МАТЕМАТИКА ИНФОРМАЦИОННОГО ВЕКА

Современные приложения математики

НЕЛЛИ ЛИТВАК
АНДРЕЙ РАЙГОРОДСКИЙ

2016
ИЗДАТЕЛЬСТВО

Оглавление

Глава 1

Введение

1.1 О чем эта книга

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

- Задачи планирования и составление расписаний.
  (глава “Менеджмент и многогранники”)
- Кодирование текстов для хранения и передачи в цифровом виде.
  (глава “Мир нулей и единиц”)
- Структура соединения серверов каналами связи в Интеренете.
  (глава “Устойчивость Интернета”)
- Балансирование нагрузки в теллекоммуникациях.
  (глава “Сила выбора из двух”).
- Шифрование конфиденциальных сообщений.
  (глава “Секретные числа”).
- Операции подсчета при анализе больших даных.
  (глава “Счетчики с короткой памятью”)
- Распределение рекламных мест в поисковых системах, как Гугл 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, создание подвижной трехмерной картинки на экране при компьютерной томографии, или нахождение мутаций в геномах клеток рака. Мы также оставили в стороне приложения в высокотехнологичном производстве, например, в машиностроении и авиации, а также широкий спектр применений в экономике.

Темы, которые мы выбрали, 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 может творить те чудеса, к которым мы уже так привыкли: показывать тексты и фотографии, делать сложные рассчеты, искать информацию, и пересылать все это по всему миру. За всем этим стоит математика, без которой компьютер остался бы безжизненным и практически бесполезным аппаратом.

Конечно, даже эту тему мы не можем полностью раскрыть в одной книжке. Мы выбрали только несколько примеров, которые наиболее близки к теме наших собственных исследований. Если бы эту книгу писали другие авторы, то и темы были бы другие, и не менее впечатляющие.

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

1.2 Для кого эта книга

Эта книга для широкого круга читателей и не требует никакой подготовки. При этом для наиболее заинтересованных и подготовленных читателей мы добавили дополнительные детали. Поэтому кроме основного текста, в книку входят дополнительные объяснения и приложения.

Основной текст. Основной текст книги не требует абсолютно никакой математической подготовки. Мы постарались, чтобы каждому читателю было интересно и понятно, о чем идет речь. В каком-то смысле наш рассказ можно сравнить с научно-популярными программами по телевизору.

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

Приложения. Наконец, в конце книги, мы включили Приложения, где мы приводим более строгие математические формулировки, включая формулы, теоремы, и доказательства (или хотя бы идеи доказательств). Приложения рассчитаны на уровень старшеклассников, которые интересуются математикой.

Мы включили Приложения, чтобы книжку можно было использовать в качестве учебника, например, для спецкурса в старших классах или для вводных лекций в вузе.

1.3 Спасибо!

Мы благодарим издательство МИФ и его директора Артема Степанова за возможность опубликовать эту рукопись. Больше спасибо Ренату Шагабутдинову за быструю и позитивную обратную связь... (редакторы, и т.д.).

Мы от всего сердца благодарим коллег по всему миру за энтузиазм и поддержку этого проекта. Спасибо за материалы, интервью, и экспертные отзывы, которые нам были нужны в каждой главе: Сергей Измалков, Алексей Герасимов, Александр Веремьев, Даниил Мусатов, Алексей Савватеев, Robert Fitzner, Gerard Post, Sebastiano Vigna, Peter-Tjerk de Boer, Mor Harchol-Balter, Robert Erra, Frank Thuijsman, Kavita Ramanan, Fan Chung, Sara van de Geer, Alexander Schrijver, Jaroslav Krystul, Nardo Borgman, Maartje van de Vrugt. Особая благодарность Константину Авраченкову за экспертные комментарии по всей рукописи и в течение всего проекта.

Нелли:
Это был большой и сложный проект. Список людей, которых я хочу поблагодарить, будет очень длинный, и я все равно не сумею назвать всех.

В первую очередь, я очень благодарна Андрею за сотрудничество. Мне было интересно и приятно 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 91 год!) читал все главы в день появления. Я с нетерпением ждала его развернутых комментариев, которые дали нам много идей, например, именно из этих комментариев возник Раздел 4.6 про кодирование цветов. Спасибо, деда Витя, ты неподражаемый пример настоящего ученого! Мой брат Петр Антонец – гумманитарий 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 всегда позитивные отзывы по каждой главе. Особое спасибо маме за то, что несколько лет назад подтолкнула меня к писательству, и за постоянную вдохновляющую поддержку моей писательской карьеры.

Мы получили необыкновенную поддержку от широкого круга друзей, коллег и потенциальных читателей. Кроме имен, уже перечисленных выше, 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, Naum Shimkin, Nico van Dijk, Erwin Hans, Pim van der Hoorn, Maarten van Steen, Marc Uetz, Jan Willem Polderman, Richard Boucherie, Ingeborg Blikker, и многие другие. Огромное спасибо!

Спасибо моей взрослой дочери Наталии Литвак, прекрасной подруге и специалисту по инновациям. Наши разговоры дали мне 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 уделить ему должное время. Я очень благодарен Нелли за то, что она меня втянула в это :)

Глава 2

“Зачем нужна математика?”

2.1 Нелли

После длинного перелета и полутора часов 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.

– На какую тему конференция?

– 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. Он сканирует мой паспорт, в долю секунды компьютер находит мои отпечатки пальцев среди миллионов других отпечатков и сравнивает мой файл с сотнями тоненьких линий на маленьком сканере. Каким образом, интересно, 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 каким образом компьютер – кучка пластмассы и железа – может решить, совпадают ли две картинки, отфильтровав при этом неизбежные помехи и неточности?

Можно копать и дальше, разбираться в конструкции самолета (максимальная прочность при минимальном весе), вникать в таинственную систему определения стоимости билетов, и так далее, и тому подобное. И ничего этого — то есть абсолютно ничего! -– нельзя было бы сделать без математики. Самолеты, цены, линии на пальцах — все это было описано с помощью переменных, функций и уравнений. А потом были найдены эффективные и точные способы решения. Потому что у компьютера нет глаз, и при этом он должен узнать мой отпечаток пальца быстро и точно. Це-

лая 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, это 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 из них 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 книжке мне хотелось понятно и интересно рассказать именно о полезности математики. Ну и про красоту расскажем тоже. И надеемся, что читатель сможет ее увидеть и оценить.

2.2 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 заканчивал знаменитую 2-ю школу. 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) – сперва у Королева в Подлипках, потом у Лавочкина в Химках. Он, пожалуй, и оказал наибольшее влияние на меня. Я тоже учился на мехмате МГУ. Там огромное влияние на мой выбор математики в качестве профессии оказал 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 я 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 считаю, 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 мате-

матику оправдывают ее приложения (она прекрасна сама по себе), но сами приложения возникают за счет того, что так мир устроен, и математика как раз об этом, об устройстве мира.

Моя наука - комбинаторика - замечательна тем, что очень многие формулировки и доказанные сложные результаты понятны даже школьнику, который интересуется 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 расскажем о некоторых таких задачах, которыые до сих пор остаются открытыми, несмотря на их актуальность.

2.3 Лучший ответ на вопрос «зачем нужна математика?»

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

Какое-то время назад немецкое правительство решило выделить значительные суммы целевым образом 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 о том, как без их науки миру и Германии придет конец. Все они щедро перебирали отпущенное время. Гротшел выступал последним. Заседание уже совсем 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, я просто приведу пример. Недавно мы получили заказ от большой страховой компании, которая хотела организовать автосервис для своих клиентов. Идея очень простая: если у клиента в дороге сломалась машина, то он может позвонить по телефону, и аварийная служба быстро приедет к нему на помощь. Вопрос в том, как правильно организовать такой сервис. В принципе, задачу можно решить очень просто. Например, можно к каждому клиенту приставить личную аварийную машину с механиком. Тогда клиент получит помощь немедленно и в любой момент! Но это очень дорого! Другой вариант – это вообще не связываться с аварийным сервисом. Клиенты могут ждать до бесконечности, зато это не будет стоить ни цента. Так вот. Если вас эти решения не устраивают, то я должен вам сообщить, что для любых других вариантов вам необходима 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 ответила, что без математики невозможно было бы составить расписание поездов, и поезда бы все время сталкивались друг с другом. О расписаниях поездов мы расскажем подробнее в Главе 3. Но сначала немножко поговорим о том, чем занимаются профессиональные математики, от выпускников вуза в компаниях до ведущих ученых-теоретиков.

2.4 Математика на каждый день

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

Сфера деятельности математиков очень широкая: логистика, планирование, высокотехнологичное производство, био-медицинские 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, целыми днями решаю дифференциальные уровнения. А ребята, которые торгуют, сидят тут же, в трех метрах от меня. Вот сейчас досчитаю, скажу им, что покупать."

Среди ученых-математиков есть такие, кто напрямую работает с приложениями.

Например, Мор Харкол-Балтер (Mor Harchol-Balter) из университета Карнеги-Меллон говорит, что все ее исследования основаны на приложениях. Например, в 2011 году она сотрудничала с Фейсбук. По оценкам Мор, Фейсбук задействовал свои включенные серверы не больше чем наполовину, а остальное время серверы простаивали. Включенный и незадействованный сервер тратит примерно 2/3 энергии работающего сервера. Но компании бояться выключать сервера, потому что чем больше серверов, 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 Фейсбук 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 рекоммендациям, и как они сами утверждают, это экономит им 10-15% энергии.

Профессора университета Твенте Ричард Бушери (Richard Boucherie) и Эрвин Ханс (Erwin Hans) и их ученики занимаются логистикой здравоохранения. В результате этих исследований в больницах в Нидерландах произошли существенные изменения. Например, больница в Роттердаме раньше держала наготове специальную операционную для экстренных операций. Бо́льшую часть времени операционная пустовала, драгоценное время тратилось впустую. Но менеджмент опасался, что без

специальной операционной экстренным пациентам придется ждать слишком долго. При этом ждать им приходилось все равно, например, если вдруг привозили сразу двух экстренных пациентов. Математические подсчеты показали, что при правильно составленном расписании плановых операций (еще одна нетривиальная задача!) можно 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 так близко, как в примерах выше. Разработка новых теорий важна для практики не меньше, чем решение непосредственных практических задач. Об этом мы поговорим подробнее в следующих разделах.

2.5 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 Силиам Госсет. Госсет работал на пивоваренном заводе Гинесс в Дублине. Его исследования по статистике 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, и поэтому он публиковался под псевдонимом Стьюдент. Госсет вывел новое распределение вероятностей, распределение Сьюдента, и на его основе разработал теперь уже классическую статистическую процедуру, знаменитый t-тест.

t - тест часто используется, когда нужно 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 вы хотите сравнить урожайность при двух разных видах удобрений. Такие тесты широко используются на практике, для них разработано стандартное программное обеспечение, t-тест не проходят разве что на филфаке.

За сто лет статистка ушла далеко вперед. Сара ван дер Гер (Sara van der Geer), профессор Швейцарской высшей технической школы Цюриха, работает над тестами с многомерными данными. Задача, также как и задача Госсета, пришла из практики. Компания DSM в Швейцарии производит витамины и пищевые добавки. Витамин B2 производится с помощью бациллы сенной палочки. Компания хочет увеличить производство витамина с помощью генной инженерии. Имеются измерения производительности 115 бактерий, генный состав которых включает 4088 возможных генов. Спрашивается, какие гены способствуют увеличению производства витамина B2.

Это очень сложная задача, потому что данных мало, а параметров много, причем все параметры связаны между собой. Существующие теории не годятся для этого случая, и поэтому Сара и ее сотрудники работают над созданием 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 ответила совершенно серьезно: "Конечно, очень скоро это будет стандартная статистика".

Современная реальность становится все сложнее, и существующего математического аппарата часто не хватает. Это, безусловно, мощный источник для новых задач и теорий.

2.6 Математика неизвестного будущего

Не все математические задачи приходят из практики. Так и должно быть, потому что мы не можем с уверенностью предсказать развитие общества и технологий даже на ближайшее будущее. Это не по силам даже самым информированным людям с самой необыкновенной фантазией. Например, хорошо известно, что писатели-фантасты практически ничего не сумели предсказать. В основном они описывали технологии своего времени, приукрашенные фантастическими деталями.

Никто не смог предсказать появление Интернета. Наоборот, Нобелевский лауреат Деннис Габор, изобретатель голографии, в 1962 году сказал, что передача документов по телефону, хоть и возможна в принципе, но требует таких огромных расходов, что эта идея никогда не найдет применения на практике. При этом первый успешный модем появился в том же самом году! А Кен Олсен, один из создателей Digital Equipment Corporation (DEC), в 1977 году заявил, что нет такого человека, которому может дома понадобиться компьютер. Через сколько лет после этого компьютер появился в вашем доме?

Мы не знаем, какая абстрактная теория завтра может понадобиться на практике. Потрясающий пример – теория чисел, область математики, которая изучает числа и их закономерности. Теория чисел оставалась абстрактной наукой со времен древней Греции до второй половины двадцатого века. Сегодня эта теория широко используется для шифрования сообщений, которые передаются через Интернет. Именно благодаря теории чисел ваши пароли и номера кредитных карточек остаются конфиденциальными, когда вы вводите их на многочисленных веб-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. Они нужны просто ради красоты этих теорий! Красивая математика имеет полное право на существование. В научном мире должно оставаться что-то от Касталии 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 несет функцию просветительства. Это единственная сфера деятельности, в которой человек может работать, движимый чистым непрактическим любопытством. Грубо говоря, наука делает мир умнее, и это необходимо человечеству так же, как искусство, которое делает мир духовнее.

Глава 3

Менеджмент и многогранники

3.1 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, который работает системным администратором в международной компании. Тот не задумался ни на секунду:

– Конечно, нужна. Вот недавно клиенты заказали программу, чтобы распределять товары по вагонам. Мы сразу поняли, что такую задачу каждый день решают все поставщики всех товаров. Значит, задача известная. Через полчаса мы уже знали, что это “задача об упаковке” и могли предложить несолько решений. Кстати, пришлось объяснять клиентам, 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 насколько часто его пополнять? Как составить расписание поездов, самолетов, большого производства, и даже спортивных соревнований?

Этими вопросами занимается область прикладной математики, которая называется *исследование операций*. По большому счету, это наука о том, как оптимально организовать процессы бизнеса и производства. Сюда, безусловно, относится логистика, но и 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 свой путь к применению на практике. В 2002 году, в специальном юбилейном выпуске в честь 50-летия журнала “Исследование Операций” (Operations Research), Чарльз Холт (Charles C. Holt) делится своими воспоминаниями о том, как он и его коллеги Франко Модильяни (Franco Modigliani), Джон Муф (John F. Muth), и Герберт Симон (Herbert A.

Simon), разрабатывали и внедряли научные методы планирования производства [18]:

> *"Мы взяли интервью у менеджеров 15 компаний. Поначалу менеджеры отрицали наличие каких-либо проблем. По крайней мере, таких проблем, с которыми коллеги-профессора могли хоть как-то помочь. Но когда мы расспрашивали более подробно, то возникали картины, напоминающие телегу на разваливающихся колесах: системы, которые катятся без всякого контроля, от одного кризиса к другому."*

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 Люкаса (Robert Lucas), который в последствии тоже получил Нобелевскую премию.

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 прикладной математике, но и в бизнес-школах.

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

3.2 Проклятие размерности

Сложность задач оптимизации заключается в том, что возможных решений невообразимо много. 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 выполнять задания? "Выгода" может зависить от срока выполнения, времени в очереди, и так далее.

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

Теоретически, все правильно. Но прежде, чем запустить нашу 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. Продлжаем дальше: 23 варианта для третьего места, 22 для четвертого, и так далее. Всего у нас способов получается

$$\begin{aligned}25 \times 24 \times 23 \times 22 \times 21 \times 20 \times 19 \times 18 \times \times 17 \times 16 \times 15 \times 14 \times \\ \times 13 \times 12 \times 11 \times 10 \times 9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = \\ = 15\,511\,210\,043\,330\,985\,984\,000\,000.\end{aligned}$$

Это число называется *двадцать пять факториал* и обозначается "25!" Насколько велико это число? Если взять современный процессор с тактовой частотой в 2ГГц (2 миллиарда операций в секунду), то чтобы выполнить такое количество операций, ему понадобится 245 миллионов лет! А на то, чтобы просчитать все варианты, с прибылью и убытками, да еще и перемещать информацию в памяти компьютера, уйдет гораздо больше времени. А ведь задачка была совсем простая, всего один прибор, всего 25 заданий. Не сравнить с серьезным современным производством.

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

Для некоторых задач удается найти гарантированно лучший ответ относительно быстро. Но есть и целый разряд так называемых *NP-трудных* задач, как задача об упаковке, о которой мы упоминали выше. Грубо говоря, для таких задач мы не можем придумать метода, который бы работал сильно быстрее, чем тривиальный полный перебор всех вариантов. Сможем ли когда-нибудь? Это открытый вопрос, но большинство ученых считают, что нет, не сможем, 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 смириться с тем, что ответ достаточно хороший, но не всегда самый выгодный из всех возможных.

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

3.3 Линейное программирование

Мы объясним, как возникают задачи линейного программирования, на еще одном простом примере.

Допустим, у нас есть два склада – на Северном и Южном конце города. В офис поступили заказы от двух клиентов. Клиент А заказал 60 листов железа, а клиент Б заказал 40 листов. На Южном складе у нас есть 70 листов железа, а на Северном 35, то есть общего запаса хватает. Мы хотим, чтобы расходы на доставку были минимальные. Цены доставки приведены в таблице ниже:

	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 наличие	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. И поскольку Северный склад сильно дороже, то решение неочевидно.

Как решить такую задачу? В таком простом примере можно решить и перебором всех вариантов. Но если клиентов и складов станет больше, то решить задачу вручную не удастся. Поэтому давайте посмотрим, как решить задачу 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 склада клиенту Б доставлено (40−*БЮ*) листов. Теперь по табличке можно посчитать общую стоимость доставки:

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

Если раскрыть скобки, то получается:

$$\text{общая стоимость доставки} = 1020 - 2 \times \text{АЮ} - 5 \times \text{БЮ} \quad \text{(рублей)}. \tag{3.1}$$

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

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

$$\text{АЮ} \le 60, \tag{3.2}$$
$$\text{БЮ} \le 40. \tag{3.3}$$

Во-вторых, нужно учесть, что запас на каждом складе ограниченый. С Южного склада мы отправляем *АЮ*+*БЮ* листов, а всего на этом складе 70 листов. Поэтому

$$\text{АЮ} + \text{БЮ} \le 70. \tag{3.4}$$

Аналогично, с Северного склада мы не можем отправить больше 35 листов:

$$(40 - \text{АЮ}) + (60 - \text{БЮ}) \le 35.$$

Раскрыв скобки в этом выражении, получаем

$$\text{АЮ} + \text{БЮ} \ge 65. \tag{3.5}$$

Это ограничение можно интерпретировать еще и так: поскольку на Северном скаладе всего 35 листов, а нам нужно доставить всего 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 (3.1), и при этом соблюсти ограничения (3.2)–(3.5). Внизу в серой рамке мы приводим нашу задачу в окончательном варианте.

Задача линейного программирования.

Выбрать АЮ и БЮ так, чтобы
минимизировать: $1020 - 2 \times \text{АЮ} - 5 \times \text{БЮ},$
при ограничениях:
$0 \leq \text{АЮ} \leq 60,$
$0 \leq \text{БЮ} \leq 40,$
$\text{АЮ} + \text{БЮ} \leq 70,$
$\text{АЮ} + \text{БЮ} \geq 65.$

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

Программирование в данном контексте означает не программирование на компьютере, а скорее "оптимизация". Слово *линейное* употребляется потому, что мы пользуемся исключительно линейными выражениями. Это означает, что переменные можно умножать на число, а также вычитать и складывать. И все. Никакие другие операции не используются. Например, у нас нет выражений типа $АЮ \times БЮ$ 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 нужно либо миинмизировать (если речь идет о стоимости), либо максимизировать (если речь идет о доходе). В данном случае наша целевая функция это стоимость (3.1), и ее нужно минимизировать.

3.4 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 году вышла его фундаментальная статья *Об одном эффективном методе решения некоторых классов экстремальных проблем* [5]. В этой работе Канторович заложил математические основы линейного программирования (которое тогда еще

так не называлось).

Канторович интересовался этими проблемами прежде всего из-за их ценности для практики. В 1975 году он получил Нобелевскую премию "за вклад в теорию оптимального распределения ресурсов". Премию он разделил с американским экономистом-математиком голландского происхождения Тьяллингом Купмансом (Tjalling Koopmans). Купманс тоже занимался разработкой теории линейного программирования и ее приложениями в экономике.

Одна из классических и знаменитых задач линейного программирования – это задача о диете Стиглера (G.J. Stigler), которая датируется 1945 годом. Формулировка этой задачи звучит примерно так: какие из 77 продуктов должны входить в потребительскую корзину одного человека (скажем, мужчины среднего веса), чтобы он получил необходимую норму питательных веществ, и чтобы 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, миниралов, и так далее, в каждом продукте известно. Ограничения – это минимальное количество питательных веществ. А минимизировать надо общую стоимость продуктов, то есть количество каждого продукта, помноженное на его цену.

Уже к концу 1950-х годов линейное программирование достаточно широко использовалось в нефтяной индустрии. Сейчас линейное программирование 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.

3.5 От задачи к решению

Несмотря на простую формулировку, решить задачу линейного программирования совсем 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 на Южном складе всего 70 листов, а нам нужно 100.

Чем больше переменных и ограничений, тем сложнее задача. В задаче о диете 77 переменных и 21 ограничение, и эта задача уже представляет из себя 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 Джоржа Данцига (George Dantzig) появились в конце 1940-х годов – несколько позже, чем работы Канторовича. Тем не менее, безо всяких сомнений, Данцига тоже относят к основателям линейного программирования. Именно он придумал так называемый *симплекс-метод*, который позволил, с помощью компьютера, быстро решать задачи линейного программирования с большим количе-

ством переменных и ограничений.

