数据科学与大数据技术

漫画算法与数据结构
(大规模数据集)

[波黑] 黛拉·梅杰多维奇(Dzejla Medjedovic)
埃明·塔希罗维奇(Emin Tahirovic) 著
[波黑]伊内斯·德多维奇(Ines Dedovic) 绘
郭 涛 袁洪斌 译

清華大學出版社
北 京

北京市版权局著作权合同登记号　图字：01-2023-1110

Dzejla Medjedovic，Emin Tahirovic，Illustrated by Ines Dedovic
Algorithms and Data Structures for Massive Datasets
EISBN: 978-161729-803-5

图书在版编目(CIP)数据

漫画算法与数据结构：大规模数据集 / (波黑)黛拉·梅杰多维奇，(波黑)埃明·塔希罗维奇著；(波黑)伊内斯·德多维奇绘；郭涛，袁洪斌译. —北京：清华大学出版社，2024.1
(数据科学与大数据技术)
书名原文：Algorithms and Data Structures for Massive Datasets
ISBN 978-7-302-64520-7

I. ①漫…　II. ①黛…②埃…③伊…④郭…⑤袁…　III. ①算法分析－通俗读物②数据结构－通俗读物　IV. ①TP301.6-49②TP311.12-49

中国国家版本馆 CIP 数据核字(2023)第 166613 号

责任编辑：王　军
封面设计：孔祥峰
版式设计：思创景点
责任校对：成凤进
责任印制：丛怀宇

出版发行：清华大学出版社
网　　址：https://www.tup.com.cn，https://www.wqxuetang.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-83470000　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：艺通印刷(天津)有限公司
经　　销：全国新华书店
开　　本：148mm×210mm　　印　　张：10.625　　字　　数：296 千字
版　　次：2024 年 2 月第 1 版　　印　　次：2024 年 2 月第 1 次印刷
定　　价：79.80 元

产品编号：097409-01

译 者 序

数据结构主要研究数据的组织存储方式，算法分析是对算法运行时间和存储空间进行定量的评估，即算法复杂度分析。大数据和人工智能时代的来临给算法和数据结构的设计带来一系列挑战。第一，算法具有一定的难度，想要通过合适的数据结构和编程语言表达精巧的逻辑，从而具备计算机底层思维能力，这本身就不是一件易事；第二，要巧妙运用数据结构和算法分析解决复杂的业务场景和大型系统设计，同样难度很大；第三，数据结构和算法设计依赖于数学分析、理论计算机，因此设计人员需要具备良好的计算机理论基础。本书作者遵循“问题源于实践、理论先于实践、技术高于实践”的原则，由实际问题驱动，注重理论积累，以解决实际生产中的问题。

本书的重点并不是介绍通用的数据结构与算法分析。在大数据和人工智能的时代背景下，传统的经典算法往往性能不佳，甚至可能不起作用。本书以分布式数据集、流式数据结构与算法设计为主线，对流式数据采集、数据库中的数据结构设计、外部存储器算法进行介绍。目前，实际生产中已经形成了流式数据采集、存储、分析和计算的产品且成果显著。针对流式数据的采集和存储的产品主要有 Apache Kafka、Apache Pulsar 和 Pravega。流式数据的计算与分析主要经历了两代产品，第一代为 Apache Storm、Spark Streaming，目前流行的是第二代产品 Apache Flink。此外，还出现了 MPP(Shared Nothing 架构)的分布式并行架构数据库集群，主要有 Greenplum、HAWQ、HashData 等分布式数据库系统。通过在 MPP 架构基础上对流式数据的存储和计算支持，单节点每秒可处理多达 100 亿行数

据，支持大规模数据实时写入且保证秒级实时性，主要的产品有Apache Doris、StarRocks 和 MatrixDB。这些优秀的产品无不把流式数据的数据结构和算法体现得淋漓尽致。本书针对流式数据场景，对常见的大规模数据集算法和数据结构进行了梳理和讲解。这些流式数据产品的出现有效解决了海量流式数据的采集、存储和极速全场景分析计算等问题。本书可作为从事算法设计与分析、大数据平台分析、模式识别与人工智能和数据库等领域研究工作的工程师、计算机科学家的参考书。

在翻译本书的过程中，得到了很多人的帮助。吉林财经大学外国语学院的张煜琪、吉林大学外国语学院的吴禹林和对外经济贸易大学英语学院的许瀚等对本书进行了校对和复审工作，感谢他们所做的工作。最后，感谢清华大学出版社的编辑，他们做了大量的编辑与校对工作，保证了本书的质量，使得符合出版要求。

由于本书涉及的广度和深度较大，加上译者水平有限，在翻译过程中难免有不足之处，若各位读者在阅读过程中发现问题，欢迎批评指正。

序　言

撰写本书的想法萌生于笔者在萨拉热窝国际大学任教期间。与在本地公司工作的学生讨论时，我们意识到，大规模数据的数据结构在数据工程师和数据科学家的日常使用中变得非常普遍。世界上使用这些技术来解决其可扩展性问题的不只是 Google 和 Facebook(现在一般指 Meta 公司)，还有一些小规模的公司，它们的系统对数据处理速度的需求正日益增长。

午餐时，我们会思考，那些学习将 HyperLogLog 或 Bloom 过滤器部署到工作生产系统中的学生该如何获得对应用程序友好的概述。从数学角度看，介绍此类数据结构的原始论文通常非常深入，但对于试图将此类数据结构应用到具有真实数据的真实系统中的数据工程师来说，这些论文却远远不够。除了偶尔出现一篇介绍数据结构实现的博文，具有这种大规模数据领域特定算法知识的资源仍十分稀缺。

