ВЕРОЯТНОСТНЫЕ СТРУКТУРЫ ДАННЫХ И АЛГОРИТМЫ

В ПРИЛОЖЕНИЯХ BIG DATA

АНДРЕЙ ГАХОВ

gakhov

Вероятностные структуры данных и алгоритмы
в приложениях Big Data

Русское издание, 2022

Translation from the English language edition: Probabilistic Data Structures and Algorithms for Big Data application. 1st edition, 2019

The paperback edition is printed and distributed on behalf of the author:
tredition GmbH
Halenreie 40-44, 22359 Hamburg
Germany

ISBN (paperback): 978-3-347-54322-5

Посвящается
моей маме Валентине Гаховой

Оглавление

Предисловие

Большие данные (англ. Big Data) характеризуются тремя фундаментальными измерениями: *Объёмом* (англ. Volume), *Скоростью* (англ. Velocity) и *Многообразием* (англ. Variety), вместе известные как **три "V" больших данных**. *Объём* отображает количество, *Скорость* описывает быстроту поступления и обработки данных, а *Многообразие* определяет множество типов данных.

В настоящее время всё может выступать в качестве источника данных — социальные сети, всевозможные датчики и сенсоры, финансовые операции и многое другое. Компания IBM заявила, что человечество уже создает **2.5 квинтиллиона** байт данных **ежедневно** и их объём постоянно растёт, причём большая его часть не может быть даже сохранена и обычно пропадает без какой-либо обработки. Сегодня на практике нередко приходится обрабатывать массивы данных размером в терабайты и петабайты, как и гигабитные потоки данных.

С другой стороны, каждая компания хочет полностью анализировать имеющиеся в её распоряжении данные, чтобы найти в них бизнес-ценность и действовать в соответствии с ней. Это приводит к быстрому росту рынка программного обеспечения для

работы с Big Data. Однако традиционные технологии, включающие структуры данных и алгоритмы, теряют свою эффективность при работе с большими данными. Поэтому специалисты по программному обеспечению снова и снова обращаются к математике и компьютерным наукам в поисках подходящих решений, и одним из них являются вероятностные структуры данных и алгоритмы.

Вероятностные структуры данных — это общее название для структур данных, основанных в основном на различных техниках хеширования и эвристических наблюдениях. В отличие от обычных (или детерминированных) структур данных, они предоставляют приближённые ответы, но с надёжными способами оценки возможных неточностей. К счастью, потенциальные потери и ошибки полностью компенсируются чрезвычайно низкими требованиями к памяти, ограниченным временем выполнения запросов и масштабируемостью — факторами, особенно важными в приложениях Big Data.

О книге

Цель этой книги — познакомить специалистов-практиков в области технологий, включая архитекторов и разработчиков программного обеспечения, а также лиц, принимающих технологические решения, с вероятностными структурами данных и алгоритмами. Прочитав эту книгу, вы получите теоретическое и практическое представление о вероятностных структурах данных и узнаете об их наиболее распространенных применениях.

Конечно же, вам не нужно быть научным сотрудником, но чтобы извлечь из книги максимальную пользу необходимы базовые математические знания и понимание общей теории структур данных и алгоритмов. Если у вас нет опыта в области компьютерных наук, настоятельно рекомендуем вам прочитать книгу “*Алгоритмы. Построение и анализ*” Томаса Кормена, Чарльза Лейзерсона, Рональда Ривеста и Клиффорда Штайна, представляющую собой полное руководство по современным компьютерным алгоритмам.

Разумеется, невозможно охватить все существующие удивительные решения, а задача этой книги — выделить их общие идеи и важные области применения, такие как запросы о принадлежности, подсчёт количества элементов и их частоты, анализ потоковых данных и оценка сходства документов.

Структура книги

Эта книга состоит из шести глав, каждой из которых предшествует введение, а завершает — краткое резюме и библиография для дальнейшего чтения, относящаяся к данной главе. Каждая глава посвящена одной конкретной проблеме в приложениях Big Data и начинается с подробного объяснения проблемы, а затем представляет структуры данных и алгоритмы, которые могут быть использованы для её эффективного решения.

В первой главе даётся краткий обзор популярных хеш-функций и хеш-таблиц, которые широко используются в вероятностных структурах данных. Глава 2 посвящена приблизительным запросам о принадлежности, наиболее известному случаю использования таких структур. В главе 3 обсуждаются структуры данных, помогающие оценить число уникальных элементов, а главы 4 и 5 посвящены важным вычислениям метрик, связанных с частотой и рангами в потоковых приложениях. Глава 6 состоит из структур данных и алгоритмов для решения проблем сходства, в частности — поиска ближайшего соседа.

