Лекция «Построение кратчайших путей в графе»

Поздняков Сергей Николаевич

запись конспекта: Кацер Евгений

дата лекции: 21 декабря 2018 г.

§2 Построение кратчайших путей в графе

00:23

Эта задача является классической. Ее должны знать все, кто так или иначе связан с программированием или компьютерной математикой. Она интересна тем, что есть очень похожие задачи, имеющие разную сложность. Например, задача коммивояжера, заключающаяся в том, что нужно объехать N городов так, чтобы суммарный, путь или время были минимальные. Оказывается, эта задача относится к классу NP. Её можно решить перебирая все варианты (перебор всегда будет давать экспоненциальную трудоемкость по времени), в то время как проверить решение легко, правда, в определении класса NP проверка решения задачи должна давать ответ «да» или «нет», поэтому, когда проверяют решение, ставят вопрос по-другому: не говорят, что «является ли этот путь минимальным», а говорят, что «верно ли, что построенный путь меньше такого-то числа». Поэтому здесь есть переход от классической постановки к немножко другой, эквивалентной исходной по трудоемкости.

Другие задачи имеют эффективные алгоритмы. Мы сегодня рассмотрим три алгоритма, имеющие полиномиальную сложность: Форда-Беллмана, Дейкстры и Флойда. Они решают три задачи:

1. Найти кратчайший путь между двумя заданными вершинами.

2. Найти длины кратчайших путей от заданной вершины до всех остальных.

3. Найти кратчайшие пути между любыми парами вершин.

То есть в одном случае две вершины фиксированы, во втором одна, а в третьем обе вершины не фиксированы. В каком порядке нужно решать эти задачи? Можно предположить, что сначала нужно решать первую, затем вторую и потом третью задачи, но это не лучший вариант.

Как искать пути между двумя фиксированными вершинами? Если это просто путь, то это легко, но если нужно найти кратчайший путь, то как узнать, что выбранный путь кратчайший?

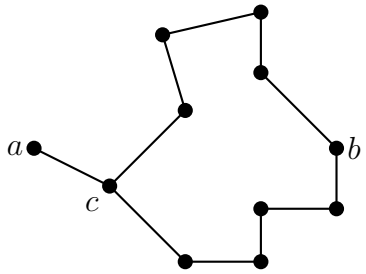


Рис. 1: Пример двух путей из a в b

Нам нужно сравнить его с длинами других путей, которые соединяют эти же две точки. Получается, решая первую задачу, мы все равно обращаемся ко второй или третьей, поэтому лучше начинать не с первой задачи, а со второй или третьей (трудоемкость обсуждаемых алгоритмов для них одинакова), однако решение третьей задачи требует большей памяти, так как нужно найти кратчайшие расстояния между всеми парами вершин, поэтому самый лучший вариант решения первой задачи — начинать со второй.

Для решения третьей задачи аналогично может возникнуть мысль использовать алгоритм для решения второй задачи, поставив его в цикл по всем вершинам. Этот путь неправильный, так как те алгоритмы, которые мы сегодня рассмотрим, имеют сложность N^3 или, как алгоритм Дейкстры, когда ребра неотрицательные — N^2, а лишний цикл увеличит показатель степени на единицу. 06:04

Во всех алгоритмах будет следующая идея — идея последовательного улучшения пометки вершины, где пометка вершины на последнем шаге — это кратчайшее расстояние. Интересный вопрос «какой смысл имеют пометки вершин на промежуточном шаге внешнего цикла». На этот вопрос студенты отвечают туманно: «неточное кратчайшее расстояние», «приближённое» или «почти кратчайшее». Это плохо, потому что, если мы не понимаем какой смысл имеют пометки на промежуточных шагах, то корректность алгоритма доказать аккуратно не сможем. Поэтому сегодня мы обязательно должны будем выяснить, какой смысл имеют эти пометки не на конечных, а на промежуточных шагах.