我们想写一本书，既能以友好的方式介绍这些技术性很强的主题，又能更好地回答学生一直以来的疑问“这些知识可以应用到什么地方”。将概率数据结构、流式数据结构和外部存储器数据结构与一个使用中的大规模数据集生态系统联系起来并展示实际用例是个不小的挑战。我们还没有准备好彻底放弃数学，因此将把尽可能多地传达数学直觉作为一项挑战，而不包含任何证明。

我们非常幸运能与具有高级工程背景的插画家 Ines 合作，她创作了极佳的插画来说明一些较复杂的算法内容。你在尝试解释算法的过程中会发现，算法本质上是视觉的，但关于计算机算法的书籍通常没有很多视觉线索。希望本书有助于改变这一点。

每个好故事都需要一个冲突，本书的主要冲突是大数据带来的约束所产生的权衡，一个主题是牺牲数据结构的准确性来节省空间。在复杂的数据管道中找到性能的最佳点并学会如何平衡不同的竞争目标是大规模数据带来的主要挑战，也是本书的主要内容。

很高兴有机会就这样一个令人兴奋且重要的话题写一本书，也非常感谢所有在本书编写过程中提供反馈的人。我们以学者的身份开始撰写本书，但完成时已是数据公司的工程师(这确实是一本实用的书)。希望这些知识可以丰富你的算法工具包并使你能够带着好奇心和信心解决接下来的大数据问题。

作 者 简 介

Dzejla Medjedovic 于 2014 年在纽约石溪大学计算机科学系的应用算法实验室获得博士学位。Dzejla 曾参与多个涉及大规模数据算法的项目，讲授不同级别的算法并曾在微软工作过一段时间。Dzejla 热衷于教学、推动计算机科学教育以及技术转让。目前，她在 Social Explorer 公司担任数据副总裁。

Emin Tahirovic 于 2008 年获得法兰克福歌德大学理论计算机科学硕士学位，于 2016 年获得宾夕法尼亚大学生物统计学博士学位。由于具备统计方法和理论计算机科学背景，他成为计算和统计交叉领域的代表性数据科学研究人员。他曾在 DBahn AG 担任 IT 顾问并定期为制药和科技公司的项目提供咨询服务。Emin 曾在萨拉热窝国际大学担任软件工程助理教授，现任 HAProxy Technologies 高级数据科学家。

Ines Dedovic 在德国亚琛工业大学电气工程系成像和计算机视觉研究所获得博士学位。她曾在 Jülich 研究中心担任研究员，现任自动化公司 Jonas & Redmann 的相机系统的软件开发人员。十多年来，Ines 还从事 3D 动画师、漫画家和教科书插图画家工作。在本书中，她利用艺术和技术技能为技术性概念创建了直观视觉效果。

致　　谢

从开始写书到完成的过程中会发生许多事，在处理生活与工作中所有琐事的同时逐步推进章节创作也并非易事。幸运的是，在这一过程中，我们得到了许多人的支持与鼓励，他们还在临近截止日期前的工作夜里为我们送来食物。

首先，感谢我们的父母：Merdzana 和 Safer 以及 Zikreta 和 Esad。你们平生的经历和引导让我们可以自由地阅读、学习和探索，让我们觉得自己有能力写书，也应该去写书。还要感谢我们亲爱的兄弟姐妹和侄女：Dzejra、Ensar、Ajla、Serif、Mersad、Dalal 和 Lejna，感谢他们在整个写作过程中给予的支持。Emin 还要感谢他的姨妈 Indira 在法兰克福学习期间对他的照顾。

其次，感谢我们的朋友，他们持续关注本书的进展情况。我们的许多朋友都来自计算机科学以外的领域，因此他们对于通读前几章的热情更意义非凡。

感谢萨拉热窝科技学院和萨拉热窝国际大学的学生，他们激发了我们的写作灵感并帮助在写作的不同时期审阅稿件。

此外，还要感谢我们的编辑 Karen Miller，她非常专业、为人亲切，出色地指导我们完成整个过程。她的见解和经验对打造本书至关重要。

在写作过程中，我们曾与 Manning 的许多工作人员合作。Manning 团队追求完美并采用敏捷和早期反馈的方法来出版图书。作为作者，我们发现这非常有用且备受鼓舞。

感谢 Manning 的审稿人：Alejandro Bellogin、Alex Gout、Anto Aravinth、Arno Bastenhof、Christopher Kottmyer、Chunxu Tang、

Clifford Thurber、Daniel Vasquez、Diego Casella、German Gonzalez-Morris、Hilde Van Gysel、Jean-François Morin、Jens Christian Bredahl Madsen、Jim Amrhein、Juan José Durillo Barrionuevo、Juan Antonio Rufes de Vicente、Kelum Senanayake、Manu Sareena、Marcus Young、Mark Bower、Nick Vazquez、Raushan Jha、Rui Liu、Satej Sahu、Sébastien Janas、Stuart Perks、Tim Van Deurzen、Travis Nelson 和 Yuri Kushch。他们的建议为本书增色添彩。

最后，感谢我们的试读者，他们的参与和投入使本书更适合目标读者。

前　言

本书旨在帮助人们构建可扩展的应用程序并了解大规模数据系统下的算法构建块。本书涵盖了构建大规模应用程序的不同算法，包括使用概率数据结构节省空间、处理流式数据、使用磁盘上的数据以及了解数据库系统中的性能权衡。

本书读者对象

本书适合了解基本数据结构和算法的读者。书中的许多内容都以早期数据结构/算法课程中通常涵盖的内容为基础：大部分章节都从展示问题的传统解决方案开始并说明该算法或数据结构在大规模数据的背景下失败的原因。尽管各章的介绍部分讨论了一些基本算法，但这些内容只是对读者本应了解的主题的简短复习。本书的读者还应该掌握中级编程知识以及基本的概率知识。除了需要熟悉Python和伪代码这些基本知识，学习本书无须了解其他任何特定系统或架构(这就是算法的奥妙之处)。

