12堂趣味逻辑课

避免低效
烦琐的重复工作

畅销书
《神逻辑：不讲道理的人怎么总有理》
作者新作

Bad Choices

How Algorithms Can
Help You Think Smarter and Live Happier

[美] 阿里·阿莫萨维 著
(Ali Almossawi)
高洁 译

MIT媒体实验室老师，
苹果公司软件专家**阿里·阿莫萨维** 作品

互联网之父、图灵奖得主、谷歌首席互联网科学家，
欧洲核子研究中心粒子物理学家，MIT多媒体实验室教授 联袂推荐

中信出版集团

目录

文前

你可以终日在知识的海洋中遨游，却不会因此而弄湿自己。

——诺顿·加斯特《神奇的收费亭》

前言 生活中，逻辑无处不在

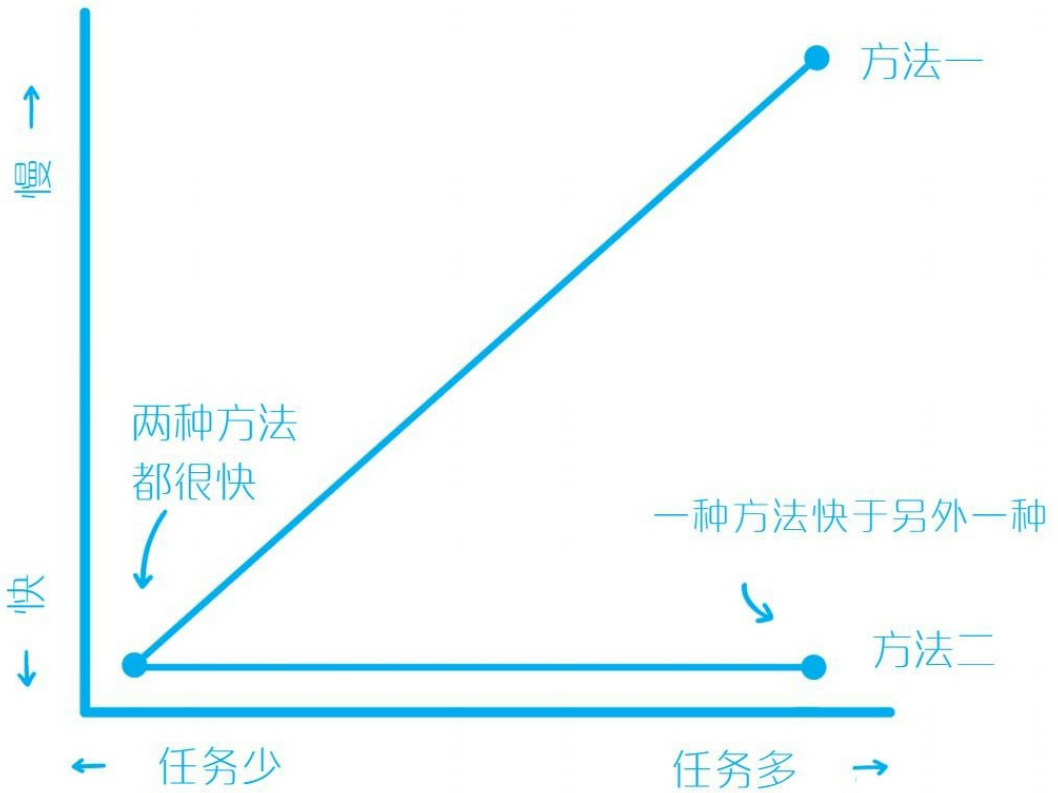
不知你是否听说过以下有趣的故事：著名物理学家理查德·费曼（Richard Feynman）在看到有人将盘子扔到空中后受到启发，从而创造出了荣获诺贝尔物理学奖的方程式；约翰·冯·诺依曼（John von Neumann）依据一位朋友关于人脑记忆储存的理论模拟出了计算机的存储模式；查尔斯·达尔文（Charles Darwin）在动物园看到的一只猩猩像孩子一样的顽劣，进而衍生出了进化论的想法。费曼、冯·诺依曼、达尔文还有很多其他伟大的科学家其实有一共同点，那便是在他们的眼中，物理、数学、科学无处不在，时时围绕在他们的生活之中，远远超越了实验室的方寸之地。

纵然你没有要得诺贝尔奖的壮志，在日常生活中，你也可以通过算法建模的方法来解决

问题。事实上，算法模型的应用在不经意间已经融入了人们的生活，用于解决各种各样的问题。比如，在一堆衣服中找到一双袜子，决定什么时候去杂货店，如何安排当天的任务排序等。算法是一系列解决问题的清晰指令，代表着用系统的方法描述解决问题的策略机制，它从一些输入开始，止于一个输出，这就是算法的特性。有趣的是，现存公元前1800年到公元前1600年左右的古巴比伦碑上就留下了类似的痕迹。古巴比伦人在石头上记载了他们的决策过程，例如复利或者测量出水池的宽度和长度、根据高度来计算容积的方法等。这也就是说，古巴比伦人的程序是由一系列明确的指令和步骤构成的：他们有输入，也有输出，最终得到结论后终止，且结论为有效。因此，算法可以说是由来已久，在过去几个世纪里，无数的数学家、科学家为这一学科奠定了基础。而在计算机科学诞生之后，这些特征变得尤为重

要——正是这种特点可以让计算机通过人为控制的方式来执行任务。

尽管算法在我们的生活中如此重要，但关于算法的讨论往往聚焦于复杂的细节之上——“怎么做”，从而忽略了算法带来的实际的应用效果。我们刚刚提到的看似简单的日常工作，可以通过多种不同的方式执行。我们对实际应用的意识越强，便越能以最有效的方式来发挥所长完成任务。想想看，这种意识就像是强化了我们的直觉一样。这便是我写这本书的灵感来源。本书旨在通过强调解决日常任务的不同方式来向读者介绍逻辑算法思维，并指出这些方法彼此间的相关性。例如，寻找一件适合你的衬衣时可能有两种方法，用图形表示出来就如下图所示。[1]



这些线条的表现形式名为线性方程和对数方程，关于这两种模型，本书将会进行集中整理和探讨。对于有些事情来说，这两种方法是有可比性的，需注意的是随着事件数量的增加而发生的变化。本书重点选择了12个人们熟知的场景，如客厅、裁缝店和百货商店。在每一个场景中，我们都有很多潜在任务需要完成。在每个案例之后，会有一段场景描述、评论和

开放讨论，将这些应用场景与计算机科学的具体概念相关联。此外，本书还会介绍至少两种可行的解决方案来解决基本任务。一种方法速度较慢，另一种较快。通过这种对比的方式，向大家揭示逻辑算法思维的优势，虽然这听来有点挑衅的意思。本书的标题受计算机科学家唐纳德·克努特（Donald Knuth，中文名高德纳）的启发，他认为逻辑算法是“好的”，是一种更加快速而有效的解决方案。[2]

---

[1] 本书中的所有线条都是基于重对数图尺绘制的。

[2] 在这里需要强调的一点是，在生活中，这些好与坏的定义并不能一概而论，比如在学习的过程中，“快”就不是一种美德。经验告诉我，在任何学习过程中，都不可拔苗助长，因为欲速则不达。

引言 一种更高效的思维工具

为何要强调相对性？

对比的效果令人震惊。对于大部分孩子来说，人生中首先学会的往往是抽象的概念，例如大和小。比如，当一个孩子问“在自然博物馆里的泰坦龙有多高”时，如果你回答“小一些的有17英尺（约5.2米）高”，对他们来说，这样的回答并不具有实际的意义，因为他们并没有这种具象的概念。对孩子而言，或许这样的回答才更有意义——“如果苏珊女士、玛格丽特女士和雅沙先生依次站在前一个人的肩膀上，那么雅沙先生可能刚好够到泰坦龙的下巴。”

或许，以相对大小的模式来进行思考是我们与生俱来的能力。近年来的实验似乎表明，婴儿在面对图像变化时，要比面对图像数量的变化时，大脑活动更强烈。还有一些针对偏远

地区人口进行的研究实验表明，没有受过正规教育的人，对于数字也有数量级大小的概念。这似乎是一种人类与生俱来的直觉。

有一类人希望将这种直觉放大，为大家所用，这类人就是计算机科学家。正是这种与生俱来的“直觉”给了他们认知能力和快速解决问题的能力，并且这种方式在解决问题时，比其他方法更有效。这一结论给我们带来了启发，在深挖某一领域时，以相对的逻辑思维模式来看待事物，也会是一种非常有效的方式。如果把这种相对的逻辑思维模式看作一种在小学时学到的数学符号，这种符号会在求学生涯中一直延用下去。

这一理念是我撰写本书的动机之一。在过去很长一段时间里，尤其是在学校求学期间，我一直在运用对比、预估和近似值的方法来理解各种概念，但我一直不敢向别人承认，因为

这看起来似乎并不是一种高级的学习方法。直到我读了《量子怪杰：保罗·狄拉克传》（*The Strangest Man*）[1]和《心智社会》（*The Society of Mind*）这两本书之后，我才意识到，认为这种思维方式有效的不止我一个人。再后来，我读了《科学与工程中的洞察力》（*The Art of Insight in Science and Engineering*）等著作后，我发现他们都在讨论类似的观点和洞察。

我希望本书可以给读者带来更多的思考，在生活中权衡各种决策所带来的利弊时有所参考。本书并非想告诉你如何更好地将袜子进行配对，对于大多数人来说，他们早就具备这种本能了，我希望本书可以给你带来更多的反思，启发你对着镜子反问自己：“我没有想到，原来我还可以用这种方法来思考问题。”与批判性思维类似，逻辑算法思维是一种非常奏效的思维工具，拥有无限的潜能让决策和行为变得

更高效。

为何聚焦于日常生活？

逻辑算法可能很复杂，但也很重要，而且它已经成了我们生活的一部分，只是我们并没意识到或者说并没有在意而已。当我们把生活中的某些方面作为逻辑算法的典型模型来呈现时，我们就会发现，从很多方面来看，逻辑算法对我们的生活大有裨益。

逻辑算法能产生共鸣：本书中列出了很多插图来证明这一点，通过图解说明算法是非常有效的。图解不仅会为原本平淡的行文带来吸引力，也更容易让读者置身一个与实际情况相关联的环境中，吸引和鼓励读者进行更为复杂的推理，将已知和未知的知识进行关联。这也就解释了类比法为何是行之有效的。