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

3.6 Идея симплекс-метода

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

Для начала давайте посмотрим, какие в принципе значения могу принимать переменные, чтобы не нарушить наши ограничения. Например, мы можем взять $АЮ = 58$, $БЮ = 8$. В этом случае получается решение, которое мы записали в виде таблицы:

План поставки листового железа					
	Заказано	Юж. склад	Сев. склад	Всего	Стоимость
Кол-во листов клиенту А	60	58	2	60	$5 \times 58 + 2 \times 7 = 304$ 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, скажем, 60 листов клиенту А и 10 листов клиенту Б. Легко увидеть, что доставка клиенту А теперь обойдется в

$$5 \times 60 = 300 \quad \text{руб.},$$

а доставка клиенту Б будет стоить

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

Теперь общая стоимость получается не 864, а 850 рублей, то есть немножко меньше.

Чтобы не выбирать наугад, нужно посмотреть на все возможные решения, которые удовлетворяют ограничениям. Мы их изобразили на Рисунке 3.1. По оси x мы откладываем *АЮ*, а по оси y откладываем *БЮ*. Любая точка в заштрихованной области удовлетворяет ограничениям. В том числе точка (58,8), как в таблице выше.

Ниже в серой рамке мы объясняем, как получилась заштрихованная область. Объяснения на уровне средней школы, но их можно пропустить.

Как получена заштрихованная область на Рисунке 3.1.
Все значения *АЮ* и *БЮ* положительные.
Вертикальная лиловая прямая АЮ=60 обеспечивает ограничение АЮ ≤ 60. Все возможные значения находятся слева от этой прямой.
Аналогично, горизонтальная фиолетовая прямая БЮ $= 40$ обеспечивает ограничение БЮ ≤ 40. Все возможные значения находятся под этой прямой.
Выражение для синей прямой АЮ + БЮ $= 70$ можно переписать в более при-

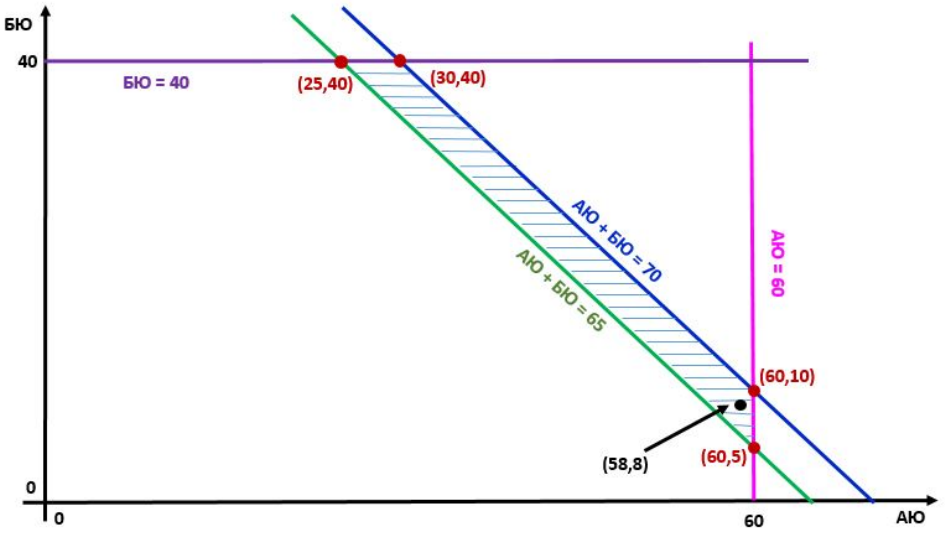

Рис. 3.1: Любая точка в заштрихованной области удовлетворяет всем ограничениям. Черная точка (58,8) это решение из таблицы выше. Угловые красные точки (25,40), (30,40), (60,10) и (60,5) – кандидаты на оптимальное решение (см. объяснение в тексте).

вычном виде:

$$\text{БЮ} = 70 - \text{АЮ},$$

поэтому прямая идет под отрицательным углом в 45^0. Заметьте, что прямая 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, которые удовлетворяют этому неравенству, находятся *под* синей прямой.
Аналогично, зеленая прямая АЮ $+$ БЮ $= 65$ обеспечивает ограничение АЮ $+$ БЮ ≥ 65. Значения, которые удовлетворяют этому неравенству, находятся *над* зеленой прямой.
Заштрихованная область удовлетворяет всем ограничениям.

Важно заметить, что заштрихованная область – это четырехугольник с прямыми сторонами, потому что все наши ограничения *линейные*, то есть их можно изобразить с помощью прямых линий. Эта область называется *областью допустимых значений*, потому что все значения в этой области удовлетворяют всем ограничениям. То есть любое решение из это области физически возможно, или, *допустимо*.

Самое главное 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, – то стоимость может либо только уменьшаться, либо только увеличиваться, пока мы не наткнемся на угол, так что идти дальше по той же прямой станет невозможно.

> Для подготовленного читателя в Приложении 10.1 мы приводим более формальное обоснование, 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 подставиль 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 нужно отправить с Северного склада. Результат мы снова приведем в таблице:

Оптимальный план поставки листового железа					
	Заказано	Юж. склад	Сев. 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 получилась 760 рублей, это намного меньше, чем 864 рублей – наше первое, выбранное наобум решение. Выгода получилась 12%, и это очень существенно, особенно когда таких доставок много каждый день.

То, что решение нужно искать “по углам”, сказано уже на самой первой странице в работе Канторовича. Это понятно любому математику.

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

Заслуга Данцига в том, что он придумал способ, осонованный 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, который применяется 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 находит оптимальное решение очень быстро.

3.7 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 наш самый первый пример, когда 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 огранивения и посложнее. Например: "задание 3 нужно (или желательно) выполнить раньше, чем задание 10."

> В Приложении 10.1.2 для подготовленного читателя мы более подробно рассмартриваем еще один маленький пример составления расписаний.

В реальности даже для относительно небольшого расписания имеет смысл воспользоваться математической моделью.

Например, несколько лет назад студенты прикладной математики университета Твенте составили 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 всего за несколько минут. Восхищенные певцы пришли в университет на презентацию проекта и спели студентам благодарственную арию!

3.8 Почему целые числа сложнее дробных

Как только нам нужно целочисленное решение, это 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 *метод ветвей и границ*, который основан на том, чтобы "разветвлять" дробные решения на допустимые целочисленные, и исключать неперспективные "ветки".

В Приложении 10.1.3 мы объясняем основные идеи этого метода чуть более подробно, для **интересующегося** читателя. Заметим, что это объяснение **не требует** математической подготовки.

Для практического применения, наивное использование метода ветвей и границ абсолютно не годится. Математики дополнили его многими другими методами. Например, сейчас на практике широко используется метод "рассекающих плоскостей" (алгоритм Гомори), который позволяет эффективно отсекать дробные значения.

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

3.9 Математика, обогнавшая компьютер

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

По словам Биксби, к 1953 году, после нескольких усовершенствований симплекс-метода, 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 большой калькулятор, который программировался с помощью перфокарточек. Решение заняло 8 часов, из которых бо́льшая часть времени ушла на то, чтобы вручную вставлять перфокарточки в машину. Настойчивости и терпению ученых остается только удивляться.

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

В компьютерных технологиях есть известное утверждение, что мощность процесоров удваивается каждые 18 месяцев. Это часто называют "законом Мура". И хотя это не закон природы, и Мур (Gordon E. Moore, один из основателей Intel) говорил не совсем об этом, все же утверждение примерно соответствует действительности.

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

В 2007 году Биксби провел впечатляющий эксперимент. Он взял все версии пакета *CPLEX*, 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 1892 из них. Дальше он сравнил скорость решения, от версии к версии, на одном и том же компьютере.

Оказалось, что за 15 лет скорость решения задач увеличилась в 29 000 раз! Интересно, что самое большое ускорение, почти в 10 раз, произошло в 1998 году. И это не

случайно. До этого в течение 30 лет математики разрабатывали новые теории и методы, из которых очень мало было внедрено в практику. В 1998 году в версии *CPLEX* 6.5 была поставлена задача реализовать по максимому 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 году ускорился в 16,2 раза. То есть общий эффект в 1991–2012 годах – в $29\,000 \times 16,2 = 469\,800$ раз! Повторимся, что это независимо от скорости компьютера, а только благодаря развитию математический идей.

Если верить закону Мура, то между 1992 и 2012 годом компьютеры ускорились примерно в 8000 раз. Сравните с почти полумилионным 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. Математика достигла гораздо бо́льшего прогресса, и никто даже не заметил! Многогранники, плоскости, двойственные задачи и разветвления зашиты в программных пакетах и решают задачи планирования, как будто так и было всегда, и в этом нет ничего особенного.

Конечно, самое поразительное – это совместный эффект математики и компьютеров. Ускорение в 4 миллиарда раз. Задачи, которые требовали 126 лет в 1991 году, в 2012-м мы научились решать за одну секунду! И это не предел. В статье 2015 года Дмитрис Бертсимас (Dimitris Bertsimas) и Анжела Кинг (Angela King) из Масачусетского Технологического Института приводят новую цифру – 450 миллиардов – и предлагают новые приложения линейного программирования в статистике.

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

3.10 Расписание голландских железных дорог

Самая престижная награда в исследовании операций – это премия Франца Эдельмана (Franz Edelman Award), за выдающиеся успехи науки в приложениях. В 2008 году эту награду получили Железные Дороги Нидерландов, за новое железнодорожное расписание, которое начало действовать в 2006 году.

Нидерланды – маленькая и густонаселенная страна. На территории размером примерно с Нижегородскую область проживает около 17 миллионов человек. Железные дороги – это основа всей голландской логистики. Многие каждый день ездят на поезде на работу. Совещание в другом конце страны – это обычное дело. Голландцы – чемпионы Европы по пассажирским железнодорожным перевозкам. В 2006 году Железные Дороги Нидерландов перевезили 15.8 миллиардов пассажиров.

До 2006 года действовало расписание, которое было составлено в 1970 году. Перевозки увеличивались, и постепенно поезда удлиннялись и, где возможно, добавлялись новые поезда. Пока наконец к началу 2000-х увеличивать перевозки в рамках старого расписания стало невозможно. Строить новые пути фактически тоже невозможно,

это безумно дорого, и просто недостаточно 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 имеющийся 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 отбытия 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 и была присуждена премия Эдельмана.

Каждый этап составления расписания требовал новой модели и новых подходов. Некоторые задачи (например, распределение путей прибытия и отбытия), 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 были бессильны. Пришлось придумывать новые модели и подходы.

Внедрение не сразу прошло гладко. И все-таки благодаря новому расписанию больше поездов перевозит больше людей по тем же рельсам. Пассажиропоток увеличивается, но расписание по-прежнему справляется. В рамках старого расписания это было невозможно.

3.11 Что такое оптимальное решение

Если вы недавно бывали в Нидерландах, то последний раздел вас мог удивить. Движение железных дорог далеко от совершентсва. Мелкие (и крупные) задержки случаются сплошь и рядом. Пересадки бывают очень короткие, их легко пропустить при малейшей задержке. Поезда часто переполнены, особенно вагоны наиболее популярного 2-го класса. Далеко не все были рады новому расписанию. Влиятельная голландская газета *NRC Handelsblad* написала:

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

Александер Схрайвер (Alexander Schrijver), 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, рассчитаной 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 это удастся, как взвесить эти факторы по отношению друг к другу?

Очень важно понимать, что *оптимальное* решение не означает решение *идеальное*. Оптимизация происходит при массе ограничений, и пожелания к решению противоречат друг другу. Например, максимальное количество пассажиров и дешевизна перевозок противоречит максимальному комфорту. Оптимальное решение – это лучшее, что мы можем сделать при заданных ограничения и заданных 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 увидят свет в значительных проектах. Как нам написал Схрайвер: “Это был масштабный проект, но за ним стояла очень интересная, чистая математика”. А свою статью он закончил словами: “Математика железных дорог пока далека от завершения.”

Глава 4

Мир нулей и единиц

4.1 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 невозможно передать напечатанные слова *“мама”*, *“небо”* и так далее. Можно передать только сигналы. Значит, каждой букве или слову нужно сопоставить свой сигнал.

Как перевести слова в сигналы? Если подумать, то на ум приходит что-то 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 с помощью всего двух символов: 0 и 1.

Понятно, что мы только для примера написали фразы по-русски. А вдруг нам завтра 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 витеиватых алфавитов Индии, и тысячи, если не десятки тысяч, китайских и японских иероглифов! Обучить компьютер отличать один язык от другого можно, 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 слов, которые употребляются часто, особенно если таких слов не слишком много.

Конечно, можно построить компьютеры, которые будут различать три или больше разных сигналов. Но *двоичная* система (два сигнала, два символа) намного удобнее. Ток либо есть (1), либо нет (0) – здесь машина не может ошибиться. Поэтому устройства с двоичной системой проще сделать, и они более надежные.

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 – это самый минимальный объем информации, который называется один *бит*. Восемь битов, то есть последовательность из восьми нулей и единиц, называется один *байт*. А всем известный *килобайт*, тот самый КБ, количеством которых определяется объем наших е-мейлов и документов, это 1024 байта, или 8192 нулей и единиц. 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. Совершенно аналогично, в нашей привычной *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 степени десять:

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

Многие счиатют, что килобайт – это тысяча байтов. Это, строго говоря, не так, но приблизительно это верно, и больше соответствует нашей привычке считать десятками, сотнями и тысячами.

Сколько килобайт в вашем сообщении? Это 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 единицами 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.

4.2 Что такое кодирование

Итак, наша задача — *закодировать* каждый символ или каждое слово текста с помощью нулей и единиц. Фактически, *код* – это словарь, который состоит из *кодовых слов*. Каждое кодовое слово – это последовательность нулей и единиц, желательно небольшой длины, которая что-то означает (буква, цифра, знак препинания или целое слово). Построить код можно самыми разными разными способами, а, значит, эта задача очень интересная.

Пусть, для примера, мы кодируем отдельно каждую букву русского алфавита. Забавно, кстати, что, когда ставишь эту задачу студентам, то почти всегда кто-нибудь спрашивает, сколько букв надо учитывать: 32 или 33. По-видимому, они считают букву “ё” не вполне самостоятельной буквой, которую в тексте можно заменить “е”. Давайте 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 будет достаточно, потому что мы можем каждую букву обозначить кодом из 32 нулей и одной единицы – на той позиции, на которой стоит эта буква в алфавитном порядке. Этот наивный код будет выглядеть

так:

а: 10000000000000000000000000000000
б: 01000000000000000000000000000000
в: 00100000000000000000000000000000
г: 00010000000000000000000000000000
и так далее
я: 00000000000000000000000000000001

Сразу видно, что такое кодирование не годится на практике. Слишком много драгоценных байтов уходит на нули, которые несут очень мало информации.

Какая *минимальная* длина кода нам понадобится, чтобы закодировать русский алфавит? Допустим, 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 так далее до 11111. Всего таких последовательностей 32. Получить этот ответ очень просто. Оказывается, последовательностей длины 5 не хватает, и вопрос студентов был в самую точку! Всего из-за одной "лишней" буквы, нам нужно, как минимум, 6 нулей и единиц в каждом "кодовом слове".

Дли интересующихся читателей в Приложении 10.2.1 мы приводим вычисления числа последовательностей из нулей и единиц заданной длины.

Интересно, что добавление всего одной позиции кода очень сильно меняет дело. Для русского алфавита нам нужны последовательности длины 6, а их уже 64. Значит, последовательностей длины 6 нам хватит не только на русский алфвит, но можно добавить еще и латинский алфавит из 26 букв, и в запасе еще останется 5 свободных последовательностей для знаков припенания.

Самое главное наблюдение – это что добавление всего одной позиции кода увеличивыет количество разных последовательностей вдвое. Потому что лишнюю позицию можно заполнить двумя способами – либо нулем, либо единицей. В результате количество букв, слов или сообщений, которые мы можем закодировать, возрастает с длинной кода по так называемому *экспоненциальному* закону, как степень двойки.

"Растет по экспоненциальному закону" на общедоступном языке означает "растет очень-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 зернышка, на третью четыре, и так далее – в два раза больше на каждую следующую клетку. Король согласился, и был потрясен, когда всего 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, и так далее.

Экспоненциальная зависимость между количеством разных кодовых слов и их длинной – это абсолютно фундаментальная концепция в информатике и проблемах передачи информации.

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

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

4.3 Коды, исправляющие ошибки

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

Представим себе, что мы начали передвать наши кодовые слова по каналу связи. В канале иногда возникают помехи. В результате каждой 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, сделать это можно, надо лишь правильно сформулировать соответствующую математическую задачу.

Допустим, наши кодовые слова длины 6, на каждое кодовое слово приходится не более одной ошибки. Поскольку это простой пример, представим себе, что наш словарный запас хуже, 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 этого достаточно. Еще важно, что получатель знает наш “словарь”, то есть ожидает от нас либо 111000, либо 001110, либо 100011, и ничего другого.

Предположим, что мы сначала передаем слово 111000. В результате не более чем одной ошибки (ошибки мы выделили красным цветом), оно может превратиться в

одно из слов

$$111000, \mathbf{0}11000, 1\mathbf{0}1000, 11\mathbf{0}000, 111\mathbf{1}00, 1110\mathbf{1}0, 11100\mathbf{1}, \tag{4.1}$$

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

$$001110, \mathbf{1}01110, 0\mathbf{1}1110, 00\mathbf{0}110, 001\mathbf{0}10, 0011\mathbf{0}0, 001111. \tag{4.2}$$

Наконец, для 100011 у нас получится:

$$100011, \mathbf{0}00011, 1\mathbf{1}0011, 10\mathbf{1}011, 100\mathbf{1}11, 1000\mathbf{1}1, 10001\mathbf{1}. \tag{4.3}$$

Замечательно то, что списки (4.1)–(4.3) попарно не пересекаются. Иными словами, если на другом конце канала связи появляется любое слово из списка (4.1), то получатель точно знает, что ему передавали именно слово 111000, а если появляется любое слово из списка (4.2), то передавалось слово 001110, и то же самое и со списком (4.3). В этом случае говорят, что наш код *исправил одну ошибку*.

За счет чего получилось исправление? За счет двух факторов. Во-первых, получатель знал весь "словарь". Когда код передавался всего с одной ошибкой, то получалось слово, которое в словарь не входит.

Во-вторых, слова в словаре были выбраны особенным образом. Даже если появлялась ошибка, получатель не мог перепутать одно слово с другим. Например, если наш словарь состоит из слов *"дочка"*, *"точка"*, *"кочка"*, и при передаче получилось *"вочка"*, то получатель знает, что слова *"вочка"* не бывает, но исправить ошибку он не в состоянии – любое из трех слов могло быть правильным. А если взять словарь *"точка"*, *"галка"*, *"ветка"*, и мы знаем, что допускается не больше одной ошибки, то *"вочка"* это заведомо *"точка"*, а не *"галка"*. В кодах, исправляющих ошибки, слова выбираются именно так, чтобы их можно было "узнать" даже после ошибки. Разница только в том, что в кодовом "алфавите" всего две буквы – ноль и единица.

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

4.4 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. Это расстояние обычно обозначается буквой r – радиус. В нашем привычном трехмерном пространстве, где расстояния измеряются в метрах, это всем привычный шар, как на Рисунке 4.1.

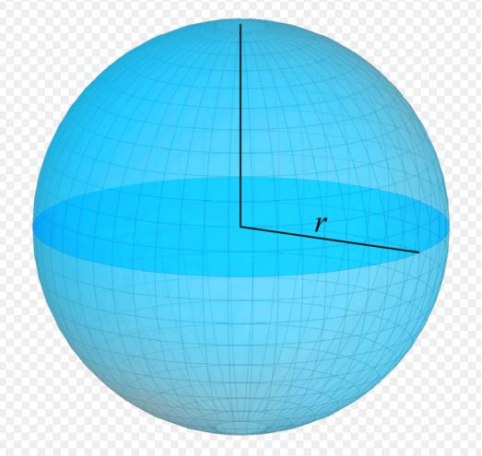

Рис. 4.1: Шар радиуса r в трехмерном пространстве. Все точки удалены от центра не больше, чем на расстояние r. Источник: `https://en.wikipedia.org/wiki/Sphere`.

Зато понятия *точка* и *расстояние* в математике абсолютно абстрактные. Есть несколько простых правил, которые должны всегда выполняться, но в рамках этих правил – полная свобода. Точками могут быть сигналы, а могут быть кривые, и даже результаты случайных экспериментов. И расстояния между ними можно определить самыми разными способами. В результате получаются "шары", которые практически невозможно себе представить. 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 Рисунке 4.2 расстояние Хэмминга между двумя кодовыми словами равно 3. Мы выделили в красные рамки те позиции, где эти два кодовые слова отличаются друг от друга.

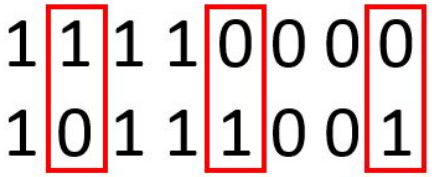

Рис. 4.2: Расстояние Хэмминга между двумя кодовыми словами – число позиций, на которых у этих слов стоят разные символы. На рисунке это расстояние равно три. Позиции, где два кодовых слова отличаются друг от друга выделены в красные рамки.

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

Получится другое слово, которое будет отличаться от 111000 всего на одной позиции. Получается, что если у нас при передаче происходит не больше одной ошибки, то расстояние Хэмминга между отправленным и полученным кодовым словом будет не больше, чем один. Давайте снова посмотрим на список (4.1) предыдущего раздела:

$$111000, 011000, 101000, 110000, 111100, 111010, 111001.$$

Расстояние Хэмминга между словом 111000 и любым другим словом из этого списка 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 язывa одинаковой длины. Например, расстояние Хэмминга между словами *"дочка"* и *"точка"* равно единице, а между словами *"точка"* и *"галка"* расстояние Хэмминга равно трем. Если слово *"точка"* – это центр шара Хэмминга, то слово *"дочка"* входит в этот шар, а слово *"галка"* – нет.

