Алгоритмы на графах (семинар)

В. Матюхин

Семинар — это форма теоретического занятия, когда школьникам предлагаются различные задачи на доказательство каких-либо свойств разобранных алгоритмов, на их модификацию и т. д. Все это позволяет лучше понять как сами алгоритмы, так и области их применения и особенности реализации.

Предлагаемый семинар по графам разбит на несколько разделов. В каждом разделе сначала приводятся задачи, а затем обсуждаются их решения. Будет полезно, если перед тем, как читать разборы задач, читатель некоторое время подумает над задачами и попытается их решить самостоятельно.

Предполагается, что читателю знакомы алгоритмы обхода графа в ширину и глубину, алгоритмы поиска кратчайших путей (Дейкстры, Флойда, Форда–Белмана) и построения каркасов минимального веса (Краскала и Прима).

Раздел 1. Обход в глубину и в ширину

В задачах этого раздела предполагается, что граф связный и неориентированный.

1. Докажите, что процедура обхода *в глубину* обойдет все вершины графа (т. е. не может оказаться так, что мы не побывали в некоторой вершине).

2. Докажите, что процедура обхода *в ширину* обойдет все вершины графа.

3. Докажите, что в результате обхода (как в глубину, так и в ширину) те ребра, по которым мы проходим, образуют остовное дерево.

4. В результате обхода в глубину получается дерево. Можно рассмотреть это дерево как корневое (корнем является вершина, из которой мы начали делать обход). Есть ребра исходного графа, которые в это дерево не попали. Докажите, что все такие ребра идут из более глубокой вершины какой-либо ветви построенного дерева в менее глубокую вершину той же ветви, и не бывает ребер, соединяющих вершины, принадлежащие разным ветвям.

5. С помощью алгоритма обхода в ширину можно искать кратчайшие пути в невзвешенном графе. Обоснуйте, почему найденные пути являются кратчайшими.

Обсуждение задач раздела 1

Задача 1 очень проста и интуитивно понятна. Давайте все-таки попробуем этот факт доказать строго. Предположим, что утверждение не верно. Т. е. в какую-то вершину мы в процессе обхода не попали (пусть это вершина B). Тогда, так как граф связен, то из исходной вершины A (из которой мы запускаем алгоритм обхода) существует путь в вершину B. Рассмотрим этот путь. В вершине A наш обход побывал, а в вершине B — нет. Тогда в этом пути есть две соседние (т. е. соединенные ребром) вершины (v_1 и v_2) такие, что в первой из них обход побывал, а во второй — нет. Вот тут-то мы и получаем противоречие. Раз обход побывал в вершине v_1, то, обрабатывая эту вершину, мы перебирали всех ее соседей. В том числе и вершину v_2. А так как мы не были в вершине v_2 ранее (вспомните, по нашему предположению, в вершине v_2 мы вообще не были!), то мы должны были пойти в вершину v_2. А значит, в вершине v_2 мы бы все-таки побывали.

Решение **задачи 2** полностью аналогично.

Задача 3. Давайте чуть-чуть переформулируем утверждение, которое нужно доказать. Во-первых, как мы уже доказали выше, мы побываем во всех вершинах, а значит, ребра, по которым мы пройдем в процессе обхода, образуют связный граф. Таким образом, нам осталось доказать, что в этом графе не будет циклов. Это можно сделать по-разному. Например, предположить, что цикл есть и доказать, что тогда в процессе обхода в какую-то вершину мы приходили дважды, получив тем самым противоречие. Но есть очень изящный способ.

Заметим, что каждый раз, когда мы проходим по ребру, мы добавляем новую вершину в список пройденных. Всего в процессе обхода мы добавим к списку пройденных все вершины графа (кроме начальной — она считается пройденной изначально). А значит, мы добавим $N-1$ вершину (если считать, что всего в графе N вершин). Следовательно, мы пройдем по $N-1$ ребру. Осталось вспомнить, что связный граф из N вершин, в котором $N-1$ ребро — дерево.

