

ОГЛАВЛЕНИЕ

ГЛАВА 7. КОМБИНАТОРНЫЙ ПОИСК	19
7.1. НУЛИ И ЕДИНИЦЫ	71
7.1.1. Основы булевой алгебры	71
7.1.2. Булевы вычисления	124
7.1.3. Битовые трюки и технологии	165
7.1.4. Бинарные диаграммы решений	242
7.2. ГЕНЕРАЦИЯ ВСЕХ ВОЗМОЖНЫХ ОБЪЕКТОВ	329
7.2.1. Генерация основных комбинаторных объектов	329
7.2.1.1. Генерация всех n -кортежей	329
7.2.1.2. Генерация всех перестановок	369
7.2.1.3. Генерация всех сочетаний	408
7.2.1.4. Генерация всех разбиений	444
7.2.1.5. Генерация всех разбиений множеств	471
7.2.1.6. Генерация всех деревьев	498
7.2.1.7. Исторические и иные сведения	547
ОТВЕТЫ К УПРАЖНЕНИЯМ	577
ПРИЛОЖЕНИЕ А. ТАБЛИЦЫ ЗНАЧЕНИЙ НЕКОТОРЫХ КОНСТАНТ	904
ПРИЛОЖЕНИЕ Б. ОСНОВНЫЕ ОБОЗНАЧЕНИЯ	908
ПРИЛОЖЕНИЕ В. СПИСОК АЛГОРИТМОВ И ТЕОРЕМ	914
ПРИЛОЖЕНИЕ Г. УКАЗАТЕЛЬ КОМБИНАТОРНЫХ ЗАДАЧ	916
ПРЕДМЕТНО-ИМЕННОЙ УКАЗАТЕЛЬ	920

ПРЕДИСЛОВИЕ

Поместить все стоящее в одну книгу совершенно невозможно, и даже попытки сколь-нибудь полно осветить хотя бы некоторые аспекты рассматриваемого предмета, скорее всего, приведут к стремительному увеличению книги.

— ДЖЕРАЛЬД Б. ФОЛЛАНД (GERALD B. FOLLAND), *Editor's Corner* (2005)

Том 4 называется *Комбинаторные алгоритмы*. Когда я предлагал это название, я очень хотел добавить к нему подзаголовок *Мой любимый раздел программирования*. Редакторы уговорили меня не делать этого, но факт остается фактом — программы с комбинаторным привкусом всегда были моими любимцами.

С другой стороны, я часто с удивлением обнаруживал, что в сознании многих людей слово “комбинаторный” ассоциируется с вычислительной сложностью. И действительно, Сэмюэль Джонсон (Samuel Johnson) в своем знаменитом Словаре английского языка (1755) писал, что соответствующее существительное “в настоящее время обычно используется в неверном смысле”. От коллег мне приходится слышать горестные рассказы о том, как “комбинаторика нас победила”. Так почему же мне комбинаторика доставляет удовольствие, а у других вызывает панику?

Да, это правда, что комбинаторные задачи часто связаны с чрезвычайно большими числами. Статья в словаре Джонсона включает цитату Эфраима Чамберса (Ephraim Chambers), который указал, что общее количество слов длиной не более 24 букв при 24-буквенном алфавите равно 1 391 724 288 887 252 999 425 128 493 402 200. Соответствующее число для 10-буквенного алфавита равно 11 111 111 110; когда же количество букв алфавита равно всего лишь 5, общее число слов сокращается до 3905. Таким образом, здесь мы, определенно, наблюдаем комбинаторный взрыв с ростом размера задачи от 5 до 10 и далее до 24 и более.

На протяжении моей жизни мощност вычислительной техники выросла неимоверно. Сейчас, когда я набираю эти слова, я знаю, что мой ноутбук обрабатывает их со скоростью, более чем в сто тысяч раз большей, чем старый добрый IBM Type 650, которому я посвящал свои книги. Количество памяти моего нынешнего компьютера также примерно в сто тысяч раз больше. Компьютеры завтрашнего дня будут еще более быстрыми и с еще бóльшим объемом памяти. Но эти удивительные достижения не уменьшили тягу людей к получению ответов на комбинаторные вопросы — скорее даже наоборот. Наша совершенно невообразимая в прошлом способность к быстрым вычислениям повысила наши ожидания и еще больше разожгла аппетит к дальнейшим исследованиям, потому что размер комбинаторной задачи может увеличиться более чем в сто тысяч раз, когда n увеличится всего лишь на единицу.

Неформально комбинаторные алгоритмы можно определить как методы высокоскоростной работы с такими комбинаторными объектами, как перестановки

или графы. Обычно мы пытаемся найти шаблоны или схемы, которые наилучшим образом удовлетворяют определенным ограничениям. Количество задач такого рода огромно, а искусство написания соответствующих программ особенно важно, поскольку одна хорошая идея способна сэкономить годы и даже века машинного времени.

В действительности тот факт, что хорошие алгоритмы для комбинаторных задач могут давать потрясающий выигрыш, привел к огромному подъему современного состояния данной науки. Многие ранее считавшиеся неразрешимыми задачи теперь могут быть с легкостью решены, а многие алгоритмы, считавшиеся ранее очень хорошими, стали гораздо лучше. Начиная примерно с 1970 года кибернетики стали сталкиваться с явлением, которое окрестили леммой Флойда: задачи, которые представляются требующими для решения n^3 операций, на самом деле могут быть решены за $O(n^2)$ операций; задачи, которые, как кажется, требуют n^2 операций, могут быть решены за $O(n \log n)$; задачи со временем решения $n \log n$ зачастую сводятся к задачам со временем решения $O(n)$. Для более сложных задач удается добиться сокращения времени выполнения с $O(2^n)$ до $O(1.5^n)$ или $O(1.3^n)$, и т. д. Прочие задачи остаются сложными в общем случае, но могут быть найдены важные частные случаи, которые решаются гораздо проще. Многие комбинаторные вопросы, о которых я думал, что ответ на них не будет получен при моей жизни, оказались успешно решенными. И это произошло главным образом благодаря усовершенствованию алгоритмов, а не повышению скорости процессоров.

К 1975 году исследования в этой области проводились такими темпами, что значительная доля статей, опубликованных в то время в ведущих журналах по информатике, была посвящена комбинаторным алгоритмам. И этот прорыв был осуществлен не только учеными-кибернетиками; значительный вклад внесли работники в таких областях знаний, как электротехника, искусственный интеллект, исследование операций, математика, физика, статистика и многие другие. Я пытался завершить том 4 *Искусства программирования*, но вместо этого я почувствовал себя сидящим на крышке кипящего чайника: я столкнулся с комбинаторным взрывом другого рода — огромным взрывом новых идей!

Данная серия книг родилась в начале 1962 года, когда я наивно набросал список рабочих названий глав для книги, состоящей из 12 глав. В тот момент я решил включить краткую главу о комбинаторных алгоритмах просто для собственного удовольствия. “Эй, глянь — большинство людей используют компьютеры, чтобы работать с числами, но можно писать программы, которые имеют дело со схемами”. В те времена было легко дать достаточно полное описание почти всех известных комбинаторных алгоритмов. И даже к 1966 году, когда я закончил первый черновой вариант из около трех тысяч рукописных страниц (который уже перерос изначально запланированную книгу), к главе 7 относилось менее ста из них. Я совершенно не представлял в тот момент, что то, что я планировал как “легкую закуску”, окажется основным блюдом.

Великое комбинаторное брожение 1975 года захватило множество людей. Новые идеи улучшали старые, но при этом редко заменяли их или делали устаревшими. Все это привело к тому, что я был вынужден оставить надежду когда-либо написать достаточно полную книгу, в которой был бы упорядочен весь имеющийся материал и которая была бы достаточно универсальной, чтобы помочь каждому, перед кем

встает та или иная комбинаторная задача. Множество применяемых методов разрослось до таких размеров, что я уже практически не мог рассмотреть некоторую подтему и сказать “Это решение окончательное: на этом мы ставим точку”. Вместо этого я должен был ограничиться пояснением наиболее важных принципов, которые, как мне кажется, лежат в основе всех эффективных комбинаторных методов, с которыми я встречался до сих пор. В настоящее время для тома 4 у меня накоплено в два с лишним раза больше материала, чем было собрано для всех предыдущих томов вместе взятых.

Такое количество материала неизбежно ведет к разделению запланированного “тома 4” на несколько физических томов. Сейчас перед вами том 4, А. В один прекрасный день вы увидите тома 4, Б и 4, В, если, конечно, написать их позволит мое здоровье. И, как знать, быть может, наступит очередь томов 4, Г, 4, Д ...? Впрочем, я уверен, что тома 4, Я не будет.

Я планирую по мере возможности систематически разбирать накопленные с 1962 года папки и в меру своих способностей рассказывать истории, которые все еще ждут своего часа. Я не могу гарантировать полноту изложения, но очень хочу воздать должное всем первооткрывателям ключевых идей, а потому я не буду экономить на исторических деталях. Кроме того, всякий раз, когда я знакомлюсь с чем-то, что, как мне кажется, останется важным еще лет 50 и что можно пояснить парой абзацев, я не могу пройти мимо этого и не внести его в книгу. С другой стороны, сложный материал, который требует длительных доказательств, выходит за рамки этих книг (конечно, за исключением действительно фундаментальных тем).

Итак, понятно, что вся область комбинаторных алгоритмов огромна, и охватить ее всю не в моих силах. Какие же темы из тех, которые я опускаю, наиболее важные? Я думаю, мое наибольшее “слепое пятно” — геометрия, потому что я всегда лучше чувствовал себя при работе с визуализацией и алгебраическими формулами, чем с объектами в пространстве. Поэтому я не пытаюсь заниматься в этих книгах комбинаторными задачами, связанными с вычислительной геометрией, такими как плотная упаковка сфер, кластеризация точек данных в n -мерном евклидовом пространстве или деревьями Штайнера на плоскости. Еще более существенно то, что я уклоняюсь от такой темы, как полиэдрическая комбинаторика, и от подходов, основанных, в первую очередь, на линейном программировании, целочисленном программировании или полуопределенном программировании. Эти вопросы рассматриваются во многих других книгах по данной тематике и рассчитаны на геометрическую интуицию читателя. Для моего понимания чисто комбинаторные подходы гораздо проще.

Должен также признаться в некотором предубеждении против алгоритмов, эффективных только в асимптотическом смысле, превосходная производительность которых начинает проявляться, только когда размер задачи превосходит размер вселенной. Алгоритмам такого рода посвящено множество современных публикаций. Я могу понять, почему медитация над беспредельными пределами столь интеллектуально привлекательна и так легко попадает в академическую печать. Однако в *Искусстве программирования* любые методы, которые я сам не испытал в реальных программах, ожидает быстрая расправа. (Конечно же, из этого правила есть исключения, главным образом касающиеся базовых концепций. Одни непрактичные методы просто слишком красивы и/или интеллектуальны, чтобы их можно

было исключить; другие представляют собой поучительные примеры того, как *не* следует поступать.)

Кроме того, как и в предыдущих томах данной серии, я преднамеренно практически полностью ограничиваюсь *последовательными* алгоритмами, несмотря на все возрастающую способность компьютеров к параллельным вычислениям. Я не могу судить, какие из идей, относящихся к параллельным вычислениям, будут полезны через пять или десять лет, не говоря уж о больших сроках, а потому оставляю эти вопросы другим, более сведущим авторам. Последовательные методы сами по себе уже являются испытанием пределов моей способности разглядеть, что искусные программисты захотят знать завтра.

Основные решения, которые мне нужно было принять при планировании представления данного материала, — как его лучше организовать: в соответствии с решаемыми задачами или в соответствии с применяемыми методами. Так, например, глава 5 в томе 3 посвящена единственной задаче — сортировке данных; к различным ее аспектам оказались применимыми более двух десятков разнообразных методов. Комбинаторные алгоритмы, напротив, включают множество различных задач, которые, как правило, решаются с помощью существенно меньшего набора методов. В конечном итоге я пришел к выводу, что смешанная стратегия будет работать лучше любого “чистого” подхода. Так, например, в разделе 7.3 рассматривается задача поиска кратчайших путей, а в разделе 7.4.1 — задача связности. При этом многие другие разделы посвящены базовым методам, таким как применение булевой алгебры (раздел 7.1), откат (раздел 7.2.2), теория матроидов (раздел 7.6) или динамическое программирование (раздел 7.7). Знаменитая задача коммивояжера и другие классические комбинаторные задачи, связанные с покрытием, раскраской и упаковкой, не имеют собственных разделов, но неоднократно появляются в разных местах, так как они могут решаться различными способами.

Я упомянул большой прогресс в области комбинаторных вычислений, но это совсем не означает, что все комбинаторные задачи действительно “приручены”. Когда время работы компьютерной программы взлетает не хуже баллистической ракеты, ее разработчикам не следует рассчитывать на то, что в этой книге найдется “серебряная пуля”, некий универсальный рецепт для решения их проблемы. Описанные здесь методы зачастую будут работать намного быстрее, чем подходы, которые пытались применить эти программисты. Но давайте посмотрим правде в глаза: комбинаторные задачи очень быстро становятся невообразимо огромными. Можно даже строго доказать, что некоторая небольшая, естественная задача *никогда* не будет иметь практического решения, хотя принципиально она разрешима (см. теорему Стокмейера и Мейера в разделе 7.1.2). В других случаях мы знаем, что подходящий алгоритм для решения данной задачи вряд ли существует, хотя и не можем этого доказать. Дело в том, что любой такой эффективный алгоритм позволит решить тысячи других задач, которые давно ставят в тупик крупнейших специалистов в мире (см. описание NP-полноты в разделе 7.9).

Опыт показывает, что изобретение новых комбинаторных алгоритмов будет продолжаться по-прежнему как для новых комбинаторных задач, так и для вновь выявленных вариаций или частных случаев старых. Аппетит программистов к таким алгоритмам со временем будет только расти. Когда программисты сталкиваются с задачами такого рода (а это происходит постоянно), это дает толчок искусству

программирования для достижения новых высот. Однако, скорее всего, сегодняшние методы также останутся в строю.

Большая часть данной книги самодостаточна, хотя и содержит частые ссылки на темы в предыдущих томах. В них достаточно широко рассматривались детали низкоуровневого программирования на машинном языке, так что в данной книге алгоритмы обычно описываются на абстрактном уровне, не зависящем от конкретной машины. Однако некоторые аспекты комбинаторного программирования существенно зависят от низкоуровневых деталей, которые ранее не рассматривались. В таких случаях все примеры в данной книге основаны на компьютере MMIX, который заменяет машину MIX, описанную в ранних изданиях тома 1. Подробная информация о MMIX приведена в дополнении к этому тому, выпущенному в мягкой обложке под названием *The Art of Computer Programming, Volume 1, Fascicle 1*,* в котором содержатся разделы 1.3.1', 1.3.2' и т. д. Эти разделы, а также соответствующие ассемблеры и симуляторы, доступны по сети Интернет.

Еще одним загружаемым ресурсом является коллекция программ и данных под названием *The Stanford GraphBase*, постоянно упоминаемая в примерах этой книги. Читателям при изучении комбинаторных алгоритмов настоятельно рекомендуется поработать с ней — мне представляется, что это наиболее эффективный и приятный способ изучения.

Кстати, при написании вводного материала в начале главы 7 мне было приятно отметить, что в ней самым естественным образом оказались упомянутыми некоторые работы моего научного руководителя Маршалла Холла-мл. (Marshall Hall, Jr.) (1910–1990), так же как и некоторые работы *его* научного руководителя Ойстейна Оре (Oystein Ore) (1899–1968) и некоторые работы *его* научного руководителя Торальфа Сколема (Thoralf Skolem) (1887–1963). Научный руководитель Сколема, Аксель Тью (Axel Thue) (1863–1922), уже упомянут в главе 6.

