Лекция 6

ЗАДАЧА ПОСТРОЕНИЯ КРАТЧАЙШИХ ПУТЕЙ

Задан ориентированный граф $G = \{N, A, c\}$, где N – множество вершин графа, A – множество дуг, c – вещественная функция, заданная на дугах: для $a \in A$ $c(a)$ - длина дуги.

Длиной пути $W = (a_0, a_1, ..., a_k, ..., a_l)$, называется сумма длин входящих в него дуг: $c(W) = \sum_{k=0}^{l} c(a_k)$.

Обозначим через $s(W)$ начальную вершину, а через $t(W)$ - конечную вершину пути W. Если из контекста понятно, про какой путь идет речь, то будем обозначать эти вершины просто s и t.

Кратчайшим путем между вершинами s и t называется путь минимальной длины среди всех путей, соединяющих эти вершины. Задача построения кратчайших путей на сети является одной из самых известных сетевых оптимизационных задач. Эта задача широко используется на практике и как самостоятельная, и как вспомогательная при решении более сложных сетевых задач. Разработано много алгоритмов, позволяющих эффективно строить кратчайшие пути. Мы разберем некоторые из этих алгоритмов, может быть, не самые эффективные теоретически, но вполне пригодные для использования на практике.

Классификация сетей

Сначала классифицируем задачи построения кратчайших путей. В зависимости от того, какой именно тип задачи нужно решать, следует применять тот или иной алгоритм. Первый признак, по которому мы будем классифицировать задачи – тип сети. Можно выделить следующие виды сетей, влияющие на выбор алгоритма решения задачи построения кратчайших путей:

1. Сеть с единичными длинами дуг.
2. Сеть с неотрицательными длинами дуг.
3. Сеть с произвольными длинами дуг. Отсутствие отрицательного цикла гарантировано.
4. Сеть с произвольными длинами дуг. Отсутствие отрицательного цикла не гарантировано.
5. Ациклическая сеть.

Обратите внимание, что для сетей 1-4, каждый следующий тип является обобщением предыдущего. То есть алгоритм, решающий задачу на сети типа 3, сумеет решить эту задачу и на сетях типа 1 и

2. Так, может быть, нам достаточно одного алгоритма для сетей типа 4, а применять мы его будет к сетям всех типов. Так, конечно, можно поступать, однако для более узких задач могут быть предложены алгоритмы проще и эффективней.

В стороне от общего ряда находится сеть пятого типа – ациклическая. Сеть называется ациклической, если в ней нет ориентированных циклов. Этот признак не имеет отношения к длинам дуг, а определяется топологией сети. Ациклические сети появляются в некоторых специальных задачах и для них существуют особые, очень эффективные алгоритмы построения кратчайших путей.

Задача построения кратчайших путей на сети имеет смысл только в том случае, если в сети нет отрицательных циклов, то есть циклов, длина которых отрицательна. Если в сети есть такой цикл, то, двигаясь из вершины i в вершину j, мы доберемся до него и начнем ездить вдоль этого цикла, «накручивая петли», от чего длина пути будет только уменьшаться. Само понятие кратчайшего пути в этих условиях становится бессмысленным. Поэтому, строить кратчайшие пути можно только на сетях без отрицательных циклов.

Отсюда следует, что при решении задачи типа 4, то есть задачи на сети, для которой априори не гарантировано отсутствие отрицательных циклов, можно придерживаться двух разных стратегий.

Можно сначала произвести «препроцессную обработку» сети, то есть запустить специальный алгоритм, который обследует сеть и ответит на вопрос, есть ли в ней отрицательные циклы. При ответе «да» дальнейшая работа прекращается, а при ответе «нет» можно воспользоваться алгоритмами, решающими задачу на сетях с гарантированным отсутствием отрицательных циклов.

Можно поступить по другому: воспользоваться алгоритмом, который содержит специальные проверки, гарантирующие, что в процессе построения кратчайшего пути мы не натолкнемся на отрицательный цикл. К сожалению, такие проверки усложняют алгоритмы построения кратчайших расстояний и делают их менее эффективными.

Целесообразно, особенно если на сети надо решать не одну, а несколько задач построения кратчайших путей, применить первую стратегию и сначала убедиться, что отрицательных циклов нет.

