Нелли **Литвак** /
Андрей **Райгородский**

КОМУ *нужна* МАТЕМАТИКА?

ПОНЯТНАЯ КНИГА О ТОМ, КАК УСТРОЕН ЦИФРОВОЙ МИР

Эту книгу хорошо дополняют:

Удовольствие от *x*
Стивен Строгац

Голая статистика
Чарльз Уилан

Теория игр
Авинаш Диксит, Барри Нейлбафф

Как не ошибаться
Джордан Элленберг

Нелли Литвак,
Андрей Райгородский

Кому нужна математика?

Понятная книга о том, как устроен цифровой мир

Москва
«Манн, Иванов и Фербер»
2017

УДК 512
ББК 22.1я9
Л64

Литвак, Нелли

Л64 Кому нужна математика? Понятная книга о том, как устроен цифровой мир / Нелли Литвак, Андрей Райгородский. — М. : Манн, Иванов и Фербер, 2017. — 192 с.

ISBN 978-5-00100-521-6

Если вы хотите найти ответ на вопрос «Зачем мне математика?», эта книга для вас. В ней рассказывается о современных приложениях математики, без которых невозможно существование авиации, страхования, железных дорог, медицины, интернета, экономики… Список можно продолжать долго, но проще будет сказать — невозможно существование современного мира, каким мы его знаем.

Эта книга будет полезна широкому кругу читателей, но для наиболее заинтересованных и подготовленных читателей авторы добавили дополнительные сведения, объединив их в специальном приложении.

УДК 512
ББК 22.1я9

ISBN 978-5-00100-521-6

Оглавление

Введение

О чем эта книга

В этой книге мы расскажем о некоторых современных приложениях математики. Мы выбрали семь тем, по одной на каждую главу:

1) задачи планирования и составление расписаний (глава «Менеджмент и многогранники»);
2) кодирование текстов для хранения и передачи в цифровом виде (глава «Мир нулей и единиц»);
3) структура соединения серверов каналами связи в интернете (глава «Надежность интернета»);
4) балансирование нагрузки в телекоммуникациях (глава «Сила выбора из двух»);
5) шифрование конфиденциальных сообщений (глава «Секретные числа»);
6) операции подсчета при анализе больших данных (глава «Счетчики с короткой памятью»);
7) распределение рекламных мест в поисковых системах, таких как Google и «Яндекс» (глава «Миллион аукционов в минуту»).

Мы ни в коем случае не претендуем на хоть сколько-нибудь полный обзор бесчисленных приложений математики. На это понадобится не одна книга, а целая библиотека! Например, мы вообще не упоминаем о медицинских приложениях, скажем, о создании подвижной трехмерной картинки на экране при компьютерной томографии или

нахождении мутаций в геномах клеток рака. Мы также не касались приложений в высокотехнологичном производстве, например в машиностроении и авиации, и широкого спектра применений в экономике.

Выбранные нами темы объединены одной идеей. Все они связаны с компьютерами и интернетом. Мы хотим на ярких и конкретных примерах показать, что сам по себе компьютер, даже самый мощный, не способен творить ставшие уже столь привычными для нас чудеса: показывать тексты и фотографии, делать сложные расчеты, искать информацию и пересылать данные по всему миру. За всем этим стоит математика; без нее компьютер остался бы просто безжизненным и практически бесполезным устройством.

Конечно, даже эту тему невозможно полностью раскрыть в одной книге. Мы выбрали лишь несколько наиболее близких к теме наших собственных исследований примеров. Если бы эту книгу писали другие авторы, то и темы были бы другими, но не менее впечатляющими.

Мы очень хотим разделить с вами если не наше увлечение, то хотя бы наше восхищение математикой — точной и красивой, древней и всегда современной и, безусловно, невероятно полезной!

Для кого эта книга

Эта книга написана для широкого круга читателей и не требует специальной подготовки. При этом для наиболее заинтересованных и подготовленных читателей мы добавили, кроме основного текста, дополнительные объяснения и приложения.

Основной текст. Основной текст книги не требует абсолютно никакой математической подготовки. Мы постарались писать его так, чтобы он был интересен и понятен каждому

читателю. В каком-то смысле наш рассказ можно сравнить с научно-популярными телевизионными программами.

Дополнительные объяснения. Иногда для интересующегося читателя мы объясняем основные идеи чуть более подробно. Этот текст мы приводим во врезках. В основном он не требует математической подготовки. Но при желании его можно пропустить без ущерба для понимания остального содержания главы.

Приложения*. В конце книги мы поместили приложения к каждой главе, где приводим более строгие математические формулировки, включая формулы, теоремы и доказательства (или хотя бы идеи доказательств). Сноски на них ищите в соответствующих главах. Приложения рассчитаны на уровень старшеклассников, увлекающихся математикой.

---

* Мы включили в книгу приложения для подготовленного читателя, чтобы ее можно было использовать в качестве учебника, например для спецкурса в старших классах или для вводных лекций в вузе.

Глава 1
«Кому-то еще нужна математика?»

Нелли

После долгого перелета и полуторачасового стояния в очереди я наконец предъявляю паспорт и кладу указательный палец на маленький сканер в аэропорту Атланты. Унесенные ветром...

— С какой целью вы приехали в США? — в голосе пограничника нет ни капли интереса.

— Я приехала на конференцию.

— Какую?

— По математике.

Со мной все ясно, пограничник ухмыляется и берется за печать:

— И что, кому-то еще нужна математика?

Нет, вы только подумайте! Он сидит в аэропорту Атланты, где буквально каждые десять минут приземляется самолет, сканирует мой паспорт, компьютер в долю секунды находит мои отпечатки пальцев среди миллионов других отпечатков и сравнивает мой файл с сотнями тоненьких линий на маленьком сканере. Каким образом, хотелось бы знать, было решено, во сколько, куда и какой самолет должен садиться? Как сохранить отпечатки пальцев в компьютере, который хоть и показывает на экране всякие картинки, на самом деле не умеет хранить ничего, кроме нулей и единиц? Как быстро найти нужную запись среди миллионов других? И как компьютер — кучка пластмассы и железа — может решить, совпадают ли две картинки, отфильтровав при этом неизбежные помехи и неточности?

Можно копать и дальше, разбираясь в конструкции самолета (максимальная прочность при минимальном весе), вникать в таинственную систему определения стоимости билетов и так далее и тому подобное. И ничего этого — заметьте, абсолютно ничего! — нельзя было бы сделать без математики. Самолеты, цены, линии на пальцах — все это описано с помощью переменных, функций и уравнений. И для всех этих задач найдены эффективные и точные способы решения. Потому что у компьютера нет глаз, и при этом он должен узнать мой отпечаток пальца быстро и безошибочно. Целая команда математиков могла бы работать всю жизнь, занимаясь исключительно проблемами, связанными с аэропортом Атланты.

Я беру свой паспорт и улыбаюсь пограничнику:

— Конечно нужна!

Развивать дискуссию бесполезно. И потом, при всей бессмысленности вопроса не его вина, что он представляет себе математику как бесконечный ряд никому не нужных экзерциций с числами и формулами. И уж конечно, не он один такой.

Недавно я обсуждала эту ситуацию на занятии со студентами-математиками. Вас спрашивают: где вы учитесь? Вы отвечаете: на прикладной математике. Вас спрашивают: и зачем это нужно? В аудитории смущенные понимающие улыбки. Каждый не раз слышал этот вопрос.

Я считаю, что математика должна быть либо красивой, либо полезной. А лучше — как это часто бывает в настоящей науке — и то и другое! Наверное, без специальной подготовки красоту математики понять довольно сложно. Но мне кажется удивительным, что в эпоху цифровых технологий широкой публике так мало известно о невероятной полезности математики. Скептицизм американского пограничника — скорее правило, чем исключение. В этой книге мне хотелось понятно и интересно рассказать именно о пользе математики. Ну и о красоте, конечно, тоже. Надеемся, читатель сможет ее увидеть и оценить.

Андрей

В моей семье многие имели отношение к математике. Мама с папой, например, познакомились в МИИТе, где мама училась на факультете прикладной математики, а папа — на автоматизации систем управления (так тогда называли программистские факультеты). Папа в свое время учился в знаменитой 2-й школе. А мамин папа, мой дедушка, перед самой войной окончил мехмат МГУ и потом всю жизнь работал над расчетами траекторий космических аппаратов (скажем, тех же первых луноходов) — сначала у Королева в Подлипках, потом у Лавочкина в Химках. Он, пожалуй, и оказал на меня наибольшее влияние. Я тоже учился на мехмате МГУ. Там на мой выбор математики в качестве профессии радикально повлиял мой научный руководитель — Николай Германович Мощевитин.

До мехмата я учился в школе с французским уклоном и любил многие предметы. Меня интересовали языки — в том числе с точки зрения лингвистики. В старших классах я имел возможность сменить школу на школу с математическим уклоном, но сознательно предпочел остаться и доучить французский.

Я не считаю, что математика — это естественная наука, как физика, химия или биология. Это некий вид искусства. Знаменитый математик Эрдеш говорил, что у Бога есть книга, в которой содержатся идеальные математические доказательства, «доказательства из книги». Я тоже думаю, что математика открывает истины, содержащиеся в идеальном мире, и только потому она и имеет приложения, что видит «высшую реальность», проекцией которой служит этот мир. Иными словами, не математику оправдывают ее приложения (она прекрасна сама по себе), они возникают за счет того, что так устроен мир, и математика как раз об этом, об устройстве мира.

Моя наука — комбинаторика — замечательна тем, что делает очень многие формулировки и доказанные сложные результаты понятными даже школьнику, интересующемуся математикой.

Поэтому рассказывать о ней исключительно круто. Однако эта наука богата и задачами, которые при всей простоте своих постановок пока совершенно не поддаются решению. В книге мы рассмотрим некоторые из них: они до сих пор остаются открытыми, несмотря на их актуальность.

Лучший ответ на вопрос «Кому нужна математика?»

Пожалуй, приз за лучший ответ на вопрос «Кому нужна математика?» можно смело отдать выдающемуся немецкому математику по имени Мартин Гротшел. Не гарантируем точности изложения, но байка, которую рассказывают на конференциях, звучит так:

Как-то раз немецкое правительство решило выделить целевым образом значительные суммы на развитие самых передовых и необходимых областей науки. На заседание государственной комиссии были приглашены физики, химики, биологи — представители всех наук. Гротшел представлял математику. Все ораторы с огромным энтузиазмом рассказывали о необыкновенных достижениях своей науки и том, как без нее мир и Германия рухнут. Естественно, все докладчики выходили за рамки отпущенного времени. Гротшел выступал последним. Заседание уже подходило к концу, чиновники сидели осоловевшие от обрушенного на них потока информации. Гротшел вышел на трибуну и сказал примерно следующее:

— Уважаемые господа! Я не буду утомлять вас длинной речью, а просто приведу пример. Недавно мы получили заказ от большой страховой компании, планирующей создать автосервис для своих клиентов. Идея очень проста: если у клиента в дороге сломалась машина, он может позвонить

по телефону и к нему тут же приедет аварийная служба. Вопрос в том, как правильно организовать такой сервис. В принципе, задачу можно решить довольно просто — например, приставить к каждому клиенту личную аварийную машину с механиком. Тогда клиент в любой момент немедленно получит помощь. Но это очень дорого! Другой вариант — вообще не связываться с аварийным сервисом. Клиенты могут ждать до бесконечности, зато это не будет стоить им ни цента. Так вот. Если вас эти решения не устраивают, то я должен вам сообщить, что для любых других вариантов понадобится математика! Спасибо за внимание.

Нужно ли говорить, что математика получила колоссальные правительственные субсидии. Результаты этих инвестиций во всех областях, от транспорта до медицины, абсолютно потрясающие!

Кстати, среди студентов Нелли приз за лучший ответ получила Клара, которая сказала, что без математики невозможно было бы составить расписание поездов и они все время сталкивались бы друг с другом. О расписаниях поездов мы подробнее расскажем в главе 2, а пока немножко поговорим о том, чем занимаются профессиональные математики, от выпускников вуза в компаниях до ведущих ученых-теоретиков.

Математика на каждый день

На выпускников с дипломом математика в Европе большой спрос. Даже средненькие студенты легко находят работу. Причем они далеко не всегда становятся программистами, даже если их компания и производит программное обеспечение. Оптимальный красивый код — это задача инженеров-программистов. Задача математиков — придумать методы решения проблемы.

Сфера деятельности математиков очень широкая: логистика, планирование, высокотехнологичное производство, биомедицинские технологии, финансы.

Бывший коллега Нелли защитил диссертацию по финансовой математике, а потом пошел работать в компанию. «Мы управляем активами пенсионных фондов на рынке ценных бумаг. Многие думают, что это занятие типа купи-продай. А я тут сижу и целыми днями решаю дифференциальные уравнения. И ребята, которые торгуют, сидят тут же, в трех метрах от меня. Вот сейчас досчитаю и скажу им, что покупать».

Среди ученых-математиков есть те, кто напрямую работает с приложениями. Мор Харкол-Балтер из университета Карнеги — Меллон говорит, что все ее исследования основаны на приложениях. Например, в 2011 году она сотрудничала с «Фейсбуком». По оценкам Мор, «Фейсбук» задействовал свои включенные серверы не более чем наполовину, а остальное время они простаивали. Включенный и незадействованный сервер тратит примерно две трети энергии работающего сервера. Но компании боятся выключать серверы, потому что чем их больше, тем быстрее они справляются с запросами пользователей. При этом на включение сервера уйдет 4–5 минут, а «Фейсбук» хочет выполнять запрос за полсекунды! Однако Мор не сомневалась, что серверы можно спокойно отключать. Из математической теории — *теории массового обслуживания* — ясно следовало, что если серверов много (а у «Фейсбука» их очень много!), то время, затраченное на включение, не оказывает никакого влияния. Мор и ее ученики разработали метод, при котором серверы включались и выключались без какого-либо ущерба для пользователей. «Фейсбук» последовал рекомендациям и, по утверждению компании, теперь экономит 10–15 % энергии.

Профессора университета Твенте Ричард Бушери и Эрвин Ханс и их ученики занимаются логистикой здравоохране-

ния. В результате их исследований в больницах Нидерландов произошли существенные изменения. Например, больница в Роттердаме раньше всегда держала наготове специальную операционную для экстренных операций. Бóльшую часть времени операционная пустовала, драгоценное время тратилось впустую. Но менеджмент опасался, что в противном случае экстренным пациентам придется ждать слишком долго. При этом им все равно приходилось ждать, скажем, если вдруг привозили сразу двух экстренных пациентов. Математические подсчеты показали, что правильно составленное расписание плановых операций (еще одна нетривиальная задача!) позволяет быстро принять практически всех экстренных пациентов. В результате экстренную операционную упразднили и отдали под плановые операции.

Многие математики работают с приложениями, но далеко не все настолько вплотную, как в приведенных выше примерах. Разработка новых теорий важна для практики не меньше, чем решение непосредственных практических задач. Об этом мы поговорим подробнее в следующих разделах.

Новые теории для современной практики

В 2008 году международное статистическое сообщество отпраздновало столетие со дня появления распределения Стьюдента. Стьюдент — это псевдоним очень талантливого математика по имени Вилльям Госсет. Госсет работал на пивоваренном заводе «Гиннесс» в Дублине. Его исследования в области статистики имели чисто коммерческие цели: они применялись при тестировании качества сырьевых продуктов, из которых делали пиво. Госсету не разрешалось публиковать труды по статистике под собственным именем,

поэтому он публиковался под псевдонимом Стьюдент. Госсет вывел новое распределение вероятностей (распределение Стьюдента) и на его основе разработал теперь уже классическую статистическую процедуру, знаменитый *t*-тест.

t-тест обычно используется при необходимости сравнить случайную выборку с какой-то нормой или две случайные выборки между собой. Например, вы выпускаете шурупы и хотите проверить, соответствуют ли они норме по длине. Или вам нужно сравнить урожайность при использовании двух разных видов удобрений. Такие тесты широко применяются на практике, для них разработано стандартное программное обеспечение, *t*-тест не проходят разве что на филфаке.

За 100 лет статистика ушла далеко вперед. Сара ван де Гейр, профессор Швейцарской высшей технической школы Цюриха, работает над тестами с многомерными данными. Задача, так же как и задача Госсета, пришла из практики. Компания DSM в Швейцарии выпускает витамины и пищевые добавки. Витамин B_2 производится с помощью бациллы сенной палочки. Компания хочет увеличить выпуск витамина благодаря генной инженерии. Имеются измерения производительности 115 бактерий, генный состав которых включает 4088 возможных генов. Спрашивается, какие гены способствуют росту производства витамина B_2?

Это очень сложная задача, учитывая, что данных мало, а параметров много, причем все они взаимосвязаны. Существующие теории для этого случая не подходят, поэтому Сара и ее сотрудники сосредоточились на создании новых теорий. Это очень сложная математика, доступная только специалистам. Но то же самое сто лет назад можно было сказать и о работе Госсета! И мы совершенно не удивимся, если статистические процедуры, разработанные Сарой, через пару десятков лет займут свое место в университетских учебниках

по статистике и задачка про сенную палочку будет предложена студентам-биологам на экзамене. Когда мы поделились этими мыслями с Сарой, она абсолютно серьезно сказала: «Конечно, очень скоро это будет стандартная статистика».

Поскольку современная реальность постоянно усложняется, существующего математического аппарата часто не хватает. И это, безусловно, мощный стимул для появления новых задач и теорий.

Математика неизвестного будущего

Не все математические задачи взяты из практики. Так и должно быть, потому что мы не можем с уверенностью предсказать пути развития общества и технологий даже в ближайшем будущем. Это не по силам даже самым информированным людям с совершенно неуемной фантазией. Например, хорошо известно, что писатели-фантасты практически ничего не сумели предугадать. В основном они описывали технологии своего времени, приукрашивая их фантастическими деталями.

Никто не предрек появления интернета. Наоборот, Нобелевский лауреат Деннис Габор, изобретатель голографии, в 1962 году заявил, что передача документов по телефону хоть и возможна в принципе, но требует таких огромных расходов, что эта идея никогда не найдет практического воплощения. При этом первый успешный модем был представлен в том же году! А Кен Олсен, один из создателей Digital Equipment Corporation (DEC), в 1977 году сказал, что вряд ли найдется человек, которому может дома понадобиться компьютер. Через сколько лет после этого компьютер появился в вашем доме?

Никто не знает, какая абстрактная теория завтра может найти практическое применение. Потрясающий пример — теория чисел, область математики, изучающая числа и их закономерности. Теория чисел оставалась абстрактной наукой со времен Древней Греции до второй половины XX века. Сегодня эта теория широко используется для шифрования сообщений, передаваемых через интернет. Именно благодаря ей сохраняется конфиденциальность ваших паролей и номеров кредитных карточек, когда вы вводите их на многочисленных сайтах. Мы расскажем об этом подробнее в главе 7.

Наконец, нам трудно удержаться от еще одного варианта ответа на вопрос, зачем нужны новые сложные теории. Да просто ради красоты этих теорий! Красивая математика имеет полное право на существование. В научном мире должно оставаться что-то от Касталии Германа Гессе, где ученым разрешено заниматься чем угодно, где целью жизни может стать «игра в бисер»* — «самая блистательная и самая бесполезная». Почему? Потому что нельзя поставить науку полностью на службу материальным нуждам общества. Наука выполняет функцию просветительства. Это единственная сфера деятельности, в которой человек может работать, движимый исключительно непрактическим любопытством. Грубо говоря, наука делает мир умнее и нужна человечеству так же, как и искусство, которое делает мир более духовным.

---

* Отсылка к знаменитому одноименному роману лауреата Нобелевской премии Германа Гессе. *Прим. ред.*

Глава 2
Менеджмент и многогранники

Компьютерные будни логистики

На специальности «Прикладная математика» в основном обучают математике. Доля программирования не так уж велика по сравнению с бесконечным матанализом, алгеброй и матфизикой. При этом выпускники часто становятся программистами.

— Интересно, насколько тебе нужна вся эта математика? — спросила Нелли у друга и бывшего однокурсника, а сегодня системного администратора в международной компании. Тот не задумался ни на секунду:

— Конечно нужна! Вот недавно клиенты заказали программу для распределения товаров по вагонам. Мы сразу поняли, что такую задачу ежедневно решают все поставщики всех товаров. Значит, она известная. Через полчаса мы уже знали, что это «задача об упаковке», и могли предложить несколько решений. Кстати, клиентам пришлось объяснять, что задача NP-трудная, то есть мы не можем гарантировать самое лучшее из всех возможных решений. И они согласились. А что им оставалось?

Вся современная логистика основана на математических методах. Где расположить склады и сервисные пункты? Как распределить товары по вагонам и грузовикам и какими маршрутами все это отправить? Сколько товара держать

на складе и как часто его пополнять? Как составить расписание поездов, самолетов, большого производства и даже спортивных соревнований?

Этими вопросами занимается область прикладной математики под названием исследование операций. По большому счету это наука о том, как оптимально организовать процессы бизнеса и производства. Сюда, безусловно, относится логистика, а также многие другие задачи, например из области финансов или телекоммуникаций.

Исследование операций начало развиваться относительно недавно, после Второй мировой войны. И далеко не сразу научные результаты нашли практическое применение. В 2002 году в специальном юбилейном выпуске в честь 50-летия журнала «Исследование операций» Чарльз Холт делится своими воспоминаниями о том, как он и его коллеги Франко Модильяни, Джон Муф и Герберт Симон разрабатывали и внедряли научные методы планирования производства:

> Мы взяли интервью у менеджеров пятнадцати компаний. Поначалу менеджеры отрицали наличие каких-либо проблем. По крайней мере таких, с которыми коллеги-профессора могли хоть как-то помочь. Но когда мы расспрашивали более подробно, возникали картины наподобие телеги на разваливающихся колесах — системы, катящиеся без всякого контроля от одного кризиса к другому [18].

В процессе работы все менеджеры постепенно переключились на систему «коллег-профессоров». Команда написала книгу «Планирование производства, инвентаря и трудовых ресурсов». Модильяни и Симон получили Нобелевские премии по экономике, а работы Муфа легли в основу исследований Роберта Лукаса, тоже впоследствии лауреата Нобелевской премии.

Методы исследования операций глубоко внедрились в современный бизнес. Никому не придет в голову планировать большое производство или составлять расписание самолетов вручную. Для этого есть подготовленные специалисты и стандартное коммерческое программное обеспечение. Даже самый элементарный подход в рамках исследования операций всегда превзойдет любое решение «на глазок». Исследование операций преподают не только на факультетах прикладной математики, но и в бизнес-школах.

В этой главе мы расскажем о задачах оптимизации, которые, в частности, возникают при планировании и составлении расписаний.

Проклятие размерности

Сложность задач оптимизации заключается в невообразимом множестве возможных решений. Чтобы продемонстрировать масштаб проблемы, давайте посмотрим на самый простой вариант расписания.

У нас есть один прибор, на котором нужно выполнить 25 заданий. Спрашивается: в каком порядке выгоднее всего это делать? «Выгода» может зависеть от срока выполнения, времени, проведенного в очереди, и других факторов.

Задача непростая, о ней написана не одна диссертация. Но, допустим, мы решили поступить наипростейшим образом. Берем самый мощный компьютер и пишем программу, которая считает прибыль и убытки для каждой возможной последовательности заданий. После этого выбираем наиболее выгодную последовательность.

Теоретически все правильно. Но прежде чем запустить программу, давайте посчитаем, сколько разных последовательностей ей придется перебрать.

На первое место можно поставить любое из 25 заданий. Для каждого из 25 вариантов для первого места у нас есть 24 варианта для второго места. Получается, что первые два места можно заполнить

$$25 \times 24 = 600$$

способами. Продолжаем: 23 варианта для третьего места, 22 — для четвертого и так далее. Всего у нас получается

$$25 \times 24 \times 23 \times 22 \times 21 \times 20 \times 19 \times 18 \times 17 \times 16 \times 15 \times 14 \times 13 \times 12 \times 11 \times 10 \times \\ \times 9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times = 15511210043330985984000000$$

способов.

Это число называется *двадцать пять факториал* и обозначается «25!». Насколько оно велико? Если взять современный процессор с тактовой частотой 2 ГГц (2 млрд операций в секунду), то для выполнения такого количества операций ему понадобится 245 млн лет! А на то, чтобы просчитать все варианты, с прибылью и убытками, да еще и перемещать информацию в памяти компьютера, — и того больше. А ведь задачка казалась совсем простой, всего один прибор, всего 25 заданий. Не сравнить с серьезным современным производством.

Такое явление называется *проклятием размерности*. Даже при скромном количестве вводных данных степень свободы в выборе решения колоссальна. Перебрать все варианты просто невозможно. Значит, понадобятся другие подходы, более умные и нетривиальные, и именно для этого нужна математика.

Для некоторых задач удается найти гарантированно лучший ответ относительно быстро. Но для целого разряда так называемых *NP-трудных* задач, как, например, упомянутая выше задача об упаковке, сложно придумать метод, который работал бы намного быстрее, чем тривиальный полный перебор всех вариантов. Удастся ли когда-нибудь? Это открытый вопрос, но большинство ученых считают,

что нет, потому что таких методов просто не существует. Многие практические задачи NP-трудные. В этом случае математики стремятся к быстрым и «почти» оптимальным решениям. А на практике приходится мириться с тем, что ответ достаточно хороший, но не всегда самый выгодный из возможных.

Разных методик для разных задач придумано множество. Мы расскажем о *линейном программировании*. Это мощная и уже ставшая классической теория, которая невероятно успешно применяется на практике.

Линейное программирование

Как возникают задачи линейного программирования, мы объясним на еще одном простом примере.

Допустим, у нас есть два склада: на северном и на южном конце города. В офис поступили заказы от двух клиентов. Клиент А заказал 60 листов железа, а клиент Б — 40 листов. На южном складе у нас в наличии 70 листов железа, а на северном — 35, то есть общего запаса хватает. Но мы хотим свести расходы на доставку к минимуму. Цены доставки приведены в табл. 2.1.

Таблица 2.1. Пример цен доставки

	Южный склад	Северный склад
Цена доставки клиенту А	5 руб. за лист	7 руб. за лист
Цена доставки клиенту Б	10 руб. за лист	15 руб. за лист
Количество листов железа в наличии	70	35

Спрашивается: сколько листов отправить клиентам А и Б с южного склада, а сколько — с северного?

Все было бы просто, если бы мы могли доставить весь товар с «дешевого» южного склада. Но, к сожалению, там всего 70 листов, на обоих клиентов не хватит. А поскольку северный склад гораздо дороже, решение не очевидно.

Как же его найти? В нашем конкретном примере, в принципе, можно пойти путем перебора всех вариантов. Но если количество клиентов и складов увеличится, решить задачу вручную не удастся. Поэтому давайте посмотрим, как это сделать с помощью математики. Для начала, как учили в средней школе, введем переменные.

Клиенту А с южного склада доставлено АЮ листов железа. Тогда с северного склада клиенту А доставлено (60 – – АЮ) листов. Аналогично клиенту Б с южного склада доставлено БЮ листов железа, а с северного — (40 – БЮ) листов. Теперь по табл. 2.1 можно рассчитать общую стоимость доставки:

$$5 \times АЮ + 7 \times (60 - АЮ) + 10 \times БЮ + 15 \times (40 - БЮ) \text{ (рублей)}.$$

Если раскрыть скобки, то получается:

$$\begin{aligned} &\text{общая стоимость доставки} = \\ &= 1020 - 2 \times АЮ - 5 \times БЮ \text{ (рублей)}. \end{aligned} \tag{2.1}$$

Нам нужно выбрать АЮ и БЮ так, чтобы стоимость была как можно меньше.

Но это еще не все. АЮ и БЮ нельзя выбрать просто так. В задаче есть существенные *ограничения*. Во-первых, мы не будем отправлять клиентам больше листов, чем они просили. Клиент А заказал 60 листов, а клиент Б — 40 листов. Поэтому в любом случае

$$АЮ \le 60, \tag{2.2}$$

$$БЮ \le 40. \tag{2.3}$$

Во-вторых, нужно учесть, что запас на каждом складе ограниченный. С южного склада мы отправляем АЮ + БЮ листов, а всего на этом складе 70 листов. Поэтому АЮ + БЮ не больше 70:

$$АЮ + БЮ \leq 70. \quad (2.4)$$

Аналогично с северного склада мы не можем отправить больше 35 листов:

$$(40 - АЮ) + (60 - БЮ) \leq 35.$$

Раскрыв скобки в этом выражении, получаем:

$$АЮ + БЮ \geq 65. \quad (2.5)$$

Это ограничение можно интерпретировать еще и так: поскольку на северном складе 35 листов, а нам в совокупности необходимо доставить 100 листов, то как минимум 65 листов должны быть доставлены с южного склада.

Вот теперь все! Это и есть *задача линейного программирования*: нам нужно минимизировать стоимость, которая задана выражением (2.1), и при этом соблюсти ограничения (2.2)–(2.5). Внизу, во врезке, задача приведена в окончательном варианте.

Задача линейного программирования

Выбрать АЮ и БЮ так, чтобы минимизировать:

$1020 - 2 \times АЮ - 5 \times БЮ,$

при ограничениях:

$0 \leq АЮ \leq 60,$

$0 \leq БЮ \leq 40,$

$АЮ + БЮ \leq 70,$

$АЮ + БЮ \geq 65.$

Мы добавили, что АЮ и БЮ либо ноль, либо больше нуля, потому что доставить клиенту отрицательное количество листов железа невозможно.

Программирование в данном контексте скорее «оптимизация», а не программирование на компьютере. Слово *линейное* употребляется потому, что используются исключительно линейные выражения, то есть переменные можно умножать на число, а также вычитать и складывать. И все. Никакие другие операции не применяются. Например, у нас нет выражений типа АЮ × БЮ или АЮ^2. Оказывается, в такой линейной формулировке можно представить очень многие задачи оптимизации.

В практических задачах переменных и ограничений намного больше. При этом всегда есть только *одно* выражение, так называемая *целевая функция*, которую следует либо минимизировать (если речь идет о стоимости), либо максимизировать (если речь идет о доходе). В данном случае наша целевая функция — стоимость (2.1), и ее нужно минимизировать.

Теория для практики

Пионер и основатель теории линейного программирования — советский ученый Леонид Витальевич Канторович. Над подобными проблемами он работал в конце 1930-х годов. В 1940-м вышла его фундаментальная статья «Об одном эффективном методе решения некоторых классов экстремальных проблем» [3]. В ней Канторович заложил математические основы линейного программирования (правда, тогда оно еще так не называлось).

Канторович интересовался этими проблемами прежде всего из-за их практической ценности. В 1975 году он получил Нобелевскую премию «за вклад в теорию оптимального распределения ресурсов», которую разделил с американским экономистом-математиком голландского происхождения Тьяллингом Купмансом, тоже занимавшимся разработкой теории линейного программирования и ее приложениями в экономике.

Одна из классических знаменитых задач линейного программирования — задача о диете Стиглера, датируемая 1945 годом. Звучит она примерно так: какие из 77 продуктов должны входить в потребительскую корзину одного человека (скажем, мужчины среднего веса), чтобы он получил необходимую норму девяти питательных веществ (включая калории) и при этом стоимость продуктов была минимальной? Это очень важная задача в экономике, потому что ее решение определяет минимальную потребительскую стоимость полноценного питания.

В математической формулировке переменные — это количество каждого продукта. Содержание белков, жиров, витаминов, минералов в каждом продукте известно. Ограничения — это минимальное количество питательных веществ. А минимизировать надо общую стоимость продуктов, которая складывается из количества каждого продукта, помноженного на его цену.

Уже к концу 1950-х линейное программирование достаточно широко использовалось в нефтяной индустрии. Сегодня оно лежит в основе огромного класса задач оптимизации, включая задачи менеджмента и микроэкономики: планирование, логистика, составление расписаний. Задачи, где нужно минимизировать стоимость или максимизировать доход при заданных ограничениях.

От задачи к решению

Несмотря на простую формулировку, решить задачу линейного программирования вовсе не просто. Самая большая сложность заключается в ограничениях. Это видно даже на нашем маленьком примере. Понятно, что выгоднее всего доставить товар обоим клиентам с дешевого южного склада.

Трудность в том, что это невозможно, потому что там всего 70 листов, а нам нужно 100.

Чем больше переменных и ограничений, тем сложнее задача. В классической задаче о диете 77 переменных и 9 ограничений, и она уже представляет собой серьезную проблему с точки зрения вычислений. Линейное программирование стало рядовым инструментом менеджмента и планирования только благодаря тому, что математики придумали для таких задач множество совершенно нетривиальных методов решения.

Работы американского математика Джорджа Данцига появились в конце 1940-х годов — несколько позже, чем работы Канторовича. Тем не менее Данцига тоже по праву относят к основателям линейного программирования. Именно он придумал так называемый симплекс-метод, позволивший с помощью компьютера быстро решать задачи линейного программирования с большим количеством переменных и ограничений.

Симплекс-метод, сильно улучшенный и усиленный другими методами, по-прежнему остается неотъемлемой частью современного программного обеспечения.

Идея симплекс-метода

Подробности симплекс-метода выходят за рамки этой книги, но мы постараемся объяснить его суть на нашем маленьком примере.

Для начала давайте посмотрим, какие в принципе значения могут принимать переменные, чтобы не нарушить наших ограничений. Например, мы можем взять АЮ = 58, БЮ = 8. В этом случае получается решение, которое мы записали в виде табл. 2.2.

