Лекция 12

Комбинаторные игры

Игры, в которые играют люди, многочисленны и разнообразны. Мы здесь будем обсуждать только такие игры, в которых игра делится на ходы, игроки ходят по очереди и знают всю информацию о состоянии игры, правила точно указывают возможности игроков и правило определения выигрыша. К играм такого типа относятся крестики-нолики, шашки, шахматы, го и многие другие. А карточные игры, такие как преферанс и бридж, не относятся к этому классу игр: в них игрокам известна не вся информация о состоянии игры (раскладе карт).

Интересовать нас будет вопрос «кто выигрывает в игре?» и те игры, про которые есть надежда получить ответ на этот вопрос (поэтому го и шахматы мы разбирать не будем). Даже точная формулировка такого вопроса неочевидна.

Мы дадим в этой главе все необходимые формулировки и приведём примеры игр, для которых ответ на такой вопрос удаётся получить. В лекции **??** игры будут применены при анализе трудности алгоритмов (так называемый «метод противника»).

12.1 Позиции

Начнём с примеров.

Пример 12.1 (игра в монетницу)**.** Игроки по очереди кладут монеты в монетницу, которая вмещает самое большее 20 монет. Игроки ходят по очереди. Мы их будем называть в порядке вступления в игру: первым ходит Первый игрок, вторым — Второй. На каждом ходе можно положить 2 или 3 монеты. Тот, кто не может сделать ход (потому что в монетнице уже 20 монет), — проиграл.

Как видим, правила игры определяют точно, какие ходы в игре возможны, и кто является победителем, когда игра закончилась. А она рано или поздно закончится: ведь на каждом ходе в монетнице увеличивается количество монет.

Игра эта довольно простая для анализа. Игрок, который ходит вторым, может обеспечить себе выигрыш, если будет действовать по следующему простому правилу: на каждом ходе он должен делать количество монет кратным 5: если Первый

положил 2 монеты, то Второй по этоум правилу кладёт 3 монеты и наоборот.

После каждой пары ходов количество монет увеличивается на 5. Значит, после 8 ходов оно станет равным 20, а очередь хода будет за Первым. Поскольку больше монет уже не положить, Первый проигрывает.

Задача 12.2. Проанализируйте вариант игры с монетницей, в котором ирокам разрешено класть 3 или 4 монеты.

Пример 12.3 (Игра «штриховка»)**.** Игровое поле: полоска бумаги, расчерченная на квадратики. Игроки ходят по очереди. Как и раньше, мы их будем называть Первым и Вторым в порядке вступления в игру. На каждом ходе игрок может заштриховать какую-то часть ещё не заштрихованных квадратиков, причём хотя бы один квадратик нужно заштриховать. Проигрывает тот игрок, который не может сделать ход (заштриховать хотя бы один квадратик).

Если квадратик всего один, то анализ игры вполне очевиден. Первый игрок делает единственно возможный ход, а Второй уже не может сделать ход. Игру всегда выигрывает Первый.

Но если квадратиков больше, анализ игры немногим сложнее. Если Первый заштрихует все квадратики, то Второй уже не сможет сделать ход. Итак, всегда выигрывает Первый.[1]

Выделим то общее, что есть в рассмотренных примерах. Два игрока ходят по очереди. Каждый ход изменяет *позицию* игры. Позиция игры определяет все возможности дальнейшего течения игры и её результата. Поэтому в позицию нужно включать и указание того игрока, чей ход в этой позиции. В первом примере позициями были состояния монетницы. Поэтому в ней всего $2 \cdot 21 = 42$ позиции.

Во втором примере, если исходных квадратиков было n, то количество равно $2(n+1)$, поскольку для игры важно лишь количество незаштрихованных к данному ходу квадратиков.

Обратите внимание, что мы считаем позициями игры всё, что соответствует описанию игры. Некоторые позиции могут оказаться недостижимыми из начальной. Скажем, в игре в монетницу 1 монета в монетнице невозможна.

В общем случае в определении игры нужно указать множество позиций. Мы будем ограничиваться только такими играми, в которых это множество конечно. Одна из позиций объявляется *начальной*: эта та позиция, в которой игра начинается.

Некоторые позиции в игре являются *заключительными*: в этих позициях игра заканчивается. Для заключительных позиций определяется *результат игры*: какой из игроков выиграл, для некоторых игр — сколько именно выиграл, в некоторых играх возможны ничейные исходы. Для единообразия мы будем считать, что результат игры всегда является числом. Для игр на выигрыш (как в рассмотренных примерах), будем считать, что выигрыш одного игрока обозначается числом $+1$, а второго — числом -1. Ничьей, если она возможна, будем сопоставлять результат 0.

[1] Сделаем уточнение. Как уже не раз бывало, полезно рассмотреть вырожденный случай игры — с 0 квадратиков. Тогда уже Первый не может сделать ход и проигрывает.

Замечание 12.1. Мы здесь рассматриваем только *антагонистические игры*. В таких играх выигрыш одного игрока противоположен выигрышу другого.

Естественным образом возникают и неантагонистические игры. В таких играх заключительная позиция определяет результат для каждого из игроков, и эти результаты не обязательно противоположны.

Замечание 12.2. Иногда понять, что является позицией в игре, не так уж легко. Скажем, в шахматах помимо расположения фигур и очерёдности хода в позиции нужно указывать, какие ходы в ней возможны (рокировка не всегда возможна при одном и том же расположении фигур), встречалась ли раньше эта позиция (троекратное повторение позиции — это ничья), и сколько ходов было сделано без взятия фигур и ходов пешек (см. правило 50 ходов в шахматном кодексе).

Для каждой позиции правила игры определяют *возможные ходы* игроков. Ход состоит в переходе к новой позиции. Поэтому ходы игрока можно задать ориентированным графом на множестве позиций: каждое ребро такого графа указывает на возможное по правилам игры изменение позиции при ходе данного игрока. (Напомним, что мы договорились включать в позицию очерёдность хода).

В рассмотренных выше примерах (и во многих других играх) возможные ходы были одинаковы для каждого из игроков в каждой позиции, если не учитывать очерёдность хода. Такие игры называются *беспристрастными*. Крестики-нолики, шашки, шахматы и го являются примерами пристрастных игр.

Для беспристрастных игр удобно считать позиции, не учитывая очерёдность хода. Далее мы так и будем делать при анализе конкретных игр, всякий раз уточняя это отступление от общего определения.

Бывают игры, в которых результат выражается числом.

Пример 12.4 (игра в монетницу «на интерес»)**.** Рассмотрим ещё один вариант игры в монетницу. Теперь игрок на каждом ходе обязательно использует 3 монеты. Если он кладёт в монетницу 2 монеты, то третью отдаёт противнику.

Игра заканчивается, когда в монетнице набралось 20 монет. Сделавший последний ход забирает все монеты из монетницы.

Результатом такой игры естественно считать прибавление монет у одного игрока (оно в точности равно убавлению монет у другого). Для определённости считаем прибавление монет у того, кто ходит первым.

Как мы видели, есть правило игры для второго, при котором он делает последний ход и забирает всю монетницу. Однако результат в новой игре пока не определён: он зависит от ходов первого игрока. Легко видеть, что если второй игрок придерживается указанного выше правила, то первый игрок может обеспечить себе 4 монеты, делая каждый раз ход в 3 монеты. Общий результат в такой игре: -8 (первый получает 4 монеты и теряет $4 \cdot 3 = 12$).

Является ли этот результат оптимальным? Мы ещё вернёмся к этому вопросу. Попробуйте сейчас ответить на него самостоятельно.

Задача 12.5. Опишите множество позиций для игры в монетницу «на интерес». Является ли эта игра беспристрастной?

Завершим раздел формальным определением игры, которому будут удовлетворять (после некоторых уточнений) все интересующие нас игры.

Определение 12.1. (Конечная) *комбинаторная игра* двух игроков, назовём их Макс (M) и Мин (m), задаётся

- (Конечным) множеством позиций S, разбитым на два непересекающихся подмножества S_M и S_m (в позициях из первого множества ходит Макс, в позициях второго — Мин);
- множеством ходов игроков в каждой позиции, то есть ориентированным графом (S, E);
- начальной позицией $s_0 \in S$;
- множеством заключительных позиций $S_f \subseteq S$;
- функцией выигрыша $v\colon S_f \to \mathbb{R}$, которая определяет результат игры в заключительных позициях.

Замечание 12.3. Данное определение не запрещает одному из игроков сделать несколько ходов подряд. Впрочем, возможность делать несколько ходов подряд не очень существенна. Всегда можно объединить несколько ходов в один и перейти к играм, в которых игроки ходят строго по очереди).

Задача 12.6. Составьте граф игры в монетницу.

12.2 Стратегии

Теперь вернёмся к вопросу «кто выигрывает в игре?» и точно сформулируем его смысл. Весь данный раздел состоит по существу из одного длинного определения, разбитого на части для удобства.

Из определения игры известен результат игры в заключительных позициях. Следующим шагом определим результат игры в *партии*.

Определение 12.2. Партия игры: это такая последовательность позиций игры $s_0, \ldots, s_n$, что

- s_0 — начальная позиция;
- s_n — заключительная позиция игры, а все предыдущие не являются заключительными;
- для всех $0 \leqslant i < n$ в графе возможных ходов (S, E) есть ребро (e_i, e_{i+1}).

  Результатом партии является функция выигрыша $v(s_n)$ от заключительной позиции в партии.

Другими словами, партия — это ориентированный маршрут из начальной позиции в одну из заключительных, не проходящий через заключительные позиции.

Аналогично можно определить и бесконечную партию: это бесконечный маршрут в графе игры, не проходящий через заключительную позицию. Результат такой партии не определён.[2]

Бесконечные партии возможны и в играх с конечным числом позиций. Скажем, если в беспристрастной игре есть ориентированный цикл, достижимый из начальной позиции, игроки могут перейти в позицию на этом цикле и двигаться по нему бесконечно долго.

Мы далее предполагаем всюду, что граф игры является не только конечным, но и ациклическим, т.е. в нём отсутствуют ориентированные циклы. На таком графе бесконечные партии невозможны.

Партий игры много и результаты в них, как правило, разные. Чтобы анализировать игры, нужно ввести следующее важное понятие.

Определение 12.3. *Стратегия* игрока — это функция α, которая ставит в соответствие позиции s_k один из возможных ходов игрока в позиции s_k, т.е. следующую позицию игры.

Говорят, что в партии $s_0, \ldots, s_n$ игрок *придерживается стратегии* α, если для тех i, в которых его очередь хода, выполняется равенство $s_{i+1} = \alpha(s_i)$.

В примере 12.1 мы называли стратегию правилом игры.