Задача 4. Для решения этой задачи нужно хорошо понимать, как устроен алгоритм обхода в глубину. Предположим, что ребро, соединяющее две разные ветви дерева, есть (пусть его концы — вершины u и v). Пусть, для определенности, обход в глубину сначала побывал в вершине u, а затем — в вершине v. Причем, поскольку эти вершины по нашему предположению принадлежат разным поддеревьям, то в вершину v обход попал, когда полностью закончил рассмотрение вершины u. Тогда, перебирая в вершине u всех ее соседей, обход в глубину увидел бы вершину v (ведь они соединены ребром) и должен был бы пойти в v

(ведь мы там еще не были!), но не пошел (по нашему предположению). Противоречие.

Задача 5. Изначально мы побывали только в начальной вершине (до нее расстояние 0). Затем мы добавили всех ее соседей (расстояние до них равно 1). Затем — их соседей (вершины на расстоянии 2) и т. д. Почему для всех вершин найденное расстояние будет кратчайшим? Действительно, пусть до некоторой вершины мы нашли не кратчайшее расстояние. Рассмотрим все такие вершины и выберем наиболее близкую к начальной вершине (назовем эту вершину v). Пусть реальное расстояние до нее равно x (соответственно, когда мы нашли расстояние, оно оказалось больше, чем x). Тогда у этой вершины есть сосед (обозначим его u), до которого расстояние $x-1$, причем это расстояние найдено правильно. Тогда, рассматривая эту вершину, мы из нее пойдем в вершину v (так как в вершине v мы еще не были) и запишем в вершину v число x. Противоречие. Почему, когда мы просматриваем u, вершина v еще не просмотрена? Потому что, как отмечено вначале, мы сначала просмотрим все вершины с расстоянием 0, потом — с расстоянием 1, 2, 3 и т. д. (подумайте, почему). А, значит, когда мы будем просматривать вершину u, расстояние до которой равно $x-1$, вершину, из которой мы пришли в v по нашему предположению мы еще не просмотрим (так как, раз в v мы по предположению запишем число больше x, то расстояние до вершины, из которой мы в нее придем, будет x или больше).

Раздел 2. Алгоритмы Дейкстры, Флойда и Форда-Белмана

В этом разделе все графы предполагаются ориентированными (рассматривать неориентированные графы с отрицательными весами ребер неинтересно).

1. Как с помощью алгоритма Флойда проверить, есть ли в графе циклы отрицательного веса? Как это же проверить с помощью алгоритма Форда-Белмана?

2. Как для двух вершин a и b установить, существует ли *кратчайший* путь из a в b?

3. Постройте пример графа (с отрицательными весами ребер), в котором путь, найденный алгоритмом Дейкстры, не будет кратчайшим.

4. Можно ли модифицировать алгоритм Дейкстры, чтобы искать длины путей в графах с отрицательными весами ребер, следующим образом? Найдем сначала самое маленькое ребро в графе (пусть оно имеет вес $-a$). Теперь прибавим к весу каждого ребра число $(a+1)$. После этого все ребра будут иметь положительный вес. Теперь с помощью алгоритма Дейкстры найдем кратчайший путь в новом графе, и будем считать, что он же является кратчайшим в исходном.

5. Рассмотрим следующую задачу. Пусть каждому ребру приписан вес. Пусть стоимость пути вычисляется как произведение весов ребер пути. Можно ли применять для подсчета кратчайшего пути в этом случае алгоритмы Дейкстры, Флойда, Форда-Белмана: а) если веса всех ребер положительны? б) если веса ребер неотрицательны? в) если веса ребер произвольные?

Обсуждение задач раздела 2

Первые две задачи этого раздела являются скорее контрольными вопросами или упражнениями, в то время как задачи 3, 4 и 5 требуют довольно серьезных размышлений.

Задача 1. Начнем с алгоритма Флойда. Что такое цикл отрицательного веса? Это путь из вершины в саму себя. А значит, нужно лишь просмотреть диагональ построенной алгоритмом Флойда матрицы и проверить, есть ли на ней отрицательные числа (подумайте, почему если цикл отрицательного веса существует, он будет найден алгоритмом Флойда).