Я чрезвычайно благодарен сотням читателей, которые помогли мне выявить многочисленные ошибки, допущенные в черновых вариантах этого тома, которые изначально были доступны через Интернет, а впоследствии напечатаны в виде брошюр в мягких обложках. В особенности следует отметить комментарии Торстена Далхаймера (Thorsten Dahlheimer), Марка ван Льювена (Marc van Leeuwen) и Удо Вермута (Udo Wermuth). Но, боюсь, в материале скрываются и другие ошибки, которые я хотел бы исправить как можно скорее. Поэтому я с удовольствием заплачу 2 доллара 56 центов тому, кто первым сообщит мне о любой замеченной типографской*, технической или исторической ошибке. На веб-сайте по адресу <http://www-cs-faculty.stanford.edu/~knuth/taocp.html> можно найти текущий список всех поправок к данной книге.

Станфорд, Калифорния
Октябрь 2010

Д. Е. К.

* Имеется русский перевод: Кнут Д. Э. *Искусство программирования*, том 1, выпуск 1. MMIX — RISC-компьютер для нового тысячелетия. — М.: ООО “И. Д. Вильямс”, 2007. — *Примеч. пер.*

** Конечно же, имеется в виду оригинальное издание. — *Примеч. пер.*

В предисловии к первому изданию я просил читателей не обращать внимания на ошибки. Теперь я не хочу этого делать и благодарен тем читателям, которые не обратили внимания на мою просьбу.

— СТЮАРТ САЗЕРЛЕНД (STUART SUTHERLAND),
The International Dictionary of Psychology (1996)

Естественно, я несу ответственность за все оставшиеся ошибки — хотя, как мне кажется, мои друзья могли бы выловить и больше.

— ХРИСТОС Х. ПАПАДИМИТРИУ (CHRISTOS H. PAPADIMITRIOU),
Computational Complexity (1994)

Мне нравится работать в разных областях, потому что так мои ошибки размазываются более тонким слоем.

— ВИКТОР КЛИ (VICTOR KLEE) (1999)

Примечание к ссылкам. Несколько часто цитируемых журналов и трудов конференций имеют специальные коды, которые встречаются в предметно-именном указателе в конце этой книги. Различные серии *IEEE Transactions* цитируются с использованием буквенного кода, указывающего серию и приводимого полужирным шрифтом перед номером тома. Так, ‘**IEEE Trans. C-35**’ означает *IEEE Transactions on Computers*, том 35. IEEE больше не использует эти столь удобные буквенные коды, но их нетрудно расшифровать: ‘**EC**’ означает “Electronic Computers”, ‘**IT**’ — “Information Theory”, ‘**SE**’ — “Software Engineering”, ‘**SP**’ — “Signal Processing” и т. д.; ‘**CAD**’ означает “Computer-Aided Design of Integrated Circuits and Systems”.

Перекрестная ссылка вида ‘упр. 7.10–00’ указывает на будущее упражнение в разделе 7.10, номер которого пока что неизвестен.

Примечание к обозначениям. Простые и интуитивно понятные соглашения и обозначения для алгебраического представления математических концепций всегда были благом для прогресса, в особенности когда большинство исследователей в мире стали использовать единый символичный язык. Текущее состояние дел в комбинаторной математике, к сожалению, в этом отношении находится в беспорядке, так как одни и те же символы различными группами исследователей могут использоваться для совершенно разных целей. Некоторые специалисты, работающие в сравнительно узких областях, непреднамеренно создали противоречивую символику. Информатике — которая по своей природе очень широко взаимодействует с математикой — следует избегать этой опасности, принимая для себя, насколько это возможно, непротиворечивые обозначения. Поэтому мне часто приходилось делать выбор среди конкурирующих систем обозначений, при этом отдавая себе отчет в том, что угодить всем будет невозможно. Я изо всех сил пытался сделать все, чтобы придумать обозначения, которые, как мне кажется, будут наилучшими для будущего применения, зачастую после многих лет экспериментов и обсуждений с коллегами. Не редкостью были и метания между альтернативными вариантами, пока не удавалось остановиться на одном из них. Обычно все же удавалось найти

удобные обозначения, еще не используемые другими учеными способом, противоречащим моему.

Приложение Б представляет собой всеобъемлющий предметный указатель для всех основных обозначений, которые использованы в настоящей книге, включая и те, которые (пока что?) не являются стандартными. Если вы столкнетесь в книге с формулой, которая выглядит для вас странно и/или непонятно, достаточно высоки шансы, что приложение Б направит вас к странице, поясняющей мои намерения. Однако я хотел бы сделать несколько пояснений прямо сейчас, чтобы вы могли ознакомиться с ними еще до первого прочтения книги.

- Шестнадцатеричные константы предваряются знаком фунта ($\#$), т. е., например, $\#123$ означает $(123)_{16}$.
- Обозначение $x \dot{-} y$ применяется для операции “монус”, иногда именуемой “минус с точкой” или вычитанием с насыщением, результатом которой является $\max(0, x - y)$.
- Медиана трех чисел $\{x, y, z\}$ обозначается как $\langle xyz \rangle$.
- Множество, такое как $\{x\}$, состоящее из одного элемента, часто обозначается просто как x в контекстах наподобие $X \cup x$ или $X \setminus x$.
- Если n — неотрицательное целое число, количество единичных битов в бинарном представлении n равно νn . Кроме того, если $n > 0$, то крайний слева и крайний справа единичные биты n представляют собой соответственно $2^{\lambda n}$ и $2^{\rho n}$. Например, $\nu 10 = 2$, $\lambda 10 = 3$, $\rho 10 = 1$.
- Декартово произведение графов G и H обозначается как $G \square H$. Например, $C_m \square C_n$ представляет собой тор $m \times n$, так как C_n является циклом из n вершин.

ПРИМЕЧАНИЯ К УПРАЖНЕНИЯМ

УПРАЖНЕНИЯ, приведенные в этом многотомнике, предназначены как для самостоятельной проработки, так и для семинарских занятий. Очень трудно и, наверное, просто невозможно выучить предмет, только читая теорию и не применяя ее для решения конкретных задач, которые заставляют задуматься о прочитанном. Более того, мы лучше всего заучиваем то, до чего дошли самостоятельно, своим умом. Поэтому упражнения занимают важное место в данном издании. Я приложил немало усилий, чтобы сделать их как можно более информативными, а также отобрать задачи, которые были бы не только поучительными, но и позволяли бы читателю получить удовольствие от их решения.

Во многих книгах простые упражнения даются вперемешку с исключительно сложными. Это не всегда удобно, так как читателю хочется знать заранее, сколько времени ему придется затратить на решение задач (иначе в лучшем случае он их только просмотрит). В качестве классического примера подобной ситуации можно привести книгу Ричарда Беллмана (Richard Bellman) *Динамическое программирование* (М.: Изд-во иностр. лит., 1960). Это очень важная, новаторская работа, но у нее есть один недостаток: в конце некоторых глав в разделе “Упражнения и научные проблемы” среди серьезных, еще нерешенных проблем, попадаются простейшие вопросы. Говорят, что кто-то однажды спросил д-ра Беллмана, как отличить упражнения от научных проблем, и он ответил: “Если вы можете решить задачу, значит, это упражнение; в противном случае это научная проблема”.

Совершенно очевидно, что в книге, подобной этой, должны быть приведены и сложные научные проблемы, и простейшие упражнения. Поэтому, чтобы читатель не ломал голову, пытаясь отличить одно от другого, были введены *рейтинги*, которые определяют степень сложности каждого упражнения. Эти рейтинги имеют следующее значение.

Рейтинг Объяснение

- 00 Чрезвычайно простое упражнение, которое можно выполнить сразу же, если прочитанный материал понят. Упражнения подобного типа почти всегда можно решить “в уме”.
- 10 Простая задача, которая заставляет задуматься над прочитанным, но не представляет особых трудностей. На ее решение вы затратите не больше минуты; в процессе решения могут понадобиться карандаш и бумага.
- 20 Средняя задача, которая позволяет проверить, понял ли читатель основные положения изложенного материала. Чтобы получить исчерпывающий ответ, может понадобиться примерно 15–20 минут.
- 30 Задача умеренной сложности. Для ее решения может понадобиться более двух часов (а если при этом включен телевизор, то еще больше).

- 40 Достаточно сложная или трудоемкая задача, которую вполне можно включить в план семинарских занятий. Предполагается, что студент должен справиться с ней, затратив не слишком много времени, и решение будет нетривиальным.
- 50 Научная проблема, которая (насколько известно автору в момент написания книги) пока еще не получила удовлетворительного решения, хотя найти его пытались очень многие. Если вы нашли решение подобной проблемы, то опубликуйте его; более того, автор данной книги будет очень признателен, если ему сообщат решение как можно скорее (при условии, что оно правильно).

Интерполируя по этой “логарифмической” шкале, можно понять, что означает любой промежуточный рейтинг. Например, рейтинг 17 говорит о том, что упражнение немного проще, чем задача средней сложности. Если задача с рейтингом 50 будет впоследствии решена каким-либо читателем, то в следующих изданиях данной книги и в списке ошибок, опубликованных в Интернете, она может иметь рейтинг 40.

Остаток от деления рейтинга на 5 показывает, какой объем рутинной работы потребуется для решения данной задачи. Таким образом, для выполнения упражнения с рейтингом 24 может потребоваться больше времени, чем для упражнения с рейтингом 25, но для последнего необходим более творческий подход.

Автор очень старался правильно присвоить рейтинги упражнениям, но тому, кто составляет задачи, трудно предвидеть, насколько сложными они окажутся для кого-то другого. К тому же одному человеку некая задача может показаться простой, а другому — сложной. Таким образом, определение рейтингов — дело достаточно субъективное и относительное. Я надеюсь, что рейтинги помогут вам получить правильное представление о степени трудности задач, но их следует воспринимать в качестве ориентира, а не в качестве абсолюта.

Эта книга написана для читателей с различным уровнем математической подготовки и научного кругозора, поэтому некоторые упражнения рассчитаны исключительно на тех, кто серьезно интересуется математикой или занимается ею профессионально. Если рейтингу предшествует буква *M*, значит, математические понятия и обоснования используются в упражнении в большей степени, чем это необходимо тому, кто интересуется в основном программированием алгоритмов. Если же упражнение отмечено буквами *HM*, то для его решения необходимо знание высшей математики в большем объеме, чем дается в настоящей книге. Но пометка *HM* совсем *необязательно* означает, что упражнение трудное.

Перед некоторыми упражнениями стоит стрелка “►”, которая означает, что они особенно поучительны и их очень рекомендуется выполнить. Само собой разумеется, никто не ожидает, что читатель (или студент) будет решать *все* задачи, поэтому наиболее важные из них и были выделены. Но это ни в коем случае не означает, что другие упражнения выполнять не стоит! Каждый читатель должен хотя бы попытаться решить все задачи, рейтинг которых меньше или равен 10. Стрелки помогут выбрать задачи с более высокими рейтингами, которые следует решать в первую очередь.

В некоторых разделах имеется более сотни упражнений. Как же найти свой путь среди такого множества? В целом последовательность упражнений стремится

следовать последовательности идей в основном тексте. Идущие подряд упражнения могут основываться одно на другом, как в новаторских задачниках По́йя (Pólya) и Сегё (Szegő). В последних упражнениях разделов часто охватывается материал всего раздела или вводятся дополнительные темы.

К большинству упражнений приведены ответы, помещенные в отдельном разделе в конце книги. Пожалуйста, пользуйтесь ими разумно: ответ смотрите только после того, как приложите все усилия, чтобы решить задачу самостоятельно, либо если у вас совершенно нет времени на ее решение. Ответ будет поучителен и полезен для вас только в том случае, если вы ознакомитесь с ним *после* того, как найдете свое решение или изрядно потрудитесь над задачей. Ответы к задачам излагаются очень кратко и схематично, так как предполагается, что читатель честно пытался решить задачу собственными силами. Иногда в приведенном решении дается меньше информации, чем спрашивалось, но чаще бывает наоборот. Вполне возможно, что полученный вами ответ окажется лучше того, который помещен в книге, или вы найдете ошибку в ответе. В таком случае автор был бы очень признателен, если бы вы как можно скорее подробно сообщили ему об этом; тогда в последующих изданиях книги будет опубликовано более удачное решение, а также имя его автора.

Решая задачи, вы, как правило, можете пользоваться ответами к предыдущим упражнениям, за исключением случаев, когда это будет оговорено особо. Рейтинги упражнениям присваивались в расчете именно на это, и вполне возможно, что рейтинг упражнения $n + 1$ ниже рейтинга упражнения n , даже если результат упражнения n является его частным случаем.

Условные обозначения	00	Простейшее (ответ дать немедленно)
	10	Простое (на одну минуту)
► Рекомендуются	20	Средней трудности (на четверть часа)
<i>M</i> С математическим уклоном	30	Повышенной трудности
<i>HM</i> Требуется знания высшей математики	40	Высокой трудности
	50	Научная проблема

УПРАЖНЕНИЯ

- 1. [00] Что означает рейтинг *M20*?
- 2. [10] В чем ценность упражнений в данной книге для читателя?
- 3. [HM45] Докажите, что всякое односвязное компактное трехмерное многообразие гомотоморфно трехмерной сфере.

*Искусство является значительной составляющей такого
благотворного занятия, как противостояние предубеждениям.*

— ГЕНРИ ДЖЕЙМС (HENRY JAMES), *The Art of Fiction* (1884)

*Я признателен всем моим друзьям, но особая моя
благодарность тем из них, которые по прошествии
времени перестают терзать меня вопросом
“Какая книга будет следующей?”*

— ПИТЕР Д. ГОМЕС (PETER J. GOMES), *The Good Book* (1996)

*Наконец-то после долгих обещаний я представляю миру
мою работу. Однако я боюсь, что по отношению к ней
были поставлены слишком высокие ожидания. Причина
задержки ее публикации в значительной мере кроется
в чрезвычайном рвении, которое было проявлено
высокопочтенными особами со всех сторон в обеспечении
меня дополнительной информацией.*

— ДЖЕЙМС БОСВЕЛЛ (JAMES BOSWELL), *The Life of Samuel Johnson, LL. D.* (1791)

*Автор особенно признателен издательству Addison-Wesley
за терпение, проявленное в течение десятилетия
ожидания этой книги после подписания контракта.*

— ФРЭНК ХАРАРИ (FRANK HARARY), *Graph Theory* (1969)

*Средний парень, ненавидящий квадратные корни или алгебру, приходит
в восторг от головоломок, построенных на тех же принципах, и может
таким способом развить свои математические и изобретательские
шишки, которые приведут в изумление семейного френолога*.*

— СЭМ ЛОЙД (SAM LOYD), *The World of Puzzledom* (1896)

Пожалуйста, еще немного!

— Слоган Bitburger Brauerei** (1951)

* Френология — псевдонаука о связи психики человека и строения поверхности его черепа. Создателем френологии является австрийский врач и анатом Ф. Й. Галль (F. J. Gall), который утверждал, что все психические свойства, якобы локализующиеся в полушариях мозга, при развитии вызывают разрастание определенного участка мозга, а это, в свою очередь, — образование выпуклости (“шишки”) на соответствующем участке черепа. — *Примеч. пер.*

** Одно из крупнейших пивоваренных предприятий Германии. — *Примеч. пер.*



Hommage à Bach.