Таблица 2.2. Пример решения, где АЮ = 58, БЮ = 8

	План поставки листового железа				
	Заказано	**Южный склад**	**Северный склад**	**Всего**	**Стоимость**
Кол-во листов клиенту А	60	58	2	60	$5 \times 58 + 7 \times 2 = 304$ руб.
Кол-во листов клиенту Б	40	8	32	40	$10 \times 8 + 15 \times 32 = 560$ руб.
Всего	100	66	34	100	864 руб.

Ограничения выполнены, и оба клиента получили заказанное количество листов.

Но это не единственное решение. Например, мы могли отправить больше листов с дешевого южного склада клиенту А, скажем 60 листов, и 10 листов клиенту Б. Легко увидеть, что доставка клиенту А теперь обойдется в

$$5 \times 60 = 300 \text{ руб.},$$

а доставка клиенту Б будет стоить

$$10 \times 10 + 30 \times 15 = 550 \text{ руб.}$$

Тогда общая стоимость получается не 864, а 850 рублей, то есть немного меньше, чем указано в табл. 2.2.

Чтобы не выбирать наугад, нужно посмотреть на все возможные решения, которые удовлетворяют ограничениям. Мы их изобразили на рис. 2.1. По оси *x* мы откладываем АЮ, а по оси *y* — БЮ. Любая точка в заштрихованной области удовлетворяет ограничениям. В том числе точка (58,8), как в таблице выше.

Ниже во врезке мы объясняем, как получилась заштрихованная область. Объяснения соответствуют уровню средней школы. При желании их можно пропустить.

Как получена заштрихованная область на рис. 2.1

Все значения АЮ и БЮ положительные.

Вертикальная прямая линия АЮ = 60 обеспечивает ограничение АЮ ≤ 60. Все возможные значения находятся либо на ней, либо слева от нее.

Аналогично горизонтальная прямая линия БЮ = 40 обеспечивает ограничение БЮ ≤ 40. Все возможные значения находятся либо на ней, либо под ней. Выражение штрихпунктирной прямой АЮ + БЮ = 70 можно переписать в более привычном виде:

$$\text{БЮ} = 70 - \text{АЮ},$$

поэтому прямая идет под отрицательным углом 45°. Заметьте, что она пересекает ось *x* (в нашем случае ось АЮ), когда БЮ = 0 и, соответственно, АЮ = 70. Нам нужно, чтобы выполнялось неравенство АЮ + БЮ ≤ 70, то есть

$$\text{БЮ} \leq 70 - \text{АЮ}.$$

Значения, удовлетворяющие этому неравенству, расположены на штрихпунктирной прямой или *под* ней.

Аналогично пунктирная прямая АЮ + БЮ = 65 обеспечивает ограничение АЮ + БЮ ≥ 65. Значения, которые удовлетворяют этому неравенству, находятся на этой прямой или *над* ней.

Заштрихованная область, включая границы, удовлетворяет всем ограничениям.

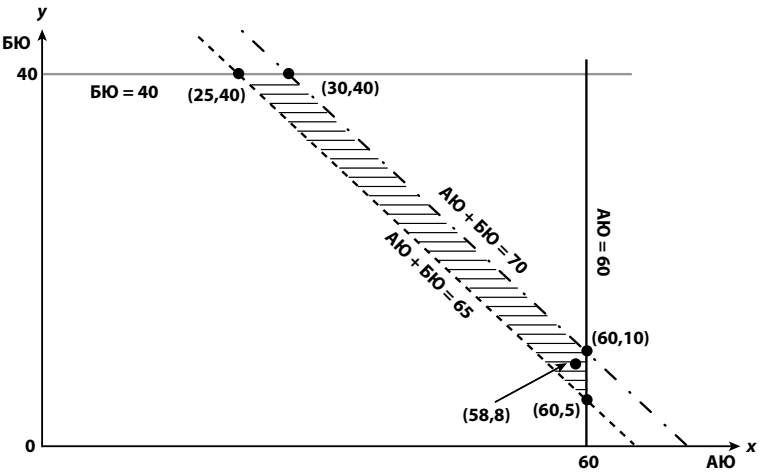

Рис. 2.1. Решения, удовлетворяющие ограничениям

Примечание: любая точка в заштрихованной области удовлетворяет всем ограничениям. Точка (58,8) — это решение из таблицы выше. Угловые точки (25,40), (30,40), (60,10) и (60,5) — кандидаты на оптимальное решение (см. объяснение в тексте).

Важно отметить, что заштрихованная область — это четырехугольник с прямыми сторонами, поскольку все наши ограничения линейные, то есть их можно изобразить с помощью прямых линий. Данная область называется областью допустимых значений, потому что все значения в ней удовлетворяют всем ограничениям. Иначе говоря, любое решение из этой области физически возможно или допустимо.

Фундаментальное свойство задач линейного программирования заключается в том, что оптимальное решение обязательно находится в углах области допустимых значений. Это происходит потому, что наша стоимость тоже линейная. Когда мы движемся по прямой — горизонтальной, вертикальной или наклонной, — стоимость может либо только уменьшаться, либо только увеличиваться, пока мы не наткнемся на угол и идти дальше по той же прямой станет невозможно.

Для подготовленного читателя в приложении в конце книги мы приводим более формальное обоснование того, почему оптимальное решение задачи линейного программирования обязательно найдется в одном из углов области допустимых значений.

В нашем маленьком примере углов всего четыре: (25,40), (30,40), (60,10) и (60,5). Мы можем легко подставить значения и подсчитать, что самое лучшее решение в точке (30,40), то есть с южного склада нужно отправить 30 листов клиенту А и 40 листов клиенту Б. Оставшиеся 30 листов клиенту А следует отправить с северного склада. Результат приведен в табл. 2.3:

Таблица 2.3. Оптимальный план поставки листов железа

	Заказано	Южный склад	Северный склад	Всего	Стоимость
Кол-во листов клиенту А	60	30	30	60	$5 \times 30 + 7 \times 30 = 360$ руб.
Кол-во листов клиенту Б	40	40	0	40	$10 \times 40 = 400$ руб.
Всего	100	70	30	100	760 руб.

Общая стоимость — 760 рублей, что гораздо меньше, чем 864 рубля — наше первое, выбранное наобум решение. Выгода — 12 %, и это очень существенно, особенно если таких доставок много.

То, что решение нужно искать «по углам», сказано уже на первой странице работы Канторовича. Это понятно любому математику.

Если же ограничений много, то у нас получается уже не четырехугольник, а многоугольник. А если много переменных, у нас будет не многоугольник, а многогранник! Найти и перебрать все углы многогранника невероятно сложно, на это может уйти очень много времени.

Заслуга Данцига состоит в том, что он придумал способ, основанный на линейной алгебре, который позволяет перемещаться от одного угла многогранника к другому не наобум, а в определенном порядке, чтобы при переходе от одного угла к другому стоимость только уменьшалась. Это и есть знаменитый симплекс-метод, применяемый практически во всех приложениях. Доказано, что в самых худших случаях искать решение придется очень долго. Тем не менее на практике симплекс-метод в его современных вариантах быстро находит оптимальное решение.

Составление расписаний

В планировании часто приходится оперировать с целыми числами. Нельзя отправить на объект «два землекопа и две трети», как в стихотворении Маршака. В этом случае мы имеем дело с задачей целочисленного линейного программирования.

Такие задачи часто встречаются при составлении расписаний. Например, посмотрим на самый первый наш пример, в котором один прибор должен выполнить 25 заданий и нужно найти самую выгодную последовательность. Тогда мы можем ввести переменные x для каждой комбинации (*задание, очередность выполнения*). Если задание 3 выполняется самым первым, то мы пишем

$$x \text{ (задание 3, очередность 1)} = 1.$$

А если этого не происходит, то

$$x \text{ (задание 3, очередность 1)} = 0.$$

Каждая переменная в решении — это целое число: 0 или 1.

С помощью этих переменных можно записать стоимость любой последовательности и практически любые

ограничения. Типичное строгое ограничение: прибор не может выполнять два задания одновременно. Но можно добавить ограничения и посложнее. Например, «задание 3 нужно (или желательно) выполнить раньше, чем задание 10»*.

В реальности даже для составления относительно небольшого расписания имеет смысл воспользоваться математической моделью. Например, несколько лет назад студенты факультета прикладной математики Университета Твенте разработали модель для расписания ежегодного фестиваля хоров. Там несколько десятков хоров, несколько сцен, не каждый хор может петь на любой сцене, и у некоторых хоров один и тот же дирижер. Раньше организаторы бились над расписанием не один день. А компьютерная программа, которую написали студенты, выдавала решение буквально за несколько минут. Восхищенные певцы пришли в университет на презентацию проекта и спели студентам благодарственную арию!

Почему целые числа сложнее дробных

Потребность в целочисленном решении кардинально усложняет задачу. Симплекс-метод не даст готового решения, потому что координаты углов многогранника вовсе не обязаны быть целыми и обычно целыми не будут. Целочисленное линейное программирование относится к разряду NP-трудных задач.

Классический подход к решению называется методом ветвей и границ. Он основан на «разветвлении» дробных

* В приложении к главе 2 для подготовленного читателя мы более подробно рассматриваем еще один маленький пример составления расписаний.

решений на допустимые целочисленные и исключении неперспективных «веток»*.

Для практического применения наивное использование метода ветвей и границ абсолютно не годится. Математики дополнили его многими другими методами. Например, сейчас на практике широко применяется метод «отсекающих плоскостей» (алгоритм Гомори), позволяющий эффективно отсекать дробные значения.

За относительно недолгий срок своего существования эта область математики достигла невероятного прогресса.

Математика, обогнавшая компьютер

Американский ученый Роберт Биксби написал целую серию статей об истории линейного программирования. В нашем рассказе мы воспользовались его статьей 2012 года [11].

По словам Биксби, после нескольких усовершенствований симплекс-метода к 1953 году удалось решить вариант знаменитой задачи о диете с 71 переменной и 26 ограничениями.

Аппарат, на котором производились вычисления, не был компьютером в строгом смысле этого слова. Это был огромный численный прибор, программируемый с помощью перфокарт. Решение заняло 8 часов, бóльшая часть которых ушла на то, чтобы вручную вставлять перфокарты в машину. Настойчивости и терпению ученых остается только удивляться.

Сегодня для решения задач линейного программирования на практике в основном используется коммерческое программное обеспечение. Среди самых известных пакетов — CPLEX и более недавний Gurobi.

---

* В приложении 3 к главе 2 мы объясняем для заинтересованного читателя основные идеи этого метода более подробно. Заметим, что это объяснение не требует математической подготовки.

В компьютерных технологиях действует одно эмпирическое правило, часто называемое законом Мура, согласно которому мощность процессоров удваивается каждые 18 месяцев. И хотя это не закон природы и Мур (один из основателей Intel) говорил не совсем об этом, все же данное утверждение примерно соответствует действительности.

Абсолютно очевидно, что любой алгоритм на современном компьютере будет работать гораздо быстрее, чем на давно устаревшей махине с перфокартами. Но и математика не стоит на месте. Старые алгоритмы совершенствуются, появляются новые. Как оценить роль математики в успехе коммерческих пакетов?

В 2007 году Биксби провел впечатляющий эксперимент. Он взял все версии пакета CPLEX, начиная с его первого появления в 1991 году, и опробовал их на большом количестве известных практических задач целочисленного линейного программирования. Ученые собрали внушительные коллекции таких задач. Биксби выбрал из них 1892, а затем сравнил скорость их решения, от версии к версии, на одном и том же компьютере.

Оказалось, что за 15 лет скорость решения увеличилась в 29 000 раз! Интересно, что самое большое ускорение, почти десятикратное, произошло в 1998 году, причем не случайно. До этого математики в течение 30 лет разрабатывали новые теории и методы, из которых очень мало было внедрено в практику. В 1998 году в версии CPLEX6.5 была поставлена задача реализовать по максимуму все эти идеи. В результате наши возможности в линейном программировании вышли на качественно новый уровень.

Процесс продолжается. Gurobi появился в 2009 году и к 2012-му ускорился в 16,2 раза. А общий эффект в 1991–2012 годах — в $29000 \times 16{,}2 = 469800$ раз! Повторим, что это произошло независимо от скорости компьютера, иными

словами, исключительно благодаря развитию математических идей.

Если верить закону Мура, то за 1992–2012 годы компьютеры ускорились примерно в 8000 раз. Сравните с почти полумиллионным ускорением алгоритмов! Получается, что если вам нужно решить задачу линейного программирования, то лучше использовать старый компьютер и современные методы, чем наоборот, новейший компьютер и методы начала 1990-х.

Мы не устаем восхищаться прогрессом компьютерных технологий. При этом математика достигла гораздо большего прогресса, и никто даже не заметил! Многогранники, плоскости, двойственные задачи и разветвления зашиты в программных пакетах и решают задачи планирования, как будто так было всегда и в этом нет ничего особенного.

Конечно, самое поразительное — совместный эффект математики и компьютеров. Ускорение в 4 миллиарда раз. Задачи, на которые требовалось 126 лет в 1991 году, в 2012-м мы научились решать за одну секунду! И это не предел. В статье 2015 года Димитрис Бертсимас и Анжела Кинг из Массачусетского технологического института приводят новую цифру — 450 миллиардов — и предлагают новые приложения линейного программирования в статистике.

При столь невероятной эффективности для линейного программирования открываются новые горизонты, немыслимые ранее, но вполне реальные сегодня.

Расписание движения поездов на голландских железных дорогах

Самая престижная награда в исследовании операций — премия Франца Эдельмана за выдающиеся успехи науки

в приложениях. В 2008 году ее получили Железные дороги Нидерландов за новое железнодорожное расписание, которое начало действовать в 2006 году.

Нидерланды — маленькая, но густонаселенная страна. На территории размером примерно с Нижегородскую область проживает около 17 миллионов человек. Железные дороги — основа всей голландской логистики. Многие каждый день ездят на поезде на работу. Совещание в другом конце страны — обычное дело. Голландцы — чемпионы Европы по пассажирским железнодорожным перевозкам. В 2006 году Железные дороги Нидерландов перевезли 15,8 миллиарда пассажиров.

До 2006 года действовало расписание, составленное в 1970 году. Перевозки увеличивались, составы постепенно удлинялись, а где возможно, добавлялись новые поезда. Пока наконец к началу 2000-х увеличивать перевозки в рамках старого расписания стало невозможно. Прокладывать новые пути фактически тоже не возможно из-за их безумной дороговизны, да и просто нехватки земли. Железнодорожные пути в Нидерландах строились и расширялись очень мало еще со времен Второй мировой войны. Задача, которая встала перед менеджментом железных дорог, — обеспечить требуемый объем перевозок при имеющейся инфраструктуре. Как это сделать? В 2002 году было решено составить новое расписание.

Расписание железных дорог — дело очень сложное. Нужно, чтобы два поезда одновременно не претендовали на один и тот же участок рельсов. Маршруты с пересадками тоже должны быть удобными, без получасового ожидания на платформе. Кроме того, нужно распределить пути прибытия и отправления на каждой станции, определить количество и тип вагонов для каждого состава, составить расписание кондукторов и машинистов. И все это для 5500 поездов в день!

Над задачей работала целая команда математиков. Основные сложности и идеи они описали в статье [20], за которую им была присуждена премия Эдельмана.

Каждый этап составления расписания требовал новой модели и новых подходов. Некоторые задачи, например распределение путей прибытия и отправления, после нескольких шагов предварительной подготовки удалось решить с помощью пакета CPLEX. Задача расписания движения поездов упрощалась благодаря цикличности: поезда отправляются в одно и то же время каждый час. Но даже в этом случае коммерческие пакеты оказались бессильны. Справиться с задачей помогли новые математические идеи. Внедрение не сразу прошло гладко. И все же теперь больше поездов перевозит больше людей по тем же рельсам. Пассажиропоток увеличивается, но расписание по-прежнему справляется. В рамках старого расписания это было бы невозможно.

Что такое оптимальное решение

Если вы недавно посещали Нидерланды, то последний раздел вас мог удивить. Железнодорожное движение далеко от совершенства. Мелкие (и крупные) задержки случаются сплошь и рядом. Пересадки порой очень короткие, их легко пропустить при малейшем опоздании. Поезда часто переполнены, особенно вагоны наиболее популярного 2-го класса. Далеко не все обрадовались новому расписанию. Влиятельная голландская газета NRC Handelsblad писала:

> Это единственная форма высшей математики, которая вызвала в обществе такую бурю эмоций.

Александр Схрейвер, знаменитый голландский математик, один из лучших в мире специалистов по оптимизации,

играл ведущую роль в составлении нового расписания. Критика журналистов его не очень взволновала. В одной из статей, рассчитанной на широкую публику, он пишет:

> Что определяет оптимальность? Комфорт пассажиров? Общий доход? Расписание персонала? Циркуляция материалов? Или пунктуальность? Каждый из этих аспектов сам по себе уже трудно оценить. Но даже если удастся, как взвесить эти факторы по отношению друг к другу?

Очень важно понимать, что оптимальное решение вовсе не означает решение идеальное. Оптимизация происходит с массой ограничений, и пожелания к решению противоречат друг другу. Например, максимальное количество пассажиров и дешевизна перевозок противоречат максимальному комфорту. Оптимальное решение — это лучшее, что мы можем сделать при заданных ограничениях и приоритетах.

В реальных задачах необходимо, чтобы математики и менеджеры сотрудничали и прислушивались друг к другу. Менеджеры должны уметь расставить приоритеты и обозначить ограничения. Математики должны уметь не только запустить в ход свои многогранники, но и вникнуть в особенности данной задачи. И тогда мы получим новые красивые результаты, которые воплотятся в значительных проектах. Как написал авторам сам Схрейвер: «Это был масштабный проект, но за ним стояла очень интересная, чистая математика». А свою статью он закончил словами: «Математика железных дорог пока далека от совершенства».

Глава 3
Мир нулей и единиц

Перевод текста в килобайты

Как передаются тексты с одного компьютера на другой? Например, мы хотим переслать текст «Мама мыла раму» или «Над всей Испанией безоблачное небо». Как это сделать? Разумеется, по проводам нельзя передать напечатанные слова «мама», «небо» и другие. Можно передать только сигналы. Значит, каждой букве или слову нужно подобрать свой сигнал.

Как перевести слова в сигналы? Если подумать, то на ум приходит нечто вроде азбуки Морзе, когда все буквы и слова передаются в виде точек и тире. Совершенно аналогично компьютеры пользуются всего двумя сигналами. Поэтому вся информация в компьютере записывается посредством двух символов: 0 и 1.

Понятно, что мы только для примера написали фразы по-русски. А вдруг завтра нам понадобится передать сообщение на английском или французском, где совсем другие (латинские) буквы, не говоря уже о разных «акцентах», характерных для французского языка? А еще есть арабские буквы, несколько витиеватых алфавитов Индии и тысячи, если не десятки тысяч, китайских и японских иероглифов! Обучить компьютер отличать один язык от другого — не проблема, такие программы уже есть. Но для хранения и передачи информации это совершенно бесперспективно. Гораздо проще для каждого символа, будь то буквы, иероглифы или знаки препинания, составить свою, уникальную, последовательность из нулей и единиц. К этому можно добавить

отдельные последовательности для часто употребляемых слов, особенно если их не слишком много.

Конечно, можно построить компьютеры, которые будут различать три разных сигнала или больше. Но двоичная система (два сигнала, два символа) гораздо удобнее. Ток либо есть (1), либо нет (0) — здесь машина не может ошибиться. Поэтому устройства с двоичной системой проще в изготовлении и надежнее.

Все наши компьютеры устроены именно так. Мы печатаем тексты на всех языках мира, рисуем таблички и сохраняем фотографии, а в памяти компьютера нет ничего, кроме нулей и единиц. На самом деле это поразительно!

Соответственно, объем информации в компьютере определяется тем, сколько нулей и единиц нам понадобилось. Каждый ноль или единица представляют собой минимальный объем информации, так называемый один бит. Восемь битов, то есть последовательность из восьми нулей и единиц, называется один байт. Всем известный килобайт (Кбайт), та самая единица измерения объема наших имейлов и документов, равен 1024 байта, или 8192 нуля и единицы. Аналогично мегабайт — это 1024 килобайта.

Почему 1024, а не просто 1000? Потому что в двоичной системе, где всего две цифры, проще всего пользоваться степенями двойки: 2, 4, 8, 16, 32 и так далее. Вполне логично, что в привычной нам десятичной системе, где у нас 10 цифр от 0 до 9, удобно пользоваться степенями десятки: 10, 100, 1000 и так далее. Число 1024 — это два в десятой степени:

$$1024 = 2^{10} = 2\times2\times2\times2\times2\times2\times2\times2\times2\times2.$$

Многие считают, что килобайт — это тысяча байтов. Строго говоря, это не так, но приблизительно верно и больше соответствует нашей привычке считать десятками, сотнями и тысячами.

Сколько килобайтов в вашем сообщении, зависит не только от самого текста, но и от того, каким образом ваши слова были переведены в нули и единицы.

Как сделать сообщение компактным? Где гарантия, что замена нескольких нулей на единицы из-за помех в каналах связи не изменит смысл на полностью противоположный? Проблемы передачи информации послужили толчком для развития целой области современной математики — теории кодирования.

Что такое кодирование

Итак, наша цель — закодировать каждый символ или каждое слово текста с помощью нулей и единиц. Фактически код — это словарь, состоящий из кодовых слов. Каждое кодовое слово представляет собой последовательность нулей и единиц (желательно небольшой длины), которая что-то означает (буква, цифра, знак препинания или целое слово). Построить код можно множеством разных способов, а значит, эта задача очень интересная.

Давайте, к примеру, закодируем отдельно каждую букву русского алфавита. Забавно, кстати, что, когда ставишь эту задачу студентам, почти всегда кто-нибудь спрашивает, сколько букв надо учитывать: 32 или 33. По-видимому, они считают букву «ё» не вполне самостоятельной, потому что в текстах ее обычно меняют на «е». Будем все-таки считать, что букв у нас 33. Сколько байтов (нулей и единиц) нам понадобится, чтобы закодировать 33 буквы?

Совершенно ясно, что тридцати трех байтов вполне достаточно, потому что мы можем каждую букву обозначить кодом из 32 нулей и одной единицы — на той позиции, которую занимает эта буква в алфавите. Такой наивный код будет выглядеть так:

а: 10000000000000000000000000000000
б: 01000000000000000000000000000000
в: 00100000000000000000000000000000
г: 00010000000000000000000000000000

и так далее

я: 00000000000000000000000000000001

Сразу видно, что подобное кодирование неприменимо на практике. Слишком много драгоценных байтов уходит на нули, которые несут очень мало информации.

Какая минимальная длина кода нам понадобится, чтобы закодировать русский алфавит? Скажем, хватит ли нам кодов длины 5? Это зависит от того, сколько разных последовательностей из нулей и единиц длины 5 мы можем составить: 00000, 00001, 00010, 00011 и далее до 11111. Всего 32 такие последовательности. Получить данный ответ довольно просто: это 2 в степени 5, то есть $2 \times 2 \times 2 \times 2 \times 2$*.

Оказывается, последовательностей длины 5 не хватает, так что вопрос студентов попал в самую точку! Всего из-за одной «лишней» буквы нам понадобится как минимум 6 нулей и единиц в каждом «кодовом слове».

Интересно, что добавление всего одной позиции кода очень сильно меняет дело. Для русского алфавита нам нужны последовательности длины 6, а их уже 64. Значит, нам их хватит не только на русский алфавит, но и, например, на латинский из 26 букв, и в запасе еще останется пять свободных последовательностей для знаков препинания.

Ключевой вывод: *добавление всего одной позиции кода увеличивает количество разных последовательностей вдвое*. Потому что лишнюю позицию можно заполнить двумя

* Для интересующихся читателей в приложении 1 к главе 3 мы приводим вычисления числа последовательностей из нулей и единиц заданной длины.

способами — либо нулем, либо единицей. В результате количество букв, слов или сообщений, которые мы можем закодировать, возрастает с длиной кода по так называемому *экспоненциальному* закону, как степень двойки.

«Растет по экспоненциальному закону» на общедоступном языке означает «растет очень быстро»! Помните легенду о том, как король хотел наградить изобретателя шахмат? Умный изобретатель попросил короля положить на первую клетку доски одно зернышко, на вторую — два, на третью — четыре и далее в том же порядке: в два раза больше на каждую следующую клетку. Король согласился и был потрясен, когда зерна в его амбарах не хватило и на половину доски. Точно так же и количество возможных последовательностей из нулей и единиц возрастает очень быстро с их длиной: 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024…

Экспоненциальная зависимость между количеством разных кодовых слов и их длиной — абсолютно фундаментальная концепция в информатике и вопросах передачи информации.

Заметим, что количество информации зависит не только от длины кода в килобайтах, но и от того, насколько информативны кодируемые слова. Естественная иллюстрация — это отправка сообщения по телеграфу. Там каждое слово стоит денег, и люди стараются не использовать лишних слов, избегая союзов и предлогов, потому что они менее информативны, чем глаголы и существительные.

Основные концепции о том, как измерить количество информации, изложены в фундаментальной работе Клода Шэннона, опубликованной в 1949 году [24]. Эти концепции во многих отношениях положили начало развитию информатики и строятся все на той же основополагающей экспоненциальной зависимости. Однако мы не будем углубляться в теорию информации, а вернемся к вопросу о том, как составить надежные и эффективные коды.

Коды, исправляющие ошибки

Как вы уже поняли, код — это просто любой набор последовательностей из нулей и единиц. Но не все коды одинаково хороши. Разумеется, неплохо, если каждое кодовое слово достаточно короткое. Об этом мы как раз говорили в прошлом разделе. Однако есть куда более интересные характеристики кода, нежели длина кодовых слов.

Представим, что мы начали передавать кодовые слова по каналу связи, но в нем иногда возникают помехи. Из-за каждой такой помехи передаваемый символ меняется на противоположный: ноль — на единицу, единица — на ноль. Можно ли подобрать кодовые слова так, чтобы все передаваемые символы, несмотря на ошибки, удалось однозначно восстановить? Звучит как научная фантастика! Но оказывается, можно! Достаточно лишь правильно сформулировать соответствующую математическую задачу.

Допустим, наши кодовые слова длины 6 и на каждое кодовое слово приходится не более одной ошибки. Поскольку это простой пример, представим, что наш словарный запас беднее, чем у Эллочки-людоедки из «Двенадцати стульев», и состоит всего из трех слов, которые мы закодировали тремя кодовыми словами:

$$111000,\ 001110,\ 100011.$$

Конечно, много таким кодом не передашь, но для примера вполне достаточно. Еще важно, что получатель знает наш «словарь», то есть ожидает от нас либо 111000, либо 001110, либо 100011 и ничего другого.

Предположим, мы сначала передаем слово 111000. В результате не более чем одной ошибки (ошибки мы выделили жирным шрифтом) оно может превратиться в одно из слов:

$$\begin{gathered}111000,\ \mathbf{0}11000,\ 1\mathbf{0}1000,\ 11\mathbf{0}000,\\ 111\mathbf{1}00,\ 1110\mathbf{1}0,\ 11100\mathbf{1},\end{gathered} \tag{3.1}$$

включая, как видите, само себя. Аналогично при передаче слова 001110 может получиться любое из слов:

$$001110,\ \mathbf{1}01110,\ 0\mathbf{1}1110,\ 00\mathbf{0}110,\\ 001\mathbf{0}10,\ 0011\mathbf{0}0,\ 00111\mathbf{1}. \tag{3.2}$$

Наконец, для 100011 у нас выйдет:

$$100011,\ \mathbf{0}00011,\ 1\mathbf{1}0011,\ 10\mathbf{1}011,\\ 100\mathbf{1}11,\ 1000\mathbf{0}1,\ 10001\mathbf{0}. \tag{3.3}$$

Замечательно то, что списки (3.1)–(3.3) попарно не пересекаются. Иными словами, если на другом конце канала связи появляется любое слово из списка (3.1), получатель точно знает, что ему передавали именно слово 111000, а если появляется любое слово из списка (3.2) — слово 001110; то же самое касается и списка (3.3). В этом случае говорят, что наш код *исправил одну ошибку.*

За счет чего произошло исправление? За счет двух факторов. Во-первых, получатель знал весь «словарь». Когда код передавался всего с одной ошибкой, выходило слово, которого в словаре не было.

Во-вторых, слова в словаре были подобраны особенным образом. Даже при возникновении ошибки получатель не мог перепутать одно слово с другим. Например, если словарь состоит из слов «*дочка*», «*точка*», «*кочка*» и при передаче получалось «*вочка*», то получатель, зная, что такого слова не бывает, исправить ошибку не смог бы — любое из трех слов может оказаться правильным. Если же в словарь входят «*точка*», «*галка*», «*ветка*» и нам известно, что допускается не больше одной ошибки, то «*вочка*» это заведомо «*точка*», а не «*галка*». В кодах, исправляющих ошибки, слова выбираются именно так, чтобы они были «узнаваемы» даже после ошибки. Разница лишь в том, что в кодовом «алфавите» всего две буквы — ноль и единица.

Совершенно ясно, что наугад такие коды составить невозможно. За этим стоит целый математический аппарат. Нам нужно научиться измерять расстояния между словами и даже работать с шарами из слов. Что это такое и как это делается, может понять практически любой человек. Ниже мы попробуем объяснить, как создаются коды, исправляющие ошибки, и какие при этом возникают проблемы.

Шары Хэмминга

Математики уже давно договорились, что такое *шар*. Шар состоит из всех точек, которые удалены от центра не больше чем на какое-то заданное расстояние, обозначаемое буквой *r,* — радиус. В нашем привычном трехмерном пространстве, где расстояния измеряются в метрах, это обычный шар, как на рис. 3.1.

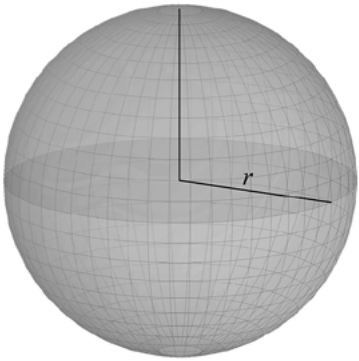

Рис. 3.1. Шар радиуса ***r*** *в трехмерном пространстве. Все точки удалены от центра не больше чем на расстояние* ***r***

Зато понятия *точка* и *расстояние* в математике абсолютно абстрактные. Есть несколько простых правил, которых следует строго придерживаться, но в рамках этих правил — полная свобода. Точками могут быть сигналы, а могут — кривые

и даже результаты случайных экспериментов. И расстояния между ними можно определить самыми разными способами. В итоге получаются «шары», не поддающиеся воображению. Тем не менее эти абстрактные шары играют в математике очень большую роль.

В теории кодирования точка — это кодовое слово, то есть последовательность нулей и единиц заданной длины. А расстоянием принято считать так называемое *расстояние Хэмминга*.

Расстояние Хэмминга между двумя кодовыми словами — это всего-навсего число позиций, на которых у этих слов стоят разные символы: у одного 0, а у другого 1. Например, на рис. 3.2 расстояние Хэмминга между двумя кодовыми словами равно трем. Мы заключили в рамки те позиции, где эти два кодовых слова отличаются друг от друга.

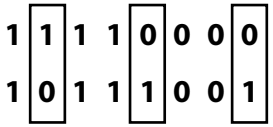

Рис. 3.2. Расстояние Хэмминга между двумя кодовыми словами — число позиций, на которых у этих слов стоят разные символы. На рисунке это расстояние равно трем. Позиции, где два кодовых слова отличаются друг от друга, заключены в рамки

Что происходит, если при передаче, скажем, слова 111000 произошла одна ошибка?

Получится другое слово, которое будет отличаться от 111000 всего на одной позиции. Иначе говоря, если у нас при передаче происходит не больше одной ошибки, расстояние Хэмминга между отправленным и полученным кодовым словом будет не больше единицы. Давайте снова посмотрим на перечень (3.1) предыдущего раздела:

111000, **0**11000, 1**0**1000, 11**0**000, 111**1**00, 1110**1**0, 11100**1**.

Расстояние Хэмминга между словом 111000 и любым другим словом из перечня не превосходит 1. Значит, этот список — не что иное, как шар радиуса 1 с центром 111000!

Кстати, подобное определение можно ввести и для обычных слов русского языка одинаковой длины. Например, расстояние Хэмминга между словами «*дочка*» *и* «*точка*» равно единице, а между словами «*точка*» и «*галка*» — трем. Если слово «*точка*» — это центр шара Хэмминга радиуса 1, то слово «*дочка*» входит в этот шар, а слово «*галка*» — нет.

Шары Хэмминга очень трудно себе представить даже для маленьких кодов. На рис. 3.3 мы изобразили расстояния Хэмминга между кодовыми словами длины 3. Расстояния Хэмминга могут быть 0, 1, 2 или 3. На рисунке чем темнее цвет, тем больше расстояние. Если взять, например, колонку 000, то шар Хэмминга радиуса 1 — это все белые и светло-серые квадратики в этой колонке: 000, 001, 010, 100. Сразу видно, что расположение белых и, скажем, самых темных квадратиков в колонках неодинаковое, хотя, конечно, в рисунке много закономерностей. Например, рисунок из самых светлых тонов (белый и светло-серый) абсолютно симметричен рисунку из двух оставшихся более темных тонов.