本书主要结构

本书分为3部分，共11章。第Ⅰ部分介绍概率型简洁数据结构，第Ⅱ部分介绍流式数据结构和算法，第Ⅲ部分介绍外部存储器数据结构和算法。以下是各章内容的简要说明。

第Ⅰ部分：基于哈希的草图

- 第 1 章解释大规模数据在现代系统中存在严峻挑战的原因，以及这些挑战对算法和数据结构设计的影响。
- 第 2 章回顾哈希并解释哈希表如何发展以满足大型数据集和复杂分布式系统的需求(如一致性哈希)。哈希方法将在接下来的章节中大量使用，因此该章是第Ⅰ部分的基础。
- 第 3 章介绍近似成员关系问题和有助于解决该问题的两种数据结构：Bloom 过滤器和商过滤器。该章展示用例和误报率分析，以及使用每种数据结构的优缺点。
- 第 4 章描述频率估计问题并介绍 count-min sketch(计数-最小草图，这是一种以非常节省空间的方式解决频率估计问题的数据结构)。该章讨论 NLP、传感器数据和其他领域的用例，以及 count-min sketch 在范围查询等问题中的应用。
- 第 5 章深入了解基数估计、HyperLogLog 算法及其应用。该章通过一个小型实验，展示了从简单的概率计数到完整的 HyperLogLog 数据结构在准确性方面的演变。

第Ⅱ部分：实时分析

- 第 6 章循序渐进地介绍数据流这一算法概念，以及其现实世界表现形式——流式数据(应用程序)。该章使用流式数据架构中的几个实际用例，展示了前几章中的数据结构如何用于流式数据上下文。
- 第 7 章介绍如何保留数据流中的代表性样本或数据流上的滑动窗口。该章指出人们何时可能对有偏差的样本感兴趣，并给出代码示例来展示如何实现将样本偏向于最近看到的数据元组。
- 第 8 章涉及计算连续数据流中数值数据的近似分位数，描述了两种草图数据结构：t-digest 和 q-digest。该章还解释了它们背后的算法并在一个真实的数据集上将两者进行对比。

第Ⅲ部分：数据库的数据结构和外部存储器算法

- 第 9 章介绍外部存储器模型及相关示例，这些示例用于说明在远程存储上处理数据时，输入/输出成本如何支配 CPU。对于习惯于从 CPU 成本方面考虑优化算法的算法设计者来说，该章会转换他们的视角。
- 第 10 章展示支持主流数据库的数据结构(例如，*B* 树和 LSM 树)，并且涵盖数据库引擎设计中的各种读/写权衡。从高层次上理解这些数据结构的工作原理有助于辨别不同风格的数据库，并为应用程序选择适合的数据库。
- 第 11 章着眼于对外部存储器的排序，并且展示了使用合并排序和快速排序的外部存储器优化版本对磁盘上的文件进行排序的最佳算法。该章以排序为例，说明在将批处理问题移至外部存储器时可以进行哪些优化。

本书的各部分相互关联，但第Ⅰ、Ⅱ部分之间的联系更紧密，因为这两部分都涉及 RAM 中的数据结构以及在节省空间的同时最大限度地提高准确性这一主题。第III部分具有独立的主题，只对它感兴趣的读者可以直接跳至这一部分。第Ⅰ、Ⅱ部分之间的阅读顺序也并不绝对，但先阅读第Ⅰ部分可能比直接跳入第Ⅱ部分更容易理解第Ⅱ部分。

第Ⅱ部分和第III部分都以解释模型和背景的章节(第 6 章和第 9 章)开始，强烈建议阅读这些章节，为理解相应部分中的其他章节奠定基础。了解这一点后，可以随意探索本书。编写时，我们尽力使各章自成体系。如果需要，可以随时返回前置章节以获取更多相关知识。我们建议所有读者阅读第 1 章，因为该章解释了当涉及部署在繁忙的大型基础设施中的算法和数据结构时，大规模数据会产生范式转变的原因。

配套资源

书中示例的完整代码及配套资源可通过扫描本书封底的二维码

获取并下载。

书中有几章包含代码，对于一些较复杂的算法以及上下文会明显加大代码复杂度的算法(如外部存储器算法)，我们会返回使用伪代码。我们将 Python 和 R 用于大多数代码片段，并在某些章节中用其创建一些小型实验来演示数据结构性能。读者应能够随意用自己选择的语言来实现编程练习，因为所涵盖的主题并不特定于某一语言或技术。

本书包含代码清单和类似普通文本形式的许多源代码示例。这两种情况下，源代码都被格式化为 Courier New 字体，从而区分于普通文本。有时代码也以粗体突出显示与书中先前步骤相比发生变化的代码，例如将新特性添加到现有代码行时。

许多情况下，原始源代码已被重新格式化；我们添加了换行符并重新设置缩进以适合纸质书中可用的页面空间。极少数情况下，这些还不够，代码清单中还包含续行符(➥)。此外，在正文中描述代码时，源代码中的注释通常已删除。不过许多代码清单都带有代码注释，突出了重要的概念。

关于封面插图

本书封面上的插图名为 *Roussienne*，意为“俄罗斯女人”，取自 Jacques Grasset de Saint-Sauveur 于 1788 年出版的一本书，书中每幅插图皆经手工精心绘制着色而成。

在那个年代，仅通过衣着就很容易识别人们的居住环境、职业和地位。Manning 出版社用图集中的图片重现了几个世纪前地区文化的丰富多样性，以此彰显当今计算机界的创造性和主动性。

