Лекция 13

Разрешающие деревья

13.1 Задача об угадывании числа. Деление пополам. Мощностная нижняя оценка

Рассмотрим следующую игру. Алиса загадывает натуральное число от 1 до N, а Боб пытается это число отгадать. При этом Бобу разрешается задавать вопросы, на которые Алиса может ответить "да" или "нет", и Алиса должна на эти вопросы давать правильные ответы. Цель Боба состоит в том, чтобы задать как можно меньше вопросов. При этом мы не хотим полагаться на удачу, то есть нужно, чтобы число вопрос было гарантировано небольшим. Другими словами, мы хотим найти такое минимальное k, что у Боба есть алгоритм, позволяющий отгадать число за не более чем k вопросов, какое бы число ни загадала Алиса.

Оказывается, что Бобу всегда достаточно задать не более чем $\lceil \log_2 N \rceil$ вопросов. Чтобы это доказать мы воспользуемся *методом деления пополам*. Идея в том, что Боб каждым своим вопросом будет сокращать количество оставшихся возможных чисел примерно в два раза. Этого можно добиться, например, так. Обозначим число, загаданное Алисой, через x. На каждом шаге Боб будет знать, что x лежит в некотором "отрезке" $\{y \mid a \leqslant y \leqslant b\}$ для каких-то a и b. Изначально $a = 1$ и $b = N$. На очередном шаге Боб будет вычислять $c = \lfloor (a + b)/2 \rfloor$ и спрашивать, верно ли, что $x \leqslant c$. Если Алиса отвечает "да", то Боб переходит к отрезку $\{y \mid a \leqslant y \leqslant c\}$ и повторяет процедуру. Иначе, Боб переходит к отрезку $\{y \mid c + 1 \leqslant y \leqslant b\}$ и также повторяет процедуру. Нетрудно видеть, что каждый раз длина отрезка уменьшается почти в два раза (если в отрезке было нечетное число точек, то в следующем отрезке может оказаться чуть больше чем половина точек). Так что через приблизительно $\log_2 N$ шагов в отрезке останется одна точка и Боб узнает число Алисы. На самом деле, нетрудно доказать, что достаточно $\lceil \log_2 N \rceil$ вопросов, что мы сейчас и сделаем.

Чтобы максимально упростить оценку числа вопросов мы прибегнем к небольшому трюку и немного модифицируем алгоритм. Обозначим $k = \lceil \log_2 N \rceil$ и $N' = 2^k$. Видно, что $N' \geqslant N$. Пусть теперь Боб изначально считает, что Алиса может загадать число от 1 до N'. Поскольку на самом деле Алиса может загадывать только

числа от 1 до N, то Боб только рассматривает дополнительные возможности, которые никогда не будут реализовываться, и тем самым, Боб только усложняет свою задачу. Но при этом видно, что если Боб в предыдущем алгоритме начнет с отрезка $\{y \mid 1 \leqslant y \leqslant N'\}$, то на каждом шаге в каждом отрезке будет четное число точек и каждый отрезок будет делиться ровно пополам (N' – степень двойки). Так что, чтобы длина отрезка стала равной 1, потребуется ровно $\log_2 N' = k$ вопросов.

Задача 13.1. Мы доказали, что в модифицированном алгоритме потребуется не более $\lceil \log_2 N \rceil$ вопросов, но не доказали, что для изначального (не модифицированного) алгоритма верна такая же оценка (почему?). Докажите, что и для изначального алгоритма эта оценка верна.

Оказывается, что доказанная только что оценка точная: не существует алгоритма, который для всякого загаданного Алисой числа задавал бы меньше $\lceil \log_2 N \rceil$ вопросов. Сейчас мы это докажем, то есть мы докажем, что если вновь через k обозначить минимальное число вопросов, то $k \geqslant \lceil \log_2 N \rceil$. Таким образом, мы докажем *нижнюю оценку* сложности нашей задачи.