Давайте рассмотрим задачу №2: допустим у нас есть вершина, которая называется источник (S). Мы будем искать кратчайшие пути до всех остальных вершин. Эти пути мы будем обозначать красным цветом:

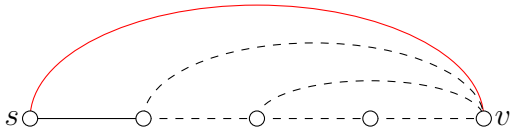


Рис. 2: Задача №2

Рассмотрим идею на примере пошагового увеличения числа ребер в пути. На первом шаге найти длины кратчайших путей, состоящих из одного ребра, легко — это будут длины ребер. Рассмотрим, как будет меняться ситуация, если, кроме путей из одного ребра, рассматривать также пути из двух ребер, потом к ним добавим пути из трех рёбер и т.д.

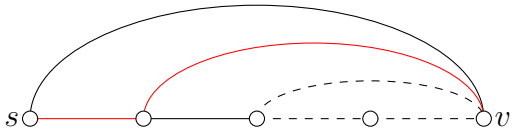


Рис. 3: После добавления двух новых ребер

Например, на третьем шаге путь уже может состоять не из одного, а из двух или трех ребер. Может быть, он будет короче, чем найденный ранее? Нужно их сравнить. Если первый оказался короче (рисунок 2), то он и остается, а если второй (рисунок 3), то будем считать его самым коротким. Затем добавим еще ребер:

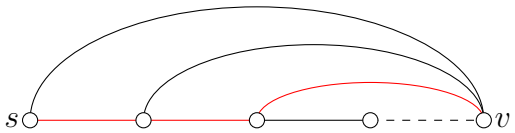


Рис. 4: После добавления еще двух новых ребер

И так далее на каждом шаге будем добавлять в пути по одному новому ребру и сравнивать пути:

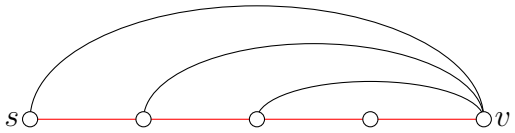


Рис. 5: После добавления еще двух новых ребер

Когда мы добавляем новое ребро, мы уже имеем предыдущее кратчайшее расстояние $\mathrm{D}(v)$ (по путям из меньшего количества рёбер). Рассмотрим все вершины, из которых можно попасть в вершину v:

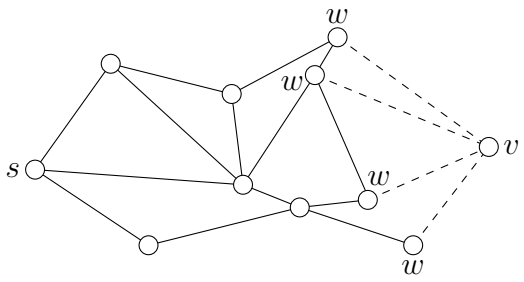


Рис. 6: Модификация пометки в вершине v

Такие вершины обозначим w (рис. 6). Путь до w будет иметь на одно ребро меньше, чем до v, а для путей не более чем с таким количеством рёбер кратчайшее расстояние до вершин уже известно. Теперь рассмотрим длины рёбер от вершин w до v, которые обозначим $\mathrm{A}(w; v)$. Мы можем сравнить суммы из кратчайших расстояний до w (по путям с количеством ребер на 1 меньше рассматриваемых) с длинами этих рёбер и взять минимальное значение из вновь рассмотренных путей и тех. что были построены на предыдущем шаге (первая пометка соответствует путям с количеством ребер на одно меньше, вторая — путям после добавления одного нового ребра):

$$\min_w(\mathrm{D}(v); \mathrm{D}(w) + \mathrm{A}(w; v))$$

$\mathrm{D}(w) + \mathrm{A}(w; v)$ означает, что мы прошли до предыдущей вершины и прибавили к её пометке длину последнего в пути ребра. Минимум по всем вершинам w, по которым можно сюда попасть. Получится, что на каждом шаге будет добавляться одно ребро. В качестве начальных пометок нужно взять длины ребер. Если из s в v есть ребро, это и будет инициализация этой пометки. На каждом шаге мы будем пометку менять следующим способом:

$$\mathrm{D}(v) := \min_w(\mathrm{D}(v); \mathrm{D}(w) + \mathrm{A}(w; v))$$

10:21

Сколько надо сделать таких шагов, если у нас на начальном, нулевом шаге инициализации уже даны расстояния по путям из одного ребра, то есть, длины рёбер? Иными словами, сколько раз нужно повторить процедуру наращивания рёбер, чтобы убедиться, что найдены кратчайшие