С алгоритмом Форда-Белмана чуть сложнее. Во-первых, если окажется, что цикл отрицательного веса недостижим из той вершины, которая является начальной для алгоритма, то мы никак не узнаем о его существовании. Если же цикл отрицательного веса есть, то узнать об этом довольно просто. Напомним, что алгоритм Форда-Белмана состоит из $N-1$ шага, на каждом из которых мы перебираем все ребра и обновляем те значения расстояний до вершин, которые удается улучшить. Сделаем еще один аналогичный (N-й) шаг. Оказывается, что если на этом шаге меняется хотя бы одно из значений, то в графе есть цикл отрицательного веса, если нет — то циклов отрицательного веса, достижимых из начальной вершины, нет. Действительно, если в графе нет цикла отрицательного веса, то любой кратчайший путь из начальной вершины куда-либо состоит не более, чем из $N-1$ ребра, и, следовательно, будет найден за $N-1$ шаг алгоритма. При дальнейших шагах значения меняться уже не будут. Если же в графе есть цикл отрицательного веса, то нам выгодно «бегать» по этому циклу сколь угодно долго, и от этого расстояния до каких-то вершин будут уменьшаться. А значит, такое уменьшение какого-либо из значений произойдет и на N-м шаге (ведь если на каком-то шаге ни одно значение не меняется, то, из-за того, что шаги делаются одинаково, дальше уже ничего никогда меняться не будет).

Задача 2. Формулировка задачи кажется довольно странной. В то же время, задача отнюдь не так тривиальна, как кажется на первый взгляд. Итак, что же нужно проверить, чтобы с уверенностью уметь сказать, существует ли кратчайший путь между двумя вершинами или

нет? Первое, что нужно проверить, это существует ли вообще путь из u в v. Если пути нет, то, естественно, нет и кратчайшего пути.

Как же может случиться, что путь есть, а кратчайшего пути нет? Такое бывает, когда могут быть пути сколь угодно маленькой длины (т. е. в графе есть цикл отрицательного веса). Тогда какой бы путь мы ни приняли за кратчайший, всегда можно найти путь еще меньшей длины.

Оказывается, впрочем, что просто проверить граф на наличие циклов отрицательного веса недостаточно. Может так случиться, что в графе есть цикл отрицательного веса, и в то же время кратчайший путь из u в v все-таки существует (остановитесь на минуту и подумайте, как такое возможно).

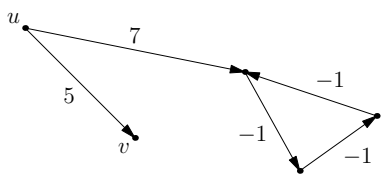
Итак, чтобы можно было найти путь из u в v сколь угодно маленькой длины, должны выполняться следующие условия:

— в графе существует цикл отрицательного веса;

— этот цикл достижим из вершины u;

— из этого цикла достижима вершина v.

Действительно, пусть в графе есть цикл отрицательного веса. Но если из него нельзя попасть в вершину v (см. рис.), то для уменьшения пути из u в v этот цикл использовать не удастся, и кратчайший путь из u в v вполне может существовать.

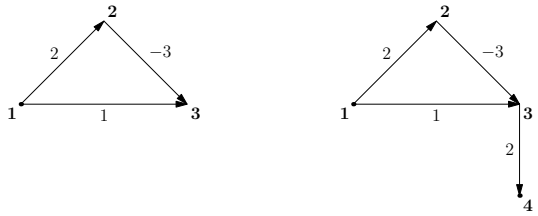


Теперь давайте сформулируем, как все-таки проверить существование кратчайшего пути из u в v, если мы пользуемся алгоритмом Флойда. Алгоритмом Флойда построим матрицу расстояний между всеми парами вершин. Дальше проверяем:

— Если пути из u в v не существует, то кратчайшего пути также не существует.

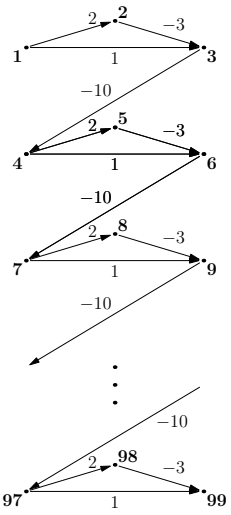
— Если удается найти такую вершину w (просто перебираем все вершины, и для каждой из них проверяем условие), что существует путь из u в w, путь из w в w имеет отрицательный вес (что соответствует циклу отрицательного веса, проходящему через w) и существует путь из w в v, то путь из u в v может иметь сколь угодно маленький вес.