Алгоритмы поиска отрицательных циклов, хотя и похожи на алгоритмы построения кратчайших путей, но все же обладают определенной спецификой, и здесь рассматриваться не будут. В ближайших лекциях мы разберем алгоритмы построения кратчайших путей на сетях первых трех типов. В этой лекции будет рассмотрен алгоритм построения кратчайших путей на сети с

единичными длинами дуг – метод «поиска в ширину» («breadth first search»). А в двух последующих – алгоритм Беллмана-Форда для сетей с произвольными длинами дуг и Алгоритм Дейкстры – для сетей с положительными длинами дуг.

Вообще-то, про задачу построения кратчайших путей можно рассказать так много интересного, что это заняло бы по меньшей мере пару месяцев. Тем не менее, ограничимся перечисленным джентльменским набором алгоритмов.

Классификация задач

Кроме того, что приходится строить кратчайшие пути на сетях различного типа, сами по себе задачи тоже могут несколько различаться. Принято выделять три вида задач:

1. От одной до одной.
2. От одной до всех.
3. От всех до всех.

В первой задаче нужно построить кратчайший путь между заданной парой вершин. В второй – построить кратчайшие пути от данной вершины до всех остальных вершин сети. А в третьей – построить кратчайшие пути между всеми парами вершин.

Конечно, обобщением всех трех задач была бы задача построения кратчайших путей от любой вершины подмножества M до любой вершины подмножества N, но почему-то в такой формулировке задачу построения кратчайших путей рассматривать не принято.

Мы будем разбирать алгоритмы решения второй задачи. Дело в том, что решение задачи 2 не намного сложнее решения задачи 1. Хотя формально задача 2 эквивалентна n задачам 1 (n - число вершин сети), но, на самом деле, задачи 1 и 2 решаются примерно за одинаковое время. До последнего времени алгоритмы построения кратчайших путей фактически не могли решить задачу 1, не решив почти полностью задачу 2. И только в последние годы положение начало меняться. Стали появляться алгоритмы, которые решают задачу 1 намного эффективней, чем задачу 2, но, к сожалению, они находятся за пределами наших рассмотрений.

Задача 3 сводится к n задачам 2. При этом до сих пор не придумано никаких принципиальных усовершенствований, которые могли бы улучшить оценку сложности или существенно уменьшить время решения задачи 3 по сравнению с n решениями задачи 2. Существуют специальные алгоритмы, которые решают именно задачу 3 вместо n решений задачи 2 (алгоритм Флойда, алгоритм Данцига), однако ни по оценке сложности для разреженных сетей, ни по своей фактической эффективности

использовать их не лучше, чем n раз решить задачу 2. Так что задачей 3 мы тоже специально заниматься не будем, а порекомендуем каждому, кто захочет получить полную матрицу кратчайших путей, n раз решить задачу 2.

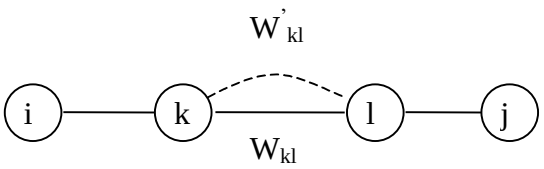
Дерево кратчайших путей

Решением задачи 1 является одно число – длина кратчайшего пути и сам путь - последовательность дуг сети.

Решением задачи 3 является матрица кратчайших расстояний и, соответственно, матрица кратчайших путей.

Решение задачи 2 при построении кратчайших путей от заданной вершины s – это массив кратчайших расстояний $R[0..n-1]$, где R_i - длина кратчайшего пути от вершины s до вершины i. Кроме того, надо каким-то образом запомнить кратчайшие пути от s до всех достижимых из s вершин сети. Казалось бы, что для этого придется хранить целую матрицу, поскольку каждый путь может содержать максимум $n-1$ дугу, а количество путей - $n-1$. Однако скоро мы увидим, что для хранения всех кратчайших путей хватит одного массива. Это связано с тем, что можно так подобрать кратчайшие пути, выходящие из одной вершины, что они будут образовывать дерево – граф без циклов. А для кодировки дерева хватит и одного массива.