Пример 12.7. Какую именно функцию задаёт правило, сформулированное в примере 12.1? Для позиций, номера которых имеют остаток 2 или 3 при делении на 5, эта функция указывает на ход в позицию с номером, кратным 5. Для позиции 11 такая функция не определена.

Обычно требуют, чтобы стратегия была всюду определённой функцией. На самом деле, достаточно более слабого свойства: стратегия, пусть и частично определённая, должна быть *всегда применимой*. Это означает, что как бы ни играл противник, в любой возникающей позиции стратегия определена. Именно так обстоит дело с игрой в монетницу.

Замечание 12.4. Есть более общее определение стратегии, в котором ход игрока определяется не только позицией, а всей предысторией партии. Определённые нами стратегии называются в таком случае *позиционными* и являются лишь частным случаем.

Для игр, которые мы рассматриваем, разница между общими стратегиями и позиционными несущественна.

Сформулируем теперь предположение о предпочтениях игроков: Макс стремится максимизировать результат игры, а Мин — минимизировать. Поэтому качество стратегий для игроков определяется по-разному.

[2] Есть много способов доопределить результат для бесконечных партий. Здесь такие случаи не рассматриваются.

Определение 12.4. Стратегия α *гарантирует игроку Макс* выигрыш W, если в любой партии, в которой Макс придерживается стратегии α, результат игры **не меньше** W.

Аналогично определяем *гарантированный выигрыш для Мина*: теперь в любой партии, в которой Мин придерживается стратегии α, результат игры **не больше** W.

В примере игры в монетницу стратегия второго игрока (пусть он будет Мином) гарантировала ему выигрыш -1, т.е. просто выигрыш по сделанному нами соглашению.

После такого длинного введения мы наконец дадим ответ на вопрос, кто выигрывает в игре. Если функция выигрыша игры принимает значения ± 1 («выиграл»/«проиграл»), то Макс выигрывает, если у него есть стратегия, гарантирующая $+1$; Мин выигрывает, если у него есть стратегия, гарантирующая -1. В этом важном частном случае говорят также о наличии *выигрышной стратегии* у одного из игроков.

Важно отметить, что в определении стратегии, гарантирующей выигрыш W, речь идёт обо всех партиях, в которых игрок придерживается данной стратегии.

Для игр с произвольной функцией выигрыша определим более общее понятие цены игры.

Определение 12.5. Число C называется *ценой игры*, если у Макса и у Мина есть стратегии, гарантирующие выигрыш C.

Если у игры есть цена, то играть в неё становится необязательно. Как бы ни играл Макс, больше C он не получит, если Мин придерживается той стратегии, которая гарантирует C. Аналогично Мин не сможет получить меньше C, если Макс придерживается той стратегии, которая гарантирует C.

Замечание 12.5. Тем не менее, в реальной жизни люди иногда играют в игры с известной ценой. Один из самых наглядных примеров: крестики-нолики. Маленькие дети во всем мире играют в эту игру, хотя хорошо известно, что её цена равна 0 (ничья).

Нас будет интересовать как раз вопрос о существовании цены игры и описании соответствующих стратегий.

12.3 Разбор с конца

Докажем, что для тех игр, которые мы рассматриваем, цена игры существует всегда. В доказательстве возникает очень полезный приём построения стратегий, который мы назовём разбором с конца. По сути это ещё один вариант рассуждения по индукции. В дальнейшем мы значительно обобщим его, см. раздел 10.7. Пока докажем лемму об ациклических графах.

Лемма 12.1. *Пусть конечный граф (S, E) ациклический, а $X \subset S$ — собственное подмножество его вершин. Тогда в графе найдётся такая вершина s, что все рёбра, начинающиеся в s, ведут в множество X.*

Доказательство. По определению собственного подмножества, среди вершин графа (S, E) найдётся не принадлежащая X вершина s_0.

Построим маршрут, начинающийся в s_0, последовательно продлевая его. Пусть уже построен маршрут $s_0, \ldots, s_k$. Для вершины s_k есть две возможности: либо есть ребро (s_k, s_{k+1}), которое начинается в s_k и заканчивается вне X, либо такого ребра нет. Во втором случае построение маршрута закончено. Заметим, что последняя вершина маршрута удовлетворяет желаемому свойству: все рёбра из неё ведут в множество X.

Граф конечный, поэтому бесконечно продолжать маршрут не получится. Значит, рано или поздно маршрут закончится в вершине, удовлетворяющей желаемому свойству. □

Теорема 12.2. *Для антагонистической игры на ациклическом графе существует цена при любой функции результатов.*

Доказательство. Мы будем рассматривать не только саму игру Γ, для которой доказываем существование цены $C(\Gamma)$, но и все игры, которые получаются из неё изменением начальной позиции. Для позиции s игра Γ_s отличается от игры Γ заменой начальной позиции s_0 на s.

Идея доказательства состоит в том, что если известны цены игры для всех позиций, в которые можно пойти из данной, то цена игры определена и для данной позиции (вспомним, что при известной цене игру можно не разыгрывать, а сразу объявить результат).

Если s_f — заключительная позиция, то игра Γ_{s_f} тривиальна. Ни один из игроков не может сделать ход, а результат $v(s_f)$ определён. Ясно, что это и есть цена игры:

$$C(\Gamma_{s_f}) = v(s_f).$$

Далее можно определеить цену для позиций, из которых все ходы ведут в заключительные и т.д. Мы опишем это рассуждение подробно.

Построим функцию $C(s)$, определяя её значения итеративно. На первой итерации функция определяется для заключительных позиций $C(s) = v(s)$, где $s \in S_f = X_1$.

После i-й итерации функция определена для позиций из множества X_i. На $(i+1)$-й итерации значение функции определяется для тех позиций s, из которых все рёбра ведут в X_i.

Если в позиции s ход Макса (формально $s \in S_M$), то

$$C(s) = \max_{(s,s')\in E} C(s').$$

Если в позиции s ход Мина (формально $s \in S_m$), то

$$C(s) = \min_{(s,s')\in E} C(s').$$

Доказанная выше лемма 12.1 гарантирует, что на каждой итерации область определения функции увеличивается. Поскольку граф конечный, рано или поздно функция окажется определена для всех позиций.

Осталось доказать, что функция $C(s)$ совпадает с ценой игры $C(\Gamma_s)$. Докажем утверждение индукцией по номеру итерации, на которой значение функции становится определённым в данной позиции.

Мы уже проверили утверждение для залючительных позиций. Это база индукции.

Теперь индуктивный переход. Предположим, что для позиций из множества X_i (те позиции, в которых значение функции C определено после i итераций) утверждение справедливо: $C(s) = C(\Gamma_s)$.

Рассмотрим позицию $s \in X_{i+1}$. Пусть это позиция Макса. Тогда стратегия, гарантирующая ему выигрыш $C(s)$, состоит в том, чтобы из s пойти в ту вершину s', на которой достигается максимум $\max_{(s,s')\in E} C(s')$. В игре $\Gamma_{s'}$ Макс должен придерживаться стратегии, гарантирующей выигрыш $C(s')$.

Для Мина есть стратегия, гарантирующая $C(s')$ во всех позициях, куда может сделать ход Макс. Эта же стратегия гарантирует ему выигрыш $C(s)$ в игре Γ_s.

Аналогично разбирается случай позиции Мина. Игроки меняются местами: стратегия Мина состоит в том, чтобы сделать ход в позицию s', на которой достигается минимум $\min_{(s,s')\in E} C(s')$ и далее придерживаться стратегии, гарантирующей выигрыш $C(s')$.

У Макса есть стратегия, гарантирующая выигрыш $C(s')$ в каждой вершине s', куда может сделать ход Мин. Эта же стратегия гарантирует ему выигрыш $C(s)$ в игре Γ_s. □

Замечание 12.6. При неформальном обсуждении игр очень часто используются выражения «лучший ход в данной позиции» или «выигрышный ход в данной позиции». Разбор с конца позволяет придать этому выражению точный смысл: «лучшим» («выигрышным» при игре на выигрыш) ходом будем называть тот ход, который ведёт в позицию с той же ценой игры, что и данная.

Заметим, что при этом «выигрышный» ход может вести в позицию, в которой данный игрок проигрывает! (Если он проигрывает во всех позициях, куда может сделать ход.) Поэтому такое словоупотребление неудачно.

Пример 12.8 (Пристрастная игра в монетницу)**.** Рассмотрим вариант игры в монетницу, в котором Максу разрешается класть 2 или 3 монеты, а Мину — 1 или 4 монеты. Первый ход за Максом. У кого есть выигрышная стратегия?

Всего есть 42 позиции (число монет + очередность хода). Будем указывать не количество монет в монетнице, а количество монет, которые в монетницу ещё можно положить. Так позиция (Макс,0) означает, что ход за Максом и в монетницу уже нельзя положить ни одной монеты.

Найдём цену игры для каждой позиции. Из функции результата мы уже знаем цену игры в заключительных позициях. Запишем их в таблицу (столбцы отвечают

количеству монет, строки — тому игроку, который делает ход):

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Макс	-1	-1																			
Мин	$+1$																				

Из позиции (Мин,1) Мин может сделать ровно один ход в позицию (Макс,0). Поэтому цена игры в позиции (Мин,1) равна -1 (Мин выигрывает). То же самое выполняется для позиции (Мин,2).

Из позиции (Макс, 2) Макс также может сделать единственный ход и цена этой позиции $+1$. Получаем уточнённую таблицу:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Макс	-1	-1	$+1$																		
Мин	$+1$	-1	-1																		

Для позиции (Макс, 3) есть два хода. Один ведёт в позицию (Мин,1) с ценой -1, а второй — в позицию (Мин,0) с ценой $+1$. Для определения цены игры нужно взять максимум (ход за Максом). Поэтому цена позиции (Макс, 3) равна $+1$. Продолжая этот процесс, получим следующую таблицу:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Макс	-1	-1	$+1$	$+1$	-1	$+1$	$+1$	-1	$+1$	$+1$	$+1$	$+1$	$+1$	$+1$	$+1$	$+1$	$+1$	$+1$	$+1$	$+1$	$+1$
Мин	$+1$	-1	-1	$+1$	-1	-1	$+1$	$+1$	-1	$+1$	$+1$	-1	$+1$	$+1$	$+1$	$+1$	$+1$	$+1$	$+1$	$+1$	$+1$

Из этой таблицы мы видим, что у Макса есть выигрышная стратегия для всех позиций, в которых больше 11 монет в монетнице. Поэтому есть выигрышная стратегия, которая в позиции (Макс, 20) выбирает ход 2, равно как и есть выигрышная стратегия, которая в этой позиции выбирает ход 3. С другой стороны, в позиции (Макс, 14) ход 3 приводит к позиции, в которой выигрывает Мин. Поэтому любая выигрышная стратегия должна в позиции (Макс, 14) выбирать ход 2.