Для доказательства нижней оценки мы применим так называемый *мощностной метод*. Пусть у Боба есть какой-то алгоритм сложности k (то есть, в нем всегда задается не более k вопросов), Алиса загадала какое-то число и Боб задал свои вопросы. Рассмотрим цепочку ответов Алисы. Для удобства будем обозначать ответ “да” цифрой 1, а ответ “нет” цифрой 0. Тогда последовательность ответов Алисы – это последовательность из 0 и 1 длины не больше k. Заметим, что для двух разных загаданных Алисой чисел последовательности не могут совпадать. Действительно, если для двух различных x и y Алиса дает Бобу на его вопросы полностью одинаковые ответы, то для Боба случаи этих чисел не различимы: его диалоги с Алисой для x и для y выглядят одинаково. При этом Боб после этого диалога выдает какой-то ответ, который определяется только состоявшимся диалогом. Значит в одном из случаев его ответ будет неправильным. Далее, заметим, что не может быть так, что для двух различных x и y, загаданных Алисой, цепочка ответов для x является началом цепочки ответов для y. Действительно, иначе диалог Боба с Алисой выглядит одинаково для x и y до того момента, когда будут заданы все вопросы из цепочки ответов для x. Значит к этому моменту Боб не может отличить x от y и должен делать для них одно и то же, тогда как он в одном случае задает следующий вопрос, а в другом нет.

Таким образом, мы получили, что каждому числу от 1 до N соответствует последовательность из не более чем k нулей и единиц, все эти последовательности различны, и ни одна не является началом другой. Заметим, что семейство этих последовательностей содержит не более 2^k элементов. Действительно, если какая-то из них имеет длину меньше k, то продолжим ее, например, нулями. Тогда для различных x и y полученные последовательности длины k различны: иначе они либо совпадают, либо одна (более короткая) является началом другой. Таким образом, каждому числу от 1 до N соответствует последовательность длины k из нулей и единиц, и все эти последовательности различны. Всего последовательностей длины

k из нулей и единиц 2^k. По принципу Дирихле, чисел от 1 до N должно быть не больше 2^k (иначе двум разным числам соответствуют одинаковые последовательности). Значит $N \leqslant 2^k$, то есть $k \geqslant \log_2 N$. Поскольку k –целое число, то отсюда следует, что $k \geqslant \lceil \log_2 N \rceil$.

Таким образом, мы получили следующий результат.

Лемма 13.1. *Для угадывания числа от* 1 *до* N *необходимо и достаточно* $\lceil \log_2 N \rceil$ *вопросов.*

13.2 Формализация модели

Рассуждения прошлого раздела не очень формальны. Например, мы не определяли, что мы подразумеваем под алгоритмом. В этом разделе мы формализуем задачу и заодно рассмотрим ее общую постановку.

Начнем с общей постановки задачи. Пусть фиксирована некоторая функция $f\colon A \to B$, где A, B – какие-то конечные множества. На вход подается произвольный $x \in A$, требуется найти $f(x)$. При этом разрешается задавать вопросы вида $x \in S$ для подмножеств S множества A. Можно заметить, что в примере с угадыванием числа любой вопрос с ответом “да” или “нет” сводится к вопросу о том, принадлежит ли загаданное число некоторому подмножеству – подмножеству тех чисел, для которых ответ “да”.

Теперь формализуем нашу вычислительную модель. Протоколом вычисления мы будем называть двоичное дерево. Каждая промежуточная вершина дерева (не лист) помечена некоторым подмножеством $S \subseteq A$. Каждый лист помечен элементом $b \in B$. Из каждой промежуточной вершины, выходит три ребра: одно к корню и два к листьям. Для каждой промежуточной вершины одно из ребер, ведущих к листьям, помечено единицей, а другое – нулем.

Вычисление согласно протоколу происходит следующим образом. Мы будем строить путь из корня дерева в какой-то из листьев. Элемент множества B, которым помечен лист, будет результатом вычисления. В начале пути мы находимся в корне дерева. Корень, как и всякая промежуточная вершина, помечен каким-то подмножеством множества A. Если вход x принадлежит этому подмножеству, то мы переходим по ребру, помеченному единицей, иначе – по ребру помеченному нулем. Если следующая вершина – лист, то путь построен. Если же она является промежуточной вершиной, то мы повторяем процедуру: спрашиваем, лежит ли x в подмножестве, которым помечена текущая вершина, и переходим по ребру, помеченному единицей, если x лежит в подмножестве, и по ребру, помеченному нулем, иначе. Мы говорим, что протокол вычисляет функцию f, если для всякого $x \in A$ протокол выдает $f(x)$. Сложностью протокола называется глубина дерева (нетрудно видеть, что она равна числу вопросов, которое потребуется задать в худшем случае).