Сначала покажем, что любой фрагмент кратчайшего пути также является кратчайшим путем. Пусть W_{kl} - фрагмент кратчайшего пути W_{ij}, а W'_{kl} - какой-то другой путь, соединяющий вершины k и l. Рассмотрим путь $W'_{ij} = W_{ik} + W'_{kl} + W_{lj}$. Поскольку путь W_{ij} кратчайший, то $c(W_{ij}) \le c(W'_{ij})$. Значит, $c(W_{ik}) + c(W_{kl}) + c(W_{lj}) \le c(W_{ik}) + c(W'_{kl}) + c(W_{lj})$, а, поэтому, и $c(W_{kl}) \le c(W'_{kl})$. То есть путь W_{kl} - кратчайший среди всех путей, соединяющих вершины k и l.



Теперь докажем, что существует такой набор кратчайших путей от заданной вершины s до остальных достижимых из s вершин сети, объединение которых являлось бы деревом.

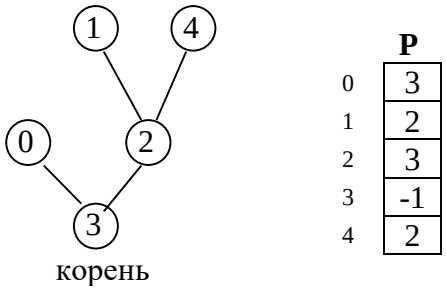
Напомним, что n - число вершин сети. Возьмем произвольное множество кратчайших путей от вершины s до всех достижимых вершин сети. Обозначим через H граф, образованный всеми кратчайшими путями, выходящими из вершины s, а через m - число дуг в этом графе. Граф H является связным. Если $m = n-1$, то H - дерево. Если $m > n-1$, то найдется хотя бы одна вершина, в которую входит больше одной дуги графа H, потому что если бы в каждую вершину, кроме s, входила ровно одна дуга (в s не входит ни одной дуги), то количество дуг было бы в точности равно $n-1$. Пусть два кратчайших пути W_{si} и W_{sj} входят в вершину t. Обозначим через (k,t) дугу пути W_{si}, входящую в вершину t, а через (l,t) - соответствующую дугу пути W_{sj}.

Удалим из графа H дугу (l,t). Докажем, что новый граф H' тоже будет содержать кратчайшие пути из s во все достижимые вершины. Если кратчайший путь в графе H не содержал дугу (l,t), то этот путь присутствует и в графе H'. Если же путь W_{sx} содержал дугу (l,t), то этот путь состоял из двух фрагментов: $W_{sx} = W_{slt} + W_{tx}$. Рассмотрим путь $W'_{sx} = W_{skt} + W_{tx}$. Поскольку оба пути W_{si} и W_{sj} кратчайшие, то их фрагменты W_{skt} и W_{slt} - кратчайшие пути, соединяющие s и t. Значит $c(W_{skt}) = c(W_{slt})$ и, следовательно, $c(W'_{st}) = c(W_{st})$. Отсюда следует, что путь W'_{sx} тоже кратчайший, и граф H' содержит кратчайший путь до вершины x.

Мы доказали, что если количество дуг в графе H, являющегося объединением кратчайших путей от s до всех достижимых вершин, больше $n-1$, то найдется граф H', содержащий все кратчайшие пути, но с числом дуг на единицу меньше. Отсюда следует, что минимальное число дуг в графе, являющегося объединением кратчайших путей во все достижимые вершины, равно $n-1$, где n - число достижимых вершин. А значит, этот граф – обозначим его T_s - является деревом, что непосредственно следует из одного из определений дерева. Будем называть граф T_s *деревом кратчайших путей* с корнем в вершине s. При построении кратчайших путей от одной вершины до всех остальных будет строиться именно это дерево.

Представление в компьютере дерева кратчайших путей

Существует много различных способов кодировки деревьев. Какой из них целесообразно использовать, зависит от конкретной ситуации. В нашем случае в дереве T есть одна выделенная вершина s, которую мы будем называть корнем. Для каждой вершины i существует единственный путь W_i, ведущий в нее из корня. Обозначим через p_i вершину, которая непосредственно предшествует i, в пути W_i. Эта вершина называется *предком* вершины i в дереве T. У корня предка нет. Для того, чтобы задать дерево, достаточно для каждой вершины указать ее предка. То есть для кодировки дерева нужно хранить один массив `P[0..n-1]`, в котором в переменной `P[i]` содержится номер предка вершины i. Такой массив `P` обычно называют *списком предков*. На рисунке изображено дерево и приведен его список предков.