Пример 12.9 (продолжение примера 12.1)**.** А каковы цены позиций в первоначальном варианте игры в монетницу? Эта игра беспристрастная, то есть множество ходов для каждого игрока одинаково. В заключительных позициях цена игры противоположна (проигрывает тот, кто не может сделать ход). Поэтому и в каждой позиции цена игры для игроков противоположна (докажите это по индукции!).

Поэтому таблицу позиций можно сократить, и указывать лишь одну строчку в ней. Однако принят немного другой способ описания цены для таких игр: указывается тот игрок, который выигрывает в данной сокращённой позиции (без очерёдности хода), Мы будем отмечать позицию буквой N, если выгрывает тот игрок, который делает ход в данной позиции, и буквой P, если выгрывает игрок, который сделал ход в данную позицию.[3]

[3]Такое обозначение выглядит не вполне понятным для русскоязычного читателя, оно используется в англоязычных книгах по играм. “N” означает “next” (игрок, делающий следующий ход), а “P” означает “previous” (игрок, который сделал предыдущий ход).

Индуктивное правило определения цены игры в данном случае также упрощается. Поскольку игроки ходят по очереди, то Первый игрок выигрывает, если у него есть ход в позицию с ценой P (после следанного хода Первый будет ходить вторым!). В противном случае выигрывает Второй игрок.

Разбором с конца нетрудно составить таблицу цены игры в монетницу. В принятых выше обозначениях получаем такую таблицу

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
P	P	N	N	N	P	P	N	N	N	P	P	N	N	N	P	P	N	N	N	P

Из этой таблицы видно, что в начальной позиции 20 выигрывает игрок, который делает ход вторым. А в позиции 19 выигрывает игрок, который делает ход первым (выигрышная стратегия сопоставляет этой позиции ход 3, так как для позиции 16 цена P).

Пример 12.10 (продолжение примера 12.4)**.** Вернёмся к игре в монетницу «на интерес». Мы уже видели, что у второго (Мина) есть стратегия, гарантирующая ему -8 (то есть выигрыш 8 монет).

Даёт ли эта стратегия цену игры? Для ответа на этот вопрос нужно понять, есть ли у Макса, который ходит первым, стратегия, гарантирующая -8.

Такая стратегия есть. Для её описания удобно использовать разметку позиций, сделанную в предыдущем примере. В позициях, которые кратны 5, Макс должен делать ход 3. В позициях, помеченных N, он должен придерживаться стратегии, гарантирующей выигрыш в обычной игре в монетницу. Стратегия Макса в остальных позициях несущественна.

Давайте оценим, какой выигрыш гарантирует данная стратегия для Макса. Если Макс оказался в N-позиции, то он заберёт в итоге монетницу. Его выигрыш при этом будет неотрицательным (дополнительно он отдаст не больше 4 монет, а получит не меньше 4: хотя бы два хода Мин сделает).

Осталлось рассмотреть только тот случай, когда Макс делает ход только в P-позициях. Глядя на таблицу, легко проверить, что партия тогда однозначно определена стратегией Макса: Макс кладёт 3 монеты, Мин кладёт 2 (и одну отдаёт Максу по правилам игры «на интерес») и т.д.

В этой партии Макс кладёт 12 монет в монетницу, но получает от Мина 4 монеты. Значит, его выигрыш равен -8.

Итак, если Макс придерживается описанной выше стратегии, то его выигрыш либо неотрицательный, либо равен -8. По определению это означает, что данная стратегия гарантирует ему выигрыш -8.

12.4 Симметричные стратегии

Разбор с конца требует анализа всех позиций игры. Для большинства игр это очень трудоёмкая, а иногда и непосильная задача. Скажем, теорема 12.2 гарантирует, что

в шахматах есть цена.[4] То есть, либо у белых есть стратегия, гарантирующая выигрыш, либо у чёрных есть стратегия, гарантирующая выигрыш, либо у обоих игроков есть стратегии, гарантирующие ничью. Однако перебрать все шахматные позиции нереально на существующих компьютерах. Поэтому до сих пор неизвестно, какой из трёх вариантов выполняется на самом деле.

Однако доказательство существования выгрышной стратегии иногда возможно и тогда, когда разбор всех позиций игры неосуществим из-за их гигантского количества.

Одним из важных приёмов в доказательствах существования стратегий являются соображения симметрии. Приведём несколько примеров.

Пример 12.11 (Баловство с ладьёй)**.** Игровое поле: шахматная доска $n \times n$. (Настоящая шахматная доска получается при $n = 8$, но в данную игру можно играть и при других n.) На доске есть ровно одна фигура: ладья. Вначале ладья стоит в правом верхнем углу (рис. 12.1 слева). Далее игроки делают по очереди ходы. На каждом ходу игрок может сдвинуть ладью либо по горизонтали влево, либо по вертикали вниз (хотя бы на одну клетку), см. пример на рис. 12.1 в центре). Проигрывает тот, кто не может сделать ход.

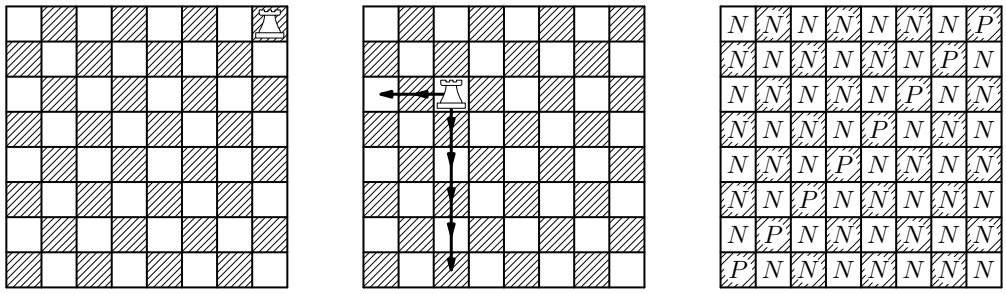

Рис. 12.1: Начальная позиция, возможные ходы и оценка позиций для баловства с ладьёй

Докажем, что позиции, в которых Первый проигрывает (P-позиции), — это диагональ из левого нижнего угла в правый верхний. Про левый нижний угол это очевидно: в этой позиции невозможно сделать ход. Стратегия Второго в остальных случаях — восстанавливать симметрию. Если Первый сделал ход, уводящий с диагонали, Второй на своём ходе может вернуть ладью на диагональ.

Если же исходное положение ладьи не на диагонали, то выигрывает Первый (N-позиция). Стратегия состоит в том, чтобы первым ходом перевести ладью на диагональ, а дальше следовать описанной выше симметричной стратегии Второго.

Пример 12.12 (монеты на стол)**.** В этой игре есть круглый стол и два игрока с неограниченным запасом одинаковых круглых монет. Игроки по очереди кладут

[4] Возможность применения к шахматам теоремы 12.2 неочевидна. Если вы знаете шахматные правила, попробуйте точно сформулировать, что считать шахматными позициями, если хотеть получить конечный ациклический граф позиций.

монеты на стол так, чтобы монета полностью помещалась на столе и не накрывала уже положенные монеты. Выигрывает тот, кто положил последнюю монету.[5]

Это пример беспристрастной игры и выигрывает в ней тот, кто ходит первым. Выигрышная стратегия первым ходом помещает монету в центр стола (рис. 12.2 слева). На каждом следующем ходу Первый кладёт монету центрально-симметрично последнему ходу противника (пример на рис. 12.2 в центре).

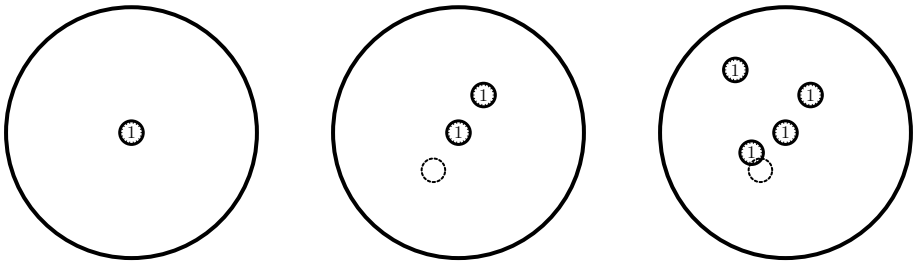

Рис. 12.2:

Обратите внимание, что эта стратегия не всюду определена: возможны позиции, в которых центрально-симметричный ход сделать невозможно, см. рис. 12.2 справа. Однако она всегда применима. После первого хода для свободной области доски выполняется такое свойство: если в какое-то место можно положить монету, то её можно положить в центрально-симметричное. Если Первый следует описанной выше стратегии, то это свойство сохраняется после каждого хода.

Симметричная стратегия Первого гарантирует, что он всегда может сделать ход и потому не может проиграть. Значит, проигрывает Второй.

Пример 12.13 (гекс)**.** Игра «гекс» происходит на доске 11×11, составленной из правильных 6-угольников, см. рис. 12.3. Играют два игрока: Синий и Красный. Левая и правая стороны доски принадлежат Синему, а верхняя и нижняя — Красному (см. рис.).

Игроки ходят по очереди, наичнает Красный.

На каждом ходе игрок ставит на одно из свободных 6-угольных полей фишку своего цвета. Игрок выигрывает, если на доске возникает путь цвета этого игрока, соединяющий его стороны (пример см. на рис. 12.4).

В гексе не бывает ничьей: если есть синий путь, соединяющий синие стороны как на рисунке, то заведомо нет красного пути, соединяющего красные стороны (докажите это утверждение!).

Поэтому из теоремы 12.2 заключаем, что у одного из игроков есть выигрышная стратегия. Более того, этот игрок— Красный.

Это следует из симметрии доски. При отражении относительно диагонали синие и красные стороны меняются местами.

[5] В этой игре множество позиций бесконечно, так как для положений монет есть бесконечно много вариантов. Однако любая партия в этой игре конечна. Докажите, что в таком случае теорема 12.2 также справедлива.