Шары Хэмминга очень трудно себе представить даже для маленьких кодов. На Рисунке 4.3 мы изобразили расстояния Хэмминга между кодовыми словами длины 3. Расстояние Хэмминга равное нулу обозначено белым цветом, один – красным, два – оранжевым, и три – желтым. Если взять одну из последовательностей за центр

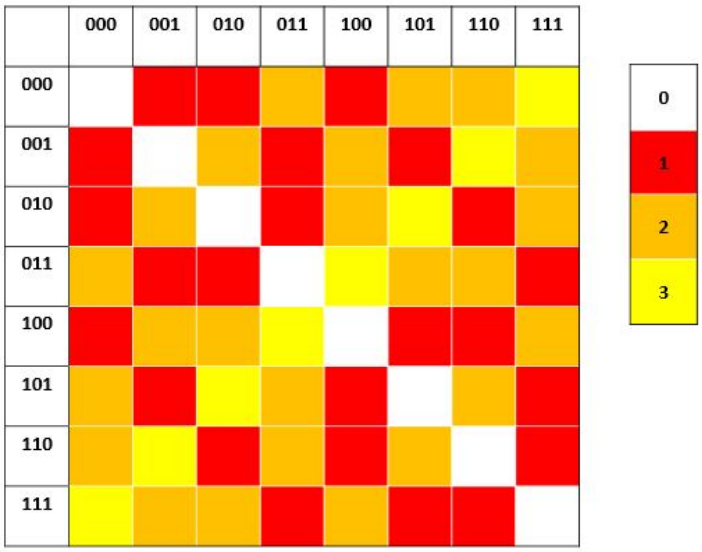

Рис. 4.3: Расстояние Хэмминга между кодовыми длины три. Справа изображен цветовой код. Расстояние Хэмминга: 0=белый, 1=красный, 2=оранжевый, 3=желтый.

и посмотореть на соответствующею колонку, то шар Хэмминга радиуса один – это все квадратики, которые обозначены белым и красным. Например, для слова 000 это слова 000, 001, 010, 100. Сразу видно, что расположение белых и красных квадратиков в колонках неодинаковое, хотя конечно в рисунке много закономерностей. Например, бело-красный рисунок абсолютно симметричен оранжево-желтому.

Кодовые слова длиной 3 это 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. Для примера, на Рисунке 4.4(а) изображены расстояния Хэмминга между кодовыми словами длины 8, а это всего один байт. Таких слов 256. Этот рисунок мы взяли с вебстраницы профессора Джона К. Слэни из Австралийского Национального Университета. На Рисунке 4.4(б) мы увеличили центральный квадрат (обозначенный на рисунке (а) черной рамочкой), чтобы показать, что мы по-прежнему видим ту же симметрию и те же структуры, что и в кодовых словах длины три. Шары Хэмминга, например, радиуса два, можно найти

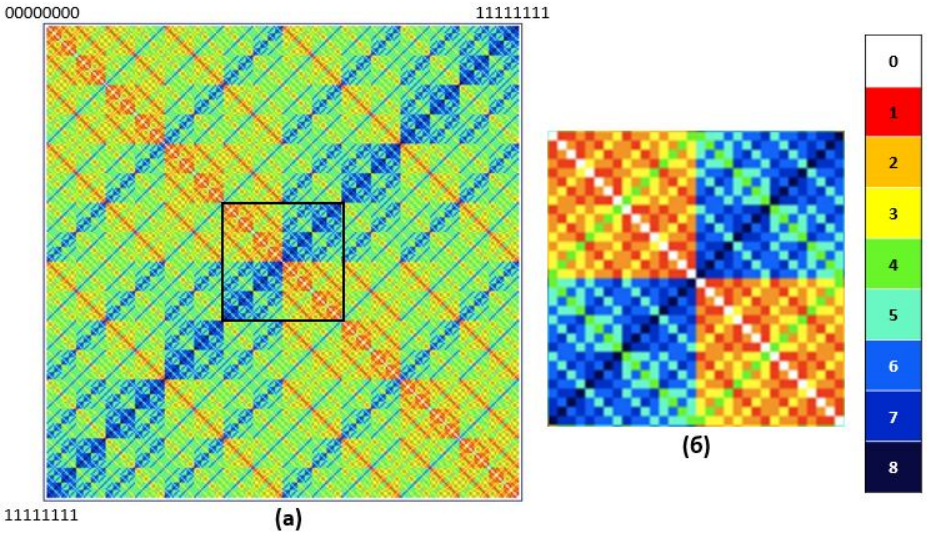

Рис. 4.4: (а) Расстояние Хэмминга между кодовыми длины восемь. Справа изображен цветовой код. Расстояние Хэмминга: 0=белый, 1=красный, 2=оранжевый, 3=желтый, 4=зеленый, 5=светло-голубой, 6=светло-синий, 7=синий, 8=темно-синий. (б) Центральный квадрат, выделенный на рисунке (а) черной рамкой. Создатель рисунка: Джон К. Слейни (John K. Slaney). Источник `http://users.cecs.anu.edu.au/~jks/Hamming.html`.

на Рисунке 4.4(а), если взять одну вертикальную колонку и отметить все крошечные квадратики, закрашенные белым, красным и оранжвым цветом.

Понятно, что такая картинка никак не поможет нам найти хороший код. Она скорее похожа на 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.

Возможно, вы уже увидели связь между шарами Хэмминга и кодами, исправляющими ошибки. Допустим, мы передаем кодовые слова длины 10 и хотим, чтобы код исправлял 2 ошибки. Тогда надо построить такой код, чтобы шары с центрами в кодовых словах и радиусами 2 попарно не пересекались. Все последовательности нулей и единиц в таком шаре будут означать одно и то же кодовое слово. Выража-

ясь по-другому, кодовые слова должны отличаться друг от друга достаточно, чтобы при наличие двух ошибок их невозможно было перепутать. На языке математики это означает, что расстояния Хэмминга между кодовыми словами должны быть как минимум 5.

При построении кодов возникает большое количество интересных вопросов. Например, очень важно, чтобы количество кодовых слов было как можно б*о*льшим: чем оно больше, тем больше информации мы сможем передать по каналу связи, 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, хоть и мотивирована она совершенно практическими вопросами кодирования информации.

4.5 История кодов, исправляющих ошибки

Одним из основателей современной теории кодирования был Ричард Хэмминг (Richard Hamming), именем которого названы расстояния между кодовыми словами. Это был конец 40-х годов XX века, 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 растоянии 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, чтобы исправлять ошибки при передаче буквы *"а"*. Такой шар содержит 56 последовательностей. Получается, что на каждое кодовое слово, которое что-то означает, приходится 56 слов, чтобы исправить ошибки. Поэтому мы можем закодировать не 1024, а всего лишь 1024/56, то есть 18 разных символов или сообщений. Если этого мало, например, если мы хотим закодировать русский алфавит, то длину кодовых слов придется увеличить. Это увеличит количество килобайтов, но такова цена за исправление ошибок.

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

> Для подготовленного читателя в Приложении 10.2.2 мы приводим точную формулу границы Хэмминга и ее доказательство в общем случае.

Заметим, граница Хэмминга – это *максимально* возможное число слов в коде, исправляющем ошибки. Этот еще не означает, что такой максимальный код можно найти. Представьте себе, что мы имеем дело с обычными шарами. Невозможно 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, чтобы эти шары не пересекались, и между ними не было свободного места? Оказывается, такое разбиение во многих случаях действительно возможно. Мы не станем здесь описывать конструкцию, ее можно найти в более специальной литературе (см. [1]).

Впрочем, сделано в этом направлении далеко не все. Конструкции есть, но только *асимптотические*, то есть когда длина кодового слова очень большая и увеличивается до бесконечности, а количество ошибок при это маленькое и фиксированное. Однако во многих практических задачах чем длиннее кодовое слово, тем больше ошибок. В этой 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, принадлежащая замечательному российскому математику Владимиру Иосифовичу Левенштейну, и так называемые границы *линейного программирования*. Эти конструкции очень сложны и выходят далеко за рамки нашей книги.

Особый интерес вызывают так называемые *равновесные* коды. В этих кодах все кодовые слова содержат одинаковое количество единиц. Для таких кодов можно найти границу, аналогичную границе Хэмминга. И снова встает вопрос о существовании равновесного кода, достигающего полученную границу. В начале 80-х годов XX века Войцех Рёдль (Vojtěch Rödl)придумал очень хитрую вероятностную технику, позволившую доказать существование равновесных кодов, в которых число кодовых слов асимптотически равно верхней границе (см. [2]).

Только пару лет назад Питер Киваш (Peter Keevash) из Оксфордского универтситета объявил о построении равновесных кодов, в которых количество слов достигает теоретическую верхнюю границу. Это очень объемная и трудная математическая работа, которая до сих пор тщательно проверяется и потому опубликована только в Интернете (см. [19]).

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

4.6 Можем ли мы закодировать все подряд

Все, что мы написали в этой главе в основном применимо к кодированию текстов. С другими видами информации возникают свои особенности.

Как, например, закодировать цвет? Скорее всего, вы не раз отправляли, получали или по крайней мере видели в Интернете цветные фотографии. Значит, закодировать

цвет можно, и люди уже научились это делать. На самом деле, задача эта очень нетривиальная, и решить ее получилось только потому, что, оказывается, любой цвет можно разбить на три компонента: *красный*, *зеленый* и *синий*. Любой оттенок определяется всего лишь интенсивностью этих трех основных цветов. Так устроен наш цветной мир, такой вот подарок для кодирования. Если бы природа цвета была другая, то никаких цветных фотографий нам бы переслать не удалось.

Если вам когда-нибудь приходилось использовать нестрандартные цвета в обычных программах типа *Word* или *PowerPoint*, то вы возможно заметили эту опцию, где интенсивность красного, зеленого и синего можно задать вручную. Обычно эта интенсивность определяется числом между 0 и 255. Всего 256 вариантов – ровно по количеству разных последовательностей из нулей и единиц длины восемь, то есть один байт. В результате нам нужно всего три байта, чтобы закодировать $256 \cdot 256 \cdot 256 = 16\,777\,216$ – более шестнадцати с половиной миллионов оттенков компьютерной палитры.

Если обозначать интенсивность цветов цифрами того же цвета, то **0**-**0**-**0** – это черный цвет, **255**-**255**-**255** – это белый. Если интенсивность красного, зеленого и синего одинаковая, то между черным и белым получится целых 254 оттенка серого. А желтый можно получить, если взять с одинаковой интенсивностью красный и зеленый. На Рисунке 4.5 мы изобразили пример красно-зелено-синего кода для цветов радуги.

Цвет	Интенсивность основных цветов		
	красный	зеленый	синий
	255	0	0
	255	128	0
	255	255	0
	0	255	0
	0	255	255
	0	0	255
	255	0	255

Рис. 4.5: Пример цветового кода для цветов радуги. Интенсивность основных цветов – красного, зеленого и синего – определяется числом от 0 до 255.

Дополнительные очень серьезные проблемы возникают, когда нужно передать фильм. Если кодировать каждую точку каждого кадра, то понадобися такое количество гигабайтов, что это не уместится в пямяти ни одного компьютера. Поэтому цифровое видео появилось относительно недавно.

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

Стандартный способ сжатия данных для фильма – это кодировать полностью первый кадр, а затем кодировать не каждый кадр отдельно, а только изменения. Но если нам нужно сохранить информацию другого рода, например, кто с кем дружит на *Фейсбук*, то требуются совершенно другие способы сжатия данных. И они, кстати, тоже уже разработаны.

А вот со звуком все намного сложнее. Мы до сих пор не умеем кодировать звучание симфонического оркестра так, чтобы его трансляция звучала 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 те 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 складывается. Но, к сожалению, сделать это исчерпывающим образом пока никому не удалось.

Глава 5

Надежность Интернета

5.1 Связанные одной сетью

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

Сигнал идет со скоростью света, поэтому совершенно неважно, где находятся серверы – в России, в США или Австралии. Пройденные расстояния практически не влияют на скорость передачи. Мы все уже давно привыкли, что е-мейлы и ватсап доходят в считанные секунды, веб-страницы грузятся быстро, а наш голос и даже изображение передается по Скайпу в реальном времени.

Но если задуматься, то где гарантия, что в любой момент времени 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 централизированным образом, их план прошел множества 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 собстенную сеть, присоединять новые сервера и компьютеры. Возникают локальные сети, они соединятюся 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 означает *сеть*.

Ни один человек и ни одна компания в мире не отвечает за то, чтобы сервер, через который вы присоединились к Интернету, был связан с другим сервером на острове

Кенгуру.[1] Но вся система по своей природе устроена так, что связь гарантирована. Интернет – гигансткая международная технологическая и коммерческая конструкция, без которой мы уже не можем жить, – прекрасно обходится без правления и правительста. Если задуматься, то это просто поразительно!

Еще поразительнее, что связь практически никогда не теряется, хотя в каналах связи случаются неполадки, и абсолютно регулярные перегрузки. Может ли Интернет, хотя бы временно, "развалиться на кусочки"? Может ли случиться, что из-за неполадок 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, что если ваш сервер и сервер получателя исправен, то информация всегда проходит через сеть безо всяких проблем.

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

5.2 Сети и помехи

Начнем с простого примера. Допустим, наш Интернет состоит всего из трех компьютеров, которые соеденины друг с другом, как на Рисунке 5.1. Если все три канала

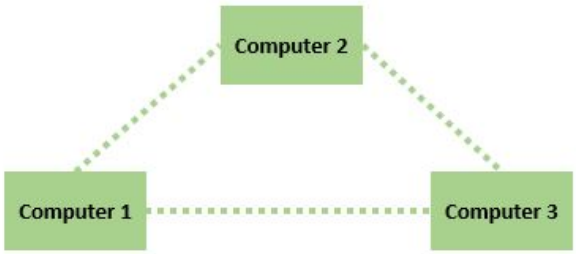

Рис. 5.1: Мини-Интернет из трех компьютеров, соединенных каналами связи. Все три канала работают, все три компьютера могут обмениваться информацией.

связи работают, то никаких проблем, все три компьютера могут обмениваться информацией.

Теперь допустим, в одном из каналов связи возникли помехи, и передать по нему в данный момент ничего нельзя. Мы изобразили эту ситуацию на Рисунке 5.2. Сразу видно, что наш мини-Интернет не распался. Если прямая связь между Компьютерами 1 и 2 потеряна, то они по-прежнему могут передать друг другу информацию через Компьютер 3. Заметит ли пользователь неполадку в канале? Скорее всего, нет. Поскольку сигнал идет со скоростью света, то нет никакой разницы в скорости доставки

[1] Такой остров правда есть, население острова 4,5 тысячи человек, кенгуру там гораздо больше!

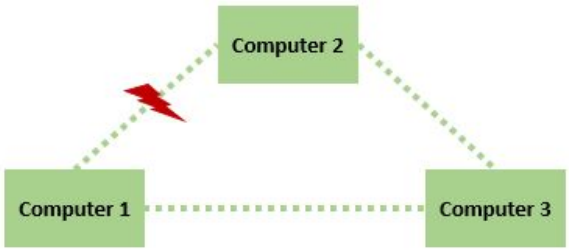

Рис. 5.2: Мини-Интернет из трех компьютеров. Канал связи между Компьютерами 1 и 2 недоступен, но при этом Компьютеры 1 и 2 по-прежнему могут обменивать информацией через Компьютер 3.

информации — пойдет ли сигнал напрямую из Москвы в Нижний Новгород, или даст круголя через Сидней или Нью-Йорк.

Чтобы развалить нашу маленькую сеть, нужно вывести из строя как минимум два, или все три канала связи, как на Рисунке 5.3.

Рис. 5.3: Мини-Интернет из трех компьютеров. Слева: вышли из строя два канала связи, Компьютер 1 оказался отвезанным от сети. Справа: вышли из строя все три канала связи, связь между компьютерами полностью потеряна.

Насколько устойчива наша мини-сеть? Это сосчитать совем нетрудно. Допустим, что 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 это многовато для реального Интернета, но для примера подойдет.

Компьютер 1 может оказаться отвезанным от сети как на Рисунке 5.3 слева с вероятностью $0.4 \cdot 0.4 \cdot 0.6 = 0.096(\times 100\%) = 9.6\%$. В аналогичной ситуации могут оказаться Компьютеры 2 и 3, ровно с той же вероятностью. Наконец, надо добавить вероятность самой плохой ситуации, на Рисунке 5.3, которая равна $0.4 \cdot 0.4 \cdot 0.4 = 0.064(\times 100\%) = 6.4\%$. В результате получается, что наша сеть “развалится” с вероятностью $3 \cdot 9.6\% + 6.4\% = 35.2\%$.

Конечно, 35.2% это 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 оказывается более устойчивой, чем отдельно взятый канал!

Даже на нашем мини-примере видно, откуда берется устойчивость сети. В сети компьютеры могут связаться друг с другом не одним, а несколькими способами, через другие компьютеры. Если один канал недоступен, то можно найти альтернативный маршрут. Более того, этот эффект заметно усиливается при меньшей вероятности помех. Для примера, мы приводим несколько результатов в Таблице 5.1.

> В Приложении 10.3.1 для подготовленного читателя мы приводим формулы, которыми мы воспользовались в Таблице 5.1, в общем виде.

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%

Таблица 5.1: Вероятности потери связи в мини-сети (правая колонка) при заданной вероятности потери связи в одном канале (левая колонка).

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!

Кончено, 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 нам опять необходима 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, которые проникают в самую суть устройства сетей и уже стали классикой, мы расскажем в следующих разделах.

5.3 Случайные графы

Математическая теория, которая, в частности, позволяет ответить на вопрос об устойчивости больших сетей, возникла на рубеже 50-х и 60-х годов XX века. Ее основателями стали два замечательных венгерских математика Пол Эрдеш (Paul Erdős) и Альфред Реньи (Alfréd Rényi) в статьях [22, 15].

Эрдеш — это настоящий классик современной комбинаторики, теории чисел, теории вероятностей. Ему принадлежат более *полутора тысяч* статей, множество решенных проблем и еще больше постановок задач, которые определили развитие 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 В.А. Стеклова.

Теория, которую заложили Эрдеш и Реньи, называется *теорией случайных графов*. А математическая модель, рассмотренная в их первых работах, называется *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 параграфа очень легко избразить в виде графа. Мы это сделали на Рисунке 5.4 слева. Сеть изображена в виде трех *узлов* (компьютеров), которые соеденины линиями (каналами связи). В математике узлы называются *вершинами* графа, а линии между ними – *ребрами*. Чтобы не затруднять восприятия 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 8 мы вернемся к графам, когда будем обсуждать социальную сеть. В этом случае узлы – это люди, а линии между ними – это дружбы в социальной сети. Мы изобразили пример графа из Главы 8 на Рисунке 5.4 справа.

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

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

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. Если есть помехи, то канал связи становится недоступным, линия между узлами исчезает.

Случайный граф – естественная модель во многих ситуациях. Например, дружбы в социальных сетях возникают непредсказуемым образом. В телекоммуникациях или электрических сетях на линиях связи могут случаться помехи. Если попытаться смоделировать нейронную сеть мозга, то взаимодействия нейронов можно выявить только с определенной вероятностью.

В последнее время, в связи с Интернетом, социальными сетями, и небывалой до-

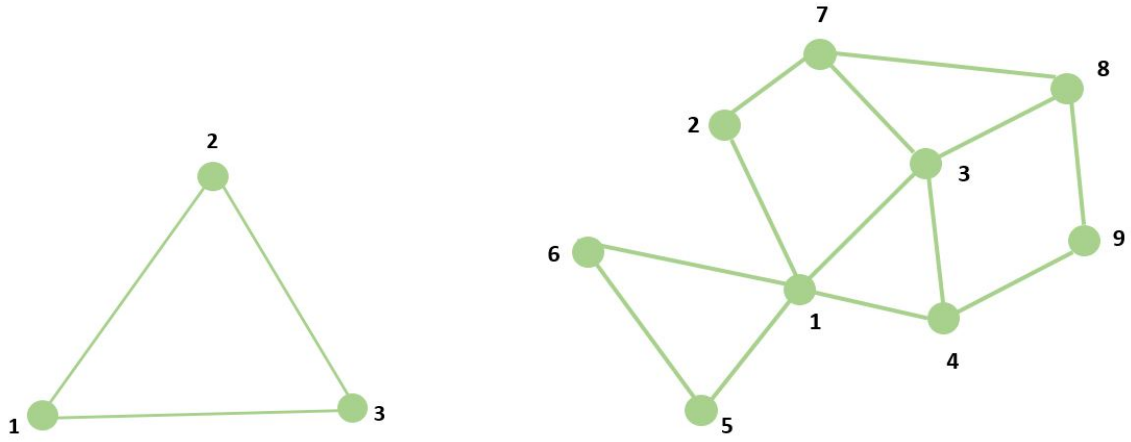

Рис. 5.4: Слева: мини-сеть в виде графа. Узлы – это компьютеры, а линии – это каналы связи. Справа: социальная сеть из Главы 8 в виде графа. Узлы – это люди, а линии – это дружбы в социальной сети.

ступностью данных во всех областях – от энергоснабжения до биологии – интерес к теории случайных графов особенно возрос. Новые, очень сложные, результаты появляются почти каждый день.

Что же говорит теория случайных графов об устойчивости сети?

5.4 Результат Эрдеша-Реньи

В связи с устойчивостью Интернета, нас интересует вопрос о *связности* случайного графа. Граф называется *связным*, если между любыми двумя его вершинами можно пройти по цепочке ребер, то есть все узлы связаны друг с другом. Оба графа на Рисунке 5.4 – связные. На Рисунке 5.5 мы сделали оба эти графа *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 если вероятность помех очень велика.

Например, возмем сеть из 100 компьютеров, связанных между собой. Тогда оказывается, что каждый отдельный канал может быть недоступен с вероятностью аж 86%, и тем не менее сеть останется связной с вероятностью как минимум... 99%! Мы изобразили эту ситуацию на Рисунке 5.6: 86% из всех возможных линий отсутствует, и тем не менее сразу видно, что из любого узла можно добраться до любого другого.

А если взять сеть из 1000 узлов, то и вовсе что-то фантастическое. Канал связи может быть недоступен с вероятностью 98% , а связность сохраняется с вероятностью 99.9%! Чем больше сеть, тем сильнее результат.

