поэтому надо проверить вообще все вершины.

■