Если вершины w с описанным свойством нет, то кратчайший путь существует.



Задача 3. Как правило, первый пример, который удается найти в этой задаче, выглядит примерно так, как показано на рисунке слева. Однако здесь можно предложить модифицировать алгоритм так, чтобы значения в том числе и в тех вершинах, из которых мы уже «ходили» тоже обновлялись. Впрочем, тогда нетрудно модифицировать и предложенный пример, добавив к нему еще одно ребро (см. рис. справа). Очевидно, что на первом шаге мы найдем расстояния до вершин 2 (расстояние 2) и 3 (расстояние 1). Далее мы будем ходить из вершины 1, и найдем расстояние до вершины 4 (оно равно $1 + 2 = 3$). Далее мы будем ходить из вершины 2, и обновим значение в вершине 3 (записав туда $2 + (-3) = -1$). Но, так как из 3-й вершины мы уже ходили, то значение в вершине 4 так и останется равно 3, и путь длины 1 (через вершины 2 и 3) мы не найдем.

Здесь хочется сформулировать еще одну **задачу 3′**. Давайте модифицируем алгоритм следующим образом. Если мы обновляем значение в вершине, из которой мы уже ходили, мы снимаем пометку о том, что мы из нее ходили и теперь (когда до нее дойдет очередь) мы будем ходить из этой вершины еще раз. В этом случае в приведенном примере алгоритм будет давать правильный ответ, как и в других случаях. Что же получается, мы придумали модификацию алгоритма Дейкстры для графов с отрицательными весами ребер?



И да, и нет.

Рассмотрим пример,изображенный на рисунке (например, пусть граф будет состоять из 99 вершин, хотя, понятно, что его можно продлить и дальше).

Рассмотрим, как будет работать алгоритм Дейкстры. Сначала он пойдет из вершины 1, затем — из 3, затем — из 4, из 6, из 7, из 9, ..., из 97, из 99. Далее алгоритм сделает ход из вершины 98, затем снова из 99. Далее алгоритм будет ходить из вершины 95 (в ней среди вершин, из которых мы еще не ходили, на данный момент будет записано самое маленькое число). Далее (поскольку это обновит

значение в вершине 96), мы будем ходить из 96 — мы обновим значение в 97, далее будем ходить из 97, из 99, из 98. Далее мы будем ходить из вершины 92, от этого обновится 93, пойдем из 93, дойдем до самого низа, потом обновим нижний треугольник, опять пойдем из вершины 95, в результате чего опять обновим нижний треугольник. И так далее. Каждый раз, ходя из вершины какого-либо треугольника, мы будем доходить до низу, обновлять нижний треугольник, далее ходить из предпоследнего треугольника и снова обновлять нижний треугольник. После чего ходить из третьего снизу треугольника, опять доходить до низу, опять обновлять второй снизу и опять доходить до низу и т. д. Нетрудно заметить, что этот алгоритм будет иметь уже не полиномиальную, а экспоненциальную сложность (докажите это!)

Задача 4. Задача довольно интересна тем, что этот метод талантливые школьники часто придумывают сами. Кажется, от того, что мы все ребра графа увеличили на одну и ту же величину, ничего принципиально не изменилось, и кратчайший путь должен остаться кратчайшим путем. Оказывается, что это все-таки не так. Пусть мы увеличили длины всех ребер на величину b. Тогда длина пути, состоящего из одного ребра, увеличится на b, пути, состоящего из двух ребер — на $2b$, из трех — на $3b$. За счет этого кратчайший путь в исходном графе может оказаться не самым коротким в модифицированном. Например, пусть в графе было три ребра: ребро $(1, 2)$ веса 1, ребро $(1, 3)$ веса 5, ребро $(3, 2)$ веса -50. Тогда кратчайшим путем из 1 в 2 является путь через вершину 3 (его вес равен -45). Однако если веса всех ребер увеличить, как предлагается, на 51, то ребро $(1, 2)$ будет весить 52, ребро $(1, 3)$ будет весить 56, ребро $(3, 2)$ будет весить 1. Тогда вес пути напрямую будет 52, а вес пути через вершину 3 — 57. То есть кратчайшим в этом графе будет прямой путь.