В Таблице 5.4 мы приводим результаты для сетей разных размеров. Легко заме-

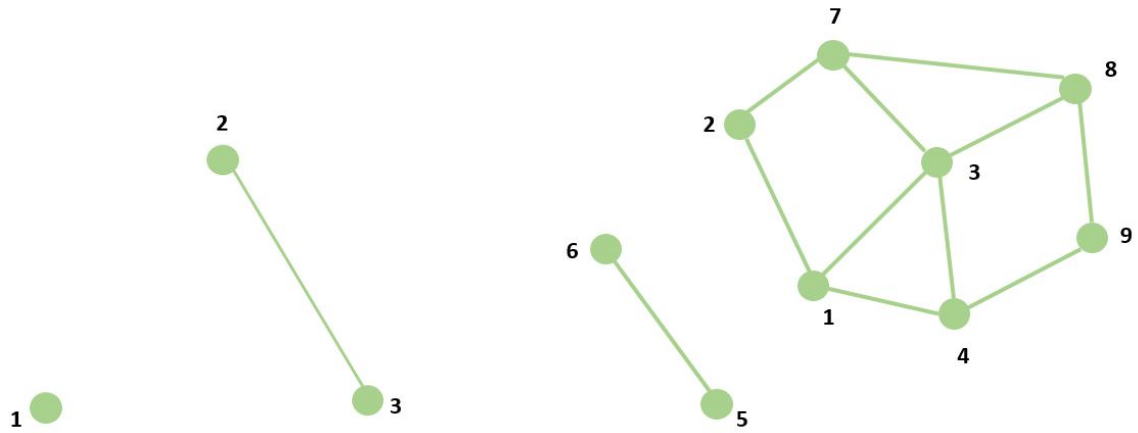

Рис. 5.5: Слева: мини-сеть в виде графа; каналы 1-2 и 1-3 недоступны; граф несвязный, из вершины 1 нельзя попасть в вершины 2 и 3. Справа: социальная сеть в виде графа; пользователь 1 не знаком с пользователями 5 и 6; граф несвязный; нет цепочки знакомых между пользователями 5, 6 и остальными пользователями.

тить, что число в самой правой колонке не что иное как

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

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%

Таблица 5.2: Результат Эрдеша-Реньи.

Ниже в серой рамке мы приводим более общую математическую формулировку результата. Этот текст рассчитан на уровень средней школы, но если вы не хотите вдаваться в подробности, то его можно пропустить.

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

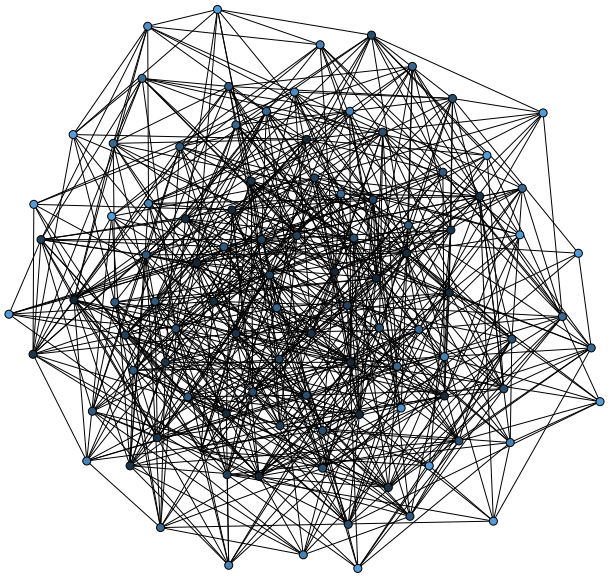

Рис. 5.6: Сеть из 100 компьютеров в виде графа. Вероятность недоступности канала 86%. Рисунок сделан в интерактивном режиме на вебсайте Гравитационной Программы *NETWORKS* Голландской Научной Организации (NWO), `http://www.networkpages.nl`. Создатель рисунка: Роберт Фицнер (Robert Fitzner).

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

$$q(n) \leq 1 - \frac{3 \ln n}{n}, \tag{5.1}$$

то связность сети сохраняется с вероятностью, не меньшей, чем

$$1 - \frac{1}{n}. \tag{5.2}$$

В Таблице 5.1 все значения умножены на 100%. Во второй колонке выписаны значения, полученные с помощью формулы (5.1). Поскольку $ln(n)$ растет гораздо медленнее, чем n, то допустимая вероятность помех увеличивается. В третьей колонке – значения (5.2).

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

Надежность сети — это и есть, по сути, 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.

5.5 Фазовый переход

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

Самый знаменитый фазовый переход – это изменение состояний воды в зависимости от температуры. При нуле градусах Цельсия вода превращается в лед, а при ста градусах – в пар. Ноль градусов и сто градусов – это *критические значения*, при которых состояние резко изменяется.

Очень похожее явление происходит с вероятностью связности сети, если изменять вероятность недоступности каналов. Оказывается, надежность сети изменяется не постепенно, а очень резко. Если вероятность помехи меньше критического значения, то с подавляющей вероятностью связность сохраняется. Но стоит хоть немножко пересечь критическую черту – и сеть почти наверняка распадется.

На Рисунке 5.7 мы привели пример сети размера 100. Согласно результатам Эрдеша-Реньи, критическая вероятность помех в такой сети равна 95.4%. Слева мы использовали вероятность помехи 95%, то есть меньше критического значения. Как видите, связность сети сохранилась. Мы повторили эксперимент много раз, 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!

Если взять ровно критическую вероятность помех, то будет примерно то же самое, что и с водой и снегом при нуле градусов: может получиться и так, и так. В нашем случае вероятность сохранения связности будет примерно 36.79%.

Критическая вероятность недоступности канала для сети размера n вычисляется по формуле $(1 - \ln(n)/n) \times 100\%$. Для примера, мы приводим несколько значений в Таблице 5.5.

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%

Таблица 5.3: Результат Эрдеша-Реньи: критическая вероятность помех. Если вероятность помех меньше критической, то связность сети сохраняется, а если больше – то разрушается.

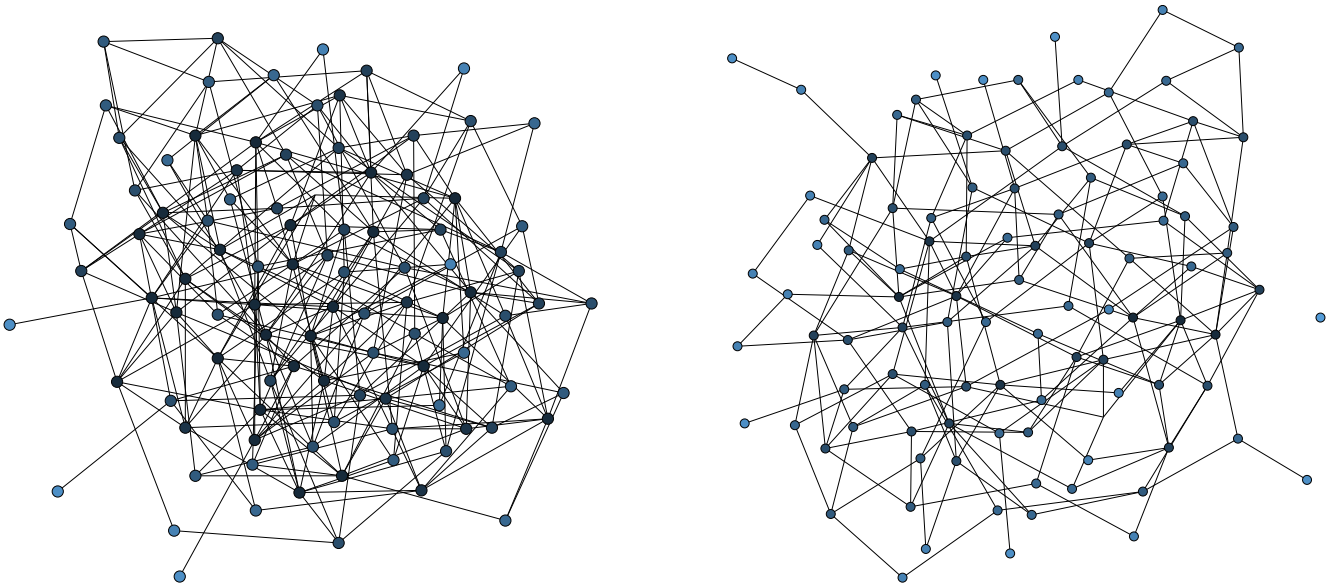

Рис. 5.7: Сеть из 100 компьютеров в виде графа. Слева: недоступности канала 95%; связность сохраняется. Справа: недоступности канала 96%; связность нарушена. Рисунок сделан в интерактивном режиме на вебсайте Гравитационной Программы *NETOWKRS* Голландской Научной Организации (NWO), `http://www.networkpages.nl`. Создатель рисунка: Роберт Фицнер.

В Приложении 10.3.2 для подготовленного читателя мы приводим математическую формулировку теоремы Эрдеша-Реньи о фазовом переходе.

Подобные фазовые переходы типичны для теории случайных графов. Эти результаты самые интересные, потому что они много говорят о природе сетей на практике. Состояние сети – это обычно две крайности. Социальная сеть достигает большой популярности или умирает. Компьютерный вирус распространяется с огромной скоростью и размахом, или сходит на нет в самом начале. И реальность, и математика, говорят об одном и том же: среднего не дано. К сожалению, далеко не всегда нам известны критические значения, а главное, каким образом удержаться по правильную сторону от фазового перехода.

5.6 Как доказывается результат Эрдеша-Реньи

Глубокие математические доказательства часто строятся на очень простых интуитивных идеях. Результат Эрдеша-Реньи – блестящий пример этой закономерности.

Математики заметили, что наиболее вероятный способ разрушить связность сети – это отрезать один узел от всех каналов связи. Группу узлов отрезать гораздо труднее, потому что число каналов, которые связывают группу узлов и остальную часть сети

– относительно большое. Маловероятно, что все эти каналы недоступны.

Тогда изначальный 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, нужны 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

$$(0.96)^{99}(\times 100\%).$$

Это очень специфическое выражение: число, близкое к единице, возведенное в большую степень. Такие выражения хорошо известны в математике, они относятся к так называемым "замечательным пределам", из которых, по сути дела, и следует результат.

> В Приложении 10.3.3 для подготовленного читателя мы приводим идею доказательства в более точной математической формулировке. Подробно это доказательство приводится в книге Андрея "Модели случайных графов" [4].

5.7 Что мы знаем и не знаем о надежности Интернета

Результаты Эрдеша-Реньи не рашают полностью проблемы устойчивости Интернета. Их модель не очень похожа на Интернет в реальности. Например, в модели Эрдеша-Реньи число линий у разных узлов обычно близко к среднему. В Интернете разброс между серверами очень большой. У некоторых 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 числе которых очень влиятельный и знаменитый ученый Ласло Барабаши (László Barabási), взяли данные небольшой части Интернета и с помощью компьютерных экспериментов посмотрели, что получится, если начать 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 целевым образом 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 хрупкий. Он устойчив к помехам, но чувствителен к атакам!

Эта работа быстро приобрела широкую известность. Только инженеры телекома с недоумением пожимали плечами: “Не может быть, наши сети очень надежные!”. Результаты Барабаши и соавторов получили много критики, особенно в 2005 году в статье специалистов по телекоммуникациям [12].

Один из главных пунктов критики в том, что в Интернете ключевые серверы не те, у которых много каналов, а те, через которые проходит наибольшее количество информационого трафика. Посмотрите, например, на представительный “кусочек Интернета” на Рисунке 5.8. Как мы видим, у некоторых серверов действительно много

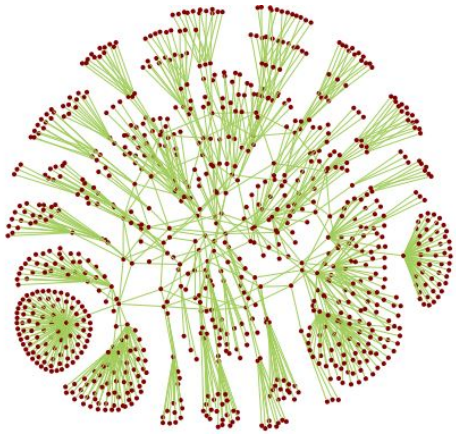

Рис. 5.8: Типичный граф сетей Интернета. Рисунок взят из статьи Прии Махадеван (Priya Mahadevan) и соавторов [21].

каналов связи, но только эти серверы находятся на переферии. За трафик в основном отвечает плотная сеть узлов посередине – опорная сеть. Интуитивно понятно, что нужно вывести из строя большое количество серверов, чтобы разрушить эту густую сеть маршрутов. 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, которая включает в себя основные свойства Интернета. И для этой модели нам нужны результаты типа Эрдеша-Реньи, которые говорят о том, что будет, если выввести из строя те или иные серверы или каналы связи. Только тогда у нас будут однозначно правильные, строго доказанные результаты. Продвижения в этих направлениях есть, но до точных ответов пока далеко. Работа продолжается.

5.8 Интернет в картинках

Если вы хотите посмотреть, как выглядит Интернет, то зайдите на вебсайт проекта *Opte*.

`http://www.opte.org/`

Создатель проекта, программист и художник Баррет Лион (Barrett Lyon) задался невыполнимой целью: нарисовать Интернет! Нарисовать 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 между ними плотные белые почти светящиеся магистрали – опорная сеть. Ксати, вы заметите, что структура Интернета действительно очень напоминает схематический рисунок 5.8 из предыдущего параграфа.

Рисунки *Opte* известны по всему миру. Они получили множество наград и выставляются в Музее Современного Искусства в Нью Йорке. Смотрите, это Интернет!

5.9 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 соавторов. Рассказывают, что он появлялся на пороге, и говорил: *My brain is open*|[2]. Дальше 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 (Fan Chung) 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 удобств 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 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 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.

[2]В русском переводе: Мой мозг открыт для вас

Глава 6

Сила выбора из двух

6.1 Очереди, которые 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 уж совсем непереносимые очереди из машин – автомобильные пробки. Но кроме всех этих "живых" очередей, мы, сами того не замечая, участвуем в огромном количестве "компьютерных" очередей. Это происходит практически каждый раз, когда мы подключаемся к Интернету.

Допустим, вы хотите посмотреть любую веб-страницу, скажем, главную страницу Московского Физико-Технического Института (МФТИ), где работает Андрей. Вы вводите веб-адрес `https://mipt.ru` в своем веб-браузере (например, *Internet Explorer*, *Fire Fox*, или *Google Chrome*). Что происходит дальше?

Веб-страница – это всего лишь файл, похожий на обычный документ в Ворде, только в несколько другом формате. Этот файл включает в себя все, что есть на странице: тексты, рисунки, линки, и так далее. Ваш браузер – это фактически интерфейс, который позволяет вам запросить файл с содержанием нужной вам страницы.

Отправка цифровой информации сравнима в отправкой обычной почты или грузов. Запрос с вашего веб-браузера поступает на так называемый *веб-сервер* под именем `mipt.ru`, это веб-сервер МФТИ. Веб-сервер – это специально выделенный компьютер, который отвечает за то, чтобы найти нужный файл и отправить его на браузер пользователя. У каждого сайта, или *домена* – `mipt.ru`, `rzd.ru`, `google.com`, и так далее – есть свой веб-сервер, и часто не один.

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

И это еще не все. Как и почтовая пересылка, информация попадает с одного ком-

пьютера 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 е-мейл, фотографии на Фейсбуке или ваш голос по Скайпу.

Насколько длинной будет очередь? Можно ли огранизовать сервис – например, отправку веб-страниц или передачу голоса и видео – чтобы задержки были как можно меньше? Этими вопросами занимается специальная область математики, которая называется "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 20-го века, когда датский математик Агнер Крарруп 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, которая называется *балансировка нагрузки*, и об одном ее относительно недавнем и необыкновенно элегантном решении.

6.2 Параллельные серверы

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

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

Схематически, мы изобразили параллельные серверы на Рисунке 6.1. Запросы на рисунке разной величины, потому что все запросы разного объема. Кто-то запросил страницу с коротеньким текстом, а кто-то – годовой отчет на 100 страниц с графиками и фотографиями. Если информации больше, но на ее отправку нужно больше времени.

Поскольку серверов много, то возникает проблема: на какой сервер отправить ваш запрос? Проблема распределения заданий по параллельным серверам далеко не тривиальная. Например, нельзя допустить, чтобы один сервер был перегружен, а другой простаивал. Желательно распределить нагрузку по серверам равномерно, и свести очереди к минимуму. Как это сделать? В общих чертах, это и есть задача о балансировании нарузки.

Эта задача возникает не только при отправке и пересылке информации. Другой распространенный и очень похожий пример – это вычисления на удаленном компь-

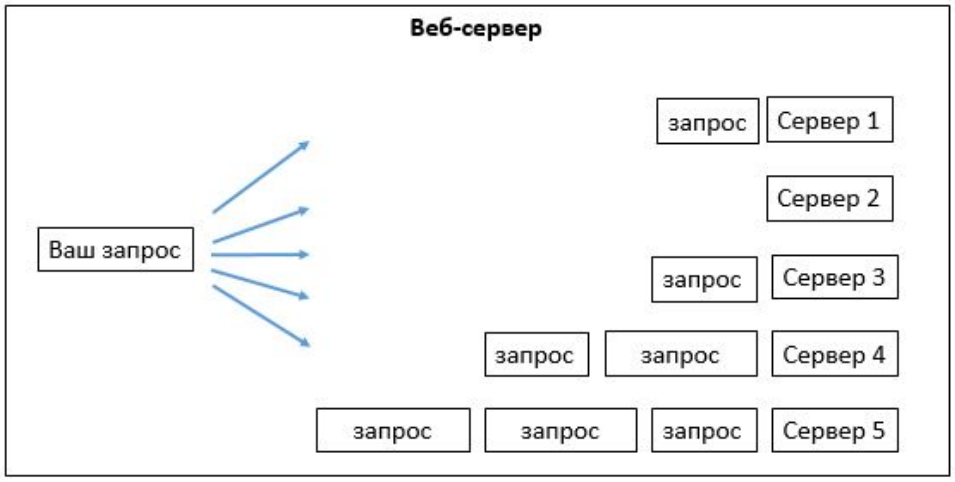

Рис. 6.1: Несколько параллельных серверов. В реальности серверов намного больше. Запросы разной величины, потому что они содержат разное количество информации, и соответственно, для их отправки требуется разное время.

тере. Например, когда в научных вычислениях одним супер-мощным 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 все одновременно.

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

6.3 Какой сервер выбрать?

Возможно, некоторые серверы в данный момент простаивают, а некоторые сильно загружены. Например, на Рисунке 6.1, Сервер 5 загружен больше всех.

Совершенно очевидно, что лучше всего отправить ваш запрос на второй сервер – тот, у которого самая короткая очередь (в данном случае, очереди просто нет). Такая стратегия не всегда оптимальна. Может оказаться так, что в очереди всего один запрос, но зато этот запрос очень объемный, в то время как в другой очереди пять коротеньких запросов, которые будут выполнены моментально. Все мы видели 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 длинее других, вас практически всегда обслужат быстрее.

Если подумать про пример с супермаркетом, то понятно, что лучше всего выбрать

сервер, у которого меньше всего *работы*, то есть который освободится раньше всех. Математически это тоже нетрудно доказать. Однако, есть существенная разница с супермаркетом. В супермаркете мы *видим*, у кого сколько покупок, и примерно знаем, какая очередь пойдет быстрее. Чтобы применить такую тактику на веб-сервере, сервер должен "знать", сколько времени займет обработка каждого запроса в очереди.

Мы привыкли, что компьютеры "умные". Тем не менее, очень важно понять, что компьютер не человек, он не может "взглянуть" и "оценить". Он не умеет "думать". Все, что он может, – это использовать информацию, которая у него есть, и включить эту информацию в *протокол*, то есть заранее установленный порядок действий. Это означает: оценивать объем каждого запроса при входе, хранить эту информацию, и удалить ее, когда запрос выполнен. На это уходит время, тратится память, работа сервера замедляется. Поэтому такие протоколы используются, только когда это действительно необходимо. Например, когда очень большие порции информации должны дойти в реальном времени, скажем, голос или видео по Скайпу.

Если нам неизвестен объем заданий в очереди, то лучшее, что мы можем сделать, это 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. На обмен информацией уйдет какое-то время. Доставленная информация будет хоть немножко, но устраевшей. Информация со всех серверов вместе загрузит канал, который нам нужен для обработки запроса. Кроме этого, нужно где-то сохранить полученную информацию, скорректировать на задержку, просчитать решение. Да, все это происходит быстро. Но в итоге, может быть, выгоднее отправить ваш 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, и в зависимости от ситуации, задача решается разными способами. Один из самых известных и простых методов называется "Power of two choices" - сила выбора из двух.

6.4 Сила выбора из двух

Давайте еще раз посмотрим на два крайних случая. Собирать информацию от всех серверов неудобно и занимает много времени. Отправлять запрос наугад, при полном отсутствии информации, тоже плохо, потому что тогда мы рискуем попасть на самый загруженный сервер. Если крайние случаи нас не устраивают, то скорее всего самое лучшее решение лежит где-то посерединке. 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? Ответ от двух серверов мы получим быстро, и на

загрузку системы это практически не повлияет. Дальше мы посмотрим, на каком из двух серверов очередь меньше, и именно на этот сервер мы отправим наш запрос. У этого простого метода несколько названий: “парадигма двух выборов”, “метод двух случайных выборов”, “метод выбора из двух”. А иногда эту модель называют “модель 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 допустим, что система загружена на 90%. Это означает, что каждый серевер простаивает примерно 10% времени. Это вполне разумно. Если загрузка больше, то из-за неравномерности поступления работы и из-за того, что каждый запрос требует разного времени обработки, очереди будут непозволительно большими. При загрузке 90%, если выбирать сервер наудачу, то в среднем на сервере будет девять соощений. А если применить выбор из двух, то средняя очередь будет примерно 2.35 – эффект почти в четыре раза!

Еще интереснее посмотреть подробнее. Сколько процентов серверов пустуют? Какая длина очереди наиболее типична? В Таблице 6.1 мы посчитали и сравнили процент серверов с очередью 0, 1, 2, 3, 4, 5, 6, и больше 6, при выборе сервера наугад и если применить метода выбора из двух. Загрузка по-прежнему 90%. Результаты довольно точные, когда серверов много.

	Процент серверов, с длинной очереди, как в первой колонке	
Длина очереди	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%

Таблица 6.1: Процент серверов с очередью 1 – 6, и 7 и больше. Загрузка системы 90%. Если выбирать случайный сервер, то почти у половины серверов очередь 7 и больше. При методе выбора из двух наиболее часто встречается очередь длины три, а очередь 7 или больше почти не встречается. Формулы, по которым мы подсчитали эти результаты, приведены ниже в рамке.