Список предков во многих случаях, в том числе и при построения кратчайших путей, является удобным представлением дерева. Если в результате работы какого-либо алгоритма построения дерева кратчайших путей будет получен список предков, то этого достаточно, для того, чтобы восстановить весь путь из вершины s в любую достижимую вершину. Возьмем какую-то вершину i. Вершина $P[i]$ - ее предок – предпоследняя вершина на пути из s в i. $P[P[i]]$ - вершина, находящаяся на этом пути перед $P[i]$, и так далее. Один за другим находя предков, мы дойдем до корня, а, переписав эту последовательность вершин в обратном порядке, получим кратчайший путь, ведущий из s в i.

Таким образом, решение задачи построения дерева кратчайших путей будет представлено в двух массивах – списке предков `P[0..n-1]` и в массиве расстояний `R[0..n-1]`, в котором в переменной `R[i]`, хранится величина кратчайшего расстояния от s до i, если вершина i достижима, и неопределенная величина, если она не достижима.

В качестве предка корня s мы будем указывать что-нибудь фиктивное, например, «-1», а в качестве предка недостижимой вершины – что-нибудь «еще более фиктивное», например, «-2».

На самом деле, в задачах построения кратчайших путей целесообразно в списке предков в переменной `P[i]` задавать не номер предка, а номер дуги, которая в дереве кратчайших путей, ведет из предка в вершину i. Это ничуть не усложняет задачу нахождения вершин пути, поскольку, зная номер дуги, легко найти и ее начальную, и ее конечную вершину. С другой стороны, если известен номер дуги, а не номер вершины, то будет гораздо легче вычислять разные характеристики пути, поскольку, как правило, они привязаны именно к дугам, а не к вершинам. К тому же, если в сети есть параллельные дуги (несколько дуг соединяющих одну и ту же пару вершин), то будет гораздо легче определить, какая именно из этих дуг входит в дерево кратчайших расстояний.

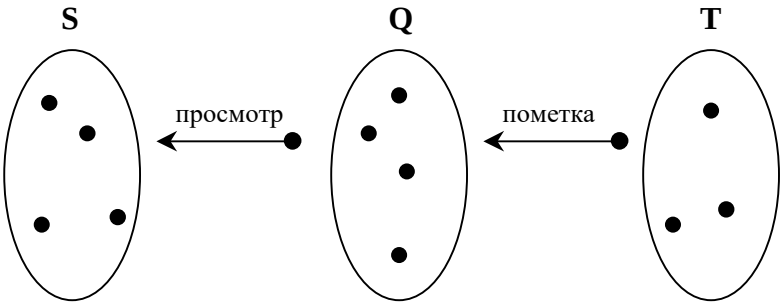
Итак, решением задачи построения дерева кратчайших путей будут два массива: `P` и `R`. В `R` хранятся величины кратчайших расстояний, а в `P` - модифицированный список предков: `P[i]` – номер дуги кратчайшего пути из s, входящей в вершину i.

Алгоритм построения кратчайших путей на сети с единичными длинами дуг – поиск в ширину

Если длины всех дуг равны единице, то длина пути равна количеству входящих в него дуг. На такой сети для построения дерева кратчайших путей можно воспользоваться обходом графа, который называется «поиск в ширину» (breadth first search).

В процессе работы алгоритма вершины могут находиться в следующих альтернативных состояниях: помечена-непомечена, просмотрена-непросмотрена. В начале работы алгоритма все вершины являются непросмотренными, а все вершины, кроме исходной - непомеченными. По ходу работы алгоритма достижимые вершины получают пометки. На очередной итерации выбираем помеченную, но непросмотренную вершину и просматриваем все выходящие из нее дуги. После этого вершина становится просмотренной и остается таковой до конца работы алгоритма.

На рисунке проиллюстрирована эволюция вершин в ходе работы алгоритма. Через S обозначено множество просмотренных вершин, через T - множество непомеченных, а через Q - множество помеченных, но непросмотренных..