13.3 Угадывание числа, неадаптивный вариант задачи

Когда мы рассматривали задачу об угадывании числа, то следующие вопросы могли зависеть от ответов на предыдущие. Такие вычислительные модели обычно называют *адаптивными*. Можно также рассмотреть и неадаптивную постановку той же задачи: в ней вопросы не должны зависеть от ответов на предыдущие вопросы. Можно считать, что Боб должен составить на бумажке список вопросов и передать его Алисе. Алиса должна ответить на все эти вопросы и после этого Боб должен назвать загаданное число.

Во-первых, можно заметить, что неадаптивная модель слабее адаптивной: задача Боба стала только сложнее. Это значит, что в неадаптивной модели Бобу потребуется не меньше вопросов, чем в адаптивной. Действительно, если Боб может угадать число за k вопросов в неадаптивной модели, то он может сделать то же самое и в адаптивной модели – достаточно просто задать те же вопросы.

Таким образом, в новой модели Бобу также потребуется не меньше $\lceil \log_2 N \rceil$ вопросов, чтобы угадать число от 1 до N. Но хватит ли такого числа вопросов и в этой модели?

Оказывается, что ответ на этот вопрос положительный. Есть несколько способов объяснить, почему это так. Мы приведем два.

Первое рассуждение – прямое и универсальное. Можно заметить, что рассуждение для адаптивного алгоритма напрямую не проходит – вопросы зависят от ответов на предыдущие. Но это можно исправить за счет удлинения и усложнения вопросов. Для этого достаточно добавить перебор случаев в вопросы. Первый вопрос остается таким же, как и в адаптивном протоколе. Второй вопрос будет иметь следующий вид:

> “Если ответ на первый вопрос был ‘да’, то верно ли, что ..., если же ответ на первый вопрос был ‘нет’, то верно ли, что ... ?”,

где на места многоточий нужно подставить вторые вопросы из дерева адаптивного протокола. В общем виде, l-ый вопрос будет иметь следующий вид:

> “Верно ли следующее утверждение: (ответы на предыдущие вопросы были ‘нет’, ‘нет’, ..., ‘нет’ и ...) или (ответы на предыдущие вопросы были ‘нет’, ‘нет’, ..., ‘да’ и ...) или ... или (ответы на предыдущие вопросы были ‘да’, ‘да’, ..., ‘да’ и ...)”,

где в ‘или’ перебираются все варианты ответов на предыдущие вопросы, а вместо многоточий в скобках нужно подставить соответствующие вопросы из адаптивного алгоритма. Видно, что вопросы получаются очень длинными, но у нас нет ограничений на длину вопроса.

Предыдущее рассуждение универсально в том смысле, что оно работает для любой задачи такого типа и любого адаптивного алгоритма. В нашей частной задаче об угадывании числа это же рассуждение можно изложить очень коротко и просто:

i-ым вопросом Боб спрашивает, верно ли, что i-ый бит двоичной записи загаданного числа x – единица. После $k = \lceil \log_2 N \rceil$ вопросов Боб знает всю двоичную запись числа x, а значит знает само загаданное число.

Задача 13.2. Осознайте, что два приведенных выше алгоритма – по существу один и тот же.

13.4 Ограниченные модели деревьев разрешения. Сортировка, взвешивания, булевы функции

Часто кроме описанной выше общей модели рассматриваются модели деревьев разрешения с разными ограничениями. В большинстве из них накладываются ограничения на множества S, о которых можно задавать вопросы. Мы рассмотрим несколько таких примеров.

Сортировка. Первый пример – это задача о сортировке. Неформальная формулировка этой задачи такая. Дано n объектов, все разного веса. За один шаг разрешается сравнить веса двух объектов (мы узнаем, какой из этих объектов тяжелее). Требуется расположить эти объекты в порядке возрастания веса.

Опишем теперь задачу формально. Удобно считать, что объекты изначально расположены в виде последовательности. Обозначим в этой последовательности самый тяжелый объект единицей, второй по тяжести – двойкой, и так далее, самый легкий объект обозначим n. Таким образом, нам на вход по существу подается перестановка n-элементного множества. Чтобы упорядочить объекты по возрастанию нам нужно найти данную перестановку. Таким образом, в этом примере A – множество перестановок n-элементного множества и требуется вычислить тождественную функцию на A, то есть $f(x) = x$ для всякого $x \in A$.