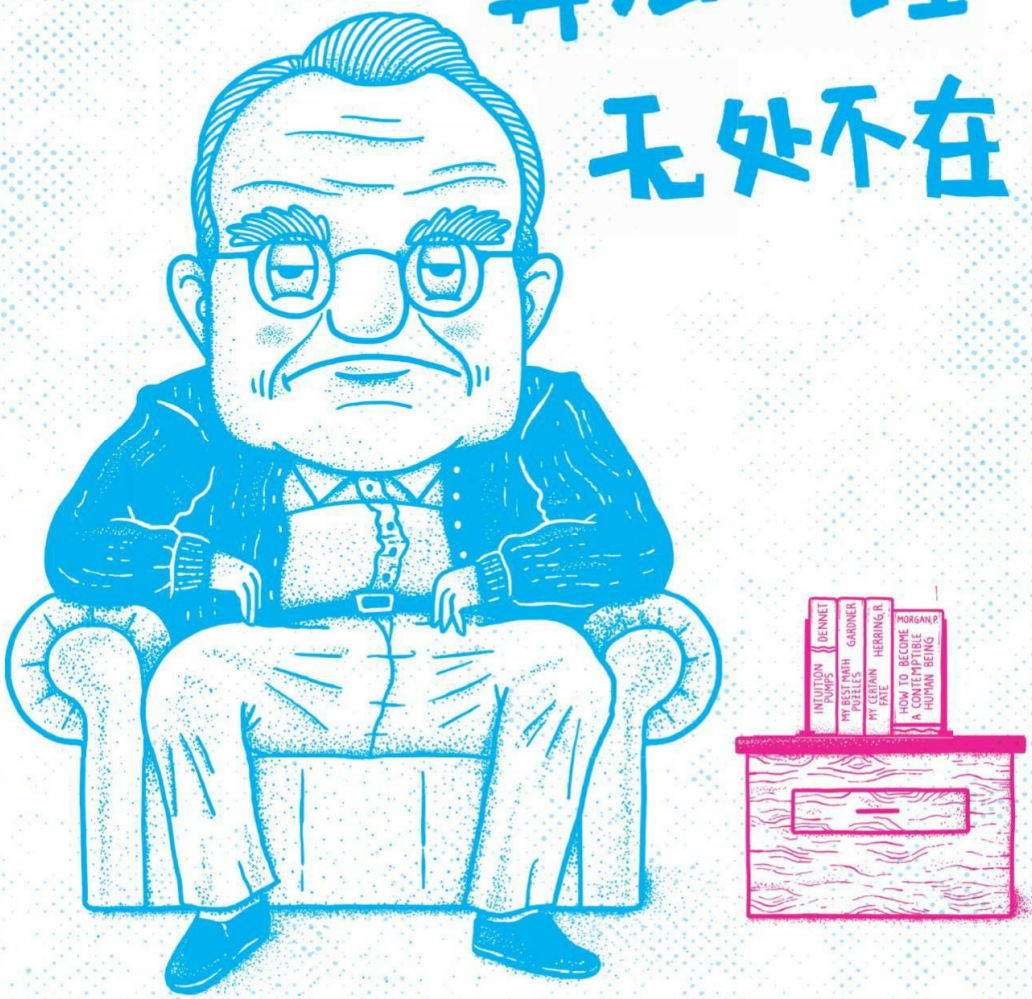
逻辑算法具有互动性：如果你回溯人类的

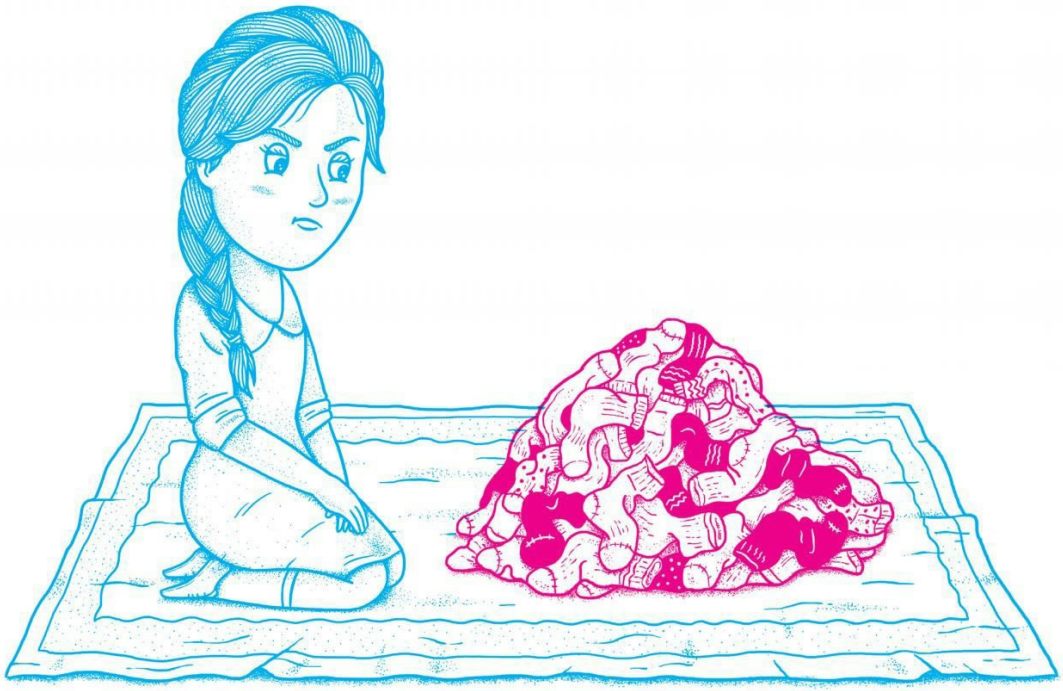
历史，可能会发现，很多你可以记得住名字的人其实都是勤于动脑的学徒而不是只会记笔记的人。逻辑算法常常被类比为食谱，但是我却觉得食谱还是太像古板的笔记了：枯燥、生硬而乏味。在这样的模式中，你就像一个容器，导师的任务是将知识灌进去。借用另一个比喻，这就像是看一场情景喜剧，你作为观众发笑，而对于别人来说，你就变成了喜剧中的演员，别人也在看着你发笑。在本书中，每一课都是通过一个日常情景来呈现的，希望你可以借助这些场景之间的互动、交流和思考，关联自己的生活日常，形成自己的理解，让理论源于生活又服务于生活。我相信这种互动方式能带来更引人入胜的阅读和学习体验。我关于童年时期学习的记忆，最清晰的就是同家长或者老师的对话中，他们都强调学习方法跟智力一样重要。

逻辑算法承认多种结论：我最喜欢的一句

关于学习的名言来自弗朗西斯·培根（Francis Bacon），他说：“这种附带的、有干扰性的使用并不比主要的、有意的使用价值低。”（That use which is collateral and intervenient is no less worthy than that which is principal and intended.）一个问题可能有很多种答案。因此，更具探索性的作品能给不同的人带来不同的结论。有人可能会把它们想象成科学博物馆——爸爸妈妈带着孩子走到一台陈列的设备前，看一下旁边的标签，然后试着向孩子们介绍这台设备。没有人天生就是科学家，也没有人摇身一变就能成为科学家，但是从这场体验中，每个人都获得了有价值的信息。

在生活中，
算法思维
无处不在





---

[1] 在本书中，有一个关于奥利弗·海维赛德（Oliver Heaviside）的有趣片段，人们叫他“刻薄的隐士”，他教授工程数学的方法是实用主义的。“工程师称赞海维赛德的教学方法非常实用，而数学家却嘲笑他不够严谨。海维赛德可没有时间卖弄学问。（‘难道我应该因为我不懂消化的原理而拒绝吃晚饭吗？’）”

第1课 配对 别笑，这个低效的人可能就是你

女男爵玛吉·万（Baroness Margie Wan）来自一个曾经颇具影响力的维也纳家族，而最近她却因为向美国走私健达奇趣蛋而被指控了。玛吉如今身在伯尔尼做换工，这是她第一次叠衣服。玛吉发现她的寄宿家庭成员每半个小时就能因为出汗弄湿一双袜子，这就让找袜子和袜子配对变成了最耗时的工作。幸运的是，这些袜子的尺寸和颜色不尽相同。

提示：这里可能存在多个任务，可能需要从最基础的做起。

你是否曾考虑过生物学特征记忆对人类的重要性？当人仰面坐在椅子上，把一只手放在前额，闭上眼睛，回想一首诗歌或者一个方程式，抑或一个电话号码——这是典型的人类形象。想象一下，如果人类没有这些特征会经历

多么痛苦而挣扎的一生，就像痴呆症患者一样。对于初学者来说，你不得不重复很多相同的工作。就像电影《记忆碎片》（*Memento*）的主人公莱昂纳多·谢尔比一样，由于患有短期记忆丧失症，他必须根据自己支离破碎的记忆来寻找杀害自己妻子的凶手。

我在本书的开篇就提到了这一点。事实上，那种快速解决问题的方法之所以快，是因为有效利用了内存。[1]比如AlphaGo（阿尔法围棋）在2016年击败了世界围棋冠军，它的能力不仅来自专家，更来自自我学习，从而积累了更强大的工作记忆。[2]在本书中，我们将看到更多解决问题的方法，这些方法虽很简单，但很快捷，因为它们可以避免重复作业。

让我们先回归正题，回到袜子的问题上。回到可怜的因走私巧克力而被指控的老玛吉·万的问题上，具有讽刺意味的是，她的女男爵身

份对联邦探员来说并不起作用。此刻，她正在对着那一大堆需要配对的袜子发愁。针对这个问题，我提出两种可行的解决方法：

目标：从一堆衣服中将袜子拣出来并进行配对。

方法一：选择一只袜子，然后从这一堆衣服中找出配对的一只，放在一旁。然后再选一只袜子，再从这一堆衣服中找出配对的一只，放在一旁。重复操作。

方法二：选择一只袜子，放在一旁，再挑拣出另外一只袜子，如果这只袜子跟已经选出的袜子可以配对，就放在一起，如果不能配对，就将这一只袜子放在单只袜子的行列中，直到找出尺码和颜色均合适的一只为止。[3]