КОМБИНАТОРНЫЙ ПОИСК

...Чтобы их найти, надо искать весь день,
а найдешь — увидишь, что и искать не стоило.*

— БАССАНИО (BASSANIO), в *Венецианский купец* (акт I, сцена 1)

*Любые действия или противодействия каждой капли этого людского моря
могут сложиться в самые непредсказуемые и необъяснимые комбинации.*

*Иногда происходят поразительные, невообразимые события...***

— ШЕРЛОК ХОЛМС (SHERLOCK HOLMES),
в *The Adventure of the Blue Carbuncle* (1892)

*Тема комбинаторных алгоритмов слишком обширна,
чтобы охватить ее в одной статье или даже в одной книге.*

— РОБЕРТ Э. ТАРЖАН (ROBERT E. TARJAN) (1976)

*Постоянно сталкиваясь с массой людей, я был впечатлен
тем фактом, что наиболее успешными среди них были те,
кто обладал врожденным даром разгадывать головоломки.*

*Жизнь полна загадок, которые мы должны разгадывать по
мере того, как судьба подбрасывает их нам.*

— УОЛТЕР ЛОЙД (СЭМ ЛОЙД-МЛ.) (SAM LOYD, JR.) (1926)

КОМБИНАТОРИКА изучает способы, которыми дискретные объекты могут быть упорядочены в шаблоны различного вида. Например, объекты могут представлять собой $2n$ чисел $\{1, 1, 2, 2, \dots, n, n\}$, и мы хотим разместить их в строку таким образом, чтобы между двумя появлениями каждого числа k находилось ровно k других чисел. Когда $n = 3$, имеется по сути единственный способ размещения таких “пар Лэнгфорда”, а именно 231213 (и его зеркальное отображение 312132); аналогично имеется единственное решение и для $n = 4$. Многие другие типы комбинаторных шаблонов будут рассматриваться далее в этой книге.

Обычно при изучении комбинаторных задач возникает пять основных типов вопросов, одни — потруднее, другие — попроще.

i) *Существование*: существует ли упорядочение X , отвечающее данному шаблону?

ii) *Построение*: если существует, то можно ли быстро найти такое X ?

* Перевод Т. Щепкиной-Куперник.

** Перевод В. Михалюка.

- iii) *Подсчет*: сколько имеется различных упорядочений X ?
- iv) *Генерация*: могут ли все упорядочения $X_1, X_2, \dots$ быть посещены систематическим образом?
- v) *Оптимизация*: какие упорядочения максимизируют или минимизируют $f(X)$ для заданной целевой функции f ?

Каждый из приведенных вопросов оказывается достаточно интересным по отношению к парам Лэнгфорда.

Например, рассмотрим вопрос существования. Методом проб и ошибок достаточно быстро выясняется, что, когда $n = 5$, разместить $\{1, 1, 2, 2, \dots, 5, 5\}$ требуемым образом в десяти позициях невозможно. Обе единицы должны находиться либо в четных позициях, либо в нечетных. Точно так же и тройки и пятерки должны находиться парами либо в четных, либо в нечетных позициях; двойки и четверки используют по одной четной и одной нечетной позиции. Таким образом, мы не можем заполнить по пяти позиций каждой четности. Такие же рассуждения доказывают, что задача не имеет решения и при $n = 6$ или, в общем случае, когда количество нечетных значений в множестве $\{1, 2, \dots, n\}$ нечетно.

Другими словами, решение задачи Лэнгфорда может существовать только для $n = 4m - 1$ или $n = 4m$, для некоторого целого m . И обратно, когда n имеет указанный вид, существует элегантный способ построения требуемого размещения, найденный Роем О. Дэйвисом (Roy O. Davies) и приведенный в упр. 1.

Сколько же имеется существенно разных решений задачи L_n ? С ростом n — много, очень много:

$$\begin{array}{ll}
 L_3 = 1; & L_4 = 1; \\
 L_7 = 26; & L_8 = 150; \\
 L_{11} = 17\,792; & L_{12} = 108\,144; \\
 L_{15} = 39\,809\,640; & L_{16} = 326\,721\,800; \\
 L_{19} = 256\,814\,891\,280; & L_{20} = 2\,636\,337\,861\,200; \\
 L_{23} = 3\,799\,455\,942\,515\,488; & L_{24} = 46\,845\,158\,056\,515\,936.
 \end{array} \tag{1}$$

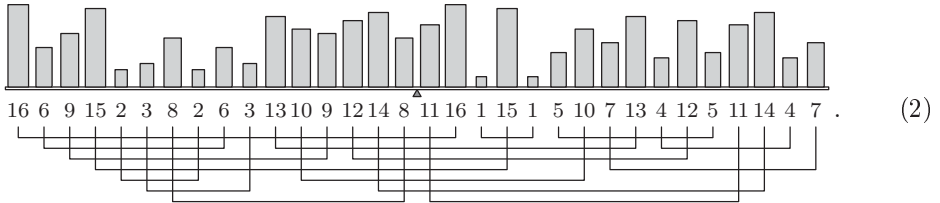
[Значения L_{23} и L_{24} были найдены в 2004 и 2005 годах М. Краецки (M. Krajecki), К. Джейллетом (C. Jaillet) и А. Бью (A. Bui); см. *Studia Informatica Universalis* 4 (2005), 151–190.] Наброски “на коленке” дают грубую оценку значения L_n (когда оно не равно нулю), равную $(4n/e^3)^{n+1/2}$ (см. упр. 5); и эта оценка оказывается в целом корректной для всех известных значений. Но простой формулы для этих значений, похоже, не существует.

Задача Лэнгфорда представляет собой простой частный случай общего класса комбинаторных задач, называющихся *задачами точного покрытия*. В разделе 7.2.2.1 мы познакомимся с так называемым алгоритмом “танцующих связей”, который представляет собой удобное средство генерации всех решений таких задач. Например, при $n = 16$ этот метод требует выполнения всего лишь около 3200 обращений к памяти для каждого найденного решения задачи Лэнгфорда. Таким образом, значение L_{16} может быть вычислено за разумное время путем простой генерации всех решений и их подсчета.

Заметим, однако, что число L_{24} *огромно* — огрубленно оно равно 5×10^{16} , или около 1500 МИР-лет. (Вспомним, что “МИР-год” представляет собой количество

инструкций, которые за год может выполнить машина, выполняющая миллион инструкций в секунду, а именно 31 556 952 000 000.) Следовательно, очевидно, что точное значение L_{24} было определено с помощью иной методики, *не* включающей генерацию всех расстановок. Действительно, имеется гораздо, гораздо более быстрый способ вычисления L_n с помощью полиномиальной алгебры. Поучительный метод, описанный в упр. 6, требует $O(4^n n)$ операций, что может показаться неэффективным; но это лучше метода генерации и подсчета на громадный множитель порядка $\Theta((n/e^3)^{n-1/2})$, так что даже при $n = 16$ этот метод работает примерно в 20 раз быстрее. С другой стороны, точное значение L_{100} , вероятно, никогда не станет известным, несмотря на все большую и большую скорость компьютеров.

Можно также рассмотреть решения задачи Лэнгфорда, *оптимальные* в том или ином отношении. Для примера можно расположить шестнадцать пар равновесов $\{1, 1, 2, 2, \dots, 16, 16\}$ так, чтобы они удовлетворяли условиям Лэнгфорда и обладали дополнительным свойством “сбалансированности”, в том смысле, что, будучи расставленными в соответствующем порядке, они оставят в равновесии рычажные весы:



Другими словами, $15.5 \cdot 16 + 14.5 \cdot 6 + \dots + 0.5 \cdot 8 = 0.5 \cdot 11 + \dots + 14.5 \cdot 4 + 15.5 \cdot 7$; и в данном конкретном примере мы сталкиваемся с еще одним условием равновесия, $16 + 6 + \dots + 8 = 11 + 16 + \dots + 7$. Следовательно, справедливо также $16 \cdot 16 + 15 \cdot 6 + \dots + 1 \cdot 8 = 1 \cdot 11 + \dots + 15 \cdot 4 + 16 \cdot 7$.

Кроме того, размещение (2) имеет *минимальную ширину* среди всех решений задачи Лэнгфорда порядка 16: соединяющие линии под диаграммой показывают, что в любой точке диаграммы одновременно незавершенными оказываются не более семи пар; можно также показать, что ширина, равная шести, в данном случае недостижима (см. упр. 7).

Какое размещение $a_1 a_2 \dots a_{32}$ чисел $\{1, 1, \dots, 16, 16\}$ оказывается *наименее* сбалансированным, в том смысле, что сумма $\sum_{k=1}^{32} k a_k$ максимальна? Оказывается, что максимально возможное значение равно 5268. Одним из таких решений (а всего их 12 016) является

$$2 \ 3 \ 4 \ 2 \ 1 \ 3 \ 1 \ 4 \ 16 \ 13 \ 15 \ 5 \ 14 \ 7 \ 9 \ 6 \ 11 \ 5 \ 12 \ 10 \ 8 \ 7 \ 6 \ 13 \ 9 \ 16 \ 15 \ 14 \ 11 \ 8 \ 10 \ 12. \quad (3)$$

Более интересный вопрос — о решениях задачи Лэнгфорда, являющихся минимальными и максимальными в лексикографическом порядке. Ответами для $n = 24$ являются

$$\{abacbdcefcgdoersfpgqtuwvwjklonhmirpsjqkhltiunmwvx, \\ xvwsquntkigrdapaodgiknqsvxwutmrpohljcfbecbhmfj1\}, \quad (4)$$

если использовать буквы a, b, ..., w, x вместо чисел 1, 2, ..., 23, 24.

Мы рассмотрим множество методов комбинаторной оптимизации в следующих разделах этой главы. Наша цель, конечно же, заключается в том, чтобы решать



Рис. 1. Беспорядок в карточном королевстве: никакого совпадения строк. (Это всего лишь одно из множества решений популярной головоломки XVIII века.)

такие задачи с помощью изучения только небольшой части пространства всех возможных размещений.

Ортогональные латинские квадраты. Давайте ненадолго вернемся в ранние дни комбинаторики. Посмертное издание книги Жака Озанама (Jacques Ozanam) *Recreations mathematiques et physiques* (Paris: 1725) включает занимательную головоломку на с. 434 тома 4: “Возьмите всех тузов, королей, дам и валетов из обычной колоды игральных карт и расположите их в виде квадрата так, чтобы в каждой строке и каждом столбце содержались карты всех мастей и всех достоинств.” Сумеете ли вы сделать это? Решение Озанама, показанное на рис. 1, делает даже больше: полный набор достоинств и мастей встречается также на обеих диагоналях квадрата.

В 1779 году подобная головоломка, распространившаяся в Санкт-Петербурге, привлекла внимание великого математика Леонарда Эйлера. “Тридцать шесть офицеров шести различных званий, взятые из шести различных полков, хотят выстроиться квадратным строем 6×6 так, чтобы в каждой строке и каждом столбце этого квадрата имелись офицеры всех званий и полков. Как они должны это сделать?” Никто не мог найти удовлетворительного решения этой задачи, так что Эйлер решил взяться за эту проблему (несмотря на то, что в 1771 году он практически ослеп и диктовал все свои работы ассистентам). Он написал большую работу на эту тему [в конце концов опубликованную в *Verhandelingen uitgegeven door het Zeeuwsch Genootschap der Wetenschappen te Vlissingen* 9 (1782), 85–239], в которой привел решения подобной задачи для n званий и n полков для $n = 1, 3, 4, 5, 7, 8, 9, 11, 12, 13, 15, 16, \dots$; ему не поддались только случаи $n \bmod 4 = 2$.

Очевидно, что при $n = 2$ решения не существует. Но Эйлер был озадачен случаем $n = 6$, когда рассмотрел “очень значительное число” квадратных расста-

новок, которые не работали. Он показал, что каждое решение должно приводить ко многим другим решениям, выглядящим иными, и не мог поверить, что все такие решения сумели избежать его внимания. Поэтому он говорил: “Я не сомневаюсь в своем выводе о том, что найти полный квадрат из 36 ячеек невозможно, точно так же, как невозможно найти решения для $n = 10$, $n = 14 \dots$ в общем случае для всех нечетно-четных чисел”.

Эйлер дал 36 офицерам имена $a\alpha, a\beta, a\gamma, a\delta, a\epsilon, a\zeta, b\alpha, b\beta, b\gamma, b\delta, b\epsilon, b\zeta, c\alpha, c\beta, c\gamma, c\delta, c\epsilon, c\zeta, d\alpha, d\beta, d\gamma, d\delta, d\epsilon, d\zeta, e\alpha, e\beta, e\gamma, e\delta, e\epsilon, e\zeta, f\alpha, f\beta, f\gamma, f\delta, f\epsilon, f\zeta$, основываясь на их полках и званиях. Он заметил, что любое решение состоит из двух *отдельных* квадратов, одного — для латинских букв и второго — для греческих. Каждый из этих квадратов, как предполагается, содержит разные записи в строках и столбцах; так что Эйлер начал свое исследование с изучения возможных конфигураций для $\{a, b, c, d, e, f\}$, которые он назвал *латинскими квадратами*. Латинский квадрат может быть объединен в пару с греческим квадратом, образуя “греко-латинский квадрат” только в том случае, когда квадраты *ортогональны* друг к другу, что означает, что пары букв (латинская, греческая) не могут встретиться более одного раза при наложении квадратов. Например, если положить $a = A, b = K, c = Q, d = J, \alpha = \clubsuit, \beta = \spadesuit, \gamma = \diamondsuit$ и $\delta = \heartsuit$, то рис. 1 эквивалентен латинскому, греческому и греко-латинскому квадратам

$$\begin{pmatrix} d & a & b & c \\ c & b & a & d \\ a & d & c & b \\ b & c & d & a \end{pmatrix}, \begin{pmatrix} \gamma & \delta & \beta & \alpha \\ \beta & \alpha & \gamma & \delta \\ \alpha & \beta & \delta & \gamma \\ \delta & \gamma & \alpha & \beta \end{pmatrix} \text{ и } \begin{pmatrix} d\gamma & a\delta & b\beta & c\alpha \\ c\beta & b\alpha & a\gamma & d\delta \\ a\alpha & d\beta & c\delta & b\gamma \\ b\delta & c\gamma & d\alpha & a\beta \end{pmatrix}. \quad (5)$$

Конечно, можно воспользоваться *любыми* n различными символами в латинском квадрате размером $n \times n$; все, что имеет значение, — это чтобы ни один символ не встречался дважды в одной строке или в одном столбце. Так что мы вполне можем использовать для записей числовые значения $\{0, 1, \dots, n-1\}$. Кроме того, мы будем говорить просто о латинских квадратах, не подразделяя их на греческие или латинские, поскольку ортогональность — отношение симметричное.

Утверждение Эйлера о том, что два латинских квадрата размером 6×6 не могут быть ортогональны, было подтверждено Томасом Клаузенем (Thomas Clausen), который свел задачу к проверке 17 фундаментально различных случаев. Об этом в письме, датированном 10 августа 1842 года, К. Ф. Гауссу (C. F. Gauss) сообщил Г. К. Шумахер (H. C. Schumacher). Сам Клаузен свой анализ не публиковал. Первая появившаяся в печати демонстрация этого факта принадлежит Г. Тарри (G. Tarry) [*Comptes rendus, Association française pour l'avancement des sciences* **29**, part 2 (1901), 170–203], который, идя собственным путем, открыл, что латинские квадраты размером 6×6 могут быть разделены на 17 семейств. (В разделе 7.2.3 мы рассмотрим, как разложить задачу на комбинаторно неэквивалентные классы размещений.)