Задача 5. Пункт а. Пусть веса всех ребер положительны. Припишем каждому ребру новый вес — логарифм по некоторому (одинаковому для всех) основанию от исходного веса ребра. Для определенности возьмем логарифм по основанию 2. Заметим теперь, что если у нас есть путь и мы просуммируем новые веса ребер вдоль него, то чтобы получить вес пути в исходной задаче, нам достаточно 2 возвести в получившуюся степень. Действительно, пусть путь проходит через три ребра, имеющие веса a, b, c. Тогда вес этого пути равен abc. Но эту же величину можно записать как $2^{\log a+\log b+\log c}$. Заметим, что в силу монотонности, чем больше сумма логарифмов, тем больше и произведение. Т. е. кратчайший путь в новом смысле является кратчайшим путем в исходной задаче.

Тем самым, для решения исходной задачи мы можем заменить веса ребер их логарифмами, а правило вычисления стоимости пути — на

стандартное правило суммирования весов ребер. А дальше можем применить стандартные алгоритмы: Флойда, Форда-Белмана, а если все исходные веса не меньше 1, то и алгоритм Дейкстры.

Пункт б. Если исходные веса ребер могут быть как положительны, так и 0, то взять логарифмы от ребер веса 0 не получится. При этом реально можно их считать минус бесконечностью (очень маленьким числом). А можно подойти и по-другому. Сначала попробовать найти путь, проходящий через нулевое ребро (его вес будет равен 0, веса всех остальных путей заведомо не меньше). А если найти путь с нулевым ребром не получилось, то можно из графа удалить все ребра веса 0, после чего получится задача из пункта а.

Кстати, подумайте, как найти какой-нибудь путь, проходящий через нулевое ребро.

Пункт в мы оставим читателю для самостоятельного исследования.

Раздел 3. Ориентированные графы

1. Докажите, что если ориентированный граф не содержит ориентированных циклов, то топологическая сортировка его вершин возможна. Иначе говоря, вершины графа можно пронумеровать натуральными числами от 1 до N так, чтобы все дуги (дугами принято называть ребра ориентированного графа) шли из вершин с меньшим номером в вершины с большим.

2. Пусть в ориентированном графе для любых двух вершин i и j есть либо дуга из i в j, либо из j в i (такой граф называется турниром). Докажите, что в этом графе можно построить гамильтонову цепь (т. е. путь, проходящий через каждую вершину ровно один раз). Иначе говоря, если N команд сыграли полный турнир (каждая с каждой) без ничьих (результаты можно изобразить графом, проведя дуги от выигравшей команды к проигравшей), то эти команды можно выстроить в шеренгу так, что каждая команда проиграла своему левому соседу в шеренге, и выиграла у правого.

3. Граф задан матрицей смежности. Требуется за $O(N)$ найти вершину, в которую входят ребра из всех вершин и из которой не выходит ни одного ребра, либо установить, что такой вершины нет.

Обсуждение задач раздела 3

Задача 1. Утверждение. *Если в графе нет ориентированных циклов, то существует вершина, в которую не входит ни одна дуга.* Действительно, если в каждую вершину входит хотя бы одна дуга, то можно сделать следующее. Встанем в какой-нибудь вершине, и пойдем в любую вершину, из которой в данную вершину есть дуга (т. е. мы идем в направлении, обратном направлению дуг), оттуда — опять же в вершину,

из которой есть дуга в эту вершину и т. д. Так как в каждую вершину входит хотя бы одна дуга, то процесс бесконечен. Так как вершин в графе N, то рано ли поздно мы окажемся в вершине, в которой мы уже были. А тогда в графе есть цикл (мы прошли по нему в обратном направлении). Утверждение доказано.