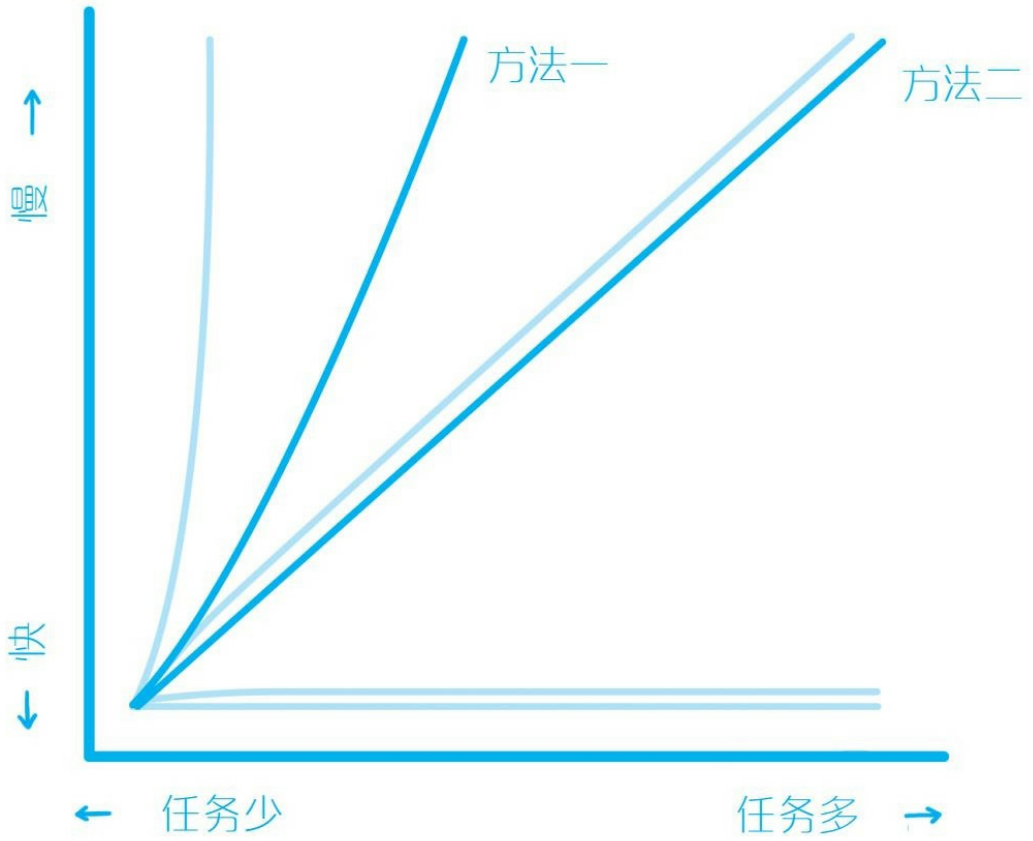
在继续往下读之前，我建议大家拿出笔、纸和道具或者你觉得舒服的东西来重建一个类

似的场景，设想在每一个步骤和假设上，如何来实现目标。试着根据以下场景进行操作。

如果说这一堆袜子大概有4双，那么玛吉用什么样的方法来做配对都没有太大影响，因为这项工作很快就能完成。但是，如果玛吉面前有成百上千双袜子呢？如果她选择第一种方法，那么她碰到同一只袜子的概率会相当高，因为她没有将这只袜子拣出来。当玛吉第一次碰到这只袜子的时候，她没有从这只袜子上收集任何信息。然而，如果采用第二种方法，她就能摆出一排没有配对的单只袜子，因此就保证了同一只袜子只会遇到一次。所以不难理解第二种方法自然会更快一些，因为它依赖于记忆。更确切地说，这是我们所说的查找表格或者缓存带来的效果。虽然可能不需要，但是这种思维方式很有效，我们可以将查找表格作为唯一标识符（键）的集合，每一个标识符指向相关数据项（值），然后我们就可以查询键值

了，我们称此为“键值对”。在这个例子中，键可以设为“颜色”。当玛吉遇到一只红色的袜子时，就要在拣出来的单只袜子中寻找“红色”。如果她找到了“红色”的区域，那么就应该寻找其他的标识符，例如样式或色调。否则，她就需要另外创建一个新的“红色”区域，将这只红袜子放在其中。

对比一下这两种方法，[4]我们可以注意到，袜子数量越多，方法一就越慢（见下图）。在本书中，我们还将介绍很多种方法来处理类似的任务。这一论题的目的是强调两种方法在渐进增长率上的显著不同，其他方法的斜率可能介于两者之间。类似这样的场景，比如玛吉可能会选择通过鸽巢原理（pigeonhole principle）完成匹配，这就需要每次从这一堆袜子中拣出6只来进行配对。



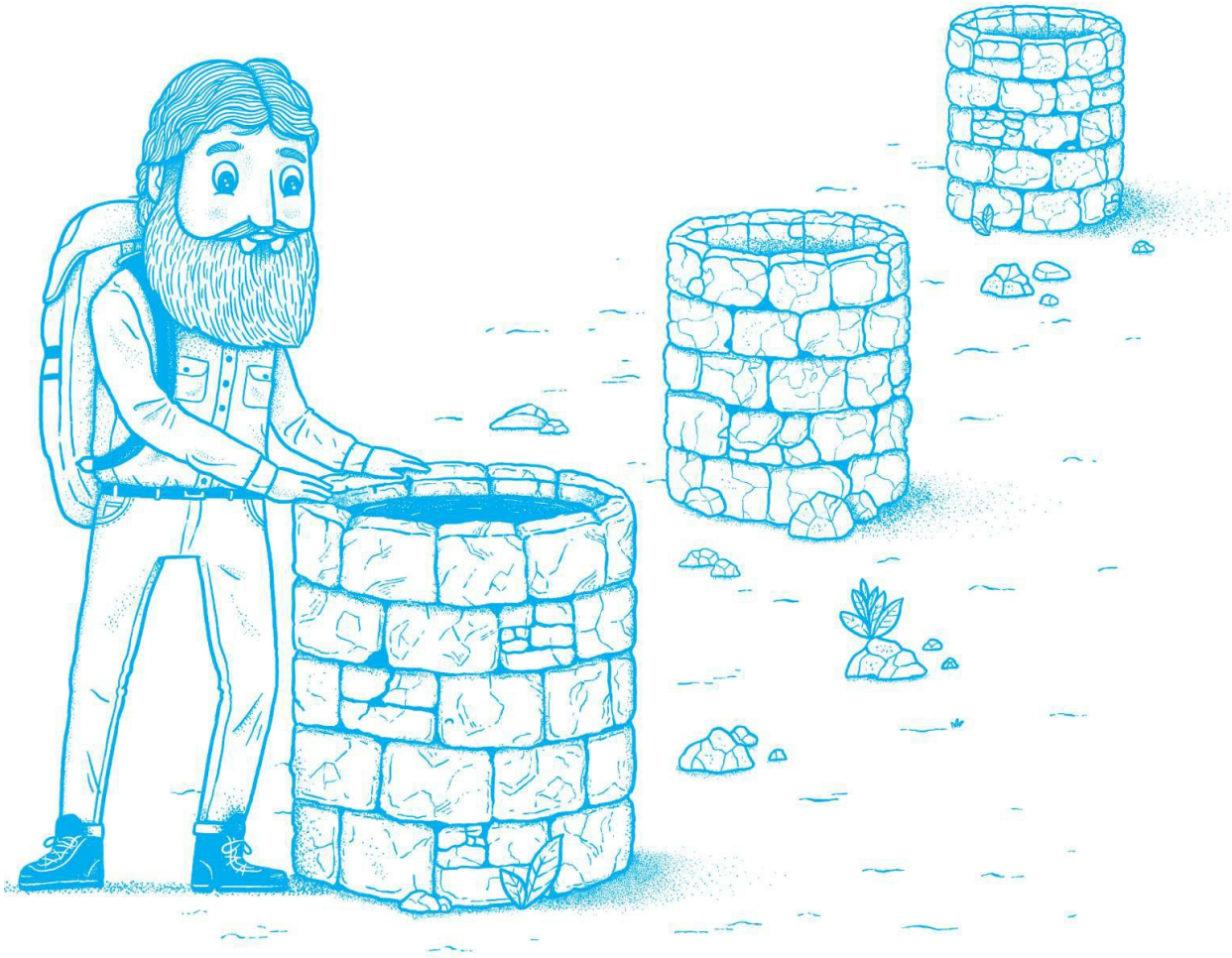
现在，当我们从这一堆衣服中选出一只袜子来，如果我们能找到跟它配对的另外一只，那么我们可以很快地分辨出来。大部分人的短期记忆都很好，可以记住大概半打数量的事物。因此，刚刚被我们放置在一旁的袜子，我们只要看到类似的就能够立刻产生记忆，“啊，我见过这个！”如果你曾经玩过像《记忆》这样的配对游戏，那么你就应该熟悉这种短期记忆

能力的局限性。

如果袜子的样式和颜色差别很大，那么我们那组未配对的袜子队列可能就会很长，我们每次选出一只新袜子时，就得将这个队列扫视一遍。当数量庞大时，扫描一行东西、一个数组，花费的时间就会很长，因为你要找的东西很有可能在这一个队列的最后，所以我们不得不把这个队列所有的组成部分都过一遍。

1953年，数学家汉斯·彼得·路恩（Hans Peter Luhn）还在IBM（国际商业机器公司）任职时构思了一种算法，通过一种替代结构来解决数组查找的耗时问题。这种结构也被称为“关联数组”、“字典”或“哈希表”（这简直就是给可怜的玛吉伤口上撒盐）。哈希表能够精确定位数据元素，它通过把关键码值映射到表中的一个位置来访问记录，而这组数组就会被全部记录下来并储存在合集中。然而，由于哈希表的

存取是随机的，并不会记录顺序（即，先对小码的红色袜子进行配对，然后是大码的黑色袜子），这种查找模式被称为常数时间查找（constant-time lookups）。[5]



之所以被称为常数时间，是因为查找不再取决于数组的长度，相反，其速度与数组的长

度无关。这一洞察是真实存在于现实中的，虽然这并不是常态。在很多软件中情况都是如此，这让很多研究人员和从业者倍感苦恼。软件不像自然界那般有自己的规律。在这个案例中，我们假设袜子都不一样，而且数量较少，玛吉的大脑皮层会在瞬间激活，并立即做出反应。

正如我们将在后文看到的那样，当我们能对任务建模时，大多数情况都会用到常数时间查找，这样可以避免逐步去完成任务和重复地计算手上的所有数据。[6]这个公式的作用是在一堆物品中快速地检索出你需要的那一个。