目　录

第Ⅰ部分

基于哈希的草图

本部分首先对本书结构及读者对象作一个概述并介绍大规模数据集带来的挑战，然后将探讨概率型简洁数据结构。我们将看到随着数据量的增长和经典数据结构开始从 RAM 中溢出，常规算法领域中的基本问题(如频率估计、成员关系查询和计数不同问题)如何变得更难解决。我们的重点是有助于解决相同问题但所需空间更少的数据结构，但这些数据结构的准确性并非总是 100%。好消息是，误差率通常很低，并且因数据结构存储方面的重大进步得到了很大的补偿。第Ⅰ部分展示的数据结构包括 Bloom 过滤器、商过滤器、count-min sketch、HyperLogLog 和一些紧凑的哈希表变体。这些数据结构对于期望误差率具有高度可配置性并因此具有高度通用性。之后的几章都是关于如何在最少的 RAM 空间中加入最多的功能，并且每一位都很重要。但首先回顾的是哈希表和哈希，它们是许多将要介绍的数据结构的构建块。

第1章

导　论

本章主要内容

- 本书的内容和结构
- 本书与其他算法书籍的区别
- 大规模数据集如何塑造算法和数据结构的设计
- 本书如何帮助你在工作时设计实用算法
- 使大量数据对于当今的系统具有挑战性的计算机和系统架构基础

从你拿起本书开始，就可能存在这样的疑问：用于大规模数据集的算法和数据结构是什么，它们与迄今为止可能遇到过的“普通”算法有何区别。本书的书名是否暗示了经典算法(如二进制搜索、合并排序、快速排序、深度优先搜索、广度优先搜索和许多其他基本算法)以及典型数据结构(如数组、矩阵、哈希表、二叉搜索树、堆)是专门为小型数据集构建的？

这个问题的答案既不简短也不简单，但如果必须用一个字回答，那就是“是”。“大规模数据集由什么构成”这一概念是相对的，取

决于诸多因素，但事实是，我们日常了解和使用的大多数基本算法和数据结构都隐式地假设所有数据都适用于计算机的主内存或随机存取存储器(RAM)。因此，一旦将所有数据加载到 RAM 中，访问其中任何元素都会相对快速和方便。此时，从效率的角度看，最终目标是在最少的 CPU 周期内实现最大的生产力。这就是过去的大 *O* 分析(*O*(.))表达的内容：它通常表示算法在最坏情况下为解决问题而必须执行的基本操作的数量。这些单元操作可以是比较、算术、位操作、内存单元读/写/复制或者任何直接转换为少量 CPU 周期的操作。

但对于为一家从用户那里收集数据的公司工作的数据科学家、开发人员或后端工程师而言，将所有数据存储到计算机的工作内存中通常是不可行的。当今的许多应用程序，例如在银行业、电子商务、科学应用和物联网(IoT)中，通常会处理 TB 或 PB 大小的数据集(即无须在 Meta 和 Google 工作，也可能在工作中遇到大规模数据)。

你可能会问自己，要想通过书中所述技术受益，数据集需要有多大。我们刻意避免量化大规模数据集或“大数据公司”，因为这取决于要解决的问题、工程师可用的计算资源、系统要求等。一些拥有大量数据集的公司也拥有丰富的资源，并且可以通过投资基础设施(如购买大量 RAM)来延缓对可扩展性问题的创造性思考。在具有中等规模数据集，但基础设施预算有限且客户对系统性能要求极高的情况下，开发人员可以像其他人一样从本书介绍的技术中受益。然而，正如我们将看到的那样，即使是拥有几乎无限资源的公司也会选择用巧妙节省空间的数据结构来填充额外的 RAM。

大规模数据问题的存在时间比社交网络和互联网要长得多。最早介绍外部存储器算法(一类忽略程序计算成本而优化更耗时的数据传输成本的算法)的论文之一[1]出现在 1988 年。作为研究的实际动机，作者以大型银行为例，他要在下一个工作日之前，通过当时的工作内存(2~4MB)连夜对 200 万张支票(约 800MB)排序。因此，弄清楚如何在一次只能分类 4MB 支票的情况下对所有支票进行排序与如何以最少的磁盘访问次数进行排序为相关问题，且相关性不断增

加。那时的数据已经开始大幅增长，但更重要的是，它的增长速度比 RAM 中内存的平均大小要快得多。

数据快速增长的主要结果以及本书中激励算法的主要思想是，当今大多数应用程序都是数据密集型的。数据密集型(与 CPU 密集型相反)意味着应用程序的瓶颈来自来回传输数据和访问数据，而不是在数据可用时对其进行计算。这一事实是设计大型数据集算法的核心，简洁数据结构和面向外部存储器的算法的想法正是源于此。在 1.4 节中，将深入探讨计算机中数据访问速度远低于计算速度的原因。

当跳出单个计算机的视角时，情况会变得更复杂。如今大多数应用程序都是分布式的，有着复杂的数据管道，成千上万的计算机通过网络交换数据。数据库和缓存是分布式的，许多用户可以同时添加和查询大量内容。现在的数据格式变得多样化、多维化和动态化。为达到有效的目的，应用程序需要快速地适应变化。

在流式应用程序[2]中，数据在不经存储的情况下有效地得到应用，应用程序需要以一定程度的准确性捕获数据的相关特征，使其相关和有用，而无须再次扫描。这种新的环境需要新一代的算法和数据结构，即一个新的应用程序构建器工具箱，它经过优化可以解决大规模数据系统特有的挑战性问题。本书的目的就是准确地讲授用于开发可扩展应用程序的基本算法技术和数据结构。

1.1　示例

为说明本书的主题，请考虑以下示例：你正在一家媒体公司参与一个有关新闻文章评论的项目。你将获得一个包含以下相关基本元数据信息的大型评论库。

```
{
    comment-id: 2833908010
    article-id: 779284
```

```
    user-id: 9153647
    text: this recipe needs more butter
    views: 14375
    likes: 43
}
```

你将查看大约 30 亿条用户评论，数据大小总计 600GB。关于数据集，需要回答一些问题，包括确定最主流的评论和文章，等根据主题和评论中出现的常见关键字对文章进行分类，等等。但首先需要解决在多次抓取中产生的重复问题，并确定数据集中不同评论的总数。