Алгоритм

Строим дерево кратчайших путей от заданной вершины s до всех достижимых из s вершин. Алгоритм состоит из последовательности итераций. На каждой итерации сначала выбираем помеченную, но непросмотренную вершину, а затем просматриваем все выходящие из нее дуги.

Инициализация

Все вершины объявляем непросмотренными, а все вершины, кроме исходной вершины s - непомеченными. Вершину s помещаем в множество Q - помеченных, но непросмотренных вершин. Текущее расстояние r_i для всех $i \neq s$ делаем равным «бесконечности», а r_s полагаем равным нулю.

В качестве «бесконечности» можно использовать число, которое заведомо больше самого длинного кратчайшего пути. Поскольку количество дуг в любом пути не превышает $n-1$ (n - число вершин сети), то длина пути тоже не может быть больше $n-1$ и, значит, «бесконечностью» может служить число n.

Выбор вершины

Вершины из множества Q выбираются для просмотра в порядке очереди. То есть вершина, которая раньше была помечена, и, соответственно, раньше попала в множество Q, будет раньше выбрана для просмотра. Такой порядок выбора можно выразить следующей формулой: «первым помечен – первым просмотрен» («first labeled – first scanned»).

Выбираем для просмотра вершину i, находящуюся в начале очереди. Удаляем i из множества Q.

На первой итерации для просмотра будет выбрана вершина s, поскольку после инициализации других вершин в очереди нет.

Когда очередь станет пустой, работа алгоритма заканчивается – дерево кратчайших расстояний построено.

Просмотр ребер

Берем первую дугу, выходящую из просматриваемой вершины i. Пусть j - вершина, находящаяся на другом конце этой дуги.

Если вершина j помечена, то ничего не делаем и переходим к просмотру следующей дуги, выходящей из i.

Если j не помечена, то помечаем ее и записываем в очередь на просмотр.

Фиксируем, что в дереве кратчайших путей i является предком j, а r_j - расстояние до j - полагаем равным $r_i + 1$.

Переходим к просмотру следующей дуги.

После того, как все выходящие из i ребра обработаны, эта вершина объявляется просмотренной.

Обоснование алгоритма и оценка сложности

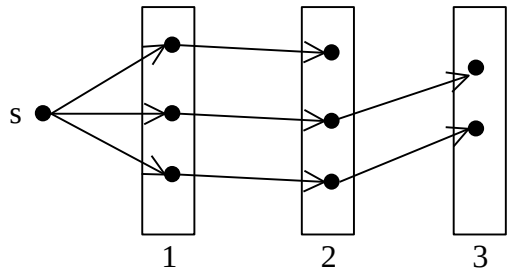
Как видите, алгоритм очень прост. Столь же просто доказательство его корректности. Рангом вершины i будем называть число дуг в пути, ведущем в дереве кратчайших путей из s в i. В сети с единичными длинами дуг ранг равен кратчайшему расстоянию, поэтому будем обозначать его тоже r_i.

Индукцией по величине ранга докажем следующие утверждения:

- Все вершины ранга k просматриваются последовательно: никакая вершина иного ранга не может быть просмотрена в промежутке между двумя вершинами ранга k.
- После завершения просмотра вершин ранга k в очереди на просмотр будут записаны только вершины ранга $k+1$, и их текущее расстояние будет равно $k+1$.

В начале работы алгоритма в очереди на просмотр находится единственная вершина s ранга 0. Поэтому первое утверждение для ранга 0 очевидно.

В каждую вершину ранга 1 ведет дуга из s. Из алгоритма просмотра следует, что после просмотра дуг, выходящих из s, все вершины первого ранга будут помечены, записаны в очередь Q, текущим расстоянием у них будет 1, и никаких других вершин в Q не будет. То есть второе утверждение для ранга 0 также выполняется.



Пусть оба утверждения выполняются для ранга k. Докажем, что они справедливы и для ранга $k+1$.

