ОГЛАВЛЕНИЕ

ГЛАВА 2.
MapReduce и новый программный стек 39

ПРЕДИСЛОВИЕ

В основу этой книги положен материал односеместрового курса, который Ананд Раджараман и Джефф Ульман в течение нескольких лет читали в Стэнфордском университете. Курс CS345A под названием «Добыча данных в вебе» задумывался как спецкурс для аспирантов, но оказался доступным и полезным также старшекурсникам. Когда в Стэнфорд пришел преподавать Юре Лесковец, мы существенно изменили организацию материала. Он начал читать новый курс CS224W по анализу сетей и расширил материал курса CS345A, который получил номер CS246. Втроем авторы также подготовили курс CS341, посвященный крупномасштабному проекту в области добычи данных. В своем теперешнем виде книга содержит материал всех трех курсов.

О чем эта книга

В самых общих словах, эта книга о добыче данных. Но акцент сделан на анализе данных очень большого объема, не помещающихся в оперативную память. Поэтому многие примеры относятся к вебу или к данным, полученным из веба. Кроме того, в книге принят алгоритмический подход: добыча данных – это применение алгоритмов к данным, а не использование данных для «обучения» той или иной машины. Ниже перечислены основные рассматриваемые темы.

1. Распределенные файловые системы и технология распределения-редукции (map-reduce) как средство создания параллельных алгоритмов, успешно справляющихся с очень большими объемами данных.
2. Поиск по сходству, в том числе такие важнейшие алгоритмы, как MinHash и хэширование с учетом близости (locality sensitive hashing).
3. Обработка потоков данных и специализированные алгоритмы для работы с данными, которые поступают настолько быстро, что либо обрабатываются немедленно, либо теряются.
4. Принципы работы поисковых систем, в том числе алгоритм Google PageRank, распознавание ссылочного спама и метод авторитетных и хаб-документов.
5. Частые предметные наборы, в том числе поиск ассоциативных правил, анализ корзины, алгоритм Apriori и его усовершенствованные варианты.
6. Алгоритмы кластеризации очень больших многомерных наборов данных.
7. Две важные для веб-приложений задачи: управление рекламой и рекомендательные системы.

8. Алгоритмы анализа структуры очень больших графов, в особенности графов социальных сетей.
9. Методы получения важных свойств большого набора данных с помощью понижения размерности, в том числе сингулярное разложение и латентно-семантическое индексирование.
10. Алгоритмы машинного обучения, применимые к очень большим наборам данных, в том числе перцептроны, метод опорных векторов и градиентный спуск.

Требования к читателю

Для полного понимания изложенного в книге материала мы рекомендуем:

1. Прослушать вводный курс по системам баз данных, включая основы SQL и сопутствующих систем программирования.
2. Иметь знания о структурах данных, алгоритмах и дискретной математике в объеме второго курса университета.
3. Иметь знания о программных системах, программной инженерии и языках программирования в объеме второго курса университета.

Упражнения

В книге много упражнений, они есть почти в каждом разделе. Более трудные упражнения или их части отмечены восклицательным знаком, а самые трудные – двумя восклицательными знаками.

Поддержка в вебе

Слайды, домашние задания, проектные требования и экзаменационные задачи из курсов, примыкающих к этой книге, можно найти по адресу `http://www.mmds.org`.

Автоматизированные домашние задания

На основе этой книги составлены автоматизированные упражнения с применением системы проверочных вопросов Gradiance, доступной по адресу `www.gradiance.com/services`. Студенты могут стать членами открытой группы, создав на этом сайте учетную запись и присоединившись к группе с кодом `1EDD8A1D`. Преподаватели также могут воспользоваться этим сайтом, для этого нужно создать учетную запись и отправить сообщение на адрес `support@gradiance.com`, указав в нем свой логин, название учебного заведения и запрос на право использования материалов к книге (MMDS).

ГЛАВА 1. Добыча данных

В этой вводной главе мы опишем, в чем состоит сущность добычи данных, и обсудим, как добыча данных трактуется в различных дисциплинах, которые вносят свой вклад в эту область. Мы рассмотрим «принцип Бонферрони», предупреждающий об опасностях чрезмерного увлечения добычей данных. В этой же главе мы кратко упомянем некоторые идеи, которые, хотя сами и не относятся к добыче данных, но полезны для понимания ряда важных идей, относящихся к этой тематике. Мы имеем в виду метрику важности слов TF.IDF, поведение хэш-функций и индексов, а также некоторые тождества, содержащие число e, основание натуральных логарифмов. Наконец, мы расскажем о темах, рассматриваемых в этой книге.

1.1. Что такое добыча данных?

Многие разделяют определение «добычи данных» как выявление «моделей» данных. Однако под моделью можно понимать разные вещи. Ниже описываются наиболее важные направления моделирования.