При этом нам разрешается задавать не любые вопросы, а только вопросы о сравнении двух элементов перестановки. Формально это означает, что в вершинах разрешающего дерева могут стоять не любые подмножества множества перестановок, а только множества $S_{i,j}$ для $i, j = 1, \ldots, n$, состоящие из всех перестановок $(a_1, \ldots, a_n)$, в которых $a_i > a_j$.

Заметим, что если бы не было ограничения на вид множеств, то задача была бы полностью аналогична задаче об угадывании числа: на вход подается один из $n!$ объектов и требуется угадать, какой именно. Поскольку у нас добавляется ограничение на тип вопросов, то наша задача усложняется, а значит в задаче о сортировке требуется не меньше вопросов.

Следствие 13.2. *Сложность задачи о сортировке n объектов не меньше* $\lceil \log_2 n! \rceil$.

Что касается верхней оценки, то из оценки для задачи угадывания числа так сразу ничего не следует. Мы можем доказать следующую оценку.

Лемма 13.3. *Сложность задачи о сортировке n объектов не больше* $\sum_{k=1}^{n} \lceil \log_2 k \rceil$.

Доказательство. Доказательство будем вести индукцией по n. Для $n = 1$ оценка верна – никаких сравнений не требуется.

Пусть утверждение доказано для n, докажем его для $n+1$. Сначала возьмем первые n объект и упорядочим их, пользуясь предположением индукции. После этого у нас остается $\lceil \log_2(n+1) \rceil$ сравнений и нам нужно одну оставшийся объект поместить в уже упорядоченный список из n объектов. То есть, для $(n+1)$-го объекта есть $n+1$ место среди упорядоченного списка из n объектов и нам нужно его найти. Пользуясь тем же рассуждением, что и в задаче про угадывание числа это можно сделать как раз за $\lceil \log_2(n+1) \rceil$ сравнение (сравниваем со средним объектом и сокращаем количество возможных позиций почти в два раза). □

Итак, мы получили верхнюю и нижнюю оценку числа сравнений, необходимого для сортировки n объектов. Можно заметить, что наша верхняя оценка есть $O(n \log n)$, а наша нижняя оценка есть $\Omega(n \log n)$. Так что порядок роста сложности задачи о сортировке при росте n в нашей модели мы установили. Тем не менее, наши верхняя и нижняя оценка не совпадают. Более того, они обе не точны. Для $n = 5$ наша верхняя оценка дает 8 сравнений, тогда как можно убедиться, что сортировку можно сделать за 7 сравнений (попробуйте это сделать). Для $n = 12$ наша нижняя оценка дает 29 сравнений, тогда как на самом деле за 29 сравнений этого сделать также нельзя (в этом можно убедиться компьютерным перебором). Точное значение числа сравнений необходимое и достаточное для сортировки для произвольного n не известно.

Взвешивания. Второй пример – это задачи на взвешивание. Для примера мы рассмотрим задачу поиска самого тяжелого из n объектов разного веса. Более точно, есть n объектов, за один ход разрешается сравнить по весу два из них. Требуется найти самый тяжелый объект. Видно, что формально модель точно такая же, как и в предыдущем примере. Меняется только функция, которую мы хотим вычислить: теперь по данной перестановке мы хотим найти номер позиции, в которой стоит число 1.

Лемма 13.4. *Для нахождения самого тяжелого из n объектов необходимо и достаточно $n - 1$ взвешивания.*

Доказательство. Сначала докажем, что $n-1$ взвешивания достаточно. Проще всего вести рассуждение по индукции. Если $n = 1$, то ничего взвешивать не нужно. Пусть мы доказали утверждение для $n - 1$. Рассмотрим n объектов. Возьмем любые два и сравним их. Заметим, что более легкий из них не может быть самым тяжелым, так что его можно выбросить из рассмотрения. Таким образом у нас остается $n - 1$ объект и по предположению индукции мы можем найти самый тяжелый из них за $n - 2$ оставшихся взвешивания.