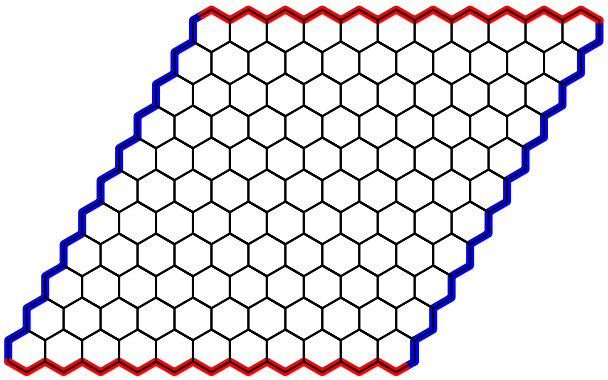

Рис. 12.3: Поле для игры в гекс

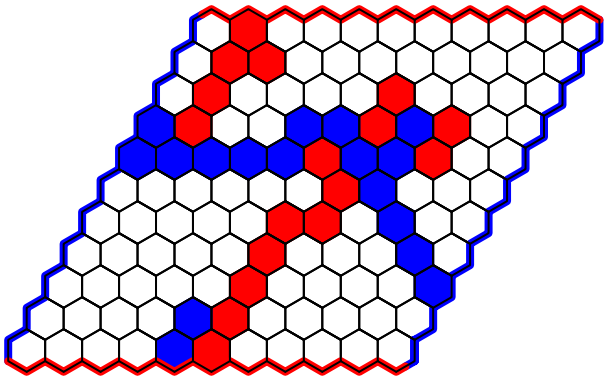

Рис. 12.4: Выигрыш Синего

Докажем от противного существование выигрышной стратегии для Красного. Предположим, что выигрышная стратегия α есть у Синего. Рассмотрим такую стратегию для Красного: сделать первый ход произвольно, а на каждом следующем ходе играть по стратегии α, гарантирующей выигрыш Синего с точностью до симметрии (то есть отражать позицию на доске, меняя красные фишки на синие и наоборот, выбирать ход по стратегии α за Синего и делать ход в симметричную позицию). Может оказаться так, что поле, на которое нужно сделать ход по данному правилу, уже занято. Тогда Красный делает ход на любое свободное поле.

В этом описании есть неточность. Позиция при ходе Красного содержит одинаковое количество красных и синих фишек, а при ходе Синего красных фишек на одну больше. Поэтому Красный мысленно удаляет с поля одну из своих фишек и только после этого выбирает ход, симметричный ходу по стратегии α как описано выше.

Почему это выигрышная стратегия? Предположим, что партия, в которой Красный придерживался симметричной α стратегии, закончилась проигрышем Красного. Последний ход Синего создал синий путь между синими сторонами. Но это

означает, что в симметричной позиции у Красного был ход, создающий красный путь между красными сторонами, что невозможно по предположению о том, что α — выигрышная стратегия для Синего.

Гекс интересен тем, что хотя цена игры известна, играть в неё достаточно интересно. Доказательство от противного не даёт никакого намёка на то, как играть Красному в реальной партии.

12.5 Ним

Игра «ним» — ещё один пример беспристрастной игры. Имеется два игрока и три кучки камней. За один ход игрок может взять из какой-то кучки любое количество камней (хотя бы один нужно взять). Проигрывает тот игрок, который не может сделать ход.

Из такого описания видно, что (сокращённая) позиция в игре ним — это три натуральных числа. Причём позиция $(0,0,0)$ по определению проигрышная для первого игрока и выигрышная для второго (мы, как и выше, нумеруем игроков в том порядке, в каком они вступают в игру). В принятых нами обозначениях это P-позиция. Про остальные позиции нужно выяснить, кто в них выигрывает (напомним, что более точно говорить «имеет выигрышную стратегию»).

Часть случаев мы уже фактически разобрали раньше.

Позиции $(n,0,0)$ — это по сути позиции игры в штриховку (пример 12.3). Число n означает количество незаштрихованных квадратиков. Поэтому при $n>0$ такие позиции выигрышны для Первого (N-позиции).

Позиции $(n,m,0)$ — это позиции игры «баловство с ладьёй» (пример 12.11). Числа (n,m) — это попросту координаты ладьи при стандартном понимании координат (начало — левый нижний угол, координаты возрастают слева направо и снизу вверх).

Поэтому при $n\ne m$ позиции $(n,m,0)$ являются N-позициями, а при $n=m$ являются P-позициями.

Отсюда также следует, что позиции (n,m,m) также выигрышны для Первого: своим первым ходом он забирает n камней из первой кучки и получается P-позиция $(0,m,m)$.[6]

Разбором с конца легко проверить, что позиция $(1,2,3)$ является P-позицией (выигрышна для Второго).

Задача 12.14. Проведите полностью разбор с конца для позиции $(1,2,3)$.

Но тогда позиции $(k,2,3)$, где $k\ne 1$; $(k,1,3)$, где $k\ne 2$; $(k,1,2)$, где $k\ne 3$, являются N-позициями.

Если $k>3$, то из каждой такой позиции Первый может перейти в позицию $(1,2,3)$ и далее следовать выигрышной стратегии Второго для этой позиции.

[6] Заметим, что порядок кучек неважен и оценка игры в позиции не изменяется при перестановке кучек.

Задача 12.15. Докажите это утверждение при остальных значениях k.

Разбор с конца можно продолжать и расширять список позиций, для которых известна оценка игры. Прежде чем читать далее, попробуйте сделать такой разбор ещё для нескольких типов позиций и угадать общий ответ. Это не так легко сделать — ответ весьма неожиданный!

Пусть мы анализируем позицию (x, y, z). Запишем числа x, y, z в двоичной системе, начиная с младших битов:

$$\begin{aligned} x =& x_0x_1x_2\ldots,\\ y =& y_0y_1y_2\ldots,\\ z =& z_0z_1z_2\ldots. \end{aligned}$$

Сложим биты в каждом столбце по модулю 2, получим двоичную запись

$$a_0a_1a_2\ldots$$

Оказывается, у Первого игрока есть выигрышная стратегия тогда и только тогда, когда последовательность a_i содержит хотя бы одну 1.

Пример 12.16. Оценим ним в позиции $(1, 2, 3)$. Двоичная запись даёт

$$\begin{aligned} 1 =& 10,\\ 2 =& 01,\\ 3 =& 11. \end{aligned}$$

Поразрядная сумма по модулю 2 этих последовательностей равна 00. Значит, это P-позиция, как и утверждалось выше.

Пример 12.17. Оценим ним в позиции $(5, 15, 26)$. Двоичная запись даёт

$$\begin{aligned} 5 =& 10100,\\ 15 =& 11110,\\ 30 =& 01011. \end{aligned}$$

Поразрядная сумма по модулю 2 этих последовательностей равна 00001. Сформулированное правило утверждает, что Первый выигрывает. Пока неясно, в чём состоит его стратегия.

На самом деле, если поверить в справедливость правила, то стратегия Первого на первом ходе ясна: нужно перейти в позицию, где среди поразрядных сумм нет единиц, — в такой позиции делающий первый ход проигрывает. Такой ход есть: нужно из 30 камней забрать 16. Получится позиция $(5, 15, 14)$.

Но почему эти позиции проигрышны для Первого? Опять-таки, если верить правилу, то на каждый ход Первого Второй должен ответить так, чтобы вернуться в позицию, в которой все порязрядные суммы чётные. Для каждой конкретной позиции это можно проверить перебором вариантов. Но, видимо, пора переходить к общему доказательству.

Общее доказательство происходит индукцией по числу камней во всех кучках. База индукции: 0 камней, поразрядная сумма нулевая.

Теперь предположим, что для всех позиций с количеством камней $< N$ корректность правила оценки позиции доказана. Рассмотрим позицию с N камнями. Учитывая правило оценки, удобно представлять эту позицию как таблицу из трёх строк и какого-то количества столбцов. Таблица заполнена нулями и единицами.

Ход состоит в том, что в одной из строк часть нулей заменяется на единицы и наоборот. Конечно, нужно позаботиться, чтобы число, представленное новым набором нулей и единиц было меньше числа, представленного исходным набором нулей и единиц. Для определённости мы считаем, как и в примерах выше, что старшинство разрядов идёт по столбцам слева направо.

Нужно доказать два утверждения:

(I) если в таблице есть столбец с нечётным числом единиц, то можно сделать ход, который приведёт в таблицу, каждый столбец которой содержит чётное число единиц;

(II) если в таблице все столбцы содержат чётное количество единиц, то любой допустимый ход приводит в таблицу, в которой есть столбец с нечётным числом единиц.

Доказательство (I). Возьмём самый правый столбец, в котором нечётное количество единиц (а значит, хотя бы одна единица есть). Выберем строку, в которой в выбранном столбце стоит 1. Ход делаем в этой строке. Заменим единицу в выбранном столбце на 0. Во всех предыдущих столбцах действуем по следующему правилу: если в столбце чётное количество единиц, то ничего не меняем, а если нечётное — инвертируем бит в данном столбце. После этих действий в каждом столбце будет чётное число единиц. Но нужно проверить, что такие изменения отвечают допустимому ходу. Для этого вспомним, что числа сравниваются в двоичной записи от старших разрядов к младшим. Мы изменили в каком-то разряде (столбце) 1 на 0, а далее все изменения происходили в разрядах, которые младше (левее). Поэтому полученная двоичная запись будет давать число, меньшее исходного.

Пример 12.18. Пусть дана таблица

0	1	1	0	1	1	1	0	1	число 374
1	0	1	1	0	0	0	1	0	число 141
1	0	1	0	1	1	0	1	0	число 181

Сделав ход по указанному правилу, получим таблицу

0	0	0	1	1	1	0	0	0	число 56
1	0	1	1	0	0	0	1	0	число 141
1	0	1	0	1	1	0	1	0	число 181

Задача 12.19. Придумайте таблицу, в которой ход по указанному правилу изменяет количество камней на 1.

Доказательство (II). Сейчас мы предполагаем, что в каждом столбце таблицы чётное количество единиц. Допустимый ход уменьшает количество камней. Поэтому в каком-то разряде (столбце) 1 заменилась на 0. После такого хода в этом столбце чётность числа единиц изменится.

12.6 Сумма игр и функция Шпрага–Гранди

Правило для игры в ним согласуется с играми, в которых одна или две кучки камней. Оказывается, можно утверждать и больше: это правило допускает обобщение на произвольное количество кучек. Более того, оно применимо в более общей ситуации.

Мы будем рассматривать беспристрастные игры, в которых выигрывает сделавший последний ход, а граф позиций ациклический (то есть невозможны бесконечные партии). Ним с произвольным количеством кучек камней является примером такой игры.

Оценка позиции игры принимает два значения N или P: выигрышная стратегия есть либо у Первого, либо у Второго. Но оказывается осмысленным приписать позициям не двоичное значение «N-P», а натуральное число. При этом значение 0 будет означать наличие выигрышной стратегии у Второго (P-позиция), а положительное значение — наличие выигрышной стратегии у Первого (N-позиция). Смысл этих чисел примерно такой: любая беспристрастная игра «похожа» на ним с одной кучкой камней; а число, приписанное позиции, — количество камней в этой виртуальной кучке.