1.1.1. Статистическое моделирование

Первыми термин «добыча данных» ввели в обиход специалисты по математической статистике. Первоначально словосочетание «data mining» (добыча данных) или «data dredging» (вычерпывание данных) имело несколько пренебрежительный оттенок и обозначало попытки извлечь информацию, которая явно не присутствовала в данных. В разделе 1.2 демонстрируются различные ошибки, которые могут возникнуть, если пытаться извлечь то, чего в данных на самом деле нет. В наши дни термин «добыча данных» употребляется в положительном смысле. Теперь статистики рассматривают добычу данных как средство построения статистической модели, т. е. закона, в соответствии с которым распределены видимые данные.

Пример 1.1. Пусть данными будет множество чисел. Эти данные намного проще тех, что подвергаются добыче, но для примера вполне подойдут. Статистик может предположить, что данные имеют гауссово распределение и по известным формулам вычислить наиболее вероятные параметры этого распределения.

Среднее и стандартное отклонение полностью определяют гауссово распределение и потому могут служить моделью данных.

1.1.2. Машинное обучение

Некоторые считают, что добыча данных и машинное обучение – синонимы. Безусловно, для добычи данных иногда используются алгоритмы, применяемые в машинном обучении. Специалисты по машинному обучению используют данные как обучающий набор и на них обучают алгоритм того или иного вида, например: байесовские сети, метод опорных векторов, решающие деревья, скрытые марковские модели и т. п.

В некоторых ситуациях использование данных подобным образом имеет смысл. В частности, машинное обучение дает хороший результат, когда мы плохо представляем себе, что искать в данных. Например, совсем неясно, из-за каких особенностей одним людям фильм нравится, а другим – нет. Поэтому принявшие «вызов Netflix» – изобрести алгоритм, который предсказывал бы оценку фильма пользователями на основе выборки из их прошлых ответов, – с большим успехом применили алгоритмы машинного обучения. Мы обсудим простую форму алгоритма такого типа в разделе 9.4.

С другой стороны, машинное обучение не приносит успеха в ситуациях, когда цели добычи данных можно описать более конкретно. Интересный пример – попытка компании WhizBang! Labs[1] использовать методы машинного обучения для поиска резюме, которые люди размещают в сети. У нее не получилось добиться результатов, лучших, чем дают вручную составленные алгоритмы, которые ищут очевидные слова и фразы, встречающиеся в типичном резюме. Всякий, кто читал или писал резюме, довольно отчетливо представляет, что в нем содержится, поэтому как выглядит веб-страница, содержащая резюме, – никакая не тайна. Потому-то применение машинного обучение не дало выигрыша по сравнению с составленным в лоб алгоритмом распознавания резюме.

1.1.3. Вычислительные подходы к моделированию

Сравнительно недавно на добычу данных стали смотреть как на алгоритмическую задачу. В этом случае модель данных – это просто ответ на сложный запрос к данным. Например, если дано множество чисел, как в примере 1.1, то можно было бы вычислить их среднее и стандартное отклонение. Отметим, что эти значения необязательно являются параметрами гауссова распределения, которое лучше всего аппроксимирует данные, хотя при достаточно большом наборе данных они почти наверняка будут близки к ним.

Есть много подходов к моделированию данных. Мы уже упомянули одну возможность: построить статистический процесс, с помощью которого данные могли

[1] Эта компания пыталась использовать методы машинного обучения для анализа очень большого объема данных и наняла для этого много высококлассных специалистов. К сожалению, выжить ей не удалось.

быть сгенерированы. Большинство прочих подходов к моделированию можно отнести к одной из двух категорий.

1. Краткое и приближенное обобщение данных или
2. Извлечение из данных наиболее существенных признаков с отбрасыванием всего остального.

В следующих разделах мы исследуем оба подхода.

1.1.4. Обобщение

Одна из самых интересных форм обобщения – идея алгоритма PageRank, так успешно примененная Google; мы будем рассматривать ее в главе 5. При такой форме добычи данных вся сложная структура веба сводится к одному числу для каждой страницы. Несколько упрощая, это число, «ранг страницы» (PageRank), можно описать как вероятность того, что пользователь, случайно обходящий граф, окажется на этой странице в любой заданный момент времени. Замечательное свойство такого ранжирования заключается том, что оно очень хорошо отражает «важность» страницы – в какой мере типичный пользователь поисковой системы хотел бы видеть данную страницу в ответе на свой запрос.

Еще один важный вид обобщения – кластеризация – будет рассмотрен в главе 7. В этом случае данные рассматриваются как точки в многомерном пространстве. Те точки, которые в некотором смысле «близки», помещаются в один кластер. Сами кластеры также обобщаются, например, путем указания центроида кластера и среднего расстояния от центроида до всех точек. Совокупность обобщенных характеристик кластеров становится обобщением всего набора данных.

Пример 1.2. Знаменитый пример применения кластеризации для решения задачи имел место много лет назад в Лондоне, когда никаких компьютеров еще не было[2]. Врач Джон Сноу, сражаясь со вспышкой холеры, нанес места проживания заболевших на карту города. На рис. 1.1 показана упрощенная иллюстрация этой процедуры.