1.1.1 示例解决方法

在数据结构中存储唯一元素的常用方法是创建一个键值字典，其中每个不同元素的唯一 ID 映射到其频率。实现键值字典的库有很多，如 C++中的 map、Java 中的 HashMap、Python 中的 dict 等。键值字典通常被实现为平衡二叉树(如 C++的 map 中的红黑树)或哈希表(如 Python 的 dict)。

红黑树与哈希表实现

除了以快速对数时间运行的查找/插入/删除外，树形字典实现还提供同样快速的前继节点/后继节点操作，即使用字典顺序有效地来回探索数据的能力。大多数哈希表实现不具备按字典顺序有效遍历项的能力，但哈希表实现在最常见的查找/插入/删除操作上提供了快速常数时间性能。

为简单起见，假设我们使用的是 Python 的 dict 这一哈希表。将评论 ID 作为键和将评论 ID 的出现次数作为值可以有效地帮助消除重复(见图 1.1 左侧的<评论 ID；频率>字典)。

但是，存储 30 亿条评论的<评论 ID;频率>对可能需要多达 24 GB 的空间，其中每对使用 8 字节(评论 ID 占 4 字节，频率占 4 字节)。根据底层哈希表的实现方法，数据结构需要 1.5 或 2 倍于用于簿记的

元素(空槽、指针等)所占用的空间，这样就需要接近 40 GB 的空间。

如果还要根据某些感兴趣的主题对文章进行分类，可以再次使用字典(也可用其他方法)，为每个主题(如体育、政治、科学等)构建一个单独的字典，如图 1.1 右侧部分所示。此处<文章 ID；关键字频率>字典的作用是统计所有评论中与主题相关的关键字出现的次数，例如文章 ID 为 745 的文章在其相关评论中有 23 个与政治相关的关键字。我们使用大的<评论 ID；频率>字典预过滤每个评论 ID，仅考虑不同的评论。这种类型的单个表可以包含数千万个项，总计接近 1GB；假设为 30 个主题维护这样的哈希表，仅数据的成本就可能高达 30GB，总共大约为 50GB。

我们希望这个小例子能证明我们可以从一个相当普遍和朴素的问题开始，逐步探索一些无法放入内存的笨重数据结构。

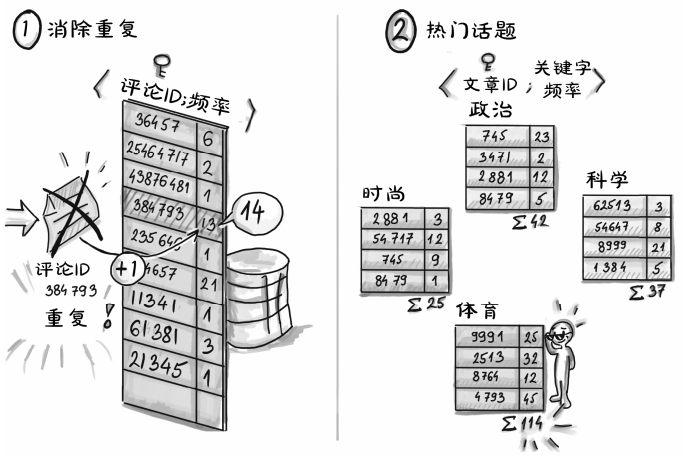

图 1.1　本例中构建了一个<评论 ID;频率>哈希表来帮助存储不同的评论 ID 及其频率。传入的评论 ID 384793 已包含在表中，并且其频率计数有所增长。我们还构建了与主题相关的哈希表，对于每篇文章，计算相关关键字在其评论中出现的次数(例如，在体育主题中，关键字可能是“足球”“球员”“进球”等)。对于有 30 亿条评论的大型数据集，这些数据结构可能需要几十到 100GB 的 RAM 内存

你可能会想，我们不能事先将几个数字相乘，然后轻松预测数据结构将变得多大吗？在现实生活中，通常并不能实现，因为很少有人会在考虑大规模数据的情况下从头开始设计系统。公司一开始往往试图创建一个行之有效的系统，但后来可能因为自己的成功而成为受害者，用户群在短时间内迅速增长，而由已经离职的开发人员设计的旧系统需要应对新的高要求工作量。大多数情况下，系统的某些部分会根据需要重新进行设计。

当数据集中的项数量变大时，每项的每一额外位都会增加系统的负担。软件开发人员所熟知的常见数据结构可能会变得太大而无法有效使用，需要更简洁的替代方案(见图 1.2)。

图1.2 包括哈希表在内的大多数常见数据结构在存储和管理大量数据时都会变得非常困难

1.1.2 本书给出的解决方法

面对令人生畏的数据集大小，存在多种选择。事实证明，如果满足于一个小的误差范围，则可以构建一个在功能上类似于哈希表的数据结构，它相较之下更紧凑。本书的第 I 部分包含一系列简洁数据结构，这些数据结构使用少量空间来解决下列常见问题。

- 成员关系——评论/用户 X 是否存在？
- 频率——用户 X 评论了多少次？最主流的关键字是什么？

- 基数——真正不同的评论/用户有多少？

与哈希表相比，这些数据结构在处理包含 *n* 项的数据集时使用的空间要少得多(每项仅需要 1 或更少的字节，而哈希表中的每项需要 8～16 字节)。

第 3 章讨论的 Bloom 过滤器将使用只有<评论 ID；频率>哈希表 1/8 的空间，并且将以大约 2%的误报率回答成员关系查询。由于本章为介绍性章节，因此这里不对得出这些数字的详细数学方法作深入阐述，但值得强调的是 Bloom 过滤器和哈希表之间的区别，即 Bloom 过滤器不存储键本身。Bloom 过滤器计算哈希键并使用它们来修改数据结构。因此，Bloom 过滤器的大小主要取决于插入的键的数量，而不是它们的大小或数据类型。