Правило приписывания чисел позициям обобщает разбор случаев с конца. Позициям, из которых нельзя сделать хода, приписываем значение 0 (в них Первый проигрывает). Пусть теперь части позиций (не всем) уже приписаны числа. По лемме 12.1 должна найтись позиция x, из которой все ходы ведут в позиции, которым уже приписаны номера Пусть ходы ведут в позиции y_1, y_2, …, которым приписаны номера v_1, v_2, … Тогда позиции x припишем номер

$$v_x = \operatorname{mex}(v_1, v_2, \dots),$$

где значение функции mex равно наименьшему натуральному числу, которое не встречается среди её аргументов.

Пример 12.20. $\operatorname{mex}(0,2,3) = 1$; $\operatorname{mex}(0,1,2,10) = 3$, $\operatorname{mex}(2,3) = 0$.

Отсюда получаем, что если все ходы ведут в позиции с положительными номерами, то позиции x приписываем значение 0. Это как раз соответствует тому, что если из позиции все ходы ведут в N-позиции, то данная позиция является P-позицией.

Верно и обратное. Если из позиции x есть хотя бы один ход в позицию со значением 0, то $v_x > 0$. Другими словами, если есть ход в P-позицию, данная позиция является N-позицией.

Хотя порядок, в котором позиции получают значения, может быть различным, само значение позиции от этого порядка не зависит. Это можно доказать аналогично доказательству теоремы 12.2 (индукция по частичному порядку, определённому графом позиций, см. раздел 10.7).

Функция $s \mapsto v(s)$, которая сопоставляет позиции её значение, называется *функцией Шпрага–Гранди*.

Пример 12.21. Вычислим функцию Шпрага–Гранди для нима с одной кучкой. Позиции нумеруем количеством камней в кучке.

Тогда $v(0) = 0$ (это P-позиция), поэтому

$$\begin{aligned} v(1) &= \operatorname{mex}(0) = 1, \\ v(2) &= \operatorname{mex}(0,1) = 2, \\ v(3) &= \operatorname{mex}(0,1,2) = 3 \end{aligned}$$

и т.д. По индукции получаем $v(n) = n$.

Пример 12.22. Вычислим функцию Шпрага–Гранди игры в монетницу. Позиции нумеруем количеством монет в монетнице.

Тогда $v(0) = v(1) = 0$ (это заключительные позиции), далее получем

$$\begin{aligned} v(2) &= \operatorname{mex}(0,0) = 1, \\ v(3) &= \operatorname{mex}(0,0) = 1, \\ v(4) &= \operatorname{mex}(0,1) = 2, \\ v(5) &= \operatorname{mex}(1,1) = 0, \\ v(6) &= \operatorname{mex}(1,2) = 0, \\ v(7) &= \operatorname{mex}(0,2) = 1 \end{aligned}$$

и по индукции можно проверить, что далее функция Шпрага–Гранди периодична с периодом 5, т.е. $v(n+5) = v(n)$.

Функцию Шпрага–Гранди можно вычислять в некоторых случаях, выражая игру через более простые. Для этого определим *сумму игр*.

Пусть есть две игры рассматриваемого типа с множествами позиций P_1 и P_2. Суммой $P_1 \oplus P_2$ называется игра, в которой позиции — это пары позиций в первой и второй игре, а ход состоит в переходе от позиции (p_1, p_2) либо к позиции (p_1', p_2), либо к позиции (p_1, p_2'), где в первой игре возможне ход из позиции p_1 в позицию p_1', а во второй — из позиции p_2 в позицию p_2'.

Другими словами это можно объяснить так: игроки параллельно играют в игры P_1 и P_2, на каждом ходу игрок должен сделать ход в одной из игр.

Теорема 12.3. *Пусть $v_1 \colon P_1 \to \mathbb{N}$; $v_2 \colon P_1 \to \mathbb{N}$ — функции Шпрага–Гранди для игр P_1, P_2. Тогда функция Шпрага–Гранди для игры $P_1 \oplus P_2$ задается как*

$$v(p_1, p_2) = v(p_1) \oplus v(p_2),$$

где $x \oplus y$ — число, двоичная запись которого является поразрядной суммой по модулю 2 двоичных записей чисел x и y.

Задача 12.23. Проверьте, что из теоремы следует правило оценки позиций игры в ним с тремя кучками.

В доказательстве теоремы нам потребуются свойства поразрядного сложения по модулю 2.

Лемма 12.4. *1. $x \oplus y = y \oplus x$;*

2. если $a \neq a'$, то $a \oplus x \neq a' \oplus x$ для любого x;

3. если $x < a \oplus b$, то либо $x = a' \oplus b$, $a' < a$; либо $x = a \oplus b'$, $b' < b$.

Доказательство. Первое свойство очевидно из определения.

Второе свойство тоже вполне очевидно: если поразрядно прибавить к двум разным двоичным строкам одну и ту же строку, полученные суммы будут различаться в тех же самых позициях, что и исходные строки.

Теперь докажем третье свойство. В самом старшем разряде, в котором x отличается от $a \oplus b$, в двоичной записи числа x стоит 0, а в двоичной записи $a \oplus b$ стоит 1. Поэтому в двоичных записях чисел a, b в этом разряде ровно одна единица. Считаем без ограничения общности, что в этом разряде запись a содержит 1, а запись b содержит 0.

Построим число a', поместив в этот разряд 0, а в более младших разрядах поставим такие значения, чтобы выполнялось равенство $x = a' \oplus b$ (число a' однозначно задаётся этими условиями и числами x, b). По правилу сравнения чисел $a' < a$, что и требовалось. □

Доказательство теоремы 12.3. Для позиций, из которых нельзя сделать хода, утверждение теоремы очевидно, так как $0 \oplus 0 = 0$.

Теперь проверим, что число $v(p_1) \oplus v(p_2)$ отличается от всех чисел $v(p_1') \oplus v(p_2)$; $v(p_1) \oplus v(p_2')$, где из p_1 есть ход в p_1' в игре P_1, а из p_2 есть ход в p_2' в игре P_2.

По построению функции Шпрага–Гранди $v(p_1') \neq v(p_1)$; $v(p_2') \neq v(p_2)$. Поэтому достаточно применить второе свойство из леммы 12.4.

Осталось доказать, что $v(p_1) \oplus v(p_2)$ — наименьшее из чисел, которые отличаются от всех чисел $v(p_1') \oplus v(p_2)$; $v(p_1) \oplus v(p_2')$. Для этого используем третье свойство из леммы 12.4.

Пусть $k < v(p_1) \oplus v(p_2)$. Тогда по указанному свойству либо $k = v_1 \oplus v(p_2)$, $v_1 < v(p_1)$; либо $k = v(p_1) \oplus v_2$, $v_2 < v(p_2)$. В обоих случаях применяем свойство функции Шпрага–Гранди к соответствующей игре и видим, что $v_1 = v(p_1')$ либо $v_2 = v(p_2')$. Поэтому $k = v(p_1') \oplus v(p_2)$ или $k = v(p_1) \oplus v(p_2')$, причём в первом случае в первой игре есть ход из p_1 в p_1', а во втором случае во второй игре есть ход из p_2 в p_2'. □

Задача 12.24. Рассмотрим игру в три монетницы (ним с тремя кучками камней, разрешается брать 2 или 3 камня за ход). У кого из игроков есть выигрышная стратегия, если в монетницу помещается 12 монет?

Часть III

Вычислимость

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.

Лекция 14

Булевы схемы и формулы

14.1 Булевы схемы

Булевой схемой от переменных $x_1, \ldots, x_n$ мы будем называть последовательность булевых функций $g_1, \ldots, g_s$, в которой всякая g_i или равна одной из переменных, или получается из предыдущих применением одной из логических операций отрицание, конъюнкция и дизъюнкция. Другими словами, для всякого i или $g_i = x_j$, $1 \leqslant j \leqslant n$, или существуют такие $j, k < i$, что $g_i = g_j \wedge g_k$, $g_i = g_j \vee g_k$ или $g_i = \neg g_j$. Также в булевой схеме задано некоторое число $m \geqslant 1$ и члены последовательности $g_{s-m+1}, \ldots, g_s$ называются выходами схемы (их как раз m). Число m называют числом выходов схемы. Мы говорим, что схема вычисляет булево отображение $f\colon \{0,1\}^n \to \{0,1\}^m$, если для всякого $x \in \{0,1\}^n$ верно $f(x) = (g_{s-m+1}(x), \ldots, g_s(x))$. Размером схемы называют число s.

Пример 14.1. Схема

$$x_1, x_2, x_1 \wedge x_2$$

с одним выходом вычисляет конъюнкцию переменных x_1 и x_2. Размер этой схемы равен 3.

Схема

$$x_1, x_2, \neg x_1, \neg x_2, x_1 \wedge \neg x_2, \neg x_1 \wedge x_2, (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$$

с одним выходом вычисляет, как нетрудно проверить, функцию $x_1 \oplus x_2$. Размер этой схемы равен 7.

Схема

$$x_1, x_2, x_3, x_1 \vee x_2, (x_1 \vee x_2) \wedge x_3, (x_1 \wedge x_2), ((x_1 \vee x_2) \wedge x_3) \vee (x_1 \wedge x_2)$$

с одним выходом вычисляет функцию $MAJ_3(x_1, x_2, x_3)$. Эта функция равна 1 тогда и только тогда, когда хотя бы две из ее переменных равны 1. Размер этой схемы равен 7.

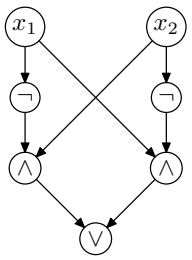

Рис. 14.1: Схема для функции $x_1 \oplus x_2$

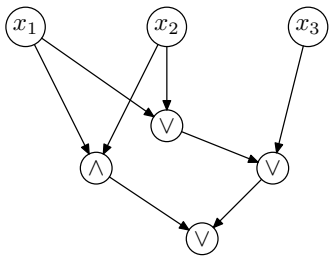

Рис. 14.2: Схема для функции MAJ_3

На практике удобно изображать схемы не в виде последовательности функций, а в виде ориентированного графа с s вершинами $v_1, \ldots, v_s$ (см. примеры на Рис. 14.1, 14.2). Вершины этого графа соответствуют функциям схемы: вершина v_i соответствует функции g_i. Вершины $v_1, \ldots, v_n$ при этом помечены переменными $x_1, \ldots, x_n$ соответственно. Всякая вершина v_i для $i > n$ помечена логической связкой, с помощью которой функция g_i получена из предыдущих. При этом, если функция g_i была получена из функций g_j и g_k то мы проводим ребра из вершин v_j и v_k в вершину v_i. Если функция g_i получена как отрицание g_j, то мы проводим только ребро из v_j в v_i. Вершины $v_{s-m+1}, \ldots, v_s$ помечены как выходные вершины схемы.

Рассмотрим более содержательные примеры.