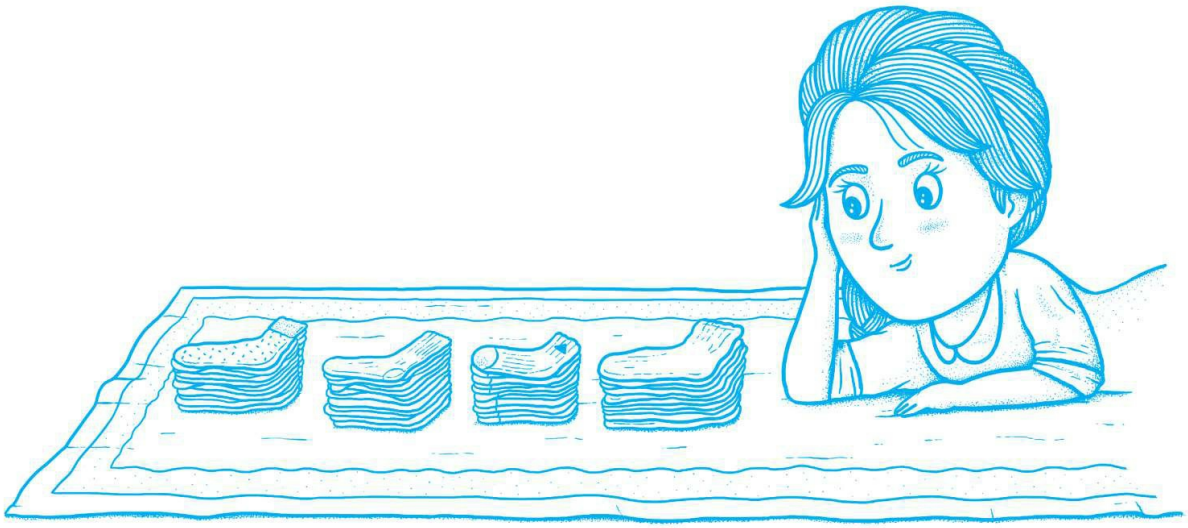
然而，刚刚提到的这些都只是一方面。我们从这个应用场景可以得出的结论是：重复使用知识来解决问题要比只使用一次更快捷。更重要的是，当摆在我们面前的任务需要一遍一遍地重复作业时，这种方法更有效——比如你

在一家商店里，为女儿寻找生日蛋糕的字母蜡烛；或者在洗衣服前，把白色的衣服和其他颜色的衣服分开；再或者你希望赢得一场拼字游戏，类似英国电视节目《倒计时》里的那样，选手要从9个字母中选择合适的字母，在30秒内组成单词，包含字母数量最多的获胜。

遇到上述这些场景时，你要问一问自己，解决手头的任务，是不是可以通过记忆来寻找更快捷的解决方案——无论这个记忆是你自己的还是世界的。在袜子配对的场景中，我们发现，将未能配对的袜子排成一列放在一边时，最多只能放5种颜色。而在选择蜡烛的故事中，我们会在第一时间选出4个我们需要的字母，而不是先去寻找L，然后再去寻找U。

在给脏衣服分类时，我们会把衣服放在三个不同的篮子里，而不会在洗衣服之前才开始翻找。而在拼写最长单词的情境中，我们可能

也会先组合熟悉的单词，然后再考虑能否进行延展处理，比如变形或复数。在这种情况下，我们最初选择的单词会作为前缀。有一种实用的树形结构，名为前缀树（或字典树），采用的就是同一个道理。这种树形结构利用的原理是单词或数字共享前缀。这一逻辑可以用于拼写检查和单词自动填充，能够为用户带来更快捷的搜索和输入体验。



很有趣吧，只要稍微换一下思维模式，很平常的事情也会变得趣味横生。

---

[1] 日常生活中，人们常用一个短语来描述这个事实，即“用记忆交换时间”。

[2] 这种方法是十多年前在多伦多大学开创的，被称为“深度学习”。

[3] 请注意，这两种方法的前提，我们都忽略了将袜子和其他衣服分开的任务，因为我们重点关注袜子配对的基本任务。

[4] 研究这些增长率，还有很多更具体同时有着细微差别的方法。其中一种是符号O，是用另一个（通常更简单的）函数来描述一个函数数量级的渐近上界，表示函数在增长到一定程度时总小于一个特定函数的常数倍。或者跟符号O很相似的符号Ω，表示函数在增长到一定程度时总大于一个特定函数的常数倍，用来描述一个函数数量级的渐近下界。另外一种则是考虑增长率是否描述了最优、最差或平均情况。在后文我们将进一步讨论这些不同的情况。

[5] 在这个例子中，玛吉并不关心那些没有配对的袜子是如何排序的，她关心的是袜子有没有配对。因此排序在这个案例中只居于附属地位，并不是最重要的。

[6] 举个例子，对于求前n个数字的和这个命题来看，如果你一个一个地加总每一个数字自然会比较慢，而如果你用公式nx（n+1）/2就会快很多。

第2课 查找 男人和女人逛商店的逻辑不一样

圣诞节后的第二天，来自苏格兰因弗内斯的护士艾培·托马（Eppy Toam）就开始在百货公司转悠，为当年的节礼日（Boxing Day）做准备。她的身材相对大众，所以她要确保自己第一个进入商店，这样她才能有挑选到自己心仪的衬衣的机会。她要快点快点再快点。可能很多人都有这样的想法，所以事情就变得不那么美好了。去年，在抢购中有15人受伤，警察最后不得不介入。在这种情况下，怎么能增加艾培抢先于别人找到适合自己的衣服的机会呢？

提示：我们来做一个极限假设。如果货架跟商店一样大呢？

如果我们正在从一个商品品类集合中选择一件商品，那么，在我们寻找这件商品之前，

需要把所有的商品都查看一遍吗？换言之，如果有100件商品，那么我们需要把这100件商品都挑一遍吗？也就是说，要采用线性时间的原则吗？通常来看，线性函数意味着如果我们花一分钟从100件商品中找到一件，那么就意味着我们要花两分钟时间从200件商品中找到一件。理论上来看应该是这样的。然而，由于这些商品的集合都有相对合理的排序，那么我们就可以在对数时间内找到适合自己的那一件。换句话说，可以从7件商品中找到自己需要的那一件，而总量不再是100。回忆一下，对数函数是指数函数的反函数。在编写计算机程序时，我们假设以2为底，100的对数为$\log_2 100$，约等于7。从线性时间到对数时间，你可以看到这种巨大的效率提升，这也就是为什么对数是一个如此重要的概念，特别在涉及增长率的问题时。对于这一概念，我们在本节课中将会进行一次更为全面的解析。

让我们来描述一下艾培将会如何伴随人群拥进商店吧。她的脸上似乎闪着荣耀的光辉，格子呢的披肩搭在身后，仿佛置身一场恶战。她紧紧地咬着双唇，身体靠着商店的墙，以免被后面排队的人挤倒在地。一大早她就开始精神紧张，不断地给自己加油。