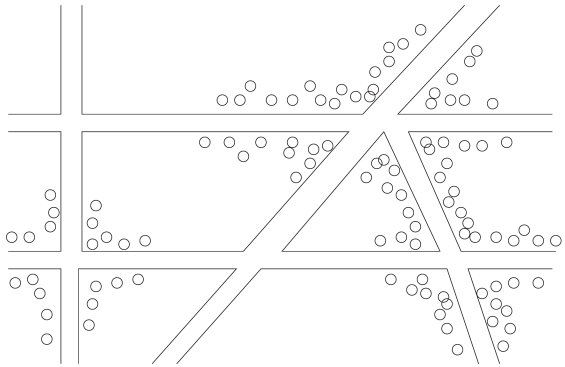

Рис. 1.1. Случаи холеры на карте Лондона

[2] См. http://en.wikipedia.org/wiki/1854 Broad Street cholera outbreak

Как видно, образовалось несколько кластеров в районе перекрестков. На этих перекрестках находились зараженные водоразборные колонки; жившие поблизости от них заболели, те же, кто жил рядом с незараженными колонками, остались здоровы. Не будь возможности кластеризовать данные, причина холеры осталась бы невыясненной.

1.1.5. Выделение признаков

В типичной модели на основе признаков ищутся экстремальные примеры некоторого явления, и данные представляются с помощью этих примеров. Если вы знакомы с *байесовскими сетями*, одной из ветвей машинного обучения, которая в этой книге не рассматривается, то знаете, что в них сложные связи между объектами представляются с помощью отыскания самых сильных статистических зависимостей и использования только их для представления всех статистических связей. Мы изучим следующие важные формы выделения признаков из больших наборов данных.

1. *Частые предметные наборы*. Эта модель имеет смысл, когда данные состоят из «корзин», содержащих небольшие наборы предметов, как, например, в задаче об анализе корзин покупок, обсуждаемой в главе 6. Мы ищем небольшие наборы предметов, которые встречаются вместе во многих корзинах, и считаем эти «частые предметные наборы» искомой характеристикой данных. Первоначально такой вид добычи данных применялся к настоящим корзинам покупок: поиску предметов, например гамбургер и кетчуп, которые люди покупают вместе в небольшой лавке или в супермаркете.
2. *Похожие предметы*. Часто данные имеют вид коллекции наборов, а цель состоит в том, чтобы найти пары наборов, в которых относительно много общих элементов. Например, покупателей в интернет-магазине типа Amazon можно рассматривать как наборы купленных ими товаров. Чтобы предложить покупателю еще что-нибудь, что могло бы ему понравиться, Amazon может поискать «похожих» покупателей и порекомендовать товары, которые покупали многие из них. Этот процесс называется «коллаборативной фильтрацией». Если бы все покупатели были целеустремленными, т. е. покупали бы только одну вещь, то могла бы сработать кластеризация покупателей. Но обычно покупателей интересуют разные вещи, поэтому полезнее для каждого покупателя найти небольшое число покупателей со схожими вкусами и представить данные такими связями. Проблему сходства мы будем обсуждать в главе 3.

1.2. Статистические пределы добычи данных

Типичная задача добычи данных – обнаружение необычных событий, скрытых в массивном объеме данных. В этом разделе мы рассмотрим эту проблему и заодно

«принцип Бонферрони» – предостережение против излишне ревностных попыток добыть данные.

1.2.1. Тотальное владение информацией

В 2002 году администрация Буша выдвинула план – подвергнуть анализу все данные, до которых можно дотянуться, в том числе чеки, оплаченные кредитной картой, данные о регистрации в гостиницах, данные о поездках и многие иные виды информации, – с целью отслеживания террористической деятельности. Эта идея, естественно, вызвала недовольство у поборников защиты частной жизни, и в итоге весь проект, названный TIA, или *Total Information Awareness* (*тотальное владение информацией*), был похоронен Конгрессом, хотя не исключено, что он все же существует под другим именем. В этой книге мы не собираемся обсуждать трудную проблему поиска компромисса между безопасностью и конфиденциальностью. Однако в связи с проектом TIA или подобной системой возникает ряд технических вопросов касательно практической осуществимости и реалистичности предположений.

Многие задавались вопросом: если исследовать так много данных, пытаясь найти следы деятельности, характерной для террористов, то не получится ли, что мы найдем много совершенно невинных действий – или даже незаконных, но не относящихся к терроризму, – и человеку придется свести знакомство с полицией, а, может, и не просто знакомство? Здесь все зависит от того, насколько узко определена интересующая нас деятельность. Статистики сталкивались с многообразными проявлениями этой проблемы и выдвинули теорию, начатки которой мы изложим в следующем разделе.