Кодовые слова длины 3 — очень простой пример, их всего восемь. Стоит чуть увеличить длину кодового слова, и из-за уже знакомого нам экспоненциального закона слов станет так много, что картинка нам не поможет. Профессор Джон Слэни из Австралийского национального университета сделал замечательный рисунок, на котором изображены расстояния Хэмминга между кодовыми словами длины 8, а это всего один байт. Таких слов 256. Советуем заглянуть на веб-страницу Слэни http://users.cecs.anu.edu.au/~jks/Hamming.html и посмотреть на этот рисунок. Вам сразу станет понятно, что он никак не поможет найти хороший код. Картинка скорее напоминает красивый коврик. Нам нужен

другой математический аппарат, и, к счастью, такой аппарат есть. Теория кодирования тесно связана с комбинаторикой — наукой о комбинациях тех или иных объектов.

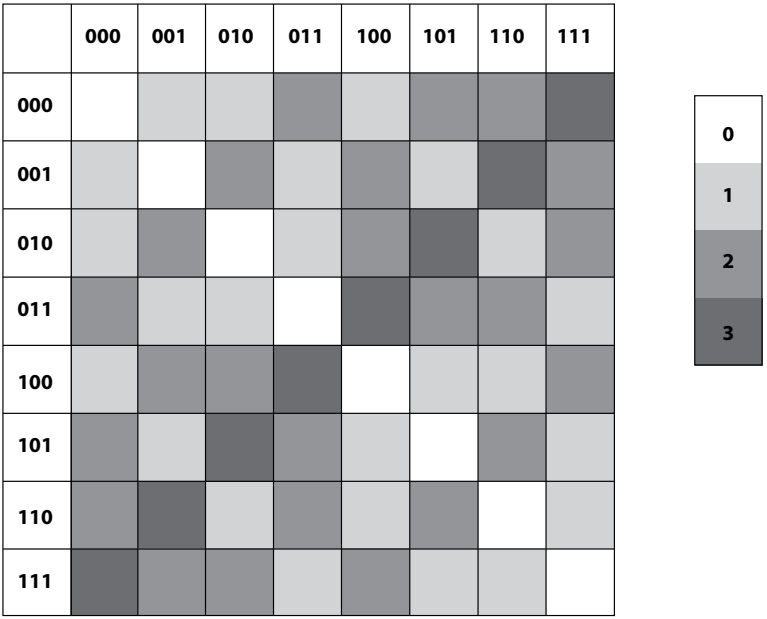

Рис. 3.3. Расстояние Хэмминга между кодовыми словами длины 3. Справа изображен цветовой код. Расстояния: 0, 1, 2, 3. Чем темнее цвет, тем больше расстояние

Возможно, вы уже заметили связь между шарами Хэмминга и кодами, исправляющими ошибки. Допустим, мы передаем кодовые слова длины 10 и хотим, чтобы код исправлял две ошибки. Тогда надо построить код, в котором шары с центрами в кодовых словах и радиусами 2 попарно не пересекались бы. Все последовательности нулей и единиц в таком шаре будут означать одно и то же кодовое слово. Иначе говоря, кодовые слова должны отличаться друг от друга настолько, чтобы при наличии двух ошибок их невозможно было перепутать. На языке математики это значит, что

расстояния Хэмминга между кодовыми словами должны быть как минимум равны 5.

При создании кодов возникает немало интересных вопросов. Например, очень важно, чтобы количество кодовых слов было как можно бóльшим, так как это позволит передать по каналу связи больше информации, исправляя при этом заданное наперед количество ошибок (характерное для данного канала связи). Отыскание максимальных кодов при заданной длине кодового слова и количестве ошибок — крайне трудная и не до конца решенная математическая задача. По сути, это задача комбинаторики, хотя и мотивированная совершенно практическими вопросами кодирования информации.

История кодов, исправляющих ошибки

Ричард Хэмминг, именем которого названы расстояния между кодовыми словами, — один из основателей современной теории кодирования. Эта теория начала развиваться в конце 40-х годов XX века, когда Хэмминг работал в Bell Labs и занимался проблемами передачи информации.

Хэмминг заметил, что в коде, исправляющем ошибки, количество возможных кодовых слов неизбежно должно быть ограничено. Например, мы хотим построить код из слов длины 10, который исправляет две ошибки. Сколько разных кодовых слов мы можем использовать? Допустим, мы выбрали кодовое слово 0000011111, чтобы закодировать букву «а». Теперь все последовательности на расстоянии Хэмминга один или два от 0000011111, то есть в шаре радиуса 2 с центром 0000011111, нельзя использовать для кодирования никакой другой информации. Они нам нужны для исправления

ошибок при передаче буквы «а». Такой шар содержит 56 последовательностей. Получается, что на каждое кодовое слово, которое что-то означает, приходится 56 слов на исправление ошибок. Поэтому мы можем закодировать не 1024, а всего лишь $^{1024}/_{56}$, то есть 18 разных символов или сообщений. Если этого мало — например, мы хотим закодировать русский алфавит, — то длину кодовых слов придется увеличить. Это увеличит и количество килобайтов, но такова цена за исправление ошибок.

Рассуждая примерно так, как мы описывали в предыдущем параграфе, Хэмминг получил максимальное число слов любой длины с исправлением любого заданного количества ошибок. Эта формула называется *границей Хэмминга**.

Заметим, что граница Хэмминга — это *максимально* возможное число слов в коде, исправляющем ошибки. Но это еще не означает, что такой максимальный код можно найти. Представьте, что мы имеем дело с обычными шарами. Нельзя уложить круглые апельсины в ведро или коробку так, чтобы между соседними апельсинами не было пустот! Где же гарантия, что все последовательности из нулей и единиц можно разбить на шары Хэмминга так, чтобы они не пересекались и между ними не было свободного места? Оказывается, такое разбиение во многих случаях возможно. Мы не станем здесь описывать конструкцию, ее можно найти в специальной литературе (см. [6]).

Впрочем, сделано в этом направлении далеко не все. Конструкции есть, но только *асимптотические,* то есть такие, в которых длина кодового слова очень большая и увеличивается до бесконечности, а количество ошибок при этом маленькое и фиксированное. Однако во многих практических задачах чем длиннее кодовое слово, тем больше ошибок. В такой

* Для подготовленного читателя в приложении 2 к главе 3 приведена точная формула границы Хэмминга и ее доказательство в общем случае.

ситуации проблема отыскания правильных верхних границ и построения максимальных кодов по-прежнему открыта! Более того, в этой ситуации известно, что граница Хэмминга, как правило, не точна, и есть еще масса более аккуратных оценок, среди которых и оценка Владимира Иосифовича Левенштейна, замечательного российского математика, и границы *линейного программирования.* Эти конструкции очень сложны и выходят далеко за рамки нашей книги.

Особый интерес вызывают так называемые *равновесные* коды. В них все кодовые слова содержат одинаковое количество единиц. Для таких кодов можно найти границу, аналогичную границе Хэмминга. И снова возникает вопрос о существовании равновесного кода, достигающего полученной границы. В начале 80-х годов XX века Войцех Рёдль придумал очень хитрую вероятностную технику, позволившую доказать наличие равновесных кодов, в которых число кодовых слов асимптотически равно верхней границе [7].

Буквально пару лет назад Питер Киваш из Оксфордского университета объявил о построении равновесных кодов, в которых количество слов достигает теоретической верхней границы. Это очень объемная и трудная математическая работа; она до сих пор тщательно проверяется и потому опубликована только в интернете [19].

Что же касается случаев, когда число ошибок и число единиц в равновесном коде пропорциональны длине кодового слова, то здесь пока все безнадежно. А именно эти случаи особенно важны! Вот такая она, математика. На первый взгляд кажется, что все давно известно. А на самом деле вопросов, на которые по-прежнему нет ответов, несмотря на всю естественность их возникновения и важность для практики, гораздо больше, чем тех, ответы на которые уже получены.

Можем ли мы закодировать все подряд

Все описанное в этой главе в основном применимо к кодированию текстов. Другим видам информации присущи свои особенности.

Как, например, закодировать цвет? Наверное, вы не раз отправляли, получали или по крайней мере видели в интернете цветные фотографии. Значит, закодировать цвет можно, и люди уже этому научились. На самом деле это вовсе не тривиальная задача, и решить ее удалось только потому, что, оказывается, любой цвет можно разбить на три компонента: *красный, зеленый* и *синий.* Иными словами, любой оттенок определяется лишь интенсивностью этих трех основных цветов. Так устроен наш цветной мир, такой вот подарок для кодирования. Если бы природа цвета была иной, то никаких цветных фотографий мы не могли бы пересылать.

Если вам когда-нибудь приходилось использовать нестандартные цвета в обычных программах типа Word или PowerPoint, вы, возможно, заметили опцию, где интенсивность красного, зеленого и синего можно задать вручную. Обычно интенсивность определяется числом от 0 до 255. Всего 256 вариантов — ровно по количеству разных последовательностей нулей и единиц длины 8, то есть один байт. В результате нам нужно всего три байта, чтобы закодировать $256 \times 256 \times 256 = 16777216$ — более шестнадцати с половиной миллионов оттенков компьютерной палитры.

Если обозначать интенсивность цветов цифрами в стандартном порядке красный-зеленый-синий, то 0–0–0 — это черный цвет, а 255–255–255 — белый. Если интенсивность красного, зеленого и синего одинаковая, то между черным и белым получится целых 254 оттенка серого. А желтый

можно получить, если использовать красный и зеленый с одинаковой интенсивностью. В табл. 3.1 приводится пример красно-зелено-синего кода для цветов радуги.

Таблица 3.1. Пример цветового кода для цветов радуги. Интенсивность основных цветов (красного, зеленого и синего) определяется числом от 0 до 255

Цвет	Интенсивность основных цветов		
	Красный	Зеленый	Синий
Красный	255	0	0
Оранжевый	255	128	0
Желтый	255	255	0
Зеленый	0	255	0
Голубой	0	255	255
Синий	0	0	255
Фиолетовый	255	0	255

Дополнительные серьезные проблемы возникают при необходимости передать фильм. Если кодировать каждую точку каждого кадра, понадобится такое количество гигабайтов, что они не уместятся в памяти ни одного компьютера. Поэтому цифровое видео появилось относительно недавно.

Здесь требовалось решить задачу иного рода — задачу *сжатия данных:* как сделать код как можно короче и при этом не потерять информацию. Такие задачи часто возникают, когда нужно сохранить и использовать большое количество информации. Именно эту задачу решают программы для архивирования типа Zip.

Стандартный способ сжатия данных для фильма — полностью кодировать первый кадр, а затем кодировать не каждый кадр отдельно, а только изменения. Но при необходимости сохранить информацию иного рода, например кто с кем дружит в «Фейсбуке», понадобятся совершенно другие способы сжатия данных. И они, кстати, уже разработаны.

А вот со звуком все намного сложнее. Мы до сих пор не умеем кодировать звучание симфонического оркестра так, чтобы оно воспроизводилось как в концертном зале.

Эта задача выходит за рамки не только нашей книги, но и математики. Математика — великая наука, но ее возможности распространяются только на объекты, которые поддаются *формальному* описанию, буквами и числами. Математики научились кодировать текст потому, что когда-то люди изобрели слова и алфавит. Математики умеют кодировать цвет потому, что физики обнаружили, что любой оттенок определяется интенсивностью красного, синего и зеленого. Несомненно, математики придумали бы эффективные коды для живого звука, если бы хоть кто-нибудь сумел описать, из каких сигналов и интенсивностей он состоит. Но, к сожалению, сделать это исчерпывающим образом пока никому не удалось.

Глава 4
Надежность интернета

Связанные одной сетью

Практически каждый из нас ежедневно пользуется интернетом. Интернет — это сеть компьютеров и серверов, которые физически соединены каналами связи для передачи цифровой информации с одного сервера на другой.

Сигнал идет со скоростью света, поэтому совершенно неважно, где находятся серверы — в России, США или Австралии. Пройденные расстояния практически не влияют на скорость передачи. Мы все уже давно привыкли, что имейлы и WhatsApp доходят в считаные секунды, веб-страницы грузятся быстро, а наш голос и даже изображение передаются по скайпу в реальном времени. Но, если задуматься, где гарантия, что в любой момент любой сервер мира может связаться с любым другим?

В какой-то степени интернет можно сравнить с системой железных дорог. От любой станции можно добраться до любой другой. Но железные дороги спланированы централизованным образом, их план прошел множество инстанций. Интернет — совсем другое дело. Основные каналы связи (обычно это волоконно-оптические линии) принадлежат самым разным владельцам: компаниям и организациям, например крупным операторам телефонной и мобильной связи. Вместе они составляют так называемую *опорную сеть интернета.*

Большинство компаний, в том числе и многие интернет-провайдеры, заключают договоры на пользование каналами связи и платят аренду. Как только появляется выход к опорной сети, можно начинать строить собственную сеть, присоединять новые серверы и компьютеры. Возникают локальные сети, они соединяются друг с другом, образуют более крупные сети и так далее. И все эти гигантские сети сетей соединены центральной, опорной сетью. Отсюда и название интернет (Internet*): net* по-английски — *сеть*.

Ни один человек и ни одна компания в мире не отвечают за то, чтобы сервер, через который вы присоединились к интернету, был связан с другим сервером, скажем, на острове Кенгуру*. Но вся система по своей природе устроена так, что связь гарантирована. Интернет — гигантская международная технологическая и коммерческая конструкция, без которой мы уже не представляем своей жизни, — прекрасно обходится без правления и правительства. Если вдуматься, это просто поразительно!

Еще поразительнее то, что связь практически никогда не теряется, хотя в каналах связи случаются неполадки и неизбежные регулярные перегрузки. Может ли интернет, хотя бы временно, «развалиться на кусочки»? Может ли случиться так, что из-за сбоев где-то по дороге ваш сервер окажется полностью отрезанным от острова Кенгуру? На самом деле это очень сложный вопрос, на который нет однозначного ответа. При этом из опыта совершенно ясно, что интернет невероятно устойчив к помехам. Согласитесь: если ваш сервер и сервер получателя исправны, то информация всегда проходит через сеть безо всяких проблем.

Эта глава о том, как мы можем хотя бы частично понять и объяснить удивительную надежность интернета.

---

* Такой остров действительно есть, его население составляет 4,5 тысяч человек; кенгуру там гораздо больше!

Сети и помехи

Начнем с простого примера. Допустим, наш интернет состоит всего из трех компьютеров, которые соединены друг с другом как на рис. 4.1. Если все три канала связи работают, нет никаких проблем: все три компьютера могут обмениваться информацией.

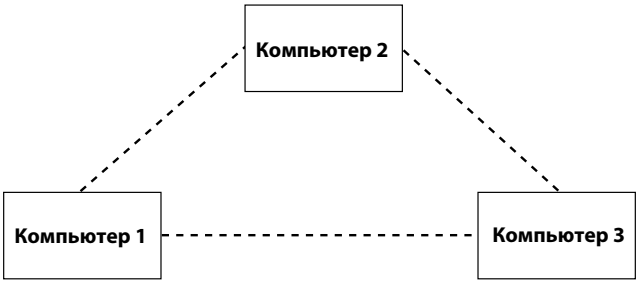

Рис. 4.1. Мини-интернет из трех компьютеров, соединенных каналами связи. Все три канала работают, все три компьютера могут обмениваться информацией

Теперь допустим, что в одном из каналов связи возникли помехи и передать по нему в данный момент ничего нельзя. Мы изобразили эту ситуацию на рис. 4.2. Сразу видно, что наш мини-интернет не распался. Несмотря на то что прямая связь между компьютерами 1 и 2 утеряна, они по-прежнему могут передавать друг другу информацию через компьютер 3. Заметит ли пользователь неполадку в канале? Скорее всего, нет. Поскольку сигнал идет со скоростью света, нет никакой разницы в скорости доставки информации — пойдет ли сигнал напрямую из Москвы в Нижний Новгород или даст кругаля через Сидней или Нью-Йорк.

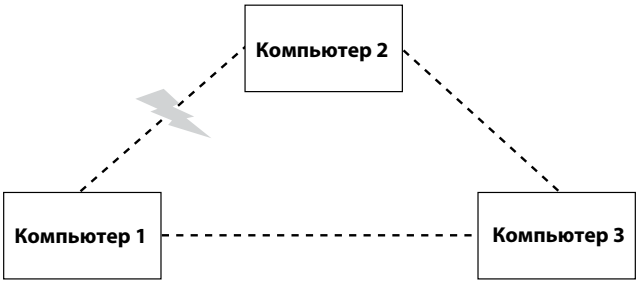

Рис. 4.2. Мини-интернет из трех компьютеров. Хотя канал связи между компьютерами 1 и 2 недоступен, они по-прежнему могут обмениваться информацией через компьютер 3

Чтобы развалить нашу маленькую сеть, нужно вывести из строя как минимум два, а то и все три канала связи, как показано на рис. 4.3.

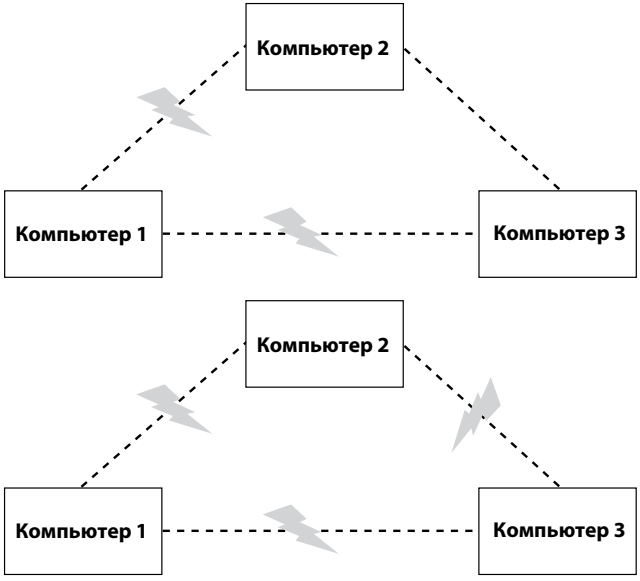

Рис. 4.3. Мини-интернет из трех компьютеров. Сверху: вышли из строя два канала связи, компьютер 1 оказался отрезанным от сети. Снизу: вышли из строя все три канала связи, связь между компьютерами полностью прервана

Насколько устойчива наша мини-сеть? Сосчитать это совсем нетрудно. Допустим, помехи в отдельных каналах связи возникают независимо друг от друга с какой-то вероятностью, скажем 40 %. На практике это означает, что в среднем в четырех из десяти случаев канал оказывается недоступным. Сорок процентов — многовато для реального интернета, но для примера подойдет.

Компьютер 1 может оказаться отрезанным от сети, как на рис. 4.3 сверху с вероятностью $0{,}4 \times 0{,}4 \times 0{,}6 = 0{,}096 (\times 100\,\%) = 9{,}6\,\%$. В аналогичную ситуацию могут попасть компьютеры 2 и 3 с той же долей вероятности. Наконец, надо добавить вероятность самой плохой ситуации, как на рис. 4.3 снизу, которая равна $0{,}4 \times 0{,}4 \times 0{,}4 = 0{,}064 (\times 100\,\%) = 6{,}4\,\%$. В результате получается, что наша сеть «развалится» с вероятностью $3 \times 9{,}6\,\% + 6{,}4\,\% = 35{,}2\,\%$.

Конечно, 35,2 % — довольно много, но мы взяли нереально большую вероятность помех. Самое интересное, что вероятность потери связи в сети *меньше,* чем вероятность потери связи в одном канале: 35,2 % меньше, чем 40 %. Сеть более устойчива, чем отдельно взятый канал!

Даже из нашего мини-примера понятно, откуда берется устойчивость сети. В сети компьютеры могут связаться друг с другом не одним, а несколькими способами, через другие компьютеры. Если один канал недоступен, можно найти альтернативный маршрут. Более того, этот эффект заметно усиливается при меньшей вероятности помех. Для примера мы приводим несколько результатов в табл. 4.1*.

* В приложении 1 к главе 4 для подготовленного читателя приведены формулы, которыми мы воспользовались при подготовке в табл. 4.1, в общем виде.

Таблица 4.1. Вероятности потери связи в мини-сети (правая колонка) при заданной вероятности потери связи в одном канале (левая колонка)

Вероятность помехи в канале	Вероятность потери связи в мини-сети
50%	50%
40%	35,2%
30%	21,6%
20%	10,4%
10%	2,8%
5%	0,725%
2%	0,12%
1%	0,03%

Мы видим, что значения в правой колонке убывают гораздо быстрее, чем в левой. Если вероятность недоступности канала 1% — величина вполне реальная на практике, — то наша скромная мини-сеть в 33 раза устойчивее, чем отдельный канал связи!

Конечно, наш мини-интернет очень далек от реальности. Что, если у нас не три компьютера, а целая сеть из десятков, сотен, тысяч машин? Скорость света по-прежнему позволит передавать информацию не напрямую, а по длинным цепочкам. Однако подсчет вероятностей значительно затруднится.

И вот тут опять понадобится математика! Задачи об устойчивости больших сетей требуют глубоких концепций и новых моделей на стыке комбинаторики и теории вероятностей. К счастью, необязательно вникать в длинные доказательства, чтобы понять основные идеи. О некоторых таких фундаментальных идеях, проникающих в самую суть устройства сетей и уже ставших классикой, мы расскажем в следующих разделах.

Случайные графы

Математическая теория, которая, в частности, позволяет ответить на вопрос об устойчивости больших сетей, возникла на рубеже 50–60-х годов XX века. Ее авторами стали два замечательных венгерских математика Пол Эрдеш и Альфред Реньи [22, 15].

Эрдеш — настоящий классик современной комбинаторики, теории чисел, теории вероятностей. Он написал более *полутора тысяч* статей, решил множество проблем и еще больше поставил задач, которые определили пути развития науки на долгие годы. Реньи — выдающийся венгерский специалист по теории вероятностей. Его именем назван математический институт в Будапеште, подобно тому как математический институт в Москве носит имя Владимира Андреевича Стеклова.

Пол Эрдеш

Пол Эрдеш (1913–1996) — очень необычная фигура в математике. Он написал около 1500 статей с 509 соавторами.

Практически вся его собственность умещалась в один чемодан. Деньги его совсем не интересовали. Он жил в дороге, ездил с одной конференции на другую или останавливался у коллег. Рассказывают, что он появлялся на пороге и говорил: «Мой мозг открыт». Затем он работал с хозяевами несколько дней, получал результаты для нескольких статей и ехал дальше, в следующий дом и к другим задачам.

Его соавтор Фэн Чжун написала в своих воспоминаниях: «*Все 83 года своей жизни он был абсолютно верен себе. Его не соблазняли посты и деньги. Большинство из нас окружили себя множеством земных благ и обязательств. Каждая встреча с ним напоминала мне, что это все-таки возможно, вот так идти за своей мечтой, не обращая никакого внимания на мелочи жизни. Именно по этому качеству дяди Пола я скучаю больше всего*».

В математике, как в искусстве или моде, есть индивидуальные стили и вкусы. Соавторы Эрдеша рассказывают, что ему удавалось найти задачи, подходящие именно для них. Так хорошо он понимал своих соавторов и столько разных задач у него было в запасе! Результаты Эрдеша — разнообразные и в огромном количестве — сильно повлияли на современную науку.

Среди математиков есть понятие *число Эрдеша*. Это количество соавторов, которые отделяют математика от Эрдеша. У соавторов Эрдеша число Эрдеша равно 1. У их соавторов, которые с Эрдешем не работали, число Эрдеша — 2. И так далее. Самое распространенное значение — 5, и очень редко у кого из математиков число Эрдеша равно 8 или больше.

У одного из нас число Эрдеша 3, а у другого 2. Мы оба работаем со случайными графами и вносим свой посильный вклад в решение пока нерешенных проблем.

Теория, основы которой заложили Эрдеш и Реньи, называется *теорией случайных графов*. А математическая модель, рассмотренная в их первых работах, носит имя *случайный граф Эрдеша — Реньи*. Конечно, эти графы не имеют ничего общего с князьями и баронами. Слово «граф» в данном случае имеет то же происхождение, что и слово «график», хорошо известное со школы. Граф — это просто рисунок.

Например, нашу мини-сеть из предыдущего раздела очень легко представить в виде графа. Мы это сделали на рис. 4.4 слева. Сеть изображена в виде трех *узлов* (компьютеров), которые соединены линиями (каналами связи). В математике узлы называются *вершинами* графа, а линии между ними — *ребрами*. Чтобы не затруднять восприятие абстрактными терминами, мы в основном будем пользоваться более интуитивными терминами «узлы» и «линии»*.

* В приложениях для подготовленного читателя мы будем придерживаться математической терминологии: «вершины» и «ребра».

В главе 7 мы вернемся к графам, когда будем обсуждать социальную сеть. В этом случае узлы — это люди, а линии между ними — «дружба» в социальной сети. Мы изобразили пример графа из главы 7 на рис. 4.4 справа.

Естественно, узлов у графа может быть и тысяча, и миллион, и миллиард...

Теория графов — это классическая область математики с огромным количеством приложений. В виде графа можно представить систему железных дорог, газопровод, последовательность операций на крупном производстве или слов в русской речи и многое другое.

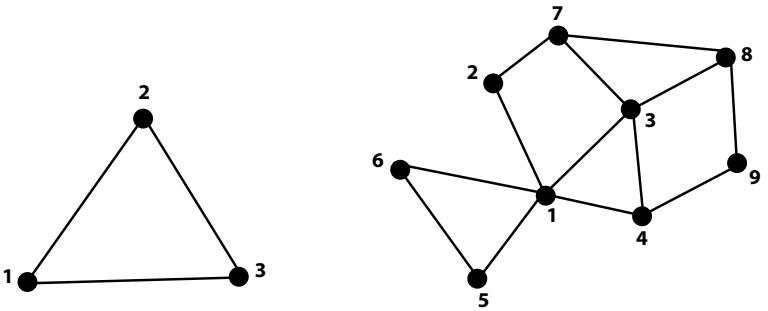

Рис. 4.4. Слева: мини-сеть в виде графа. Узлы — это компьютеры, а линии — каналы связи. Справа: социальная сеть из главы 7 в виде графа. Узлы — это люди, а линии — «дружба» в социальной сети

У *случайных* графов есть еще одна особенность. Нам неизвестно заранее, какие узлы связаны линией, а какие — нет. Линии между узлами могут существовать или нет с определенной вероятностью. Именно эту ситуацию мы обсуждали в примере с мини-сетью. При наличии помех канал связи становится недоступным, линия между узлами исчезает.

Случайный граф — естественная модель во многих ситуациях. Например, дружба в социальных сетях возникает непредсказуемым образом. В телекоммуникациях или электрических сетях на линиях связи могут случаться сбои.

Если попытаться смоделировать нейронную сеть мозга, то взаимодействия нейронов можно выявить только с определенной вероятностью.

В последнее время в связи с развитием интернета и социальных сетей и небывалой доступностью данных во всех областях — от энергоснабжения до биологии — интерес к теории случайных графов особенно вырос. Новые, очень сложные результаты появляются почти каждый день.

Что же говорит теория случайных графов об устойчивости сети?

Результат Эрдеша — Реньи

В связи с устойчивостью интернета нас интересует вопрос о *связности* случайного графа. Граф называется *связным,* если между двумя любыми его вершинами можно пройти по цепочке ребер, то есть все узлы связаны друг с другом. Оба графа на рис. 4.4 — связные. На рис. 4.5 мы сделали их *несвязными,* удалив по два ребра.

Эрдеш и Реньи задались вопросом: при какой вероятности помех сеть заданного размера остается связной? Результат получился поразительным! Оказывается, в больших сетях связность сохраняется даже при повышенной вероятности помех.

Например, возьмем сеть из 100 связанных между собой компьютеров. Получается, что каждый отдельный канал может быть недоступен с вероятностью аж 86 %, тем не менее сеть останется связной с вероятностью как минимум... 99 %! Эта ситуация изображена на рис. 4.6: 86 % из всех возможных линий отсутствует, однако сразу видно, что из любого узла можно добраться до любого другого.

А сеть из 1000 узлов — это и вовсе нечто фантастическое. Канал связи может быть недоступен с вероятностью 98 %, а связность сохраняется с вероятностью 99,9 %! Чем больше сеть, тем сильнее результат.

В табл. 4.2 мы приводим результаты для сетей разных размеров. Легко заметить, что число в самой правой колонке не что иное, как

$$\left(1-\frac{1}{\text{размер сети}}\right)\times 100\%.$$

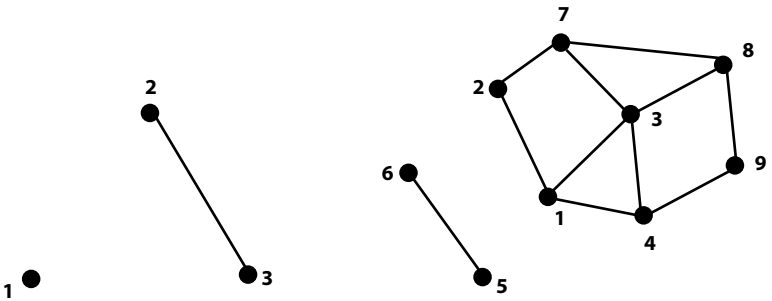

Рис. 4.5. Слева: мини-сеть в виде графа; каналы 1–2 и 1–3 недоступны; граф несвязный, из вершины 1 нельзя попасть в вершины 2 и 3. Справа: социальная сеть в виде графа; пользователь 1 не знаком с пользователями 5 и 6; граф несвязный; нет цепочки знакомых между пользователями 5, 6 и остальными

Таблица 4.2. Результат Эрдеша — Реньи

Размер сети	Вероятность помехи	Вероятность сохранения связности не меньше, чем
50	76,5 %	98 %
100	86,2 %	99 %
1000	97,9 %	99,9 %
10 000	99 %	99,99 %
100 000	99,97 %	99,999 %
1 000 000	99,996 %	99,9999 %

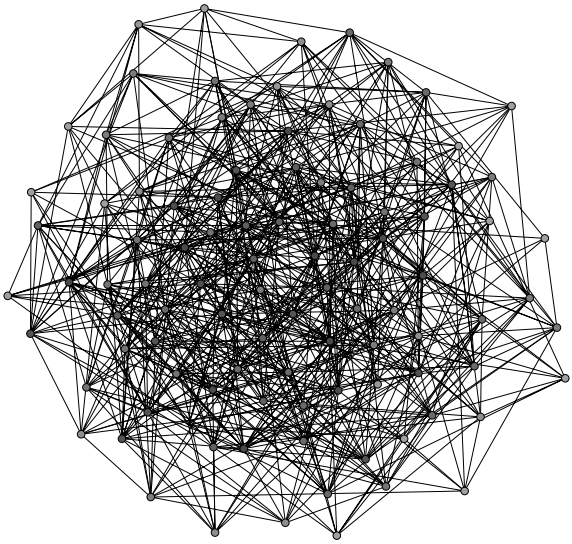

Рис. 4.6. Сеть из 100 компьютеров в виде графа. Вероятность недоступности канала 86 %

Ниже во врезке приведена более общая математическая формулировка результата. Этот текст рассчитан на уровень средней школы, но если вы не хотите вдаваться в подробности, можете его пропустить.

Простейший вариант теоремы Эрдеша — Реньи

Символ $\ln(n)$ означает натуральный логарифм числа n. В данном случае нам важно только то, что если n увеличивается, то $\ln(n)$ тоже увеличивается, но очень медленно.

Теорема Эрдеша — Реньи. Допустим, сеть состоит из n узлов. Предположим, что связь между любыми двумя узлами недоступна с вероятностью $q(n)$ независимо от других связей в сети. Если

$$q(n) \le 1 - \frac{3\ln(n)}{n}, \tag{4.1}$$

то связность сети сохраняется с вероятностью не меньше, чем

$$1 - \frac{1}{n}. \qquad (4.2)$$

В табл. 4.2 во втором и третьем столбце все значения умножены на 100%. Во втором столбце указаны значения, полученные с помощью формулы (4.1). Поскольку $\ln(n)$ растет гораздо медленнее, чем n, то допустимая вероятность помех увеличивается. В третьем столбце — значения (4.2) , то есть $\left(1 - \frac{1}{n}\right) \times 100\%$.

Для многих приложений важно умение работать с сетями, в которых изначально присутствуют не все возможные связи. Например, таковы сети автомобильных дорог, социальные сети и тот же интернет.

Надежность сети, по сути, и есть та вероятность уничтожения отдельной связи в ней, начиная с которой общая связность маловероятна. Для описанной выше ситуации надежность исключительно высока, и это строго доказанный результат.

Фазовый переход

На самом деле теорема Эрдеша — Реньи несколько точнее и еще удивительнее, чем мы описали в предыдущем разделе. Эта теорема выявила интересное явление, которое физики называют *фазовым переходом*. Фазовый переход — это резкий скачок от одного состояния системы к совершенно другому*.

Самый знаменитый фазовый переход — изменение состояния воды в зависимости от температуры. При 0° Цельсия вода превращается в лед, а при 100° — в пар. 0° и 100° — *критические значения,* при которых состояние резко меняется.

---

* В приложении 2 к главе 4 мы приводим математическую формулировку теоремы Эрдеша — Реньи о фазовом переходе.

Нечто похожее происходит и с вероятностью связности сети, если изменять вероятность недоступности каналов. Оказывается, надежность сети меняется не постепенно, а очень резко. Если вероятность помехи меньше критического значения, то с подавляющей вероятностью связность сохраняется. Но стоит хотя бы немного пересечь критическую черту — и сеть почти наверняка распадется.