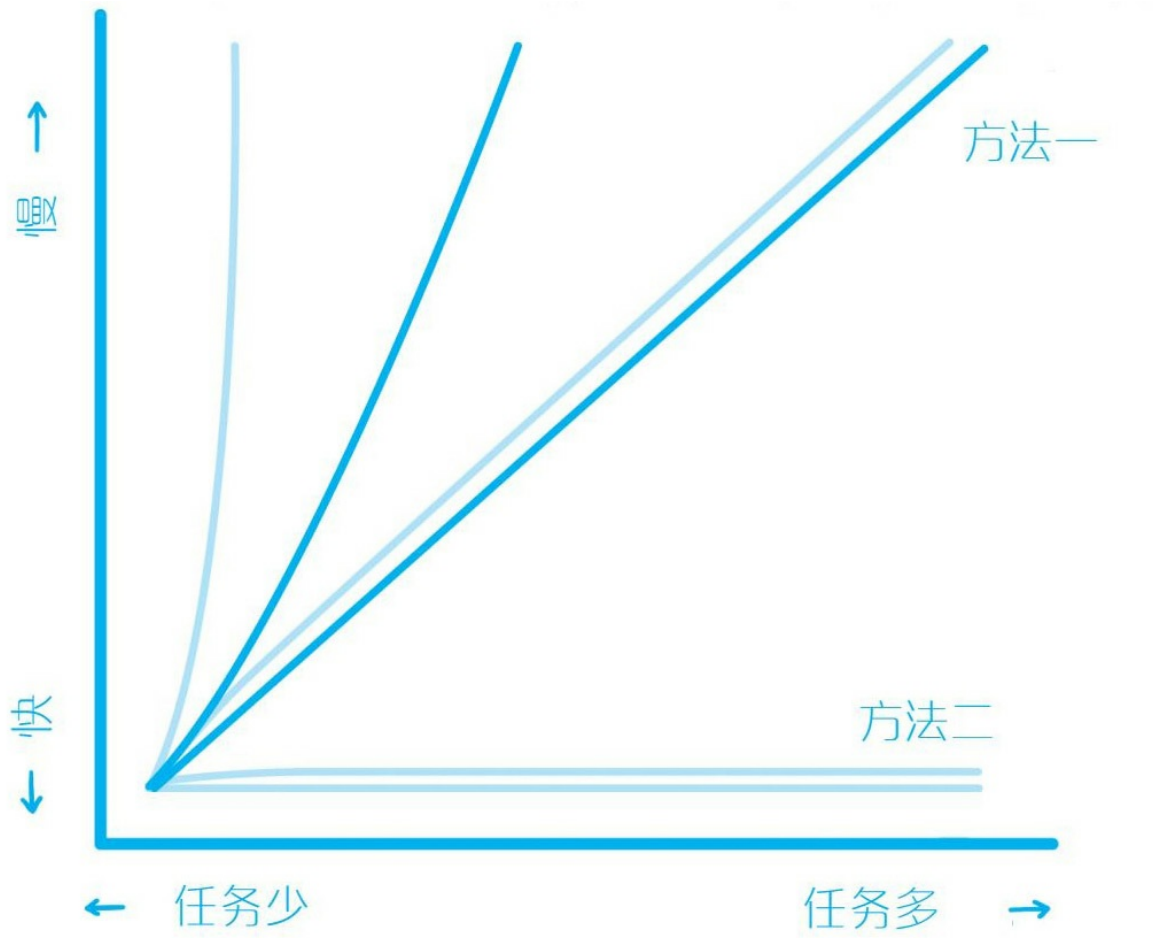
目标：在特定的货架上，找到适合艾培尺寸的衬衫。

方法一：在特定的货架上，从头到尾一件一件挑选。

方法二：在特定的货架上，从货架中部的尺寸开始寻找。如果尺寸过大，就到货架的左边寻找；如果尺寸过小，就到货架右边寻找。以此类推。

下图为这两种方式的耗时对比。可以看到，随着货架上衬衫数量的增加，方法一明显

慢于方法二。

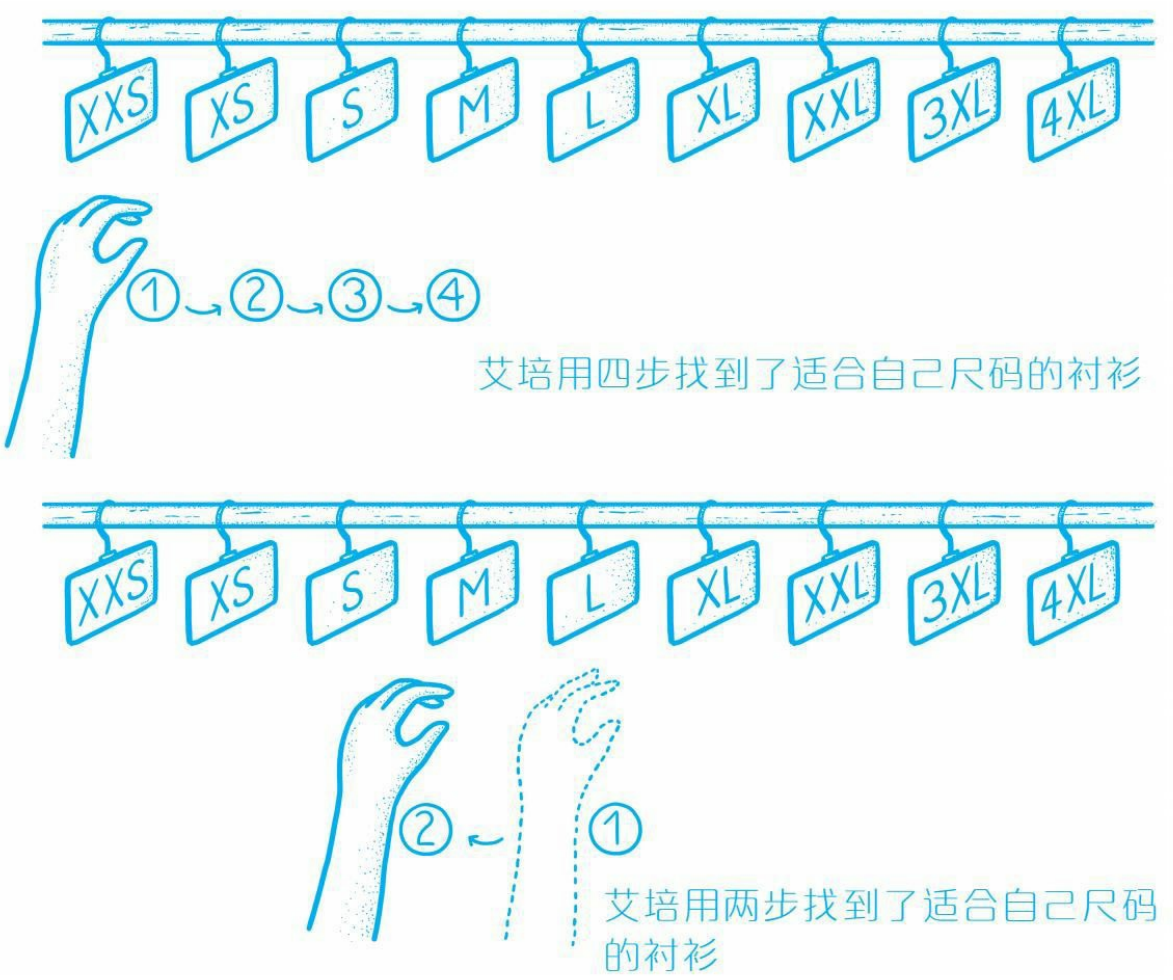


聪明如你，也许已经猜到了，方法二用到了两种常识。一是衬衫在货架的摆放极有可能是按照尺寸大小排列的；二是艾培的尺寸是比较大众化的，也就是说属于平均尺码，那么就极有可能会最接近中间。通过这两种常识，艾培可以选择从中间位置开始挑选，也容易让她

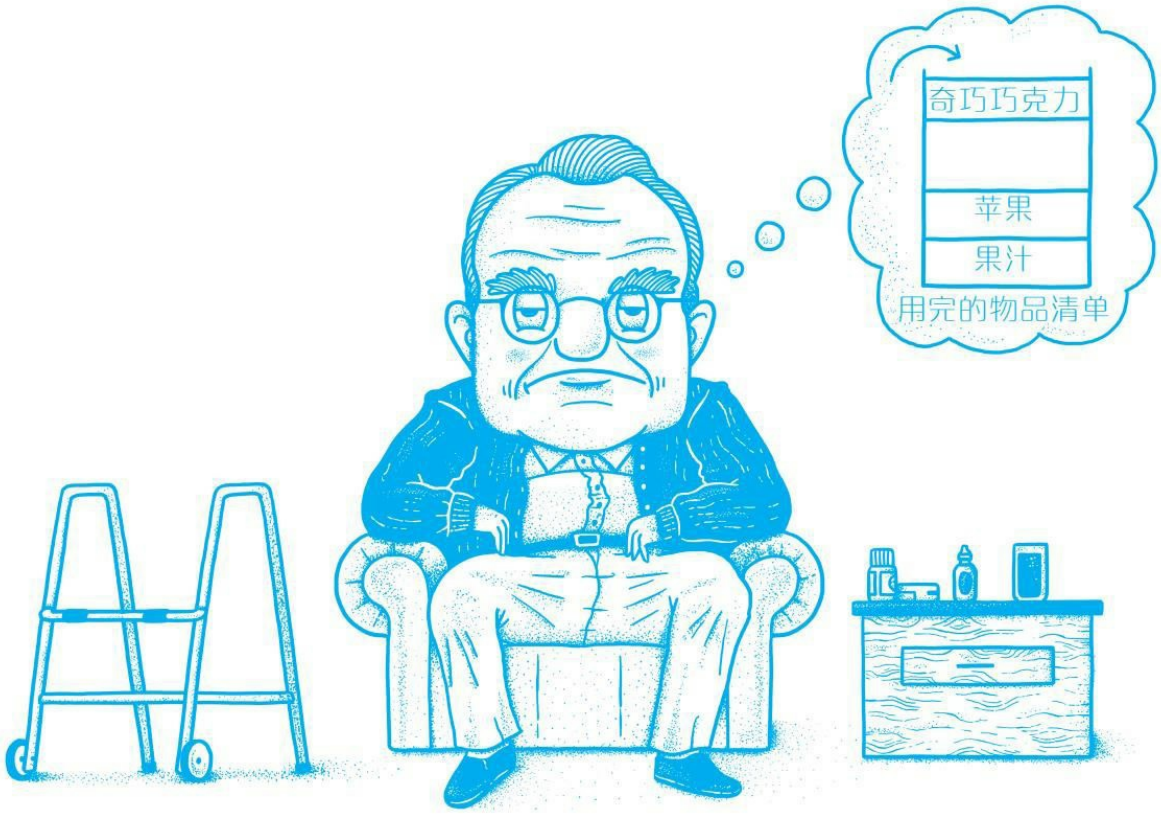
知道应该向左还是向右做进一步的寻找，从而一下子缩减了二次选择时间，这就是一种典型的对数时间算法（logarith mictime algorithm）。[1]同样的方法，我们也会在查字典或者从通讯录中查找电话的过程中使用。此外，如果我们看了一本乏味的小说，看着看着睡着了，那么下一次就会从上一次看到的地方开始读。通常来说，我们会把这种方法视为信息过滤。

在我们的日常生活中，对数之于我们关联性最高的因素就是增长速度慢，正如下图所示。我们通常更喜欢解决那些循序渐进的问题，因为这意味着解决问题的方法不会因为事物数量的增长而发生大的变化。在这个情境下，艾培可能会在100件衬衫中仅用7步就能找到适合自己的那一件，而假设有1 000衬衫的时候，她也许只需要10个步骤。这种在排序集合中对数搜索某物的方法被称为二分查找

（binarysearch）。这种方法相比于方法一（线性查找）具有巨大的优势，利用这种方法艾培自然会快速找到满意的那件衬衫。




指数
线性
对数



---

[1] 类似地，求数字从1到n翻了多少倍的过程也是对数运算，在计算从1到n的时候最多需要log n次步骤。比如，如果我们以1美元作为基数，每年翻倍，要花多少年才能赚到100万美元？这个计算过程可以手动完成，或者可以利用公式：log2 1 000 000=19.93年。

第3课 排序 用这个方法，我玩拼图就没输过

伊恩·帕托伊思（Iain Patoys）是一位退休的语言学教师，家住东伦敦。他在几年前摔伤了背部。由于他很怕邻居家的狗，所以他讨厌外出，但出于生活需要，他不得不去商店采购生活物品。况且这里是伦敦，下雨很频繁，而伊恩从小就很讨厌下雨。鉴于此，在一周内，伊恩应该如何既能最大限度地减少外出去商店的次数，又能满足生活所需呢？

在英国双人表演史上有一部不朽的艺术之作，名为《两个罗尼》（*The Two Ronnies*），[1]其中有一个场景是这样的：一位顾客走进了一家五金店，然后对着自己的购物清单一个一个读了出来。店主没有等顾客把清单全部念完再去找，而是顾客每读出一件商品的名字之后，就去找一次商品，因为清单上的