1.2.2. Принцип Бонферрони

Пусть имеются какие-то данные, и мы ищем в них события определенного вида. Можно ожидать, что такие событие встретятся, даже если данные выбраны абсолютно случайно, а количество событий будет расти вместе с объемом данных. Эти события «фиктивные» в том смысле, что у них нет никакой причины, помимо случайности данных, а в случайных данных всегда встретится какое-то количество необычных признаков, которые, хотя и выглядят значимыми, на самом деле таковыми не являются. Теорема математической статистики, известная под названием *поправка Бонферрони*, дает статистически корректный способ избежать большинства таких ложноположительных ответов на поисковый запрос. Не вдаваясь в технические детали, мы предложим ее неформальный вариант, *принцип Бонферрони*, который поможет избежать трактовки случайных фактов как реальных. Вычислите ожидаемое число искомых событий в предположении, что данные случайны. Если это число существенно больше количества реальных событий, которые вы надеетесь обнаружить, то следует ожидать, что почти все найденные события фиктивные, т. е. являются статистическими артефактами, а не свидетельством в пользу того, что вы ищете. Это наблюдение и есть неформальный принцип Бонферрони.

В случае поиска террористов, когда мы ожидаем, что сколько-то террористов действуют в любой момент времени, принцип Бонферрони гласит, что обнаружить террористов можно, только выискивая события настолько редкие, что в случайных данных их появление крайне маловероятно. Развернутый пример мы приведем в следующем разделе.

1.2.3. Пример применения принципа Бонферрони

Допустим, мы полагаем, что где-то действуют «злоумышленники», и хотим их обнаружить. Допустим также, что есть основания полагать, что злоумышленники периодически встречаются в гостинице, чтобы спланировать свой злой умысел. Сделаем следующие предположения о размере задачи:

1. Есть миллиард людей, среди которых могут быть злоумышленники.
2. Любой человек останавливается в гостинице один день из 100.
3. Гостиница вмещает 100 человек. Следовательно, 100 000 гостиниц будет достаточно, чтобы разместить 1 % от миллиарда людей, которые останавливаются в гостинице в каждый конкретный день.
4. Мы изучаем данные о регистрации в гостиницах за 1000 дней.

Чтобы найти в этих данных злоумышленников, мы будем искать людей, которые в два разных дня останавливались в одной и той же гостинице. Допустим, однако, что в действительности никаких злоумышленников нет. То есть все ведут себя случайным образом, с вероятностью 0,01 решая в данный день остановиться в какой-то гостинице и при этом случайно выбирая одну из 10^5 гостиниц. Найдем ли мы пары людей, которые выглядят как злоумышленники? Можно выполнить простое вычисление. Вероятность того, что два произвольных человека решат остановиться в гостинице в данный день, составляет 0,0001. Вероятность того, что они остановятся в одной и той же гостинице в один и тот же день равна 10^{-9}. Вероятность, что они остановятся в одной и той же гостинице в два разных дня, равна квадрату этого числа, т. е. 10^{-18}. Отметим, что выбранные в эти дни гостиницы могут быть разными.

Теперь надо посчитать, сколько событий указывают на злой умысел. Под «событием» здесь понимается пара людей и пара дней такие, что оба человека в каждый из этих двух дней останавливались в одной и той же гостинице. Чтобы упростить вычисления, заметим, что для больших n, $\binom{n}{2}$ приблизительно равно $n^2/2$. Таким образом, количество пар людей равно $\binom{10^9}{2} = 5 \times 10^{17}$. Количество пар дней равно $\binom{1000}{2} = 5 \times 10^5$. Ожидаемое число событий, выглядящих как злоумышление, равно произведению количества пар людей на количество пар дней и на вероятность того, что пара людей и пара дней демонстрируют искомое поведение. Это число равно

$$5 \times 10^{17} \times 5 \times 10^5 \times 10^{-18} = 250\,000.$$

То есть четверть миллиона людей будут казаться злоумышленниками, даже если не являются таковыми.

Теперь предположим, что в действительности существует 10 пар злоумышленников. Полиции придется проверить четверть миллиона других пар, чтобы найти настоящих злоумышленников. Мало того что это означает вторжение в частную жизнь полумиллиона ни в чем неповинных людей, так еще и объем работы настолько велик, что такой подход к поиску злоумышленников практически неосуществим.

1.2.4. Упражнения к разделу 1.2

Упражнение 1.2.1. Используя сведения из раздела 1.2.3, найдите количество подозрительных пар, если внести в данные следующие изменения (сохранив все прочие числа)?

(*а*) Увеличить количество дней наблюдения до 2000.

(*б*) Увеличить количество наблюдаемых людей до 2 миллиардов (а количество гостиниц соответственно до 200 000).

(*с*) Считать двух человек подозрительными, если они останавливались в одной и той же гостинице в три разных дня.

! **Упражнение 1.2.2.** Предположим, что у нас есть информация о покупках 100 миллионов людей в супермаркетах. Каждый человек заходит в супермаркет 100 раз в год и покупает 10 из 1000 предлагаемых там товаров. Мы думаем, что двое террористов в какой-то день на протяжении года купят в точности один и тот же набор предметов (быть может, компонентов бомбы). Если мы будем искать пары людей, купивших одинаковые наборы предметов, то можно ли ожидать, что найденные люди действительно террористы[3]?

1.3. Кое-какие полезные сведения

В этом разделе содержится краткое введение в темы, с которыми вы, возможно, знакомились на других курсах. Все эти сведения будут полезны при изучении добычи данных.