пути? Если в графе N вершин, то в худшем случае кратчайший путь будет содержать $N-1$ ребро (если самый короткий путь проходит через все вершины). Так как у нас уже есть расстояние из одного ребра, то нам нужно повторить эту процедуру $N-2$ раза. Это и есть *алгоритм Форда-Беллмана*. Сейчас мы его аккуратно запишем, но перед этим попробуем ответить на такой вопрос: может ли кратчайший путь состоять более чем из $N-1$ ребра? Нарисуем картинку (граф может быть как ориентированный, так и неориентированный, в отличие от остовных деревьев):

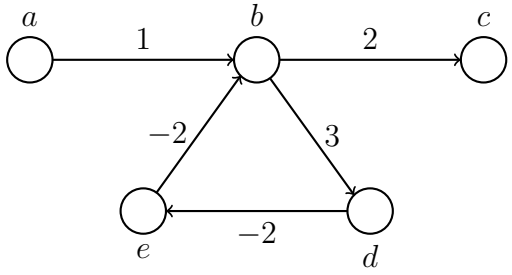


Рис. 7: Граф, у которого нет кратчайшего пути от a до c

Примечание. Если путь без стрелок, значит они есть в две стороны.

У этого графа нет кратчайшего пути, потому что, если мы идем напрямую, получаем 3, если сделаем один круг, получим два, если сделаем два круга, получим один и так далее. У алгоритма Форда-Беллмана есть естественное ограничение на применение, если нет кратчайшего пути, то алгоритм работать не будет, поэтому нужно исключить один случай — циклы отрицательной длины.

Применимость этого алгоритма — ориентированные или неориентированные графы. Они даже могут быть несвязные, тогда расстояние до некоторых вершин будет бесконечным, если в них попасть нельзя.

Пусть у нас есть граф ориентированный или неориентированный, взвешенный, с, возможно, отрицательными ребрами, но без циклов отрицательной длины. Напишем следующий алгоритм и докажем его корректность: 15:00

Algorithm 1 Алгоритм Форда-Беллмана

Инициализация:
for all $v \in V$ **do** ▷ V — множество всех вершин
    $\mathrm{D}(v) := \mathrm{A}(s; v)$ ▷ Пометка — это ребро, соединяющее s и v
end for
s — источник, из которого мы ищем пути
$\mathrm{A}(s; v)$ — длина ребра $(s; v)$

Основной цикл:
for i от 1 до $N - 2$ **do** ▷ $|V| = N$ — число вершин
    Эту процедуру мы должны повторить для каждой вершины v и для каждой промежуточной вершины w. Здесь в зависимости от того, как у вас представлен граф, можно делать либо грубый, либо красивый алгоритм.

    Грубый алгоритм — это просто два цикла по всем вершинам. Его нужно применять, когда мы работаем с матрицами смежности.

    Если у нас граф представлен списками смежности, то есть, мы для каждой вершины знаем соседей, то можно рассматривать только соседей вершины.
    for all $v \in V$ **do**
        for all $w \in V$ **do** ▷ Можно брать только соседей v
            $\mathrm{D}(v) := \min(\mathrm{D}(v); \mathrm{D}(w) + \mathrm{A}(w; v))$
        end for
    end for
end for

Примечание. Алгоритм Форда-Беллмана решает задачу №2.

Теперь мы ответим на вопрос из начала лекции: какой смысл имеет пометка, допустим, на k-ом шаге внешнего цикла (который повторяется $N - 2$ раза)? *Пометка* — это кратчайшее расстояние, но не абсолютно кратчайшее, а по путям, у которых количество ребер не более k. То есть, когда мы сделали инициализацию, то пометки — кратчайшие расстояния по путям из одного ребра. Когда мы выполним тело цикла один раз, пометки уже будут соответствовать кратчайшим расстояниям по путям из двух (или одного) рёбер. Заметим, что в алгоритме рассматриваются кратчайшие расстояния по путям, состоящим не более чем из k ребер, а не ровно из k ребер, так как может остаться путь из меньшего числа рёбер короче и увеличение числа рёбер в пути не приводит к уменьшению его длины.