物品太多，最终导致店主非常崩溃。

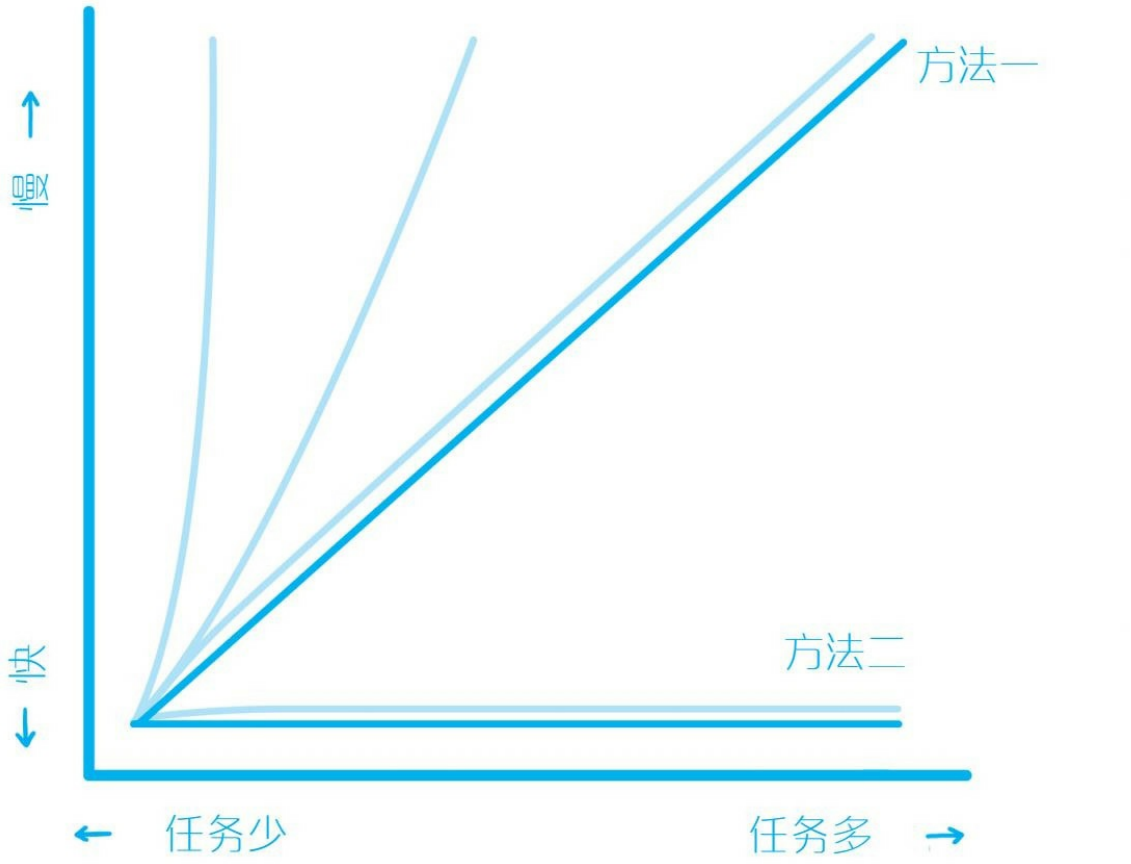
先把这个小插曲放到一边，我们还是回来看看伊恩的问题吧。

目标：在一周之内尽量减少去商店的次数。

方法一：发现某物用完之后，直接去商店购买。

方法二：准备一份清单，将用完的物品列在上面，一旦清单上的物品达到了一定的数量，或者用以维持生命的必需品用光的时候，就去商店采购。

下图想必大家都不会陌生了，在这种直观的图形中，我们可以看出两种方法的对比。



关于该场景的一种解释就是如何避免重复劳动，这很像为何一位秘书在装订文件时，是用打孔机一次性装订10份报告而不是分开一份一份地去打孔装订。同理，你也会把很多盘子一起放进洗碗机清洗，而不是一个一个清洗烘干。还有一个例子，高层电梯会有一个目的地调控系统，让到同一层的乘客乘坐同一部电梯。还有一个更微妙的观察，这与伊恩去商店

的原因有关。我们来探讨一下。

在计算中，有很多种存储项目集合的方式。我们之前在玛吉袜子配对的数组问题中看到了最基本的一种模式。接着，我们在第二个场景中看到了数组如何通过内容分类来最大化某种特殊的品质，即可搜索性。这就是艾培寻找合适尺码衬衫的故事。这些问题从本质上来看，都可以称为数据结构（data structures），有时也被称为抽象数据类型（abstract data types）。在这样的结构中，它们可以最大化某一种或者多种我们所关注的特性，而忽略其他方面。通俗地讲，我们可能会认为某些特性是相互对立的。例如，安全性和易用性——当我们启动一个应用程序时，如果每点击一次按钮都需要输入一次密码，这可能会更加安全，但是在易用性上却大打折扣。

我认为，与我们探讨的问题相关的结构应

该是栈（stack）。正如这个名字所包含的寓意那样，栈会最大化我们最关注的某一特性，将其置于最顶端，而忽略其他的特性。如果你走进一家咖啡馆，看到一堆报纸，你只会瞥一眼最上面的那一张，因为你知道这是今天的报纸，而你只关心今天的头条新闻。同理，在栈中，我们关注的是顶部项目。[2]

在伊恩的这个情境中，他的认知栈包含了他消耗光的物品，他只关心自己要去商店买什么能填补自己的空缺，也就是说，重复删除栈上的第一个项目，直到该栈为空。对于伊恩来说就是，当他把奇巧巧克力推到购物清单的第一位时，栈就可以认为空了。这时，他就可以开开心心地把购物清单上的东西忘掉去忙自己的事情了。那么我们再回到《两个罗尼》的故事中来，如果五金店的店主也建立一个认知栈，或者每排货架都建立一个认知栈，那么他就不需要一次次爬上爬下地去找商品，从而大

大减少他的工作量。等顾客读完他的购物清单，店主就可以建立起自己的栈，然后走到摆放相应物品的货架处，找到该栈最顶端的物品。

1946年，艾伦·图灵（Alan Turing）发表了一篇报告，介绍“栈”这一概念时，使用了“埋”（burying）这个词。正如安德鲁·霍奇斯（Andrew Hodges）在《艾伦·图灵传》中所写的，这一想法启发了冯·诺依曼。下文是图灵报告中的一段摘录：

> 当我们需要调用子程序时，我们只需要标记一下我们离开主过程的位置，然后切到子表的第一条指令。当子过程完成时，我们找到那个标记，返回主过程。每一个子过程，都用“RET”这个指令作为终止。那么如何标记呢？办法当然有很多。其中一个办法是，将这些标记的队列储存

> 在一组标准号的延迟线上（1024）……包括当前最近的标记。而最近的标记，会同时保持在一条特定的TS（临时存储）中，每次调用或终止子过程时，都会修改这个标记。

那些如今我们看起来直观易懂的概念，产生的过程是何其复杂！在读到这样的故事时，实在是让人倍感谦卑。这让我意识到，这些概念是经过先驱的不断完善才得以成型的。有趣的是，我们也会听到类似弗林效应（Flynn effect）这样的概念，它表明人类一直在变得更加智慧，其中一部分原因就是我们的直觉感官正在变得越来越成熟和复杂。如今，新一代人相比先辈而言，天生就有更加敏锐的直觉。在阅读旧书本和论文时，有时会觉得很有趣，仿佛这些文字正在诉说我们一路走来的历程。我记得有一次，当我拿起1530年出版的德西德里乌斯·伊拉斯谟（Desiderius Erasmus）写的《论

儿童的教养》（*Handbook on Good Manners for Children*）时，看到书中提到的几条建议，例如“不要让你的鼻孔里充满了鼻涕，搞得脏兮兮的”，“苏格拉底就因为太不修边幅而遭人诟病”。对于21世纪的读者来说，这样的建议似乎无须多言，大家都习以为常了，但是在那个时代，这就是一种理想的状态。

图灵关于子过程的讨论让我想起了一个现实生活中的例子，它可以证明栈在日常生活中的有效性。想象一下，邮递员在某天早晨出现在伊恩家门前，但两人并没有用眼神交流。这时候，一抹孤独的泪水从邮递员的脸颊流下，他嘴唇微微颤抖。

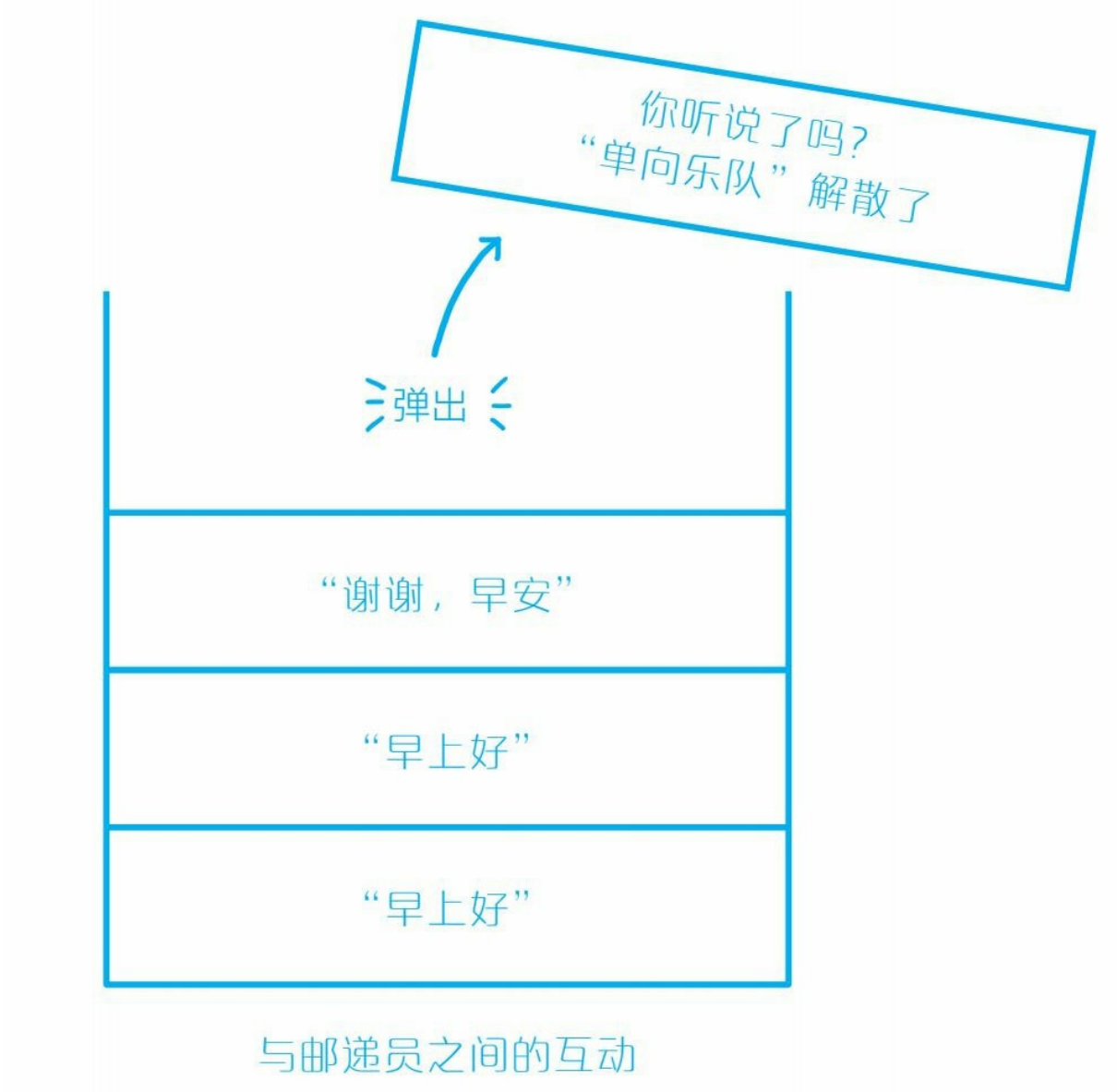
“不好意思，请问一下，是我让你难过了吗？”

“嗯，是的，确实是这样的。”邮递员一边

说着，一边把目光转向地平线。

这时，伊恩会尝试回忆他到底对邮递员做了什么才让他如此伤心，就像从栈中挑出一件物品一样。我们姑且把这个称为“他的邮递员栈”（见下图）。这种比喻相对比较恰当，因为他最近一次与邮递员之间的互动相比之前的一次互动更有可能是引起邮递员伤心的主因，而他之前与邮递员的互动比再之前的互动更有可能引起邮递员的伤心，以此类推。