1. Мера важности слов TF.IDF.
2. Хэш-функции и их применение.
3. Внешняя память (диск) и ее влияние на время работы алгоритмов.
4. Основание натуральных логарифмов *e* и тождества, содержащие эту константу.
5. Степенные зависимости.

1.3.1. Важность слов в документах

В нескольких приложениях добычи данных мы столкнемся с проблемой классификации документов (последовательностей слов) по тематике. Как правило, тема определяется путем поиска специальных слов, которые характеризуют относя-

[3] То есть наша гипотеза состоит в том, что террористы наверняка купят набор из десяти одинаковых предметов в какой-то день на протяжении года. Мы не хотим обсуждать вопрос о том, характерно ли такое поведение для настоящих террористов.

щиеся к ней документы. Например, в статьях о бейсболе будут часто встречаться слова «мяч», «бита», «бросок», «пробежка» и т. д. После того как документы отнесены к теме бейсбола, нетрудно заметить, что подобные слова встречаются в них аномально часто. Но пока классификация не произведена, выделить эти слова как характеристические невозможно.

Таким образом, классификация часто начинается с изучения документов и отыскания в них важных слов. Первая гипотеза может состоять в том, что слова, которые чаще всего встречаются в документе, и есть самые важные. Но в данном случае интуиция подсказывает ответ, прямо противоположный истинному положению дел. Самыми частыми, конечно же, будут наиболее употребительные слова типа «the» и «and», которые помогают выразить мысль, но сами по себе не несут никакого смысла. И действительно, несколько сотен наиболее употребительных слов английского языка (они называются *стоп-словами*), обычно исключаются из документов еще до попытки классификации.

На самом деле, индикаторами темы являются относительно редко встречающиеся слова. Но не все редкие слова одинаково полезны в качестве индикаторов. Некоторые слова, например «notwithstanding» (несмотря на) или «albeit» (пусть даже), хоть редко встречаются в коллекции документов, но ничего полезного не сообщают. С другой стороны, слово «chukker» (период в игре в поло), пожалуй, встречается не менее редко, но подсказывает, что данный документ посвящен игре в поло. Разница между значимыми и незначимыми редкими словами определяется концентрацией полезных слов в немногих документах. То есть присутствие в документе слова типа «albeit» не повышает вероятность его повторного появления. Но если в статье один раз встречается слово «chukker», то весьма вероятно, что оно входит в составе словосочетания «first chukker» (первый период), еще раз в «second chukker» (второй период) и т. д. То есть, если слово вообще встречается, то с большой вероятностью оно встретится несколько раз.

Формальная мера концентрации данного слова в относительно небольшом количестве документов называется TF.IDF (частота терма, помноженная на обратную частоту документа). Обычно она вычисляется следующим образом. Пусть есть коллекция из N документов. Обозначим f_{ij} *частоту* (количество вхождений) терма (слова) i в документ j и определим *частоту терма* TF_{ij} такой формулой:

$$TF_{ij} = \frac{f_{ij}}{\max_k f_{kj}}.$$

Иначе говоря, частота терма i в документе j равна величине f_{ij}, нормированной путем деления на максимальное количество вхождений этого терма (возможно, после исключения стоп-слов) в один и тот же документ. Следовательно, у самого часто встречающего терма в документе j TF будет равно 1, а у всех остальных меньше.

IDF терма определяется следующим образом. Пусть терм i встречается в n_i документах из коллекции, содержащей всего N документов. Тогда $IDF_i = \log_2(N/n_i)$. Оценка TF.IDF для терма i в документе j определяется как $TF_{ij} \times IDF_i$. Именно тер-

мы с наибольшей оценкой TF.IDF часто наилучшим образом характеризуют тему документа.

Пример 1.3. Предположим, что репозиторий содержит 2^{20} = 1 048 576 документов. Пусть слово *w* встречается в 2^{10} = 1024 документов. Тогда $IDF_w = \log_2(2^{20}/2^{10}) = \log_2(2^{10}) = 10$. Рассмотрим документ *j*, в котором терм *w* встречается 20 раз, и пусть это максимальное количество вхождений одного слова (возможно, после исключения стоп-слов). Тогда $TF_{wj} = 1$ и оценка TF.IDF терма *w* в документе *j* равна 10.

Предположим, что в документе *k* слово *w* встречается один раз, тогда как максимальное количество вхождений одного слова в этом документе равно 20. Тогда $TF_{wk} = 1/20$, а оценка TF.IDF для *w* в документе *k* равна 1/2.

1.3.2. Хэш-функции

Вы, вероятно, слышали о хэш-таблицах и, возможно, использовали их в Java-классах или других подобных пакетах. Хэш-функции, лежащие в основе хэш-таблиц, находят важное применение и во многих алгоритмах добычи данных, в которых хэш-таблицы принимают необычную форму. В этом разделе мы рассмотрим основные понятия.