第 4 章介绍的另一种数据结构 count-min sketch 将使用只有<评论 ID；频率>哈希表 1/24 的空间来估计每个评论 ID 的频率，在超过 99%的案例中估计得到的频率略高。还可以使用 count-min sketch 来替换<文章 ID；关键字频率>哈希表，每个主题哈希表仅使用大约 3MB 的空间，这只有原始方案成本的约 1/20。

最后，第 5 章的数据结构 HyperLogLog 可以仅用12KB 估计集合的基数，误差小于真实基数的 1%。

如果进一步放宽对这些数据结构准确性的要求，则需要使用的空间会更少。因为原始数据集仍然保留在磁盘上，所以也有方法可以控制偶然误差，使我们免于受误报所困。我们只需要一点额外的努力来进行验证。

1. 流式评论数据

我们很有可能会在快速移动的事件流而不是静态数据集的上下文中遇到新闻评论和文章的问题。假设本例中的事件构成对数据集的任何修改(例如点赞或者插入/删除评论或文章)，并且事件作为流式数据实时到达系统。第 6 章将介绍有关流式数据上下文的更多信息。

注意，在此设置中，也可能会遇到重复的评论 ID，但原因不同：

每次有人在特定评论上点赞时，我们都会收到具有相同评论 ID 的事件，但点赞属性的计数会更改。考虑到事件会全天候快速到达，并且我们无法存储所有事件，因此对于许多感兴趣的问题，只能采取近似解决方案。大多数情况下，我们感兴趣的是实时计算数据的基本统计数据(例如过去一周每条评论的平均点赞数)，如果无法存储每条评论的点赞数，则可以采用随机抽样的方法。

可以使用第 7 章介绍的伯努利抽样算法从数据流中抽取一个随机样本。例如，如果你曾经通过摘花瓣的方式随机判断“他/她爱我，他/她不爱我”，可能最后得到的就是“伯努利抽样”花瓣。这种抽样方法十分便利，适用于单次遍历的数据上下文。

回答有关评论数据的一些更细化的问题(例如位于被点赞前10%的评论需要获得多少点赞)也会牺牲准确性以获得空间。可以在有限的、现实的、快速的内存空间内维持一种可以完整查看数据的动态直方图(见第 8 章)，并使用此草图或数据摘要来回答有关查询完整数据的任何分位数的问题，但会有一些误差。

2. 数据库中的评论数据

最后，我们可能希望以持久格式(例如磁盘/云数据库)存储所有评论数据并在其上构建一个系统，以便随着时间的推移快速插入、检索和修改实时数据。这种设置追求准确性而不是速度，因此只要能够保证查询的准确性为 100%，那么在磁盘上存储并检索大量数据的速度即使很慢也无妨。

将数据存储在远程存储器上并对其进行组织以进行高效检索涉及的是一个被称为外部存储器算法的算法范式主题，将在第 9 章探讨。外部存储器算法解决了现代应用程序中一些最相关的问题，如数据库引擎及其索引的设计和实现。在前面的评论数据示例中，我们需要询问自己是在构建一个包含大部分静态数据但用户不断查询即读优化)的系统，还是一个用户经常添加新数据并修改数据但只偶尔查询数据的系统(即写优化)。或者，也许是两者的组合，快速插入和快速查询数据同样重要(即读写优化)。

很少有工程师实现自己的存储引擎，但几乎所有人都使用存储引擎。为在不同的备选方案之间做出明智的选择，需要了解它们背后的数据结构是什么。插入/查找权衡是数据库固有的，反映在运行于 MySQL、TokuDB、LevelDB 和许多其他内存引擎之下的数据结构的设计中。用于构建数据库的最流行的数据结构包括 B 树、B^ϵ 树和 LSM 树，各自为不同的工作负载服务，将在第 10 章讨论这些不同类型数据结构的性能和权衡。此外，我们可能对解决磁盘上数据的其他问题感兴趣，例如按字典顺序或出现次数对评论排序。为此，需要一种排序算法，以便有效地对数据库或磁盘文件中的数据进行排序。本书的最后一章(第 11 章)中将介绍如何做到这一点。

1.2　本书的结构

正如前面部分所述，本书围绕 3 个主题展开，分为 3 部分。

第Ⅰ部分在引入主题后主要介绍基于哈希的草图数据结构。首先回顾哈希表和为大规模数据集开发的特定哈希技术。尽管有关哈希的章节为回顾章节，但建议你将其用作哈希的复习章节，并借此机会了解为处理大型数据集而设计的现代哈希技术。草图是基于哈希的，因此第 2 章也可以作为第 3～5 章的基础。第 3～5 章介绍的数据结构(如 Bloom 过滤器、count-min sketch、HyperLogLog 及其替代结构)已经在数据库、网络等领域大量应用。

第Ⅱ部分介绍数据流。从经典技术(如伯努利和蓄水池抽样)到更复杂的方法(如从移动窗口抽样)，包含许多适用于不同流式数据模型的抽样算法，然后使用创建的样本来计算总和或平均值的估计值等。该部分还介绍计算 ε 近似分位数的算法，如 q-digest 和 t-digest。

第Ⅲ部分介绍数据保留在 SSD/磁盘上时的算法技术。首先介绍外部存储器模型，然后介绍解决搜索和排序等基本问题的最佳算法，阐明该模型中的关键算法技巧。这一部分还涵盖支持现代数据库的数据结构，如 B 树、B^ϵ 树和 LSM 树。

1.3 本书的不同之处及目标读者

目前已有许多关于经典算法和数据结构的优秀图书。用于大规模数据集的算法和数据结构正在慢慢进入主流教科书，但世界也在快速发展，我们希望本书能够提供最先进的算法和数据结构的纲要，从而帮助数据科学家或开发人员在工作中处理大型数据集。