Гипотеза Эйлера о значениях $n = 10, n = 14, \dots$ была “доказана” трижды: Ю. Петерсеном (J. Petersen) [*Annuaire des mathématiciens* (Paris: 1902), 413–427], П. Вернике (P. Wernicke) [*Jahresbericht der Deutschen Math.-Vereinigung* **19** (1910), 264–267] и Г. Ф. Мак-Нейшем (H. F. MacNeish) [*Annals of Math.* (2) **23** (1922), 221–227]. Однако во всех трех доказательствах известны недостатки; так что вопрос оставался открытым до тех пор, пока много лет спустя не стали доступны быст-

родействующие компьютеры. Одной из первых комбинаторных задач, решенных с применением ЭВМ, стала загадка греко-латинского квадрата размером 10×10 : существует он или нет?

В 1957 году Л. Д. Пейдж (L. J. Paige) и Ч. Б. Томпкинс (C. B. Tompkins) запрограммировали ЭВМ SWAC для поиска контрпримера для предсказания Эйлера. Они выбирали один отдельный латинский квадрат 10×10 “почти случайно”, и их программа пыталась найти другой квадрат, ортогональный первому. Но результат оказался обескураживающим, и они решили прекратить вычисления через пять часов работы. За это время машина сгенерировала достаточно данных, чтобы можно было предсказать, что вся работа потребует как минимум 4.8×10^{11} часов машинного времени!

Вскоре после этого трем математикам удалось совершить прорыв, который вынес латинские квадраты на страницы одной из ведущих мировых газет (New York Times от 29 апреля 1959 года): Р. Ч. Боze (R. C. Bose), Ш. Ш. Шрикханде (S. S. Shrikhande) и Э. Т. Паркер (E. T. Parker) нашли замечательный ряд построений, которые дают ортогональные квадраты размером $n \times n$ для всех $n > 6$ [*Proc. Nat. Acad. Sci.* **45** (1959), 734–737, 859–862; *Canadian J. Math.* **12** (1960), 189–203]. Таким образом, после сопротивления атакам в течение 180 лет гипотеза Эйлера оказалась почти полностью ошибочной.

Их открытие было сделано без привлечения компьютеров. Но Паркер работал на UNIVAC, и его навыки программиста помогли получить решение задачи Пейджа и Томпкинса менее чем за час машинного времени ЭВМ UNIVAC 1206 Military Computer. [См. *Proc. Symp. Applied Math.* **10** (1960), 71–83; **15** (1963), 73–81.]

Давайте взглянем, как работали программисты тех времен и как Паркер превошел их. Пейдж и Томпкинс начали со следующего квадрата L размером 10×10 и его пока неизвестного ортогонального напарника M :

$$L = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 8 & 3 & 2 & 5 & 4 & 7 & 6 & 9 & 0 \\ 2 & 9 & 5 & 6 & 3 & 0 & 8 & 4 & 7 & 1 \\ 3 & 7 & 0 & 9 & 8 & 6 & 1 & 5 & 2 & 4 \\ 4 & 6 & 7 & 5 & 2 & 9 & 0 & 8 & 1 & 3 \\ 5 & 0 & 9 & 4 & 7 & 8 & 3 & 1 & 6 & 2 \\ 6 & 5 & 4 & 7 & 1 & 3 & 2 & 9 & 0 & 8 \\ 7 & 4 & 1 & 8 & 0 & 2 & 9 & 3 & 5 & 6 \\ 8 & 3 & 6 & 0 & 9 & 1 & 5 & 2 & 4 & 7 \\ 9 & 2 & 8 & 1 & 6 & 7 & 4 & 0 & 3 & 5 \end{pmatrix} \quad \text{и} \quad M = \begin{pmatrix} 0 & \square & \square & \square & \square & \square & \square & \square & \square & \square \\ 1 & \square & \square & \square & \square & \square & \square & \square & \square & \square \\ 2 & \square & \square & \square & \square & \square & \square & \square & \square & \square \\ 3 & \square & \square & \square & \square & \square & \square & \square & \square & \square \\ 4 & \square & \square & \square & \square & \square & \square & \square & \square & \square \\ 5 & \square & \square & \square & \square & \square & \square & \square & \square & \square \\ 6 & \square & \square & \square & \square & \square & \square & \square & \square & \square \\ 7 & \square & \square & \square & \square & \square & \square & \square & \square & \square \\ 8 & \square & \square & \square & \square & \square & \square & \square & \square & \square \\ 9 & \square & \square & \square & \square & \square & \square & \square & \square & \square \end{pmatrix}. \quad (6)$$

Без потери общности можно считать, что строки M начинаются, как показано, с 0, 1, ..., 9. Задача заключается в заполнении 90 остающихся пустыми записей, и исходная программа для SWAC работала сверху вниз и слева направо. Верхняя левая пустая запись $\square$ не может содержать 0, поскольку 0 уже имеется в верхней левой строке M . Это не может быть и 1, так как пара (1, 1) уже имеется слева в следующей строке квадрата (L, M) . Однако мы можем умозрительно вставить в это место 2. Следующей можно вставить цифру 1; и так постепенно мы найдем лексикографически наименьшую верхнюю строку, которая может быть в M , а именно 0214365897. Аналогично наименьшими строками, которые могут размещаться под 0214365897, являются 1023456789 и 2108537946; очередная наименьшая строка имеет вид 3540619278. К сожалению, после этого мы попадаем в переплет:

нет никакой возможности завершить очередную строку так, чтобы не вступить в конфликт с уже имеющимися. Поэтому заменим 3540619278 на 3540629178 (но и этот выбор не дает возможности продолжать работу), затем на 3540698172, и так еще несколько шагов, пока наконец не достигнем строки 3546109278, за которой может следовать строка 4397028651, после которой мы снова встретимся с неприятностями.

В разделе 7.2.3 мы изучим методы оценки поведения таких поисков без реального их проведения. Такие оценки говорят нам о том, что в данном случае метод Пейджа–Томпкинса, по сути, обходит неявное дерево поиска, содержащее около 2.5×10^{18} узлов. Большинство этих узлов принадлежит только нескольким уровням дерева; больше половины из них работают с выбором правой части шестой строки M , после того как около 50 из 90 пустых мест заполнены. Типичный узел дерева поиска требует для проверки, вероятно, около 75 обращений к памяти. Следовательно, общее время работы на современном компьютере можно оценить как время, необходимое для выполнения 2×10^{20} обращений к памяти.

Паркер же вернулся к методу, который Эйлер использовал для поиска ортогональных пар в 1779 году. Сначала он находил все так называемые *секущие* (*transversals*) матрицы L , а именно все способы выбора некоторых из ее элементов так, чтобы в каждой строке и каждом столбце имелся ровно один элемент, причем чтобы были использованы все возможные значения элементов. Например, одним из сечений в записи Эйлера является 0859734216, т. е. в столбце 0 мы выбираем 0, в столбце 1 — 8, ..., в столбце 9 — 6. Каждая секущая, включающая k в крайнем слева столбце L , представляет корректный способ размещения десяти k в квадрате M . Задача поиска секущих в действительности достаточно проста. Для данной матрицы L , оказывается, их имеется ровно 808; для $k = (0, 1, \dots, 9)$ имеется соответственно (79, 96, 76, 87, 70, 84, 83, 75, 95, 63) секущих.

После того как нам становятся известными секущие, остается решить задачу точного покрытия из десяти этапов, что существенно проще, чем исходная 90-этапная задача в (6). Все, что нам нужно, — это покрыть квадрат десятью непересекающимися секущими, поскольку каждое такое множество из десяти секущих эквивалентно латинскому квадрату M , ортогональному L .

Конкретный квадрат L из (6) в действительности имеет ровно одну ортогональную пару:

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 8 & 3 & 2 & 5 & 4 & 7 & 6 & 9 & 0 \\ 2 & 9 & 5 & 6 & 3 & 0 & 8 & 4 & 7 & 1 \\ 3 & 7 & 0 & 9 & 8 & 6 & 1 & 5 & 2 & 4 \\ 4 & 6 & 7 & 5 & 2 & 9 & 0 & 8 & 1 & 3 \\ 5 & 0 & 9 & 4 & 7 & 8 & 3 & 1 & 6 & 2 \\ 6 & 5 & 4 & 7 & 1 & 3 & 2 & 9 & 0 & 8 \\ 7 & 4 & 1 & 8 & 0 & 2 & 9 & 3 & 5 & 6 \\ 8 & 3 & 6 & 0 & 9 & 1 & 5 & 2 & 4 & 7 \\ 9 & 2 & 8 & 1 & 6 & 7 & 4 & 0 & 3 & 5 \end{pmatrix} \perp \begin{pmatrix} 0 & 2 & 8 & 5 & 9 & 4 & 7 & 3 & 6 & 1 \\ 1 & 7 & 4 & 9 & 3 & 6 & 5 & 0 & 2 & 8 \\ 2 & 5 & 6 & 4 & 8 & 7 & 0 & 1 & 9 & 3 \\ 3 & 6 & 9 & 0 & 4 & 5 & 8 & 2 & 1 & 7 \\ 4 & 8 & 1 & 7 & 5 & 3 & 6 & 9 & 0 & 2 \\ 5 & 1 & 7 & 8 & 0 & 2 & 9 & 4 & 3 & 6 \\ 6 & 9 & 0 & 2 & 7 & 1 & 3 & 8 & 4 & 5 \\ 7 & 3 & 5 & 1 & 2 & 0 & 4 & 6 & 8 & 9 \\ 8 & 0 & 2 & 3 & 6 & 9 & 1 & 7 & 5 & 4 \\ 9 & 4 & 3 & 6 & 1 & 8 & 2 & 5 & 7 & 0 \end{pmatrix}. \quad (7)$$

Алгоритм танцующих связей находит это решение и доказывает его единственность всего лишь примерно за 1.7×10^8 обращений к памяти для данных 808 секущих. Стоимость же фазы поиска секущих (около пяти миллионов обращений к памяти) пренебрежимо мала по сравнению с поиском матрицы. Таким образом, исходное

время работы, равное 2×10^{20} обращениям к памяти и считавшееся неизбежной платой за решение задачи с 10^{90} возможными способами заполнения пустых слотов, оказалось уменьшено более чем в $10^{12}(!)$ раз.

Позже мы увидим, что можно внести усовершенствования и в методы решения 90-уровневой задачи наподобие (6). Действительно, оказывается, (6) непосредственно представимо в качестве задачи точного покрытия (см. упр. 17), которая решается процедурой танцующих связей из раздела 7.2.2.1 ценой всего лишь 1.3×10^{11} обращений к памяти. Но даже в этом случае подход Эйлера–Паркера оказывается в тысячу раз лучше подхода Пейджа–Томпкинса. “Разложив” задачу на две отдельные фазы, одну для поиска секущих и вторую — для их объединения, Эйлер и Паркер, по сути, уменьшили стоимость вычислений с произведения $T_1 T_2$ до суммы $T_1 + T_2$.

Мораль этой истории очевидна: комбинаторные задачи могут поставить нас перед лицом огромной совокупности возможностей, но мы не должны сразу же опускать руки. Одна хорошая идея может на много порядков сократить количество требующихся вычислений.

Головоломки и реальный мир. Многие из комбинаторных задач, которые мы будем рассматривать в этой главе, наподобие задач Лэнгфорда о парах или Озанама о шестнадцати картах, изначально представляли собой не более чем головоломки. Некоторых особо серьезно настроенных читателей такая легкомысленность некоторых задач может попросту отпугнуть. В самом деле, стоит ли тратить попусту машинное время, неужели ему нет более полезного применения? И не должна ли книга, посвященная компьютерам и программированию, в первую очередь озабочиться важными промышленными и научными приложениями, определяющими мировой прогресс?

Начнем с того, что автор данной книги совершенно не озадачивался ее полезностью для прогресса человечества. Но он твердо убежден, что такая книга, как эта, должна обратить особое внимание на *методы* решения задач, а также на математические идеи и *модели*, которые помогут решать многие различные проблемы, а не на причины, по которым такие методы и модели могут быть полезными. Мы изучим множество красивых и эффективных способов решения комбинаторных задач, и главной причиной их изучения будет элегантность этих методов. Комбинаторные задачи встречаются повсюду, и ежедневно появляются новые способы применения описываемых в этой главе методик. Так что давайте не будем заранее ограничивать сами себя, пытаясь заранее классифицировать, для чего годятся те или иные идеи.

Например, оказывается, что ортогональные латинские квадраты чрезвычайно полезны, в особенности при планировании экспериментов. Уже в 1788 году Франсуа Кретте де Паллюль (Francois Cretté de Palluel) использовал латинский квадрат размером 4×4 для изучения, что будет с шестнадцатью овцами четырех различных пород при применении четырех различных кормов и четырех разных периодов стрижки. [*Mémoires d'Agriculture* (Paris: Société Royale d'Agriculture, trimestre d'été, 1788), 17–23.] Латинский квадрат позволил ему ограничиться 16 овцами вместо 64; при использовании греко-латинского квадрата он мог бы варьировать еще один параметр, например количество корма или вид пастбища.

Если бы мы сосредоточились на его воззрениях на животноводство, то могли бы увязнуть в деталях разведения овец, сравнении корнеплодов с зерновыми

и расходами на их выращивание и т. д. Читатели, не интересующиеся сельским хозяйством, могли бы пропустить весь материал, несмотря на то, что латинские квадраты применимы к широкому спектру исследований (представьте, например, проверку действия лекарства на пациентов на пяти стадиях некоторого заболевания, пяти возрастных и пяти весовых группах). Кроме того, концентрация на опытно-исследовательском применении латинских квадратов может привести к тому, что читатели упустят тот факт, что латинские квадраты имеют важные приложения в дискретной геометрии и кодах с исправлением ошибок (см. упр. 18–24).

Даже задача Лэнгфорда, на первый взгляд кажущаяся чистым развлечением, также имеет практическую ценность. Т. Сколем (T. Skolem) использовал последовательности Лэнгфорда для построения Штейнеровской системы троек, которую мы применяли для запросов к базам данных в разделе 6.5 [см. *Math. Scandinavica* 6 (1958), 273–280]; а в 1960-х годах Э. Д. Грот (E. J. Groth) из Motorola Corporation применил пары Лэнгфорда в разработке электронных схем для умножения. Кроме того, алгоритмы, которые эффективно решают задачу Лэнгфорда и текущих латинских квадратов, такие как метод танцующих связей, применимы к решению задач точного покрытия в общем случае; а задача точного покрытия имеет большое значение для таких важнейших проблем, как, например, справедливое распределение участков для избирательных округов и т. п.

Ни приложения, ни головоломки не являются наиболее важными вещами. Наша главная цель — внедрить в свои головы базовые концепции наподобие понятий латинских квадратов или точного покрытия. Эти концепции обеспечат нас строительными блоками, словарем и пониманием, необходимыми для решения *застраших* задач.

Глупо обсуждать решение задач без практического решения каких бы то ни было задач. Для стимуляции нашего мышления и воображения, упорядочения серых клеточек и знакомства с базовыми концепциями нужны подходящие задачи. И для этой цели часто лучше всего подходят именно головоломки, поскольку их можно изложить буквально несколькими словами и они не требуют сложных базовых знаний.

Жизнь — слишком сложная вещь, чтобы относиться к ней, как к очередной головоломке, которую следует решить и которую можно решить, просто снабдив машину соответствующими инструкциями и данными. Но, к счастью, есть загадки, которые *могут* быть решены! Головоломки стоят того, чтобы причислить их к наибольшим удовольствиям жизни, которыми, как и любыми другими удовольствиями, следует пользоваться умеренно и без злоупотреблений.