Книга в сети Интернет

Примеры, дополнительную информацию и выявленные ошибки можно найти на сайте https://pdsa.gakhov.com. Если у вас есть комментарий, технический вопрос по книге, вы хотите сообщить о найденной ошибке или по любому другому вопросу, отправьте письмо по адресу pdsa@gakhov.com.

Если вас также интересует программная реализация на Cython, включающая многие структуры данных и алгоритмы из этой книги, пожалуйста, ознакомьтесь с нашей бесплатной Python-библиотекой с открытым исходным кодом под названием PDSA, посетив https://github.com/gakhov/pdsa. Все желающие могут внести свой вклад в любое время.

Об авторе

Андрей Гахов — математик и разработчик программного обеспечения, кандидат физико-математических наук в области математического моделирования и численных методов. В течение нескольких лет преподавал на факультете компьютерных наук Харьковского национального университета имени В. Н. Каразина (Украина), а в настоящее время занимает должность CTO в компании ferret go GmbH, ведущей компании Германии в области модерации, автоматизации и аналитики сообществ. В сферу его интересов входят машинное обучение, обработка и анализ данных.

Лучший способ связаться с автором — через Twitter @gakhov или посетив его веб-страницу https://www.gakhov.com.

Благодарности

Автор хотел бы поблагодарить Азмира Мустафик и Жоа Ванкоппенолле за вклад в рецензирование английского издания книги и за их полезные рекомендации. Большая благодарность научным рецензентам Катерине Несвит и Дхаравату Рамешу за их бесценные предложения и замечания. Особая благодарность Теду Даннингу, автору алгоритма t-дайджест, за доскональный обзор соответствующей главы, проницательные вопросы и обсуждение.

Наконец, спасибо всем, кто оставил свои отзывы и помог закончить эту книгу.

1

Хеширование

Хеширование играет важную роль в вероятностных структурах данных, которые используют его для рандомизации и компактного представления данных. *Хеш-функция* сжимает блоки входных данных произвольной длины, генерируя идентификатор меньшей (и в большинстве случаев — фиксированной) длины, известный как *значение хеш-функции* или просто *хеш*.

Выбор хеш-функции является решающим для предотвращения перекосов в распределении и зачастую зависит от входных данных и соответствующих сценариев использования, однако есть и общие свойства, которым должна обладать подходящая хеш-функция.

Хеш-функции сжимают входные данные и, следовательно, невозможно полностью избежать случаев генерации одинаковых значений для двух различных блоков данных, известных как *коллизии хеш-функции*, но можно ограничить вероятность их появления путём выбора хеш-функций с соответствующими свойствами.

В 1979 году Дж. Лоуренс Картер и Марк Вегман описали *универсальные хеш-функции*, математические свойства которых гарантируют низкое ожидаемое число коллизий даже для произвольных входных данных.

Семейство универсальных хеш-функций $\mathrm{H} = \{h\}$ ставит в соответствие входным элементам числа из диапазона $\{0, 1, \ldots, m-1\}$ и гарантирует, что при случайном выборе хеш-функции h из такого семейства, вероятность коллизий ограничена:

$$\Pr\Big(h(x) = h(y)\Big) \leq \frac{1}{m}, \text{ для любых } x, y : x \neq y. \tag{1.1}$$

Таким образом, случайный выбор хеш-функции из семейства со свойствами (1.1) фактически соответствует случайному равномерному выбору элемента, что используется в вероятностных структурах данных.

Практически важное семейство универсальных хеш-функций, предназначенное для работы с целыми числами, определяется как

$$h_{\{k,q\}}(x) = ((k \cdot x + q) \bmod p) \bmod m, \tag{1.2}$$

где k и q — случайно выбранные целые числа по модулю p с $k \neq 0$. В качестве параметра p выбирается простое число $p \geq m$ и, как правило, используется одно из известных чисел Мерсенна (англ. Mersenne prime) — например, для $m = 10^9$ можно выбрать $p = \mathrm{M}_{31} = 2^{31} - 1 \approx 2 \cdot 10^9$.

Для некоторых приложений достаточно и более простой версии семейства (1.2), например,

$$h_{\{k\}}(x) = (k \cdot x \bmod p) \bmod m, \tag{1.3}$$

которое только приблизительно универсально, но всё ещё обладает низкой вероятностью коллизий, ожидаемо меньшей, чем $\frac{2}{m}$.