那么在日常生活中，还有什么是依据栈的原理来运转的呢？或许互联网就是一个恰当的例子。每当你点击一个链接时，你就会将这个网站推到一个栈中，而每次回到上一个链接时，就会从该栈中弹出一个网站。也许你根本不关心你访问过的网站数量，只要你能从当前页面跳转回前一个页面就可以了。



与邮递员之间的互动

在这里，只是希望伊恩能够从他的认知栈里找到引起邮递员伤心的原因，并加以修缮，然后也能合理地运用这一原理决定何时前往商店购物。

---

[1] 读者可以通过bookofbadchoices.com/links/ronnies观看视频内容。文中提到的相关故事情节在该视频的中间位置。

[2] 这些是栈操作的实际名称。正如你想象的那样，这些操作都需要很长一段时间。

第4课 关联性 让路痴不再迷路的方法

伊奥尼斯（Ioannis）在自己的服装厂里迷路了。在雅典，他的服装厂以一种不可思议的速度成长，所以他也被人称为囤积者。他的服装厂占地庞大，但其中大部分是闲置的。在过去30年里，伊奥尼斯为了增加一些通道和货架而不断扩建自己的服装厂，这就让他的“囤积”习惯愈演愈烈。如今，他站在自己的服装厂里，仿佛置身于混乱的迷宫中，迷宫的墙上挂满了线卷、衣服，墙边堆满了破旧的缝纫机。他到底怎样才能走出去呢？或者说，他注定要在自己创造的这个迷宫中灭亡吗？

在希腊神话中，有一则故事讲的是，当人身牛头怪弥诺陶洛斯（Minotaur）降生的时候，伟大的建筑师代达罗斯（Daedalus）在克里特岛为它修建了一个迷宫以困住这只凶猛的

怪物。

“由于弥诺斯的儿子安德洛革俄斯在阿提喀被人阴谋杀害，弥诺斯起兵为儿子报仇，给那里的居民带来很大的灾难。为了平息弥诺斯的愤恨，解除雅典的灾难，雅典人向弥诺斯求和，答应每九年送七对童男童女到克里特作为贡品。弥诺斯接到童男童女后，将他们关进半人半牛怪弥诺陶洛斯居住的克里特迷宫里，由弥诺陶洛斯把他们杀死。进了这个迷宫后，那些年轻的雅典人就会沿着迷宫蜿蜒崎岖的道路寻找出口，但是从未有人成功过。”

忒修斯（Theseus）很幸运，他被送去给弥诺斯进贡时，国王的女儿阿里阿德涅（Ariadne）已与他坠入爱河，并为他设计了一个逃跑计划。

“阿里阿德涅要求代达罗斯告诉她走出迷宫

的线路，并教给了忒修斯，但是阿里阿德涅要忒修斯必须答应带她回雅典并与她完婚。于是忒修斯按照女朋友的建议，手持线团走入迷宫。他溜到正在喝开胃酒的人身牛头怪身后，把它钉在地上，徒手消灭了弥诺陶洛斯。”[1]

我们暂时把这个小插曲放在一边，先来看看让伊奥尼斯从服装厂走出来的三个方法吧。

目标：回到店面。

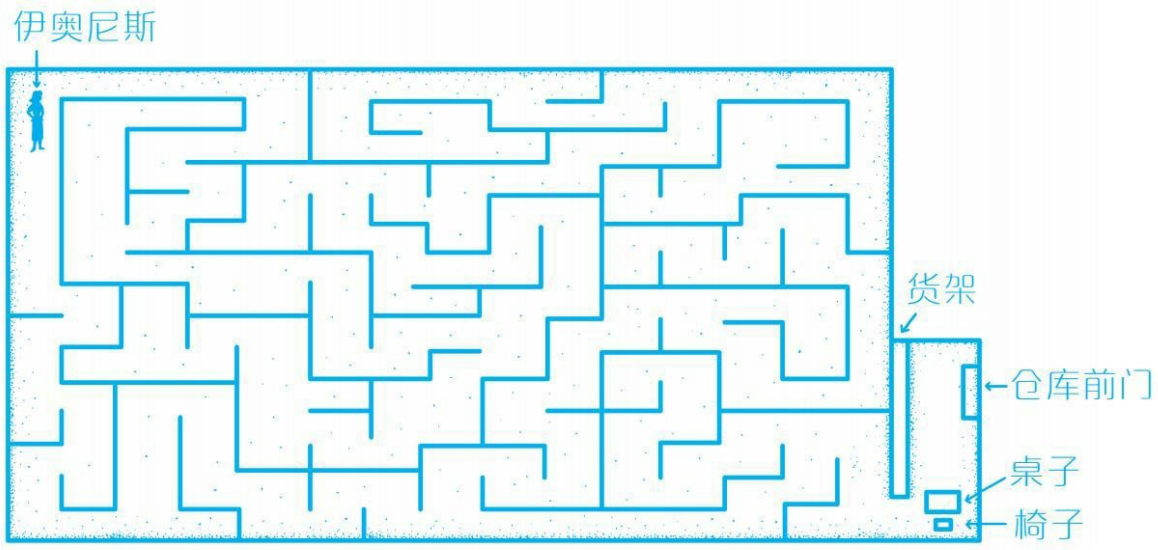
方法一：沿着小路走，遇到岔路随机选择，直到找到正确的方向。

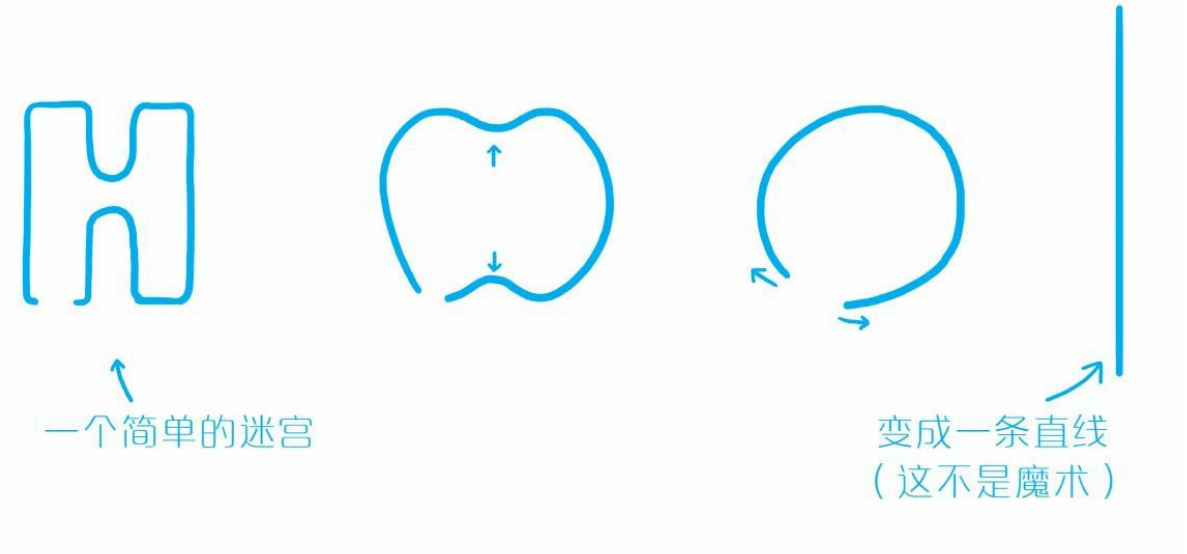
方法二：一直沿着墙的右侧走，遇到岔路口向右转。

方法三：从货架上取下一卷线，一边走一边放线，如果遇到死胡同或者遇到自己放过线的地方，就转身向相反的方向前进。

方法一其实就是模拟老鼠走迷宫的状态。老鼠没有高级的认知能力，只是从一个地方随机走到另一个地方，直到偶然找到一块奶酪。事实上，这种方法也被称为随机鼠（random mouse）。不难理解，这种方法全凭运气，可能会花费很长时间才能找到出口。

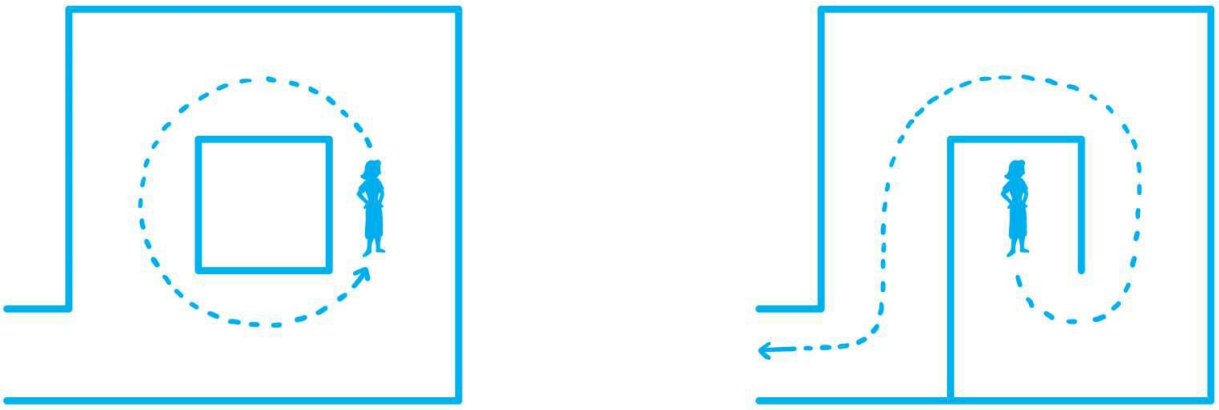
方法二相对有趣一些，尽管依然很简单。在这里，伊奥尼斯一直沿着墙的右侧走，最终找到前往店面的道路（见下图）。为什么说这种方式是可行的呢？因为如果你将这个迷宫的墙壁重新排列，可能会得到一条直线。这种情况下，你可以把这个迷宫理解成一条绳子，只要沿着一条绳子走，你必然会从一段走到另一端。[2]





虽然这种方法在速度上比第一种快，但核心的问题是，如果这个迷宫碰巧有环岛或者环路，那么就完全行不通了。因为那样的话，这些迷宫里的墙互不相连，无论是直接还是间接

都不会与外墙相连。19世纪20年代，数学家、第四代斯坦霍普伯爵（Earl Stanhope）在肯特的切文宁建造的第一座园林迷宫，就有类似的环岛。这是基于第二代斯坦霍普伯爵的设计进行改良创建的，其目的是创建一个无法用第二种方法来解决问题的迷宫（见左下方图）。