Прежде всего, хэш-функция *h* принимает *ключ хэширования* в качестве аргумента и возвращает *номер ячейки* (*bucket*). Номер ячейки – это целое число, обычно в диапазоне от 0 до $B - 1$, где B – количество ячеек. Тип ключа хэширования может быть любым. На интуитивном уровне хэш-функция «рандомизирует» ключи хэширования. Точнее, если ключи хэширования случайным образом выбираются из разумной совокупности возможных ключей, то *h* поместит в каждую из B ячеек примерно одинаковое количество ключей. Это было бы невозможно, если, к примеру, размер совокупности ключей хэширования меньше B. Такая совокупность не считается «разумной». Однако есть немало более тонких причин, по которым хэш-функция может не давать приблизительно равномерного распределения по ячейкам.

Пример 1.4. Допустим, что ключи хэширования – положительные целые числа. Простая и употребительная хэш-функция – $h(x) = x \bmod B$ – возвращает остаток от деления *x* на B. Она неплохо работает, если совокупность ключей хэширования – множество всех положительных целых чисел. Тогда доля ключей, попавших в каждую ячейку, составит $1/B$. Но предположим, что наша совокупность содержит только четные числа и пусть B = 10. Тогда значениями $h(x)$ могут быть только ячейки с номерами 0, 2, 4, 6, 8, так что поведение хэш-функции заведомо не случайно. С другой стороны, если взять B = 11, то окажется, что доля четных чисел в каждой из 11 ячеек равна 1/11, т. е. хэш-функция работает очень хорошо.

Обобщая пример 1.4, можно сказать, что если ключами хэширования являются целые числа, то при выборе в качестве B числа, имеющего общий множитель со

всеми возможными ключами (или хотя бы с их большинством), распределение по ячейкам будет неравномерным. Поэтому обычно в качестве B берут простое число. При таком выборе снижаются шансы неслучайного поведения, хотя по-прежнему необходимо рассмотреть случай, когда все ключи делятся на B. Разумеется, есть много других типов хэш-функций, не зависящих от арифметики по модулю. Мы не станем пытаться систематизировать их здесь, но приведем несколько источников в списке литературы.

А что, если ключи хэширования не являются целыми числами? Вообще говоря, у любого типа данных имеется значение, состоящее из битов, а последовательность битов всегда можно интерпретировать как целое число. Однако существуют простые правила, позволяющие преобразовать наиболее употребительные типы в целые числа. Например, если ключ хэширования – строка, то мы можем преобразовать каждый символ в его значение в кодировке ASCII или Unicode, которое можно интерпретировать как небольшое целое число. Затем, перед делением на B, эти числа складываются. Если B меньше типичной суммы кодов символов для некоторой совокупности строк, то распределение по ячейкам будет близко в равномерному. Если же B больше, то мы можем разбить все символы строки на несколько групп. Затем конкатенируем коды символов в одной группе и рассматриваем результат как одно целое число. Складываем все получившиеся таким образом числа и делим на B, как и раньше. Например, если B порядка миллиарда, т. е. 2^{30}, то группировка символов по четыре даст 32-разрядные целые числа. Их суммы распределяются по миллиарду ячеек приблизительно равномерно.

Эта идея рекурсивно обобщается на более сложные типы данных.

- Если тип является записью, каждая компонента которой имеет свой тип, то рекурсивно преобразовать значение каждой компоненты в целое число, применяя алгоритм, соответствующий типу компоненты. Сложить целые числа, получившиеся для всех компонент, и преобразовать сумму в номер ячейки, разделив ее на B.
- Если тип является массивом, множеством или коллекцией элементов одного и того же типа, то преобразовать значения его элементов в целые числа, сложить результаты и разделить сумму на B.

1.3.3. Индексы

Индекс – это структура данных, которая позволяет эффективно находить объект по значениям одного или нескольких его элементов. Наиболее типична ситуация, когда объекты являются записями, а индекс строится по одному из полей каждой записи. Если известно значение v, то индекс позволяет найти все записи, в которых это поле имеет такое значение. Например, мы можем располагать файлом, содержащим тройки (имя, адрес, телефон), и построить индекс по полю «телефон». Зная номер телефона, мы можем с помощью индекса быстро найти одну или несколько записей с таким номером.

Есть много способов реализации индексов, и мы не собираемся приводить обзор на эту тему. В списке литературы имеются ссылки на источники для дальнейшего изучения. Однако отметим, что хэш-таблица – один из простых методов построения индекса. Одно или несколько полей, по которым строится индекс, образует ключ хэширования, подаваемый на вход хэш-функции. Хэш-функция применяется к ключу хэширования записи, а сама запись помещается в ячейку с номером, возвращенным хэш-функцией. Ячейка может представлять собой список записей в оперативной памяти или, скажем, блок на диске.

Впоследствии, имея значение ключа хэширования, мы можем хэшировать его, вычислить соответствующую ячейку и производить поиск только в этой ячейке, т. е. среди записей с указанным значением ключа хэширования. Если выбрать количество ячеек B примерно того же порядка, что и количество записей в файле, то в каждой ячейке окажется сравнительно мало записей, и поиск по ячейке будет занимать мало времени.