На рис. 4.7 показан пример сети из 100 компьютеров. Согласно результатам Эрдеша — Реньи, критическая вероятность помех в такой сети равна 95,4 %. Слева вероятность помехи 95 %, то есть меньше критического значения. Как видите, связность сети сохранилась. Мы неоднократно повторили эксперимент, но получить несвязную сеть нам так и не удалось. На рисунке справа вероятность помехи 96 %. И что же? Одна точка оторвалась от сети, связность потеряна. Опять же как мы ни старались повторить эксперимент, связной сети мы не получили ни разу. Результат впечатляет тем, насколько тонкой оказывается грань между связностью и несвязностью!

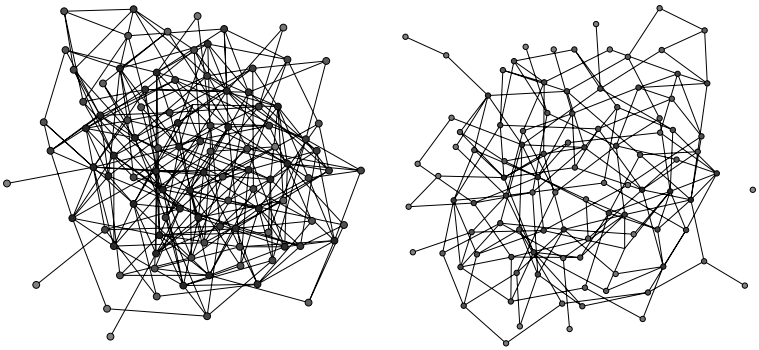

Рис. 4.7. Сеть из 100 компьютеров в виде графа. Слева: вероятность недоступности канала 95 %; связность сохраняется. Справа: вероятность недоступности канала 96 %; связность нарушена

Если вероятность помех в точности равна критической, то произойдет примерно то же самое, что и с водой и снегом при нуле градусов: может получиться и так и эдак. В нашем случае вероятность сохранения связности приблизительно составит 36,79 %.

Критическая вероятность недоступности канала для сети размера n вычисляется по формуле $\left(1-\frac{\ln(n)}{n}\right)\times 100\%$. Для примера мы приводим несколько значений в табл. 4.3.

Таблица 4.3. Результат Эрдеша — Реньи: критическая вероятность помех. Если вероятность помех меньше критической, связность сети сохраняется, а если больше — разрушается

Размер сети	Критическая вероятность помехи
50	92,16 %
100	95,39 %
1000	99,31 %
10 000	99,91 %
100 000	99,99 %
1 000 000	99,999 %

Подобные фазовые переходы типичны для теории случайных графов. Эти результаты самые интересные, потому что многое говорят о природе сетей на практике. Состояние сети — это, как правило, две крайности. Социальная сеть либо становится популярной, либо умирает. Компьютерный вирус распространяется с огромной скоростью и размахом или сходит на нет в самом начале. И реальность, и математика подтверждают: среднего не дано. К сожалению, нам далеко не всегда известны критические значения и главное мы не знаем каким образом удержаться по правильную сторону от фазового перехода.

Как доказывается результат Эрдеша — Реньи

Глубокие математические доказательства часто строятся на очень простых интуитивных идеях. Результат Эрдеша — Реньи — блестящий пример данной закономерности*.

Математики заметили, что наиболее вероятный способ разрушить связность сети — отрезать один узел от всех каналов связи. Группу узлов отрезать гораздо труднее, потому что число каналов, которые связывают ее и остальную часть сети, относительно большое. Маловероятно, что все эти каналы недоступны. Тогда изначально сложный вопрос:

> С какой вероятностью разрушится связность сети?

сводится к гораздо более простому вопросу:

> С какой вероятностью хотя бы один из узлов потеряет все свои каналы связи?

Чтобы доказать, что эти вероятности приблизительно равны, понадобятся длинные и нетривиальные математические выкладки. Но доказать это можно, и усилия оправдываются, потому что второй вопрос гораздо проще первого.

Например, если у нас 100 узлов и вероятность помехи 0,96, то каждый узел может оказаться оторванным от всех 99 других узлов с вероятностью

$$(0,96)^{99}\left(\times 100\,\%\right).$$

Это очень специфическое выражение: число, близкое к единице, возведенное в большую степень. Такие выражения

---

* В приложении 3 к главе 4 мы приводим идею доказательства результата Эрдеша — Реньи в более точной математической формулировке. Подробно это доказательство рассматривается в книге Андрея «Модели случайных графов» [4].

хорошо известны в математике и относятся к так называемым *замечательным пределам*, из которых, по сути дела, и следует результат.

Что мы знаем и чего не знаем о надежности интернета

Результаты Эрдеша — Реньи полностью не решают проблемы устойчивости интернета. Их модель не очень похожа на реальный интернет. Например, в модели Эрдеша — Реньи число линий у разных узлов обычно близко к среднему. В интернете же разброс между серверами очень большой. У одних серверов сотни каналов связи, а у других — всего два-три.

В 2000 году журнал Nature опубликовал статью «Устойчивость к помехам и атакам в больших сетях» [9]. Авторы-физики, в том числе и весьма влиятельный и знаменитый ученый Ласло Барабаши, взяли данные небольшой части интернета и с помощью компьютерных экспериментов решили посмотреть, что произойдет, если по одному выводить из строя серверы.

Результаты получились нетривиальные. Если серверы выходят из строя случайным образом, например из-за помех, то связность сети долго сохраняется. Но если целенаправленно вывести из строя несколько серверов с самым большим количеством каналов связи, то сеть быстро распадется. Вывод звучал сенсационно: интернет надежный и в то же время довольно хрупкий. Он устойчив к помехам, но чувствителен к атакам!

Эта работа быстро приобрела широкую известность. Только инженеры, работающие в этой сфере, с недоумением пожимали плечами: «Не может быть, наши сети очень надежные!» Результаты Барабаши и соавторов вызвали волну

критики, особенно резкая критика прозвучала в вышедшей в 2005 году статье специалистов по телекоммуникациям [12].

Одним из главных аргументов критиков было то, что в интернете ключевыми являются не многоканальные серверы, а те, через которые проходит наибольшее количество информационного трафика. В интернете у некоторых серверов действительно много каналов связи, но зачастую эти серверы находятся на периферии. За трафик в основном отвечает плотная сеть узлов посередине — опорная сеть. Интуитивно понятно, что для того чтобы разрушить столь густую сеть маршрутов, нужно вывести из строя большое количество серверов. Скорее всего, в ближайшем будущем интернет не развалится!

Благодаря широкому взгляду физиков и специализированному анализу информатиков и инженеров мы знаем достаточно много об устойчивости интернета. Но вопрос пока не закрыт.

В идеале нам нужна строгая математическая модель, включающая в себя основные свойства интернета. И для этой модели необходимы результаты типа Эрдеша — Реньи, показывающие, что произойдет, если вывести из строя те или иные серверы или каналы связи. Только тогда у нас будут однозначно правильные, строго доказанные результаты. Продвижение в этих направлениях есть, но до точных ответов пока далеко. Работа продолжается.

Интернет в картинках

Если вы хотите посмотреть, как выглядит интернет, зайдите на сайт проекта Opte по адресу http://www.opte.org/.

Создатель проекта, программист и художник Баррет Лайон, задался целью, казалось бы, невыполнимой — нарисовать интернет! Причем именно в виде графа: серверы и каналы связи между ними.

Серверы принадлежат разным странам и компаниям. Собрать информацию обо всех каналах связи — колоссальная техническая задача. И это только начало. Не менее сложно изобразить все эти гигантские данные на одном понятном глазу рисунке.

Красочное изображение на черном фоне напоминает фейерверк. Линии разных цветов обозначают разные континенты, а плотные, белые, почти светящиеся магистрали между ними — опорная сеть. Рисунки Opte известны по всему миру. Они получили множество наград и выставляются в Музее современного искусства в Нью-Йорке. Смотрите, это интернет!

Глава 5
Сила выбора из двух

Очереди, которых мы не видим

Всем нам хорошо знакомы самые разные очереди. Очередь в кассу супермаркета, очередь в приемной врача, очередь у лифта в большом здании, толкотня перед входом в метро и непереносимые очереди из машин — автомобильные пробки. Но кроме всех этих «живых» очередей, мы, сами того не замечая, стоим в огромном количестве «компьютерных» очередей. Например, это происходит практически каждый раз, когда мы подключаемся к интернету.

Допустим, вы хотите посмотреть какую-то веб-страницу, скажем главную страницу Московского физико-технического института (МФТИ), где работает Андрей. Вы вводите веб-адрес https://mipt.ru в своем браузере (например, Internet Explorer, Fire Fox или Google Chrome). Что же дальше?

Веб-страница — всего лишь файл, похожий на обычный вордовский документ, только в несколько ином формате. Этот файл включает в себя все, что есть на странице: тексты, рисунки, ссылки и тому подобное. Ваш браузер — это фактически интерфейс, который позволяет вам запросить файл с содержанием нужной страницы.

Отправка цифровой информации сравнима с отправкой обычной почты или грузов. Запрос с вашего браузера поступает на так называемый *веб-сервер* под именем mipt.ru, веб-сервер МФТИ. Веб-сервер — это специально выделенный компьютер, который отвечает за поиск нужного файла и его отправку в браузер пользователя. У каждого сайта,

или *домена* — mipt.ru, rzd.ru, google.com и других, — свой веб-сервер, и часто не один.

Получив ваш запрос, веб-сервер МФТИ отправляет в ваш браузер файл с содержанием главной страницы. Дальше вы начинаете ее читать, кликаете на ссылки, загружаете документы. И каждый раз веб-сервер МФТИ отправляет вам новые странички, фотографии, тексты и все остальное. Кроме вас на сайт заходят другие пользователи, с другими запросами. У веб-сервера работы хватает! И когда ее слишком много, ваши запросы должны дожидаться своей очереди.

И это еще не все. Как и почтовая пересылка, информация попадает с одного компьютера на другой не напрямую, а через несколько промежуточных узлов. Каждый узел — это тоже серверы, и там тоже могут образоваться очереди на доставку и отправку. Это происходит при отправке любой цифровой информации, будь то имейл, фотографии на «Фейсбуке» или ваш голос по скайпу.

Насколько длинной будет очередь? Можно ли организовать сервис, например отправку веб-страниц или передачу голоса и видео, так, чтобы задержки были как можно меньше? Этими вопросами занимается специальная область математики под названием теория очередей, или теория массового обслуживания. Среди ее основателей крупные российские математики — Александр Яковлевич Хинчин и Борис Владимирович Гнеденко.

Теория очередей возникла из практики в начале XX века, когда датский математик Агнер Эрланг решил проанализировать работу телефонной станции. С появлением современных телекоммуникаций эта теория получила новое необъятное поле приложений и мощный толчок к развитию. В этой главе мы расскажем только об одной задаче, так называемой *балансировке нагрузки,* и об одном ее относительно недавнем и необыкновенно элегантном решении.

Параллельные серверы

Как вы уже поняли, запросов на веб-сервер поступает множество. Поэтому часто используется не один, а сразу несколько серверов, которые могут обрабатывать запросы одновременно. Серверов может быть очень много. Например, современные поисковые системы, такие как «Яндекс» и Google, получают миллиарды запросов в день. Поэтому они оснащены огромным количеством мощных серверов, занимающих внушительные территории.

Когда вы посылаете запрос, вас совершенно не волнует, какой из параллельных серверов будет его обрабатывать. Это внутренняя кухня веб-серверов. Но для того, чтобы наши запросы выполнялись быстро и эффективно, вопрос налаженной параллельной работы очень актуален.

Схематически мы изобразили параллельные серверы на рис. 5.1. Запросы на рисунке разной величины, потому что все они разного объема. Кому-то нужна страница с коротеньким текстом, а кому-то — годовой отчет на 100 страниц с графиками и фотографиями. Если информации больше, то и времени на ее отправку понадобится больше.

Поскольку серверов много, возникает проблема: на какой сервер отправить ваш запрос? Вопрос распределения заданий между параллельными серверами далеко не тривиален. Например, нельзя допустить, чтобы один сервер был перегружен, а другой простаивал. Желательно распределить нагрузку на них равномерно и свести очереди к минимуму. Как это сделать? В общих чертах это и есть задача о балансировании нагрузки.

Эта задача возникает не только при отправке и пересылке информации. Другой распространенный и очень похожий пример — вычисления на удаленном компьютере. Например, когда в научных вычислениях одним супермощным

компьютером пользуется несколько исследовательских групп или когда мы что-то храним или вычисляем на «облаке». Проблемы возникают, если запросов на вычисления много, вычисления объемные и их невозможно выполнить одновременно.

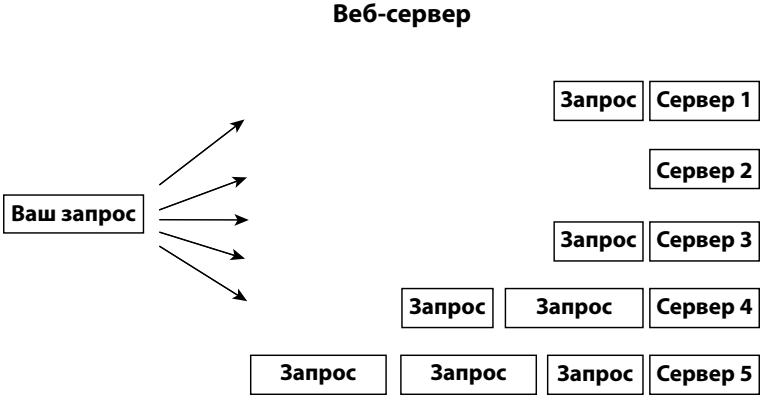

Рис. 5.1. Несколько параллельных серверов. В реальности серверов намного больше. Запросы разной величины содержат разное количество информации, и, соответственно, для их отправки требуется разное время

У задачи балансировки нагрузки много разных решений. И нам опять необходима математика, потому что иначе мы не сможем с уверенностью сказать, какое из решений лучше. Решение, о котором мы расскажем в этой главе, очень простое и красивое, и его эффективность строго доказана. Должны отметить, что российские математики внесли в его получение серьезный вклад.

Какой сервер выбрать?

Возможно, одни серверы в данный момент простаивают, а другие сильно загружены. Например, на рис. 5.1 сервер

5 загружен больше всех. Совершенно очевидно, что лучше всего отправить запрос на второй сервер — тот, у которого самая короткая очередь (в данном случае очереди просто нет). Однако такая стратегия не всегда оптимальна. Может оказаться, что в очереди всего один запрос, зато очень объемный, в то время как в другой очереди пять коротеньких запросов, которые будут выполнены моментально. Все мы наблюдали похожую ситуацию в обычном супермаркете. Лучше встать в очередь из трех человек с маленькими корзиночками, чем оказаться позади одной большой тележки! Иногда супермаркеты даже открывают специальную кассу для тех, у кого мало покупок. И в этой очереди, даже если она длиннее, вас практически всегда обслужат быстрее.

В контексте примера с супермаркетом понятно, что лучше всего выбрать сервер, у которого меньше всего *работы,* то есть тот, что освободится раньше всех. Математически это тоже нетрудно доказать. Однако есть существенная разница с супермаркетом. В супермаркете мы *видим,* у кого сколько покупок, и примерно представляем, какая очередь пойдет быстрее. Чтобы применить такую же тактику на веб-сервере, сервер должен «знать», сколько времени займет обработка каждого запроса в очереди.

Мы привыкли, что компьютеры умные. Тем не менее важно понять, что компьютер не человек, он не может взглянуть и оценить. Он не умеет думать. Все, что он может, — это использовать имеющуюся информацию и включить ее в *протокол,* то есть заранее установленный порядок действий. Это означает: оценить объем каждого запроса при входе, сохранить эту информацию и удалить после выполнения запроса. На это уходит время, тратится память, работа сервера замедляется. Если нам неизвестен объем заданий в очереди, то лучшее, что мы можем сделать, — выбрать самую короткую очередь. Такая стратегия тоже хорошо изучена

математиками. Она очень эффективна, почти оптимальна. Итак, отправляем запрос в самую короткую очередь? Но вы, наверное, уже догадались, что в реальности возникают проблемы. Откуда система «знает» длину очереди каждого сервера на данный момент?

Каким образом получить информацию о загрузке, скажем, 1000 серверов? Можно послать серверам запрос. Но на обмен информацией уйдет какое-то время, и полученные данные будут хоть немножко, но устаревшими. Информация сразу со всех серверов загрузит канал, который нам нужен для обработки запроса. Кроме этого, полученную информацию необходимо где-то сохранить, скорректировать на задержку, просчитать решение. Да, все это происходит быстро. Но в итоге, может быть, выгоднее отправить запрос на любой сервер наугад, зато сразу? В редких случаях вам не повезет и запрос застрянет в очереди. Зато система будет работать быстро, и каналы передачи не будут загружены лишней информацией.

В разных случаях в зависимости от ситуации задача решается разными способами. Один из самых известных и простых называется Power of two choices — сила выбора из двух.

Сила выбора из двух

Давайте еще раз проанализируем два крайних случая. Собирать информацию от всех серверов неудобно, и это занимает много времени. Отправлять запрос наугад, при полном отсутствии информации, тоже плохо, потому что так мы рискуем попасть на самый загруженный сервер. Если крайние случаи нас не устраивают, то, скорее всего, самое лучшее решение лежит где-то посередине. И часто уже первый шаг в этом направлении ведет к серьезным улучшениям.

Что, если мы затребуем информацию о длине очереди не на всех серверах, а только на двух, выбранных наугад, а дальше посмотрим, на каком из них очередь меньше, и именно туда отправим запрос. Ответ от двух серверов придет быстро, и на загрузку системы это практически не повлияет. У этого простого метода есть несколько названий: парадигма двух выборов, метод двух случайных выборов, метод выбора из двух. А иногда его называют моделью супермаркета.

Понятно, что выбирать самую короткую очередь из двух случайных серверов лучше, чем выбирать сервер вслепую. Но насколько лучше? Оказывается, результаты просто несравнимы!

Чтобы воочию убедиться в эффективности выбора из двух, давайте предположим, что серверов много и система загружена на 90 %. Это означает, что каждый сервер простаивает примерно 10 % времени, что вполне разумно. Если загрузка больше, то из-за неравномерности поступления работы и разного времени обработки запросов очереди будут непозволительно большими. При загрузке 90 % и выборе сервера наудачу в среднем на нем будет девять сообщений. А если применить выбор из двух, то средняя очередь составит примерно 2,35 — очередь уменьшилась почти в четыре раза!

Еще интереснее подробный анализ. Сколько процентов серверов пустуют? Какая длина очереди наиболее типична? В табл. 5.1 мы привели результаты расчетов и сравнили процент серверов с очередью 0, 1, 2, 3, 4, 5, 6, 7 и больше при выборе сервера наугад и применении метода выбора из двух. Загрузка по-прежнему равна 90 %. Результаты довольно точные, когда серверов много.

Давайте посмотрим, что означают эти числа. Начнем с того, что и в том и в другом случае 10 % серверов пустуют. Это естественно, потому что система загружена на 90 %. Зато дальше разница совершенно очевидна. При методе выбора

из двух нагрузка распределяется равномерно. Больше чем у половины серверов очередь два или три, а очередь семь или больше почти не встречается. По сравнению с этим картина при случайном выборе сервера довольно печальная. Почти у половины серверов очередь семь и больше!

Таблица 5.1. Процент серверов с очередью 1–7 и больше. Загрузка системы — 90%

Длина очереди	Процент серверов с длиной очереди, как в первой колонке	
	Выбор сервера наугад	Метод выбора из двух
0	10%	10,0%
1	9,0%	17,1%
2	8,1%	25,1%
3	7,3%	27,2%
4	6,6%	16,8%
5	5,9%	3,7%
6	5,3%	0,1%
7 и больше	47,8%	0,0%

Самое поразительное в этой истории то, что минимальное количество информации производит такой колоссальный эффект! Знание очереди всего на двух серверах кардинально меняет картину. Именно эта красивая и нетривиальная идея делает задачу интересной с научной точки зрения.

Кто придумал и обосновал метод выбора из двух

Впервые метод выбора из двух предложили ученые-информатики Дерек Игер, Эдвард Лазовска и Джон Захорджан в 1986 году [13] именно в контексте разделения вычислительной работы между ресурсами. Это практически та же ситуация, что и наш пример с веб-сервером. Ученые

заметили, насколько сильно влияет на систему небольшое количество информации, и обратили внимание на то, что именно выбор из двух (а не из трех или четырех) особенно эффективен.

Казалось бы, открытие сделано? Но, как всегда в науке, это было только начало. Дело в том, что все результаты в статье Игера и соавторов основывались исключительно на так называемом *имитационном моделировании.*

Что такое имитационное моделирование и в чем его сила и недостатки, понять совсем нетрудно. Многие, наверное, играли в компьютерные игры, где надо, скажем, управлять автомобилем. Конечно, никакого автомобиля нет. Но вы проедете по виртуальной дороге за виртуальным рулем, и вам покажут всю статистику: скорость, время, качество вождения и другие показатели. Аналогично можно виртуально изобразить работу серверов и посмотреть, как они будут функционировать при использовании разных методов распределения нагрузки. Это дает представление о результатах без необходимости что-то менять на реальном сервере, что бывает трудно, а порой и невозможно по техническим причинам и чревато немалым количеством ошибок. Точно так же как для большинства из нас Формулу-1 на скорости 250 км/ч лучше проходить виртуально, по крайней мере для начала.

Имитационное моделирование — мощный способ получить информацию о сложной системе. С первого взгляда можно даже подумать, что формулы вообще не нужны и имитационного моделирования достаточно. Но у этого подхода есть ограничения. Во-первых, при изменении параметров системы (например, изменилось количество поступающих сообщений в минуту) все надо пересчитывать заново. Если бы была формула, то новое значение параметра можно было бы просто в нее подставить. Во-вторых, имитационное моделирование не дает ответа на вопрос, *почему* метод так

хорошо работает, а из математического доказательства обычно сразу видно, какие факторы оказали решающее влияние.

Мы никогда не смогли бы просто так взять и за пару минут посчитать результаты, приведенные в табл. 5.1, и никогда бы точно не оценили эффективность выбора из двух, если бы не вклад математиков. Всплеск их интереса к этой теме произошел во второй половине 1990-х, и именно тогда были получены основные результаты.

В 1996 году российские математики Никита Дмитриевна Введенская, Роланд Львович Добрушин и Фридрих Израилевич Карпелевич опубликовали статью «Система обслуживания с выбором наименьшей из двух очередей — асимптотический подход» в журнале «Проблемы передачи информации» [1]. *Система обслуживания* — это наши несколько серверов, которые должны отправить (обслужить) веб-страницы. *Выбор наименьшей из двух очередей* мы уже обсудили — это и есть метод выбора из двух.

На самом деле об *асимптотическом подходе* мы уже несколько раз упоминали, просто не пользовались математической терминологией. Дело в том, что система выбора из двух не поддается точному математическому анализу. Об этом авторы пишут на первой же странице. Проблема в том, что серверов несколько, и это делает математические уравнения такими громоздкими, что их невозможно решить ни на бумаге, ни даже на компьютере. Проблема сильно упрощается, есть предположить, что серверов очень много, бесконечное количество. Тогда самые запутанные части в уравнении настолько уменьшаются, что ими можно пренебречь. А то, что остается, поддается анализу. И хотя он по-прежнему далеко не тривиальный, но уже выполнимый! Это и есть *асимптотический* анализ, от слова *асимптота* — предел, которого нельзя достигнуть. Асимптотические результаты всегда приблизительные. Никто никогда не даст вам бесконечное количество

серверов! Но когда серверов много, асимптотические результаты очень точные, а главное они позволяют нам понять, как работает система. Именно такой асимптотический анализ и сделали авторы статьи. Результаты, приведенные в табл. 5.1, взяты прямо из статьи, мы только подставили числа. При этом отдали должное асимптотическому подходу и оговорились, что результаты применимы, когда серверов много.

Практически одновременно с российскими математиками похожие результаты опубликовал Михаел Митценмахер из Гарвардского университета. Вскоре появилось еще несколько статей, в которых рассматривались разные особенности системы (например, длина очереди на самом загруженном сервере) и предлагались разные подходы к решению. Видимо, это был именно тот случай, когда идеи витали в воздухе!

В 2001 году Михаел Митценмахер с коллегами опубликовал серьезную обзорную статью на эту тему [23], которой мы воспользовались при написании этой главы. Уже на тот момент перечень литературы о выборе из двух и других похожих моделях стал достаточно обширным и продолжает расширяться до сих пор.

Где используется метод выбора из двух

В статье Митценмахера [23] и другой литературе говорится о нескольких применениях метода выбора из двух. Одно приложение мы уже рассмотрели подробно — балансировка нагрузки, когда есть несколько обслуживающих приборов. В нашем примере это были веб-серверы, и мы упомянули пересылку информации через интернет и удаленные вычисления. Но подобная ситуация возникает и во множестве других контекстов. Неслучайно одно из названий модели — модель

супермаркета. В следующий раз не обходите все кассы, окиньте взглядом две наугад и выбирайте самую короткую очередь из двух!

Есть и другие приложения — мы коротко опишем два из них, — тоже связанные с компьютерными технологиями. Первое — хеш-таблицы. Они используются для того, чтобы извлечь нужную информацию из памяти компьютера, который ищет ее по «ключу» — коротенькому коду. Когда информации немного, ключей достаточно. Но иногда происходят «коллизии»: несколько элементов, хранящихся в памяти, пользуются одним и тем же ключом. В этом случае компьютеру надо проверить всю эту группу (которая называется *цепочкой),* чтобы найти то, что нужно. Желательно, конечно, чтобы коллизий было поменьше, а цепочки покороче. Для этого часто пользуются методами типа выбора из двух.

Еще одно применение тоже связано с эффективным использованием памяти компьютера, и этот случай немного похож на наш пример с серверами. Компьютерные вычисления требуют большого объема памяти. Легче всего, когда ее необходимое количество доступно в одном месте. Но это не всегда возможно. В реальности распространены так называемые системы с распределенной памятью, где ею можно воспользоваться на нескольких разных машинах. Методы типа выбора из двух актуальны в нескольких разных вариантах при координации работы систем с распределенной памятью.

В чем секрет силы выбора из двух

Эффективность метода выбора из двух удивительна. Но если задуматься, этот результат вполне логичен, и мы постараемся предложить вам простое объяснение.

Давайте вернемся к рис. 5.1. У сервера 5 самая длинная очередь. Если выбрать сервер наудачу, наши шансы нат-

кнуться на сервер 5 будут 1:5. Это 20 % случаев! При этом шансы попасть на самый лучший сервер, сервер 2, точно такие же.

Теперь представьте, что мы сначала выбираем два сервера, а потом наименьшую очередь из двух. Во-первых, в этом случае на сервер 5 мы в принципе не попадаем никогда. В худшем случае мы попадем на сервер 4. Но для этого мы должны наудачу вытянуть сервер 4 и сервер 5. Два сервера из пяти можно выбрать десятью способами: 1 и 2, 1 и 3, 1 и 4, 1 и 5, 2 и 3, 2 и 4, 2 и 5, 3 и 4, 3 и 5, 4 и 5. Все эти пары равновероятны. Шансы, что из десяти возможных пар мы вытянем именно самую неудачную, 4 и 5, всего 1:10. При этом свободный сервер 2 входит в четыре из десяти пар. Выбрав пару, в которую входит этот сервер, мы непременно на него попадем. Значит, наши шансы попасть на сервер 2 равны 4:10; это уже не 20 %, а 40 %.

Итак, у нас набралось три причины, по которым выбор из двух выгоднее выбора наугад:

1) мы никогда не попадаем в самую длинную очередь;
2) маловероятно, что мы выберем сразу два сервера с длинной очередью;
3) увеличиваются шансы попасть на пустой сервер.

Какая из этих причин самая значительная? Из математических уравнений следует, что причина 2. Представьте, что серверов много и у 20 % серверов длинная очередь. Тогда шансы наткнуться на два таких сервера всего $0,2 \times 0,2 \times 100\,\% = 4\,\%$.

Именно это перемножение вероятностей $0,2 \times 0,2$ и приводит к совершенно другому решению уравнения, согласно которому длинные очереди становятся практически невероятными*.

---

* В приложении к главе 5 мы приводим более формальное математическое объяснение и формулы для подсчета данных, приведенных в табл. 5.1.

Так получилось, что Нелли, будучи аспиранткой, присутствовала на одном из первых докладов Никиты Введенской о силе выбора из двух на конференции в Израиле в 1996 году. Когда Никита Дмитриевна объяснила решение, оно оказалось таким красивым и простым, что создавалось впечатление, что каждый мог догадаться! В перерыве Никита Дмитриевна сказала: «На самом деле это так просто, что можно объяснять студентам!» Это был блестящий пример математической работы — красивой и полезной!

Спустя почти 20 лет, на конференции в Стамбуле в 2015 году, пленарный доклад делала Кавита Раманан из университета Брауна в США. Кавита — блестящий математик с огромным количеством фундаментальных работ и профессиональных наград. На одном из первых слайдов она сослалась на работу Введенской, Добрушина и Карпелевича. Доклад Кавиты тоже был о методе выбора из двух. Но на данный момент эти исследования уже перешли на иной уровень обобщения и абстракции. Кавита занимается очень сложными асимптотическими процессами, ее доклад выходит за рамки нашей книги. Проблема развивается и ставит перед математиками новые задачи, еще красивее, еще сложнее и, кто знает, возможно, еще полезнее.

Глава 6
Секретные числа

Массовый обмен шифровками

Классическая музыка по радио прервалась, и голос диктора стал зачитывать цифры, которые Штирлиц быстро записывал в аккуратные колонки. Диктор называл цифры привычно сухо и четко. «Для него эти цифры всего лишь цифры», — подумал Штирлиц. Когда сообщение закончилось, Штирлиц взял с полки томик Шиллера, открыл на нужной странице и начал превращать цифры в слова. «Центр — Юстасу…»

Сообщение передавалось по открытому радиоканалу, но прочитать его мог только Штирлиц, потому что только он знал, как расшифровать переданные цифры. Шифрование — это не что иное, как сокрытие информации от посторонних.

Шифрование в том или ином виде существует много тысячелетий. Однако в середине XX века произошла своего рода революция. Если раньше шифровками пользовались, как правило, представители государственных спецслужб (или, если угодно, сами спецслужбы и даже государства), то к 70-м годам XX века стало ясно, что совсем скоро шифрование понадобится самым обычным людям, причем не изредка, а буквально каждый день. Это связано с лавинообразным развитием технологий, плоды которых доступны каждому: компьютеры, сотовые телефоны и тому подобное.

Каждый раз, когда вы вводите свой пароль или номер кредитной карты на сайте, вы отправляете личную конфиденциальную информацию по открытым каналам интернета. У многих к этим каналам есть доступ, например у вашего

интернет-провайдера. В принципе перехватить ваше сообщение может даже компьютерщик-любитель с обычным ноутбуком и подходящим для этой цели программным обеспечением. Конфиденциальность информации обеспечивается именно тем, что она передается в виде шифровки. Вы можете легко узнать сайты, на которых действует протокол безопасной передачи данных: в этом случае веб-адрес начинается с *https://... HTTP* — обычный протокол передачи данных по интернету. А дополнительная буква *S* происходит от английского слова *secure* (*безопасный*) и означает, что данные будут передаваться в зашифрованном виде.

Каким образом зашифровывается и расшифровывается ежедневный гигантский поток конфиденциальной информации? Естественно, математика, как всегда, опережала технологии и стояла у их истока. Задачами шифрования занимается *криптография* — очень активная и интересная область математики и информатики*.

Ключ к шифру

В зашифрованном сообщении каждая буква заменяется какой-либо другой буквой, числом или знаком. Например, возьмем самый простой шифр. Будем зашифровывать каждую букву следующей буквой алфавита. Вместо А напишем Б, вместо Б — В и так далее, а вместо Я — А. Например, слово *ПРИВЕТ* будет выглядеть так:

РСЙГЁУ

* Заметим, что не нужно путать шифрование и кодирование. Как мы уже рассказывали в главе 3, задачи теории кодирования состоят вовсе не в том, чтобы скрыть от кого-либо информацию. Коды строятся прежде всего для того, чтобы обеспечить бесперебойную передачу данных, в том числе и тех, которые подвержены помехам и искажениям. А вот шифрование — это как раз «сокрытие информации от посторонних». Этим и занимается криптография.

Это очень простой шифр, потому что каждая буква всегда зашифровывается одной и той же буквой, и взломать его — пара пустяков. Достаточно угадать одно слово в сообщении. Например, мы догадались, что сообщение начинается со слова «привет», и вот в нашем распоряжении уже шифры для шести букв: П, Р, И, В, Е и Т. С их помощью мы можем угадать другие слова, пока наконец не расшифруем весь алфавит. Именно так расшифровал секретные послания Шерлок Холмс в рассказе «Пляшущие человечки».

Конечно, любой серьезный шифр гораздо сложнее. Например, одна и та же буква, скажем А, может каждый раз обозначать разные буквы. Или, как в фильме «Семнадцать мгновений весны», буквы могут быть зашифрованы с помощью цифрового кода. Понятно, что у Штирлица в сборнике Шиллера были не стихи, а *ключи* для расшифровки секретных сообщений. Если бы у Штирлица не было этой книги, то для него, как и для диктора, цифры так и остались бы только цифрами.