Конечно, Лэнгфорд и Озанам предназначали свои головоломки для людей, а не для компьютеров. Разве смысл не будет потерян, если мы просто передадим задачу машинам, которые будут решать ее методом грубой силы, а не с помощью искусного применения рационального мышления? Джордж Брюстер (George Brewster) в письме Мартину Гарднеру (Martin Gardner) в 1963 году выразил широко распространенное мнение следующим образом: “Скармливать головоломки для отдыха компьютерам — немногим лучше, чем глушить форель в горной реке динамитом. Оставьте человеку возможность расслабиться”.

Хотя с этим трудно не согласиться, указанная точка зрения упускает один важный момент: простые головоломки часто имеют обобщения, которые выходят за

рамки человеческих возможностей и вызывают наше любопытство. Изучение этих обобщений часто предполагает наличие поучительных методов, которые могут применяться к ряду других проблем и иметь неожиданные последствия. Действительно, многие из ключевых методов, которые мы будем изучать, родились в попытках решения различных головоломок. При написании этой главы автор не мог не наслаждаться тем фактом, что теперь, когда компьютеры становятся все более и более быстрыми, головоломки доставляют еще большее удовольствие, ведь теперь в наших руках оказывается еще более мощный динамит, с которым можно вволю поиграться. [Более подробно точка зрения автора изложена в его эссе “Полезны ли задачи-головоломки?”, написанном еще в 1976 году; см. *Selected Papers on Computer Science* (1996), 169–183.]

Опасность головоломок в том, что они могут оказаться *слишком* элегантными. Хорошие головоломки, как правило, математически чисты и хорошо структурированы, но нам надо научиться систематически работать с тем грязным, хаотическим материалом, который ежедневно нас окружает. Фактически некоторые вычислительные методы важны, в первую очередь, потому, что они предоставляют мощные способы преодоления этих сложностей. Вот почему, например, в начале главы 5 рассказывалось о загадочных правилах упорядочения библиотечных каталогов, а в разделе 2.2.5 моделировалась работа реального лифта.

Чтобы в экспериментах с комбинаторными алгоритмами можно было воспользоваться данными из реального мира, подготовлена коллекция программ и данных под названием Stanford GraphBase (SGB). Она включает, например, данные об американских автострадах и модель “затраты–выпуск” американской экономики; в ней описано распределение персонажей в *Илиаде* Гомера, *Анне Карениной* Льва Толстого и ряде других романов; здесь можно найти структуру *Thesaurus* 1879 года Роджета (Roget) и результаты сотен футбольных матчей разных колледжей; в ней есть даже значения пикселей черно-белого варианта *Джоконды* (Моны Лизы) Леонардо да Винчи. Но, пожалуй, самое важное то, что SGB содержит набор пятибуквенных английских слов, которые мы и рассмотрим следующими.

Пятибуквенные английские слова. Множество примеров в этой главе будут основываться на списке пятибуквенных слов

aargh, abaca, abaci, aback, abaft, abase, abash, ..., zooms, zowie. (8)

(Всего в нем содержится 5757 слов—слишком много, чтобы перечислить здесь их все; но вы легко можете представить себе отсутствующие в (8) слова.) Это список, лично составленный автором книги между 1972 и 1992 годами, начиная с того времени, когда он понял, что такие слова представляют собой идеальные (ideal) данные для тестирования комбинаторных алгоритмов различных видов.

Этот список преднамеренно ограничен только истинными словами английского языка, в том смысле, что автор встречался с их реальным применением. Полные словари содержат тысячи слов, известных лишь особо посвященным, такие как aalii, abamp, ..., zymin и zyxst; но такие слова нужны, в первую очередь, игрокам в Scrabble®*. Однако применение таких слов уменьшает удовольствие тех, кто

* Скрэббл (от англ. *Scrabble* — *пытаться в поисках чего-либо*) — настольная игра, в которую могут играть от 2 до 4 человек, выкладывая слова из имеющихся у них букв в поле размером 15 × 15. В русскоязычной среде известна под названием *Эрудит*. — *Примеч. пер.*

с ними не знаком. Поэтому в течение двадцати лет автор систематически записывал все слова, которые казались ему подходящими для дидактических целей *Искусства программирования*.

Наконец, эту коллекцию стало просто необходимо “заморозить” и прекратить пополнять хотя бы для того, чтобы иметь возможность получать воспроизводимые результаты экспериментов. Английский язык продолжает эволюционировать, но 5757 слов всегда останутся неизменными — несмотря на то, что временами автор порывался добавить к списку слова, с которыми он не был знаком в 1992 году, такие как *chads*, *stent*, *blogs*, *ditzy*, *phish*, *bling* и, возможно, *tetch*. Но это не было сделано — время для внесения изменений в SGB закончилось.

Этот словарь предназначен для хранения всех хорошо известных английских слов... которые можно использовать в приличном обществе и которые могут служить в качестве звеньев.

— ЛЬЮИС КЭРРОЛЛ (LEWIS CARROLL), *Doublets: A Word-Puzzle* (1879)

Собственные имена наподобие Knuth не рассматривались как корректные пятибуквенные слова. Однако, например, такое слово, как *gauss*, корректно, поскольку “gauss” представляет собой единицу измерения магнитной индукции. Слова в SGB состоят только из обычных строчных букв; в списке нет слов с переносами, сокращений или слов, требующих ударения, наподобие *blasé*. Таким образом, каждое слово можно рассматривать и как вектор с пятью компонентами в диапазоне [0..26]. В векторном смысле слова *yucca* и *abuzz* отстоят друг от друга дальше всех: евклидово расстояние между ними равно

$$\|(24, 20, 2, 2, 0) - (0, 1, 20, 25, 25)\|_2 = \sqrt{24^2 + 19^2 + 18^2 + 23^2 + 25^2} = \sqrt{2415}.$$

Полностью Stanford GraphBase со всеми программами и данными можно загрузить с веб-сайта автора <http://www-cs-faculty.stanford.edu/~knuth/sgb.html>. Получить список всех слов SGB еще проще, так как он находится по тому же адресу, но в файле ‘sgb-words.txt’. Этот файл содержит 5757 строк, в каждой из которых имеется одно слово, начиная с ‘which’ и заканчивая ‘pupal’. Слова расположены в порядке, соответствующем их частоте употребления; например, словами с рангами 1000, 2000, 3000, 4000 и 5000 являются соответственно *ditch*, *galls*, *visas*, *faker* и *pismo*. Запись ‘WORDS(*n*)’ в этой главе будет использоваться для обозначения *n* наиболее часто встречающихся слов, в соответствии с их рангами.

Кстати, пятибуквенные слова часто включают множественное число *четырёхбуквенных слов*, что, конечно, недопустимо при строгом подходе к делу, и такие слова не встретятся вам в *The Official Scrabble® Players Dictionary*, но для словаря SGB они вполне годятся. Один из способов гарантировать, что семантически неподходящие термины не попадут в статью, основанную на списке слов SGB, — ограничить рассмотрение множеством WORDS(*n*), где *n* равно, скажем, 3000.

Упр. 26–37 могут использоваться в качестве разминки для начального изучения слов SGB, с которыми мы встретимся в этой главе в самых разных комбинаторных контекстах. Например, пока мы еще недалеко ушли от задач покрытия, мы можем заметить, что четыре слова ‘third flock began jumps’ охватывают 20 из первой 21 буквы алфавита. Пять же слов могут охватить не более 24 различных букв, как

в случае {becks, fjord, glitz, nymph, squaw}, если только мы не обратимся к редко-му, отсутствующему в SGB слову наподобие *waqfs* (пожертвование для мусульманских религиозных или благотворительных целей), которое можно скомбинировать со словами {gyved, bronx, chimp, klutz} для охвата 25 букв.

Простых слов из WORDS(400) достаточно для образования *квадрата из слов*:

class	
light	
agree	.
sheep	
steps	

(9)

Однако для получения *куба из слов* требуется добраться почти до WORDS(3000):

types	yeast	pasta	ester	start
yeast	earth	armor	stove	three
pasta	armor	smoke	token	arena
ester	stove	token	event	rents
start	three	arena	rents	tease

(10)

Здесь каждый “срез” 5×5 представляет собой квадрат из слов. С помощью простого расширения базового алгоритма танцующих связей (см. раздел 7.2.2.2) можно показать ценой около 390 миллиардов обращений к памяти, что WORDS(3000) даст возможность получить только три симметричных куба из слов, таких как (10); в упр. 36 вы найдете оставшиеся два. Интересно, что из полного множества WORDS(5757) можно создать 83 576 симметричных кубов.

Графы из слов. Конечно, интересно и важно помещать объекты в строки, квадраты, кубы и прочие фигуры; но в практических приложениях имеется еще *более* интересная и важная комбинаторная структура, а именно *граф*. Вспомним из раздела 2.3.4.1, что граф представляет собой множество точек, именуемых *вершинами*, вместе с множеством линий, называющихся *ребрами*, которые соединяют определенные пары вершин. Графы применяются повсеместно, и имеется масса красивых алгоритмов, предназначенных для работы с ними, так что графы естественным образом оказываются в центре внимания многих разделов данной главы. Фактически Stanford GraphBase, в первую очередь, посвящена графам, что отражено даже в ее названии; а слова SGB собирались автором главным образом потому, что их можно использовать для определения интересных и поучительных графов.

Льюис Кэрролл (Lewis Carroll) указал путь, изобретя в конце 1877 года игру “Дублеты” (Doublets). [См. Martin Gardner, *The Universe in a Handkerchief* (1996), Chapter 6.] Идея Кэрролла, которая быстро завоевала популярность, заключалась в преобразовании одного слова в другое путем замены на каждом шаге одной буквы; например, вот как слезы (tears) превращаются в улыбку (smile):*

tears — sears — stars — stare — stale — stile — smile. (11)

Кратчайшая такая трансформация является кратчайшим *путем* в графе, в котором вершинами являются английские слова, а ребра соединяют пары слов, расстояние

* Вот как по тем же правилам в русском языке муха превращается в слона: муха — муза — луза — лоза — коза — кора — кара — каре — кафе — кафр — каюр — каюк — кряк — урюк — урок — срок — сток — стон — слон. — *Примеч. пер.*

Хэмминга между которыми равно 1 (т. е. они отличаются одно от другого только одной буквой).

Если ограничиться только словами из SGB, правило Кэрролла дает граф из Stanford GraphBase, официальное название которого — *words*(5757, 0, 0, 0). Каждый граф, определяемый SGB, имеет свой уникальный *идентификатор*, и “кэрроллообразный” граф, порожденный из слов SGB, идентифицируется как *words*(n, l, t, s). Здесь n — количество вершин графа; l либо равно 0, либо представляет собой список весов, используемый для акцентирования различных видов словаря; t представляет собой пороговое значение, с тем чтобы можно было отвергнуть слова с малыми весами; а s — исходное значение для генерации псевдослучайных чисел, которые могут потребоваться для разбиения связей между словами с одинаковыми весами. Детальная информация в данном случае нас не интересует; нескольких примеров вполне достаточно для представления об общей идее.

- *words*($n, 0, 0, 0$) — граф, получающийся при применении правила Кэрролла к множеству WORDS(n), для $1 \leq n \leq 5757$.
- *words*(1000, {0, 0, 0, 0, 0, 0, 0, 0}, 0, s) содержит 1000 выбранных случайным образом слов SGB, обычно разных при разных значениях s .
- *words*(766, {0, 0, 0, 0, 0, 0, 0, 1, 0}, 1, 0) содержит все пятибуквенные слова из книг автора о T_EX и METAFont.

В последнем графе всего лишь 766 слов, так что мы не сможем образовать очень много длинных путей наподобие (11), хотя

$$\begin{aligned} \text{basic} & \text{---} \text{basis} \text{---} \text{bases} \text{---} \text{based} \\ & \text{---} \text{baked} \text{---} \text{naked} \text{---} \text{named} \text{---} \text{names} \text{---} \text{games} \end{aligned} \quad (12)$$

является одним из стоящих внимания примеров.

Конечно, имеется много других способов определения ребер графа, вершины которого представляют пятибуквенные слова. Можно, например, вместо расстояния Хэмминга использовать расстояние Евклида. Можно также объявить два слова смежными, если у них имеется общее подслово длиной 4 буквы; эта стратегия существенно обогащает граф, делая возможным превращение *chaos* в *peace*, даже ограничиваясь 766 словами, связанными с T_EX:

$$\begin{aligned} \text{chaos} & \text{---} \text{chose} \text{---} \text{whose} \text{---} \text{whole} \text{---} \text{holes} \text{---} \text{hopes} \text{---} \text{copes} \text{---} \text{scope} \\ & \text{---} \text{score} \text{---} \text{store} \text{---} \text{stare} \text{---} \text{spare} \text{---} \text{space} \text{---} \text{paces} \text{---} \text{peace}. \end{aligned} \quad (13)$$

(Использованное правило позволяет удалить букву, а затем вставить другую — возможно, в другом месте.) Мы можем выбрать и совершенно иную стратегию, такую как размещение ребра между словами-векторами $a_1 a_2 a_3 a_4 a_5$ и $b_1 b_2 b_3 b_4 b_5$ тогда и только тогда, когда их скалярное произведение $a_1 b_1 + a_2 b_2 + a_3 b_3 + a_4 b_4 + a_5 b_5$ кратно некоторому параметру m . Алгоритмы для работы с графами умеют одинаково успешно справляться с разными видами данных.

Слова SGB приводят также к интересному семейству *ориентированных* графов, если создавать ребра $a_1 a_2 a_3 a_4 a_5 \rightarrow b_1 b_2 b_3 b_4 b_5$ тогда, когда $\{a_2, a_3, a_4, a_5\} \subseteq \{b_1, b_2, b_3, b_4, b_5\}$ при рассмотрении слов как мультимножеств. (Удалить первую букву, добавить другую и переупорядочить.) Применение этого правила позволяет, например, преобразовать *words* в *graph* с помощью кратчайшего ориентированного

пути длиной 6:

$$\text{words} \rightarrow \text{dross} \rightarrow \text{soars} \rightarrow \text{orcas} \rightarrow \text{crash} \rightarrow \text{sharp} \rightarrow \text{graph}. \quad (14)$$

Теория — первый член ряда Тейлора для практики.

— ТОМАС М. КОБЕР (THOMAS M. COVER) (1992)

Количество терминологических систем, используемых в настоящее время в теории графов, неплохо аппроксимируется количеством теоретиков в этой области.

— РИЧАРД П. СТЕНЛИ (RICHARD P. STANLEY) (1986)

Основы теории графов. Граф G состоит из множества вершин V и множества ребер E , которые представляют собой пары различных вершин. Мы будем считать, что V и E являются *конечными* множествами, если только явно не указано иное. Мы пишем $u \text{ — } v$, если u и v являются вершинами, такими, что $\{u, v\} \in E$, и $u \not\text{— } v$, если u и v являются вершинами, такими, что $\{u, v\} \notin E$. Вершины с $u \text{ — } v$ называются соседними, или смежными, в G . Одно из следствий этого определения заключается в том, что $u \text{ — } v$ тогда и только тогда, когда $v \text{ — } u$. Другое следствие заключается в том, что $v \not\text{— } v$ для всех $v \in V$; другими словами, ни одна из вершин не смежна сама себе. (Однако ниже мы рассмотрим мультиграфы, в которых разрешены петли, идущие из вершины в саму себя.)