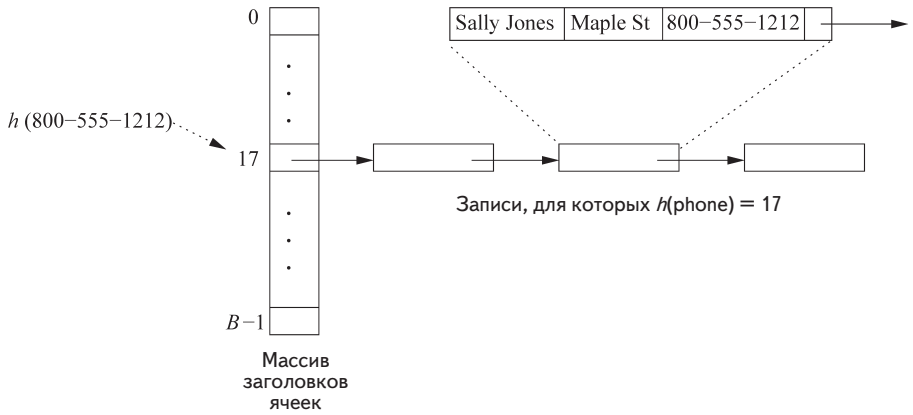

Рис. 1.2. Использование хэш-таблицы в качестве индекса; номер ячейки определяется хэш-кодом телефона, а сама запись помещается в ячейку, соответствующую хранящемуся в ней телефону

Пример 1.5. На рис. 1.2 показано, как в оперативной памяти может выглядеть индекс записей, содержащих имя, адрес и телефон. Здесь индекс построен по номеру телефона, а ячейки представляют собой связанные списки. Телефон 800-555-1212 хэшируется в ячейку 17. Существует массив *заголовков ячеек*, *i*-ый элемент которого является началом связанного списка для ячейки с номером *i*. На рисунке показан один развернутый элемент списка. Он содержит запись с полями имени, адреса и телефона. В этой конкретной записи хранится телефон 800-555-1212. В других записях той же ячейки телефон может быть таким же или отличающимся. Мы знаем лишь, что хэш-код телефона в любой записи из этой ячейки равен 17.

1.3.4. Внешняя память

При работе с большими данными важно понимать, насколько различается время вычислений в случаях, когда данные хранятся на диске и в оперативной памяти. Мы могли бы много чего сказать по поводу физических характеристик диска, но не будем увлекаться и дадим возможность интересующемуся читателю самому изучить рекомендуемую в конце главы литературу.

Диски организованы в виде совокупности *блоков* – минимальных единиц, используемых операционной системой для перемещения данных между диском и оперативной памятью. Так, в Windows размер блока составляет 64 КБ (т. е. $2^{16} = 65\,536$ байтов). Для *доступа* к диску (переместить головку считывания-записи к нужной дорожке и дождаться, пока под ней окажется нужный блок) и считывания блока требуется примерно 10 миллисекунд. Эта задержка по меньшей мере на пять порядков (в 10^5 раз) превышает время считывания слова из оперативной памяти, т. е. если нам всего-то и нужно, что получить несколько байтов, то преимущества хранения данных в оперативной памяти не вызывают ни малейших сомнений. На самом деле, если мы хотим сделать что-то очень простое с каждым байтом в дисковом блоке, например рассматривать блок как ячейку хэш-таблицы и искать в этой ячейке записи с нужным значением ключа хэширования, то время, необходимое для перемещения блока с диска в оперативную память, окажется намного больше времени вычислений.

Если сделать так, чтобы связанные между собой данные располагались на одном *цилиндре* (множество блоков, отстоящих на одно и то же расстояние от центра диска, вследствие чего для чтения другого блока из того же цилиндра не нужно перемещать головку), то прочитать все блоки из этого цилиндра в оперативную память можно меньше, чем за 10 мс на блок. Можно считать, что диск не способен передавать данные в память со скоростью более ста миллионов байтов в секунду, как бы они ни были организованы. Если размер набора данных составляет мегабайт, то никакой проблемы нет. Но набор размером порядка сотен гигабайтов или терабайт сложно даже прочитать, не говоря уже о том, чтобы сделать нечто полезное.

1.3.5. Основание натуральных логарифмов

У числа $e = 2.7182818\ldots$ есть целый ряд полезных свойств. В частности, e является пределом последовательности $\left(1+\frac{1}{x}\right)^x$ при x стремящемся к бесконечности. Значения этого выражения для $x = 1, 2, 3, 4$ приблизительно равны 2, 2.25, 2.37, 2.44, так что нетрудно поверить, что предел этой последовательности близок к 2.72.

Простые алгебраические преобразования позволяют получить аппроксимации многих, на первый взгляд, сложных выражений. Возьмем, к примеру, выражение $(1+a)^b$, где a мало. Его можно переписать в виде $(1+a)^{(1/a)(ab)}$. Если теперь подставить $a = 1/x$ и $1/a = x$, то получим

$$\left(1+\frac{1}{x}\right)^{x(ab)}, \text{ или } \left(\left(1+\frac{1}{x}\right)^{x}\right)^{ab}.$$

Так как a, по предположению, мало, то x велико, следовательно, подвыражение $\left(1+\frac{1}{x}\right)^{x}$ будет близко к своему пределу e. Таким образом, $(1 + a)^b$ можно аппроксимировать выражением e^{ab}.