В протоколах интернета ключом может быть всего лишь одно число, которое необходимо «подставить в формулу» для расшифровки. В данном случае «формула» — это очень длинная последовательность достаточно сложных операций.

Если вы сумели перехватить или вычислить ключ противника, то все его секреты — в вашем распоряжении. В этой главе мы расскажем, как с помощью математики удается генерировать ключи для передачи конфиденциальной информации по интернету. Причем завладеть этими ключами невозможно или по меньшей мере очень дорого — для это требуются колоссальные компьютерные ресурсы.

Но для начала давайте познакомимся со знаменитой шифровальной машиной «Энигма», которая использовалась немецкой армией во время Второй мировой войны. Это один из самых изощренных шифров той эпохи, когда шифрование еще не было таким массовым и обыденным явлением. А ключ к этому шифру нашел английский математик Алан Тьюринг.

Алан Тьюринг и «Энигма»

Фильм «Игра в имитацию» (2014) — именно об этой потрясающей истории. К сожалению, в нем не объясняется, как устроена «Энигма» и как Тьюринг взломал этот шифр. Наш рассказ хотя бы частично восполнит эти упущенные подробности. Мы воспользуемся видео Кембриджского университета*. Очень рекомендуем посмотреть их, чтобы увидеть «Энигму» в действии.

«Энигма» совсем небольшая, по размеру сравнима с печатной машинкой. Снаружи она состоит из клавиатуры и панели, на которой расположены буквы с подсветкой. Если нажать букву на клавиатуре, скажем *A*, то на панели высветится другая буква, например *Q*. Это означает, что в шифровке в этом месте вместо *A* появится *Q*. Внутри у машины три вращающихся диска, и их положение меняется после набора каждой буквы. Диск повернулся, провода соединились по-другому, и когда мы в следующий раз нажимаем *A*, на панели высвечивается уже не *Q*, а, скажем, *G*.

В набор «Энигмы» входят пять дисков, использовать можно любые три, в любом порядке. У каждого диска — 26 изначальных положений. И это еще не все. В военном варианте у «Энигмы» была передняя панель с буквами и 10 кабелей. Каждый кабель соединял две любые буквы между собой, и при шифровании они менялись местами. Диски можно было перебрать достаточно быстро, но количество комбинаций на панели было настолько велико, что перебрать их было невозможно. Всего у «Энигмы» было

158 962 555 217 826 360 000

---

* См. https://www.youtube.com/watch?v=G2_Q9FoD-oQ; https://www.youtube.com/watch?v=V4V2bpZlqx8.

возможных изначальных установок. Каждые сутки ровно в полночь они менялись. Новое изначальное положение дисков, новые пары букв на панели. Перебрать все комбинации за 24 часа было совершенно нереально. Шифр считался неуязвимым.

Зашифрованное сообщение передавали по радио. У немецкого офицера, который его получал, была такая же машина. Кроме того, он имел секретный документ — установки «Энигмы» на каждый день текущего месяца. Это был ключ к шифру машины. Офицер соединял нужные пары букв, ставил диски в заданное исходное положение, набирал *Q,* и на панели высвечивалась *A*. Поступая и дальше таким образом, он расшифровывал секретное сообщение.

Перехватить документ с установками хоть и было непросто, но иногда удавалось, а потом месяц заканчивался и разгадать шифр опять не представлялось возможным.

Алан Тьюринг и его команда совершили настоящий прорыв. Они научились разгадывать шифр каждое утро всего за 20 минут! Мы только вкратце объясним, как им это удалось.

Прежде всего в шифре было слабое место. Буква никогда не превращалась сама в себя. Это стало хоть какой-то зацепкой. Важным оказалось и предположение, что сообщения утром начинаются с чего-то однотипного, например с прогноза погоды. По-немецки *wetterbericht.* Зная, что буква не превращается сама в себя, мы можем найти слово в начале шифровки, которое в принципе может обозначать *wetterbericht.* Теперь нужно отыскать такую изначальную установку, чтобы шифр совпал с расшифровкой.

Очень важно определить, какие буквы соединены в пары, потому что здесь вариантов особенно много. Поскольку никаких сведений нет, начинать приходится наобум. Зато дальше из первой догадки следуют сразу несколько других.

Для ускорения процесса Алан Тьюринг сделал две существенные вещи. Во-первых, он понял, что если пара букв оказалась неправильной, то и все другие пары, следовавшие из нее, тоже неправильные. А значит, их уже не надо проверять. Во-вторых, он построил огромную машину, которая с помощью электрического тока позволяла исключить все неправильные пары одновременно. Оставалось только повторить операцию для каждой позиции дисков, а на это уходило всего 20 минут.

Интересно, что принцип решения Тьюринга заключался не в том, чтобы найти правильный вариант, а в том, чтобы исключить неправильные варианты и сделать это максимально быстро! Это была огромная работа и колоссальное достижение, сильно повлиявшее на ход Второй мировой войны.

Заметим, кстати, что в математике есть понятие «машина Тьюринга», но это вовсе не та машина, которая вычисляла ключ «Энигмы». «Машина Тьюринга» — абсолютно абстрактная концепция, формально описывающая работу компьютера. Это очень важная фундаментальная концепция в математике и информатике, но она выходит за рамки нашей книги.

Сила абстрактного подхода к шифрованию

Те или иные математические методы применялись в шифровании уже давно. Тем не менее, когда задача становится по-настоящему масштабной, возникает потребность в системном подходе. Если раньше задача решалась по-разному в каждом отдельном случае, то теперь появляется теория, из которой следует не один, а целый класс методов. Эта теория начинает широко опираться на другие теории, результаты

осмысливаются, фильтруются и уже в стройном виде вливаются в практику и попадают в университетские учебники.

В наше время криптография — это устоявшаяся наука. На языке, принятом в ней, задача шифрования звучит так. Есть два человека — Алиса и Боб (*A* и *B*), — которые желают передавать друг другу сообщения, но при этом не хотят, чтобы злонравная Ева (*E* от английского слова *eavesdropping — подслушивание*), перехватившая их, смогла прочесть, о чем в них идет речь.

Чтобы было легче понять научный подход к проблеме, давайте шифровать цифры вместо букв. В конце концов, мы всегда можем заменить буквы на числа (хотя бы 1, 2, ... 33). Разумеется, математикам так удобнее.

Что такое шифр? Это превращение одного числа в другое. На входе мы вводим число *x,* а на выходе получаем число *y.* Число *y* — это *преобразование* числа *x.* В математике это записывается известным со школы обозначением

$$y = f(x).$$

Возьмем тот же простой пример, когда буква заменяется следующей по алфавиту. Аналогично Алиса и Боб могут договориться о замене целого числа следующим по порядку: вместо 1 писать 2, вместо 2 — 3 и так далее. Тогда для произвольного числа *x* наш процесс шифрования будет выглядеть как на рис. 6.1 сверху.

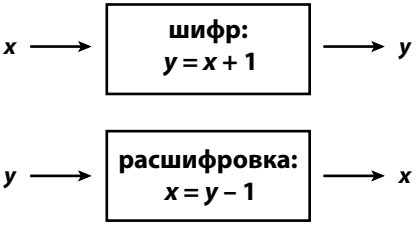

Рис. 6.1. Элементарная шифровка и расшифровка. В данном случае y = f(x) = x + 1

Теперь представим, что Ева умна и хитра и в принципе в состоянии перехватить или вычислить секретный ключ (в данном случае ключ — это договоренность, что Алиса и Боб пишут $x+1$ вместо x). Тогда ей ничего не стоит расшифровать сообщение, она должна всего лишь вычесть единицу!

Понятно, что столь элементарный шифр нас не устраивает. Если Алиса хочет защитить свои сообщения от Евы, то в идеале ей нужен такой шифр, который зашифровать было бы просто, а расшифровать трудно. О том, как будет расшифровывать Боб, мы расскажем ниже. Пока, в терминах математики, нам следует найти очень особенное преобразование f, удовлетворяющее вот такому довольно странному условию: *если мы знаем x и знаем f, мы легко можем вычислить $y = f(x)$, а вот если мы знаем y и знаем f, то вычислить x все равно очень сложно.*

Схематически наше пожелание выглядит как на рис. 6.2. И сразу возникает закономерный вопрос: *а существуют ли в природе подобные преобразования?*

Если задуматься, станет ясно, что придумать подходящую функцию f и главное *доказать,* что она обладает нужными свойствами, вовсе не легко! Почему «главное — доказать»? Совсем не потому, что математики так уж любят доказательства. А потому, что тогда мы точно будем знать, что хитроумная Ева не сможет взломать шифр. Против математических доказательств Ева бессильна.

Оказывается, такие преобразования f есть. Более того, можно сделать так, чтобы Боб и Алиса могли расшифровывать сообщения, а Ева — нет! Пока не доказано, что задача расшифровки абсолютно безнадежна. Но известно, что она относится к определенной категории трудных задач, эффективное решение которых еще не найдено. Именно эти шифры

используются для обмена секретными ключами, чтобы установить безопасную связь через интернет. Для построения таких шифров требуется глубокая математика. В данном случае — *теория чисел.*

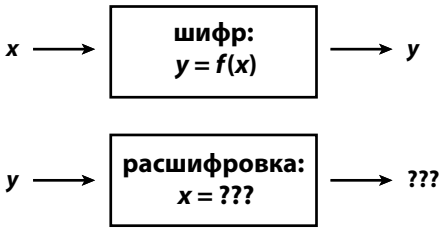

Рис. 6.2. Шифр y = f(x), который сложно расшифровать, даже зная преобразование f

Простые числа

Как вы уже поняли, шифрование имеет дело с числами и преобразованиями чисел. Этими вопросами в математике занимается *теория чисел,* один из классических, даже древних разделов математики. Казалось бы, зачем изучать числа? Лучшие математические умы занимались этой наукой, движимые чистым любопытством. Просто потому, что математики любят числа и любят отыскивать их скрытые закономерности. Умственные изощрения математиков с веками усложнялись без какой-либо перспективы для серьезного применения. И вдруг...

И вдруг в XX веке весь мир перешел на цифровые технологии! Наш бизнес, наше информационное обеспечение, даже наше общение и, конечно, наши секреты — все кодируется, шифруется и пересылается в виде чисел! Любимое хобби математиков неожиданно превратилось в жизненно важный источник знаний об основе современной жизни — числе.

В теории чисел особую роль играют так называемые *простые* числа. Простые числа знакомы нам со школы. Это числа, которые делятся только на единицу и на само себя. Многие узна́ют этот знаменитый ряд:

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31...

Все простые числа, кроме двойки, нечетные. И это понятно, потому что четные числа делятся на два.

Еще со времен древнегреческого математика Евклида известно, что простых чисел бесконечное множество. Однако устроены они крайне сложно, и в теории чисел было брошено немало усилий на изучение их свойств. Ниже во врезке для интересующегося читателя мы приводим несколько любопытных фактов, касающихся простых чисел. Этот материал не требует математической подготовки, но и не влияет на наш дальнейший рассказ, поэтому при желании можете его пропустить.

Несколько интересных свойств простых чисел

Простых чисел, которые не превосходят n, примерно $\frac{n}{\ln(n)}$ штук, где $\ln(n)$ — это натуральный логарифм. Заметим, что $\ln(n)$ растет гораздо медленнее, чем n. Получается, что простые числа попадаются довольно часто. По этой оценке, их количество от одного до миллиарда равно примерно 48 254 942. На самом деле таких чисел 50 847 534[a], то есть оценка вполне точная. Первым, кто добился серьезных продвижений в получении подобных оценок, был наш замечательный математик Пафнутий Львович Чебышев (1821–1894).

Есть и такой забавный результат: между x и $2x$ всегда находится простое число. Это так называемый постулат Бертрана. Однако проблема его уточнения очень сложна, а именно: верно ли, что между x и $1{,}1x$ есть простые числа (хотя бы при больших x)? Верно ли, что они есть между x и $x + \sqrt{x}$? Первое верно, а о втором никто не знает.

Все верят в бесконечность числа «простых близнецов»: 11 и 13, 17 и 19, 29 и 31 и так далее. Но пока никто не может это доказать.

[a] *Источник:* Prime Pages https://primes.utm.edu/howmany.html.

Важно понимать, что никаких формул для поиска простых чисел не существует в принципе! Простые числа распределены очень хитро. Как мы скоро увидим, они крайне важны на практике, особенно сверхбольшие простые числа. Целая индустрия, привлекающая как математиков, так и программистов, занимается построением все новых и новых простых чисел. Так, в январе 2016 года было найдено очередное простое число, равное $2^{74207281}-1$. В нем 22 338 618 знаков! Нечто запредельное…

Открытый обмен ключами

В настоящее время особенно популярны так называемые *схемы шифрования с открытым ключом.* Мы расскажем о *схеме Диффи — Хеллмана,* которая датируется 1970-ми годами и активно используется на практике. Эта схема основана на глубокой математике, но саму идею понять совсем несложно.

Возьмем простое число p. В реальности оно должно быть огромным, но для примера выберем маленькое простое число 19. Теперь выберем еще одно число g, тоже целое, но необязательно простое и меньше p. Например, $g = 2$.

Числа p и g знают все: и Алиса, и Боб, и даже Ева. Теперь Алиса выбирает число x, например $x = 6$, и хранит его в тайне. Боб выбирает число y, скажем $y = 8$, и тоже никому его не сообщает.

Дальше начинается шифрование. Алиса находит остаток от деления на p числа g^x:

$$2^6 = 64,$$
$$64 : 19 = 3 \text{ и } 7 \text{ в остатке.}$$

Боб делает то же самое со своим задуманным числом:

$$2^8 = 256,$$
$$256 : 19 = 13 \text{ и } 9 \text{ в остатке.}$$

Итак, наше преобразование выглядит следующим образом: возвести 2 в степень x и взять остаток от деления на 19. Мы изобразили это преобразование в общем виде на рис. 6.3. Напомним, что числа g и p известны, а число x выбрала Алиса.

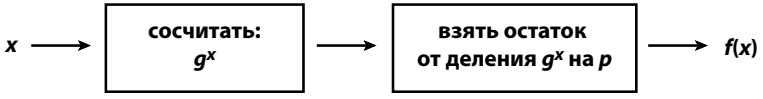

Рис. 6.3. Преобразование по схеме Диффи — Хеллмана $x \to f(x)$. Числа g и p известны, а число x выбрала Алиса

Здесь принципиально важно, что p простое число, а g меньше p, потому что в этом случае g^x на p не делится, то есть остаток от деления не может быть нулем. Есть и более глубокие причины, почему p должно быть простым. Кроме того, для заданного p есть набор «подходящих» g*.

Получив остаток от Боба, Алиса возводит его в степень x и снова берет остаток от деления на p. В нашем маленьком примере Алиса получила от Боба число 9, а Боб от Алисы число 7. Тогда у Алисы получается:

$$[\text{остаток от деления } 9^6 \text{ на } 19] = 11.$$

Боб действует аналогично. Он получил остаток от Алисы, и у него есть известный ему одному y (даже Алиса его не знает!). Он возводит полученный от Алисы остаток в степень y и снова берет остаток от деления на p:

$$[\text{остаток от деления } 7^8 \text{ на } 19] = 11.$$

---

* В приложении 2 к главе 6 мы более подробно рассказываем о выборе p и g.

Это то же самое число 11, что и у Алисы!

Естественно, это неслучайно. Математически нетрудно показать, что Алиса и Боб получат одинаковое число при любых p, g, x, и y*.

Зашифровать можно. Расшифровать нельзя!

Вычисление $f(x)$ по схеме Диффи — Хеллмана — сравнительно быстрая операция, которую можно осуществить и на домашнем компьютере, даже если p огромное, скажем стозначное число. Тем не менее лучшие математики мира вот уже несколько десятилетий бьются над построением алгоритма, который, наоборот, по остатку от деления на p выдавал бы значение x. И ничего не получается!

Самые быстрые алгоритмы работают месяцы, даже будучи параллельно запущенными на тысячах мощных компьютеров по всему миру. Ниже мы расскажем о совсем недавней наиболее успешной попытке решения. Но, как мы увидим, этот метод тоже не «быстрый», а скорее «в принципе выполнимый» и сильно опирается на конкретную имплементацию шифра.

Конечно, это не значит, что по-настоящему быстрого алгоритма не существует. Но задача оказалась настолько сложной, что схема очень активно используется на практике, и если выбрать число p по-настоящему большим, то можно не бояться взлома шифра.

В нашем маленьком примере у Алисы и Боба после открытого обмена сообщениями появился общий секретный ключ, число 11. С его помощью Алиса и Боб могут спокойно зашифровывать и расшифровывать сообщения и передавать

---

* В приложении 1 к главе 6 мы приводим простое доказательство, что по схеме Диффи — Хеллмана Боб и Алиса получат одно и то же число.

их друг другу. Но Еве этот ключ не заполучить, потому что она не сможет вычислить ни x, ни y.

Красота схемы в том, что Алиса и Боб обмениваются ключами через открытый канал. Ева может перехватить все сообщения до одного, но ей это ничего не даст!

Именно поэтому подобные схемы называются «открытым обменом ключами». Нет никакого секрета в том, как работают эти схемы. При этом несанкционированная расшифровка ваших сообщений — проблема посложнее «Энигмы». Секретность без секретов.

На практике часто используется еще одна схема открытого обмена ключами — алгоритм RSA, названный так по первым буквам имен своих авторов: Ривеста, Шамира, Адлемана (Rivest, Shamir, Adleman). Для интересующегося читателя ниже во врезке мы кратко объясняем основной принцип его работы. Если вы не хотите вдаваться в детали, можете пропустить этот текст.

Алгоритм RSA

Алиса может выбрать не одно число, а сразу пару чисел p, q. Можно считать, что p и q — простые. Возьмем очень простое преобразование

$n = p \times q$.

Это довольно быстрая операция — обычное умножение. Однако если вам дано натуральное число n и даже известно, что $n = p \times q$ с некоторыми простыми p и q, которых вы не знаете, то восстановить эти простые числа вам не удастся! Вернее, на это уйдут годы, коль скоро n достаточно велико. Вот такие чудеса! Принцип здесь тот же самый. Для вычисления n используется простое умножение, это очень быстрая операция. А вот обратный процесс, операция разложения на множители, носит в математике красивое имя *факторизация* (от англ. *factor* — сомножитель) и представляет собой большую проблему с точки зрения вычислений.

Добавим, что операцию разложения на множители легко выполнить на квантовом компьютере. Но пока у таких компьютеров очень ограниченный регистр, то есть они могут работать только с относительно маленькими числами. Для шифрования на практике используются очень большие числа. Поэтому на данный момент квантовые компьютеры особой угрозы не представляют.

Практика шифрования

Схемы типа Диффи — Хеллмана и RSA работают довольно медленно, в отличие от так называемых *симметричных криптосистем*, которые гораздо быстрее. Но они пользуются ключом, который нужно хранить в тайне. В определенном смысле это аналоги «Энигмы». Они шифруют с помощью ключа и большого количества сложных преобразований, и знание ключа необходимо для расшифровки.

Итак, у нас есть быстрые схемы, для которых нужен ключ, и медленные — для обмена ключами. Поэтому обычно сессия безопасного соединения через интернет состоит из двух этапов.

Сначала происходит так называемое рукопожатие, то есть обмен сообщениями, во время которого, в частности, устанавливаются ключи. Именно на этом этапе применяются схемы с открытыми ключами, Диффи — Хеллмана или RSA. У «рукопожатия» есть еще несколько целей — например, часто используется *аутентификация:* сервер хочет убедиться, что он общается с тем, с кем думает, а клиент — что он общается с правильным сервером.

После того как ключи установлены, в течение всей оставшейся сессии сообщения зашифровываются с помощью симметричных криптосистем. В духе эпохи интернета всем хорошо известно, как эти системы работают. Если у Евы есть ключ от вашей сессии, то все ваши секреты у нее в кармане!

В предыдущих разделах мы говорили, что за ваши ключи можно не волноваться. На страже стоят сложные математические операции, которые никто не умеет выполнять. И это правда. Тем не менее качество шифрования зависит не только от математики, но и от имплементации.

Хорошо известно, что для надежного шифрования нужно пользоваться очень большими простыми числами. Стандартная величина 1024 бита, то есть 2^{1024}. Речь идет о 308-значных числах! Таких простых чисел очень много, выбор большой. Но, к сожалению, многие протоколы пользуются одними и теми же числами, снова и снова. Просто потому, что так легче. Насколько это угрожает нашей безопасности?

В 2015 году вышла статья [8], которая сильно всколыхнула мир криптографии. Авторы впервые назвали цену взлома наиболее распространенной имплементации схемы Диффи — Хеллмана: 100 миллионов долларов!

100 миллионов долларов за число

Помните, как Алан Тьюринг взломал шифр «Энигмы»? Он построил большую машину, которая смогла быстро вычислить ключ. У этой истории две принципиальные составляющие: «машина» и «быстро».

По сути, до сих пор так все и осталось. Можно сказать, что подход Тьюринга — это прообраз современного подхода к взламыванию шифров. Нужно наличие машины и способ быстрого подсчета ключа. Машину строить необязательно, можно купить суперкомпьютер, были бы средства. Основная загвоздка, как вы уже поняли, именно в быстром способе подсчета.

И вот тут авторы статьи [8] совершили прорыв. На основной конференции по компьютерной безопасности в 2015 году они представили новую атаку Logjam. Фактически это метод вычисления ключа в схеме Диффи — Хеллмана, при котором

сами сообщения нужны лишь на последнем этапе. Эти последние шаги можно совершить очень быстро. Для основной части вычислений необходимо знать только простое число *p*. Поэтому основные — колоссальные — вычисления можно сделать заранее и держать ответы наготове. А дальше, как только начнется обмен сообщениями по открытому каналу, остается всего лишь их перехватить и с их помощью быстро завершить вычисление ключа.

Насколько колоссальны предварительные вычисления? Авторы просчитали, что если закупить оборудование стоимостью в пару сотен миллионов долларов, то за год можно взломать одно 308-значное простое число! Только одно. Предлагаем на секунду остановиться и еще раз оценить сложность задачи.

Реальна ли эта угроза? Скажем так: Logjam не для домашнего применения. Но, например, Агентству национальной безопасности США пара сотен миллионов вполне по карману. И если верить авторам, эта сумма примерно соответствует бюджету агентства на компьютерную криптографию.

Стоит ли оно того? Да. По крайней мере если большинство протоколов пользуются одними и теми же 308-значными числами, что и происходит на практике. Авторы утверждают*, что взломав одно наиболее распространенное 308-значное простое число, можно пассивно прослушивать примерно две трети трафика VPN и примерно четверть SSH-серверов. А если взломать еще и второе по популярности число, то можно добраться приблизительно до 20% от топ-миллиона HTTPS-сайтов.

По предположениям авторов, Агентство национальной безопасности в самом деле применяет подобные методы для прослушивания зашифрованного трафика. И многие верят, что так оно и есть. Вполне вероятно, что службы государственной безопасности продвинулись в криптографии гораздо дальше ученых.

---

* https://freedom-to-tinker.com/blog/haldermanheninger/how-is-nsa-breaking-so-much-crypto/.

Криптография окружена секретностью. Например, в 1996 году адвокат посоветовал американскому профессору не допускать к занятиям по криптографии иностранных студентов, потому что, оказывается, есть закон, запрещающий делиться алгоритмами криптографии с иностранцами, даже если эти алгоритмы описаны в учебниках и любой студент в состоянии их запрограммировать![*] А аспиранту Университета Беркли, который хотел опубликовать новые результаты по криптографии, пришлось четыре года судиться с американским правительством, пока очередной суд не признал, что запрет публикации противоречит Первой поправке к Конституции США, которая не позволяет создавать законы, противоречащие свободе слова[**].

На самом деле трудно понять, где заканчивается свободная наука и начинается государственная тайна. На конференции французский профессор криптографии Роберт Эрра с улыбкой сказал: «Один профессор, звезда в нашей области, утверждал, что редко встречал криптографа, который не пытался бы взломать RSA. Но, честно, если бы мне это удалось, я не отправил бы статью прямиком в журнал, а сначала позвонил бы министру обороны!»

Интересно, кстати, что ученые-криптографы изо всех сил пытаются взломать шифры. Зачем? Чтобы найти слабинку и придумать новые более надежные способы шифрования. Например, авторы Logjam рекомендуют пользоваться еще большими, 616-значными, простыми числами. А если это технически невозможно, то советуют по крайней мере выбирать разные простые числа. Сразу после появления статьи многие браузеры обновили протоколы, в том числе Chrome и Firefox[***]. Наши секреты должны оставаться в безопасности, даже если они транслируются по открытым каналам по всему миру.

---

[*] http://www.cnet.com/news/crypto-class-bans-foreign-students/.

[**] https://www.eff.org/fr/cases/bernstein-v-us-dept-justice.

[***] https://www.schneier.com/blog/archives/2015/05/the_logjam_and_.html.

Глава 7
Счетчики с короткой памятью

Большие данные

Вы, наверное, слышали о понятии *большие данные.* Существуют самые разные определения, но, по сути, термин «большие данные» означает, что нам ничего не стоит получить огромное количество разной информации. И сразу возникает вопрос, как ее обработать. Например, банковские платежи совершаются через интернет или по электронным картам. Современный банк может без особых усилий собрать и сохранить данные о каждой трансакции.

Казалось бы, теперь мы можем ответить на любые вопросы. Какие продукты наращивают популярность? Насколько широко используются наши услуги, карты, банкоматы? Но, оказывается, разобраться в миллионах электронных записей не так уж просто, даже при наличии самых мощных компьютеров. Особенно если ответы нужны прямо сейчас!

Большие данные сами по себе не дают ответа на все наши вопросы. Их анализ влечет за собой множество проблем.

В этой главе мы расскажем только об одной такой проблеме. Звучит она совершенно элементарно, но для ее решения понадобилась новая и далеко не тривиальная математическая теория. Почему? Потому что, как и во многих других задачах с большими данными, ее наивное решение приводит к абсолютно невыполнимым требованиям к компьютерной памяти.

Чтобы было понятно, мы расскажем о самой проблеме чуть позже, в разделе «Раз, два, три, четыре, пять...». А сначала вкратце объясним, как устроена компьютерная память и почему, несмотря на ее мощь и колоссальный объем, она все-таки не всесильна.

Компьютерная память

Мы уже рассказали в главе 3, что вся информация в компьютере записывается в виде нулей и единиц. Каждый ноль и единица физически занимают крошечную ячейку памяти. Чтобы сохранить одну цветную фотографию на 3 мегабайта, понадобится

$$3 \times 8 \times 2^{20} = 25\,165\,824,$$

то есть больше 25 миллионов таких ячеек.

Обычно память работает на полупроводниках. Но мы будем просто считать (для нашего рассказа этого достаточно), что компьютерная память — это физическое устройство, где каждый символ (0 или 1) занимает место. Такие устройства могут быть самыми разными, например CD, DVD или флешка. В любом компьютере информация записывается на жесткий диск. Качество устройств памяти непрерывно улучшается. На жестком диске современных ноутбуков может умещаться, скажем, 320, а то и больше гигабайтов. Этого хватит, чтобы сохранить более ста тысяч цветных фотографий!

Если вам нужно еще больше памяти, можно хранить данные на удаленном сервере. Многие компании, в том числе Google и Amazon, предлагают подобные услуги. Это очень популярный сервис. Возможно, вы тоже что-то храните «в облаке».

У таких информационных гигантов, как Google и «Яндекс», есть свои *дата-центры*. Если вы хотите получить

представление о том, как выглядят дата-центры «Яндекса», можете зайти, например, на сайт https://yandex.ru/company/technologies/datacenter. Фактически дата-центры — это целые комплексы, которые занимают большие территории, где каждое строение либо заполнено серверами, либо предназначено для их энергоснабжения, охлаждения и обслуживания.

Компьютерная память сама по себе пока еще не является ограниченным ресурсом. Дисков и серверов мы можем построить достаточно. Вопрос в том, как воспользоваться всей этой информацией. И вот здесь возникают проблемы, потому что диск не в состоянии «сообразить» или «вспомнить». Чтобы извлечь информацию, компьютер должен найти нужную ячейку памяти.

На доступ к информации на жестком диске или сервере уходит какое-то время. Это можно представить себе примерно так. Предположим, вы читаете дома книгу на английском языке со словарем. Только словарь лежит в библиотеке! И вы бегаете в библиотеку и обратно за каждым словом, причем каждый раз аккуратно ставите словарь обратно на полку и в следующий раз заново ищете его по каталогу. Конечно, компьютер найдет информацию на диске быстрее, чем вы добежите до библиотеки. Но и операций он выполняет гораздо больше, поэтому сравнение вполне уместное.

Без сомнения, намного удобнее, когда словарь лежит рядом с вами на диване. Для этого у компьютера есть так называемая *оперативная* память. Информация считывается с диска, обрабатывается в оперативной памяти, после чего результаты снова сохраняются на диске, а оперативная память освобождается для следующей задачи.

Доступ к оперативной памяти происходит очень быстро, несравнимо с жестким диском. Но именно поэтому ее объем существенно ограничен. На вашем диване не поместится полбиблиотеки. Обычный ноутбук может предложить,

например, 2 гигабайта оперативной памяти. Кстати, как раз из-за этого компьютер начинает «тормозить», если у вас открыто слишком много программ и документов.

Получается, что могущество компьютерной памяти сильно ограничено. У памяти на диске или сервере практически бесконечный *объем,* зато ограничена *скорость* доступа. А у оперативной памяти *скорость* феноменальная, зато *объем* очень маленький.

Какие последствия все это имеет для обработки больших данных? Оказывается, это фундаментальная и серьезная проблема даже для самых незамысловатых операций.

Раз, два, три, четыре, пять...

Рассмотрим небольшой пример. Допустим, банк выпустил пятьдесят кредитных карт с номерами 01, 02... 50. Всего было проведено 30 трансакций по картам с номерами:

15	48	32	31	48	27	50	01	02	50
02	37	25	16	29	18	45	30	08	32
01	42	05	48	10	33	05	46	50	21

Спрашивается: сколько клиентов воспользовалось кредитными картами?

Попробуйте сами ответить на этот вопрос. Каковы ваши действия? Поначалу все просто: 15 — это раз, 48 — два, 32 — три, 31 — четыре. Дальше снова 48, но эта та же карточка, поэтому мы ее не считаем. 27 — пять. В первом ряду всего 8 разных номеров. А вот во втором ряду уже начинаются затруднения. Попадался нам номер 02 до этого или нет? А 29? Или 45? Наша память не в состоянии это удержать! В результате до конца третьего ряда можно добраться только одним способом — сравнивать каждый номер со всеми предыдущими и ставить пометку, если он раньше не встречался.

Ответ: 22 карты. Ниже мы выделили разные номера жирным шрифтом.

15	**48**	**32**	**31**	48	**27**	**50**	**01**	**02**	50
02	**37**	**25**	**16**	**29**	**18**	**45**	**30**	**08**	32
01	**42**	**05**	48	**10**	**33**	05	**46**	50	**21**

Теперь представьте, что эту задачу выполняет компьютер. Каким образом можно сравнить номер каждой карты со всеми предыдущими? Особенно если трансакций не тридцать, а три миллиона? И вот тут возникает проблема с памятью. Считывать все номера заново с жесткого диска — непозволительно долго. А оперативная память переполнится гораздо раньше, чем мы дойдем до середины.

Что же получается? При наличии полных данных и самых мощных компьютеров мы не в состоянии подсчитать, сколько человек воспользовалось кредитными картами?! В принципе да, не в состоянии. И конечно, кредитные карты — это всего лишь пример. На самом деле мы вообще не в состоянии ничего посчитать!

У вас на сайте есть счетчик уникальных посещений? Скорее всего, он считает приблизительно.

Это и есть принципиальный подход к решению задачи о подсчете. Если точный ответ нам физически недоступен, нужно найти как можно более точный *приблизительный* ответ, который при этом использует *минимальное* количество памяти.

Общее количество заходов на сайт, кстати, можно сосчитать и точно, потому что в этом случае не нужно запоминать каждое посещение, достаточно помнить, сколько их было. Число друзей на «Фейсбуке» тоже подсчитывается точно, потому что они не повторяются. Приблизительное решение понадобится для подсчета количества *разных* объектов, когда один и тот же объект может появиться *несколько* раз.

Оговоримся, что в примере с кредитными картами проблему можно решить и по-другому, с абсолютной точностью.

Например, если выстроить все номера в порядке возрастания, то сосчитать разные номера будет гораздо проще. Очевидный недостаток состоит в том, что обычно к нам поступают неупорядоченные данные и их сортировка представляет собой отдельную, непростую задачу, которой посвящена масса математической литературы.

Еще можно заранее создать такую структуру данных, где на каждого клиента было бы заведено электронное досье. Тогда совсем нетрудно пройтись по всем досье и посчитать, сколько из них содержит трансакции по кредитным картам. К каждому досье понадобится обратиться всего раз, и это много времени не займет. Но данный способ тоже работает не всегда. Например, как посчитать количество магазинов, в которых клиенты воспользовались кредитной картой? Наверняка у банка нет досье на каждый магазин.

Проблема подсчета очень актуальна, особенно в контексте больших данных. Сколько посетителей заходит на наш сайт из разных регионов России? Сколько школьников в этом году подали заявления в вузы? Сколько людей обсуждают в социальных сетях нашу партию? Сотрудники Google в своей статье [17] пишут, что в их систему хранения и обработки данных поступает свыше пяти миллионов подобных запросов в день! Регулярно встречаются запросы, предполагающие подсчет более миллиарда объектов. В этой статье также приводится цифра: в среднем около ста таких запросов в день. Из-за ограничений памяти получить точный ответ на подобный запрос абсолютно нереально.