Граф $G' = (V', E')$ является *подграфом* $G = (V, E)$, если $V' \subseteq V$ и $E' \subseteq E$. Это *остовный* подграф G , если в действительности $V' = V$. И это *порожденный (индуцированный)* подграф G , если, когда V' представляет собой заданное подмножество вершин, E' имеет максимально возможное количество ребер. Другими словами, когда $V' \subseteq V$, подграфом $G = (V, E)$, порожденным V' , является $G' = (V', E')$, где

$$E' = \{ \{u, v\} \mid u \in V', v \in V' \text{ и } \{u, v\} \in E \}. \quad (15)$$

Этот подграф G' обозначается как $G \mid V'$ и часто называется “ G , ограниченный V' ”. В распространенном случае, когда $V' = V \setminus \{v\}$, мы записываем просто $G \setminus v$ (“ G минус вершина v ”) как сокращение для $G \mid (V \setminus \{v\})$. Аналогичная запись $G \setminus e$ (где $e \in E$) используется для обозначения подграфа $G' = (V, E \setminus \{e\})$, получаемого путем удаления ребра, а не вершины. Заметим, что все описанные ранее SGB-графы $\text{words}(n, l, t, s)$ являются порожденными подграфами основного графа $\text{words}(5757, 0, 0, 0)$; в этих рассмотренных графах изменяется только словарь, но не правило смежности.

Граф с n вершинами и e ребрами называется имеющим *порядок n и размер e* . Простейшими и наиболее важными графами порядка n являются *полный граф K_n , путь P_n и цикл C_n* . Предположим, что вершинами являются $V = \{1, 2, \dots, n\}$. Тогда

- K_n имеет $\binom{n}{2} = \frac{1}{2}n(n-1)$ ребер $u \text{ — } v$ для $1 \leq u < v \leq n$; каждый граф с n вершинами является остовным подграфом K_n ;
- P_n имеет $n-1$ ребер $v \text{ — } (v+1)$ для $1 \leq v < n$, где $n \geq 1$; это — путь длиной $n-1$ от 1 до n ;
- C_n имеет n ребер $v \text{ — } ((v \bmod n)+1)$ для $1 \leq v \leq n$, где $n \geq 1$; он является графом, только когда $n \geq 3$ (C_1 и C_2 являются мультиграфами).

В действительности можно определить K_n , P_n и C_n на вершинах $\{0, 1, \dots, n-1\}$ или на *любом* n -элементном множестве V вместо $\{1, 2, \dots, n\}$, поскольку два графа, отличающихся только именами вершин, но не структурой ребер, комбинаторно эквивалентны.

Формально графы $G = (V, E)$ и $G' = (V', E')$ *изоморфны*, если существует взаимно однозначное соответствие φ между V и V' , такое, что $u - v$ в G тогда и только тогда, когда $\varphi(u) - \varphi(v)$ в G' . Для указания изоморфности графов G и G' часто используется обозначение $G \cong G'$; впрочем, часто мы будем менее строги, рассматривая изоморфные графы, как если бы они были эквивалентными, и время от времени записывая $G = G'$, даже когда множества вершин G и G' не строго идентичны.

Небольшие графы можно определить, просто начертив диаграмму, в которой вершины изображены маленькими кружками, а ребра — линиями между ними. На рис. 2 показано несколько важных примеров, свойства которых мы изучим позже. Граф Петерсена на рис. 2, (д) назван по имени Юлиуса Петерсена (Julius Petersen), одного из основателей теории графов, который использовал его для опровержения одного правдоподобного предположения [L'*Intermédiaire des Mathématiciens* 5 (1898), 225–227]; в действительности это замечательная конфигурация, которая служит контрпримером для многих оптимистических предсказаний о том, что может оказаться истинно для всех графов в целом. Граф Чватала (см. рис. 2, (е)) описан Вацлавом Чваталом (Václav Chvátal) в *J. Combinatorial Theory* 9 (1970), 93–94.

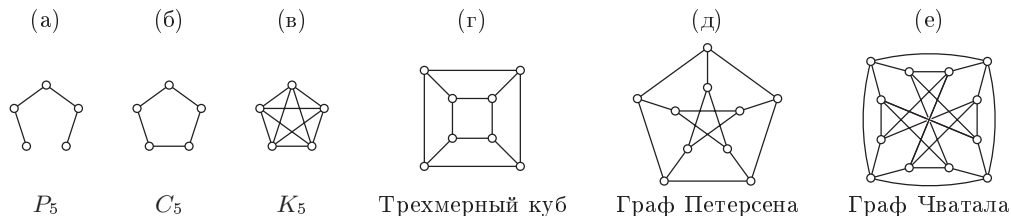


Рис. 2. Шесть примеров графов с (5, 5, 5, 8, 10, 12) вершинами и (4, 5, 10, 12, 15, 24) ребрами соответственно.

Линии на диаграмме могут пересекаться одна с другой в точках, не являющихся вершинами. Например, центральная точка на рис. 2, (е) *не* является вершиной графа Чватала. Граф называется *планарным*, если существует способ изобразить его без каких бы то ни было пересечений. Ясно, что P_n и C_n всегда планарны; представленная на рис. 2, (г) диаграмма показывает, что трехмерный куб также является планарным графом. Но K_5 имеет слишком много вершин, чтобы быть планарным (см. упр. 46).

Степенью вершины является количество ее соседей. Если все вершины имеют одну и ту же степень, граф называется *регулярным*. На рис. 2, например, P_5 является нерегулярным графом, так как две его вершины имеют степень 1, а три — степень 2. Остальные пять графов регулярны, со степенями соответственно (2, 4, 3, 3, 4). Регулярный граф степени 3 часто называется кубическим или трехвалентным.

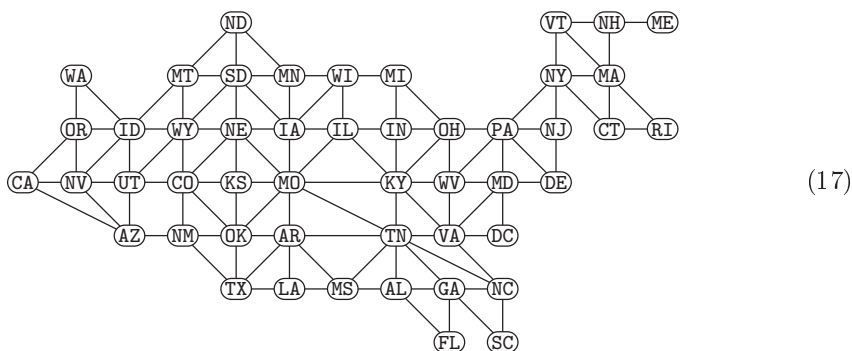
Имеется много способов изобразить тот или иной граф, и одни из них более выразительны, чем другие. Например, каждая из шести диаграмм



изоморфна трехмерному кубу (рис. 2, (г)). Диаграмма графа Чватала, показанная на рис. 2, (е) и раскрывающая неожиданную симметрию графа, придумана Адрианом Бонди (Adrian Bondy) через много лет после публикации статьи Чватала.

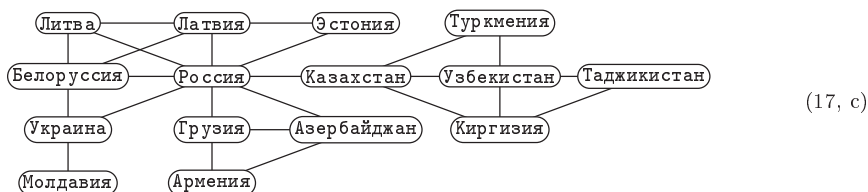
Симметриями графа, известными также как его *автоморфизмы*, являются перестановки его вершин, сохраняющие отношение смежности. Другими словами, перестановка φ является автоморфизмом G , если $\varphi(u) - \varphi(v)$, когда $u - v$ в графе G . Хорошо продуманный чертеж наподобие рис. 2, (е) может выявить симметрию графа, но одна диаграмма не всегда в состоянии показать все существующие симметрии. Например, трехмерный куб имеет 48 автоморфизмов, а граф Петерсена — 120. Мы изучим алгоритмы для работы с изоморфизмами и автоморфизмами в разделе 7.2.3. Симметрии часто помогают избавиться от излишних вычислений, делая алгоритмы при работе с графами, обладающими k автоморфизмами, почти в k раз быстрее.

Графы, связанные с объектами реального мира, обычно существенно отличаются от чисто математических графов наподобие приведенных на рис. 2. Например, вот достаточно знакомый американцам граф, вовсе не имеющий симметрии, но зато обладающий достоинством планарности.



Он представляет внутренние границы США, и позже мы используем его в нескольких примерах. Для удобства вместо изображения пустых кружков 49 вершин на этой диаграмме помечены двухбуквенными кодами штатов.*

* От переводчика: пожалуй, читателям переводного издания книги более знакомыми окажутся подобные географические графы, представляющие, например, республики бывшего СССР:



Пути и циклы. Остовный путь P_n графа называется *гамильтоновым путем*, и остовный цикл C_n — *гамильтоновым циклом*, по имени У. Р. Гамильтона (W. R. Hamilton), предложившего в 1856 году головоломку, состоящую в поиске таких путей по ребрам додекаэдра. Независимо от него эту задачу для многогранников в общем случае изучал Т. П. Киркман (T. P. Kirkman) в *Philosophical Transactions* **146** (1856), 413–418; **148** (1858), 145–161. [См. N. L. Biggs, E. K. Lloyd, R. J. Wilson *Graph Theory 1736–1936* (1998), Chapter 2.] Однако задача поиска остовного пути или цикла гораздо старше; на самом деле ее можно рассматривать как наиболее старую из комбинаторных задач, поскольку поиск путей обхода шахматной доски конем имеет долгую историю, ведущую вглубь веков — в Индию IX века (см. раздел 7.3.3). Граф называется *гамильтоновым*, если он имеет гамильтонов цикл. (Граф Петерсена, кстати, является наименьшим 3-регулярным графом, не являющимся ни планарным, ни гамильтоновым; см. C. de Polignac, *Bull. Soc. Math. de France* **27** (1899), 142–145.)

Обхватом (*girth*) графа называется длина его наименьшего цикла; обхват ациклического графа (не содержащего циклов) бесконечен. Например, шесть графов на рис. 2 имеют обхваты $(\infty, 5, 3, 4, 5, 4)$ соответственно. Нетрудно доказать, что граф с минимальной степенью k и обхватом 5 должен иметь как минимум $k^2 + 1$ вершин. Дальнейший анализ показывает, что фактически этот минимум достижим, только если $k = 2$ (C_5), $k = 3$ (граф Петерсена), $k = 7$ или, вероятно, $k = 57$ (см. упр. 63 и 65.)

Расстоянием $d(u, v)$ между двумя вершинами u и v является минимальная длина пути от u до v в графе; оно равно бесконечности, если такого пути не существует. Ясно, что $d(v, v) = 0$ и что $d(u, v) = d(v, u)$. Справедливо также неравенство треугольника

$$d(u, v) + d(v, w) \geq d(u, w). \quad (18)$$

Ибо, если $d(u, v) = p$, $d(v, w) = q$, $p < \infty$ и $q < \infty$, то существуют пути

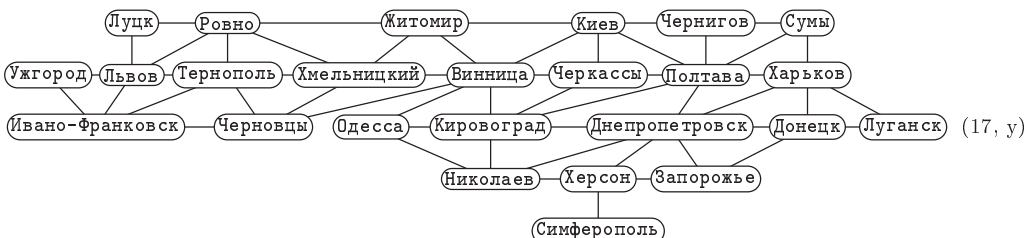
$$u = u_0 \text{ --- } u_1 \text{ --- } \cdots \text{ --- } u_p = v \quad \text{и} \quad v = v_0 \text{ --- } v_1 \text{ --- } \cdots \text{ --- } v_q = w, \quad (19)$$

и можно найти наименьший индекс r , такой, что $u_r = v_s$ для некоторого s . Тогда

$$u_0 \text{ --- } u_1 \text{ --- } \cdots \text{ --- } u_{r-1} \text{ --- } v_s \text{ --- } v_{s+1} \text{ --- } \cdots \text{ --- } v_q \quad (20)$$

представляет собой путь длиной $\leq p + q$ от u до w .

Диаметром графа является максимум $d(u, v)$ по всем вершинам u и v . Граф называется *связным*, если его диаметр конечен. Вершины графа всегда могут быть или области Украины (соответствующий граф для России занял бы слишком много места):



(Здесь вершины графа помечены областными центрами и столицей автономной республики Крым, а не названиями соответствующих областей.)

разбиты на связные *компоненты*, где две вершины, u и v , принадлежат одному и тому же компоненту тогда и только тогда, когда $d(u, v) < \infty$.

В графе *words* (5757, 0, 0, 0), например, $d(\text{tears}, \text{smile}) = 6$, поскольку (11) представляет собой кратчайший путь от *tears* к *smile*. Кроме того, $d(\text{tears}, \text{happy}) = 6$, $d(\text{smile}, \text{happy}) = 10$ и $d(\text{world}, \text{court}) = 6$. Но $d(\text{world}, \text{happy}) = \infty$; таким образом, рассматриваемый граф не является связным. В действительности он содержит 671 слово наподобие *aloof*, которые не имеют соседей и сами по себе образуют связные компоненты порядка 1. Пары слов наподобие *alpha* — *aloha*, *droid* — *druid* и *opium* — *odium* дают еще 103 связных компонента порядка 2. Одни компоненты порядка 3 наподобие *chain* — *chair* — *choir* являются путями; другие, такие как {*getup*, *letup*, *setup*}, представляют собой циклы. Имеется также немного большее количество небольших компонентов, наподобие занимательного пути

$$\text{login} — \text{logic} — \text{yogic} — \text{yogis} — \text{yogas} — \text{togas}, \quad (21)$$

слова в котором не имеют других соседей. Но подавляющее большинство пятибуквенных слов принадлежит гигантскому компоненту порядка 4493. Если вы можете пройти два шага от имеющегося у вас слова, меняя две различные буквы, то шансы, что это слово принадлежит указанному гигантскому компоненту, превышают 15 к 1.

Аналогично граф *words* ($n, 0, 0, 0$) имеет при $n = (5000, 4000, 3000, 2000, 1000)$ гигантский компонент порядка (3825, 2986, 2056, 1186, 224) соответственно. Но если n мало, то ребер для связности оказывается недостаточно. Например, *words* (500, 0, 0, 0) имеет 327 различных компонентов, причем среди них нет ни одного компонента порядка 15 или выше.

Концепцию расстояния можно обобщить и рассматривать величину $d(v_1, v_2, \dots, v_k)$ для любого значения k , которая представляет собой минимальное количество ребер в связанном подграфе, содержащем вершины $\{v_1, v_2, \dots, v_k\}$. Например, $d(\text{blood}, \text{sweat}, \text{tears})$ оказывается равным 15, поскольку подграф

$$\begin{array}{ccccccccccc} \text{blood} & — & \text{brood} & — & \text{broad} & — & \text{bread} & — & \text{tread} & — & \text{treed} & — & \text{tweed} \\ & & & & & & & & & & & & | \\ \text{tears} & — & \text{teams} & — & \text{trams} & — & \text{trims} & — & \text{tries} & — & \text{trees} & & \text{tweet} \\ & & & & & & & & & & & & | \\ & & & & & & & & & & & & \text{sweat} — \text{sweet} \end{array} \quad (22)$$

имеет 15 ребер, а подходящего подграфа из 14 ребер не существует.