Докажем корректность алгоритма: для этого докажем лемму:

Лемма. *Смысл пометки* $\mathrm{D}(v)$ *на* k*-ом шаге внешнего цикла следующий — это кратчайшее расстояние от источника до вершины* v *по всем путям, состоящим из не более чем* k *ребер.*

Примечание. Иначе эту лемму можно назвать леммой об *инварианте цикла* (инвариант цикла — это утверждение, точнее предикат — утверждение с переменными, которое остается верным как до, так и после выполнения тела цикла, то есть остаётся истинным в процессе выполнения алгоритма, несмотря на то, что зависит от k).

Доказательство. Будем доказывать по индукции.

База индукции: пусть $k = 1$. Инициализация алгоритма сделана так, чтобы обеспечить базу индукции. Мы написали явно, что расстояние по путям из одного ребра — это длина ребра. Инициализация очевидно обеспечивает корректность этого условия (фактически, инициализация — есть база индукции).

Примечание. Правильно написанная программа доказывает сама себя. Чтобы база индукции была выполнена, нужно позаботиться, чтобы изначально в алгоритме условие было выполнено.

Индукционный переход: пусть после выполнения k шагов утверждение истинно. Докажем, что оно будет выполняться и после выполнения $k + 1$ шага. Это условие обеспечивается операциями в теле цикла.

Любой путь из $k + 1$ ребра состоит из последнего ребра и пути, у которого на одно ребро меньше. Следовательно, если мы рассмотрим все входящие в эту вершину рёбра и сравним длины путей, состоящих из путей, которые рассматривались на предыдущем шаге, и еще одного ребра с пометками, которые были найдены на предыдущем шаге, мы и получим кратчайшее расстояние по путям с количеством ребер $k+1$ или меньше. Таким образом, действие $\mathrm{D}(v) := \min(\mathrm{D}(v); \mathrm{D}(w) + \mathrm{A}(w; v))$ - это в явном виде обеспечение индукционного перехода. Мы специально нарастили одно ребро (раньше было k ребер, теперь $k + 1$). Поскольку других путей нет, после выполнения этого шага будут рассмотрены все пути из $k + 1$ ребра.

□

Мы доказали лемму, теперь осталось доказать теорему. Мы уже говорили, что максимальное количество ребер в пути графа, который не содержит циклов отрицательной длины, будет $N - 1$. При инициализации путь уже состоит из одного ребра. Потом мы повторили наращивание ребер в пути $N - 2$ раза. После выполнения алгоритма по лемме будет получено, что пометка дает нам кратчайшее расстояние по всем путям с

числом ребер не более $N-1$, а это и есть кратчайшее расстояние вообще, если нет циклов отрицательной длины. **Теорема доказана.** 29:01

Следующий алгоритм, который мы рассмотрим, называется *алгоритмом Дейкстры*. Он отличается от алгоритма Форда-Беллмана тем, что все ребра имеют неотрицательную длину.

Примечание. Мы говорили на лекции о том, что от минуса иногда выгодно избавиться. В алгоритме Дейкстры мы избавились от отрицательных рёбер и сразу же получили выигрыш в эффективности. В алгоритме Форда-Беллмана для решения нам потребуется N^3 шагов, так как там тройной цикл и каждый из них примерно N шагов, в алгоритме Дейкстры нам нужно N^2 шагов.

Идея алгоритма Дейкстры состоит в следующем: первый шаг будет такой же, как в алгоритме Форда-Беллмана. Мы рассмотрим все ребра, исходящие из вершины S, и выберем из них минимальное. Первая пометка вершины, которая является минимальной, даст нам минимальный путь:

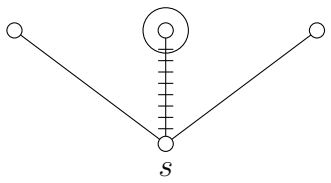
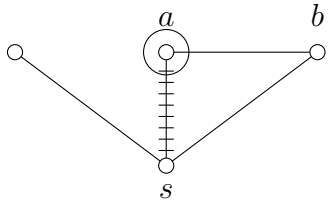


Рис. 8: Иллюстрация начала работы алгоритма Дейкстры

Пусть путь до обведенной вершины наименьший. Может ли быть более короткий путь до этой вершины? Чтобы пойти более коротким путем, мы должны пойти по другому ребру, но выбранное ребро самое маленькое, значит, если мы придём по другому пути в эту вершину, мы по крайней мере пройдём по одному ребру, которое больше либо равно наименьшему. И еще нам придется пройти по другому ребру, длина которого тоже больше либо равна нулю, то есть за счёт другого ребра уменьшить путь не удастся. Поэтому на первом шаге мы получим самый короткий путь до одной из вершин.