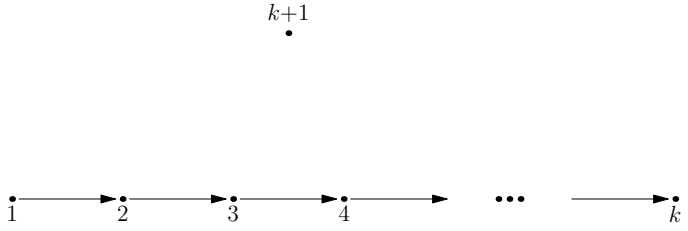
Теперь рассмотрим такой алгоритм. Найдем вершину, в которую ничего не входит. Поставим в нее число 1, и удалим эту вершину, а также все дуги, которые из нее выходили. Оставшийся граф также не будет содержать ориентированных циклов (откуда бы им взяться?), а значит, в нем будет вершина, в которую не входит ни одна дуга (в исходном графе в эту вершину могло не входить ни одной дуги, а могла входить дуга из вершины, которую мы нашли первой). Поставим в эту вершину число 2 и опять же удалим ее. Продолжим процесс. На каждом шаге нам удастся находить вершину, в которую не входит ни одной дуги, и ставить туда очередное число. Тем самым мы пронумеруем все вершины. Почему эта нумерация будет правильной? Потому что мы ставили в вершину очередное число, когда в нее не вело ни одной дуги, а это значит, что к этому моменту все вершины, из которых в исходном графе есть дуги в данную вершину, были уже из графа удалены, т.е. получили свои номера, и эти номера были меньше номера данной вершины.

Задача 2. Воспользуемся методом математической индукции.

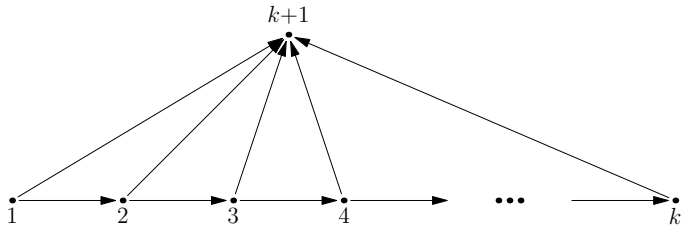
Давайте сначала оставим только две вершины. Между ними есть или дуга от первой ко второй, или от второй к первой. Таким образом, для двух вершин существует путь, проходящий через обе эти вершины.

Давайте предположим, что мы уже доказали, что для K вершин утверждение задачи верно. Докажем теперь, что оно верно для $K+1$ вершины.

Удалим вершину $K+1$ и все дуги, выходящие из нее или входящие в нее. Останется граф с тем же свойством на K вершинах. Для этого графа можно построить путь, проходящий через все K вершин (по предположению индукции). Давайте пронумеруем эти K вершин номерами от 1 до K в соответствии с их номером в пути. Теперь рассмотрим $(K+1)$-ю вершину:



Если из вершины $K+1$ идет дуга в вершину 1, то искомым является путь $(K+1, 1, 2, \ldots, K)$. Если же, наоборот, дуга идет из 1 в $K+1$, то рассмотрим, в какую сторону направлена дуга между вершинами 2 и $K+1$. Если от $K+1$ в 2, то искомым является путь $(1, K+1, 2, 3, \ldots, K)$, если же от 2 в $K+1$, то рассмотрим дугу между 3 и $K+1$. И так далее, если в какой-то момент нам встретится ситуация, что из вершины i идет дуга в $K+1$, а из $K+1$ — в $i+1$, то искомый путь будет существовать: $(1, 2, \ldots, i, K+1, i+1, i+2, \ldots, K)$. Поскольку мы рассматриваем ситуацию, когда из вершины 1 идет дуга в $K+1$ (обратную ситуацию мы рассмотрели в самом начале), то единственная возможная плохая для нас ситуация, когда такого нет, когда из всех вершин идут дуги в $K+1$ (см. рис. ниже) В том числе и из последней, K-й вершины. Но тогда искомый путь все равно существует: это путь $(1, 2, \ldots, K, K+1)$.



Задача 3. Давайте договоримся, что в матрице смежности в i-й строке будут стоять единички в тех столбцах, которые соответствуют вершинам, в которые из вершины i идут дуги.

Будем считать, что на главной диагонали матрицы стоят 0.

Давайте сформулируем в терминах матрицы смежности, что же нам нужно найти. Нам нужно найти некоторую i-ю строчку, в которой стоят все 0, а в i-м столбце стоят все 1 (за исключением диагонального элемента). В качестве упражнения докажите, что в графе не может быть двух вершин с таким свойством.