本书旨在平衡 Python 中的理论直觉、实际用例和代码片段。我们假设读者具备一些算法和数据结构的基础知识，因此如果你没有学习过基本算法和数据结构，则应该在学习该主题之前了解相关知识。大规模数据算法是一个非常广泛的主题，本书旨在提供初步的介绍。

大多数关于大规模数据集的书籍的重点在于特定的技术、系统或基础设施。但本书不关注具体的技术，也不假设熟悉任何特定技术。相反，本书涵盖了使系统具有可扩展性的底层算法和数据结构。

通常，涵盖大规模数据的算法方面的书籍侧重于机器学习。然而，处理大型数据的一个重要方面不是专门根据推理数据含义，而是无论数据内容，都要根据数据大小进行有效处理。这在已有文献中通常被忽视，本书旨在填补这一空白。

有一些优秀的书籍解决了大规模数据集的几个专业方面的问题[3]。本书意在一次性在一本书内展示这些不同的主题，并且引用相关主题的前沿研究和技术论文。最后，希望这本书能够以脚踏实地的方式讲授更高级的算法材料，提供数学直觉，而不是描述有关该主题的大多数资源的技术证明。配图有效地表达了一些更高级的技术概念，希望你能喜欢。

至此，本书的内容和结构已经基本介绍完毕，下面讨论激发本书主题的核心问题。

1.4　为什么大规模数据对当今的系统如此具有挑战性

计算机和分布式系统架构中存在的许多参数可能影响给定应用程序的性能。计算机在处理大量数据时面临的一些主要挑战源于硬件和通用计算机架构。本书不是关于硬件的，但是为了对大规模数据设计有效的算法，理解一些使数据传输成为如此巨大挑战的物理约束十分重要。本章主要讨论 CPU 和内存速度之间的巨大不对称性、不同级别的内存及每个级别的速度和大小之间的权衡，以及延迟与带宽的问题。

1.4.1　CPU 内存性能差距

将讨论的第一个重要的不对称性在于计算机中 CPU 操作和内存访问操作的速度，也称为 CPU 内存性能差距[4]。图 1.3 显示了从 1980 年开始处理器内存访问和主内存(DRAM 内存)访问的速度之间的平均差距，其中性能以每秒内存请求数(延迟的倒数)表示。

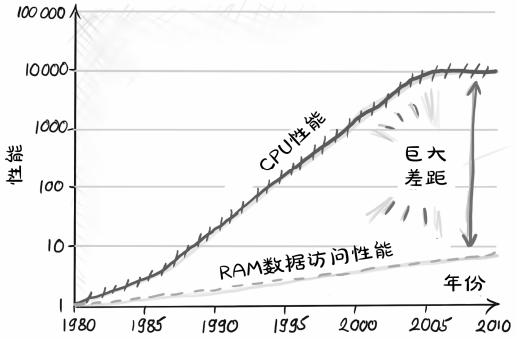

图 1.3　CPU 内存性能差距图，来自 Hennessy & Patterson 的计算机架构。该图显示了对 CPU 和 RAM 主内存的内存访问速度(随着时间的推移，每秒的平均内存访问次数)之间不断扩大的差距。纵轴为对数刻度。由图可知，截至 2005 年，处理器性能的提升约为每年 1.5 倍，而主内存访问的性能提升每年只有大约 1.1 倍。自 2005 年以来，处理器的提升速度有所放缓，但通过使用多核和并行性，这种情况正在得到改善

直观上，这个差距表明执行计算比访问数据快得多。如果固守只关心优化 CPU 计算的思维模式，那么很多情况下，分析将无法与现实相吻合。

1.4.2 内存层次结构

除了 CPU 内存差距，计算机中内置的不同类型的内存也存在层次结构，不同层次的内存具有不同的特性。最重要的权衡是，快速的内存很小(且昂贵)，而大的内存很慢(但便宜)。如图 1.4 所示，从最小和最快的开始，计算机内存层次结构通常包含以下级别：寄存器、一级缓存、二级缓存、三级缓存、主内存、固态硬盘(SSD)和/或机械硬盘(HDD)。最后两个是持久性(非易失性)内存，这意味着如果关闭计算机，数据就会保存下来，因此适合存储。

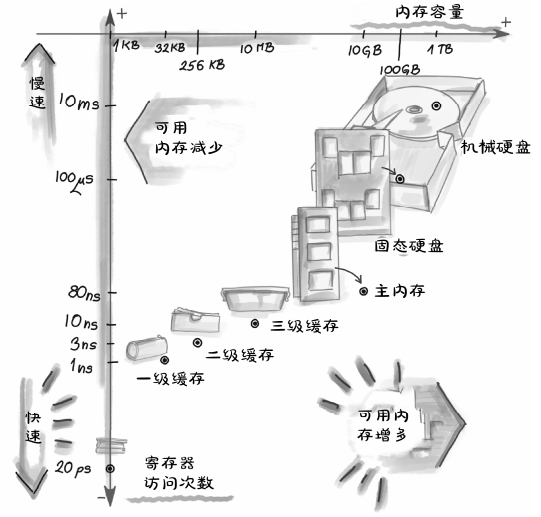

图 1.4 计算机中不同类型的内存。左下角的寄存器非常快，但内存容量非常小，从此处向上(速度变慢)和向右(内存容量变大)分别是一级缓存、二级缓存、三级缓存和主内存，一直到 SSD 和/或 HDD。在同一台计算机中混合不同的内存，通过将每个级别用作下一个更大级别的缓存，从而产生同时具有速度和存储容量的错觉

在图 1.4 中，可以看到示例架构[5]中每个级别内存的访问次数和容量。这些数字因架构而异，并且相较于具体数值，不同访问次数之间的比值更为有用。例如，从缓存中提取数据的速度大约是从磁盘中提取数据的 100 万倍。