Указанные выше известные семейства хеш-функций играют важную роль, но ограничены целочисленными входными данными, в то время как всё больше важных приложений оперируют векторами переменной длины и нуждаются в быстрых и надёжных хеш-функциях с определёнными гарантированными свойствами.

На практике используются множество различных хеш-функций, зачастую с менее выраженными математическими свойствами, но выбираемые с учетом требований соответствующих приложений. В настоящей главе мы рассмотрим популярные хеш-функции и вспомогательные структуры данных, которые находят своё применение в вероятностных структурах данных.

1.1 Криптографические хеш-функции

Практически *криптографические хеш-функции* определяются как преобразования входящих строк переменной длины в строки фиксированной длины. Они очень важны в криптографии и широко применяются для вычисления цифровых подписей, аутентификации и проверки целостности сообщений.

Как уже было указано, невозможно избежать коллизий хеш-функций, однако безопасная хеш-функция должна обладать *устойчивостью к коллизиям*, определяющей достаточную сложность целенаправленного обнаружения коллизий. Конечно, это не означает, что коллизий нет вообще, их всё ещё можно найти случайно или рассчитать заранее (поэтому такие функции всегда требуют математических доказательств перед применением).

Криптографические хеш-функции должны удовлетворять трём основным требованиям:

- *фактор трудозатрат* — получение криптографического хеша должно быть вычислительно дорогостоящим, чтобы

усложнить нахождение исходной строки атакой методом грубой силы (англ. brute force);

- *устойчивое состояние* — криптографический хеш не должен иметь внутреннего состояния, в котором он будет соответствовать какому-нибудь из вероятных входных шаблонов;
- *диффузия* — каждый бит вычисленного хеш-значения должен быть столь же сложной функцией каждого бита входного значения. Таким образом, при малейшем изменении входного значения, ожидается, что хеш значительно изменится (лавинный эффект).

Теоретически криптографические функции могут быть разделены на *хеш-функции с ключом*, использующие некое секретное значение для вычисления, и *хеш-функции без ключа*, работающие без него. Вероятностные структуры данных используют только функции без ключа, которые включают в себя *односторонние хеш-функции*, *устойчивые к коллизиям хеш-функции* и *универсальные односторонние хеш-функции*. Эти классы функций отличаются только некоторыми дополнительными свойствами.

Односторонние хеш-функции удовлетворяют следующим требованиям:

- применимость к блокам данных произвольной длины (на практике длина ограничена некой очень большой константой);
- выходное значение (хеш) фиксированной длины;
- *устойчивость к нахождению прообраза* (*односторонность*) — должно быть вычислительно невозможно найти входное значение по известному хеш-значению.

Кроме того, для устойчивых к коллизиям хеш-функций получение одинакового хеш-значения для двух отличающихся входных данных должно быть также крайне маловероятным.

В качестве альтернативы требованию устойчивости к коллизиям,

универсальные односторонние хеш-функции могут быть *устойчивы к нахождению второго прообраза*, что означает вычислительную невозможность нахождения другого входного значения, отличного от исходного, хеш-значения которых совпадают.

Отметим, что устойчивость к коллизиям включает в себя устойчивость к нахождению второго прообраза, но из-за фиксированности исходного входного значения в таких атаках, общая сложность нахождения второго прообраза более высокая, чем при поиске коллизии.

Следствием требований к криптографическим функциям (в частности, их высокого фактора трудозатрат) является большее время вычисления, чем у хеш-функций общего назначения. Например, функция SHA–1, обсуждаемая ниже, находится на уровне 540 МиБ/секунду[1], в то время как популярные некриптографические функции имеют порядок 2500 МиБ/секунду и более.

Message–Digest

Популярный *алгоритм Message–Digest*, известный как MD5, был предложен Рональдом Ривестом в 1991 году в качестве замены устаревшему к тому времени алгоритму MD4. Это популярный криптографический алгоритм хеширования, определённый в стандарте IETF RFC 1321, который для входных сообщений (англ. message) произвольной длины вычисляет хеш-значения фиксированной длины в 128-бит (англ. digest).

Алгоритм MD5 основан на схеме Меркла–Дамгарда (англ. Merkle–Damgård schema). На первом этапе он преобразовывает входной блок данных произвольной длины в последовательность блоков одинакового размера (512-битные блоки или шестнадцать 32-битных слов), последний блок дополняется нулями, если это необходимо. Для обеспечения устойчивости хеширования схема использует процедуру упрочнения Меркла-Дамгарда, согласно которой входные данные дополняются блоком, кодирующим длину

[1] Crypto++ 6.0.0 Benchmarks `https://www.cryptopp.com/benchmarks.html`

первоначального сообщения. Далее полученные блоки сжимаются один за другим при помощи функции сжатия, причём каждый последующий блок использует результат сжатия предыдущего. Итогом применения алгоритма является 128-битное хеш-значение, полученное в результате сжатия последнего блока, к которому дополнительно может применяться функция финализации для увеличения диффузии и достижения лавинного эффекта.