Давайте посмотрим, что означают эти числа. Начнем с того, что и в том и вдругом случае 10% серверов пустует. 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 никогда не встречается. По сравнению с этим, кар-

тина при случайном выборе сервера довольно печальна. Почти у половины серверов очередь семь или больше!

Самое поразительное в этой истории, что минимальное количество информации имеет 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.

6.5 Кто придумал и обосновал метод выбора из двух

Первый раз метод выбора из двух предложили ученые-информатики Д.Л. Иагер, Э.Д. Лазовска и Дж. Захорян в 1986 году [13], именно в контексе раздления вычислительной работы между ресурсами. Это практически та же ситуация, что и наш пример с веб-сервером. Авторы заметили, как сильно влияет на систему небольшое количество информации. Они также обратили внимание, что именно выбор из двух (а не из трех или четырех) часто особенно эффективен.

Казалось бы, открытие сделано? Но, как всегда в науке, это было только начало. Дело в том, что все результаты в статье Иагер 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 виртальной дороге за виртуальным рулем, и вам дадут всю статистику – скорость, время, качество вождения, и так далее. Точно так же можно виртуально изобразить работу серверов, и посмотреть, как наши вертуальные серверы будут работать при разных методах распределения нагрузки. Это дает представление о результатах без необходимости что-то менять на реальном сервере, что бывает трудно, или даже невозможно на данном сервере по техническим причинам, и чревато большим количеством ошибок. Точно также, как для большинства из нас Формулу 1 на скорости 250 км в час 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 что имитационного моделирования достаточно. Но у этого подхода есть ограничения. Во-первых, если параметры системы изменились (например, изменилось количество поступающих сообщений в минуту), то 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 довазательства обычно сразу видно, какие факторы оказали решающее влияние.

Мы никогда не смогли бы просто так взять и за пару минут посчитать результаты в Таблице 6.1, и никогда бы не смогли бы точно оценить эффективность выбора из двух, если бы не вклад математиков. Всплеск интереса среди математиков к этой теме произошел во второй половине 1990-х, и именно тогда были получены основные

результаты.

В 1996 году российские математики Н.Д. Введенская, Р.Л. Добрушин и Ф.И. Карпелевич опубликовали статью "Система обслуживания с выбором наименьшей из двух очередей – асимптотический подход" в журнале *"Проблемы передачи информации"* [27]. *Система обслуживания* – это наши несколько серверов, которые должны отправить (обслужить) пользователям веб-страницы. *Выбор наименьшей из двух очередей* мы уже обсудили – это и есть метод выбора из двух.

На самом деле, и про *асимптотический подход* мы уже упомянули несколько раз, просто не пользовались этой математической терминологией. Дело в том, что наша 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 настолько громоздкими, что их невозможно разрешить ни на бумаге, ни даже на компьютере. Эта проблема сильно упрощается, есть предположить, что серверов очень много, бесконечное количество. Тогда самые запутанные части в уравнении становятся настолько маленькими, что ими можно пренебречь. А то, что остается, поддается анализу. Этот анализ по-прежнему далеко нетривиальный, но уже выполнимый! Это и есть *асимптотический* анализ, от слова *асимптота* – предел, который нельзя достигнуть. Асимптотические результаты всегда приблизительные. Никто никогда не даст вам бесконечное количество серверов! Но когда серверов много, то асимптотические результаты очень точные, а главное, они дают нам возможность понять, как же работает система. Именно такой асимптотический анализ и сделали авторы статьи. Результаты в Таблице 6.1 взяты из статьи напрямую, мы только подставили числа. При это мы отдали должное асимптотическому подходу и огворились, что результаты применимы, когда серверов много. Если вам интересны подробности подсчетов, мы приводим их в Приложении 10.4.1.

Практически одновременно с российскими математиками, похожие результаты опубликовал Михаел Митценмахер (Michael Mitzenmacher) из Гарвардского университета. Вскоре появилось еще несколько статей, которые рассматривали разные особености системы (например, длину очереди на самом загруженном сервере) и предлагали разные подходы к решению. Видимо, это был как раз то случай, кода 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 литература о выборе из двух, и других похожих моделях, стала достаточно обширной, и продолжает расти до сих пор.

6.6 Другие применения

В статье [23] и другой литературе упоминается несколько применений 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. Не случайно 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 напишем про два из них, тоже связанные с компьютерными технологиями. Первое – это хеш-таблицы. Они используются для того, чтобы извлечь нужную информацию из памяти компьютера. Компьютер ищет эту информацию по "ключу" – коротенькому коду. Когда информации немного, то ключей достаточно. Но иногда происходят "коллизии": несколько элементов, которые хранятся в памяти, пользуются одним и тем же ключем. В этом случае компьютеру надо проверить всю эту группу (которая называется *цепочкой*), чтобы найти то, что нужно. Желательно, конечно, чтобы коллизий было как можно меньше, а цепочки как можно короче. Для этого часто пользуются методами типа выбора из двух.

Еще одно применение тоже связано с эффективным использованием памяти компьютера, и немножко похоже на наш пример с серверами. При компьютерных вычислениях требуется большое количество памяти. Легче всего, когда необходимое количество памяти доступно в одном месте. Но это не всегда возможно. В реальности распространены так называемые системы с распределенной памятью, где памятью можно воспользоваться на нескольких разных машинах. Методы типа выбора из двух актуальны в нескольких разных вариантах при координации работы систем с распределенной памятью.

6.7 В чем секрет силы выбора из двух

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

Давайте вернемся к Рисунку 6.1. У Сервера 5 самая длинная очередь. Если выбрать сервер наудачу, то наши шансы наткнуться на этот загруженный сервер 1:5. Это 20% случаев! При этом наши шансы попасть на самый лучший сервер, Сервер 2, ровно такие же.

Теперь представьте себе, что мы сначала выбираем два сервера, а потом выбираем 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 даже чтобы попасть на Сервер 4, 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 с этим сервером, 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 самая главная причина 2). Представьте себе, 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, 0.2×0.2, и приводит к совершенно другому решению уравнения, согласно которому длинные очереди становятся практически невероятны.

Для подготовленного читателя в Приложении 10.4.1 мы приводим более формальное математическое объяснение, и формулы для подсчетов в Таблице 6.1.

Так получилось, что Нелли, будучи аспиранткой, присутствовала на одном из первых докладов Никиты Введенской про силу выбора из двух на конференции в Израиле в 1996 году. Когда Никита Дмитриевна объяснила решение, оно оказалось таким красивым и простым, что казалось, 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 такой полезной!

Почти через 20 лет, на конференции в Стамбуле в 2015 году, пленарный доклад делала Кавита Раманан из Университета Браун 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 про метод выбора из двух. Но на данный момент эти исследования уже перешли на другой уровень обобщения и абстракции. Кавита занимается очень сложными асимптотическими процессами, ее доклад за пределами нашей книжки. Проблема развивается и ставит перед математиками новые задачи, еще красивее, еще сложнее, и кто знает, возможно, еще полезнее.

Глава 7

Секретные числа

7.1 Массовый обмен шифровками

Класическая музыка по радио прервалась, и голос диктора стал зачитывать цифры, которые Штирлиц быстро записывал в аккуратные колонки. Диктор читал цифры привычно сухо и четко. “Для него эти цифры были только цифрами,” – подумал Штирлиц. Когда сообщение закончилось, Штирлиц взял с полки книжку в переплете 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 и есть 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 будет необходимо самым обычным людям, причем не изредка, а буквально каждый день. Это связано с лавинообразным развитием технологий, плоды которых доступны каждому: компьютеры, сотовые телефоны и так далее.

Каждый раз, когда вы вводите свой пароль или номер кредитной карточки на веб-сайте, вы отправляете свою личную конфиденциальную информацию по открытым каналам Интернета. У многих есть доступ к этим каналам, 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 веб-сайты, на которых действует протокол безопасной передачи данных: в этом случае веб-адрес начинается с *https://.... HTTP* – это 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 гиганский поток конфиденциальной информации? Естественно, математика, как это всегда и бывало, опережала технологии и стояла у их истока. Задачами шифрования занимается *криптография* – очень активная и интересная область математики и информатики.[1]

7.2 Ключ к шифру

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

РСЙГЁУ

Это очень простой шифр, потому что каждая буква всегда зашифровывается одной и той же буквой. Взломать такой шифр очень просто. 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 помощью этих букв мы можем угадать другие слова, пока наконец не получим расшифровки всего алфавита. 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 переплете 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 всего лишь 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 вам удалось перехватить или вычислить ключ противника, то все его секреты в вашем распоряжении. В этой главе мы расскажем о том, 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, в следующем разделе, мы расскажем о знаменитой шифровальной машине Энигма, которая использовалась немецкой армией во время Второй мировой войны. Это один из самых изощренных шифров той эпохи, когда шифрование еще

[1]Заметим, что не нужно путать шифрование и кодирование. Как мы уже рассказывали в Главе 4, задачи теории кодирования состоят вовсе не в том, чтобы скрыть информацию от кого-либо. 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.

не было таким массовым и обыденным явлением. А ключ к этому шифру нашел английский математик Алан Тьюринг (Alan Turing).

7.3 Алан Тьюринг и Энигма

Фильм "Игра в имитацию" (2014) рассказывает именно об этой потрясающей истории. К сожалению, фильм не объясняет, как была устроена Энигма и как Тьюринг взломал этот шифр. Наш рассказ хотя бы частично восполнит эти упущенные подробности. Мы воспользуемся видео Кэмбриджского университета [2][3]. Очень рекомендуем посмотреть эти видео, чтобы увидеть Энигму в действии.

Энигма совсем небольшая, сравнима по размеру с печатной машинкой. Снаружи машина состоит из клавиатуры и панели, на которой расположены буквы с подстветкой. Если нажать букву на клавиатуре, скажем, A, то на панели высвечивается другая буква, например, Q. Это означает, что в шифровке в этом месте вместо A появится Q. Внутри у машины 3 вращающихся диска, и их положение меняется после набора каждый буквы. Диск повернулся, провода соединились по-другому, и когда мы в следующий раз нажимаем A, но на панели высвечивается уже не Q, а, например, 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 была панель с буквами, и 10 кабелей. Каждый кабель соединял две любые буквы между собой, и при шифровании эти буквы менялись местами. Диски можно было перебрать достаточно быстро, но количество комбинаций на панели было так велико, что перебрать их было невозможно. Всего у Энигмы было

$$158\,962\,555\,217\,826\,360\,000$$

возможных изначальных установок. Каждый день ровно в полночь установки Энигмы менялись. Новое изначальное положение дисков, новые пары букв на панели. Перебрать все комбинации за 24 часа было совершенно невозможно. Шифр считался неуязвимым.

Зашифрованное сообщение передавали по радио. У немецкого офицера, который получал сообщение, тоже была такая же машина. Кроме этого, у него был секретный документ – установки Энигмы на каждый день текущего месяца. Это был ключ к шифру Энигмы. Офицер соединял нужные пары букв, ставил диски в заданное исходное положение, набирал Q, и на панели высвечивалось A. И так дальше, при нажатии букв шифровки на клавиатуре, на панели высвечивались буквы секретного сообщения.

Перехватить документ с установками иногда удавалось, но это было непросто, а потом месяц заканчивался, и разгадать шифр опять было невозможно.

[2] `https://www.youtube.com/watch?v=G2_Q9FoD-oQ`
[3] `https://www.youtube.com/watch?v=V4V2bpZlqx8`

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 была хотя бы какая-то зацепка. Дальше было важно догадаться, что сообщения утром начинаются с чего-то однотипного, например, с прогноза погоды. По-немецки: *wetterbericht*. Поскольку мы знаем, что буква не превращается сама в себя, то мы можем найти слово в начале шифровки, которое в принципе может обозначать *wetterbericht*. Теперь нужно найти 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, то 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 оказалось неправильной, то и все другие пары, которые следовали 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 догадки одновременно. Оставалось только повторить операцию для каждой позиции дисков, а это можно было сделать за 20 минут.

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

Заметим, кстати, что в математике есть понятие “машина Тьюринга”, но речь здесь идет вовсе не о той машине, которая вычисляла ключ Энигмы. “Машина Тьюринга” – это абсолютно абстрактная концепция, которая формально описывает работу компьютера. Это очень важная фундаментальная концепция в математике и информатике. Но этот материал выходит за пределы нашей книги.

7.4 Сила абстрактного подхода к шифрованию

Те или иные математические методы применялись в шифровании уже давно. Тем не менее, когда задача становится по-настоящему масштабной, то 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 этой науке, 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

сообщение, смогла понять, о чем в нем речь.

Чтобы легче понять научный подход к проблеме, давайте будем шифровать цифры вместо букв. В конце концов, мы всегда можем заменить буквы на числа (хотя бы $1, 2, \ldots, 33$). Понятно, что 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 заменять целое число следующим по порядку: вместо 1 писать 2, вместо 2 писать 3, и так далее. Тогда для произвольного числа x наш процесс шифрования будет выглядеть как на Рисунке 7.1 сверху.

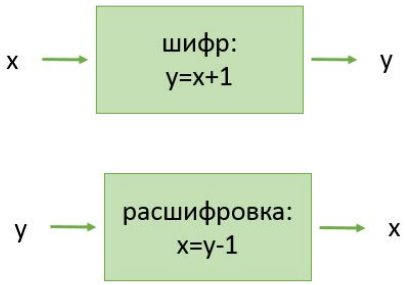

Рис. 7.1: Элементарная шифровка и расшифровка. В данном случае $y = f(x) = x + 1$.

Теперь представьте себе, что Ева умна и хитра, и что она 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 Еве ничего не стоит расшифровать сообщение, она должна всего лишь вычесть единицу!

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

Схематически, это наше пожелание выглядит как на Рисунке 7.2. И сразу возникает законный вопрос: *"Да существуют ли в природе такие преобразования?"*

Если задуматься, то станет ясно, что придумать подходящую функцию f и, главное, *доказать*, что она обладает нужными свойствами, вовсе не легко! Почему "главное доказать"? Вовсе не потому, что математики так уж любят доказательства. А потому, что если свойство доказано, то мы точно знаем, что хитроумная Ева не сможет взломать шифр. Против математических доказательств Ева бессильна.

Рис. 7.2: Шифр $y = f(x)$, который сложно расшифровать, даже если знать преобразование f.

Оказывается, такие преобразования f есть. Более того, можно сделать так, чтобы Боб и Алиса могли расшифровать 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 пока 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*.

7.5 Простые числа

Как вы уже поняли, шифрование имеет дело с числами и преобразованиями чисел. Этими вопросами в математике занимается *теория чисел*. Теория чисел – один из классических, и 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 все усложнялись, без какой-либо перспективы для серьезного применения. И вдруг...

И вдруг в двадцатом 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 вдруг превратилось в жизненно важный источник знаний об основе современной жизни – о числе.

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

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

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 бесконечно много. Однако устроены простые числа крайне сложно, и в теории чисел было брошено немало усилий на изучение их свойств. Внизу в серой рамке для интересующегося читателя мы приведем несколько интересных фактов о простых числах. Этот материал не требует математической подготовки, но и не влияет на наш дальнейший рассказ, поэтому при желании его можно пропустить.

Несколько интересных свойств простых чисел

Простых чисел, которые не превосходят n, примерно $n/\ln(n)$ 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 количество простых чисел 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 П.Л. Чебышёв (1821–1894).

Есть и такой забавный результат: между x и $2x$ всегда есть простое число. Это так называемый постулат Бертрана. Очень сложна проблема его уточнения. 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 про второе 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 отыскания простых чисел не существует в принципе! Простые числа распределены очень хитро. Как мы скоро увидим, простые числа крайне важны на практике и особенно важны 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 $2^{74207281}-1$. Это число, в котором 22338618 знаков! Что-то запредельное...

7.6 Открытый обмен ключами

В настоящее время особенно популярны так называемые *схемы шифрования с открытым ключом*. Мы расскажем о так называемой *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 это число должно быть огромным, но для примера мы возьмем маленькое простое число 19. Теперь выберем еще одно число g, тоже целое, но не обязательно 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, например, $y = 8$, и тоже никому его не сообщает.

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

$$2^6 = 64,$$

$$64 : 19 = 3 \text{ и } \mathbf{7} \text{ в остатке.}$$

Боб делает то же самое со своим задуманным числом:

$$2^8 = 256,$$

$$256 : 19 = 13 \text{ и } \mathbf{9} \text{ в остатке.}$$

То есть наше преобразование выглядит так: возвести 2 в степень x и взять остаток от деления на 19. Мы изобразили это преобразование в общем виде на Рисунке 7.3. Напомним, что числа g и p известные, а число x выбрала Алиса.

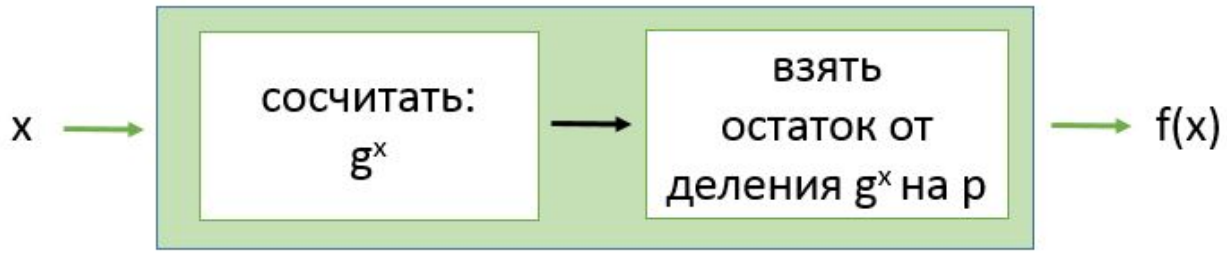

Рис. 7.3: Преобразрвание по схеме Диффи-Хеллмана $x \to f(x)$. Числа g и p известные, а число x выбрала Алиса.

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.

> В Приложениях 10.5.2 и 10.5.3 для подготовленного читателя мы рассказываем более подробно о выборе p и 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:

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

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:

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

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

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

> Для подготовленного читателя в Приложении 10.5.1 мы приводим простое доказательство, что по схеме Диффи–Хеллмана Боб и Алиса получат одно и то же число.

7.7 Зашифровать можно. Расшифровать нельзя!

Вычисление $f(x)$ по схеме Диффи-Хеллмана — это сравнительно быстрая операция, которую можно осуществить и на домашнем компьютере, даже если p огромное, скажем, стозначное число! Тем не менее, вот уже несколько десятилетий лучшие математики всего мира бьются над построением алгоритма, который бы, наоборот, по остатку от деления на p выдавал 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 используется очень активно на практике, и если выбрать число p по-настоящему большим, то можно не бояться взлома шифра. В конце этой главы рассказываем о совсем недавних оценках безопасности схемы Диффи-Хеллмана.

> В Приложении 10.5.2 для подготовленного читателя мы приводим более подробное объяснение, в чем именно трудность задачи расшифровки в схеме Диффи-Хеллмана.

В нашем маленьком примере у Алисы и у Боба после открытого обмена сообщениями появился общий секретный ключ, число 11. С помощью этого ключа Алиса и Боб могут спокойно зашифровывать и расшифровывать сообщения и передавать 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 до одного сообщения, а сделать ничего не может!

Именно поэтому подобные схемы называются “открытый обмен ключами”. Нет никакого секрета в том, как работают эти схемы. При этом несанкционированная расшифровка ваших сообщений – это 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 — Rivest, Shamir, Adleman. Для интересующегося читателя внизу в серой рамке мы кратко объясним основной принцип его работы. Если вы не хотите вдаваться в детали, то этот текст можно пропустить.

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

$$n = p \cdot q.$$

Это очень быстрая операция: обычное умножение. Однако, если вам дано натуральное число n и вам даже известно, что $n = p \cdot q$ 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 обратная к ней операция — операция разложения на множители — носит в математике красивое имя *факторизация* (от английского factor — сомножитель), и представляет из себя 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 числа очень большие. Поэтому на данный момент квантовые компьютеры особой угрозы из себя не представляют.

7.8 Практика шифрования

Схемы типа Диффи-Хеллмана и RSA работают довольно медленно. Есть другие схемы, так называемые *симметричные криптосистемы*, которые работают гораздо быстрее. Но они пользуются ключом, который нужно хранить в тайне. В определенном смысле, эти схемы сравнимы с Энигмой. 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 схемы для обмена ключами. Поэтому обычно сессия безопасного соединения через Интернет состоит из друх этапов.

Сначала происходит так называемое “рукопожатие”. Это обмен сообщениями, во время которого, в частности, устанавливаются ключи. Именно на этом этапе исполь-

зуются схемы с открытыми ключами, Диффи-Хеллман 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.

Когда 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 рассказали, 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.

Например, 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!

7.9 Сто миллионов долларов за число

Помните, как Алан Тьюринг взломал шифр Энигмы? Он построил большую машину, которая могла быстро вычислить ключ. У этой истории две принципиальных составляющих: "машина" и "быстро".

В принципе, так все и осталось до сих пор. Можно сказать, что подход Тьюринга – это прообраз современного подхода к взламыванию шифров. Нужно, чтобы была машина. И нужно найти способ быстрого подсчета ключа. Машину строить необязательно, можно купить суперкомпьютер, были бы средства. Как вы уже поняли, основная загвоздка 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*. По сути, это метод вычисления ключа в схеме Диффи-Хеллмана, при котором сами сообщения нужны только на самом последнем этапе. Эти последние шаги можно совершить очень быстро. Для основной части вычислений нужно 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 только перехватить сообщения, и с их помощью быстро завершить вычисление ключа.

Насколько колоссальны предварительные вычисления? Авторы оценили, что если закупить оборудования на 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, для Агентства национальной безопасности США пара сотен миллионов вполне по карману. И если верить авторам, то эта сумма примерно соответствует бюджету агенства 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,[4] что если взломать одно, наиболее распространенное 308-значное простое число, то 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 примерно до 20% от топ миллиона HTTPS веб-сайтов.

По предположениям авторов, Агентство национальной безопасности на самом деле использует подобные методы, чтобы прослушивать зашифрованный трафик. И многие верят, что так оно и есть. Вполне вероятно, что службы государственной безопасности продвинулись в криптографии гораздо дальше ученых.