Теперь докажем, что меньше чем за $n - 1$ взвешивание найти самый тяжелый объект нельзя. Пусть мы сделали $n - 2$ взвешивания. Рассмотрим следующий граф. Его вершинами будут наши объекты, и мы соединяем ребрами те из них, которые

мы сравнили в одном из взвешиваний. Тогда в этом графе n вершин и $n-2$ ребра. Значит этот граф не связен. Рассмотрим множество V_1 объектов в одной из его компонент связности и множество V_2 всех остальных объектов. Предположим, для определенности, что самый тяжелый объект находится в V_1. Увеличим вес всех объектов в V_2 на одно и то же очень большое число, такое чтобы все объекты в V_2 стали тяжелее всех объектов в V_1. При этом результаты всех взвешиваний не изменятся, поскольку все сравнения были либо внутри V_1, либо внутри V_2, а самая тяжелая монета станет другой (теперь она будет в V_2). Таким образом, все взвешивания дадут один и тот же результат в обеих ситуациях, а самый тяжелый объект будет разным. Значит в одной из двух ситуаций наш протокол выдает неправильный ответ. Мы пришли к противоречию, а значит для нахождения самого тяжелого объекта требуется не меньше $n-1$ сравнения. □

В этом месте уместно вспомнить о неадаптивной модели. Мы видели, что в общей модели деревьев разрешения разницы между адаптивной и неадаптивной моделью нет – сложность в обоих случаях получается одинаковая. Но это достигается за счет использования весьма нетривиальных запросов в неадаптивной модели. Так что, если мы добавляем ограничения на вид запросов, то естественно ожидать, что разница между адаптивной и неадаптивной моделью все же появится. Это хорошо видно на примере задачи поиска самого тяжелого объекта.

Лемма 13.5. *Для нахождения самого тяжелого из n объектов в неадаптивной модели необходимо и достаточно $\binom{n}{2} = n(n-1)/2$ взвешиваний.*

Доказательство. Верхняя оценка здесь получается совсем просто – достаточно сравнить попарно все объекты друг с другом. Ясно, что если мы сравнили все объекты, то мы можем сказать, какой из них самый тяжелый.

Для доказательства нижней оценки предположим, от противного, что у нас есть протокол, который делает меньше $\binom{n}{2}$ сравнений. Заметим, что сейчас у нас модель неадаптивная, так что протокол – это по существу просто список всех вопросов. Раз вопросов в нем меньше $\binom{n}{2}$, значит какие-то два объекта между собой не сравниваются. Рассмотрим два таких входа, при котором эти два объекта тяжелее всех остальных и в первом случае, первый объект тяжелее, а во втором – второй (а все остальные объекты сравниваются друг с другом одинаково). Тогда наш протокол на этих входах получит одни и те же ответы, а значит выдаст один и тот же результат. Поскольку самые тяжелые объекты в этих двух входах разные, наш протокол на одном из них ошибается, а значит мы пришли к противоречию.

□

Булевы функции. Третий пример – это разрешающие деревья для булевых функций. В этой модели требуется вычислить булеву функцию $f\colon \{0,1\}^n \to \{0,1\}$, то есть на вход подается $\vec{x} = (x_1, \ldots, x_n) \in \{0,1\}^n$ и при этом за один ход разрешается спрашивать значение одной переменной из $x_1, \ldots, x_n$. Формально в рамках нашей общей модели это означает, что в промежуточных вершинах разрешающего дерева

могут быть написаны только подмножества множества $\{0,1\}^n$, состоящие из всех векторов, в которых одна фиксированная координата одинакова. Но на самом деле удобно в случае булевых функцию в промежуточных вершинах дерева просто писать переменную, которую мы спрашиваем, и в пути, соответствующему вычислению, переходить по ребру помеченному нулем, если эта переменная равна нулю, и по ребру помеченному единицей, если переменная равна единице.

Сложность деревьев разрешения булевой функции f называют минимальную сложность (то есть, глубину) дерева, вычисляющего f. Эту величину обычно обозначают через $D(f)$.

Нетрудно заметить, что для всякой $f\colon \{0,1\}^n \to \{0,1\}$ верно $D(f) \leqslant n$. Действительно, достаточно просто спросить последовательно все переменные. После этого нам известен вход функции, и мы можем выдать ответ (как выглядит дерево этого протокола?). Оказывается, что для большинства функций $D(f) = n$, но мы не будем останавливаться на этом подробно.