Алгоритм MD5 часто используется для контроля целостности файлов. Вместо проверки, что файл не изменился путём просмотра непосредственного содержимого, достаточно просто сравнить его текущее и сохранённое MD5 хеш-значения.

Как указано в Vulnerability Note VU#836068[2], алгоритм MD5 уязвим к коллизионным атакам. Обнаруженная уязвимость в алгоритме позволяет построить отличные друг от друга сообщения с одинаковым MD5 хеш-значением. В результате атакующие могут генерировать криптографические токены или подписи, которые незаконно представляются подлинными. В настоящее время не рекомендуется использовать MD5 в качестве криптографического алгоритма хеширования, однако данные уязвимости не имеют большого значения для вероятностных структур данных.

Secure Hash Algorithms

Семейство *Secure Hash Algorithms* разработано Агентством национальной безопасности США (англ. NSA) и опубликовано Национальным институтом стандартов и технологий (англ. NIST). Первый алгоритм семейства, известный как SHA–0, был опубликован в 1993 году и достаточно быстро был заменён алгоритмом SHA–1, получившим всеобщее признание. SHA–1 выдавал более длинное 160-битное (20-байтовое) хеш-значение, а его устойчивость была улучшена исправлением уязвимостей, найденных в SHA–0.

SHA–1 использовался много лет в различных приложениях и большинство интернет-сайтов было подписано алгоритмами на его

[2]VU#836068 `http://www.kb.cert.org/vuls/id/836068`

основе. Однако в 2005 году была найдена уязвимость и в SHA–1, что привело к запрету его применения для государственных интернет-сайтов с 2010 года, а годом спустя и в целом в сети Интернет. Также как и для MD5, найденные уязвимости не влияют на их использование в вероятностных структурах данных.

Опубликованное в 2001 году семейство SHA–2 включает криптографические алгоритмы хеширования с различными размерами выходного хеш-значения: SHA–224, SHA–256, SHA–384, SHA–512 и другие. SHA–2 более сильный по сравнению с SHA–1, и ожидается, что успешные атаки на SHA–2 маловероятны при существующих вычислительных возможностях.

RadioGatún

Семейство криптографических алгоритмов хеширования RadioGatún предложено Гвидо Бертони, Йоаном Даменом, Микаэлем Петерсом и Жилем Ван Аше на Second Cryptographic Hash Workshop в 2006 году [Be06]. На момент выхода RadioGatún улучшил популярную ранее хеш-функцию *Panama*, в которой были найдены уязвимости.

Аналогично другим хеш-функциям, входное значение разбивается на последовательность блоков, которые включаются во внутреннее состояние алгоритма, используя специальную функцию. Далее итеративно на каждом цикле (раунде) применяется выбранная некриптографическая функция раунда (англ. *belt-and-mill* round function). В каждом раунде, состояние представляется двумя частями — поясом (англ. belt) и мельницей (англ. mill), каждую из которых функция раунда обрабатывает отдельно. Работа функции раунда состоит из четырех параллельных операций: 1) применение нелинейной функции к мельнице; 2) применение простой линейной функции к поясу; 3) линейная пересылка некоторых битов из мельницы в пояс; 4) линейная пересылка некоторых битов из пояса в мельницу. После включения всех входных блоков алгоритм проходит некоторое количество раундов без входных и выходных данных (пустые раунды), а полученное внутреннее состояние возвращается в качестве выходного хеш-значения.

При одинаковой тактовой частоте (англ. clock frequency) RadioGatún32, по некоторым оценкам, в 12 раз опережает SHA–256 для длинных входных данных, в 3.2 раза быстрее для коротких входных данных, при меньшем количестве логических вентилей. RadioGatún64 даже в 24 раза быстрее SHA–256 для длинных входных данных, но имеет на 50% больше логических вентилей.

Среди всего семейства функция RadioGatún64 с 64-битными словами является выбором по-умолчанию и оптимальной для 64-разрядных платформ. Для улучшенной производительности на 32-разрядных платформах может быть использована и RadioGatún32 с 32-битными словами.