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! [5] А аспиранту университета Беркли, который хотел опубликовать новые результаты по криптографии, пришлось 4 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. [6].

На самом деле, трудно понять, где заканчивается свободная наука и начинается государственноая тайна. На конференции французский профессор криптографии Роберт Эрра (Robert Erra) сказал с улыбкой: “Один профессор, звезда в нашей области, утверждал, что он редко встречал криптографа, который не пытался бы взломать RSA. Но честно, если бы мне это удалось, я бы не отправил статью прямиком в журнал. Я бы сначала позвонил 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 бо́льшими, 616-значными простыми числами. А если это технически невозможно, то они советуют по крайней мере выбирать разные простые числа. Сразу после появления статьи многие браузеры обновили протоколы, в том числе Chrome и Firefox.[7] Наши секреты должны оставаться в безопасности, даже если они транслируются по открытым каналам по всему миру.

[4] `https://freedom-to-tinker.com/blog/haldermanheninger/how-is-nsa-breaking-so-much-crypto/`

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

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

[7] Источник: `https://www.schneier.com/blog/archives/2015/05/the_logjam_and_.html`

Глава 8

Счетчики с короткой памятью

8.1 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 самой разной информации. И сразу возникает вопрос, как эту информацию 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, что разобраться в миллионах электронных записей вовсе 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, наивное решение этой проблемы приводит к абсолютно невыполнимым требованиям к компьютерной памяти.

Чтобы было понятно, мы расскажем о самой проблеме чуть позже, в разделе 8.3. А сначала объясним вкратце, как устроена компьютерная память, и почему, несмотря на ее мощь и колоссальный объем, она все-таки не всесильна.

8.2 Компьютерная память

Мы уже рассказали в главе 4, что вся информация в компьютере записывается в виде нулей и единиц. Каждый ноль и единица физически занимает крошечную ячейку

памяти. Чтобы сохранить одну цветную фотографию на 3 мегабайта, нам нужно

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

то есть больше 25 миллионов таких ячеек.

Обычно память работает на полупроводниках. Но для нашего рассказа достаточно, что компьютерная память – это физическое устройство, где каждый символ (0 или 1) занимает место. Устройства эти могут быть самые разные, например CD, DVD или флэшка. В любом компьютере информация записывается на жестком диске. Качество устройств памяти непрерывно улучшается. На жестком диске современных лэптопов может умещаться, скажем, 320 гигабайтов. Этого хватит, чтобы сохранить более ста тысяч цветных фотографий!

Если вам нужно еще больше памяти, то можно хранить данные на удаленном сервере. Многие компании, в том числе *Гугл* или *Амазон*, предлагают такие услуги. Это очень популярный сервис. Возможно, вы тоже что-то храните "на облаке".

У таких информационных гигантов как *Гугл* и *Яндекс* есть свои *дата-центры*, где каждое строение либо заполнено серверами, либо предназначено для их энергоснабжения, охлаждения и обслуживания. На рисунке 8.1 фотографии, которые дают представление о дата-центрах *Яндекса*.

Рис. 8.1: Дата-центры *Яндекса*. Слева: Модуль с двумя рядами стоек с серверами. Спарва: На переднем плане энергоцентр; это большое здание кажется маленьким по сравнению с дата-центром. Источник: `https://yandex.ru/company/technologies/datacenter`.

Из всего этого можно сделать вывод, что компьютерная память 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.

Доступ к информации на жестком диске или сервере занимает какое-то время. Это можно представить себе примерно так. Допустим, вы дома читаете книжку по-английски со словарем. Но только словарь лежит в библиотеке! И вы бегаете в библиотеку и назад за каждым словом, причем каждый раз аккуратно ставите словарь

обратно на полку, а потом занова ищите его по каталогу. Конечно, компьютер найдет информацию на диске быстрее, чем вы добежите до библиотеки. Но и операций компьютер выполняет гораздо больше, поэтому сравнение вполне уместное.

Без сомнений, гораздо удобнее, если 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 сохраняются обратно 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 сильно ограничен. На вашем диване не поместится пол-библиотеки. Обычный лэптоп может предложить, например, два 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.

8.3 Раз, два, три, четыре, пять...

Рассмотрим небольшой пример. Допустим, банк выпустил пятьдесят кредитных карт с номерами $01, 02, \ldots, 50$. 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 воспользовались кредитными картами?

Попробуйте сами дать ответ на этот вопрос. Какие ваши действия? Поначалу все просто: 15 – это раз, 48 – это два, 32 – это три, 31 – это четыре. Дальше снова 48, но эта та же карточка, поэтому мы ее не считаем. 27 – это пять. В первом ряду всего 8 разных номеров. Но только уже во втором ряду начинаются затруднения. Попадался нам до этого номер 02 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 предыдушими и ставить пометку, если этот номер нам раньше не попадался. 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 себе, что эту задачу выполняет компьютер. Каким образом 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 занова с жесткого диска непозволительно долго. А оперативная память переполнится гораздо раньше, чем мы дойдем до середины.

Что же получается? При наличие полных данных и самых мощных компьютеров мы не в состоянии посчитать, сколько человек воспользовалось кредитными карточками?! 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 вебсайте есть счетчик уникальных посещений? Скорее всего, он считает приблизительно.

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

Общее количество заходов на ваш вебсайт, кстати, можно сосчитать и точно, потому что в этом случае не нужно запоминать каждый заход, достаточно помнить сколько их было. Количество друзей на Фейсбук тоже считается точно, потому что они не повторяются. Приблизительное решение нам нужно, чтобы посчитать количество *разных* объектов, когда один и тот же объект может появиться *несколько* раз.

Оговоримся, что в примере с кредитными картами проблему можно решить и подругому, с абсолютной точностью. Например, если выстроить все номера по порядку возрастания, то сосчитать разные номера будет гораздо проще. Очевидный недостаток в том, что обычно данные поступают несортированные, и их сортировка – это отдельная непростая задача, которой посвящена масса математической литературы.

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

Проблема подсчета очень актуальна, особенно в контексте Больших Данных. Сколько поситителей заходит на наш вебсайт из разных регионов России? Сколько школьников в этом году подали заявления в вузы? Сколько людей обсуждают в социальных сетях нашу партию? Сотрудники Гугл в своей статье [17] пишут, что в их систему хранения и обработки данных поступает более пяти миллионов подобных запросов в день! Регулярно встречаются запросы, когда нужно сосчитать более миллиарда объектов. В статье [17] приводится цифра – в среднем около ста запросов в день. Из-за ограничний памяти получить точный ответ на такой запрос абсолютно нереально.

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

8.4 Как решается задача о подсчете

Мы воспользуемся блестящим блогом [3], и начнем с метода *K-Minimum Values* (*K*-минимальные величины). Идея очень простая. Допустим, значения, которые нам надо посчитать, разбросаны равномерно на каком-то интервале. В нашем примере с номерами карточек от 01 до 50 и их непредсказуемым использованием, это предположение вполне разумное. Теперь давайте не будем запоминать все значения, которые мы видели, а запомним всего лишь несколько самых маленьких значений.

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

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

Что-то порядка 25 значений. Поскольку 10 на самой границе интервала, то на самом деле делается коррекция. Для большей точности в данном случае пользуются формулой $(5-1)/(10/50) = 20$. Это, конечно, не равняется точному ответу 22, но достаточно близко. При этом нам нужно было 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 лучше точность, и для по-настоящему хорошей 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* Гугла и *Redis* компании Амазон.

8.5 *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, если у нас миллиард объектов, то двойной логарифм это всего лишь

$$\log_2(\log_2(1\,000\,000\,000)) \approx 4.9,$$

то есть порядка пяти 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.

HyperLogLog сильно улучшил точность, что позволило применить этот метод на практике.

Флажоле, как и Жируар, исходил из предположения, что записи в базе данных можно представить в виде чисел, разбросанных случайным образом. На практике это так и есть, потому что каждой записи, будь то число, имя, адрес или название, присваивается так называемое *хеш*-значение. Хеш-значения – это последовательности из нулей и единиц *одинаковой* и *небольшой* длины. Если две записи совпадают (например, одно и то же название повторяется два раза), то и хеш-значения этих записей совпадают. Например, хеш-значения длины 8 для разных веб-магазинов могут выглядеть примерно как в таблице ниже. Мы выделили жирным шрифтом название, которое повторяется два раза:

№ 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*, очень нетривиальным образом. Усмотреть связь между изначальной записью и ее хеш-значением фактически невозможно. Поэтому можно считать, что хеш-значения присваиваются случайным образом. И метод Жируара, и метод Флажоле пользуются не самими записями, а их хеш-значениями.

Напомним, что Жируар предлагал запомнить несколько самых маленьких хеш-значений. Флажоле пошел еще дальше. Он предложил запоминать только число нулей в начале хеш-значений.

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

0000$1011.

Последовательности, которые начинаются на четыре нуля, попадаются 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 встречаются еще реже, 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 число нулей, которое мы видели, например “4”. Для этого достаточно минимального объема памяти. Если нам попалось хеш-значение, у которого нулей больше, например, 5, то мы выкидываем из памяти “4” и запоминаем “5”.

> Для подготовленного читателя в Приложении 10.6.1 мы объясняем, откуда возникает двойной логарифм.

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

В *HyperLogLog* добавлено много хитрых приемов, чтобы улучшить точность. 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 [3]. Там можно даже запустить программу и посмотреть, как метод работает.

На практике стараются достичь еще более высокой точности. Собственно, об этом статья сотрудников Гугла [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.

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

8.6 Четыре виртуальных рукопожатия

Задача о подсчете важна сама по себе, но находит и другие, совершенно неожиданные применения.

Все мы знаем, что мир тесен. Знакомишься с человеком, и немедленно обнаруживаются 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 1969-х годов социолог Стэнли Милграм провел знаменитый эксперимент. Он раздал случайным людям в штатах Небраска и Вичита письма, адресованные брокеру из Бостона. Географиически и по роду занятий эти люди были дастаточно далеки от адресата. По правилам эксперимента, участники могли переправить письма только своим знакомым, которые должны были передать их дальше, своим знакомым, и так далее. Из 296 писем большинство затерялось в дороге, но 64 письма дошли-таки до адресата. При этом оказалось, что цепочка, связывающая совершенно незнакомых людей, совсем короткая. В среднем отправителя и адресата разделяло всего 5 человек! На рисунке 8.2 мы схематически избразили, как письмо пересылалось всего шесть раз, через пять промежуточных людей.

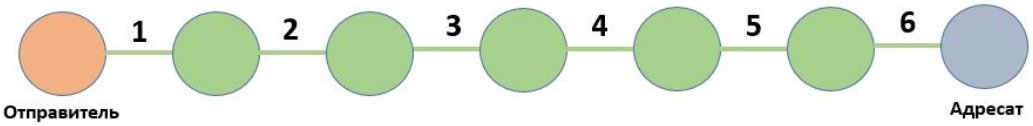

Рис. 8.2: Схематическое изображение эксперимента Стэнли Милграма. Участники могли переслать письмо только своим знакомым. Письмо от отправителя до адресатого пересылалось 6 раз, через 5 разных людей.

Так родилось понятие "шести рукопожатий". Если вы можете пожать руку всем своим знакомым, ваши знакомые могут пожать руку своим знакомым, и так далее, то понадобится всего шесть рукопожатий, чтобы соединить вас с любым человеком на Земле!

Эксперимент Милграма не идеален, не случайно столько писем затерялась в дороге. Но в то время просто не было другого способа обнаружить, кто кого знает, и кто кого и с кем соединяет.

Зато сейчас таких данных сколько угодно. Чтобы провести подобный эксперимент на Фейсбуке не нужно даже беспокоить участников. На сервере Фейсбука хранится полная информация, кто с кем дружит. Осталось только вычислить количество "виртуальных рукопожатий", отделяющих одного пользователя от другого. Можем ли мы это сделать? Именно такой вопрос задали научные сотрудники Фейсбука профессору университета Милана по имени Себастьяно Винья.

И Себастьяно, и сотрудникам Фейсбука было очевидно, что задача эта колоссальная. Не случайно она не была решена раньше. У Фейсбука более 700 милионов активных пользователей, то есть более

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

пар пользователей. И нужно вычислить длину цепочки для каждой пары! Но дело не только в этом. Самая главная проблема опять в количестве оперативной памяти.

Проблема возникает потому, что между двумя пользователями не одна, а несколько цепочек разной длины. Для примера, посмотрим на маленькую социальную сеть на рисунке 8.3. А отделяет от В одно рукопожатие, или два – через Г. Разных цепочек очень много, потому что друзья друзей очень часто 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 рисунке 8.3 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, который называется *Поиск в ширину*.

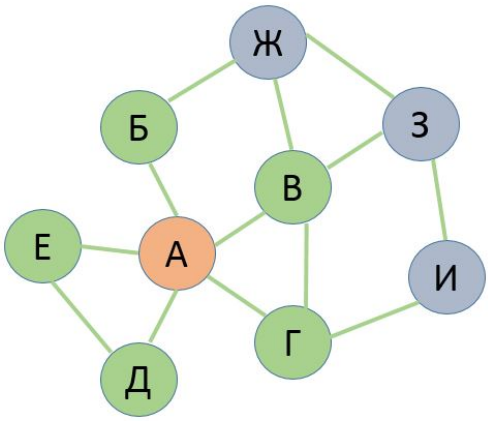

Рис. 8.3: Мини социальная сеть. Кружки с буквами означают пользователей, зеленая линия мужду двумя пользователями означает, что они друзья. Зеленым цветом обозначчены друзья пользователя А. Синим цветом обозначены пользователи, которых от А отделяют два рукопожатия.

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 совсем другой способ посчитать число рукопожатий. Они заметили, что самая главная проблема – необходимость запоминать увиденных пользователей – совершенно аналогична задаче о подсчете. А задачу о подсчете можно решить методом *HyperLogLog*.

Для интересующихся, в рамке ниже мы объясним основые идеи с помощью мини-примера на рисунке 8.3. Этот текст не требует математической подготовки, но если вы не хотите вдаваться в подробности, то его можно пропустить.

Для интересующегося читателя

Применение *HyperLogLog* счетчика для подсчета числа рукопожатий

Хорошая новость, что можно очень легко определить всех друзей любого пользователя.
Присвоим хеш-значение каждому пользователю. Запустим счетчик *HyperLogLog*, и будем просматривать 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 друзей А:

Б В Г Д Е,

HyperLogLog сообщит нам, что всего с момента запуска мы видели 6 пользовалей. Минус А, получаем 5 пользователей на расстоянии одного рукопожатия.
Теперь к А и его друзьям добавим всех пользователей, 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. Вычтем А и число его друзей, получается

$$9 - 1 - 5 = 3$$

пользователя на рассоянии двух рукопожатий от А. Заметьте, что нам не нужно запоминать самих друзей А, 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 А отделяют ровно три рукопожатия. И так далее.
Процесс завершен, когда мы увидели всех пользователей сети.
Как вы помните, *HyperLogLog* держит в памяти только максимальное число нулей в хеш-значении. Плюс к этому, надо запомнить только *число* пользователей на расстоянии одного, двух, трех и так далее, рукопожатий. Это тоже занимает очень скромное количество 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 высока.

Оказалось, что двух пользователей Фейсбука в среднем разделяет не шесть, а всего 4.74 рукопожатий! Статья Себастьяно Винья и соавторов [10] стала быстро известна

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 мир, в котором мы живем!

8.7 Филипп Флажоле

Филипп Флажоле умер в 2011 году, когда работа над врендрением *HyperLogLog* была в самом разраге. Блог под названием *"HyperLogLog — краеугольный камень инфраструктуры Больших Данных"* [3] был написан в 2012 году. Статья [10] про 4.74 виртуальных рукопожатия в Фейсбуке вышла в том же 2012 году, а статья сотрудников Гугла [17] – в 2013-м. Система *Redis* внедрила *HyperLogLog* в 2014 году. В честь Филиппа Флажоле команды *HyperLogLog* начинаются с его инициалов *PF*.

Нам не известно, успел ли Флажоле узнать, что его результаты уже были на пороге широкого внедрения. Мы закончим последними фразами из блога [3], 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 9

Миллион аукционов в минуту

9.1 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 начинается с белой странички поисковой системы. Популярность и полезность поисковиков трудно переоценить. Один Гугл обрабатывает более 100 миллиардов запросов в месяц!

Если ввести в Яндексе типичный запрос, например *"пластиковые окна"*, то первая страница результатов будет выглядеть примерно как на Рисунке 9.1.

Содержание первой страницы – тема особенно интересная, потому что большинство из нас дальше первой страницы не заходит, 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 – наверху первой страницы – появляется реклама. На рисунке мы обвели рекламу красным контуром.

Рынок он-лайн рекламы у поисковиков просто колоссальный. Реклама – это источник доходов Яндекса практически на сто процентов. Число рекламодателей Гугла уже давно превысило миллион, а доход Гугла от он-лайн рекламы в 2015 году оценивается в 44 с лишним миллиардов долларов.

Почему практически у всех крупных поисковиков он-лайн реклама выглядит именно так, как сегодня на нашем экране? Почему Вы видете 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 он-лайн рекламы.

9.2 Стоимость за один клик

Поисковики в их современном виде появились совсем недавно – в конце 1990-х годов. Реклама гораздо старее: газеты, щиты, ролики по телевизору. Мы расскажем о развитии и особенностях онлайн рекламы примерно так, как нам рассказал Сергей

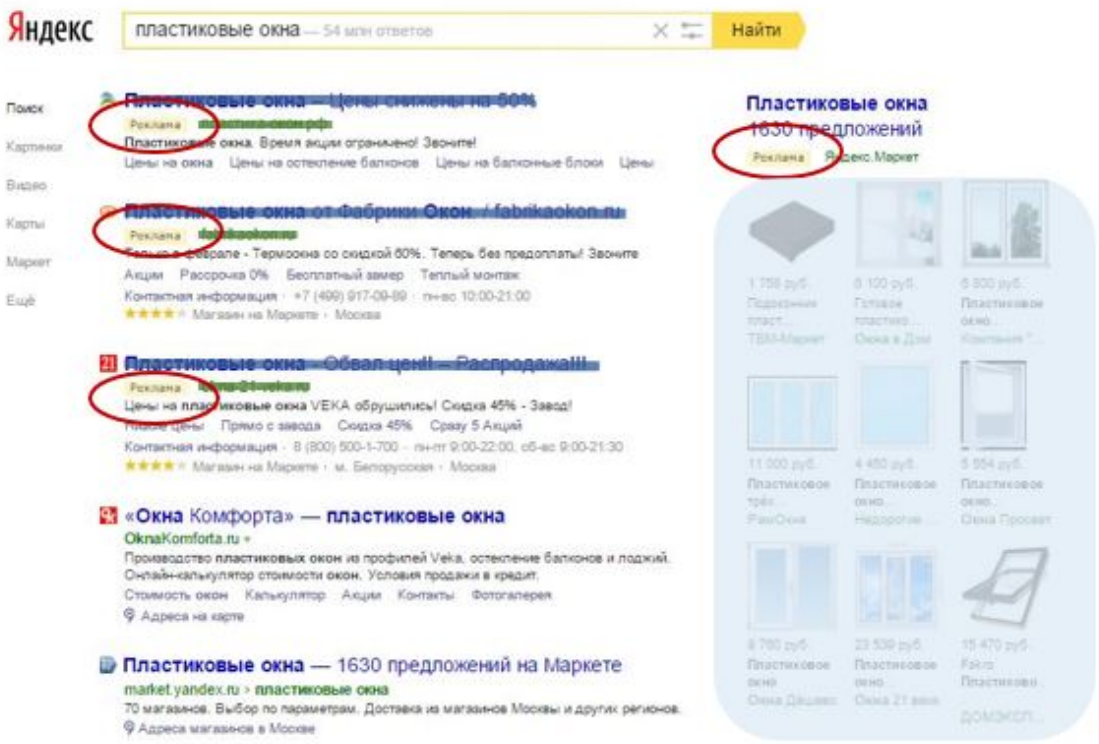

Рис. 9.1: Результаты Яндекса по поисковому запросу *"пластиковые окна"*. Мы обвели рекламу красным контуром.

Измалков – профессор Российской Экономической Школы, и один из лучших специалистов в этой области.

Все мы так привыкли к рекламе в поисковиках, что самый первый тезис Сергея застает в расплох: *"Изначально, когда поисковые системы только-только появились, выгода такой рекламы 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 любые. Показывать рекламу косметики человеку, который зашел искать байдарочные маршруты Кировской области, более-менее бесполезно.

Так возникла идея рекламы по *"ключевой фразе"*. Вместо "журнала" рекламодатель выбирает слова в поисковом запросе, которые его интересуют. Например, если компания устанавливает пластиковые окна, то они могут выбрать запросы *"пластиковые окна"*, *" окна"*, *"ремонт квартиры"*, и так далее. 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 приготовлению пиццы (а не рекламу пиццерий!).

Такой "контекстной" рекламой пользуются все поисковые системы, и все крупные сайты, включая Фейсбук и *Youtube*. 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 известное число подписчиков, и объявление можно увидеть своими глазами. Поисковая система – это совсем другое дело. Рекламодатели не готовы платить за показы, которых будет неизвестно сколько, и которые никак нельзя ни увидеть, ни проверить.

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

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

Стоимость за клик – очень красивое рыночное решение. Рекламодатель может легко подсчитать, сколько посещений пришло 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 пользователи были довольны и продолжали пользоваться системой. В результате все в выигрыше: и рекламодатели, и поисковые системы, и мы – простые пользователи и покупатели.

9.3 Аукцион – специально для вас!

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

Современные поисковые ситемы 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 Гугле обычно 1-2 доллара, но доходит и до 50 долларов, и выше, особенно в финансовом секторе.

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

Каждый раз, когда вы вводите запрос типа *"пластиковые окна"*, поисковая система

по вашему запросу проводит аукцион, и представляет вам победителей! Это делают все поисковые системы. И поскольку один Гугл обрабатывает примерно 40 тысяч запросов в секунду, то миллион аукционов в минуту – это 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 начинается 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 все интересы? 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, которая называется "Дизайн механизмов". В 2007 году ее основатели Леонид Гурвич (Leonid Hurwicz), Эрик Маскин (Eric Maskin) и Роджер Майерсон (Roger Myerson) получили Нобелевскую премию по экономике. "Механизм" в данном случае это 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.