13.5 Рассуждение с противником

В последних двух леммах рассуждения в доказательствах нижних оценок во многом похожи. И там, и там мы предполагали, что протокол есть, рассматривали его и подбирали вход таким образом, чтобы протокол ошибался. Таким образом, мы пользовались тем, что протокол должен работать для всех входов, и по существу, рассматривали "худший случай" для протокола.

У этого рассуждения есть другая (игровая) форма, которая оказывается более удобной для доказательства нижних оценок: *рассуждение с противником*. Представим себе, что вход для алгоритма выбирается не произвольно, а есть некое лицо (противник), которое его выбирает. При этом противник может выбирать вход не заранее, а по ходу работы протокола, вычисляющего функцию. То есть, изначально противник вход не фиксирует, а по мере поступления запросов от протокола дает на них ответы так, чтобы ответы были с каким-то входом согласованы, и при этом противник стремиться заставить протокол задать как можно больше вопросов. Таким образом, по существу противник старается сделать так, чтобы случился "худший случай". Это довольно просто понять на примере самой первой задачи об угадывании числа: Алиса вместо того, чтобы загадывать число заранее, может выбирать его по ходу дела, так чтобы Боб из ответов на вопросы получал как можно меньше информации.

Чем хороша такая форма рассуждения? Для этого полезно подумать о том, что нужно сделать для доказательства верхних и нижних оценок. Чтобы доказать верхнюю оценку нужно *построить* протокол. А чтобы доказать нижнюю оценку нужно доказать, что протокола *нет*. Если в первом случае нужно сделать что-то вполне конструктивное, то во втором не сразу ясно, что вообще делать. Рассуждение с противником позволяет свести доказательство нижней оценки также к чему-то конструктивному: вместо того, чтобы доказывать, что протокола *нет* (то есть, нет стратегии для вычисляющего игрока), мы можем *строить* стратегию для противника,

которая будет работать против всех протоколов.

По сути, мы рассматриваем нашу модель как игру. В игре есть два игрока: протокол и противник. Протокол задает вопросы, а противник дает на них ответы. Цель протокола – найти значение функции за не более чем k вопросов, для какого-то фиксированного k, а цель противника – помешать протоколу это сделать. Теория игр учит, что один из игроков при правильной игре может гарантировать себе победу, как бы не играл другой. Наше использование этой игровой интерпретации такое. Мы хотим доказать, что протокол не может гарантировать себе победу. Для этого мы показываем, что это может сделать противник.

Чтобы лучше понять происходящее полезно вспомнить уже рассмотренные задачи и понять, как получить в них нижние оценки с помощью рассуждений с противником. В задаче об угадывании числа противником является Алиса. Рассмотрим для нее такую стратегию. На каждом шаге Алиса помнит множество тех чисел, которые согласованы со всеми данными ранее ответами. Изначально это множество есть просто множество всех чисел от 1 до N. При каждом вопросе множество Алисы разбивается на два подмножества: те числа, для которых ответ на новый вопрос 'да', и те числа, для которых ответ на новый вопрос 'нет'. Алиса выбирает большее из двух подмножеств и дает ответ, согласованный с этим подмножеством. Таким образом, множество Алисы за один шаг уменьшается не более чем в два раза. И пока множество не станет одноэлементным протокол не может выдать ответ. Отсюда получается та же самая нижняя оценка $\lceil \log_2 N \rceil$.

В случае задачи о нахождении самого тяжелого объекта из n объектов в адаптивной модели противнику даже не нужна никакая специальная стратегия ответов на вопросы. Он просто выбирает изначально какой-то вход, дает ответы в соответствии с ним, ждет пока протокол (задающий не более $n-2$ вопросов) выдаст какой-то ответ, а затем исправляет свой вход, добавив большой вес ко всем объектам в одной из компонент связности. При этом все ответы согласованы с новым входом, а ответ протокола становится неправильным.

В случае задачи о нахождении самого тяжелого объекта из n объектов в неадаптивной модели протокол должен сразу сообщить противнику список вопросов. В этом случае стратегия противника уже по существу была нами описана. Нужно просто выбрать два объекта, которые протокол не сравнивает, самыми тяжелыми, а после выдачи протоколом ответа, выбрать какой из них сделать тяжелее.