Пример 14.2. Пусть нам даны две n-битовых двоичных записи чисел x и y и мы хотим вычислить двоичную запись их суммы $z = x + y$. Для удобства обозначим $x = x_{n-1} \ldots x_1 x_0$, где x_0 – младший разряд двоичной записи. Аналогично, $y = y_{n-1} \ldots y_1 y_0$. Во-первых, заметим, что в двоичной записи z будет не более $n+1$ разрядов. Так что мы хотим построить схему с $2n$ входами и $n+1$ выходом.

Идея конструкции схемы будет та же, что и в обычном школьном сложении в столбик. Мы будем складывать числа x и y поразрядно, попутно вычисляя биты переноса в следующий разряд.

Для удобства будем обозначать через b_i бит, который переносится в i-ый разряд из предыдущих.

Заметим, что мы уже готовы вычислить первый разряд ответа $z_0 = x_0 \oplus y_0$. Конечно, мы не можем сразу применить операцию $\oplus$, но выше мы показали, как ее можно вычислить небольшой схемой. Добавим эту маленькую схему в нашу, как подсхему. Далее, заметим, что $b_1 = x_0 \wedge y_0$, добавим соответствующий элемент в схему. Перейдем к следующему разряду. Здесь $z_1 = x_1 \oplus y_1 \oplus b_1$ и $b_2 = MAJ_3(x_1, y_1, b_1)$. Для вычисления первого добавим сначала подсхему, вычисляющую промежуточную величину $c_1 = x_1 \oplus y_1$, а затем подсхему, вычисляющую $z_1 = c_1 \oplus b_1$. Для вычисления b_2

просто добавим подсхему, вычисляющую функцию MAJ_3. Такая схема также приведена выше. Дальше, случай произвольных z_i и b_i полностью аналогичен случаю z_1 и b_1 и мы можем последовательно вычислить все эти значения.

Оценим теперь размер описанной схемы. Для каждого разряда ответа нам нужно не больше двух раз применить подсхему для вычисления функции $\oplus$ и не более одного раза подсхему для вычисления MAJ_3. Все эти схемы имеют постоянный размер, так что для вычисления каждого разряда z мы используем постоянное число элементов. Всего нам нужно $O(n)$ элементов.

Пример 14.3. Построим теперь схему для умножения n-битовых чисел. Пусть на вход снова подаются два числа $x = x_{n-1}\dots x_1x_0$ и $y = y_{n-1}\dots y_1y_0$. На этот раз мы хотим вычислить $z = x \cdot y$. Заметим, что z имеет не больше $2n$ разрядов. Действительно, $x, y < 2^n$, так что $z = x \cdot y < 2^{2n}$, а значит для его записи достаточно $2n$ разрядов.

Для вычисления z снова воспользуемся школьным методом. В нем умножение двух чисел сводится к сложению n чисел. Действительно, чтобы умножить x на y достаточно для всякого $i = 0, \dots, n-1$ умножить x на y_i, приписать в конце числа i нулей и затем сложить все полученные числа. Умножение x на y_i легко реализуется с помощью n конъюнкций. После этого остается сложить n чисел длины не более $2n$. Для этого мы можем $n-1$ раз применить схему для сложения, описанную выше. Размер каждой схемы для сложения линейный, так что суммарная сложность схемы для умножения получается $O(n^2)$.

Пример 14.4. Мы разобрали, как в модели булевых схем выполнять базовые арифметические операции. Теперь обсудим одну из простейших комбинаторных задач. Нам дан неориентированный граф G на n вершинах, нужно проверить, является ли он связным. Во-первых, стоит обсудить, как задавать граф в пригодном для схем виде. Для этого удобно воспользоваться так называемой *матрицей смежности* графа. Перенумеруем вершины графа $v_1, v_2, \dots, v_n$. Матрицей смежности графа G называется матрица $A \in \{0,1\}^{n\times n}$, в которой на пересечении строки i со столбцом j стоит 1 тогда и только тогда, когда $v_i, v_j \in E$. Матрица смежности полностью описывает, какие пары вершин соединены ребрами, так что булевой схеме достаточно подать на вход матрицу смежности графа. Более того, заметим, что матрица смежности симметрична и на диагонали у нее обязательно стоят нули (мы запрещали петли – ребра, ведущие из вершины в нее же саму). Так что на вход схеме можно подать, скажем, только верхнюю половину матрицы смежности.

Оказывается, матрица смежности также удобна для проверки связности графа. Можно матрицу A интерпретировать следующим образом: на пересечении строки i и столбца j написано количество путей длины 1 из вершины v_i в вершину v_j. Теперь возведем матрицу A в квадрат (над действительными числами). Если посмотреть на формулу для произведения матриц можно заметить, что на пересечении строки i и столбца j матрицы A^2 записано количество путей длины 2 из вершины v_i в вершину v_j. По индукции можно доказать, что на пересечении строки i и столбца j матрицы A^k записано число путей длины k из вершины v_i в вершину v_j. Заметим теперь, что

если между какими-то двумя вершинами в графе есть путь, то обязательно есть путь длины не больше $n-1$ (из пути всегда можно выкинуть циклы, если они там есть, после этого все вершины в пути разные). Так что для проверки связности у нас появляется такой план: вычислим все матрицы $A, A^2, \ldots, A^{n-1}$ и для каждой пары вершин v_i и v_j проверим, есть ли между ними путь длины не больше $n-1$. Если это так, то граф связен, иначе не связен.

Этот план можно несколько упростить. Во-первых, можно немного модифицировать матрицу смежности. Рассмотрим матрицу A', которая отличается от матрицы A тем, что у нее на главной диагонали стоят единицы, а не нули (в остальном матрицы совпадают). В терминах графов это означает, что к каждой вершине мы добавляем петлю. В модели простых неориентированных графов мы этого не допускали, но ничего не мешает нам рассмотреть графы с петлями. Идея состоит в том, что теперь, если между двумя вершинами есть путь длины меньше $n-1$, то есть и путь длины ровно $n-1$ (достаточно добавить к пути нужное количество петель). Так что теперь не обязательно смотреть на все степени матрицы смежности, достаточно взглянуть на $(A')^{n-1}$. Если в ячейках этой матрицы нет нулей, то граф связен, иначе не связен.

Второе упрощение связано со способом возведения матрицы в степень. В данный момент мы это делаем над действительными числами, что заставляет нас складывать и умножать целые числа. Чтобы делать это с помощью схем, нам придется использовать описанные выше схемы для сложения и умножения, а чтобы оценить размер получившейся схемы придется оценивать величину возникающих в процессе вычислений целых чисел. Все это не очень хочется делать. Решение состоит в том, чтобы вместо умножения матриц над целыми числами воспользоваться так называемым *булевым умножением матриц*. В нем формулы для умножения матриц такие же, как и в обычном умножении, только вместо операции умножения используется конъюнкция, а вместо сложения – дизъюнкция. Тогда можно по индукции доказать, что в (булевой) матрице $(A')^k$ на пересечении строки i и столбца j стоит 1 тогда и только тогда, когда в графе есть путь из v_i в v_j длины не больше k.

Теперь мы готовы описать схему для проверки графа на связность. На вход схема (по существу) получает матрицу смежности A'. Схема последовательно вычисляет булевы степени этой матрицы $(A')^2, \ldots, (A')^{n-1}$. Затем схема вычисляет конъюнкцию всех ячеек матрицы $(A')^{n-1}$ и подает ее на выход.

Оценим размер получившийся схемы. Для булева умножения двух булевых матриц $n \times n$ достаточно $n^2 \cdot O(n) = O(n^3)$ операций (каждая ячейка произведения матриц вычисляется за линейное число операций, всего ячеек n^2). Всего нам нужно $(n-1)$ умножение матриц, так что для вычисления матрицы $(A')^{n-1}$ достаточно $O(n^4)$ операций. На последний этап (конъюнкция ячеек $(A')^{n-1}$) нужно $O(n^2)$ операций, итого получается $O(n^4) + O(n^2) = O(n^4)$ операций.

Задача 14.5. При построении схемы, реализующей описанный нами алгоритм, и при подсчете числа ее элементов мы действовали довольно грубо. Так, то же самое можно было сделать за $O(n^3 \log n)$ операций. Как это сделать?

Мы увидели, что конкретные функции можно вычислять схемами, причем раз-

мер схем получается не слишком большим (полиномиальным). Далее заметим, что всякую булеву функцию можно вычислить булевой схемой. По существу мы уже обсуждали этот вопрос, когда обсуждали дизъюнктивную нормальную форму функции. Для полноты изложения повторим кратко это рассуждение.

Лемма 14.1. *Всякую функцию $f\colon \{0,1\}^n \to \{0,1\}$ можно вычислить схемой размера не больше $O(n2^n)$.*

Доказательство. Для всякого $a \in \{0,1\}^n$ рассмотрим функцию $f_a\colon \{0,1\}^n \to \{0,1\}$, такую что $f(x) = 1$ тогда и только тогда, когда $x = a$. Будет удобно ввести обозначение $x^1 = x$ и $x^0 = \neg x$. Тогда функцию f_a можно записать формулой

$$f_a(x) = \bigwedge_{i=1}^{n} x_i^{a_i},$$

где $x = (x_1, \ldots, x_n)$ и $a = (a_1, \ldots, a_n)$.

Для произвольной функции f уже не сложно записать формулу через функции f_a:

$$f(x) = \bigvee_{a \in f^{-1}(1)} f_a(x).$$

Теперь эти формулы можно переделать в схему. Наша схема сначала будет вычислять отрицания всех переменных, на это нужно n элементов. После этого можно вычислить все функции f_a. Для вычисления каждого нужно $n-1$ раз применить конъюнкцию. Всего получается $2^n(n-1)$ элемент. Наконец, для вычисления f нужно взять дизъюнкцию нужных функций f_a, на это уйдет не более 2^n элементов. Суммарно в нашей схеме получается $O(n2^n)$ элементов. □

Из этого несложно получить схему для произвольной функцию с многими выходами $f\colon \{0,1\}^n \to \{0,1\}^m$.

Следствие 14.2. *Всякую функцию $f\colon \{0,1\}^n \to \{0,1\}^m$ можно вычислить схемой размера не больше $O(nm2^n)$.*

Доказательство. Для данной функции f рассмотрим функции $f_i\colon \{0,1\}^n \to \{0,1\}$, для $i = 1, \ldots, m$, такие что для всякого $x \in \{0,1\}^n$ $f(x) = (f_1(x), \ldots, f_m(x))$. Иными словами f_i выдает i-ую координату f. Мы можем вычислить каждую функцию f_i схемой размера $O(n2^n)$. Объединив эти схемы в одну, мы получим схему для f размера $O(mn2^n)$. □

Мы показали, что всякую функцию можно вычислить схемой, но размер схемы при этом получился большим. Оказывается, это неизбежно.