9.4 Аукцион второй цены

При слове "аукцион" представляется комната хорошо одетых людей, и на трибуне – человек с молотком. 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 ставки и вычисляет победителей и цены. Это сравнимо с ситуацией, когда каждый участник записывает ставку и отдает организаторам в запечатанном конверте. В момент розыгрыша конверты вскрываются и товар уходит покупателю, чья ставка оказалась самой высокой.

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

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

Если задуматься, то традиционный открытый аукцион с повышающимися став-

ками – 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 стало выгодно и безпасно. 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 – это один из основных принципов теории дизаяйа механизмов, который называется *совместимость по стимулам*. Аукцион второй цены – классический пример этого принципа. Математический анализ аукциона второй цены и доказательство совместимости по стимулам впервые предложил Уильям Викри (William Vickrey). Его статья, которая заложила теорию аукционов, вышла в 1961 году [26]. В 1996 году 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.[1] Ниже в Таблице 9.1 мы выписали все четыре ключевые идеи, которые привели 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

Таблица 9.1: Четыре ключевые идеи современной рекламы в поисковых системах.

9.5 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 снова к примеру, когда Анна, Борис, и еще несколько участников участвуют в аукционе второй цены. Каждый пишет свою ставку и кладет в конверт. Допустим, для Анны ценность товара 500, и ставок доугих участников она не знает.

Что будет, если Анна поставит 600? На Рисунке 9.2 мы изобразили три возможных сценария. Ставки участников обозначены жирными точками. В синей рамке вторая по величине ставка, это окончательная стоимость товара в аукционе второй цены. Если все остальные участники поставили меньше 500, как на Рисунке 9.2 сверху, то ничего

[1]Помимо поисковых систем, аукционами второй цены пользуется корпорация электронной торговли *eBay*.

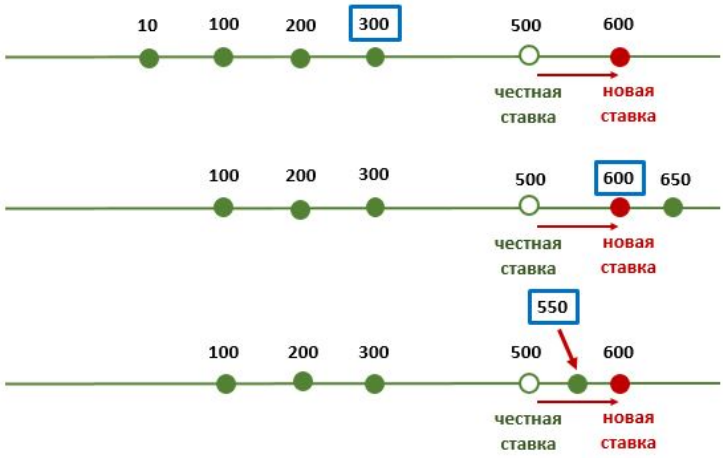

Рис. 9.2: Три сценария, когда Анна вместо правдивой ставки 500 делает более высокую ставку 600. Сверху: Остальные участники поставили меньше 500; ничего не изменилось. В середине: Кто-то из участников поставил больше 600; для Анны ничего не изменилось. Снизу: Один из участников поставил 550; Анна платит больше своей максимальной цены.

не изменилось, Анна по-прежнему получает товар за 300. Повышать ставку смысла не было. Если кто-то поставил больше 600, как на Рисунке 9.2 в середине, то Анна товар не получает в любом случае. Победитель платит больше, но для самой Анны ничего не изменилось. Опять же, повышать ставку смысла не было (разве что из вредности, но модель этого не учитывает). Теперь представьте себе, что кто-то поставил 550, как на Рисунке 9.2 снизу. 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.

Но, может, выгодно сделать более низкую ставку? Оказывается, нет! Допустим, Анна решила попробовать поставить поменьше, скажем, 400. На Рисунке 9.3 те же три сценария. Если все остальные участники поставили меньше 400, как на Рисунке 9.3 сверху, то ничего не изменилось, Анна по-прежнему получает товар за 300. Если кто-то поставил больше 500, как на Рисунке 9.3 в середине, то Анна товар не получает в любом случае. Но если кто-то поставил 450, как на Рисунке 9.3 снизу, 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.

> В Приложении 10.7.1 для подготовленного читателя мы приводим более формальное доказательство совместимости по стимулам в аукционе второй цены.

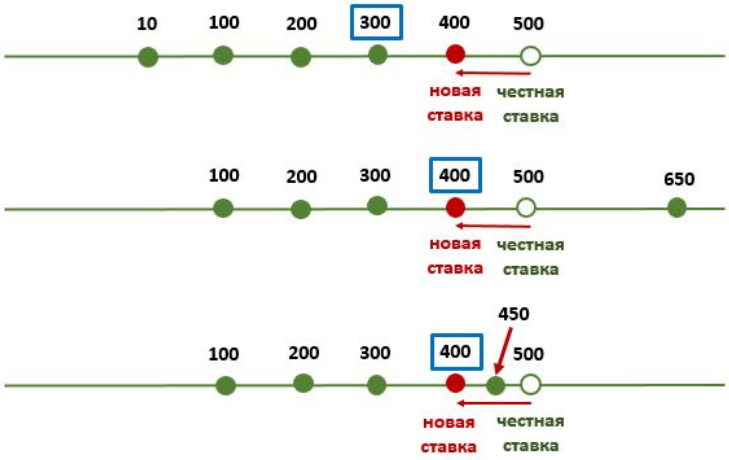

Рис. 9.3: Три сценария, когда Анна вместо правдивой ставки 500 делает более низкую ставку 400. Сверху: Остальные участники поставили меньше 400; ничего не изменилось. В середине: Кто-то из участников поставил больше 500; для Анны ничего не изменилось. Снизу: Один из участников поставил 450; Анна не получает товар, хотя цена ее устраивала.

Это очень простой и глубокий математический результат, в котором сходятся сразу две фундаментальные концепции. Одну из них мы уже обсудили, это совместимость по стимулам, когда честные ставки самые выгодные. Вторая концепция – это так называемое *равновесие по Нэшу*. Речь идет о том самом Джоне Нэше (John Nash) из фильма *"Игры разума"*.

Равновесие по Нэшу – это такая ситуация, когда ни одному из участников не выгодно в одиночку отклоняться от выбранной стратегии. Эта концепция лежит в основе *теории игр*, за разработку которой Джон Нэш полчил Нобелевскую премию по экономике 1994-го года. "Игрой" можно назвать любую ситуацию, когда исход зависит от действий всех участников, и у каждого участника свои интересы. Очевидно, что аукцион – это тоже игра. Участники выбирают ставки с целью получить товар по самой выгодной цене.

Для читателей с математической подготовкой, которых заинтересовала теория аукционов и три Нобелевские премии, рекоммендуем видеолекцию Алексея Савватеева [7], где он подробно и очень доходчиво объясняет математическую сторону вопроса.

9.6 Как распределить несколько рекламных мест

Аукцион Викри сильно упрощает реальность. Поисовые 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 Гугла, на мобильном телефоне почти 28% пользователей кликает на самую верхнюю рекламу, а на вторую сверху – чуть больше 9%. Чем больше вероятность

кликов, тем дороже место.

Три верхние строчки – самые ценные. Часто нижние позиции на первой странице тоже заняты рекламой, это 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 нескольких товаров. Это хорошо изученная модель аукциона, так называемый механизм Викри-Кларка-Гровса (Vickrey – Clarke – Groves), который часто называют VCG, по первым буквам в именах его создателей. Схема вычисления цен и математические свойста VCG довольно сложны и выходят за пределы нашей книги. Мы назовем только две основных 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 *обощенный аукцион второй цены*, изобретение Гугла. Систма простая: 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 Таблице 9.2. Разыгрываются три места. Тогда А получает первое место за 7, Б получает

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

Таблица 9.2: Четыре рекламодателя и их ставки за клик. Разыгрывается три рекламных места. При обощенном аукционе второй цены, первое место получит А по цене 7 (красный цвет), второе – Б по цене 3 (синий цвет), и третье – В по цене 1 (зеленый цвет). Рекламодатель Г не получит рекламного места.

второе место по цене 3, В – третье место по цене 1, а Г рекламного места не получает. В Таблице 9.2 первое место и соответствующая цена обозначены красным цветом, второе – синим, а третье – зеленым.

Гугл пришел к этой системе, исходя из идеии 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 более выгодными. Это одна из причин, по которой Яндекс с августа 2015 года перешел на аукционы типа VCG, где честные ставки ведут к оптимальной позиии по оптимальной цене для всех участников.

Для заинтересованного читателя ниже в серой рамке мы приводим пример, когда в обощенном аукционе второй цены одному из участиков выгодно занизить ставку. Этот пример не требует никакой математической подготовки, но если вы не хотите вникать в детали, то его можно пропустить.

Обощенный аукцион второй цены, в котором невыгодно делать честную ставку.

Рассмотрим снова пример в Таблице 9.2. 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 позиции. 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

$$[20\% \text{ от } 10] - [20\% \text{ от } 7] = 2 - 1.4 = 0.6.$$

Но если бы А поставил 6, то он бы получил вторую позицую за цену 3, то есть прибыль была бы

$$[11\% \text{ от } 10] - [11\% \text{ от } 3] = 1.1 - 0.33 = 0.77.$$

Это больше, чем 0.6. Получается, что для А выгоднее сделать заниженную ставку 6, а не правдивую ставку 10.
Подготовленному читателю мы рекоммендуем книгу Йона Клайнберга (Jon Kleinberg) и Дэвида Изли (David Easley) [14]. В частности, в Главе 15 подробно разобраны свойста обощенного 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. Качественные рекламодатели приносят 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 вебсайта они оценивали как вероятность клика. Тогда ставка рекламодателя определялась как средний доход, который он предлагает:

$$[\text{ставка в аукционе } \textit{Yahoo}] = [\text{денежная ставка рекламодателя}] \times [\text{вероятность клика}].$$

Как именно определяется качество в современных системах и как оно влияет на подсчет ставок – строгая коммерческая тайна. Это зависит от качества вебсайта, отзывов

покупателей, интересов данного пользователя, и многого другого.

Для поисковой системы, в конечном итоге, качество – это потенциальное число кликов, которое принесет объявление. Но крайне важно и то, что пользователям нравится видеть качественные объявления. Это поддерживает репутацию поисковика. А при нынешней конкуренции репутация бесценна!

9.7 По ту и другую сторону он-лайн 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 максимому, то 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 очень много, 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 траффик, который захватил ресурсы, необходимые, собственно, для поиска! И никакой особой выгоды все это не принесло.

Теоретический результат ясно 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 сегодняшнем 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, и вопросов остается много. У рекламы и поиска разные цели. Реклама приносит доход, но может раздражать. Как оценить эти убытки от рекламы? Как предсказать качество рекламы? Имеет ли смысл учитывать частную информацию для более целевой рекламы, с риском испугать пользователя, что про него так много известно? Все это экономические задачи, которые невозможно решить без математических моделей, потому что мы имеем дело с массовыми, сложными и очень взаимосвязанными процессами.

У рекламодателей вопросов не меньше. Средний веб-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 возникли тысячи маркетинговых компаний, которые специализирутся на онлайн рекламе.

Б*о*льшая часть онлайн маркетинга пока основывается на опыте и здравом смысле. Например, если у вас маленький магазин спортивной обуви, то бесполезно ставить на слово *“Найк”*, лучше поставить на *“синие женские кроссовки Найк”*. Но некоторые компании уже разрабатывают и используют компьютерные программы, чтобы

подсчитывать оптимальные ставки автоматически. Например, исходя из предыдущих ставок и продаж. При растущем размахе электронной торговли, у нас есть все основания считать, что будущее именно за автоматическими методами и математическими моделями. По ту и другую сторону онлайн рекламы, работы для математиков хватит надолго.

Заключение: ч.т.д.

В Европе компании часто обращаются к математикам по самым разнвм 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 подступаться и отступать, пока вдруг не возникнет полная ясность. Невозмножно 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 мы видели много примеров, когда приложения стали источником новых математических теорий. Например, в Главе 3 мы рассказали о линейном программировании и его применениях для составления расписаний. Основатель линейного программирования, замечательный советский математик Л. В. 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 применения во многих системах, где есть заявки, очереди и обслуживающий прибор. Будь то супермаркет, поликлиника, колл-центр, или веб-сервер, который посылает вам нужную веб-страничку через Интернет. Про задачу балансировки нарузки на веб-сервере мы рассказали в Главе 6.

Конечно, далеко не вся наука основана на приложениях. Фундаментальная наука зачастую движима исключительно любопытством ученых. Согласно источникам [25], таким ученым был Нильс Бор. Одного этого имени достаточно, чтобы понять, как важно доверять ученым в выборе собственных интересов. В Главе 7 мы рассказали о том, как любимая игра ума математиков – теория чисел – сегодня совершенно необходима для зашифровки конфиденциальных данных, которые мы передаем по каналам Интернета каждый день.

Математика полна нерешенных чисто теоретических задач. Например, в комбинаторике есть задача о так называемом *хроматическом числе*. Допустим, нам нужно расскрасить плоскость так, чтобы любые две точки, которые находятся на расстоянии ровно 1 метр друг от друга, были разного цвета. Вопрос: какое минимальное количество красок нам понадобится? Точный ответ неизвестен. Мы знаем только, что это число между 4 и 7. Если нам нужно раскрасить трехмерное пространство, то ми-

нимальное 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 о хроматических числах привели к мощному развитию комбинаторики, в том числе и прикладной. Хроматические числа используются, например, для расстановки вышек мобильной связи.

Как мы уже упомянули во Введении, в нашей книге мы рассказали лишь об очень маленькой доле того огромного влияния, которое оказывает математика на жизнь буквально каждого человека. Мы выбрали всего несколько тем, которые имеют отношение к компьютерным технологиям и к нашим собственным исследованиям. Мы надеемся, что даже с помощью этой небольшой выборки мы смогли убедить вас, что современные технологии невозможны без математики, такой красивой, такой точной и такой невероятно полезной. Что и требовалось доказать.

Нелли Литвак, Андрей Райгородский

Энсхеде – Москва
Июнь 2016

Глава 10

Приложения для подготовленного читателя

10.1 Приложение к Главе 3

10.1.1 Существует оптимальное решение, соответствующее одному из углов многогранника

Заметьте, что в выражении стоимости $1020 - 2 \times$ АЮ $- 5 \times$ БЮ в нашем примере оптимальные значения *АЮ* и *БЮ* не зависят от слагаемого 1020. Решение будет то же самое, если мы будем минимизировать -2АЮ $-$ 5БЮ, или максимиировать 2АЮ + 5БЮ.

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:
$a_{11}x_1 + a_{12}x_2 \leq b_1,$ (ограничение 1)
$a_{21}x_1 + a_{22}x_2 \leq b_2,$ (ограничение 2)
$\ldots$
$a_{m1}x_1 + a_{m2}x_2 \leq b_m,$ (ограничение m)
$c_1, c_2,\ b_i, a_{ij} \in \mathbb{R}, i = 1, 2, \ldots, m,\ j = 1, 2.$

Заметьте, что, во-первых, задача максимизации эквивалентна задаче минимизации с коэффициентами $-c_1$ и $-c_2$. Во-вторых, любое неравенство со знаком $\leq$ можно превратить в эквивалентное неравенство со знаком $\geq$, умножив обе части неравенства

на -1. Поэтому задача выше, для двух переменных и m ограничений, сформулирована действительно в общем виде. Все значения коеффициентов a, b, c – произвольные реальные числа, которые могут быть как положительными, так и отрицательными.

Каждое ограничение задает полуплоскость значений, на которой это ограничение выполняется. Если пересечение всех m полуплоскостей пусто, то допустимого решения просто не существует. Поэтому допустим, что m полуплоскостей сожержат общую ограниченную область S допустимых значений. (Мы не будем рассматривать случай, когда область неограничена.) 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 (x_1^*, x_2^*). Заметьте, что (x_1^*, x_2^*) не мржет 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 3, решение (58,8) является внутренней точкой, и поэтому не может быть оптимальным.

Значит, (x_1^*, x_2^*) 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?

Для начала, перепишем равенство в более привычное уравнение прямой:

$$x_2 = -(a_{11}/a_{12})\, x_1 + b_1/a_{12}. \tag{10.1}$$

Допустим, мы начали в точке (x_1, x_2). Теперь допустим мы немножко 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

$$-(a_{11}/a_{12})\,(x_1 + \delta) + b_1/a_{12} - (-(a_{11}/a_{12})\, x_1 + b_1/a_{12}) = -(a_{11}/a_{12})\,\delta.$$

При этом натрудно проверить, что целевая функция изменится на

$$c_1\delta + c_2(-(a_{11}/a_{12})\,\delta) = (c_1 - c_2 a_{11}/a_{12})\delta.$$

Заметьте, что это число не зависит от (x_1, x_2). Значит, в какой бы точке прямой (10.1) мы не находились, в результате перемещения по этой прямой, изменение значения целевой функции зависит только от коеффициента $(c_1 - c_2 a_{11}/a_{12})$. Если этот коеффициент отрицательный, то увеличивая x_1 и двигаясь по прямой, мы можем только уменьшить целевую функцию. Аналогично, если этот коеффициент положительный, то двигаясь по прямой в сторону увеличения x_1, мы можем целевую функцию только увеличить. Наконец, если этот коеффициет равен нулю, то значение целевой функции на всей прямой постоянно.

Значит, 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, то делаем вывод, что максимальное значение целевой функции достигается в одном из углов S, и мы можем выбрать этот угол в качестве (x_1^*, x_2^*).

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

10.1.2 Пример задачи целочисленного программирования

Допустим, нам нужно оптравить грузовики с товаром к 2-м разным клиентам. Всего у нас в разных точках 4 грузовика. Обозначим через c_{ij} цену отправки грузовика $i = 1, 2, 3, 4$ к клиенту $j = 1, 2$. На любую доставку необходимо полдня. Доставку можно осуществить либо утром (1-я половина дня), либо днем (2-я половина дня). Нужно решить, какой грузовик поедет к какому клиенту, 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 клиенту 2 в 1-й половине дня, то $x_{321} = 1$. Если этого не происходит (то есть грузовик в 1-й 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 1-й половине дня, то $x_{321} = 1$, и мы прибавим к общей стоимости c_{32}. А если грузовик 3 к клиенту 2 не поедет, то 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:

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

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 ровно один грузовик, то есть

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

Ограничения могут учитывать особенности каждого грузовика, клиента, и так далее. 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 не обязательное. Тогда к целевой функции можно добавить дополнительное слагаемое, которое будет означать штраф за то, что условие не выполнено:

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

Заметьте, что это слогаемое дейтвительно добавится только если грузовик 3 работал в утреннюю смену. Естественно, оптимальное решение будет зависить от коеффициента $c_{\text{штраф}}$. Если этот коеффициент больше любого c_{ij} в целевой функции, то оптимальный вариант – это не задействовать грузовик 3 с утра. А если коеффициент $c_{\text{штраф}}$ 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 обе половины дня, либо не работал. Тогда мы вводим ограничение

$$x_{311} + x_{321} = x_{312} + x_{322}. \tag{10.2}$$

Это условие можно немножко 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. Введем достаточно большое значение M и запишем:

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

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 взять $M = 1$. Для увеличения скорости решения M стараются выбирать “экономно” – не больше, чем нужно.

Есть еще множество интересных приемов, чтобы записать обязательные и желательные условия в виде линейных выражений. Их более подробное описание выходит за пределы нашей книги.

10.1.3 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 мы запускаем симлекс-метод и получаем дробное решение. Например, на объект A нужно отправить 2 и 2/3 землекопа. Допустим, общая стоимость работ при этом получилась 40 000 рублей. Это пока не дает нам плана работ, потому что решение не в целых числах. Но 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 мы начинаем "разветвлять" решение. У нас есть два варианта: $A \leq 2$ и $A \geq 3$. Для каждого из этих вариантов мы снова решаем задачу линейного программирования. Допустим, стоимость получилась 43 000 рублей при $A \geq 3$, и 51 000 при $A \leq 2$. Вариант $A \leq 2$ мы отсекаем, 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 пока остались дробные переменные, то каждую дробнрую переменную мы разветвляем снова. И так до тех пор, пока не найдем решения в целых числах.

10.2 Приложения к Главе 4

10.2.1 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. Аналогично для каждого из этих четырех вариантов есть два способа выбрать третий элемент последовательности, и так далее. В итоге для кодового слова длины пять мы получаем

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

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 мы получаем 2^n разных последовательностей из нулей и единиц.

10.2.2 Граница Хэмминга

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

Если код исправляет d ошибок, то шары с центрами в кодовых словах и радиусами d попарно не пересекаются. Объем шара (т.е. количество слов в нем) нетрудно вычислить. Сколько есть слов, которые отстают от центра шара на данное 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 обозначатеся C_n^k. Для того, чтобы записать это число, нам понадобятся произведения вида

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

Такое произведение принято обозначать записью

$$k!$$

и читать эту запись "k факториал". Легко увидеть, что, конечно, $1! = 1$, и принято считать, что $0! = 1$. Заметим, что факториал уже встречался нам в Главе 3, см. Раздел 3.2. Там мы показали, что факториал растет очень быстро, например, $25!$ – это колоссальное число.

Число сочетаний вычисляется по формуле

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

Мы приводим вывод этой известной формулы ниже, см. Раздел 10.2.3. Легко проверить, например, что $C_n^1 = n$, и действительно мы можем выбрать одну позицию из n ровно n способами.

Значит, всего внутри шара

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

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 $N \sum\limits_{i=0}^{d} C_n^i$ различных слов. Но это количество заведомо не превосходит числа всех возможных кодовых слов, каковое, как мы уже знаем, равно 2^n. Таким образом,

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

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

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

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

10.2.3 Число сочетаний из n по k

Мы рассмотрим число сочетаний на примере, связанным с кодированием. Давайте попробуем посчитать, сколько есть слов длины n и веса k, $k \leq n$. 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 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 воторую позиции для единиц равно $n(n-1)$. Точно так же получаем, что три позиции можно последовательно выбрать одним из $n(n-1)(n-2)$ способов. И так далее. Для данного k будет всего

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

вариантов. Это и есть ответ? Не совсем!