Как найти хорошее приближение, практически ничего не запоминая? У задачи подсчета есть несколько решений. Сходу такое решение нельзя придумать, но понять основные идеи не так уж сложно.

Как решается задача подсчета

Мы воспользуемся блестящим блогом [25] и начнем с метода *K-Minimum Values* [*K*-минимальные величины]. Идея очень проста. Допустим, значения, которые надо посчитать, равномерно разбросаны в каком-то интервале. В нашем примере с номерами карточек от 01 до 50 и их непредсказуемым использованием это предположение вполне разумно. Теперь давайте не будем запоминать все увиденные значения, а запомним лишь несколько самых маленьких.

Возьмем снова пример с кредитными картами, где трансакций было 30, а разных карт — 22. Мы можем запомнить, скажем, всего пять самых маленьких значений. В данном случае это номера 01, 02, 05, 08 и 10. Пять значений в интервале от 1 до 10. Значит, сколько разных значений мы встретим в интервале от 1 до 50? Интервал в пять раз длиннее, значения разбросаны равномерно. Стало быть, всего значений будет

$$5 \text{ (значений)} \times 5 \text{ (интервалов)} = 25 \text{ (значений)}$$

примерно двадцать пять. Поскольку число 10 находится на самой границе интервала, делается коррекция. Для большей точности в данном случае пользуются формулой

$$\frac{(5-1)}{\left(\frac{10}{50}\right)} = 20 .$$

Это, конечно, не равняется точному ответу 22, но достаточно близко к нему. При этом нам пришлось запомнить только 5 значений, а не 22.

Хранить в оперативной памяти всего несколько самых маленьких значений уже вполне реально, но по-прежнему не идеально. Чем больше значений мы сохраняем, тем выше точность, и для действительно высокой точности значений нужно хранить довольно много.

Можно ли сделать лучше? Оказывается, можно. Настоящую революцию в мире счетчиков совершил французский математик Филипп Флажоле. Его результаты были опубликованы в 2007 году в статье [16], а сейчас широко применяются в системах обработки данных, в том числе *BigQuery Google-a* и *Redis* компании Amazon.

HyperLogLog-счетчики

Филипп Флажоле и соавторы предложили новый метод подсчета под названием *HyperLogLog*.

LogLog означает, что по сравнению с числом объектов нам нужно очень маленькое количество оперативной памяти. Под LogLog здесь понимают двойной логарифм по основанию 2, и это на самом деле очень маленькое число. Например, при миллиарде объектов двойной логарифм будет

$$\log_2\left(\log_2(1000\ 000\ 000)\right) \approx 4{,}9\,,$$

то есть порядка 5 битов памяти — всего пять нулей и единиц!

Приставка Hyper использована тоже не просто так. Идею подобного метода Флажоле предлагал и раньше, но предыдущие версии давали слишком грубые результаты.

Метод *HyperLogLog* сильно улучшил точность, что позволило применить его на практике.

Как и в методе *K-Minimum Values,* Флажоле исходил из предположения, что записи в базе данных можно представить в виде разбросанных случайным образом чисел. На практике это действительно так, потому что каждой записи, будь то число, имя, адрес или название, присваивается так называемое хеш-значение. Это последовательность из нулей и единиц *одинаковой небольшой* длины. Если две записи совпадают (например, одно и то же название повторяется два раза), то и их хеш-значения совпадают. Например,

хеш-значения длины 8 для разных веб-магазинов могут выглядеть примерно как в табл. 7.1. Мы выделили жирным шрифтом название, которое повторяется два раза:

Таблица 7.1. Фиктивный пример хеш-значений веб-магазинов

Номер трансакции	Название магазина	Хеш-значение
1	**H&M**	**01011001**
2	РЖД	00011101
3	Ozon.ru	10110110
4	**H&M**	**01011001**
5	Аэрофлот	10001011

На практике обычно применяют хеш-значения длины 32 или 64. Хеш-значения генерируются с помощью специальных программ, так называемых *хеш-функций,* весьма нетривиальным образом. Усмотреть связь между изначальной записью и ее хеш-значением фактически невозможно. Поэтому можно считать, что хеш-значения присваиваются случайным образом. И метод *K-Minimum Values*, и метод Флажоле пользуются не самими записями, а их хеш-значениями.

Напомним, что по методу *K-Minimum Values* нужно запомнить несколько самых маленьких хеш-значений. Флажоле пошел еще дальше, предложив запоминать только число нулей в начале хеш-значения.

Для примера опять возьмем хеш-значения длины 8. Допустим, нам попалось хеш-значение

00001011

Последовательности, у которых в начале ровно четыре нуля, встречаются не очень часто, в среднем одна на 32. Если мы наткнулись на такую последовательность, то можно предположить, что в среднем мы видели 32 разных хеш-значения, то есть число объектов — примерно 32. Что, если мы видели еще больше нулей, скажем пять?

00000101

Такие последовательности еще более редки, одна на 64, то есть объектов примерно 64! Разве не гениально?

Идея очень простая. Чем больше нулей, тем реже встречаются такие хеш-значения. Если какому-то объекту случайно досталось очень редкое хеш-значение, значит, объектов было много.

Запомнить при этом нужно только одно число — самое большое количество нулей, которое мы видели, например 4. Для этого достаточно минимального объема памяти. Если нам попалось хеш-значение, у которого нулей больше, например 5, мы выкидываем из памяти 4 и запоминаем 5*.

Очевидно, что этот метод очень приблизительный. Например, хеш-значение с четырьмя нулями нам могло встретиться и в самом начале. Столь грубые оценки для практики не годятся.

В *HyperLogLog* для улучшения точности добавлено много хитрых приемов. Хеш-значения разбиваются на регистры, оценка считается в каждом регистре отдельно, а потом усредняется специальным образом. Кроме этого, учитывается коррекция, когда объектов очень мало или, наоборот, очень много. Подробно об этом можно прочитать в блоге [25]. Там можно даже запустить программу и посмотреть, как метод работает.

На практике стараются достичь еще более высокой точности. Собственно, об этом статья сотрудников Google [17], о которой мы упоминали выше. Она так и называется «HyperLogLog на практике». Например, авторы уделили особое внимание коррекции при маленьком числе объектов. Грубо говоря, если клиент в состоянии посчитать объекты вручную, то и компьютер должен давать абсолютно правильный ответ.

---

* В приложении 1 к главе 7 объясняется, откуда возникает двойной логарифм.

Так грубая, почти нахальная оценка Флажоле привела к решению задачи подсчета, решению с оптимальным балансом между эффективностью и точностью.

Четыре виртуальных рукопожатия

Помимо того что задача подсчета важна сама по себе, она находит и другие, совершенно неожиданные применения.

Все мы знаем, что мир тесен. Знакомишься с человеком, и тут же находятся общие знакомые или хотя бы знакомые знакомых. Многих исследователей интересовал вопрос, насколько тесен виртуальный мир социальных сетей.

Вопросы подобного рода имеют длинную историю. В конце 1960-х годов социолог Стэнли Милграм провел свой знаменитый эксперимент. Он раздал случайным людям в штатах Небраска и Канзас письма, адресованные брокеру из Бостона. Географически и по роду занятий эти люди были достаточно далеки от адресата. По правилам эксперимента, участники могли переслать письма только своим знакомым, которые должны были передать их дальше, своим знакомым, и так далее, пока они в конце концов не достигнут адресата. Из 296 писем большинство затерялось в дороге, но 64 письма дошли-таки по назначению. При этом оказалось, что цепочка, связывающая совершенно незнакомых людей, на удивление короткая. В среднем отправителя и адресата разделяло всего 5 человек! На рис. 7.1 мы схематически изобразили, как письмо пересылалось всего шесть раз, через пять промежуточных человек.

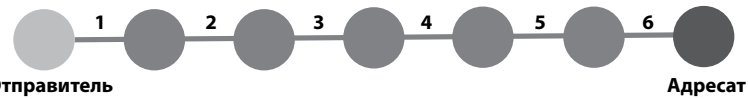

Рис. 7.1. Схематическое изображение эксперимента Стэнли Милграма. Участники могли переслать письмо только своим знакомым. Письмо от отправителя до адресата пересылалось 6 раз, через 5 разных человек

Так родилось понятие «шести рукопожатий». Если вы пожмете руку всем своим знакомым, ваши знакомые пожмут руку своим знакомым и так далее, то понадобится всего шесть рукопожатий, чтобы соединить вас с любым человеком на Земле!

Эксперимент Милграма не идеален, неслучайно столько писем затерялось в дороге. Но в то время просто не было другого способа выяснить, кто, с кем и через кого знаком. Зато сейчас таких данных сколько угодно.

Чтобы провести подобный эксперимент в «Фейсбуке», не нужно даже беспокоить участников. На сервере сети хранится полная информация, кто с кем дружит. Остается только вычислить количество «виртуальных рукопожатий», отделяющих одного пользователя от другого. Можем ли мы это сделать? Именно такой вопрос задали научные сотрудники «Фейсбука» профессору Миланского университета Себастьяно Винье.

И Себастьяно, и сотрудники социальной сети понимали, что это колоссальная задача. Неслучайно она не была решена раньше. У «Фейсбука» свыше 700 миллионов активных пользователей, то есть более

$$\frac{1}{2} \times 700\,000\,000 \times 700\,000\,000 = 245\,000\,000\,000\,000\,000$$

пар пользователей. И нужно вычислить длину цепочки для каждой пары! Но дело не только в этом. Ключевая проблема — опять же в количестве оперативной памяти, и возникает она потому, что между двумя пользователями не одна, а несколько цепочек разной длины. Для примера посмотрим на маленькую социальную сеть на рис. 7.2. А отделяет от В одно рукопожатие или два — через Г. Разных цепочек очень много, потому что друзья друзей зачастую наши друзья. В социальных сетях это известный, проверенный и доказанный феномен. При этом нас интересует *самая короткая* цепочка.

Компьютеру совсем нетрудно считать с диска всех «друзей друзей» пользователя А. Но только некоторых из них отделяет от А одно рукопожатие, а некоторых — два. Например, на рис. 7.2 и В, и Ж — это друзья друзей А. Но В друг А, а Ж — нет. Как определить, что Ж на расстоянии двух рукопожатий? Только одним способом — *запомнить* всех друзей А. Именно так работает самый распространенный метод под названием *поиск в ширину*.

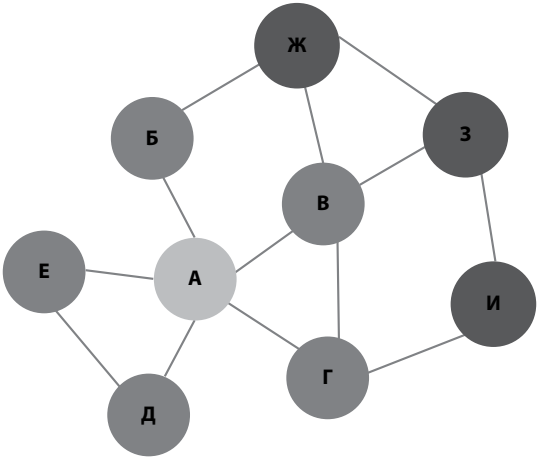

Рис. 7.2. Социальная мини-сеть. Кружками с буквами обозначены пользователи, линия между двумя пользователями означает, что они друзья. Мы обозначили пользователя А светло-серым цветом, а его друзей — серым. Темно-серым цветом обозначены пользователи, которых от А отделяют два рукопожатия

Если теперь добавить всех «друзей друзей друзей», а потом их друзей и так далее, то нужно запоминать всех, кого мы видели на расстоянии один, два, три... После пяти-шести шагов мы уже видели практически всех пользователей сети. Держать в оперативной памяти 700 миллионов имен, или разных чисел, — ну просто абсолютно нереально!

Сотрудники «Фейсбука» поверили в успех, потому что Себастьяно Винья и его коллеги придумали абсолютно иной способ подсчета числа рукопожатий. Они заметили, что самая главная проблема — необходимость запоминать увиденных пользователей — совершенно аналогична задаче о подсчете. А последнюю можно решить методом *HyperLogLog*.

Для интересующихся читателей ниже во врезках мы объясняем основные идеи с помощью примера на рис. 7.2. Этот текст не требует математической подготовки, но, если вы не хотите вдаваться в подробности, можете его пропустить.

Применение *HyperLogLog*-счетчика для подсчета числа рукопожатий

Хорошая новость: можно очень легко вычислить всех друзей любого пользователя.

Присваиваем хеш-значение каждому пользователю. Запускаем счетчик *HyperLogLog* и просматриваем пользователей начиная с А. На каждом шаге *HyperLogLog* будет сообщать, сколько всего разных пользователей мы видели. Оказывается, это все, что нам нужно.

Начнем с пользователя А. Это один пользователь, на расстоянии ноль рукопожатий от самого себя. Теперь посмотрим на его друзей:

Б В Г Д Е,

HyperLogLog сообщит нам, что всего с момента запуска мы видели 6 пользователей. Минус А, получаем 5 пользователей на расстоянии одного рукопожатия. Теперь к А и его друзьям добавляем всех тех, кто дружит с друзьями А:

друзья Б: А, Ж,

друзья В: А, Г, Ж, З

друзья Г: А, В, И

друзья Д: А, Е

друзья Е: А, Д.

После этого *HyperLogLog* сообщит, что всего с момента запуска мы видели 9 разных пользователей. Вычитаем A и число его друзей, получается

$$9 - 1 - 5 = 3$$

пользователя на расстоянии двух рукопожатий от A. Заметьте, что нам не нужно запоминать самих друзей A, а только их количество!

В данном случае больше пользователей у нас нет. В реальности процесс продолжается. На третьем шаге мы добавляем всех «друзей друзей друзей». После этого *HyperLogLog* опять сообщает, сколько разных пользователей мы видели до сих пор. Вычитаем из этого числа количество пользователей на расстоянии ноль, один и два и получаем число пользователей, которых от A отделяют ровно три рукопожатия. И так далее.

Процесс будет завершен, когда мы увидим всех пользователей сети. Как вы помните, *HyperLogLog* держит в памяти лишь максимальное число нулей в хеш-значении. В дополнение к этому надо запомнить только число пользователей на расстоянии одного, двух, трех и так далее рукопожатий. Это тоже занимает очень скромный объем оперативной памяти.

HyperLogLog требует так мало памяти, что процесс можно запустить для всех пользователей одновременно и в результате получить общее число пар на расстоянии одного, двух, трех рукопожатий и так далее. Конечно, результаты приблизительные, но их точность очень высокая.

Оказалось, что двух пользователей «Фейсбука» в среднем разделяет не шесть, а всего 4,74 рукопожатия! Статья Себастьяно Виньи и соавторов [10] быстро обрела известность в научном мире и за его пределами. О ней писали BBC и New York Times. Результаты уже сейчас попали в популярную литературу и специальные учебники. Виртуальный мир оказался еще теснее, чем реальный — тот, в котором мы живем!

Филипп Флажоле

Филипп Флажоле умер в 2011 году, когда работа над внедрением *HyperLogLog* была в самом разгаре. Блог под названием «*HyperLogLog — краеугольный камень инфраструктуры больших данных*» [25] был написан в 2012 году. Статья [10] о 4,74 виртуальных рукопожатий в «Фейсбуке» вышла в том же 2012 году, а статья сотрудников Google [17] — в 2013-м. Система *Redis* внедрила *HyperLogLog* в 2014 году. В честь Филиппа Флажоле команды *HyperLogLog* начинаются с его инициалов PF.

Нам неизвестно, успел ли Флажоле узнать, что его результаты уже стояли на пороге широкого внедрения. Мы закончим последними фразами из блога [25], где автор подробно и с большим энтузиазмом объясняет *HyperLogLog* широкой технической публике:

... Мне очень жаль, что я не был знаком с ним лично. Я уверен, что его труды будут жить дальше, но наука и индустрия, безусловно, потеряли выдающийся интеллект. Продолжай считать, Филипп, мы на тебя рассчитываем!

Глава 8
Миллион аукционов в минуту

Первая страница поисковика

Что бы мы ни собирались предпринять — от ремонта квартиры до поиска работы и планирования отпуска, — последние 10–15 лет практически любое дело мы начинаем с белой странички поисковой системы. Популярность и полезность поисковиков трудно переоценить. Один Google обрабатывает более 100 миллиардов запросов в месяц!

Мы вводим запрос, и буквально в считаные секунды на экране появляется первая страница результатов поиска. Ее содержание — тема особенно интересная, потому что большинство из нас дальше первой страницы не заходят, это широко известная статистика. Наверное, кое-какие принципы заполнения первой страницы вы уже давно заметили. Например, если вы ищете что-то связанное с товарами и услугами, то на самом видном месте — вверху первой страницы — появляется реклама. Рынок онлайн-рекламы у поисковиков просто колоссальный. Реклама — это источник доходов «Яндекса» практически на сто процентов. Число рекламодателей Google уже давно превысило миллион, а доход Google от онлайн-рекламы в 2015 году оценивался в 44 с лишним миллиарда долларов.

Почему почти у всех крупных поисковиков онлайн-реклама выглядит именно так, как сегодня на нашем экране? Почему вы видите именно эти объявления и в этом порядке?

За каждым объявлением скрываются глубокие математические идеи и не одна, а целых три Нобелевские премии. В этой главе мы расскажем о математике онлайн-рекламы.

Стоимость за один клик

Поисковики в их современном виде появились сравнительно недавно — в конце 1990-х годов. Реклама — гораздо раньше: газеты, щиты, телевизионные ролики. Мы расскажем о развитии и особенностях онлайн-рекламы примерно так, как это сделал Сергей Измалков — профессор Российской экономической школы, признанный эксперт в этой области.

Все мы настолько привыкли к рекламе в поисковиках, что первый тезис Сергея застает нас врасплох: «*Изначально, когда поисковые системы только-только появились, выгода такой рекламы для рекламодателей была совершенно неочевидна!*»

Самый первый вопрос: кто увидит эту рекламу? С журналами, например, все понятно. Сноуборд надо рекламировать в спортивном журнале, а косметику — в женском. Но интересы пользователя за экраном компьютера могут быть абсолютно любыми. Показывать рекламу косметики человеку, который зашел искать байдарочные маршруты Кировской области, практически бесполезно.

Так возникла идея рекламы по *ключевой фразе*. Вместо «журнала» рекламодатель выбирает в поисковом запросе интересующие его слова. Скажем, если компания устанавливает пластиковые окна, то они могут выбрать запросы *пластиковые окна, окна, ремонт квартиры* и тому подобные. А на другие запросы, например *пластиковые тарелки,* их реклама не появится. Поисковик точно знает, что интересует пользователя в данный момент. На запрос *жираф* можно предложить туры в Африку, а на запрос *рецепт пиццы* — мастер-класс по ее приготовлению (а не рекламу пиццерий!).

Такой «контекстной» рекламой пользуются все поисковые системы и все крупные сайты, включая «Фейсбук» и «Ютуб». Это был первый шаг к успеху онлайн-рекламы.

Второй вопрос: за что, собственно, поисковая система будет брать деньги? Например, газеты берут за показ. Но у газеты известно число подписчиков, и объявление можно увидеть своими глазами. Поисковая система — совсем другое дело. Рекламодатели не готовы платить за показы, количество которых будет неизвестно и которые никак нельзя ни увидеть, ни проверить.

Пришлось задуматься, зачем в принципе нужна онлайн-реклама? Самый разумный и прагматичный ответ абсолютно очевиден: чтобы привести потенциальных покупателей на сайт. Именно за это готовы платить рекламодатели. И это прекрасно, потому что чисто технически зарегистрировать, перешел пользователь с поисковика на сайт рекламодателя или нет, очень легко!

Так возникла еще одна ключевая идея онлайн-рекламы: *стоимость за 1 клик.* Когда вы видите рекламу, «Яндекс» ничего на этом не зарабатывает. Счетчик поворачивается, только если вы кликнете на рекламный линк.

Стоимость за клик — очень красивое рыночное решение. Рекламодатель может легко подсчитать, сколько посещений сделано через поисковик и сколько из них закончилось покупкой. С другой стороны, поисковая система заинтересована в том, чтобы кликов было как можно больше. Для этого нужно показывать рекламу правильным пользователям и в правильный момент. А еще нужно не раздражать рекламой и непрерывно улучшать результаты «бесплатного» поиска, чтобы люди были довольны и продолжали пользоваться системой. В результате все в выигрыше: и рекламодатели, и поисковые системы, и мы — простые пользователи и покупатели.

Аукцион — специально для вас!

Остается еще один существенный вопрос: сколько, собственно, стоит один клик? Логично, что ключевая фраза *ипотека* должна стоить гораздо дороже, чем *детский велосипед б/у*. Но сколько именно и насколько дороже?

Современные поисковые системы назначают цены за клик через *аукцион*. Все рекламодатели заранее предлагают свою цену за клик по той или иной ключевой фразе, и в момент поискового запроса рекламные места уходят тем, кто предложил больше. Настоящая рыночная цена! Для полноты информации сообщим, что цена за клик, например в Google, обычно 1–2 доллара, но доходит и до 50 долларов и выше, особенно в финансовом секторе.

Рекламу нужно подстраивать под пользователя, поэтому аукционы проводятся в реальном времени, для *каждого* поискового запроса в отдельности. Учитываются не только ставки, но и качество объявления, качество продавца, интересы пользователя, исходя из предыдущих запросов и кликов, и сотни других факторов. Существенно влияет местонахождение пользователя. Например, если вы в «Яндексе» в настройках измените город с Москвы на Нью-Йорк, то рекламы московских компаний вы не увидите.

Каждый раз, когда вы вводите запрос типа *пластиковые окна,* поисковая система проводит по нему аукцион и представляет вам победителей! Это делают все поисковые системы. И поскольку один только Google обрабатывает примерно 40 тысяч запросов в секунду, то миллион аукционов в минуту — далеко не преувеличение.

Естественно, процесс полностью автоматизирован. Ставки делаются онлайн, победители и цены вычисляются автоматически по заранее заданной формуле. Какая формула сработает лучше всего?

И вот тут в дело вступает математика. Потому что поисковик устанавливает правила аукциона с целью заработать как можно больше, а рекламодатель приспосабливает свои ставки к этим правилам, чтобы заплатить как можно меньше. Можно ли сбалансировать интересы всех сторон? Можно ли придумать такие правила, чтобы игра велась честно и приводила к самому выгодному для всех результату?

Этими вопросами занимается очень интересная область современной математики под названием «Дизайн механизмов». В 2007 году ее основатели Леонид Гурвич, Эрик Маскин и Роджер Майерсон получили Нобелевскую премию по экономике. «Механизм» в данном случае — не машина, а механизм управления. Теория дизайна механизмов изучает, как создать оптимальные правила игры в условиях конкуренции: аукционы, голосование, распределение ресурсов. Онлайн-реклама — всего лишь одно из многочисленных приложений.

Аукцион второй цены

При слове «аукцион» мы обычно представляем комнату, полную хорошо одетых людей, и человека с молотком на трибуне. Участники поднимают ставки. Раз, два, три — продано! Это *открытый* аукцион, все слышат ставки друг друга.

В онлайн-рекламе используются так называемые *закрытые* аукционы, потому что рекламодатели не знают ставок друг друга. Ставки сделаны заранее, и в момент «розыгрыша» поисковая система сравнивает их и вычисляет победителей и цены. Это похоже на ситуацию, когда каждый участник записывает ставку на листе бумаги и отдает его организаторам в запечатанном конверте. В момент розыгрыша конверты вскрываются, и товар уходит покупателю, чья ставка оказалась самой высокой.

Представьте себе наиболее простую ситуацию, так называемый аукцион *первой цены.* Конверты вскрыты, и товар уходит по самой высокой ставке тому, кто ее предложил. Допустим, Анна и Борис участвуют в таком аукционе. Анна готова заплатить за товар 500, а Борис 300. Оба честно написали ставки 500 и 300. Товар ушел к Анне за 500. Все справедливо, но все ли довольны? На самом деле нет. Если бы Анна написала 400 или даже 301, то получила бы товар гораздо дешевле! Получается, что правдивую ставку делать невыгодно. Самое лучшее для Анны — это разузнать или угадать ставку Бориса и поставить чуть-чуть больше.

Теперь представим, что правила изменились. Товар получает тот, кто сделал самую большую ставку, но по цене второй по величине ставки. Это называется *аукционом второй цены.* Анна и Борис честно пишут 500 и 300. Анна получает товар за 300.

Если задуматься, традиционный открытый аукцион с повышающимися ставками — это тоже аукцион второй цены. Товар уходит к победителю, как только его ставка становится чуть выше, чем предельная ставка второго претендента.

Чудо аукциона второй цены заключается в том, что теперь делать правдивые ставки становится выгодно и безопасно. Анна может спокойно писать 500 и не волноваться, потому что больше 300 с нее не возьмут. А 300 и Борис готов заплатить, поэтому меньше 300 никак не получится.

Создать правила, при которых выгодно делать правдивые ставки, — один из основных принципов теории дизайна механизмов, он называется *совместимость по стимулам.* Аукцион второй цены — классический пример этого принципа. Математический анализ аукциона второй цены и доказательство совместимости по стимулам впервые предложил американский экономист Уильям Викри. Его статья, которая заложила основы теории аукционов, вышла в 1961 году [27].

В 1996-м за эти исследования Викри получил Нобелевскую премию по экономике.

Именно аукционами второй цены, в разных вариантах, пользуются все поисковые системы. По словам Сергея Измалкова, это была «четвертая революция» в онлайн-рекламе*. Ниже на рис. 8.1 перечислены все четыре ключевые идеи, приведшие к современной онлайн-рекламе, какой мы ее видим каждый день.

1. Контекстная реклама по ключевой фразе поискового запроса
2. Оплата за клик
3. Стоимость за клик определяется через аукцион
4. Аукцион второй цены

Рис. 8.1. Четыре ключевые идеи современной рекламы в поисковых системах

Результат Викри

Главный вклад Викри — это формальный анализ аукционов и доказательство того, что различные форматы аукционов приносят одинаковый доход продавцу. Один из удивительных результатов этого анализа заключается в том, что аукцион второй цены удовлетворяет совместимости по стимулам, то есть каждому участнику выгодно делать правдивые ставки. Понять это совсем не сложно, почти элементарно.

Вернемся к примеру, в котором Анна, Борис и еще несколько человек участвуют в аукционе второй цены. Каждый пишет свою ставку и кладет в конверт. Допустим, для Анны ценность товара 500, и ставок других участников она не знает.

---

* Помимо поисковых систем аукционами второй цены пользуется корпорация электронной торговли eBay.

Что произойдет, если Анна поставит 600? На рис. 8.2 мы представили три возможных сценария. Ставки участников обозначены жирными точками. В рамке вторая по величине ставка, это окончательная стоимость товара в аукционе второй цены. Если все остальные участники поставили меньше 500, как на рис. 8.2 вверху, то ничего не изменилось, Анна по-прежнему получает товар за вторую цену (в данном примере 300). Повышать ставку смысла не было. Если кто-то поставил больше 600, как на рис. 8.2 в середине, Анна не получает товар в любом случае. Победитель платит больше, но для самой Анны ничего не меняется. Опять же повышать ставку смысла не было (разве что из вредности, но модель этого не учитывает). Теперь представьте, что кто-то поставил 550, как на рис. 8.2 внизу. Тогда Анна получает товар за 550, а это для нее слишком дорого. В контексте онлайн-рекламы она может даже потерять на рекламе, вместо того чтобы заработать. Вывод: ставить выше 500 невыгодно.

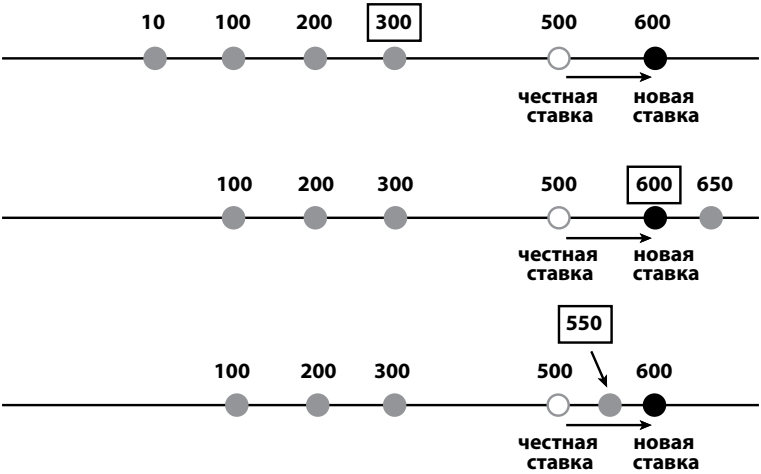

Рис. 8.2. Три сценария, в которых Анна вместо правдивой ставки (500) делает более высокую ставку (600)

Но, может, выгодно сделать более низкую ставку? Оказывается, нет! Допустим, Анна решила попробовать поставить поменьше, скажем 400. На рис. 8.3 изображены те же три сценария. Если все остальные участники поставили меньше 400, как на рис. 8.3 вверху, то ничего не изменилось, Анна по-прежнему получает товар за вторую цену (в данном случае снова 300). Если кто-то поставил больше 500, как на рис. 8.3 в середине, то Анна не получит товар в любом случае. Но если кто-то поставил 450, как на рис. 8.3 внизу, то Анна товар опять же не получает, хотя цена ее устраивала и она была готова платить больше, чем победитель. Возможность хорошей онлайн-рекламы упущена. Ставить ниже 500 тоже невыгодно!

Получается, что при аукционе второй цены ни одному из участников нет никакого смысла делать ставку, которая отличается от их честной оценки товара*.

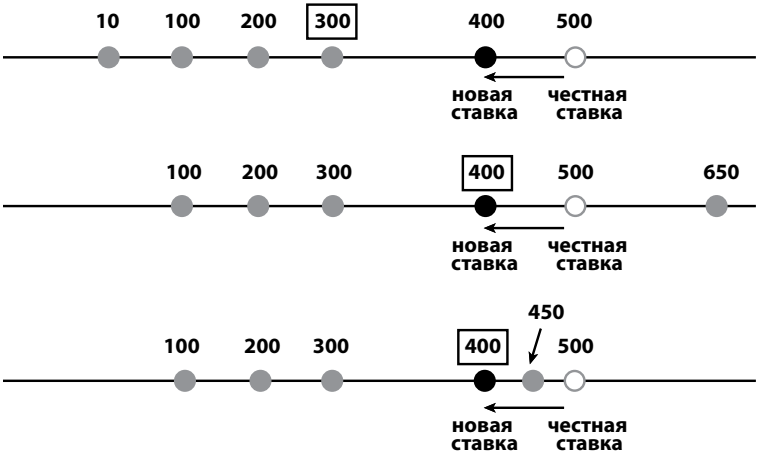

Рис. 8.3. Три сценария, в которых Анна вместо правдивой ставки (500) делает более низкую ставку (400)

* В приложении к главе 8 приведено более формальное доказательство совместимости по стимулам в аукционе второй цены.

Это очень простой и глубокий математический результат, в котором сходятся сразу две фундаментальные концепции. Одну из них мы уже обсудили, это совместимость по стимулам, при которой честные ставки самые выгодные. Вторая — это так называемое *равновесие по Нэшу*. Речь идет о том самом Джоне Нэше из фильма «Игры разума».

Равновесие по Нэшу представляет собой ситуацию, когда ни одному из участников не выгодно в одиночку отклоняться от выбранной стратегии. Данная концепция лежит в основе *теории игр*, за разработку которой Джон Нэш получил Нобелевскую премию по экономике 1994 года. «Игрой» можно назвать любую ситуацию, когда исход зависит от действий всех участников и у каждого из них свои интересы. Очевидно, что аукцион — это тоже игра. Участники выбирают ставки с целью получить товар по самой выгодной цене*.

Как распределить несколько рекламных мест

Аукцион Викри сильно упрощает реальность. Поисковые системы разыгрывают не одно, а несколько рекламных мест разной ценности. Хорошо известно, что количество кликов очень сильно зависит от позиции рекламы на странице. Например, по статистике Google, на мобильном телефоне почти 28 % пользователей кликает на самую верхнюю рекламу, а на вторую сверху — чуть больше 9 %. Чем выше вероятность кликов, тем дороже место.

Три верхние строчки — самые ценные. Часто нижние позиции на первой странице тоже заняты рекламой, это

* Для читателей с математической подготовкой, которых заинтересовала теория аукционов и три Нобелевские премии, рекомендуем видеолекцию Алексея Савватеева [5], где он подробно и очень доходчиво объясняет математическую сторону вопроса.

следующие по ценности места. Следующий ярус — на экране справа. А в особых случаях реклама попадается и на второй странице — «на пробу» для тех, кто ищет нестандартные результаты.

Теория Викри распространяется на случай, когда разыгрываются сразу несколько товаров. Это хорошо изученная модель аукциона, так называемый механизм Викри — Кларка — Гровса, который часто называют VCG, по первым буквам имен его создателей. Схема вычисления цен и математические свойства VCG довольно сложны и выходят за рамки нашей книги. Мы назовем только две основные особенности.