В разделе 2.3.4.1 мы упоминали, что связный граф с наименьшим количеством ребер называется *свободным деревом*. Подграф, соответствующий обобщенному расстоянию $d(v_1, \dots, v_k)$, всегда будет свободным деревом. Его ошибочно называют *деревом Штейнера*, поскольку Якоб Штейнер (Jacob Steiner) однажды упомянул случай $k = 3$ для точек $\{v_1, v_2, v_3\}$ на евклидовой плоскости [Crelle **13** (1835), 362–363]. Франц Хайнен (Franz Heinen) решил эту задачу в *Über Systeme von Kräften* (1834); Гаусс (Gauss) распространил анализ на случай $k = 4$ в письме к Шумахеру (Schumacher) от 21 марта 1836 года.

Раскраска графа. Граф называется *k-дольным*, или *k-раскрашиваемым*, если его вершины могут быть разбиты на k или менее частей так, что концы каждого ребра принадлежат различным частям — или, что то же самое, если имеется способ раскрасить его вершины не более чем k различными цветами, так, чтобы смежные

вершины не были окрашены в один и тот же цвет. Знаменитая Теорема четырех красок, сформулированная в 1852 году Ф. Гутри (F. Guthrie) и окончательно доказанная с применением компьютера К. Аппелем (K. Appel), В. Хакеном (W. Haken) и Д. Кохом (J. Koch) [Illinois J. Math. **21** (1977), 429–567], гласит, что *любой планарный граф является 4-раскрашиваемым*. Простое доказательство этой теоремы неизвестно, но частные случаи наподобие (17) могут быть раскрашены на глаз (см. упр. 45); в общем случае для 4-раскрашивания планарного графа достаточно $O(n^2)$ шагов [N. Robertson, D. P. Sanders, P. Seymour и R. Thomas, *STOC* **28** (1996), 571–575].

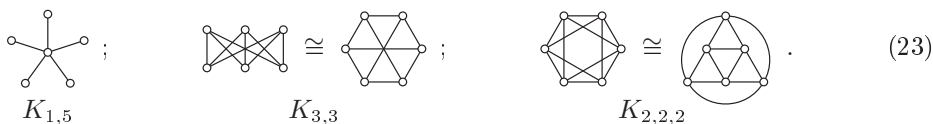
На практике особенно важен случай 2-раскрашиваемых графов. Такие графы в общем случае называются *двудольными* или просто *биграфами*; каждое ребро такого графа имеет по одной конечной точке в каждой из частей.

Теорема В. *Граф является двудольным тогда и только тогда, когда он не содержит циклов нечетной длины.*

Доказательство. [См. D. König, *Math. Annalen* **77** (1916), 453–454.] Каждый подграф k -дольного графа является k -дольным. Следовательно, цикл C_n может быть подграфом двудольного графа, только если C_n сам является двудольным, а в этом случае n должно быть четным.

И наоборот, если граф не содержит циклов нечетной длины, мы можем раскрасить его вершины двумя цветами $\{0, 1\}$, придерживаясь следующей процедуры. Изначально все вершины не окрашены. Если все соседи окрашенных вершин уже окрашены, выберем неокрашенную вершину w и окрасим ее в цвет 0. В противном случае выберем окрашенную вершину u , которая имеет неокрашенного соседа v ; назначим v противоположный u цвет. В упр. 48 доказывается, что в конечном итоге при этом получится корректная раскраска. ■

Полный двудольный граф $K_{m,n}$ представляет собой наибольший двудольный граф, вершины которого разделены на две части размерами m и n . Его можно определить на множестве вершин $\{1, 2, \dots, m+n\}$, говоря, что $u \sim v$ тогда, когда $1 \leq u \leq m < v \leq m+n$. Другими словами, $K_{m,n}$ имеет mn ребер, по одному для каждого выбора одной вершины из одной части и второй — из другой. Аналогично *полный k -дольный граф* $K_{n_1, \dots, n_k}$ имеет $N = n_1 + \dots + n_k$ вершин, разделенных на части размерами $\{n_1, \dots, n_k\}$, и имеет ребра между любыми двумя вершинами, не принадлежащими одной и той же части. Вот несколько примеров для $N = 6$:



Обратите внимание, что $K_{1,n}$ представляет собой свободное дерево; его часто называют *звездой* порядка $n + 1$.

Далее я буду говорить “орграф” вместо “ориентированный граф”.

Это просто и кратко и явно войдет в моду.

— ГЁРГИ ПОЙА (GYÖRGY PÓLYA),

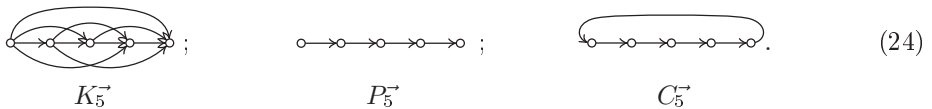
письмо ФРЭНКУ ХАРАРИ (FRANK HARARY) (ок. 1954 года)

Ориентированные графы. В разделе 2.3.4.2 мы определили *ориентированные графы* (или *орграфы*), которые очень похожи на графы, с тем отличием, что вместо ребер они имеют *дуги*. Дуга $u \rightarrow v$ направлена от одной вершины к другой, в то время как ребро $u - v$ соединяет две вершины, не различая их. Кроме того, ориентированные графы могут иметь петли $v \rightarrow v$, идущие из вершины в нее же, а также между двумя вершинами u и v может находиться более одной дуги $u \rightarrow v$.

Формально ориентированным графом $D = (V, A)$ порядка n и размера t являются множество V из n вершин и мультимножество A из t упорядоченных пар (u, v) , где $u \in V$ и $v \in V$. Упорядоченные пары называются дугами, и мы записываем $u \rightarrow v$, когда $(u, v) \in A$. Ориентированный граф называется *простым*, если A является множеством, а не мультимножеством, т. е. если для всех u и v имеется не более одной дуги (u, v) . Каждая дуга (u, v) имеет начальную вершину u и конечную вершину v , именуемую также *верхушкой* (tip). Каждая вершина имеет *исходящую степень* $d^+(v)$, равную количеству дуг, для которых v является начальной вершиной, и *входящую степень* $d^-(v)$, равную количеству дуг, для которых v является верхушкой. Вершина с нулевой входящей степенью называется “источником”, а вершина с нулевой исходящей степенью называется “стоком”. Заметим, что $\sum_{v \in V} d^+(v) = \sum_{v \in V} d^-(v)$, поскольку обе суммы равны t — общему количеству дуг.

Большинство концепций, определенных для графов, естественным путем распространяются и на ориентированные графы. Достаточно просто вставить слово “ориентированный”, когда возникает необходимость различать ребра и дуги. Например, ориентированные графы имеют ориентированные подграфы, которые могут быть остовными, порожденными или ни теми, ни другими. Изоморфизмом между ориентированными графами $D = (V, A)$ и $D' = (V', A')$ является взаимно однозначное соответствие φ между V и V' , при котором количество дуг $u \rightarrow v$ в D равно количеству дуг $\varphi(u) \rightarrow \varphi(v)$ в D' для всех $u, v \in V$.

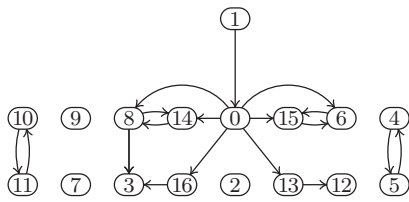
На диаграммах ориентированных графов вместо линий между вершинами используются стрелки. Простейшими и наиболее важными ориентированными графами порядка n являются ориентированные варианты графов K_n , P_n и C_n , а именно *транзитивный турнир* $K_n^{\rightarrow}$, *ориентированный путь* $P_n^{\rightarrow}$ и *ориентированный цикл* $C_n^{\rightarrow}$. Схематически для $n = 5$ они показаны на следующих диаграммах:



Имеется также *полный ориентированный граф* J_n , который представляет собой наибольший простой ориентированный граф с n вершинами; он имеет n^2 дуг $u \rightarrow v$, по одной для каждого варианта выбора u и v .

На рис. 3 показана более сложная диаграмма порядка 17, которую можно назвать отчетливо ориентированной: это ориентированный граф, описанный Эркюлем Пуаро (Hercule Poirot) в детективе Агаты Кристи (Agatha Christie) *Убийство в Восточном экспрессе* (1934).^{*} Вершины соответствуют местам в описанном в детективе

^{*} Непереводаемая игра слов: “отчетливо ориентированная” (“expressly oriented”) перекликается с оригинальным названием повести А. Кристи *Murder on the Orient Express*. Имена персонажей на диаграмме взяты из перевода Л. Беспаловой (Агата Кристи. *Восточный экспресс. Десять негритят. Романы*. — М., 1990). — *Примеч. пер.*



Условные обозначения

0 — Пьер Мишель, француз, проводник
1 — Эркюль Пуаро, бельгиец, детектив

2 — Сэмьюэл Эдуард Рэтчетт, погибший американец
3 — Каролина Марта Хаббард, американка, домохозяйка
4 — Эдуард Генри Мастермэн, англичанин, слуга
5 — Антонио Фоскарелли, итальянец, агент по продажам
6 — Гектор Уиллард Маккуин, американец, секретарь
7 — Харви Харрис, англичанин, не явился
8 — Хильдегарда Шмидт, немка, горничная
9 — (Пустое место)
10 — Грета Ольсон, шведка, экономка
11 — Мэри Хермиона Дебенхэм, англичанка, гувернантка
12 — Хелена Мария Андени, прекрасная графиня
13 — Рудольф Андени, венгр, граф, дипломат
14 — Наталья Драгомирова, русская, княгиня
15 — Полковник Арбэнтот, англичанин, офицер из Индии
16 — Сайрус Бетман Хардман, американец, детектив

Рис. 3. Ориентированный граф порядка 17 размером 18, придуманный Агатой Кристи.

спальном вагоне Стамбул–Кале, а дуги $u \rightarrow v$ означают, что пассажир с места u подтверждает алиби пассажира с места v . В этом графе имеется шесть связанных компонентов, а именно $\{0, 1, 3, 6, 8, 12, 13, 14, 15, 16\}$, $\{2\}$, $\{4, 5\}$, $\{7\}$, $\{9\}$ и $\{10, 11\}$, поскольку связность ориентированного графа определяется путем рассмотрения дуг как ребер.

Две дуги являются *последовательными*, если вершущка первой является начальной вершиной второй. Ряд последовательных дуг $(a_1, a_2, \dots, a_k)$ называется *обходом* (*walk*) длиной k ; его можно определить с помощью как дуг, так и вершин:

$$v_0 \xrightarrow{a_1} v_1 \xrightarrow{a_2} v_2 \cdots v_{k-1} \xrightarrow{a_k} v_k. \quad (25)$$

В простом ориентированном графе достаточно просто указать вершины; например, $1 \rightarrow 0 \rightarrow 8 \rightarrow 14 \rightarrow 8 \rightarrow 3$ представляет собой обход для графа на рис. 3. Обход (25) представляет собой ориентированный путь, если вершины $\{v_0, v_1, \dots, v_k\}$ различны; он является ориентированным циклом, если все вершины различны, за исключением $v_k = v_0$.

В ориентированном графе ориентированное расстояние $d(u, v)$ представляет собой количество дуг в кратчайшем *ориентированном* пути от u к v , которое также равно длине кратчайшего обхода от u к v . Оно может отличаться от $d(v, u)$; но неравенство треугольника (18) остается корректным.

Каждый граф может рассматриваться как ориентированный граф, поскольку ребро $u - v$ по сути эквивалентно соответствующей паре дуг, $u \rightarrow v$ и $v \rightarrow u$. Ориентированный граф, полученный таким образом, обладает всеми свойствами исходного графа; например, степень каждой вершины графа становится исходящей степенью ориентированного графа, а также ее входящей степенью. Кроме того, расстояния остаются неизменными.

Мультиграф (V, E) подобен графу, с тем отличием, что его ребра E могут быть *мультимножеством* пар $\{u, v\}$; в мультиграфе разрешены также ребра $v - v$, представляющие собой петли из вершины в нее же, соответствующие “мультипарам” $\{v, v\}$. Например,

$$\textcircled{1} - \textcircled{2} - \textcircled{3} \quad (26)$$

представляет собой мультиграф порядка 3 с 6 ребрами, $\{1, 1\}$, $\{1, 2\}$, $\{2, 3\}$, $\{2, 3\}$, $\{3, 3\}$ и $\{3, 3\}$. Степени вершин в этом примере равны $d(1) = d(2) = 3$ и $d(3) = 6$, поскольку каждая петля вносит в степень вершины вклад, равный 2. При рассмот-

рении мультиграфа как ориентированного графа петля $v \rightarrow v$ становится *двумя* дугами-петлями $v \rightarrow v$.

Представление графов и ориентированных графов. Любой ориентированный граф, а значит, любой граф или мультиграф, полностью описывается его *матрицей смежности* $A = (a_{uv})$, имеющей n строк и n столбцов при наличии у ориентированного графа n вершин. Каждый элемент a_{uv} этой матрицы определяет количество дуг от u до v . Например, матрицами смежности для $K_3^{\rightarrow}$, $P_3^{\rightarrow}$, $C_3^{\rightarrow}$, J_3 и (26) являются соответственно

$$K_3^{\rightarrow} = \begin{pmatrix} 011 \\ 001 \\ 000 \end{pmatrix}, \quad P_3^{\rightarrow} = \begin{pmatrix} 010 \\ 001 \\ 000 \end{pmatrix}, \quad C_3^{\rightarrow} = \begin{pmatrix} 010 \\ 001 \\ 100 \end{pmatrix}, \quad J_3 = \begin{pmatrix} 111 \\ 111 \\ 111 \end{pmatrix}, \quad A = \begin{pmatrix} 210 \\ 102 \\ 024 \end{pmatrix}. \quad (27)$$

Мощный математический инструмент теории матриц позволяет доказывать многие нетривиальные результаты для графов путем изучения их матриц смежности; в упр. 65 представлены особенно впечатляющие примеры того, что может быть сделано таким путем. Одной из основных причин этого является то, что умножение матриц в контексте ориентированных графов имеет простую интерпретацию. Рассмотрим квадрат A , в котором элемент на пересечении строки u и столбца v по определению равен

$$(A^2)_{uv} = \sum_{w \in V} a_{uw} a_{wv}. \quad (28)$$

Поскольку a_{uw} — количество дуг от u до w , мы видим, что $a_{uw} a_{wv}$ — количество обходов вида $u \rightarrow w \rightarrow v$. Следовательно, $(A^2)_{uv}$ является общим количеством обходов длиной 2 от u до v . Аналогично элементы A^k дают общее количество обходов длиной k между любыми упорядоченными парами вершин для всех $k \geq 0$. Например, матрица A в (27) дает

$$A = \begin{pmatrix} 2 & 1 & 0 \\ 1 & 0 & 2 \\ 0 & 2 & 4 \end{pmatrix}, \quad A^2 = \begin{pmatrix} 5 & 2 & 2 \\ 2 & 5 & 8 \\ 2 & 8 & 20 \end{pmatrix}, \quad A^3 = \begin{pmatrix} 12 & 9 & 12 \\ 9 & 18 & 42 \\ 12 & 42 & 96 \end{pmatrix}; \quad (29)$$

таким образом, существует 12 обходов длиной 3 от вершины 1 мультиграфа (26) до вершины 3 и 18 таких обходов из вершины 2 назад в нее же.

Переупорядочение вершин заменяет матрицу смежности A на $P^{-1}AP$, где P — матрица перестановки (0–1-матрица, в каждой строке и каждом столбце которой имеется ровно одна единица), а $P^{-1} = P^T$ представляет собой матрицу для обратной перестановки. Таким образом,

$$\begin{pmatrix} 210 \\ 102 \\ 024 \end{pmatrix}, \quad \begin{pmatrix} 201 \\ 042 \\ 120 \end{pmatrix}, \quad \begin{pmatrix} 012 \\ 120 \\ 204 \end{pmatrix}, \quad \begin{pmatrix} 021 \\ 240 \\ 102 \end{pmatrix}, \quad \begin{pmatrix} 402 \\ 021 \\ 210 \end{pmatrix} \quad \text{и} \quad \begin{pmatrix} 420 \\ 201 \\ 012 \end{pmatrix} \quad (30)$$

представляют собой все матрицы смежности для (26), и никаких других матриц смежности для этого мультиграфа не существует.