Дальше многие студенты думают, что надо этот путь наращивать, идти по пути, проходящем через первую найденную вершину. Это совсем не обязательно. Может оказаться, что пути до других вершин окажутся самыми короткими, но не будут проходить через эту вершину:



Действительно, не обязательно, чтобы попасть в вершину b, надо идти через вершину a, может быть, путь прямо в b будет короче. Поэтому мы создаем множество, называемое фронтом, состоящее из вершин, которые будут рассматриваться (соседние вершины с теми, до которых кратчайшее расстояние найдено) и будем вычислять пометки по аналогии с алгоритмом Форда-Беллмана: к уже найденным пометкам добавляем длины рёбер в соседние вершины и вычисляем минимальное значение:

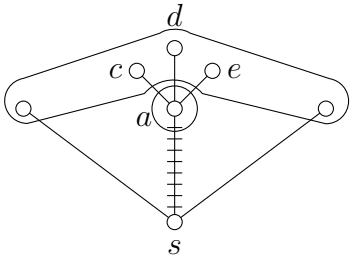


Рис. 9: Второй шаг работы алгоритма Дейкстры

Эту процедуру мы будем повторять. На каждом шаге у нас будет находиться еще одна вершина, для которой минимальное расстояние известно.

А какой смысл будет у пометок остальных (ещё не обработанных) вершин? Это будут кратчайшие пути, обладающие таким свойством: все вершины кроме последней обработаны. То есть кратчайший путь до предпоследней вершины найден, а они являются соседними с ней (например, путь до a найден, а c, d и e — ее соседи). Если по таким путям до вершин добраться нельзя, то пометкой будет «бесконечность» (достаточно большое число). Давайте рассмотрим теперь сам алгоритм (инициализация в точности как у алгоритма Форда-Беллмана): 33:59

Algorithm 2 Алгоритм Дейкстры

Инициализация:
for all $v \in V$ **do**
    $\mathrm{D}(v) := \mathrm{A}(s; v)$
end for
V' — множество обработанных вершин, то есть тех, до которых расстояние уже найдено
V/V' — множество необработанных вершин
Мы будем продолжать цикл, пока множество необработанных вершин не пусто. После завершения цикла все вершины будут обработаны, то есть до всех них кратчайшие расстояния найдены.
$V' = \{s\}$ ▷ множество обработанных вершин в начале состоит только из источника, до которого расстояние равно нулю

Основной цикл:
while $V/V' \neq 0$ **do** ▷ Пока есть необработанные вершины
    Найти вершину с минимальной пометкой $\mathrm{D}(v)$
    ▷ Для этого мы должны просмотреть все вершины

    $V' := V' \bigcup \{v\}$
    Теперь мы модифицируем пометки, добавляя те вершины, которые являются соседними к v (на рисунке 8 роль v играет a, а вершины, находящиеся в выделенной области — соседи. Обозначим их через w).
    for all $w \in V$ **do** ▷ По соседям v будет быстрее
        $\mathrm{D}(w) = \min(\mathrm{D}(w); \mathrm{D}(w) + \mathrm{A}(v; w))$
    end for
end while

Можно заметить, что у нас основной цикл имеет N шагов, а вложенный в него цикл менее N шагов, поэтому сложность будет порядка N^2, что делает его быстрее алгоритма Форда-Беллмана.

Докажем корректность алгоритма Дейкстры. Для этого сформулируем лемму:

Лемма. *На k-ом шаге внешнего цикла пометка $\mathrm{D}(v)$ имеет следующий смысл:*

1. *Для обработанных вершин это кратчайшее расстояние до источника.*

2. *Для необработанных вершин — это кратчайшее расстояние до источника по путям, у которых все вершины кроме последней обработаны.*

Доказательство. Докажем по индукции.