Во-первых, в аукционе VCG сохраняется свойство совместимости по стимулам: выгоднее всего делать честные ставки. Во-вторых, главная идея этого механизма та же, что и у аукциона второй цены. Возьмем снова наш простой пример. У нас один товар, Анна ставит 500, Борис 300, а остальные — еще меньше. В аукционе второй цены Анна получает товар, но платит 300, то есть максимальную ставку *других* участников. Примерно то же самое происходит в VCG при розыгрыше нескольких товаров. Товары уходят к тем, кто сделал на них максимальные ставки, а цены определяются не ставкой победителя, а ставками других участников.

Поисковые системы также используют *обобщенный аукцион второй цены,* изобретение Google. Система проста: каждый платит не свою цену, а следующую по величине. Например, рекламодатели А, Б, В и Г поставили за клик 10, 7, 3 и 1, как в табл. 8.1. Разыгрываются три места. Тогда А получает первое место за 7, Б получает второе место по цене 3, В — третье место по цене 1, а Г рекламного места не получает.

Таблица 8.1. Четыре рекламодателя и их ставки и цены за клик в обобщенном аукционе второй цены

Рекламодатель	А	Б	В	Г
Ставка за клик	10	7	3	1
Место	1	2	3	–
Цена	7	3	1	–

Google пришел к этой системе исходя из идеи второй цены и многочисленных экспериментов. Исчерпывающего математического анализа такого аукциона пока нет, но кое-какие основные результаты известны. Например, делать честные ставки выгодно не всегда, просто потому что более низкие рекламные позиции могут оказаться гораздо выгоднее. Это одна из причин, по которой «Яндекс» с августа 2015 года перешел на аукционы типа VCG, где честные ставки ведут к оптимальной позиции по оптимальной цене для всех участников.

Для заинтересованного читателя ниже во врезке мы приводим пример, в котором в обобщенном аукционе второй цены одному из участников выгодно занизить ставку. Этот пример не требует никакой математической подготовки, но если вы не хотите вникать в детали, можете его пропустить.

Обобщенный аукцион второй цены, в котором невыгодно делать честную ставку

Рассмотрим снова пример в табл. 8.1. Каждый клик для А представляет ценность 10. Допустим, вероятности клика на первой и второй позиции примерно как на десктопе: около 20% на первой позиции и около 11% — на второй. В этом случае ценность первой позиции для А в среднем 20% от 10, то есть 2, а ценность второй позиции в среднем 11% от 10, то есть 1,1.

При правдивой ставке А получает первую позицию по цене 7. Вспомним, что А платит только в случае клика. Это в среднем 20% случаев. Значит, средняя прибыль А равна

[20 % от 10] – [20 % от 7] = 2 – 1,4 = 0,6.

Но если бы А поставил 6, то он получил бы вторую позицию по цене 3, то есть прибыль была бы равна

[11 % от 10] – [11 % от 3] = 1,1 – 0,33 = 0,77.

Это больше, чем 0,6. Получается, что для А выгоднее сделать заниженную ставку 6, а не правдивую ставку 10.

Подготовленному читателю мы рекомендуем книгу Иона Клейнберга и Дэвида Исли [14]. В частности, в главе 15 этой книги подробно анализируются свойства обобщенного аукциона второй цены. Книга написана для широкой технической публики и есть в интернете в открытом доступе.

На самом деле картина в бизнесе поисковиков еще сложнее. При оплате за клик нет смысла отдавать козырные рекламные места под плохие объявления даже по самым высоким ценам. Качественная реклама приносит больше кликов. Допустим, рекламодатель А заплатит 7, а Б всего 3, но объявление Б генерирует в 4 раза больше кликов. Объявление А получит один клик и принесет 7, а Б на том же самом месте получит 4 клика и принесет 12. Поэтому при розыгрыше рекламных мест ставка Б считается выше, чем ставка А.

Самым простым вариантом такого аукциона пользовался Yahoo! в начале 2000-х годов. Качество сайта они оценивали как вероятность клика. Тогда ставка рекламодателя определялась как средний доход, который он предлагает:

[ставка в аукционе *Yahoo*] =
= [денежная ставка рекламодателя] × [вероятность клика].

Как именно определяется качество в современных системах и как оно влияет на подсчет ставок — строгая коммерческая тайна. Это зависит от качества сайта, отзывов покупателей, интересов данного пользователя и многого другого.

В итоге качество для поисковой системы — это потенциальное число кликов, которое принесет объявление. Но крайне важен и тот факт, что пользователям нравится видеть качественные объявления. Это поддерживает репутацию поисковика. А при нынешней конкуренции она бесценна!

По ту и другую сторону онлайн-рекламы

По словам Сергея Измалкова, аукционы второй цены в реальной жизни встречаются редко. Очень трудно не дрогнуть и отдать товар за 300, когда знаешь, что можно было взять 500! Серьезная логическая ошибка заключается в том, что взять 500 не получилось бы в любом случае. Если человек знает, что с него возьмут по максимуму, он просто не назовет свою правдивую цену. Люди приспосабливаются к правилам. На этом фундаментальном наблюдении основана вся теория дизайна механизмов.

В свое время поисковые системы ощутили эти закономерности на собственном опыте. Поначалу использовались аукционы первой цены. С победителя брали предложенную им же ставку. И это немедленно привело к ужасающей нестабильности. Аукционов разыгрывается множество, поэтому рекламодатели методом проб и ошибок меняли ставки по многу раз в день, чтобы получить лучший результат по минимальной цене. Из-за смены ставок возник жуткий трафик, который захватил ресурсы, необходимые для собственно поиска! И никакой особой выгоды все это не принесло.

Теоретический результат четко говорит о том, что при аукционе первой цены заработать больше не получится. Многие компании снова и снова попадают в эту ловушку.

Но поисковые системы при нынешнем масштабе онлайн-рекламы не могут себе этого позволить. Они стремятся к схемам, которые обеспечивают стабильность, поощряют честные ставки и поддерживают доверие.

Онлайн-реклама постоянно улучшается, но вопросов остается много. У рекламы и поиска разные цели. Реклама приносит доход, но может раздражать. Как оценить подобные убытки от рекламы? Как предсказать ее качество? Имеет ли смысл учитывать частную информацию для более целевой рекламы, с риском испугать пользователя, что о нем столько известно? Все это экономические задачи, не решаемые без математических моделей, потому что мы имеем дело с массовыми, сложными и очень взаимосвязанными процессами.

У рекламодателей вопросов не меньше. Средний интернет-магазин предлагает около десяти тысяч товаров. На какие ключевые фразы ставить и сколько? Как предсказать реальные доходы от кликов? Как оценить качество собственного объявления и объявления конкурента? В последнее время в мире появились тысячи маркетинговых компаний, специализирующихся на онлайн-рекламе.

Бóльшая часть онлайн-маркетинга пока основывается на опыте и здравом смысле. Например, если у вас маленький магазин спортивной обуви, бесполезно ставить на слово «*Найк*», лучше поставить на «*синие женские кроссовки "Найк"*». Но некоторые компании уже разрабатывают и используют компьютерные программы для автоматического подсчета оптимальных ставок — например, исходя из предыдущих ставок и продаж. При растущем размахе электронной торговли у нас есть все основания полагать, что будущее — именно за автоматическими методами и математическими моделями. По ту и другую сторону онлайн-рекламы работы для математиков — непочатый край.

Заключение: ч. т. д.

В Европе компании часто обращаются к математикам по самым разным вопросам: оптимизация, планирование, прогноз, анализ данных. Очень приятно видеть свои результаты внедренными на практике. И все-таки для нас, математиков, главная мотивация не в этом. Мы любим теорию. Мы любим найти новую интересную теорему, долго и мучительно к ней подступать и отступать, пока вдруг не возникнет полная ясность. Невозможно передать всю радость и удовлетворение от безупречной точности математических выкладок и от короткой записи «ч. т. д.» — *что и требовалось доказать.* Именно ради этого ощущения мы и работаем — больше, чем ради чего-либо другого. Как сказал один наш молодой коллега: «Мы все подсажены на “эврику”».

Противоречит ли это работе над приложениями? Конечно нет! Наоборот, в нашей книге много примеров, показывающих, что приложения стали источником новых математических теорий. Так, в главе 2 мы рассказывали о линейном программировании и его применении для составления расписаний. Основатель линейного программирования, замечательный советский математик Леонид Витальевич Канторович заинтересовался этими задачами именно благодаря их ценности для практики. Другой пример — теория массового обслуживания, которая возникла из практической задачи анализа телефонной станции и нашла применение во многих системах, где есть заявки, очереди и обслуживающий

прибор. Будь то супермаркет, поликлиника, кол-центр или веб-сервер, посылающий вам нужную веб-страничку через интернет. О задаче балансировки нагрузки на веб-сервере мы говорили в главе 5.

Конечно, далеко не вся наука основана на приложениях. Фундаментальная наука зачастую движима исключительно любопытством ученых. Согласно источникам [26], таким ученым был Нильс Бор. Одного упоминания этого имени достаточно, чтобы понять, как важно доверять ученым в выборе собственных интересов. В главе 6 мы рассказали о том, как любимая игра ума математиков — теория чисел — сегодня стала совершенно необходимой для шифрования конфиденциальных данных, которые мы ежедневно передаем по каналам интернета.

Математика полна нерешенных, чисто теоретических задач. Например, в комбинаторике есть задача нахождения так называемого *хроматического числа.* Допустим, нам нужно раскрасить плоскость так, чтобы любые две точки, находящиеся на расстоянии ровно 1 метра друг от друга, были разного цвета. Вопрос: какое минимальное количество красок понадобится? Точный ответ неизвестен. Мы знаем только то, что это число между 4 и 7. Если нам нужно раскрасить трехмерное пространство, то минимальное число красок лежит между 6 и 15, а для четырехмерного пространства — между 9 и 54. В принципе непонятно, зачем красить пространство, тем более четырехмерное! Но задачи нахождения хроматических чисел привели к мощному развитию комбинаторики, в том числе и прикладной. Хроматические числа используются, например, для расстановки вышек мобильной связи.

В нашей книге мы рассказали лишь об очень маленькой доле того огромного влияния, которое оказывает математика на жизнь каждого из нас. Мы выбрали всего несколько тем,

имеющих отношение к компьютерным технологиям и нашим собственным исследованиям, и надеемся, что даже с помощью этой небольшой выборки смогли убедить вас, что современные технологии невозможны без математики, такой красивой, такой точной и такой невероятно полезной науки. Что и требовалось доказать.

Приложения для подготовленного читателя

Приложения к главе 2

1. Существует оптимальное решение, соответствующее одному из углов многогранника

Отметим, что в выражении стоимости 1020 − 2 × АЮ − 5 × × БЮ в нашем примере оптимальные значения АЮ и БЮ не зависят от слагаемого 1020. Решение будет то же, если мы будем минимизировать –2 × АЮ − 5 × БЮ или максимизировать 2 × АЮ + 5 × БЮ.

Рассмотрим задачу линейного программирования с двумя переменными в общем виде.

Выбрать x_1 и x_2 так, чтобы максимизировать:

$c_1x_1 + c_2x_2$, (целевая функция)

при ограничениях:

$a_{11}x_1 + a_{12}x_2 \le b_1$, (ограничение 1)

$a_{21}x_1 + a_{22}x_2 \le b_2$, (ограничение 2)

...

$a_{m1}x_1 + a_{m2}x_2 \le b_m$, (ограничение m)

$c_1, c_2, b_i, a_{ij} \in \mathbb{R}$, $i = 1,2,...,m$, $j = 1,2$.

Заметьте, что, во-первых, задача максимизации эквивалентна задаче минимизации с коэффициентами $-c_1$ и $-c_2$. Во-вторых, любое неравенство со знаком ≤ можно превратить в эквивалентное неравенство со знаком ≥, умножив обе части неравенства на –1. Поэтому задача выше, для двух переменных и m ограничений, сформулирована действительно в общем виде. Все значения коэффициентов *a, b, c* — произвольные действительные числа, которые могут быть как положительными, так и отрицательными.

Каждое ограничение задает полуплоскость значений, на которой оно выполняется. Если пересечение всех m полуплоскостей пусто, то допустимого решения просто не существует. Поэтому допустим, что m полуплоскостей содержат общую ограниченную область S допустимых значений. (Мы не будем рассматривать случай, когда область не ограничена.) Очевидно, что S — это многоугольник, поскольку область S ограничена прямыми.

Утверждение. *Максимальное значение целевой функции достигается в одном из углов S.*

Доказательство. Обозначим оптимальное решение через $\left(x_1^*, x_2^*\right)$. Заметьте, что $\left(x_1^*, x_2^*\right)$ не может быть внутренней точкой *S,* потому что в этом случае оба значения переменных можно либо увеличить, либо уменьшить, таким образом увеличивая значение целевой функции. Например, в нашей задаче в главе 2 решение (58,8) является внутренней точкой, поэтому не может быть оптимальным.

Значит, $\left(x_1^*, x_2^*\right)$ лежит на одной из сторон многоугольника *S.* На каждой из сторон одно из ограничений превращается в равенство. Рассмотрим сторону, которая соответствует первому ограничению: $a_{11}x_1 + a_{12}x_2 = b_1$. Что происходит, если мы начнем двигаться вдоль этой стороны?

Не уменьшая общности, допустим, $a_{12} \neq 0$. Для начала перепишем равенство в более привычном виде как уравнение прямой:

$$x_2 = -\left(\frac{a_{11}}{a_{12}}\right)x_1 + \frac{b_1}{a_{12}}. \qquad (П.1)$$

Допустим, мы начали в точке (x_1, x_2). Теперь допустим, что мы немного изменили x_1 и получили новую координату $x_1 + \delta$, где $\delta > 0$ достаточно мало, чтобы все остальные ограничения, кроме первого, по-прежнему строго выполнялись. Тогда значение x_2 изменится на величину

$$-\left(\frac{a_{11}}{a_{12}}\right)(x_1 + \delta) + \frac{b_1}{a_{12}} - \left(-\left(\frac{a_{11}}{a_{12}}\right)x_1 + \frac{b_1}{a_{12}}\right) = -\left(\frac{a_{11}}{a_{12}}\right)\delta.$$

При этом нетрудно проверить, что целевая функция изменится на величину

$$c_1\delta + c_2\left(-\left(\frac{a_{11}}{a_{12}}\right)\delta\right) = \left(c_1 - \frac{c_2 a_{11}}{a_{12}}\right)\delta.$$

Заметьте, что это число не зависит от (x_1, x_2). Значит, в какой бы точке прямой (П.1) мы не начали движение, в результате перемещения по этой прямой, изменение значения целевой функции зависит только от коэффициента

$$\left(c_1 - \frac{c_2 a_{11}}{a_{12}}\right).$$

Если он отрицательный, то, увеличивая x_1 и двигаясь по прямой, мы можем только уменьшить целевую функцию. Аналогично если коэффициент положительный, то, двигаясь по прямой в сторону увеличения x_1, мы можем целевую функцию только увеличить. Наконец, если коэффициент равен нулю, значение целевой функции на всей прямой постоянно.

Стало быть, из любой точки на данной стороне S мы можем двигаться либо в сторону уменьшения, либо в сторону увеличения x_1 так, чтобы значение целевой функции не уменьшалось. Таким образом мы можем менять значение x_1, пока какое-то другое ограничение не превратится в равенство.

В этом случае мы столкнулись с углом многоугольника S, в котором достигается максимальное значение целевой функции на всей рассмотренной нами стороне. Поскольку сторону мы выбрали произвольно, делаем вывод, что максимальное значение целевой функции достигается в одном из углов S и мы можем выбрать этот угол в качестве $\left(x_1^*, x_2^*\right)$.

Очевидно, что это доказательство легко обобщить на любое количество n переменных.

2. Пример задачи целочисленного программирования

Допустим, нам нужно отправить грузовики с товаром к двум разным клиентам. Всего у нас в разных точках четыре грузовика. Обозначим через c_{ij} цену отправки грузовика $i = 1,2,3,4$ к клиенту $j = 1,2$. На любую доставку требуется полдня. Доставку можно осуществить либо утром (первая половина дня), либо днем (вторая половина дня). Нужно решить, к какому клиенту какой грузовик поедет и в какой момент времени.

Введем переменные x_{ijt}, $i = 1,2,3,4$; $j = 1,2$; $t = 1,2$. Эти переменные могут принимать значение 0 или 1. Например, если грузовик 3 едет к клиенту 2 в первой половине дня, то $x_{321} = 1$. Если этого не происходит (то есть грузовик 3 в первой половине дня никуда не едет или едет к другому клиенту), то $x_{321} = 0$.

В нашей небольшой задаче всего $4 \times 2 \times 2 = 16$ переменных, то есть ее можно решить и вручную.

Целевая функция — это цена доставки, и вычисляется она очень просто:

$$\text{цена доставки} = \sum_{i=1}^{4}\sum_{j=1}^{2}\sum_{t=1}^{2} c_{ij}x_{ijt}.$$

Например, если грузовик 3 едет к клиенту 2 в первой половине дня, то $x_{321} = 1$ и мы прибавим к общей стоимости c_{32}.

А если грузовик 3 к клиенту 2 не поедет, тогда $x_{321} = x_{322} = 0$ и c_{32} не войдет в общую сумму.

Самое интересное — это ограничения. Например, грузовик i не может поехать к двум клиентам в одно и то же время. Это можно записать в виде ограничения:

$$x_{i1t} + x_{i2t} \le 1,\ i = 1,2,3,4;\ t = 1,2.$$

Тогда для любого i и t только одно (или ни одно) из значений x_{i1t} или x_{i2t} может равняться единице.

Еще одно универсальное ограничение: к клиенту j нужно послать только один грузовик, то есть

$$\sum_{i=1}^{4}\sum_{t=1}^{2} x_{ijt} = 1,\ j = 1,2.$$

Ограничения могут учитывать особенности каждого грузовика, клиента и другие факторы. Например, мы не хотим, чтобы грузовик 3 работал утром (скажем, у этого грузовика запланирован техосмотр). Тогда мы просто включим ограничение:

$$x_{311} + x_{321} = 0.$$

Теперь допустим, что это условие желательное, но необязательное. Тогда к целевой функции можно добавить дополнительное слагаемое, которое будет означать штраф за невыполнение условия:

$$c_{\text{штраф}}\left(x_{311} + x_{321}\right).$$

Заметьте, что это слагаемое действительно добавится, только если грузовик 3 работал в утреннюю смену. Естественно, оптимальное решение будет зависеть от коэффициента $c_{\text{штраф}}$. Если он больше любого c_{ij} в целевой функции, то оптимальный вариант — не задействовать грузовик 3 с утра. А если коэффициент $c_{\text{штраф}}$ маленький, то, возможно, грузовик 3 все равно задействуют, если это обеспечит более низкую цену доставки.

В виде линейных ограничений можно записать самые разные условия. Например, мы хотим, чтобы грузовик 3 либо работал, либо не работал обе половины дня. Тогда мы вводим ограничение

$$x_{311} + x_{321} = x_{312} + x_{322}. \qquad (П.2)$$

Это условие можно несколько усложнить. Например, если грузовик 3 в первой половине дня поехал к клиенту 1, то мы хотим, чтобы он работал и во второй половине дня. Как это записать в виде линейного неравенства? Часто используется такой прием. Вводим достаточно большое значение M и записываем:

$$\left(x_{311} + x_{321}\right) - \left(x_{312} + x_{322}\right) \le M\left(1 - x_{311}\right).$$

Если $x_{311} = 1$, то значение справа при любом M равно нулю. Тогда неравенство выполняется (и на самом деле является равенством), только если $x_{312} + x_{322} = 1$ (вспомните, что $x_{311} + x_{321} = 1$). Но если $x_{311} = 0$, то M можно выбрать достаточно большим, чтобы ограничение не играло никакой роли. В данном случае, кстати, достаточно, чтобы $M = 1$. Для увеличения скорости решения M стараются выбирать «экономно» — не больше, чем нужно.

Есть еще множество интересных приемов записи обязательных и желательных условий в виде линейных выражений, но их более подробное описание выходит за рамки нашей книги.

3. Идея метода ветвей и границ

Допустим, нам нужно послать землекопов на объекты и мы хотим минимизировать стоимость работ. Для начала мы берем совершенно произвольное расписание и получаем стоимость работ, скажем 50 000 рублей. Это наш максимум, и мы постараемся его уменьшить.

Теперь запускаем симплекс-метод и получаем дробное решение. Например, на объект А нужно отправить 2 и $^2/_3$ землекопа. Допустим, общая стоимость работ при этом составит 40 000 рублей. Это пока не дает нам плана работ, потому что решение не в целых числах. Зато мы знаем, что это решение оптимальное, то есть при любом другом (в том числе целочисленном) решении стоимость получится никак не меньше 40 000 рублей. Значит, наша стоимость в результате будет между 40 000 и 50 000 рублей.

Дальше начинаем «разветвлять» решение. У нас есть два варианта: $A \leq 2$ и $A \geq 3$. Для каждого из них мы снова решаем задачу линейного программирования. Допустим, стоимость получилась 43 000 рублей при $A \geq 3$ и 51 000 при $A \leq 2$. Отсекаем вариант $A \leq 2$, поскольку у нас уже есть более выгодное решение. В результате делаем вывод, что $A \geq 3$, а минимальная стоимость теперь 43 000 рублей. Если при этом все переменные получились целочисленные, то мы нашли решение. А если у нас еще остались дробные переменные, то каждую из них разветвляем снова. И так до тех пор, пока не найдем решения в целых числах.

Приложения к главе 3

1. Число последовательностей из нулей и единиц заданной длины

Для начала рассмотрим последовательности длины 5. Сколькими способами мы можем выбрать первый элемент последовательности? Очевидно, что вариантов 2: ноль или единица. Теперь давайте посмотрим на второй элемент. Для него у нас тоже есть два варианта, причем при любом выборе первого элемента последовательности. Значит, число способов выставить друг за другом первые два элемента равно четырем.

Точно так же для каждого из этих четырех вариантов есть два способа выбрать третий элемент последовательности и так далее. В итоге для кодового слова длины 5 получаем

$$2 \times 2 \times 2 \times 2 \times 2 = 2^5 = 32.$$

Аналогично число разных последовательностей длины 4 равно $2^4 = 16$, а число разных последовательностей длины 8 равно $2^8 = 256$. Для любой заданной длины n получаем 2^n разных последовательностей из нулей и единиц.

2. Граница Хэмминга

Допустим, мы пользуемся словами длиной n и наш код состоит из N таких слов.

Если код исправляет d ошибок, то шары Хэмминга с центрами в кодовых словах и радиусами d попарно не пересекаются. Объем шара (то есть количество слов в нем) нетрудно вычислить. Сколько слов отстают от центра шара на заданное расстояние k? Разумеется, столько, сколькими способами можно выбрать те k позиций из n возможных, в которых произойдут помехи. Это число способов называется *числом сочетаний из n по k* и обозначается C_n^k. Для того чтобы его записать, нам понадобятся произведения вида

$$k \times (k-1) \times \ldots \times 2 \times 1.$$

Такое произведение принято обозначать записью

$$k!$$

и она читается как k факториал. Легко увидеть, что, конечно, $1! = 1$, и принято считать, что $0! = 1$. Заметим, что факториал уже встречался нам в главе 2 в разделе «Проклятие размерности». Там мы показали, что факториал растет очень быстро. Например, 25! — это колоссальное число.

Число сочетаний вычисляется по формуле

$$C_n^k = \frac{n(n-1)(n-2)\times\ldots\times(n-k+1)}{k!}.$$

Мы приводим вывод этой известной формулы ниже, в приложении 3. Легко проверить, скажем, что $C_n^1 = n$, и действительно мы можем выбрать одну позицию из n ровно n способами.

Значит, всего внутри шара

$$C_n^0 + C_n^1 + \ldots + C_n^d = \sum_{i=0}^{d} C_n^i$$

слов. Здесь слагаемое $C_n^0 = 1$ — это число слов, отстоящих от центра на расстояние 0. Такое слово только одно — сам центр. Поскольку шары с центрами в кодовых словах попарно не пересекаются, то всего в них находится $N\sum_{i=0}^{d} C_n^i$ различных слов. Но это количество заведомо не превосходит числа всех возможных кодовых слов, которое, как мы уже знаем, равно 2^n. Таким образом,

$$N\sum_{i=0}^{d} C_n^i \le 2^n, \text{ откуда } N \le \frac{2^n}{\sum_{i=0}^{d} C_n^i}.$$

Эта формула и есть *граница Хэмминга.* В нашем примере, когда $n = 10$, $d = 2$, получаем

$$C_{10}^0 + C_{10}^1 + C_{10}^2 = 1 + 10 + \frac{10\times 9}{2\times 1} = 56.$$

Всего последовательностей длины 10 ровно $2^{10} = 1024$. Получается, что максимальное количество кодовых слов не превышает $\frac{1024}{56} \approx 18{,}2857$. Поскольку число кодовых слов целое, оно не больше 18.

3. Число сочетаний из *n* по *k*

Мы рассмотрим число сочетаний на примере, связанном с кодированием. Давайте попробуем сосчитать, сколько существует слов длины n и веса k, $k \le n$. Напомним, что слово — это запись из нулей и единиц, а его вес — это количество единиц. Значит, нам нужно выбрать из n позиций k штук для расстановки на этих k выбранных позициях единиц. При этом ясно, что как только позиции будут выбраны, кодовое слово определяется однозначно. Выбрали, скажем, из шести позиций первую, четвертую и пятую — все, появилось кодовое слово 100110.

Хорошо, допустим, есть n позиций. Выбираем из них любую. Это можно сделать n способами. Для каждого из этих n способов выбора первой позиции из оставшихся $n-1$ позиций снова выбираем любую. Для этого уже есть только $n-1$ вариант. Итого количество способов зафиксировать первую и вторую позиции для единиц равно $n(n-1)$. Точно так же три позиции можно последовательно выбрать одним из $n(n-1)(n-2)$ способов. И так далее. Для данного k будет всего

$$n(n-1)(n-2)\times\ldots\times(n-k+1)$$

вариантов. Это и есть ответ? Не совсем!

Заметим, что в нашем примере, где $n = 6$ и $k = 3$, мы могли сначала выбрать, например, первую позицию, затем — четвертую и наконец — пятую, а могли сперва выбрать четвертую позицию, затем — пятую и лишь в конце — первую. И для каждого из подобных вариантов у нас получится одно и то же кодовое слово 100110. Сколько же раз в нашей формуле $n(n-1)(n-2)\times\ldots\times(n-k+1)$ мы тем самым посчитали одно и то же кодовое слово? Смотрите, получая эту формулу, мы выбирали какие-то последовательности номеров позиций: допустим, это были 1-4-5, 1-5-4, 5-1-4, 5-4-1, 4-1-5, 4-5-1.

Видно, что все эти последовательности дают одно и то же слово из нулей и единиц. И видно, что их 6. Чтобы снова прийти к этому выводу не путем унылого перебора (которым мы сейчас занимались), а «весело и с умом», надо рассудить так: из трех чисел 1, 4, 5 мы можем сначала выбрать любое (3 варианта); затем вслед за ним расположить второе уже только одним из двух способов, а третье выбирается однозначно. Рассуждение дает нужный результат: число способов упорядочить числа 1, 4, 5 равно $3 \times 2 \times 1 = 6$. Аналогично для любого k возникает формула $k(k-1)\times...\times 2\times 1 = k!$. Напомним: по принятой в математике конвенции $0! = 1$.

Что же мы имеем в итоге? Сначала мы вывели формулу $n(n-1)(n-2)\times...\times(n-k+1)$, потом сообразили, что в ней каждое множество позиций для единиц учтено *k!* раз. Это означает, что искомое количество кодовых слов равно

$$\frac{n(n-1)(n-2)\times...\times(n-k+1)}{k!}.$$

Заметим, что найденное выражение можно переписать в виде

$$\frac{n!}{k!(n-k)!}.$$

Это так называемое *число сочетаний* из *n* по *k*, или *биномиальный коэффициент*. В настоящее время для него приняты два обозначения: C_n^k и $\binom{n}{k}$.

Приложения к главе 4

В качестве дополнительного материала к этой главе рекомендуем книгу Андрея «Модели случайных графов» [4]. Там в подробностях приводится доказательство теоремы Эрдеша — Реньи и много других интересных результатов из теории случайных графов.

1. Вероятность потери связи в мини-сети

Рассмотрим мини-сеть как в примере, приведенном в главе: три компьютера — 1, 2, 3 и три канала связи: 1–2, 1–3, 2–3. Допустим, что канал связи доступен с вероятностью p и недоступен с вероятностью $1 - p$, где $0 < p < 1$. Предположим, что каналы независимы друг от друга. Связь между всеми тремя компьютерами сохраняется в двух случаях.

Случай 1. Все три канала связи доступны. Соответствующая вероятность равна

$$\underbrace{p}_{\text{канал 1–2}} \times \underbrace{p}_{\text{канал 1–3}} \times \underbrace{p}_{\text{канал 2–3}} = p^3. \qquad \text{(П.3)}$$

Вероятности перемножаются потому, что каналы независимы друг от друга. Допустим, у нас тысяча мини-сетей и $p = 0{,}6$. Тогда примерно в 600 из них будет доступен канал 1–2. Поскольку доступность канала 1–2 никак не влияет на канал 1–3, из нашей 1000 сетей оба канала 1–2 и 1–3 будут доступны в среднем $600 \times 0{,}6 = 360$ случаев. Чтобы вычислить среднее количество сетей, в которых доступны все три канала, надо взять долю 0,6 от 360, получаем $360 \times 0{,}6 = 216$. В результате вероятность доступности всех трех каналов равна

$$\frac{260}{1000} = 0{,}216 = 0{,}6 \times 0{,}6 \times 0{,}6\,.$$

Случай 2. Два канала связи доступны, и один недоступен. Недоступным может быть любой из трех каналов связи, поэтому случай 2 можно получить тремя разными способами. В результате соответствующая вероятность равна

$$\underbrace{(1-p)}_{\text{канал 1–2}} \times \underbrace{p}_{\text{канал 1–3}} \times \underbrace{p}_{\text{канал 2–3}} + \underbrace{p}_{\text{канал 1–2}} \times \underbrace{(1-p)}_{\text{канал 1–3}} \times \underbrace{p}_{\text{канал 2–3}} + \underbrace{p}_{\text{канал 1–2}} \times \underbrace{p}_{\text{канал 1–3}} \times$$

$$\times \underbrace{(1-p)}_{\text{канал 2–3}} = 3p^2(1-p). \qquad \text{(П.4)}$$

Общая вероятность сохранения связи в сети теперь равна

$$(\text{П.3}) + (\text{П.4}) = p^3 + 3p^2\,(1 - p) = 3p^2 - 2p^3.$$

Потеря связи хотя бы с одним из компьютеров тоже происходит в двух случаях, которые мы назовем случай 3 и случай 4.

Случай 3. Два канала связи недоступны и один доступен. Заметьте, что этот случай совершенно аналогичен случаю 2, только p и $(1 - p)$ поменялись местами. Соответствующая вероятность равна

$$3\,(p - 1)^2\,p. \qquad (\text{П.5})$$

Случай 4. Все три канала связи недоступны. Этот случай опять же аналогичен случаю 1, если поменять местами p и $(1 - p)$. Соответствующая вероятность равна

$$(1 - p)^3. \qquad (\text{П.6})$$

Вероятность, что хотя бы один компьютер окажется отрезанным от сети, равна

$$(\text{П.5}) + (\text{П.6}) = 3(1 - p)^2 - 2(1 - p)^3. \qquad (\text{П.7})$$

Естественно, если просуммировать все вероятности, то получится единица. Это очень красиво следует из бинома Ньютона третьей степени:

$$(\text{П.3}) + (\text{П.4}) + (\text{П.5}) + (\text{П.6}) =$$

$$= p^3 + 3p^2(1-p) + 3\left(p-1\right)^2 p + \left(1-p\right)^3 =$$

$$= \left(p + \left(1-p\right)\right)^3 = 1^3 = 1.$$

Если провести еще немножко алгебраических манипуляций, то можно переписать вероятность (П.7) по-другому:

$$3\left(1-p\right)^2 - 2\left(1-p\right)^3 = \left(1-p\right)^2\left(3 - 2\left(1-p\right)\right) = \left(1-p\right)^2\left(1+2p\right) =$$

$$= \left(1-p\right)\left(\left(1-p\right)\left(1+2p\right)\right) = \left(1-p\right)\left(1+p-2p^2\right).$$

Легко проверить, что если $p > \frac{1}{2}$, то вторая скобка меньше единицы. Получается, что если $p > \frac{1}{2}$, то вероятность потери связи в мини-сети меньше, чем вероятность потери связи в отдельно взятом канале, которая равна $(1 - p)$.

Кроме того, выражение (П.7) всегда меньше, чем $3\,(1 - p)^2$. Поэтому если вероятность неисправности канала $(1 - p)$ уменьшается, то вероятность потери связи в сети уменьшается еще быстрее. Когда вероятность $(1 - p)$ очень мала, то термином $-2\,(1 - p)^3$ можно пренебречь. Тогда вероятность потери связи в сети приблизительно равна $3\,(1 - p)^2$. Если $(1 - p) = 0{,}01$ (то есть 1 %), то эта формула верна до третьего знака после запятой, что мы и видим в последней строчке табл. 4.1.