Имеется более чем $2^{n(n-1)/2}/n!$ графов порядка n , при $n > 1$, и почти все они требуют $\Omega(n^2)$ бит данных при наиболее экономичном кодировании. Следовательно, для подавляющего большинства всех возможных графов наилучший способ их представления в компьютере с точки зрения требующейся памяти — это работа с их матрицами смежности.

Однако графы, возникающие в практических задачах, по своим характеристикам существенно отличаются от графов, выбираемых случайным образом из

множества всех возможных. Такие реальные графы, как правило, оказываются “разреженными”, имеющими, скажем, $O(n \log n)$ ребер вместо $\Omega(n^2)$, если только n не слишком малó, поскольку $\Omega(n^2)$ бит данных сложно генерировать. Например, предположим, что вершины соответствуют людям, а ребра — отношению дружбы. Если мы рассмотрим 5 иллиардов людей, мало у кого из них будет более чем 10 тысяч друзей. Но даже если каждый в среднем имеет 10 тысяч друзей, то у графа все равно будет только 2.5×10^{13} ребер, в то время как почти все графы с порядком 5 миллиардов имеют примерно 6.25×10^{18} ребер.

Таким образом, наилучший способ представления графа в машине обычно оказывается отличным от записи n^2 значений a_{uv} матрицы смежности. Вместо этого алгоритмы Stanford GraphBase разработаны для работы со структурой данных, похожей на структуру для связанного представления разреженных матриц, рассматривавшейся в разделе 2.2.6, хотя и несколько упрощенной. Такой подход не только доказанно универсален и эффективен, но и прост в использовании.

Представление ориентированного графа в SGB представляет собой комбинацию последовательного и связанного распределения с применением узлов двух базовых типов. Ряд узлов представляет вершины, другие представляют дуги. (Имеется и третий тип узлов, которые представляют весь граф в целом, предназначенный для алгоритмов, работающих одновременно с несколькими графами. Но каждый граф требует только одного узла графа, так что узлы вершин и дуг доминируют.)

Вот как это работает. Каждый ориентированный граф SGB порядка n и размера m построен на последовательном массиве из n узлов вершин, что облегчает доступ к вершине k для $0 \leq k < n$. Напротив, m узлов дуг связаны вместе в общем пуле памяти, который, по сути, неструктурирован. Каждый узел вершины обычно занимает 32 байт, а каждый узел дуги — 20 байт (узел графа занимает 220 байт); но размеры узлов могут быть легко изменены. Несколько полей каждого узла фиксированы и имеют определенное значение в любом случае. Остальные поля могут использоваться для различных целей разными алгоритмами или на разных этапах одного и того же алгоритма. Фиксированные части узла называются стандартными полями, а многоцелевые части — сервисными полями.

Каждый узел вершины имеет два стандартных поля, NAME и ARCS. Если v — переменная, указывающая на узел вершины, мы называем ее *переменной вершины*. Тогда NAME(v) указывает на строку символов, которые могут использоваться для идентификации соответствующей вершины удобным для восприятия человеком способом; например, 49 вершин графа (17) имеют имена наподобие CA, WA, OR, ..., RI. Другое стандартное поле, ARCS(v), существенно важнее для алгоритмов: оно указывает на узел дуги, первый в односвязном списке длиной $d^+(v)$, в котором имеется по одному узлу для каждой дуги, исходящей из вершины v .

Каждый узел дуги имеет два стандартных поля, TIP и NEXT; переменная a , которая указывает на узел дуги, называется *переменной дуги*. TIP(a) указывает на узел вершины, представляющий верхушку дуги a ; NEXT(a) указывает на узел дуги, представляющий следующую дугу, начальной вершиной которой является та же вершина, что и у дуги a .

Вершина v с исходящей степенью 0 представляется соответствующим узлом путем присваивания ARCS(v) = Λ (нулевой указатель). В противном случае, если, скажем, исходящая степень равна 3, структура данных содержит три узла дуг

с $\text{ARCS}(v) = a_1$, $\text{NEXT}(a_1) = a_2$, $\text{NEXT}(a_2) = a_3$ и $\text{NEXT}(a_3) = \Lambda$; и эти три дуги из вершины v ведут в $\text{TIP}(a_1)$, $\text{TIP}(a_2)$, $\text{TIP}(a_3)$.

Предположим, например, что мы хотим вычислить исходящую степень вершины v и сохранить ее в сервисном поле ODEG . Это просто.

$$\begin{aligned} &\text{Установить } a \leftarrow \text{ARCS}(v) \text{ и } d \leftarrow 0. \\ &\text{Пока } a \neq \Lambda, \text{ устанавливая } d \leftarrow d + 1 \text{ и } a \leftarrow \text{NEXT}(a). \\ &\text{Установить } \text{ODEG}(v) \leftarrow d. \end{aligned} \quad (31)$$

Когда граф или мультиграф рассматривается как ориентированный граф, как упоминалось выше, каждое его ребро $u \rightarrow v$ эквивалентно двум дугам, $u \rightarrow v$ и $v \rightarrow u$. Эти дуги называются парными (*mate*) и занимают два узла дуг, скажем, a и a' , где a появляется в списке дуг, исходящих из u , а a' — в списке дуг, исходящих из v . Тогда $\text{TIP}(a) = v$ и $\text{TIP}(a') = u$. Мы также записываем

$$\text{MATE}(a) = a' \quad \text{и} \quad \text{MATE}(a') = a \quad (32)$$

в алгоритмах, которые должны быстро переходить от одного списка к другому. Однако обычно нам не требуется хранить явно указатель от дуги на парную ей или использовать сервисное поле MATE в каждом узле дуги, поскольку необходимые связи можно вывести *неявно* при правильном построении структур данных.

Неявный вывод работает примерно следующим образом: при создании каждого ребра $u \rightarrow v$ неориентированного графа или мультиграфа мы вводим *последовательные* узлы дуг для $u \rightarrow v$ и $v \rightarrow u$. Например, если узлу дуги отводится 20 байт, мы резервируем 40 последовательных байтов для каждой новой пары. Мы можем также обеспечить, чтобы адрес в памяти первого байта был кратен 8. Тогда, если узел дуги a находится в памяти по адресу α , узел парной дуги находится по адресу

$$\left\{ \begin{array}{ll} \alpha + 20, & \text{если } \alpha \bmod 8 = 0 \\ \alpha - 20, & \text{если } \alpha \bmod 8 = 4 \end{array} \right\} = \alpha - 20 + (40 \& ((\alpha \& 4) - 1)). \quad (33)$$

Такой трюк очень полезен в комбинаторных задачах, когда операции могут выполняться триллионы раз, поскольку при этом экономия в 3.6 нс на одной операции приведет к тому, что программа завершит работу на час раньше. Но способ (33) не является непосредственно “переносимым” из одной реализации в другую. Если размер узла дуги изменится, например, с 20 до 24, то числа 40, 20, 8 и 4 в (33) нужно будет заменить на 48, 24, 16 и 8 соответственно.

В алгоритмах в этой книге не делается никаких предположений о размерах узлов. Вместо этого мы примем соглашение языка программирования C и его потомков о том, что если a указывает на узел дуги, то ‘ $a + 1$ ’ представляет собой указатель на узел дуги, следующий за первым в памяти. В общем случае

$$\text{LOC}(\text{NODE}(a + k)) = \text{LOC}(\text{NODE}(a)) + kc, \quad (34)$$

если размер каждого узла дуги равен c байт. Аналогично если v представляет собой переменную вершины, то ‘ $v + k$ ’ будет k -м узлом вершины, следующим за узлом, на который указывает v ; фактический адрес памяти этого узла будет равен v плюс умноженное на размер узла вершины значение k .

Стандартные поля узла графа g включают общее количество дуг $\text{M}(g)$ и общее количество вершин $\text{N}(g)$; указатель на первый узел вершины в последовательном

списке всех узлов вершин $\text{VERTICES}(g)$; идентификатор графа $\text{ID}(g)$, представляющий собой строку наподобие $\text{words}(5757, 0, 0, 0)$; а также некоторые другие поля, необходимые для выделения и освобождения памяти при увеличении или уменьшении графа, или для экспорта графа во внешний формат для взаимодействия с другими пользователями и системами для работы с графами. Но нам редко придется как обращаться к этим полям узла графа, так и давать полное описание формата SGB, поскольку в этой главе мы будем описывать почти все алгоритмы для работы с графами с помощью словесного описания естественным языком на достаточно абстрактном уровне, а не на битовом уровне машинных программ.

Простой алгоритм для работы с графом. Для иллюстрации алгоритма среднего уровня наподобие тех, с которыми мы будем иметь дело позже, давайте превратим доказательство теоремы В в пошаговую процедуру, окрашивающую вершины данного графа в два цвета, если граф является двудольным.

Алгоритм В (*Проверка двудольности*). Для заданного графа, представленного в формате SGB, этот алгоритм либо находит 2-раскрасивание с $\text{COLOR}(v) \in \{0, 1\}$ в каждой вершине v , либо завершается неудачей, если корректное 2-раскрасивание невозможно. Здесь COLOR представляет собой сервисное поле в каждом узле вершины. Другое сервисное поле, $\text{LINK}(v)$, представляет собой указатель на вершину и используется для поддержки стека всех окрашенных вершин, соседи которых еще не протестированы. Вспомогательная переменная вершины s указывает на вершину этого стека. Алгоритм также использует переменные u, v, w для вершин и a для дуг. Предполагается, что узлы вершин представляют собой $v_0 + k$ для $0 \leq k < n$.

- B1.** [Инициализация.] Установить $\text{COLOR}(v_0 + k) \leftarrow -1$ для $0 \leq k < n$. (Сейчас все вершины не окрашены.) Затем установить $w \leftarrow v_0 + n$.
- B2.** [Выполнено?] (В этот момент все вершины $\geq w$ окрашены, и то же относится и к соседям всех окрашенных вершин.) Завершить работу алгоритма успешно, если $w = v_0$. В противном случае установить $w \leftarrow w - 1$ (следующему узлу вершины в направлении уменьшения индекса).
- B3.** [Окраска w при необходимости.] Если $\text{COLOR}(w) \geq 0$, вернуться к шагу B2. В противном случае установить $\text{COLOR}(w) \leftarrow 0$, $\text{LINK}(w) \leftarrow \Lambda$ и $s \leftarrow w$.
- B4.** [Стек $\Rightarrow u$.] Установить $u \leftarrow s$, $s \leftarrow \text{LINK}(s)$, $a \leftarrow \text{ARCS}(u)$. (Мы будем проверять всех соседей окрашенной вершины u .)
- B5.** [Завершено с u ?] Если $a = \Lambda$, перейти к шагу B8. В противном случае установить $v \leftarrow \text{TIP}(a)$.
- B6.** [Обработка v .] Если $\text{COLOR}(v) < 0$, установить $\text{COLOR}(v) \leftarrow 1 - \text{COLOR}(u)$, $\text{LINK}(v) \leftarrow s$ и $s \leftarrow v$. Если же $\text{COLOR}(v) = \text{COLOR}(u)$, завершить работу алгоритма неудачей.
- B7.** [Цикл по a .] Установить $a \leftarrow \text{NEXT}(a)$ и вернуться к шагу B5.
- B8.** [Стек не пустой?] Если $s \neq \Lambda$, вернуться к шагу B4. В противном случае вернуться к шагу B2. ■

Этот алгоритм представляет собой вариант общей процедуры обхода графа, именуемой поиском в глубину, которую мы детально изучим в разделе 7.4.1. Время его работы составляет $O(m + n)$ при наличии m дуг и n вершин (см. упр. 70);

следовательно, он хорошо приспособлен к распространенному случаю разреженных графов. Путем небольших изменений в случае неудачного завершения алгоритм можно заставить выводить цикл нечетной длины, тем самым доказывая невозможность 2-раскраски (см. упр. 72).

Примеры графов. Stanford GraphBase включает библиотеку более чем из трех десятков подпрограмм генерации, способных создавать разнообразные ориентированные и неориентированные графы для использования в экспериментах. Мы уже рассматривали графы *слов*; давайте теперь взглянем на некоторые другие, чтобы увидеть имеющиеся возможности.

- *roget*(1022, 0, 0, 0) представляет собой ориентированный граф с 1022 вершинами и 5075 дугами. Вершины представляют категории слов или концепций, которые П. М. Роджет (P. M. Roget) и Д. Л. Роджет (J. L. Roget) включили в свой знаменитый *Thesaurus* (London: Longmans, Green, 1879). Дуги представляют собой перекрестные ссылки между категориями, имеющиеся в книге. Например, типичными дугами являются *water* $\rightarrow$ *moisture*, *discovery* $\rightarrow$ *truth*, *fool* $\rightarrow$ *sage*, *preparation* $\rightarrow$ *learning*, *vulgarity* $\rightarrow$ *ugliness*, *wit* $\rightarrow$ *amusement*.

- *book*('jean', 80, 0, 1, 356, 0, 0, 0) представляет собой граф с 80 вершинами и 254 ребрами. Вершины представляют персонажей романа *Отверженные* (*Les Misérables*) Виктора Гюго (Victor Hugo); ребра соединяют персонажей, встречавшихся в романе друг с другом. Типичными ребрами являются *Fantine* — *Javert*, *Cosette* — *Thénardier*.

- *bi_book*('jean', 80, 0, 1, 356, 0, 0, 0) представляет собой двудольный граф с 80+356 вершинами и 727 ребрами. Вершины представляют персонажей и главы упомянутого романа *Отверженные*; ребра соединяют персонажей с главами, в которых они появляются (например, *Napoleon* — 2.1.8, *Marius* — 4.14.4).

- *plane_miles*(128, 0, 0, 0, 1, 0, 0) представляет собой планарный граф со 129 вершинами и 381 ребром. Вершины представляют 128 городов США и Канады, а также специальную вершину INF для “точки в бесконечности”. Ребра определяют так называемую *триангуляцию Делоне* этих городов, основанную на их широте и долготе на плоскости; это означает, что *u* — *v* тогда и только тогда, когда существует окружность, проходящая через *u* и *v* и не охватывающая никакую иную вершину. Ребра имеются также между INF и всеми вершинами, лежащими на выпуклой оболочке всех местоположений городов. Типичными ребрами являются *Seattle, WA* — *Vancouver, BC* — INF; *Toronto, ON* — *Rochester, NY*.

- *plane_lisa*(360, 250, 15, 0, 360, 0, 250, 0, 2295000) представляет собой планарный граф с 3027 вершинами и 5967 ребрами. Он получен с помощью обработки оцифрованного изображения *Моны Лизы* Леонардо да Винчи, имеющего 360 строк и 250 столбцов пикселей, состоящей в огрублении интенсивностей пикселей до 16 уровней серого от 0 (черный) до 15 (белый). Полученные в результате 3027 областей одинаковой яркости, связанных “ходом лады”, рассматриваются затем как соседние, если имеют общую границу между пикселями (рис. 4).

- *bi_lisa*(360, 250, 0, 360, 0, 250, 8192, 0) представляет собой двудольный граф с 360+250 = 610 вершинами и 40 923 ребрами. Это другой взгляд на знаменитую картину Леонардо да Винчи, на этот раз связывающий строки и столбцы, на пересечении