В соответствии с первым утверждением все вершины ранга k просматриваются последовательно, при этом из второго утверждения следует, что во время просмотра их текущие расстояния будут равны k. Поскольку ранг равен кратчайшему расстоянию, то из вершины ранга k дуга может вести только в вершины, ранг которых не превышает $k+1$. Так как все вершины ранга, меньшего чем $k+1$, к моменту просмотра вершин ранга k помечены, то при просмотре вершин этого ранга в очередь Q могут попасть только вершины ранга $k+1$, и их текущее расстояние при этом будет равно $k+1$. При этом после завершения просмотра вершин ранга k в очереди будут все вершины ранга $k+1$, потому что в каждую вершину этого ранга из вершины ранга k ведет дуга, принадлежащая дереву кратчайших путей.

Таким образом, после завершения просмотра вершин ранга k в очереди будут только вершины ранга $k+1$ с правильно вычисленными величинами расстояний от s. То есть второе утверждение справедливо. Правильность первого утверждения для ранга $k+1$ вытекает из того, что поскольку вершины просматриваются в порядке поступления в очередь Q, а в данный момент никаких других вершин, кроме вершин ранга $k+1$, в очереди нет, то эти вершины будут просмотрены последовательно, раньше всех остальных вершин, которые появятся в очереди потом. Оба утверждения доказаны.

Корректность алгоритма непосредственно следует из второго утверждения, которое означает, что вершина попадает в очередь на просмотр с правильно вычисленным кратчайшим расстоянием. Из описания алгоритма следует, что это расстояние уже не изменится, значит и в конце работы алгоритма оно останется правильным.

Корректность алгоритма доказана.

Теперь оценим сложность алгоритма. Каждую вершину мы просматриваем ровно один раз, следовательно, каждую дугу мы просматриваем тоже ровно один раз. То есть на просмотр дуг будет потрачено порядка $O(m)$ действий, где m - число дуг сети. На помещение вершины в очередь и выбор ее из очереди мы тратим $O(1)$ действий. Поскольку вершина может попасть в очередь на просмотр не более одного раза, то суммарное число действий, потраченных на эти операции, не превышает $O(n)$. Таким образом, общая оценка сложности алгоритма составляет $O(m) = O(m) + O(n)$

Общее описание поиска в ширину

В третьей лекции в алгоритме нахождения компонент связности был использован обход графа, называемый «поиском в глубину». В алгоритме построения кратчайших путей на сети с единичными длинами мы применили другой способ обхода графа, который называется «поиском в ширину».

При поиске в ширину должны соблюдаться следующие правила:

- Вершина может находиться в множестве T (не помечена), в множестве Q (помечена, но не просмотрена) или в множестве S (просмотрена).
- В начале обхода, как правило, одна вершина (иногда несколько) помечена, но не просмотрена, а остальные - не помечены.
- Вершины могут перемещаться между подмножествами только в указанном направлении: $T \to Q \to S$.
- Очередная вершина для просмотра выбирается из множества Q в порядке очереди. То есть соблюдается правило: «первым помечен – первым просмотрен».
- При просмотре дуг, выходящих из вершины, получают пометки те и только те вершины, находящиеся на противоположном конце просматриваемых дуг, которые не были помечены ранее.
- Во время просмотра вершины мы не можем начать просматривать другую. Просмотр вершины заканчивается и она получает статус просмотренной только после того, как просмотрены все выходящие из нее дуги.

Перечислив эти правила, мы фактически еще раз описали алгоритм построения дерева кратчайших путей на сети с единичными длинами дуг.

Если вспомнить аналогию с поведением ветреного юноши, которую мы использовали при описания поиска в глубину, то

поведение при поиске в ширину выглядит гораздо основательней. Мы не переходим к просмотру новой вершины, пока не используем «все возможности» текущей. Поэтому, в отличие от поиска в глубину при поиске в ширину имеется не стек, а «фронт», состоящий из помеченных, но непросмотренных вершин.

В чистом виде поиск в ширину используется в алгоритмах на сетях реже поиска в глубину. Однако в некоторых алгоритмах применяется нечто близкое к поиску в ширину, когда нарушаются некоторые из приведенных правил. Иногда, например, вершина может вернуться из множества S в множество Q. А иногда вершины для просмотра выбираются из множества Q не по очереди, а в соответствии с более сложными правилами. Именно такие алгоритмы построения кратчайших путей мы рассмотрим в двух следующих лекциях.