Рассмотрим такой алгоритм. Встанем в первую строку, и пойдем по ней начиная с первого элемента. Как только нам встретится 1 (пусть это произошло в j-м столбце) мы перейдем на j- ю строчку и продолжим ее проверку с $j+1$-го элемента. Если нам еще встретится 1, мы снова перейдем на соответствующую строчку и продолжим ее проверку. Как только мы дойдем до конца (N-го элемента строки), так можно утверждать, что данная вершина является единственным кандидатом. Нам останется проверить целиком строчку, в которой мы оказались и соответствующий ей столбец. Если они удовлетворяют условию, то мы нашли то, что искали, если нет, то того, что мы ищем — нет (при этом

если при проверке в строке окажутся единицы, то переходить на другие строки уже не надо).

	1	**2**	**3**	**4**	**5**
1	0	0	1	1	1
2	0	0	0	1	0
3	1	0	0	1	1
4	0	0	0	0	0
5	0	1	0	1	0

Заметьте, что первый проход по строке (возможно, с переходом на строчки вниз) займет N операций, еще N операций потребуется для проверки строки, которую мы найдем и еще N — для проверки столбца. Тем самым, за $3N$ проверок мы решим нашу задачу. В качестве самостоятельного упражнения докажите, почему этот алгоритм всегда найдет искомую вершину, почему он правильно устанавливает, когда ее нет.

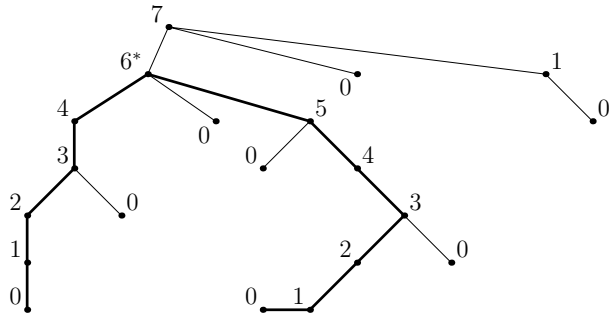
Раздел 4. Каркасы и деревья

1. Докажите, что алгоритм Краскала строит именно минимальный каркас.

2. Докажите, что алгоритм Прима строит минимальный каркас.

3. В дереве требуется найти две наиболее удаленные друг от друга вершины. Можно ли это сделать за $O(N)$? Если да, то каким образом и как для этого должно быть представлено дерево в памяти?

Задачи 1 и 2 описаны в литературе.



Задача 3. Подвесим дерево за какую-нибудь вершину неединичной степени. Тогда рассмотрим, как будет устроен самый длинный путь —

он будет от некоторого листа подниматься до некоторой вершины, после чего опускаться до другого листа. В нем есть некоторая самая высокая вершина (будем называть ее «переломной»), и он распадается на два подпути, каждый из которых идет снизу вверх, и эти пути в своих поддеревьях являются самыми длинными (см. рис.).

Давайте из вершины, за которую мы довесили дерево, запустим процедуру обхода в глубину. Пусть эта процедура для каждой вершины возвращает длину самого длинного пути от этой вершины до некоторого листа (на рисунке около каждой вершины написано значение, которое вычислить эта процедура). Понятно, что процедура обхода в глубину это значение может легко вычислять (как максимум из аналогичных значений для поддеревьев плюс 1).

Помимо этого, обход в глубину будет вычислять длину самого длинного пути, в которой данная вершина является «переломной» (и запоминать наибольшее найденное значение в некоторой глобальной переменной). Нужно найти два поддерева с наибольшими длинами наибольших путей, просуммировать эти длины и добавить к ним 2 (ребра от данной вершины до вершин, где начинаются эти деревья). На приведенном рисунке «переломной» для самого длинного пути будет вершина, помеченная звездочкой, а длина самого длинного пути равна $4 + 5 + 2$.

Сложность алгоритма равна сложности обхода в глубину и при хранении графа списком ребер, выходящих из каждой вершины, достигает $O(N + M)$, а в дереве, как известно, $M = N - 1$, и, тем самым, сложность равна $O(N)$.