硬盘和指针是计算机中仅存的一些机械部件，它们的工作原理很像留声机。访问磁盘数据时，将机械指针放在正确的轨道上会花费大量时间。一旦指针位于正确的轨道上，数据传输就会非常快，传输速度具体取决于磁盘旋转的速度。

1.4.3　延迟与带宽

一个类似的现象是“延迟滞后于带宽”[6]，且存在于不同类型的内存中。从微处理器到主内存、硬盘和网络，各种系统的带宽在过去几十年中得到了极大的改善，但延迟并没有得到同样的改善。延迟是许多场景中的重要衡量标准，在这些场景中，常见的用户行为涉及许多小的随机访问，而不是一个大的顺序访问。

为抵消昂贵的初始调用成本，不同级别内存之间的数据传输是在多个项的块中完成的。这些块被称为缓存行、页或块，具体称呼取决于使用的内存级别。它们的大小与相应级别的内存大小成正比：对于缓存，它们在 8～64 字节的范围内；对于磁盘块，它们可以达到 1MB[7]。由于空间局部性，我们希望程序访问彼此相邻且时间接近的内存位置，因此在顺序块中传输数据可以有效地预取不久之后可能需要的项。

1.4.4　分布式系统的情况

如今的大多数应用程序都在多台计算机上运行，并且将数据从一台计算机发送到另一台计算机会增加另一个级别的延迟。计算机之间的数据传输时间可以从数百毫秒到几秒不等，具体取决于系统负载(例如访问同一应用程序的用户数量)、到目的地所需的跃迁数以及架构的其他细节(见图 1.5)。

图 1.5 由于网络负载和复杂的基础设施，云访问次数可能会很多。访问云可能需要数百毫秒甚至数秒。可以将云视为比硬盘更大、更慢的另一层内存。一般很难提升云应用程序的性能，因为在云上访问或写入数据的次数是不可预测的

1.5 基于硬件来设计算法

在了解了现代计算机架构的一些关键方面之后，第一个重要的收获是，尽管技术不断改进(例如，SSD 是一个相对较新的发展方向，解决了硬盘的许多问题)，但其中一些问题(如速度和内存大小之间的权衡)不会很快消失。造成这种情况的部分原因纯粹是物理方面的：要存储大量数据，就需要大量空间，而光速设定了数据从计算机(或网络)的一部分到另一部分的传输速度的物理限制。为了将其扩展到计算机网络，本书将引用一个示例[8]，表明对于相距 300m 的两台计算机，数据交换的物理下限为 1ms。

因此，我们需要设计可以避开硬件限制的算法。设计能够适应小而快的内存级别的简洁数据结构(或获取数据样本)会有所帮助，因为这样将避免代价高昂的磁盘寻道。换句话说，减少空间可以节省时间。

然而，在许多应用程序中，仍然需要处理磁盘上的数据。因此，设计具有优化的磁盘访问模式的算法和能够实现最少内存传输的缓存机制很重要。进一步来讲，这与如何在磁盘上(例如在关系数据库中)布局和组织数据有关。基于磁盘的算法更喜欢对磁盘进行平滑扫描，而不是随机跃迁；通过这种方式，可以利用良好的带宽以避免较差延迟，因此将执行许多随机读取/写入的算法转换为执行顺序读取/写入的算法较为有意义。本书将介绍如何在考虑与空间相关问题的同时转换经典算法并设计新算法。

不过，同样重要的是要记住，现代系统除可扩展性外还有许多性能指标：安全性、可用性、可维护性等。真正的生产系统需要高效的数据结构和底层运行的算法，但为了让所有系统内容都能为客户服务，还要有很多花哨的性能起到点缀作用(见图 1.6)。事实上，随着数据量的不断增加，设计高效数据结构和算法变得比以往任何时候都更重要，希望在接下来的内容中，你能学到如何做到这一点。

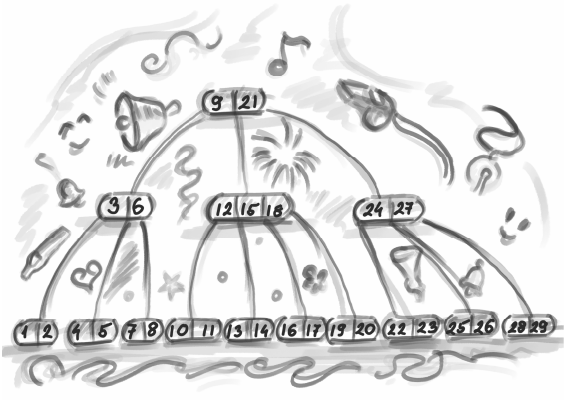

图1.6　一个花哨的高效数据结构

1.6　本章小结

- 如今的应用程序会快速生成和处理大量数据。传统的数据

结构(如键值字典)可能会变得太大而无法放入 RAM 内存，这可能会导致应用程序因输入/输出瓶颈而阻塞。

- 为有效地处理大型数据集，可以设计节省空间的基于哈希的草图，借助随机抽样和近似统计进行实时分析，或者更有效地处理磁盘和其他远程存储上的数据。
- 本书是基本算法和数据结构书籍/课程的自然延续，将介绍如何将基本算法和数据结构转换为可以很好地扩展到大型数据集的算法和数据结构。
- 大数据之所以成为当今计算机和系统的主要问题，关键原因在于 CPU(和多处理器)速度的提高比内存速度的提高快得多，我们要在计算机中不同类型内存的速度和大小之间进行权衡，并且考虑延迟与带宽的关系。这导致应用程序处理数据的速度比执行计算的速度慢。这些趋势不太可能很快改变，因此解决输入/输出成本和空间问题的算法和数据结构只会随着时间的推移变得越来越重要。
- 在数据密集型应用程序中，优化空间意味着优化时间。