Теорема 14.3. *Для всякого $n > 1$ существует функция $f\colon \{0,1\}^n \to \{0,1\}$, которую нельзя вычислить схемой размера меньше $2^n/10n$.*

Доказательство. Для доказательства применим мощностной метод: докажем, что функций больше, чем маленьких схем. Тогда маленьких схем не хватит, чтобы вычислить все функции.

Всего булевых функций от n переменных 2^{2^n}.

Далее, заметим, что всякую схему размера не больше S можно описать с помощью не больше чем $S \cdot (2 + 2\log_2 S)$ битов. Действительно, для каждого из S элементов нужно указать его тип (конъюнкция, дизъюнкция или отрицание), на что достаточно потратить два бита, а также указать из каких предыдущих элементов он получен, на что уходит $2\log S$ битов (можно просто выписать номера элементов). Поскольку $S \geqslant n \geqslant 2$, то $S \cdot (2 + 2\log_2 S) \leqslant 4S\log S$.

Если теперь взять $S = 2^n/10n$, то получается, что всякую схему размера не больше S можно описать строкой из не более $4\frac{2^n}{10n}(n - \log_2(10n)) \leqslant 2 \cdot 2^n/5$ битов. Значит всего таких схем не больше, чем количество таких строк, то есть не больше $2^{2\cdot 2^n/5}$. Видно, что это меньше 2^{2^n}, а значит не всякую функцию можно вычислить схемой размера $\leqslant S$. □

Мы доказали верхнюю и нижнюю оценку сложности функций n переменных, причем наши оценки достаточно близки: и та, и та оценка экспоненциальная. Если интересоваться более точным значением максимальной сложности функции, то оказывается, что наша нижняя оценка точнее, то есть можно доказать, что всякую функцию от n переменных можно вычислить схемой размера не больше $O(2^n/n)$. Однако, доказать этот факт не так просто, и мы не будем приводить здесь его доказательство.

Заметим, что наше доказательство существования трудной функции не конструктивное, "явной" функции мы не предъявляем. Оказывается, это сделать не так просто: предъявить достаточно простую функцию со сверхлинейной схемной сложностью – это открытый вопрос.

Для иллюстрации покажем, как можно доказывать хотя бы линейные нижние оценки сложности конкретных булевых функций.

Для этого рассмотрим функцию XOR_n от n переменных. Эта функция задается формулой

$$XOR_n(x_1, \ldots, x_n) = x_1 \oplus x_2 \oplus \ldots \oplus x_n.$$

Ясно, что ее можно вычислить схемой линейного размера. Действительно, можно сначала вычислить $x_1 \oplus x_2$, пользуясь схемой описанной выше, затем вычислить $(x_1 \oplus x_2) \oplus x_3$, пользуясь той же схемой, и так далее. На добавление каждой переменной требуется 5 элементов), так что общий размер схемы получается $n+5(n-1) = 6n-5$.

Теперь мы докажем нижнюю линейную оценку.

Лемма 14.4. *Сложность булевых схем для функции XOR_n не меньше $3n - 2$.*

Доказательство. Нам будет удобнее доказать более общий факт. Давайте рассмотрим булевы схемы, в которых используются не только операции конъюнкции, дизъюнкции и отрицания, но и произвольные операции вида $(x^a \wedge x^b)^c$. Иначе говоря, мы можем использовать конъюнкции и дизъюнкции и брать при этом отрицания

бесплатно, то есть не учитывать их при подсчете размера схемы. Ясно, что теперь мы рассматриваем более общие схемы, а значит доказывать нижние оценки будет только труднее.

Мы будем вести доказательство индукцией по n. И на самом деле, удобно будет усилить доказываемое утверждение: мы будем доказывать нижнюю оценку $3n-2$ одновременно для двух функций XOR_n и $\neg XOR_n$.

Итак, для $n=1$ нам нужно доказать, что размер схемы не меньше 1, что очевидно. Пусть мы доказали утверждение для n, докажем его для $n+1$. Пусть, от противного есть схема размера $S \leqslant 3(n+1)-3 = 3n$ для функции XOR_{n+1} или для функции $\neg XOR_{n+1}$. Будем также считать, что это минимальная схема, то есть схемы размера меньше S для этих функций нет. Рассмотрим первый элемент g_{n+2} этой схемы, после переменных. Ясно, что ему на вход подаются две переменные. Пусть, для определенности, это переменные x_1 и x_2. Заметим, что можно так зафиксировать переменную x_1, что элемент g_{n+2} всегда оказывается равен константе. Действительно, если g_{n+2} – конъюнкция, то нужно сделать то, к чему конъюнкция применяется (x_1 или ее отрицание), равным 0. Если же g_{n+2} – дизъюнкция, то нужно сделать то, к чему применяется дизъюнкция, равным 1. Далее, заметим, что g_{n+2} подается на вход какому-то еще элементу (иначе g_{n+2} можно было бы просто удалить из схемы, что противоречит минимальности S). Пусть это элемент g_k и второй его вход – элемент g_j. Заметим, что после того, как g_{n+2} обратился в константу, элемент g_k равен либо g_j, либо его отрицанию.

Зафиксируем переменную x_1 так, чтобы g_{n+2} стал константой. Удалим из схемы элементы g_1, g_{n+2} и g_k. Все элементы, в которые подставлялись g_1 и g_{n+2} попрежнему можно вычислить – они вычисляют либо какие-то предыдущие элементы, либо их отрицания. Если в какой-то элемент подставлялся g_k, то вместо него можно подставить g_j, либо его отрицание. Так что после удаления этих трех элементов мы снова получили некоторую булеву схему. Эта схема вычисляет функцию XOR_n или функцию $\neg XOR_n$ от оставшихся n переменных. При этом размер схемы равен $S-3 \leqslant 3n-3$. Мы получили противоречие с предположением индукции. □

До сих пор мы обсуждали размер схем. Вторым важным параметром схемы является ее глубина. Рассмотрим схему как ориентированный граф. Глубиной вершины в схеме называется длина наибольшего пути из какой-нибудь переменной в эту вершину. Глубиной схемы называется максимальная глубина вершины в этой схеме. Например, глубина схемы для функции $x_1 \oplus x_2$ на рисунке 14.1 равна 3. Глубина схемы для MAJ_3 на рисунке 14.2 также равна 3.

Неформально, глубина схемы характеризует время, необходимое для вычисления значения схемы на данном входе, если вычисление элементов схемы можно производить параллельно: на первом шаге можно параллельно вычислить все элементы глубины 1, на втором – все элементы глубины 2, и так далее, на некотором k-м шаге можно параллельно вычислить все элементы глубины k. Это можно сделать, так как на вход элементам глубины k подаются только элементы меньшей глубины, а их значения уже вычислены.

Пример 14.6. Рассмотрим функцию $x_1 \wedge x_2 \wedge \ldots \wedge x_n$. Ее можно вычислить схемой размера $n-1$, просто вычислив последовательно $x_1 \wedge x_2$, $x_1 \wedge x_2 \wedge x_3$ и так далее. Однако глубина такой схемы также равна $n-1$, поскольку каждый следующий элемент получает на вход предыдущий. Ту же функцию можно вычислить схемой глубины $\lceil \log_2 n \rceil$, если организовать схему как двоичное дерево. Разобьем переменные на пары x_1 и x_2, x_3 и x_4 и так далее. Вычислим конъюнкцию каждой пары. Все эти элементы имеют глубину 1. Повторим рассуждение: разобьем полученные элементы на пары и вычислим конъюнкцию парных элементов. Если количество переменных n есть степень двойки, то мы получим полное двоичное дерево глубины $\log_2 n$. Если количество элементов не равно степени двойки, то на некоторых шагах у некоторых элементов не будет парных и мы будем брать их конъюнкцию с самими собой (по существу, мы будем переносить их на следующий шаг). В результате получится поддерево полного двоичного дерева глубины $\lceil \log_2 n \rceil$.

То же самое рассуждение применимо к функции $x_1 \vee x_2 \vee \ldots \vee x_n$ и к функции XOR_n. Их обе можно вычислить схемой глубины $O(\log n)$.

Из приведенного выше примера и конструкции схемы для произвольной функции из леммы 14.1 видно, что всякую функцию можно вычислить схемой глубины $O(n)$. Действительно, в конструкции с дизъюнктивной нормальной формой возникают конъюнкции n элементов и дизъюнкции 2^n элементов. Если вычислять их так, как описано выше, то глубина схемы будет $O(n)$.

Обсудим подробнее глубину схем для функции “связность”.

Лемма 14.5. *Проверку графа на n вершинах на связность можно вычислить схемой глубины не больше $O(\log^2 n)$.*

Доказательство. Идея состоит в том, чтобы использовать уже построенную выше схему для этой задачи с некоторыми модификациями.

Мы будем вычислять степени матрицы смежности A' с единицами на диагонали. Во-первых, оценим за какую глубину можно по данной матрице $B \in \{0,1\}^{n \times n}$ вычислить матрицу B^2, где умножение матриц – булево. Ячейка (i,j) матрицы B^2 вычисляется по формуле

$$\bigvee_{k=1}^{n} (b_{ik} \wedge b_{kj}).$$

На вычисление всех конъюнкций нам потребуется глубина 1, а на вычисление дизъюнкции, как мы обсудили выше, достаточно глубины порядка $\log_2 n$. Поскольку все ячейки матрицы B^2 можно вычислять параллельно, мы получаем, что для возведения матрицы в квадрат достаточно глубины $O(\log n)$, если быть точными, то $\lceil \log_2 n \rceil + 1$.

Далее, как мы уже обсуждали, чтобы проверить, есть ли между двумя данными вершинами путь в графе на n вершинах, достаточно проверить, есть ли путь длины не больше n. Вместо этого нам будет удобнее проверить, есть ли в графе пути длины не больше 2^k, где $k = \lceil \log_2 n \rceil$. Ясно, что $2^k \geqslant n$, так что этого для нас будет

достаточно. С другой стороны, для такой проверки нам достаточно вычислить матрицу $(A')^{2^k}$. Для этого мы можем просто последовательно возвести матрицу A' в квадрат k раз. Матрица A' нам по существу подается на вход, на каждое возведение в квадрат мы тратим глубину $O(\log n)$, так что, поскольку мы возводим в квадрат k раз, суммарная глубина есть $kO(\log n) = O(\log^2 n)$.

После этого нужно лишь взять конъюнкцию всех ячеек матрицы $(A')^{2^k}$, что можно сделать за глубину $O(\log n^2) = O(\log n)$. Так что общая глубина всей схемы есть $O(\log^2 n) + O(\log n) = O(\log^2 n)$. □

Задача 14.7. В доказательстве, чтобы вписаться в глубину $O(\log^2 n)$, мы вычислили матрицу $(A')^{2^k}$ вместо матрицы $(A')^n$. Но на самом деле, можно было обойтись и без этого трюка. Докажите, что матрицу $(A')^n$ также можно вычислить схемой глубины $O(\log^2 n)$.