2. Теорема Эрдеша — Реньи о фазовом переходе

Теорема (Эрдеша — Реньи). *Допустим, граф состоит из n вершин. Предположим, что ребра независимы друг от друга и любые две вершины соединены ребром с вероятностью p(n). Обозначим вероятность помехи через q(n) = 1 − p(n) и предположим, что*

$$q(n) = 1 - \frac{c \ln n}{n}. \qquad (\text{П.8})$$

(i) Если $c > 1$, то вероятность связности стремится к единице при n, стремящемся к бесконечности, причем скорость стремления к единице тем выше, чем больше число c. Например, при $c \geq 3$ скорость стремления к единице не ниже, чем у выражения $1 - \frac{1}{n}$, как в разделе «Результат Эрдеша — Реньи» в главе 4.

(ii) Если $c < 1$, то вероятность связности стремится к нулю при n, стремящемся к бесконечности, причем скорость стремления к нулю тем выше, чем меньше число c.

(iii) При $c = 1$ вероятность связности стремится не к нулю и не к единице, а к числу $e^{-1} \approx 0{,}3679$, где $e = 2{,}71828...$ — основание натурального логарифма.

3. Идея доказательства результата Эрдеша — Реньи

Допустим, граф состоит из n вершин. Предположим, что ребра независимы друг от друга и любые две вершины соединены ребром с вероятностью $p(n)$. Обозначим вероятность помехи через $q(n) = 1 - p(n)$. Рассмотрим одну вершину. Вероятность того, что она не соединена ребром ни с одной из оставшихся $n - 1$ вершин, равна

$$\big(q(n)\big)^{n-1}.$$

Поскольку всего вершин n, то в среднем число вершин, которые не соединены ребрами ни с одной другой вершиной, составит

$$n\big(q(n)\big)^{n-1}.$$

Возьмем, как и прежде,

$$q(n) = 1 - \frac{c\ln(n)}{n}.$$

Вспомните один известный замечательный предел

$$\lim_{n\to\infty}\left(1+\frac{1}{n}\right)^{n} = e,$$

где e — основание натурального логарифма. Интуитивно результат следует из похожего предельного перехода:

$$\lim_{n\to\infty} n\left(1-\frac{c\ln(n)}{n}\right)^{n-1} = \lim_{n\to\infty} ne^{-c\ln(n)} = \lim_{n\to\infty} n^{1-c}.$$

Давайте посмотрим, о чем нам говорит эта формула.

Если $c < 1$, то в среднем число вершин, у которых нет ни одного ребра, стремится к бесконечности. В этом случае

таких вершин будет очень много, связность сети с большой вероятностью будет потеряна.

Если $c > 1$, то в среднем число вершин, у которых нет ни одного ребра, стремится к нулю. Значит, с большой вероятностью таких вершин не будет и связность сети сохранится.

Таким образом, мы видим, откуда появляется фазовый переход!

Наконец, если $c = 1$, то в среднем число вершин, у которых нет ни одного ребра, равно единице. Заметим, что единица — это среднее значение, а в реальности таких вершин может быть 0, 1, 2... Можно доказать, что соответствующее распределение вероятности близко к закону Пуассона с параметром 1:

$$[\text{вероятность, что вершин с нулем ребер ровно } k] \approx \frac{1}{k!}e^{-1}, \quad k = 0,1,2...$$

Соответственно, вероятность того, что таких вершин не будет, то есть связность сети сохранится, равна e^{-1}.

Добавим, что это еще не строгое доказательство, потому что мы проанализировали только среднее количество вершин, у которых нет ни одного ребра. Для завершения доказательства нужно еще показать, что в случае $c < 1$ и $c > 1$ число вершин без ребер относительно ма́ло отклоняется от среднего значения. Для этого разработаны стандартные методы, в частности, основанные на неравенствах Маркова и Чебышева. Эти неравенства названы в честь замечательных русских математиков, стоявших у истоков теории вероятностей.

Приложение к главе 5

Анализ метода выбора из двух

Допустим, у нас n серверов. Заявки (или задания) поступают с интенсивностью λn в единицу времени, и каждый сервер в среднем обрабатывает одно задание в единицу времени, то есть загрузка системы равна $\lambda < 1$ (если $\lambda \geq 1$, то система перегружена, очередь будет расти до бесконечности). Рассмотрим случай, когда n очень велико и стремится к бесконечности.

Обозначим через f_k долю серверов, у которых ровно k заявок (заявка, которая находится на обслуживании в данный момент, тоже учитывается). Обозначим через u_k долю серверов, у которых заявок k *или больше*. Значения u_k можно легко получить через f_k и наоборот:

$$u_k = f_k + f_{k+1} + \ldots, \; f_k = u_k - u_{k+1}.$$

Понятно, что $u_0 = 1$.

Представим, что система находится в равновесии. Тогда у нас в среднем

$$nf_k = n\left(u_k - u_{k+1}\right) \qquad (\text{П.9})$$

серверов, на которых ровно k заданий. Все эти серверы обрабатывают задания со скоростью одно задание в единицу времени. Другими словами, количество серверов с k заявками или больше *уменьшается* на $n\left(u_k - u_{k+1}\right)$ в единицу времени.

Теперь давайте посмотрим, на сколько количество серверов с k заявками или больше *увеличивается* в единицу времени. Чтобы увеличить число таких серверов, заявки должны поступать на серверы, у которых в данный момент $k - 1$ заявка. При методе выбора из двух вероятность того, что новое задание попадет на сервер с k или больше заявками, равна

$$u_k^2, \qquad (\text{П.10})$$

потому что в этом случае оба случайно и независимо выбранных сервера должны иметь k или больше заявок, и для каждого из двух серверов эта вероятность u_k*. Значит, вероятность того, что новая заявка поступит на сервер, у которого ровно $k-1$ заявка, равна

$$u_{k-1}^2 - u_k^2 .$$

Поскольку заявок в единицу времени поступает λn, то получается, что число серверов с k или больше заявками в единицу времени в среднем *увеличивается* на

$$\lambda n\left(u_{k-1}^2 - u_k^2\right). \qquad (\text{П.11})$$

Так как система находится в равновесии, число серверов с k или больше заявками должно оставаться неизменным, то есть (П.9) равняется (П.11). Отсюда получаем уравнение баланса:

$$u_k - u_{k+1} = \lambda\left(u_{k-1}^2 - u_k^2\right),\ k = 1,2,\ldots;\ u_0 = 1. \qquad (\text{П.12})$$

Результат, приведенный в работе [1], говорит, что при определенных общепринятых предположениях о законе поступления заданий и времени их выполнения, в пределе для бесконечного количества серверов, уравнение (П.12) действительно описывает равновесное состояние системы. Это достаточно сложный технический результат. Нетрудно проверить или даже догадаться, что решение уравнения (П.12) задается формулами:

$$u_k = \lambda^{2^k-1},\ f_k = \lambda^{2^k-1} - \lambda^{2^{k+1}-1} \text{ для всех } k = 0,1,2,\ldots . \qquad (\text{П.13})$$

Это так называемый двойной экспоненциальный закон. Двойка в формуле, та самая двойка — вторая степень — из выражения (П.10). Точно так же мы могли бы выбирать не из двух, а из r серверов и получили бы $u_k = \lambda^{r^k-1}$.

---

* На самом деле, если нужно выбрать два разных сервера, эта вероятность равна $\frac{u_k n(u_k n-1)}{n^2}$, но это значение очень близко к u_k^2, когда n достаточно велико.

Интересно заметить, что при тех же предположениях, но случайном выборе одного сервера, изменятся выражения (П.10) и, соответственно, (П.11). Действительно, на этот раз заявка выбирает только один сервер и вероятность попасть на сервер с k или больше заявками равна просто u_k. Тогда вместо (П.12) получаем классическое уравнение баланса:

$$u_k - u_{k+1} = \lambda\left(u_{k-1} - u_k\right),\ k = 1,2,\ldots;\ u_0 = 1,$$

решение которого задается известной формулой Эрланга:

$$u_k = \lambda^k,\ f_k = (1-\lambda)\lambda^k, \text{ для всех } k = 0,1,2,\ldots. \qquad (П.14)$$

Очевидно, что u_k в формуле (П.13) убывает гораздо быстрее.

Именно эти формулы мы использовали в табл. 5.1. В нашем случае $\lambda = 0{,}9$, и в таблице мы привели значения f_k. В первой колонке — значения k, во второй — значения f_k, подсчитанные по формуле (П.14), в третьей — значения f_k, подсчитанные по формуле (П.13).

Приложения к главе 6

1. Схема Диффи — Хеллмана

Для начала введем обозначения. Пусть p — *заданное* простое число, g — *заданное* натуральное число, $g < p$. На самом деле g это так называемый *первообразный корень* числа p. Об этом мы расскажем ниже, в приложениях 2 и 3 к главе 6. Цель данного раздела — доказать, что в схеме Диффи — Хеллмана Алиса и Боб действительно получают один и тот же ключ.

Для любых натуральных чисел n и p мы воспользуемся стандартным обозначением для остатка от деления n на p:

$$n(\mathrm{mod}\ p) = [\text{остаток от деления } n \text{ на } p].$$

(Читается «n по модулю p».)

Итак, Алиса задумала число x, а Боб число y. Схема Диффи — Хеллмана состоит из двух шагов.

Шаг 1. Алиса передает Бобу

$$a = (g^x) \pmod{p}.$$

Боб передает Алисе

$$b = (g^y) \pmod{p}.$$

Шаг 2. Алиса вычисляет ключ

$$K_A = (b^x) \pmod{p}.$$

Боб вычисляет ключ

$$K_B = (a^y) \pmod{p}.$$

Утверждение. *Боб и Алиса получили один и тот же ключ* $K = K_A = K_B$.

Доказательство. Нам нужно доказать, что $K_A = K_B$. Поскольку a и b — это остатки от деления на p, то существуют такие целые числа k и l, при которых

$$a = g^x - kp,$$
$$b = g^y - lp.$$

Подставив эти выражения в формулы для ключей, получаем:

$$K_A = \left(\left(g^y - lp\right)^x\right) \pmod{p},$$

$$K_B = \left(\left(g^x - kp\right)^y\right) \pmod{p}.$$

Заметим, что в выражении для K_A можно расписать $(g^y - lp)^x$ следующим образом:

$$\left(g^y - lp\right)^x = \underbrace{\left(g^y - lp\right)\left(g^y - lp\right)\ldots\left(g^y - lp\right)}_{x \text{ раз}} = \left(g^y\right)^x + pA,$$

где A — это целое число, то есть pA делится на p. Таким образом получаем

$$K_A = \left(\left(g^y - lp\right)^x\right) \pmod p = \left(\left(g^y\right)^x + pA\right)$$
$$\pmod p = \left(g^y\right)^x \pmod p.$$

Совершенно аналогично для какого-то целого числа B получаем

$$K_B = \left(\left(g^x - kp\right)^y\right) \pmod p = \left(\left(g^x\right)^y + pB\right)$$
$$\pmod p = \left(g^x\right)^y \pmod p.$$

Результат теперь очевиден, поскольку

$$\left(g^y\right)^x = g^{yx} = g^{xy} = \left(g^x\right)^y.$$

2. Дискретное логарифмирование

Вспомним, что логарифм числа y по основанию g — это такое число x, для которого выполняется

$$g^x = y.$$

Легко заметить, что очень похожая операция лежит в основе схемы Диффи — Хеллмана.

После возведения в степень мы берем остаток от деления на p. Как мы уже упоминали выше, в математике такая операция обозначается $g^x \pmod p$ (читается «g в степени x по модулю p»). При этом, естественно, g и x натуральные числа и у g нет общих делителей с p.

Одна нетривиальная теорема из теории чисел (см., например, [2]) утверждает, что для каждого простого p *существует* такое число g, что все числа

$$g^1 \pmod p, g^2 \pmod p, \ldots, g^{p-1} \pmod p \qquad (П.15)$$

разные, то есть служат перестановкой множества $1, 2, \ldots, p-1$ (среди них нет нуля, поскольку $g < p$ и, значит, g^x не делится на p). Например,

$$3^1 \pmod 7 = 3,\ 3^2 \pmod 7 = 2,\ 3^3 \pmod 7 = 6.$$

$$3^4 \pmod 7 = 4,\ 3^5 \pmod 7 = 5,\ 3^6 \pmod 7 = 1.$$

Из этого следует возможность корректного определения *дискретного логарифма,* который еще называется *индексом.* А именно: «логарифм» произвольного числа $y \in \{1,\ 2,..., p-1\}$ по основанию g — это тот самый, уникальный, $x \in \{1,\ 2,..., p-1\}$, при котором выполняется $g^x \pmod p = y$. Поскольку для всех $x < p$ остатки разные, то x определяется однозначно. Операция нахождения такого x называется *операцией дискретного логарифмирования.* Она очень сложная, и никто не знает, можно ли придумать способ ее быстрой реализации.

Как определить такое число g, чтобы все остатки в выражении (П.15) были разные? Число g обладает этим свойством, если оно является *первообразным корнем* числа p. Мы позволим себе рассказать об этом понятии немножко подробнее.

3. Первообразные корни

Есть такая теорема Эйлера: если x и m взаимно просты, то $g^{\varphi(m)} \equiv 1 \pmod m$. Здесь $a \equiv b \pmod m$, если $a - b$ делится на m. Другими словами, у a и b одинаковый остаток от деления на m. А $\varphi(m)$ — это функция Эйлера, равная количеству чисел, не превосходящих m и взаимно простых с ним. Например, если $m = p$, где p простое, то $\varphi(p) = p - 1$ и теорема Эйлера превращается в малую теорему Ферма.

Условие теоремы Эйлера достаточное, но не необходимое. Вполне может случиться и так, что $x^a \equiv 1 \pmod m$, хотя $a < \varphi(m)$. Самый простой пример такой ситуации — это, конечно, $x = 1$. Действительно, $x^a \equiv 1 \pmod m$ для любых натуральных a и m. Но есть и менее тривиальные примеры. Скажем, $p = 5$, а $4^2 = 16 \equiv 1 \pmod 5$, хотя $2 < p - 1 = 4$.

Формально число g называется *первообразным корнем* по модулю m, если

$$g^{\varphi(m)} \equiv 1 \pmod{m}, \text{ но } g^a \not\equiv 1 \pmod{m}$$
$$\text{при всех } a < \varphi(m) \text{ и } a \neq 0.$$

Пример (отсутствие первообразных корней у $m = 2^k$). Возьмем $m = 2^k$ при $k \geq 3$. В этом случае можно показать, что для любого натурального x выполняется $x^{2^{k-2}} \equiv 1 \pmod{m}$. При этом $\varphi(m) = 2^{k-1}$, потому что числа, взаимно простые со степенью двойки, — это все нечетные числа, а их ровно 2^{k-1}. Значит, для любого x нашлось число

$$a = 2^{k-2} < 2^{k-1} = \varphi(m),$$

для которого выполняется $x^a \equiv 1 \pmod{m}$. Получается, что у $m = 2^k$ при $k \geq 3$ первообразных корней нет.

Теперь мы можем объяснить, почему в качестве m удобно брать простое число p. Для простого p всегда существуют первообразные корни. На самом деле мы уже говорили о них выше, в приложении 2 к главе 6, только не называли этим термином. Это те самые числа g, которые дают разные остатки от деления на p в (П.15). Например, при $p = 3$ это $g = 2$, при $p = 5$ это $g = 2$, а при $p = 7$ это $g = 3$. В нашем примере в тексте главы число $g = 2$ — это один из первообразных корней числа $p = 19$.

Итак, если g — первообразный корень p, то все остатки в (П.15) разные и каждому остатку соответствует единственная степень x (дискретный логарифм, он же индекс), при которой такой остаток получается. Ничего подобного мы не можем сделать, если будем брать остаток от деления на число m, для которого первообразного корня нет. Именно поэтому работают с простыми числами.

Заметим, что первообразные корни есть еще для $m = 4$, $m = p^k$ и $m = 2p^k$. Но все равно с простыми числами работать проще.

Приложение к главе 7

Двойной логарифм в HyperLogLog

Хеш-значения — это последовательности из нулей и единиц одинаковой длины. Обозначим длину хеш-значений через m. Тогда получим 2^m разных хеш-значений (см. главу 3 и приложение 1 к ней).

Теперь допустим, что нам нужно сосчитать n различных объектов. Чтобы присвоить им всем разные хеш-значения, понадобится как минимум n хеш-значений, то есть

$$2^m > n \text{ или } m > \log_2(n).$$

Стало быть, m должно быть того же порядка величины, что и $\log_2(n)$.

Алгоритм *K-Minimum Values* запоминает K самых маленьких хеш-значений длины m, то есть для этого алгоритма нам нужен объем памяти примерно $K \log_2(n)$.

HyperLogLog запоминает только максимальное количество нулей в начале хеш-значений. Если сами хеш-значения длины m, то и нулей может быть не больше, чем m. Значит, нам нужно держать в памяти число между 0 и m. Оно тоже записывается с помощью последовательности нулей и единиц. Какой длины должны быть эти последовательности? Если последовательности длины l, то мы можем записать 2^l разных чисел. Стало быть, чтобы записать $m + 1$ разных чисел, должно выполняться

$$2^l = m + 1 \text{ или } l = \log_2(m + 1) \approx \log_2(m).$$

В памяти нам нужно держать только l битов информации (последовательность из нулей и единиц длины l). Из предыдущих формул получается:

$$l \approx \log_2(m) \approx \log_2\left(\log_2(n)\right).$$

Для повышения точности хеш-значения разбивают на r регистров и запоминают максимальное число нулей в каждом регистре отдельно. В результате требуется порядка $r\log_2(\log_2(n))$ битов памяти. Относительная точность оценки задается формулой $\frac{1{,}04}{\sqrt{r}}$ [16].

Приложение к главе 8

Доказательство совместимости по стимулам аукциона второй цены

Рассмотрим аукцион второй цены, на котором один товар разыгрывается между n участниками. Обозначим v_i истинную ценность товара для участника i (от английского слова *value — ценность)*. Далее обозначим через b_i ставку участника i (от англ. *bid — ставка*). Эти обозначения будем использовать для любого $i = 1, 2, ..., n$.

Совместимость по стимулам означает, что верна следующая теорема.

Теорема (Викри). *Участник i получает максимальную прибыль, если* $b_i = v_i$.

Доказательство. Если участник i получает товар, следовательно, его ставка b_i была самой высокой. Поскольку мы имеем дело с аукционом второй цены, то стоимость товара равна самой высокой из оставшихся ставок:

$$[\text{стоимость товара, если товар достался участнику } i] = \max_{j \neq i} b_j .$$

При этом ценность товара для участника i равна v_i. Значит, если участник i получает товар, то его прибыль составит

$$v_i - \max_{j \neq i} b_j . \qquad (\text{П.16})$$

Если участник i товар не получает, он не приобретает никакой ценности и ничего не платит, то есть его прибыль равна нулю.

Дальше доказательство ведется так же, как в разделе «Результат Викри» в главе 8, и в качестве иллюстрации мы можем по-прежнему использовать рис. 8.2 и 8.3. Допустим, что ставки всех участников, кроме i, фиксированы. Мы покажем, что при правдивой ставке $b_i = v_i$ прибыль участника i та же или больше, чем при повышенной ставке $b_i > v_i$ или пониженной $b_i < v_i$. Подчеркнем, что это утверждение верно при *любых* (фиксированных) ставках других участников.

Предположим, что $b_i > v_i$. Рассмотрим три случая относительно ставок b_j, $j \neq i$.

1. Допустим, что b_i — самая высокая ставка и, кроме того, все остальные участники *поставили меньше, чем* v_i (рис. 8.2 вверху). Тогда товар по-прежнему достается участнику i по стоимости $\max_{j \neq i} b_j$, и участник i получает точно такую же прибыль (П.16), что и при правдивой ставке v_i.

2. Теперь предположим, что другой участник k сделал ставку $b_k > b_i$ (см. рис. 8.2 в середине). Тогда участник i товар не получает, его прибыль равна нулю. Но поскольку $b_i > v_i$, то и при честной ставке v_i участник i тоже не получил бы товар. Значит, в этом случае прибыль участника i при честной ставке тоже равна нулю.

3. Наконец, допустим, что b_i — самая высокая ставка и $v_i < \max_{j \neq i} b_j < b_i$ (см. рис. 8.2 внизу). Так как $v_i < b_i$, такая ситуация возможна. Она возникает, когда самая высокая ставка других участников выше v_i, но ниже b_i. Если бы i поставил v_i, то i не получил бы товар, прибыль была бы равна нулю. Но теперь b_i — самая высокая ставка, поэтому товар достается i. Прибыль i по-прежнему вычисляется по формуле (П.16), но только прибыль становится отрицательной, поскольку ценность товара меньше его стоимости. Значит, в этом случае прибыль i меньше, чем при честной ставке.

Во всех трех случаях 1–3 участник i не получил прибыль выше, чем при честной ставке v_i.

Теперь предположим, что $b_i < v_i$, то есть ставка занижает реальную ценность. Опять рассмотрим три случая относительно ставок других участников b_j, $j \neq i$.

1′. Допустим, b_i — самая высокая ставка (рис. 8.3 вверху). Тогда товар по-прежнему достается участнику i по стоимости $\max_{j \neq i} b_j$. В этом случае прибыль участника i та же, что и прежде (П.16). Она в точности такая же, как и при честной ставке v_i.

2′. Теперь предположим, что другой участник l сделал ставку $b_l > v_i$ (рис. 8.3 в середине). В этом случае при честной ставке v_i участник i товар не получает.

Но тогда и при заниженной ставке $b_i < v_i$ участник i не получит товар. Значит, прибыль i равна нулю и при честной, и при заниженной ставке.

3′. Наконец, допустим, что $b_i < \max_{j \neq i} b_j < v_i$ (рис. 8.3 внизу). Так как $b_i < v_i$ такая ситуация возможна. Она возникает, когда самая высокая ставка других участников выше b_i, но ниже v_i. Тогда при честной ставке товар достался бы участнику i, и его прибыль, по формуле (П.16), была бы положительной. Но поскольку b_i теперь не самая высокая ставка, товар достанется другому участнику и прибыль участника i равна нулю. Значит, в этом случае прибыль i меньше, чем при честной ставке.

Во всех трех случаях 1′–3′ участник i не получил более высокую прибыль, чем при честной ставке v_i.

В результате делаем вывод, что и при заниженной, и при завышенной ставке участник i получает не больше, чем при честной ставке $b_i = v_i$. Таким образом, мы доказали, что в аукционе второй цены выгоднее всего делать честную ставку.

Литература

1. Введенская Н. Д. Система обслуживания с выбором наименьшей из двух очередей – асимптотический подход / Н. Д. Введенская, Р. Л. Добрушин, Ф. И. Карпелевич // Проблемы передачи информации. — 1996. — № 32 (1). — С. 20–34.
2. Виноградов И. М. Основы теории чисел. М.–Ижевск : НИЦ «Регулярная и хаотическая динамика, 2003.
3. Канторович Л. В. Об одном эффективном методе решения некоторых классов экстремальных проблем // Докл. АН СССР. 1940. Том 28. С. 212–215.
4. Райгородский А. М. Модели случайных графов. 2-е изд. М. : МЦНМО, 2016.
5. Савватеев А. Теория аукционов. Наиболее значимые достижения [Электронный ресурс]. — Режим доступа : https://www.youtube.com/watch?v=fCIU07zg9HQ&feature=youtu.be.
6. Слоэн Н. Дж. А., Мак-Вильямс Ф. Дж. Теория кодов, исправляющих ошибки. М. : Радио и связь, 1979.
7. Спенсер Дж., Алон Н. Вероятностный метод. М. : Бином. Лаборатория знаний, 2007.
8. Adrian David, Bhargavan Karthikeyan, Durumeric Zakir, Gaudry Pierrick, Green Matthew, et al. Imperfect forward secrecy: How diffie-hellman fails in practice // Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, p. 5–17. ACM, 2015.

9. Albert Reka, Jeong Hawoong and Barabasi Albert-Laszlo. Error and attack tolerance of complex networks // Nature, 406(6794):378–382, 2000.
10. Backstrom Lars, Boldi Paolo, Rosa Marco, Ugander Johan and Vigna Sebastiano. Four degrees of separation // Proceedings of the 4th Annual ACM Web Science Conference, p. 33–42. ACM, 2012.
11. Bixby E. Robert. A brief history of linear and mixed-integer programming computation. Documenta Mathematica, p. 107–121, 2012.
12. Doyle J. C., Alderson D. L., Li L., Low S., Roughan M., Shalunov S., Tanaka R. and Willinger W. The «robust yet fragile» nature of the Internet. PNAS, 102(41): 14497–14502, 2005.
13. Eager Derek L., Lazowska Edward D. and Zahorjan John. Adaptive load sharing in homogeneous distributed systems. Software Engineering, IEEE Transactions on, (5): 662–675, 1986.
14. Easley David and Kleinberg Jon. Networks, crowds, and markets: Reasoning about a highly connected world. Cambridge University Press, 2010.
15. Erdos Paul and Renyi Alfred. On the evolution of random graphs. Publ. Math. Inst. Hung. Acad. Sci, 5:17–61, 1960.
16. Flajolet Philippe, Fusy Eric, Olivier G. and Meunier Frederic. Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm // Philippe Jacquet, editor, Analysis of Algorithms 2007 (AofA07), volume AH of Discrete Mathematics and Theoretical Computer Science Proceedings, p. 127–146, 2007.
17. Heule Stefan, Nunkesser Marc and Hall Alexander. Hyperloglog in practice: Algorithmic engineering of a state

of the art cardinality estimation algorithm // Proceedings of the 16th International Conference on Extending Database Technology, pp. 683–692. ACM, 2013.

18. Holt Charles C. Learning how to plan production, inventories, and work force // Operations Research, 50(l):96–99, 2002.
19. Keevash Peter. The existence of designs // arXiv preprint arXiv.H01.3665, 2014.
20. Kroon Leo, Huisman Dennis, Abbink Erwin, Fioole Pieter-Jan, Fischetti Matteo, Maroti Gabor, Schrijver Alexander, Steenbeek Adri and Ybema Roelof. The new dutch timetable: The or revolution // Interfaces, 39(1):6–17, 2009.
21. Mahadevan P., Krioukov D., Fall K. and Vahdat A. Systematic topology analysis and generation using degree correlations // ACM SIGCOMM Computer Communication Review, 36(4):135–146, 2006.
22. Renyi A. and Erdos P. On random graphs // Publicationes Mathematicae, 6(290–297):5, 1959.
23. Richa Andrea W., Mitzenmacher M. and Sitaraman R. The power of two random choices: A survey of techniques and results // Combinatorial Optimization, 9:255–304, 2001.
24. Shannon Claude Elwood. A mathematical theory of communication // The Bell System technical Journal, 27:379–423, 623–656, 1949.
25. Sketch of the day: Hyperloglog — cornerstone of a big data infrastructure. http://research.neustar.biz/2012/10/25/sketch-of-the-day-hyperloglog-cornerstone-of-a-big-data-infrastructure/. Accessed: 2015-11-22.

26. Stokes Donald E. Pasteur's quadrant: Basic science and technological innovation. Brookings Institution Press, 2011.
27. Vickrey William. Counterspeculation, auctions, and competitive sealed tenders // The Journal of finance, 16(l):8–37, 1961.

Благодарности

Мы благодарим издательство «Манн, Иванов и Фербер» и его директора Артема Степанова за возможность опубликовать эту рукопись. Большое спасибо Ренату Шагабутдинову за быструю и позитивную обратную связь. Мы хотим выразить признательность ответственному редактору Наталье Шульпиной за помощь в подготовке книги к изданию. Благодаря высокому профессионализму Натальи наша совместная работа была творческой и приятной. Мы также благодарим всех сотрудников издательства, которые участвовали в создании книги.

Мы искренне признательны коллегам со всего мира за энтузиазм и поддержку этого проекта. Спасибо вам, Сергей Измалков, Алексей Герасимов, Александр Веремьев, Даниил Мусатов, Алексей Савватеев, Джерард Пост, Себастьяно Винья, Питер-Тьерк де Бур, Мор Харкол-Балтер, Роберт Ерра, Франк Тайсман, Кавита Раманан, Фэн Чжун, Сара ван де Гейр, Александр Схрейвер, Ярослав Кристул, Нардо Боргман, Мартье ван де Врухт, за материалы, интервью и экспертные отзывы, которые нам были нужны для каждой главы. Особая благодарность Константину Авраченкову за экспертные комментарии ко всей рукописи, сделанные им в ходе проекта, и Роберту Фицнеру за иллюстрации к главе 4.

Нелли

Это был большой и сложный проект. Список людей, принявших в нем участие, очень длинный, и я все равно не сумею назвать всех.

Прежде всего хочу выразить признательность Андрею за сотрудничество. Мне было интересно и комфортно с ним работать, я не уставала удивляться глубине и разнообразию его знаний. Спасибо, Андрей, что взялись со мной за этот проект!

Моя семья стала узким кругом наших первых читателей. Мой дедушка, замечательный физик и энтузиаст науки Виталий Анатольевич Зверев (ему 92 года!), читал все главы в день их появления. Я с нетерпением ждала его развернутых комментариев, которые подсказали нам много идей; например, именно из этих комментариев возник раздел о кодировании цветов в главе 3. Спасибо, деда Витя, ты неподражаемый пример настоящего ученого! Мой брат Петр Антонец — гуманитарий с искренним интересом к науке и технике — стал для меня «эталонным» читателем. Мы исправили все, что ему было непонятно! Спасибо моей маме Нине Зверевой, папе Владимиру Антонцу и сестре Катерине Петелиной за быстрые, оригинальные и неизменно позитивные отзывы о каждой главе. Особое спасибо маме за то, что несколько лет назад подтолкнула меня к писательству и оказывала постоянную вдохновляющую поддержку моей писательской карьеры.

Мы получили необыкновенную поддержку от широкого круга друзей, коллег и потенциальных читателей. Кроме уже перечисленных выше имен, попробую назвать тех, чей энтузиазм, идеи, цитаты и комментарии нашли отражение на этих страницах: Марк Хацернов, Людмила Остроумова-Прохоренкова, Нина Петелина, Татьяна Петелина, Диана Кобленкова, Михаил Федоткин, Наум Шимкин, Нико ван Дайк, Эрвин Ханс, Пим ван дер Хорн, Мартен ван Стейн, Марк Утц, Ян-Виллем Полдерман, Ричард Бушери, Ингеборг Бликкер и многие другие. Огромное спасибо!

Спасибо моей взрослой дочери Наталии Литвак, прекрасной подруге и специалисту по инновациям. Из наших разговоров я почерпнула много интересных идей о роли науки в практике. Спасибо моей дочке Пиали за веселый характер и искренний интерес к моему писательству. Надеюсь, когда-нибудь ты сможешь это прочитать! Спасибо моему мужу Пранабу за бесконечное тепло, понимание и терпение.

Андрей

Мне был очень интересен этот проект, поэтому, несмотря на жуткий постоянный цейтнот, я сделал все, чтобы посвятить ему достаточно времени. Я очень благодарен Нелли за то, что она меня втянула во все это. :-)

Об авторах

Нелли Литвак — профессор математики, преподаватель в Университете Твенте (Нидерланды), автор более 60 научных работ и книг. Мать двух дочерей.

Андрей Райгородский — федеральный профессор математики, автор более 150 научных статей, 20 книг и 10 онлайн-курсов. Директор физтех-школы прикладной математики и информатики, заведующий лабораторией продвинутой комбинаторики и сетевых приложений МФТИ, заведующий кафедрой дискретной математики МФТИ, руководитель исследовательской группы компании «Яндекс».

Максимально полезные книги

Заходите в гости:
http://www.mann-ivanov-ferber.ru/

Наш блог:
http://blog.mann-ivanov-ferber.ru/

Мы в Facebook:
http://www.facebook.com/mifbooks

Мы ВКонтакте:
http://vk.com/mifbooks

Предложите нам книгу:
http://www.mann-ivanov-ferber.ru/about/predlojite-nam-knigu/

Ищем правильных коллег:
http://www.mann-ivanov-ferber.ru/about/job/

Научно-популярное издание

Литвак Нелли

Райгородский Андрей

Кому нужна математика?
Научись превращать проблемы в возможности

Главный редактор *Артем Степанов*
Ответственный редактор *Наталья Шульпина*
Литературный редактор *Татьяна Сковородникова*
Арт-директор *Алексей Богомолов*
Дизайнер *Дана Хагабанова*
Верстка *Лариса Чернокозинская*
Корректоры *Вита Галич, Елена Попова*

КНИГА О ТОМ, КАК БЛАГОДАРЯ МАТЕМАТИКЕ ВЕРТИТСЯ СОВРЕМЕННЫЙ МИР

Даже сегодня, в эпоху цифровых технологий, многие люди считают математику абстрактной и ненужной наукой. Нелли Литвак и Андрей Райгородский решили исправить эту несправедливость, написав книгу о том, как эта наука применяется в современной жизни. Из нее вы узнаете, как математика помогает:

— планировать и составлять расписания (например, железных дорог);
— кодировать тексты;
— обеспечивать надежную связь в сети;
— балансировать нагрузку в телекоммуникациях;
— шифровать сообщения;
— распределять рекламные места в поисковых сетях.

Эта книга — легкая, интересная и понятная — будет доступна и полезна любому, даже неподготовленному читателю. Те же, кто неплохо разбирается в математике, смогут узнать больше подробностей из приложений.

ISBN 978-5-00100-521-6
9 785001 005216

издательство
МАНН, ИВАНОВ И ФЕРБЕР

Максимально полезные книги
на сайте **mann-ivanov-ferber.ru**

Like facebook.com/**mifbooks**
vk.com/**mifbooks**
instagram.com/**mifbooks**