База индукции: обеспечивается инициализацией. Если мы берем кратчайшее из ребер, которое выходит из вершины, то более короткого пути уже не будет.

Индукционный переход: предположим, что после выполнения k шагов цикла условие выполняется и докажем, что после выполнения $k+1$ шага условие тоже будет выполняться.

Докажем, что после очередного выполнения цикла у нас добавится еще одна вершина, для которой кратчайшее расстояние найдено.

От противного: пусть $\mathrm{D}(v)$ на $k{+}1$ шаге не является кратчайшим. Как мы получили минимальное из пометок, которые были на предыдущем шаге? У нас есть какое-то множество пометок на $k{+}1$ шаге и мы выбрали минимальную из пометок (минимальным является не ребро, а путь до источника):

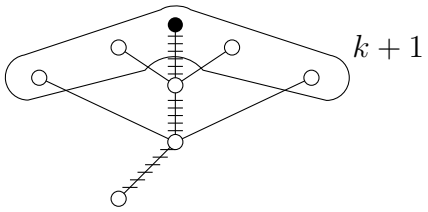


Рис. 10: Минимальный путь выделен штриховкой

Пусть есть какой-то путь до какой-нибудь вершины, который короче заштрихованного. Этот путь обязательно должен пройти через какую-то вершину, которая является последней из необработанных:

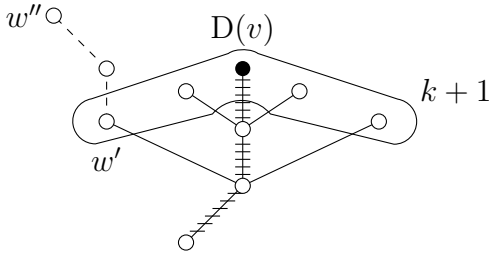


Рис. 11: Новый более короткий путь

Пометка $\mathrm{D}(w')$ не может быть меньше пометки $\mathrm{D}(v)$, в худшем случае она равна этой пометке ($\mathrm{D}(w') \geqslant \mathrm{D}(v)$). Если бы она была меньше,

мы бы её и выбрали, а $\mathrm{D}(w'')$ не может быть меньше пометки $\mathrm{D}(w')$ ($\mathrm{D}(w'') \geqslant \mathrm{D}(w')$), так как добавляются ребра неотрицательной длины, следовательно, $\mathrm{D}(w'') \geqslant \mathrm{D}(v)$ — противоречие!!!

Второе условие (для необработанных вершин) обеспечивается искусственно в алгоритме (мы просто добавляем все соседние вершины к обработанной).

□ 46:54

Можно заметить, что схема похожая: инициализацией мы обеспечиваем базу индукции, а телом основного цикла — индукционный переход.

Третий алгоритм — алгоритм Флойда. Этот алгоритм, в отличие от двух предыдущих, решает задачу №3. Он находит расстояния между всеми парами вершин. Ранее мы изучали алгоритм Уоршелла — алгоритм транзитивного замыкания. Так вот это один и тот же алгоритм, только алгоритм транзитивного замыкания применим к невзвешенным графам, а в алгоритме Флойда мы добавили еще веса. Алгоритм Флойда состоит из трёх циклов. Пометка теперь нам нужна не от одной переменной, а от двух, потому что теперь *пометка* - это кратчайшее расстояние от вершины u до вершины v (на некотором подмножестве путей), поэтому инициализация будет длинами всех ребер:

Algorithm 3 Алгоритм Флойда

```
Инициализация:
for all u ∈ V do
    for all v ∈ V do
        D(u; v) := A(u; v)          ▷ Если ребра нет, то пометка будет бесконечной
    end for
end for

Основной цикл:
for all w ∈ V do                    ▷ Идет по промежуточной вершине
    for all u ∈ V do
        for all v ∈ V do            ▷ Эти два идут по парам вершин, между которыми мы ищем кратчайшие пути
            D(u; v) := min(D(u; v); D(u; w) + D(w; v))
        end for
    end for
end for
```

Поясним картинкой:

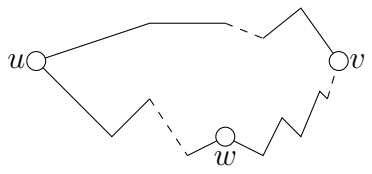


Рис. 12: Пример путей с промежуточной вершиной w и без нее

Мы можем сравнить вариант пути, когда мы вершину w не используем и идём от u до v, либо когда мы идем через w: сначала доходим до w, потом до v. Здесь очень интересно спросить вас, как вы считаете, что будет инвариантом этого цикла? То есть, если мы по внешнему циклу сделаем k шагов, какой смысл будут иметь пометки?