Аналогичные тождества имеют место для отрицательных a. То есть при x стремящемся к бесконечности предел $\left(1-\frac{1}{x}\right)^{x}$ равен $1/e$. Отсюда следует, что аппроксимация $(1 + a)^b = e^{ab}$ справедлива и тогда, когда a – малое отрицательное число. По-другому то же самое можно выразить, сказав, что $(1 - a)^b$ приближенно равно e^{-ab}, когда a мало, а b велико.

Другие полезные аппроксимации вытекают из разложения e^x в ряд Тейлора: $e^x = \sum_{i=0}^{\infty} x^i / i!$, или $e^x = 1 + x + x^2/2 + x^3/6 + x^4/24 + \ldots$. При больших x этот ряд сходится медленно, но все же сходится, потому что $n!$ растет быстрее x^n при любой константе x. Однако при малых x, все равно положительных или отрицательных, ряд сходится быстро и для получения хорошей аппроксимации достаточно всего нескольких членов.

Пример 1.6. Пусть $x = 1/2$. Тогда $e^{1/2} = 1+\frac{1}{2}+\frac{1}{8}+\frac{1}{48}+\frac{1}{348}+\ldots$ или приближенно $e^{1/2} = 1.64844$.

Пусть $x = -1$. Тогда $e^{-1} = 1 - 1+\frac{1}{2}+\frac{1}{6}+\frac{1}{24}+\frac{1}{120}+\frac{1}{720}+\frac{1}{5040}+\ldots$ или приближенно $e^{-1} = 0.36786$.

1.3.6. Степенные зависимости

Многие явления описываются уравнениями, в которых две переменные связаны *степенной зависимостью*, т. е. между логарифмами этих переменных существует линейная зависимость. На рис. 1.3 показан пример такой зависимости. Если x – горизонтальная ось, а y – вертикальная, то эта зависимость описывается формулой $\log_{10} y = 6 - 2 \log_{10} x$.

Пример 1.7. Мы могли бы заняться исследованием продаж книг на сайте Amazon.com и считать, что x представляет ранг книги по продажам. Тогда y – количество продаж книги, стоящей на x-ом месте по продажам, за некоторый период времени. Из графика на рис. 1.3 следует, что для книги, занявшей первое место, продано 1 000 000 экземпляров, для книги на 10-м месте – 10 000 экземпляров, на 100-м месте – 100 экземпляров и т. д. для всех рангов в этом диапазоне и вне его. Делать вывод о том, что для книги с рангом 1000 продана лишь часть экземпляра, было бы слишком смело – на

самом деле, мы ожидаем, что для рангов, существенно больших 1000, зависимость выйдет на плато.

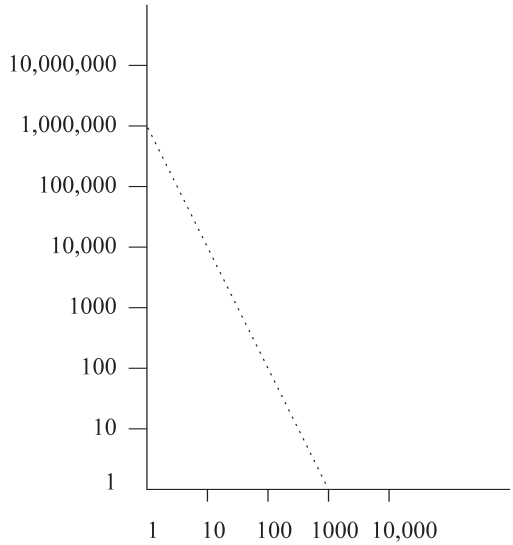

Рис. 1.3. Степенная зависимость с угловым коэффициентом –2

Эффект Матфея

Часто существование степенных зависимостей, в которых показатель степени больше 1, объясняют *эффектом Матфея*. В Евангелии от Матфея есть слова о том, что «...всякому имеющему дастся и приумножится...». Такое поведение характерно для многих явлений: если значение некоторого свойства велико, то оно и далее продолжает увеличиваться. Например, если на веб-страницу ведет много ссылок, то пользователи будут находить ее с большей вероятностью, а, значит, и ставить на нее ссылки со своих страниц. Другой пример: если книга хорошо продается в Amazon, то, скорее всего, она будет рекламироваться пользователям, когда они заходят на сайт. И кто-то решит купить эту книгу, тем самым еще больше увеличив ее продажи.

В общем случае степенная зависимость между x и y описывается формулой $\log y = b + a \log x$. Если возвести основание логарифма (которое на самом деле ни на что не влияет), скажем e, в степень, равную левой и правой части этого уравнения, то получим $y = e^b e^{a \log x} = e^b x^a$. Поскольку e^b – просто «некая константа», мы можем заменить ее константой c. Таким образом, степенная зависимость может быть записана в виде $y = cx^a$ при некоторых константах a и c.