Связность графа. Рассмотрим более содержательный пример рассуждения с противником.

А именно, рассмотрим следующую задачу. Дан неориентированный граф G на вершинах $\{1, \ldots, n\}$. За один ход разрешается спрашивать наличие или отсутствие конкретного ребра. Нужно проверить, является ли граф связным, то есть выдать 1, если является, и 0 иначе. Заметим, что эта задача – частный случай модели разрешающих деревьев для булевых функций. Действительно, можно считать, что рассматривается функция от множества переменных $\{x_{ij} \mid 1 \leqslant i < j \leqslant n\}$, где $x_{ij} = 1$ тогда и только тогда, когда между вершинами i и j есть ребро. Обозначим

функцию через $CONN$. Всего переменных у функции $\binom{n}{2} = n(n-1)/2$, а значит, как мы упоминали выше, $D(CONN) \leqslant \binom{n}{2}$.

Для начала докажем, что сложность деревьев разрешения для этой функции квадратична. Позже мы усилим эту оценку.

Лемма 13.6. *Пусть n – четно. Тогда $D(CONN) \geqslant n^2/4$.*

Доказательство. Опишем стратегию противника вынуждающую протокол задать не меньше $n^2/4$ запросов.

Поделим вершины графа на две равные доли A и B, по $n/2$ вершин в каждой. Противник отвечает на все вопросы о ребрах между долями отрицательно, а на остальные вопросы – положительно. Тогда пока протокол не задаст вопросы про все ребра между A и B он не может сказать, связен ли граф: если между A и B ребер нет, то граф не связен, но если хотя бы одно ребро присутствует, то он будет связен. Всего ребер между A и B как раз $n^2/4$. □

Таким образом, мы доказали верхнюю и нижнюю квадратичную оценку сложности деревьев для разрешения проверки графа на связность. Оказывается мы можем установить точное значение сложности этой задачи.

Лемма 13.7. $D(CONN) \geqslant \binom{n}{2}$.

Доказательство. Опять же, опишем стратегию противника. Для этого в каждый момент времени будем рассматривать два графа MAX и MIN на том же самом множестве вершин $\{1, \ldots, n\}$. В MIN будут только те ребра, про которые противник сказал “да”, а в MAX – те, про которые противник не сказал “нет”. Таким образом, MIN – это минимальный возможный граф, согласованный с уже данными ответами, а MAX – максимальный. Стратегия противника такая. Если его спрашивают про ребро e, то он отвечает “нет”, в том случае если MAX после этого остается связным, а иначе отвечает “да”.

Нам нужно доказать, что при такой стратегии противника протоколу придется задать вопросы про все ребра. Заметим, что по определению стратегии противника MAX всегда связен. Наш план – доказать, что если $MIN \neq MAX$, то MIN не связен. Это означает, что мы не знаем значение функции: с текущими ответами согласован как некоторый связный граф, так и некоторый не связный.

Заметим, что если в MAX есть цикл, то ни одно его ребро не принадлежит MIN. Действительно, пусть это не так. Рассмотрим ребро цикла, которое попало в MIN первым. Это означает, что противник ответил на вопрос об этом ребре положительно, то есть граф MAX при удалении этого ребра переставал быть связным. Но этого не может быть, потому что в любом пути, проходящем через это ребро, его можно заменить обходом по циклу. Противоречие.

В частности, из этого получается, что в MIN нет циклов: иначе такой цикл лежал бы и в MAX и все его ребра были бы при этом в MIN.

Теперь, если бы MIN был связен, то он был бы остовным деревом для MAX. При этом $MAX \neq MIN$, то есть в MAX есть ребро, которого нет в MIN. Тогда

это ребро вместе с графом MIN содержит цикл, а значит мы нашли цикл в MAX, часть ребер которого (все, кроме одного) лежат в MIN. Противоречие. Получается, что MIN не связен. □

Заметим, что в принципе рассуждение с противником не является обязательным: всякое рассуждение с противником можно перевести на язык "худшего случая". Однако это сделает, например, последнее доказательство заметно труднее. Рассуждение с противником – это удобный способ излагать и придумывать доказательства нижних оценок.