Давайте упростим вопрос: сделаем один шаг. До выполнения цикла у нас пометки — это длины кратчайших путей напрямую, когда промежуточных вершин нет. Когда мы выполним один раз тело внешнего цикла, то будут рассмотрены все пути через одну (первую по порядку выбора) вершину:

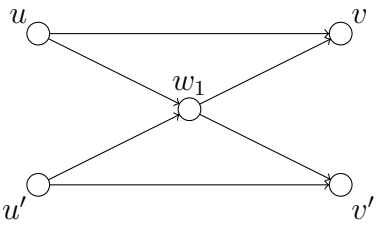


Рис. 13: Путь через первую промежуточную вершину

Пусть можно дойти от u до v по ребру. Рассмотрим первую промежуточную вершину w_1 и сравним путь через неё с длиной ребра, потом рассмотрим другую пару вершин u' до v' и сравним длину ребра между ними с путем через эту же промежуточную вершину w_1. Когда все пары вершин и ребра между ними будут рассмотрены, берем следующую промежуточную вершину и сравниваем длину пути с её использованием с длиной, рассмотренной до этого (без её использования). На каждом шаге в множество промежуточных вершин добавляется по одной новой вершине. Тем самым рассматриваются все более длинные пути.

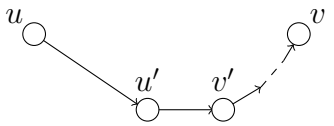


Рис. 14: Дальнейший путь

54:57

Лемма. *На k-ом шаге основного цикла пометки алгоритма Флойда имеют следующий смысл: это кратчайшие пути на множестве путей, у которых в качестве промежуточных вершин используются вершины с первыми k номерами: $w_1, w_2, \ldots, w_k$ (это упорядочивание соответствует тому, в каком порядке мы берем вершины при прохождении цикла).*

Упражнение. Справедливость леммы и корректность алгоритма доказать самостоятельно.

Примечание. До сих пор мы говорили о нахождении путей, но на самом деле мы находили не пути, а их длины. Исправить алгоритмы, чтобы они находили еще и сами пути, нетрудно. Давайте подкорректируем алгоритм Дейкстры, чтобы находить не только длины кратчайших путей, но и сами пути.

На самом деле, запоминать пути не обязательно, потому что кратчайшие пути образуют дерево кратчайших путей (так как цикл образоваться не может, иначе какой же это кратчайший путь, если мы по нему будем ходить по кругу).

Для того, чтобы сохранить дерево, достаточно помнить предыдущую вершину, то есть для каждой вершины нужно помнить, как мы в нее попали. Давайте введем еще одну пометку — $\mathrm{B}(v)$. 58:41

Algorithm 4 Алгоритм Дейкстры

Инициализация:
for all $v \in V$ **do**
    $\mathrm{D}(v) := \mathrm{A}(s; v)$
    $\mathrm{B}(v) := s$
end for
V' — множество обработанных вершин, то есть тех, до которых расстояние уже найдено
V/V' — множество необработанных вершин
$\mathrm{B}(v)$ — имя предыдущей вершины.
$V' = \{s\}$

Основной цикл:
while $V/V' \neq 0$ **do**
    Найти вершину с минимальной пометкой, пусть это v
    $V' := V' \bigcup \{v\}$
    for all $w \in V$ **do**
        if $\mathrm{D}(w) + \mathrm{A}(v; w) < \mathrm{D}(w)$ **then**
            $\mathrm{D}(w) := \mathrm{D}(w) + \mathrm{A}(v; w)$
            $\mathrm{B}(w) := v$ ▷ Если $\mathrm{D}(w)$ меньше, то оно и останется
        end if
    end for
end while