Задача 14.8. В предыдущей лемме мы рассматривали задачу проверки на связность неориентированного графа. Можно рассмотреть аналогичную задачу для ориентированных графов. Докажите, что и для этой задачи есть схема глубины $O(\log^2 n)$.

Отметим, что вопрос о том, можно ли проверить граф (ориентированный или неориентированный) на связность схемой глубины $O(\log n)$, является важным открытым вопросом сложности вычислений. Можно, однако, сказать, что ориентированный случай не проще неориентированного.

Задача 14.9. Докажите, что если для задачи проверки ориентированного графа на связность есть схема глубины $O(\log n)$, то такая схема есть и для проверки неориентированного графа на связность.

В оставшейся части лекции мы сосредоточимся на схемах с одним выходом. Такие схемы от n переменных вычисляют функции вида $f\colon \{0,1\}^n \to \{0,1\}$.

14.2 Формулы

Рассмотрим булевы схемы специального вида, которые мы будем называть *формулами*. В них из каждой вершины, кроме выхода схемы, выходит ровно одно ребро. Для единообразия удобнее будет считать, что допускается несколько вершин, помеченных одной и той же переменной, и из каждой такой вершины также выходит ровно одно ребро. На рисунке 14.3 приведен пример формулы для функции $x_1 \oplus x_2$. Таким образом, формулами называются схемы, графы которых являются деревьями. В терминах последовательности функций, схема является формулой, если в ней разрешается выписывать переменные по несколько раз, и каждый элемент, кроме последнего, используется для построения одного из следующих ровно один раз.

Формулами такие схемы называются не случайно. Несложно заметить, что они соответствуют формулам булевой логики, которые мы обсуждали в начале курса. Действительно, рассмотрим формулу булевой логики ϕ, то есть формулу от переменных $x_1, \ldots, x_n$ со связками конъюнкция, дизъюнкция и отрицание. Построим

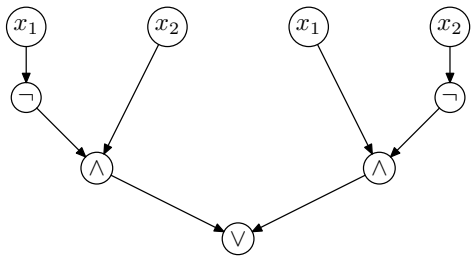

Рис. 14.3: Формула для функции $x_1 \oplus x_2$

по этой формуле схему. Для каждого вхождения переменной в формулу добавим в схему вершину, помеченную этой переменной (например, если переменная x_1 встречается в формуле 7 раз, то мы добавляем в схему 7 соответствующих вершин). Посмотрим на связку, которая применяется в формуле первой и добавим соответствующий элемент в схему. Посмотрим на связку, применяющуюся следующей, и добавим элемент, с такой же связкой, примененный к тем же подформулам, и так далее. Тогда элементы схемы будут вычислять подформулы формулы ϕ, а последний элемент будет равен самой ϕ. Видно, что при этом и из каждого элемента выходит ровно одна стрелка (ко всякой подформуле связка применяется только один раз).

С другой стороны, всякую схему-формулу (то есть, схему с одним выходным ребром из каждой вершины) можно обратно переделать в формулу булевой логики: можно просто последовательно во всех элементах выписать формулу, которая в этом элементе вычисляется. Для переменных соответствующая формула – просто переменная. Для каждого следующего элемента формула получается применением соответствующей связки к формулам, соответствующим предыдущим элементам.

Нетрудно видеть, что эти два сведения обратны друг к другу, то есть построив по формуле булевой логики схему, а затем по ней обратно формулу булевой логики, мы придем к той же формуле с которой начали.

Таким образом, формулы булевой логики – это в точности схемы, в которых из каждой вершины выходит не более одного ребра. Размер такой схемы при этом соответствует числу связок плюс числу вхождений переменных в формулу булевой логики. На рисунке 14.3 приведен пример схемы, соответствующей формуле $(\neg x_1 \wedge x_2) \vee (x_1 \wedge \neg x_2)$.

Формулы – это частный случай схем. При этом всякую схему можно переделать в формулу. Глубина при этом не увеличится, однако размер может вырасти экспоненциально.

Лемма 14.6. *По всякой схеме C от переменных $x_1, \ldots, x_n$ глубины d и размера s можно построить формулу, вычисляющую ту же функцию, глубины не больше d*

и размера не больше $2^s - 1$.

Доказательство. Доказательство можно вести индукцией по s и d. По существу, мы доказываем утверждения для глубины и размера отдельно, но рассуждение одинаково, так что мы сделаем это параллельно. В качестве базы рассмотрим формулу, вычисляющую одну из переменных. Ее размер n и глубина равна нулю. Собственно, эта схема уже является формулой и утверждение леммы очевидно ($2^n - 1 \geqslant n$ для всякого n).

Предположим, что для схем размера меньше s (глубины меньше d) утверждение уже доказано. Рассмотрим схему размера s (глубины d). Посмотрим на ее последний элемент g_s. Он получается применением какой-то логической связки к предыдущим элементам g_i и g_j. Рассмотрим подсхемы нашей схемы, вычисляющие g_i и g_j. Их размер меньше s (глубина меньше d), так что по предположению индукции их можно вычислить формулами размера не больше $2^{s-1} - 1$ (глубины меньше d).

Добавим к этой паре формул новую вершину, соответствующую элементу g_s и проведем в нее ребра из элементов, вычисляющих g_i и g_j. Полученная схема имеет размер не больше $2 \cdot (2^{s-1} - 1) + 1 = 2^s - 1$ (глубину не больше d).

Этот же процесс можно описать и напрямую. Для этого отсортируем вершины по слоям по их глубине, то есть рассмотрим отдельно вершины глубины 1, вершины глубины 2, и так далее. Будем для всех вершин последовательно добиваться того, чтобы из них выходило не более одного ребра, при этом будем рассматривать вершины по убыванию их глубины. Из вершин глубины d ребер не выходит, так что для них условие выполняется изначально. Пусть мы уже добились выполнения условия для вершин глубины больше k. Рассмотрим вершины глубины k. Если из каких-то из них выходит больше одного ребра, то сделаем несколько копий этой вершины, по числу выходящих ребер, и из каждой копии вершины выпустим одно ребро.

Зачем мы делаем этот процесс по убыванию глубины: "размножение" вершин глубины больше k может увеличить выходную степень вершин глубины k, так что мы сначала размножаем все вершины глубины больше k, и только потом смотрим на вершины глубины k, их выходная степень уже не вырастет. □

Рассмотрим теперь вопрос о соотношении глубины и размера схем. Между этими величинами несложно доказать неравенство в одну сторону.

Лемма 14.7. *Всякая схема глубины d эквивалентна (то есть, вычисляет ту же функцию) некоторой схеме глубины d и размера не больше* 2^{d+1}.

Доказательство. На самом деле, строить эквивалентную схему по существу не нужно. Достаточно просто удалить из текущей схемы все неиспользующиеся элементы. Более точно, если в схеме есть вершина, не являющаяся выходом, из которой не выходит ребер, то такую вершину можно безболезненно удалить из схемы. Будем удалять подобные вершины до тех пор, пока их не останется.

После этого можно утверждать, что размер схемы не больше 2^{d+1}. Действительно, можно переделать эту схему в формулу по предыдущей лемме. При этом

процессе размер схемы не уменьшается. Но получившаяся в конце формула является поддеревом двоичного дерева глубины не больше d, так что в ней не больше 2^{d+1} вершин.

□

Таким образом, мы доказали, что размер схемы не больше экспоненты от глубины схемы. Можно ли доказать обратное соотношение, является открытым вопросом.

Задача 14.10. Докажите, что если для всякой схемы от n переменных размера s существует эквивалентная схема глубины не больше $O(\log s)$, то достижимость в графе можно проверить схемой глубины $O(\log n)$.

Однако, оказывается, что для частного случая формул обратное утверждение также верно.

Теорема 14.8. *Для всякой формулы размера s есть эквивалентная формула глубины $3\log s$.*

Вспомнив, что формула – это дерево, можно заметить, что результат этой теоремы означает, что для всякой формулы есть эквивалентная формула, которая не сильно больше изначальной, и в которой ветви имеют примерно одинаковую длину.

Доказательство. В этой теореме нам уже придется всерьёз перестраивать формулу.

Для этого нам потребуется вспомогательное утверждение: нам потребуется "средняя" вершина в дереве.

Утверждение 14.9. *Пусть в формуле ϕ имеет размер s. Тогда в ней есть подформула ψ размера не меньше $s/2$, такая что все подформулы формулы ψ имеют размер меньше $s/2$.*

Доказательство этого утверждения совсем простое. Достаточно начать из выходного элемента и спускаться по ребрам формулы в подформулы так, чтобы каждый раз оставаться в подформуле размера не меньше $s/2$. В выходном элементе размер подформулы равен s, а в переменных размер подформулы равен 1, так что в какой-то момент мы не сможем спуститься из очередной вершины в подформулу. Это как раз и будет означать, что у текущей подформулы размер не меньше $s/2$, а во всех ее подформулах уже меньше.

Теперь сделаем следующее. Посмотрим на вершину, соответствующую подформуле ψ и удалим из дерева поддерево, соответствующее этой подформуле. Теперь у дерева образовался новый лист, в той вершине, где раньше была подформула ψ. Пометим эту вершину константой 0. Мы получили новую формулу, обозначим результат ее вычисления через ϕ_0. Аналогично сделаем для константы 1: пометим новый лист единицей и обозначим результат через ϕ_1.

Теперь рассмотрим следующую формулу: $(\phi_0 \wedge \psi) \vee (\phi_0 \wedge \neg\psi)$. Нетрудно видеть, что она вычисляет ту же функцию, что и изначальная формула. Теперь мы можем рассуждать индукцией по размеру формулы s. Поскольку размер формулы ψ не меньше $s/2$, то размер формул ϕ_0 и ϕ_1 не больше $s/2$. Так что по предположению

индукции эти функции можно вычислить и формулами глубины $3\log_2 s - 3$. Функцию ψ можно вычислить подформулой глубины $(\log_2 s - 3) + 1$ ($\log_2 s$ достаточно для самых больших подформул ψ и еще единица нужна на вычисление последней операции в ψ). Суммарно, глубина полученной формулы не превышает $3\log_2 s$.

□

Таким образом, мы получаем, что для всякой функции $f\colon \{0,1\}^n \to \{0,1\}$ глубина минимальной схемы и логарифм размера минимальной формулы полиномиально связаны. То есть, изучение размера формул – это по существу то же самое, что изучение глубины схем.