1.2 Некриптографические хеш-функции

В отличие от криптографических, требования к которым описаны ранее, некриптографические хеш-функции не имеют встроенной защиты от атак, направленных на поиск коллизий, и поэтому не требуют высокой сложности и устойчивости. Однако они должны быть быстрыми в вычислении и гарантировать низкую вероятность коллизий, позволяя хешировать большие объёмы данных за фиксированное время с приемлемой вероятностью ошибки.

Fowler/Noll/Vo

Некриптографический *алгоритм Fowler/Noll/Vo* (FNV или FNV1) берет своё начало от идеи, отправленной в качестве комментария рецензента в IEEE POSIX P1003.2. Он разработан Гленном Фаулером и Фонгом Во в 1991 году и позднее улучшен Лэндоном Ноллом [Fo18].

Алгоритм FNV поддерживает внутреннее состояние, которое инициализировано специальным базисом смещения. После чего алгоритм разбивает входное значение на блоки по 8 бит, проводит умножение значения внутреннего состояния на выбранную большую

числовую константу, известную как *FNV Prime*, и применяет логическое вычитание (исключающее "или") к входному блоку. Как только последний входной блок обработан, полученное значение внутреннего состояния возвращается в качестве хеш-значения.

Величины константы FNV Prime и базиса смещения являются предопределёнными параметрами алгоритма и зависят от длины производимых хеш-значений. Как отметил Лэндон Курт Нолл, выбор простых чисел является важным условием эффективной работы FNV алгоритма, а некоторые простые числа предпочтительнее других при одной и той же длине хеш-значений.

Модификация алгоритма, известная как FNV1a, отличается от основного алгоритма порядком применения логического вычитания и операций умножения. Несмотря на то, что оба алгоритма используют одинаковые значения FNV Prime, FNV1a обеспечивает немного лучшую диффузию без влияния на производительность, и поэтому является предпочтительным.

Семейство алгоритмов FNV включает в себя алгоритмы для вычисления 32-, 64-, 128-, 256-, 512- и 1024-битных хеш-значений.

Алгоритм FNV очень прост в реализации, а высокий разброс получаемых хеш-значений делает его подходящим для хеширования практически идентичных строк. Он широко применяется в DNS серверах, Twitter, индексировании в базах данных и поисковых системах. Несколько лет назад 32-битная версия алгоритма FNV1a была рекомендована в качестве алгоритма хеширования для генерации меток потока в протоколе IPv6 [An12].

MurmurHash

Еще одно популярное семейство хеш-функций, известное как *MurmurHash*, было опубликовано Остином Эпплби в 2008 году, а в 2011 году [Ap11] вышла улучшенная версия — *MurmurHash3*. Семейство MurmurHash3 включает в себя 32- и 64-битные версии для x86 и x64 платформ.

Алгоритмы семейства MurmurHash используют специальный

вероятностный приём приближённого вычисления глобального оптимума, эффективно смешивающий биты входного значения для получения битов выходного хеш-значения. Разные поколения различаются главным образом своими смешивающими функциями. Как утверждается, данный алгоритм в два раза быстрее, чем оптимизированный по скорости алгоритм[3] *lookup3*.

В настоящее время MurmurHash3 является одним из самых популярных алгоритмов и используется в Apache Hadoop, Apache Cassandra, Elasticsearch, libstdc++, nginx и других приложениях.

CityHash и FarmHash

В 2011 году Google опубликовал новое семейство некриптографических хеш-функций для строк, известное как *CityHash*, разработанное Джеффом Пайком и Юрки Алакуиялой [Pi11] на основе алгоритма MurmurHash2.

Алгоритмы семейства CityHash разрабатывались с фокусом на работу с короткими строками (как правило, до 64 байт), которые представляют основной интерес для хеш-таблиц и вероятностных структур данных. Семейство включает в себя 32-, 64-, 128- и 256-битные версии. Для таких коротких строк 64-битная версия CityHash64 работает быстрее MurmurHash и превосходит 128-битную версию CityHash128. Для длинных строк, размером более нескольких сотен байт, алгоритм CityHash128 является предпочтительным среди других алгоритмов семейства, однако на практике всё же лучше использовать MurmurHash3 в таких случаях.

Недостатками CityHash являются его сложность и зависимость от компиляторов, что может оказывать существенное влияние на скорость работы.

В 2014 году Google опубликовал преемника CityHash под названием *FarmHash*, разработанного Джеффом Пайком [Pi14]. Новый алгоритм, основанный на обновленной версии алгоритма MurmurHash, унаследовал большинство приёмов, использованных в

[3]Hash Functions and Block Ciphers `https://burtleburtle.net/bob/hash/`