Заметим, что в нашем примере, где $n = 6$ и $k = 3$, мы могли сперва 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 $n(n-1)(n-2)\cdot\ldots\cdot(n-k+1)$ мы посчитали, тем самым, одно и то же коловое слово? Смотрите, получая эту формулу, мы выбирали какие-то последовательности номеров позиций: пусть, для примера, это были 145, 154, 514, 541, 415, 451. Видно, что все эти последовательности дают одно и то же слово из нулей и единиц. И видно, что их 6. Чтобы снова получить этот факт не унылым перебором (каковой мы и произвели только что), а “весело и с умом”, надо рассудить так: из трех чисел 1, 4, 5 мы можем сперва выбрать любое (3 варианта); выбрав это число, мы можем вслед за ним расположить второе уже только одним из двух способов; третье же число выбирается однозначно. Рассуждение дает нужный результат: число способов упорядочить числа 1,4,5 равно $3\cdot2\cdot1 = 6$. Аналогично для любого k возникает формула $k\cdot(k-1)\cdot\ldots\cdot2\cdot1 = k!$. Напомним, по принятой в математике конвенции $0! = 1$.

Что же мы имеем в итоге? Сначала у нас возникла формула $n(n-1)(n-2)\cdot\ldots\cdot(n-k+1)$. Потом мы сообразили, что в ней каждое множество позиций для единиц учтено $k!$ раз. Это означает, что искомое количество кодовых слов равно

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

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 $\binom{n}{k}$.

10.3 Приложения к Главе 5

В качестве дополнительного материала к этой главе рекомендуем книгу Андрея "Модели случайных графов" [4]. Там приводится в подробностях доказатльство теоремы Эрдеша-Реньи и много других интересных результатов из теори случайных графов.

10.3.1 Вероятность потери связи в мини-сети

Рассмотрим мини-сеть как в примере выше: 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

$$\underbrace{p}_{\text{канал 1-2}} \cdot \underbrace{p}_{\text{канал 1-3}} \cdot \underbrace{p}_{\text{канал 2-3}} = p^3. \tag{10.3}$$

Вероятности перемножаются потому, что каналы независимы друг от друга. Допустим, у нас тысяча мини-сетей и $p = 0.6$. Тогда примерно в 600 из этих сетей будет доступен канал 1-2. Поскольку доступность канала 1-2 никак не влияет на канал 1-3, то из наших 1000 сетей оба канала 1-2 и 1-3 будут доступны в среднем $600 \cdot 0.6 = 360$ случаев. Чтобы получить среднее количество сетей, в которых доступны все три канала, надо взять долю 0.6 от 360, получаем $360 \cdot 0.6 = 216$. В результате, вероятность доступности всех трех каналов равна

$$\frac{260}{1000} = 0.216 = 0.6 \cdot 0.6 \cdot 0.6.$$

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

$$\begin{aligned}
&\underbrace{(1-p)}_{\text{канал 1-2}} \cdot \underbrace{p}_{\text{канал 1-3}} \cdot \underbrace{p}_{\text{канал 2-3}} + \underbrace{p}_{\text{канал 1-2}} \cdot \underbrace{(1-p)}_{\text{канал 1-3}} \cdot \underbrace{p}_{\text{канал 2-3}} + \\
&+ \underbrace{p}_{\text{канал 1-2}} \cdot \underbrace{p}_{\text{канал 1-3}} \cdot \underbrace{(1-p)}_{\text{канал 2-3}} + \\
&= 3p^2(1-p).
\end{aligned} \tag{10.4}$$

Общая вероятность сохранения связи в сети теперь равна

$$(10.3) + (10.4) = p^3 + 3p^2(1-p) = 3p^2 - 2p^3.$$

Потеря связи хотя бы с одним из компьтеров 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 мастами. Соответствующая вероятность равна

$$3(p-1)^2 p. \tag{10.5}$$

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

$$(1-p)^3. \tag{10.6}$$

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

$$(10.5) + (10.6) = 3(1-p)^2 - 2(1-p)^3. \tag{10.7}$$

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

$$\begin{aligned}(10.5) + (10.6) + (10.5) + (10.6) &= p^3 + 3p^2(1-p) + 3(p-1)^2 p + (1-p)^3 \\ &= \big(p + (1-p)\big)^3 = 1^3 = 1.\end{aligned}$$

Если провести еще немножко алгебраических манипуляций, то можно переписать вероятность (10.7) по-другому:

$$\begin{aligned}3(1-p)^2 - 2(1-p)^3 &= (1-p)^2\big(3 - 2(1-p)\big)\big) = (1-p)^2(1+2p) \\ &= (1-p)\big((1-p)(1+2p)\big) = (1-p)(1+p-2p^2).\end{aligned}$$

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

Кроме того, выражение (10.7) 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 принебречь. 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 Таблицы 5.1.

10.3.2 Теорема Эрдеша-Реньи о фазовом переходе.

Теорема (Эрдеш-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*

$$q(n) = 1 - \frac{c \ln n}{n}. \tag{10.8}$$

(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 Разделе 5.4.

(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, но к числу $e^{-1} \approx 0.3679$, где $e = 2.71828...$ — основание натурального логарифма.

10.3.3 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 эта вершина не соединена ребром ни с одной из оставшихся $n - 1$ вершин, равна:

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

Поскольку всего вершин n, то в среднем число вершин, которые не соеденины ребрами ни с одной другой вершиной, равно:

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

Возьмем, как и прежде,

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

Вспомните один известный замечательный предел $\lim_{n\to\infty}(1 + \frac{1}{n})^n = e$, где e – это 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 $c > 1$, 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 $0, 1, 2 \ldots$. Можно доказать, что соответствующее распределение вероятности близко к закону Пуассона с параметром 1:

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

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

Заметим, 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 $c > 1$ количество вершин без ребер относительно мало 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, которые стояли у истоков теории вероятностей.

10.4 Приложения к Главе 6

10.4.1 Анализ метода выбора из двух

Допустим, что у нас n серверов. Заявки (или задания) поступают с интенсивностью λn в единицу времени, и каждый сервер в среднем обрабатывает одно здание в единицу времени, то есть загрузка системы равна λ. Рассмотрим случай, когда n очень велико, 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:

$$u_k = f_k + f_{k+1} + \cdots , \quad f_k = u_k - u_{k+1}.$$

Понятно, что $u_0 = 1$.

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

$$nf_k = n(u_k - u_{k+1}) \tag{10.9}$$

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 $n(u_k - u_{k+1})$ 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 $k - 1$ заявок. При методе из двух вероятность того, что новое задание попадет на сервер с k или больше заявками равна

$$u_k^2, \tag{10.10}$$

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. [1] Значит, вероятность того, что новая заявка поступит на сервер, у которого ровно $k-1$ заявок, равна

$$u_{k-1}^2 - u_k^2.$$

Поскольку заявок в единицу времени поступает λn, то получаем, что число серверов с k или больше заявками в единицу времени в среднем *увеличивается* на

$$\lambda n(u_{k-1}^2 - u_k^2). \tag{10.11}$$

Поскольку система в равновесии, то число серверов с k или больше заявками должно оставаться неизменным, то есть (10.9) равняется (10.11). Отсюда мы получаем уравнение баланса:

$$u_k - u_{k+1} = \lambda(u_{k-1}^2 - u_k^2),\ k = 1, 2, \ldots; \quad u_0 = 1. \tag{10.12}$$

Результат в работе [27] говорит, что при определенных общепринятых предположениях о законе поступления заданий и времени их выполнения, в пределе для бесконечного количества серверов, уравнение (10.12) действительно описывает равновесное состояние системы. Это достаточно сложный технический результат. Нетрудно проверить, или даже догадаться, что решение уравнения (10.12) задается формулами:

$$u_k = \lambda^{2^k-1}, \quad f_k = \lambda^{2^k-1} - \lambda^{2^{k+1}-1}, \quad \text{для всех } k = 0, 1, 2, \ldots. \tag{10.13}$$

Это так называемый "двойной экспоненциальный закон". Двойка в формуле – эта та самая двойка – вторая степень – из выражения (10.10). Точно так же мы могли бы выбирать не из двух, а из r серверов, и полуили бы $u_k = \lambda^{r^k-1}$

Интересно заметить, что при тех же предположениях, но случайном выборе одного сервера, изменятся выражения (10.10) и соответственно (10.11). Действительно, на этот раз заявка выбирает только один сервер, и вероятность попасть на сервер с k или больше заявками равно просто u_k. Тогда вместо (10.12) мы получаем классическое уравнение баланса:

$$u_k - u_{k+1} = \lambda(u_{k-1} - u_k),\ k = 1, 2, \ldots; \quad u_0 = 1,$$

решение которого задается известной формулой Эрланга:

$$u_k = \lambda^k, \quad f_k = (1-\lambda)\lambda^k, \quad \text{для всех } k = 0, 1, 2, \ldots. \tag{10.14}$$

Очевидно, что u_k в формуле (10.13) убывает гораздо быстрее.

Именно эти формулы мы использовали в Таблице 6.1. В нашем случае $\lambda = 0.9$, и в таблице мы привели значения f_k. В первой колонке – значания k. Во второй колонке – значения f_k, подсчитанные по формуле (10.14). В третьей колонке – значения f_k, подсчитанные по формуле (10.13).

[1] На самом деле, если нужно выбрать два *разных* сервера, то эта вероятность равна $u_k n(u_k n - 1)/n^2$, но это значение очень близко к u_k^2, когда n достаточно велико.

10.5 Приложения в Главе 7

10.5.1 Схема Диффи-Хеллмана

Для начала введем обозначения. Пусть p звдвнное 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 разделах 10.5.2,10.5.3. 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 стандарным обозначением для остатка от деления n на p:

$$n \pmod p = [\text{остаток от деления } n \text{ на } p].$$

(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, что

$$a = g^x - kp,$$

$$b = g^y - lp.$$

Подставив эти выражения в формулы для ключей, мы получаем:

$$K_A = ((g^y - lp)^x) \pmod p,$$

$$K_B = ((g^x - kp)^y) \pmod p.$$

Заметим, что в выражении для K_A мы можем расписать $(g^y - lp)^x$ следующим образом:

$$(g^y - lp)^x = \underbrace{(g^y - lp)(g^y - lp)\cdots(g^y - lp)}_{x \text{ раз}} = (g^y)^x + pA,$$

где A – это целое число, то есть pA делится на p. Таким образом получаем, что

$$K_A = ((g^y - lp)^x) \pmod p = ((g^y)^x + pA) \pmod p = (g^y)^x \pmod p.$$

Совершенно аналогично, для какого-то целого числа B, мы получаем

$$K_B = ((g^x - kp)^y) \pmod p = ((g^x)^y + pB) \pmod p = ((g^x)^y) \pmod p.$$

Результат теперь очевиден, поскольку

$$(g^y)^x = g^{yx} = g^{xy} = (g^x)^y.$$

10.5.2 Дискретное логарифмирование

Вспомним, что логарифм числа y по основанию g – это такое число x, что

$$g^x = y.$$

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, [6]) утверждает, что для каждого простого p *существует* такое число g, что все числа

$$g^1 \pmod p, \quad g^2 \pmod p, \quad \ldots, \quad g^{p-1} \pmod p \tag{10.15}$$

разные, т.е. они 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 $y \in \{1, 2, \ldots, p-1\}$ по основанию g — это тот самый, уникальный, $x \in \{1, 2, \ldots, p-1\}$, при которым выполняется $g^x \pmod p = y$. Поскольку для всех $x < p$ остатки разные, то x определяется однозначно. Операция нахождения такого x называется *операция дискретного логарифмирования*. Именно она сложна, и никто не знает, можно ли придумать способ ее быстрой реализации.

Как определить такое число g, чтобы все остатки в (10.15) были разные? Число g обладает этим свойством, если оно является *первообразным корнем* числа p. Мы позволим себе рассказать об этом понятии немножко поподробнее.

10.5.3 Первообразные корни

Есть такая теорема Эйлера: если x и m взаимно просты, то $x^{\varphi(m)} \equiv 1 \pmod m$. 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.

Условия 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 такого 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 это 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 качстве m удобно взять простое число p. Для простого p всегда существуют первообразные корни. На самом деле мы уже говорили о них выше, в Разделе 10.5.2, только не называли их этим термином. Это те самые числа g, которые удовлетворяют условию (10.15). 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 $p = 19$.

Если g – первообразный корень p, то все остатки в (10.15) 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 мы 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.

10.6 Приложения к Главе 8

10.6.1 Двойной логарифм в *HyperLogLog*

Хеш-значения – это последовательности из нулей и единиц одинаковой длины. Обозначим длину хеш-значений через m. Тогда мы получим 2^m разных хеш-значений (см. Главу 4 и Приложение 10.2.1).

Теперь допустим, нам нужно сосчитать n различных объектов. Чтобы присвоить им всем разные хеш-значения, нам понадобится как минимум n хеш-значений, то есть

$$2^m > n \quad \text{или} \quad m > \log_2(n).$$

Значит, 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. Это число 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. Значит, чтобы записать $m+1$ разных чисел, должно выполняться

$$2^l = m+1 \quad \text{или} \quad l = \log_2(m+1) \approx \log_2(m).$$

В памяти мы должны держать только l битов информации (последовательность из нулей и единиц длины l). Из предыдущих формул получается:

$$l \approx \log_2(m) \approx \log_2(\log_2(n)).$$

Для улучшения точности, хеш-значения разбивают на r регистров и запоминают число нулей в каждом регистре отдельно. В результате требуется порядка $r \log_2(\log_2(n))$ битов памяти. Относительная точность оценки задается формулой $1.04/\sqrt{r}$ [16].

10.7 Приложения к Главе 9

10.7.1 Доказательство совместимости по стимулам аукциона второй цены

Рассмотрим аукцион второй цены, на котором один товар разыгрывается между n участниками. Обозначим через v_i истинную ценность товара для участика i (от английского слова *value*). Далее, обозначим через b_i ставку участика i (от английского слова *bid*). Эти обозначения мы будем испольовать для любого $i = 1, 2, \ldots, n$.

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, то, значит, 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 ценость товара для участника i равна v_i. Значит, если участник i получает товар, то его прибыль равна

$$v_i - \max_{j \neq i} b_j. \tag{10.16}$$

Если участник i товара не получает, то он не приобретает никакой ценности и ничего не платит, то есть его прибыль равна нулю.

Дальше доказательство ведется так же, как в Разделе 9.5, и в качестве иллюстрации мы можем по-прежнему использовать Рисунки 9.2 и 9.3. Мы покажем, при правдивой ставке $b_i = v_i$ прибыль участника i та же или больше, чем чем при повышенной ставке $b_i > v_i$ или при пониженной ставке $b_i < v_i$.

Предположим, что $b_i > v_i$. Рассмотрим три случая.

(i) Допустим, что b_i – самая высокая ставка, и, кроме того, все остальные участники *поставили меньше, чем* v_i (см. Рисунок 9.2 сверху). Тогда товар по-прежнему достается участнику i по стоимости $\max_{j \neq i} b_j$, и участник i получает точно такую же прибыль (10.16), что и при правдивой ставке v_i.

(ii) Теперь допустим, что другой участник k сделал ставку $b_k > b_i$ (см. Рисунок 9.2 в середине). Тогда участник i товара не получает, его прибыль равна нулю. Но поскольку $b_i > v_i$, то и при правдивой ставке v_i участник i тоже не получил бы товара. То есть, в этом случае, прибыль участника i при правдивой ставке тоже равна нулю.

(iii) Наконец, допустим, что b_i – самая высокая ставка, и $v_i < \max_{j \neq i} b_j < b_i$ (см. Рисунок 9.2 снизу). Поскольку $v_i < b_i$, то 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 товара, прибыль была бы равна нулю. Но теперь b_i – самая высокая ставка, поэтому товар достается i. Прибыль i по-прежнему вычисляется по формуле (10.16), но только прибыль становится отрицательной, поскольку ценность товара меньше его стоимости. Значит, в этом случае прибыль i меньше, чем при правдивой ставке.

Во всех трех случаях (i)–(iii), участник i не получил прибыль выше, чем при правдивой ставке v_i.

Теперь предположим, что $b_i < v_i$, то есть ставка занижает реальную ценность. Опять рассмотрим три случая.

(i') Допустим, b_i – самая высокая ставка, и все остальные участники *поставили меньше, чем* b_i (см. Рисунок 9.3 сверху). Тогда товар по-прежнему достается участнику i по стоимости $\max_{j \neq i} b_j$. В этом случае прибыль участника i по-прежнему (10.16). Эта прибыль в точности такая же, как и при правдивой ставке v_i.

(ii') Теперь допустим, что другой участник l сделал ставку $b_l > v_i$ (см. Рисунок 9.3 в середине). В этом случае, при правдивой ставке v_i участник i товара не получает.

Но тогда и при заниженной ставке $b_i < v_i$ участник i тоже не получит товара. Значит, прибыль i равна нулю и при правдивой, и при заниженной ставке.

(iii') Наконец, допустим, что $b_i < \max_{j \neq i} b_j < v_i$ (см. Рисунок 9.3 снизу). Поскольку $b_i < v_i$, то такая ситуация возможна. Она возникает, когда самая высокая ставка других участников выше b_i, но ниже v_i. Тогда при правдивой ставке товар достался бы участнику i, и его прибыль, по формуле (10.16), была бы положительной. Но поскольку b_i теперь не самая высокая ставка, то товар достанется другому участнику, и прибыль v_i равна нулю. Значит, в этом случае прибыль i меньше, чем при правдивой ставке.

Во всех трех случаях (i')–(iii'), участник i не получил прибыль выше, чем при правдивой ставке v_i.

В результате мы делаем вывод, что и при заниженной, и при завышенной ставке, участник i получает не больше, чем при правдивой ставке $b_i = v_i$. Таким образом, мы доказали, что в аукционе второй цены выгоднее всего делать честную ставку.

Глава 11

Литература

[1] Н.Дж.А. Слоэн Ф.Дж. Мак-Вильямс. *Теория кодов, исправляющих ошибки.* М.: Радио и связь, 1979.

[2] Дж. Спенсер Н. Алон. *Вероятностный метод.* Москва: Бином. Лаборатория знаний, 2007.

[3] 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.

[4] А. М. Райгородский. *Модели случайных графов.* МЦНМО, второе издание, 2016.

[5] ЛВ Канторович. Об одном эффективном методе решения некоторых классов экстремальных проблем. In *Докл. АН СССР*, volume 28, pages 212–215, 1940.

[6] И.М. Виноградов. *Основы теории чисел.* НИЦ "Регулярная и хаотическая динамика Москва - Ижевск, 2003.

[7] А. Савватеев. Теория аукционов. Наиболее значимые достижения. `https://www.youtube.com/watch?v=fCIU07zg9HQ&feature=youtu.be`, 2013. Accessed: 2016-02-09.

[8] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, et al. Imperfect forward secrecy: How diffie-hellman fails in practice. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 5–17. ACM, 2015.

[9] Réka Albert, Hawoong Jeong, and Albert-László Barabási. Error and attack tolerance of complex networks. *nature*, 406(6794):378–382, 2000.

[10] Lars Backstrom, Paolo Boldi, Marco Rosa, Johan Ugander, and Sebastiano Vigna. Four degrees of separation. In *Proceedings of the 4th Annual ACM Web Science Conference*, pages 33–42. ACM, 2012.

[11] Robert E Bixby. A brief history of linear and mixed-integer programming computation. *Documenta Mathematica*, pages 107–121, 2012.

[12] J.C. Doyle, D.L. Alderson, L. Li, S. Low, M. Roughan, S. Shalunov, R. Tanaka, and W. Willinger. The "robust yet fragile" nature of the Internet. *PNAS*, 102(41):14497–14502, 2005.

[13] Derek L Eager, Edward D Lazowska, and John Zahorjan. Adaptive load sharing in homogeneous distributed systems. *Software Engineering, IEEE Transactions on*, (5):662–675, 1986.

[14] David Easley and Jon Kleinberg. *Networks, crowds, and markets: Reasoning about a highly connected world*. Cambridge University Press, 2010.

[15] Paul Erdös and Alfréd Rényi. {On the evolution of random graphs}. *Publ. Math. Inst. Hung. Acad. Sci*, 5:17–61, 1960.

[16] Philippe Flajolet, Éric Fusy, G Olivier, and Frédéric Meunier. Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm. In *In AofA'07: Proceedings of the 2007 International Conference on Analysis of Algorithms*. Citeseer, 2007.

[17] Stefan Heule, Marc Nunkesser, and Alexander Hall. Hyperloglog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm. In *Proceedings of the 16th International Conference on Extending Database Technology*, pages 683–692. ACM, 2013.

[18] Charles C Holt. Learning how to plan production, inventories, and work force. *Operations Research*, 50(1):96–99, 2002.

[19] Peter Keevash. The existence of designs. *arXiv preprint arXiv:1401.3665*, 2014.

[20] Leo Kroon, Dennis Huisman, Erwin Abbink, Pieter-Jan Fioole, Matteo Fischetti, Gábor Maróti, Alexander Schrijver, Adri Steenbeek, and Roelof Ybema. The new dutch timetable: The or revolution. *Interfaces*, 39(1):6–17, 2009.

[21] P. Mahadevan, D. Krioukov, K. Fall, and A. Vahdat. Systematic topology analysis and generation using degree correlations. *ACM SIGCOMM Computer Communication Review*, 36(4):135–146, 2006.

[22] A Renyi and P Erdos. On random graphs. *Publicationes Mathematicae*, 6(290-297):5, 1959.

[23] Andrea W Richa, M Mitzenmacher, and R Sitaraman. The power of two random choices: A survey of techniques and results. *Combinatorial Optimization*, 9:255–304, 2001.

[24] Claude Elwood Shannon. A mathematical theory of communication. *The Bell System technical Journal*, 27:379–423, 623–656, 1949.

[25] Donald E Stokes. *Pasteur's quadrant: Basic science and technological innovation*. Brookings Institution Press, 2011.

[26] William Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *The Journal of finance*, 16(1):8–37, 1961.

[27] Nikita Dmitrievna Vvedenskaya, Roland L'vovich Dobrushin, and Fridrikh Izrailevich Karpelevich. Queueing system with selection of the shortest of two queues: An asymptotic approach. *Problemy Peredachi Informatsii*, 32(1):20–34, 1996.