因此，在这样一个迷宫中，被称为墙跟随（wall follower）或者右手法则[3]的特殊的逃跑方法就无法奏效。

至此，就可以进入下一个命题了。有一则关于查尔斯·达尔文的趣事，在他发表《物种起源》之前，用了近20年的时间寻找和整理所有

可能反驳进化论的观点。我一直认为完成这样一个规模的理论构建，就如同穿越一个迷宫一样，有无数的分岔路口需要选择与尝试。[4]想证明一个论点是无效的，就仿佛走进了一个死胡同。而正如走出迷宫一样，得到一个有效的结论也需要缜密的逻辑和预判。事实证明，这种对结论的认证和反驳的认知方法也同样适用于走出迷宫。我们提到的方法三和阿里阿德涅的故事正是如此。

通过这种方式，伊奥尼斯需要做的就是充分利用回溯法（backtracking）。他用线作为记录标识，每次遇到死胡同时，就要尝试回溯其他的不同路径。[5]能够回到岔路口并尝试其他选择，保证了伊奥尼斯最终能找到出口。这种逃离迷宫的策略被称为绳索探索算法（Trémaux's algorithm），这一结论被认为源自法国数学家爱德华·卢卡斯（Edouard Lucas）于1882年出版的《重塑数学》（*Récréations*

Mathématiques）一书。近期也有研究表明，其他生物（如蚂蚁）也可能会使用一种类似回溯法的方式找路。这种方法比方法一更快，但不同于方法二，而且可以解决环路的问题。

请注意，如果你身陷迷宫，我们在这里提到的三种方法都是有效的。当然，还有很多可以更快走出迷宫的方法，只是这些方法可能需要我们提前知道迷宫是什么样子才能行得通。我不能保证这里提到的办法是走出迷宫的最短路径。

至此，当你了解了这些走出迷宫的方法之后，你就不会迷失在罗马的地下墓穴之中了，也不会成为史蒂芬·金（Stephen King）小说里迷失在迷宫里的倒霉角色了。在现实世界里，大概有数十个迷宫分布在各地，有的甚至长达数英里，我们学到的方法在这里都很有效。推而广之，在受限环境中这种从一个点到另外一

个点（例如迷宫）的方法非常重要，网络或者图示都是一种描述走出迷宫的方式，比如想象通道便是边缘，交叉点是顶点，这是我们日常使用和依赖的很多应用程序的核心。比如OpenStreetMap（世界地图应用），它告诉用户道路是如何连接的，它会告诉你如何从你住的公寓开车到海边。谷歌旗下的知识图谱（Knowledge Graph）能够了解人、地点和事物之间的关联，这样就可以给用户提供更优质的搜索结果。脸书和领英是你的朋友驻扎地，可以计算出你可能认识的人。而火狐浏览器则清楚自己的组件和模块之间的联系，从而使这个系统软件可以基于连接模式和密度之间的关系，准确预测未来可能出现的缺陷。

事实上，扫地机器人也是一个很好的例子。扫地机器人在设计上也是有差别的，因为覆盖的清扫面积不同，这些机器人的算法复杂程度就不同。简单的那些可能是在房间里随机

游走，经过的路线是直线或者圆圈，而复杂的机器人则会先为这个房间绘制一张地图，确定墙壁和角落的位置，然后从一端向另一端来回移动进行清扫，这就会形成一种网格状的线路。换言之，当机器人明白如何更好地覆盖房间的所有空地，从一端移动到另一端时，它就能够最大限度地提高效率，为主人带来一个更加洁净的房间。[6]

在伊奥尼斯的这个故事里，无论他用什么方法，最终都会走出自己的工厂。然而，如果他无限制地囤积物品，或者说他的存货无限增长，那么他就需要在裤兜里放一个线团了，不然他真的会迷失在自己的商品迷宫里了。

---

[1] 这是一则晚安故事。

[2] 下图的灵感来自杰米斯·巴克（Jamis Buck）的一幅插图。

[3] 或者，如果一个人选择沿着墙往左手边走，那就称为“左手法则”。

[4] 每一个论点都需要有一组论据作为支撑，或为真，或为假。

[5] 他可以通过用粉笔做标记或者放几块布来达到同样的目的。

[6] 2016年3月出版的《烹饪画报》（Cook's Illustrated）对这些方法进行了一次漂亮的对比，对不同的扫地机器人进行了一次长时间拍摄。

第5课 信息拆分 外卖送到家之前都经历了什么

查理·麦格纳（Charlie Magna）拿到了今天要分发的信件之后似乎略有惆怅。7月中旬的开普敦气温在45°C以上。更糟糕的是，因为自己的不小心，查理打翻了盒里分拣好的33封信件。这些信件原本是按照门牌号顺序排列好的，查理沿着自己的职责区域范围不用走回头路就可以投递完成，但这下可坏事儿了。更糟的是，查理还把帽子和防晒霜忘在了家里，而他又是光敏性皮肤。他跪在灼热的沙地上，以最快的速度分拣着邮件，尽量赶在自己的皮肤晒伤之前结束工作。

提示：考虑如何将大问题拆分成小问题后再逐一解决。

秩序可以帮助我们更快地做事。试想一下，如果一份报纸没有按照时间来排列当地的

新闻，或者说如果你正在看的电视剧没有按照顺序播出，这将会多么让人心烦啊。你在看的电视剧顺序乱了，原本这个时间可以用来看一个命运多舛的药贩子是如何被这个残酷的世界不断打击的，但是现在你却不得不费时费力搜索下一集内容，这是多么恼人的一件事情。

我们回到查理的故事，来看看他应该如何解决自己的困境。

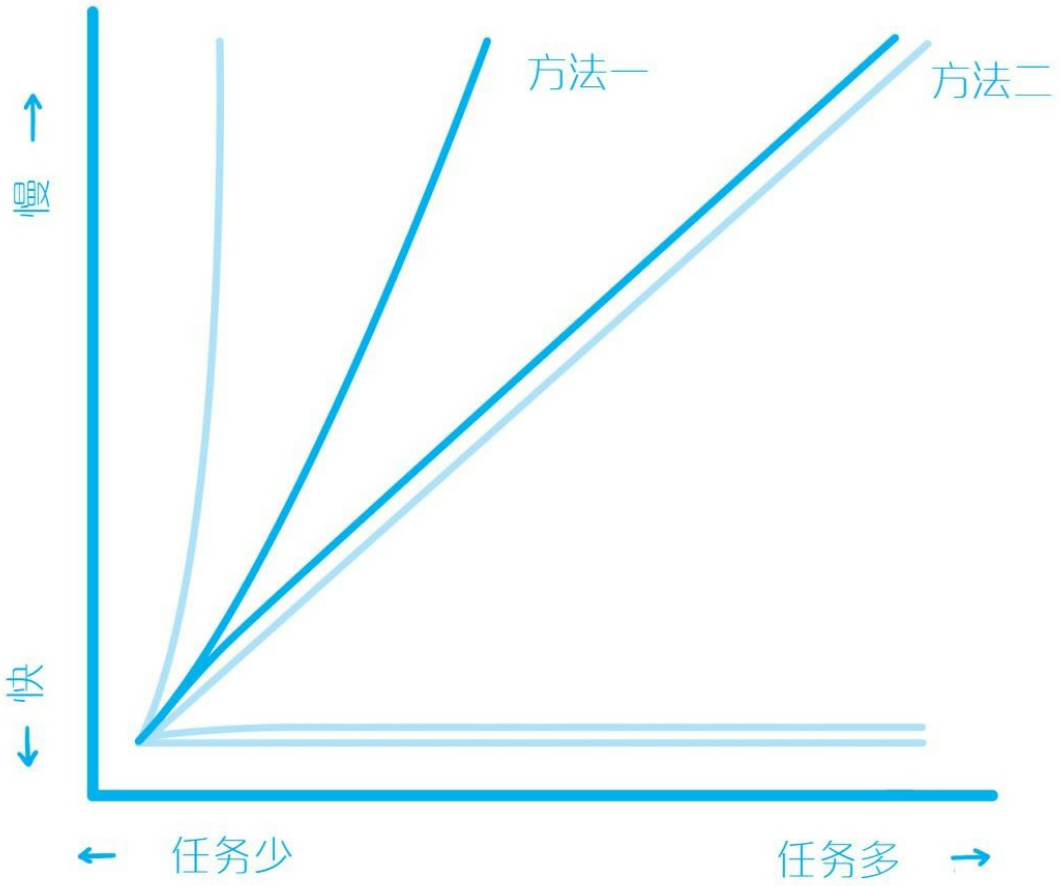
目标：将散落的信件放回正确的位置。

方法一：将一摞信封放在他面前的地上。再拿起第二摞，如果地址更近就把这一摞信封放在第一摞的左边。以此类推，直到最近的邮件在最左边，最远的邮件在最右边，完成排序。

方法二：把信封在他面前排成一排。将这些邮件分成两部分，再对这两部分进行第二次

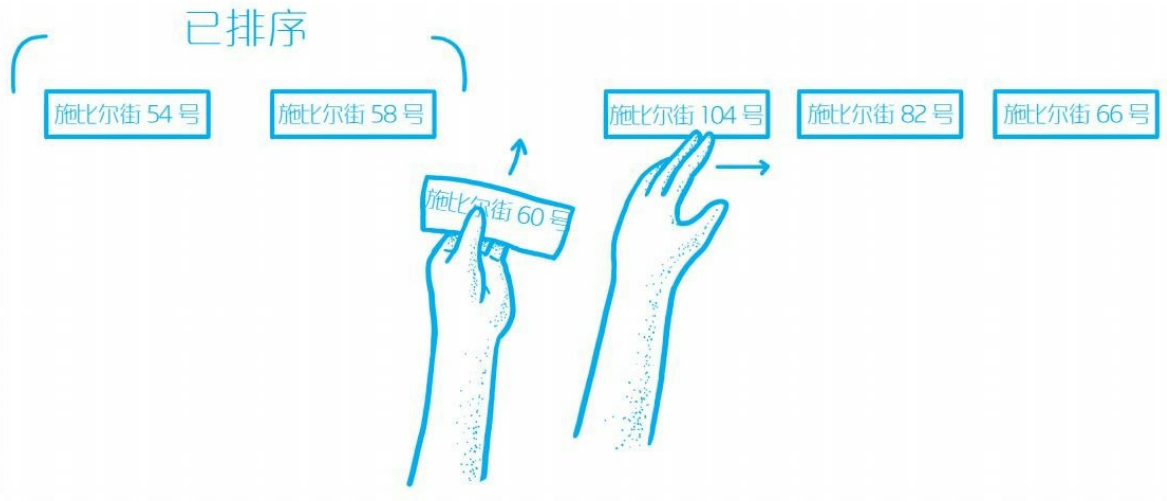
对拆。将距离最近的邮件放在左侧，远的放在右侧。

下图为这两种方法的对比图示。



在现实生活中，每当我们手动分拣一些物品时，正如查理遇见的这种情况，我们可能会常常使用方法一的一些变体。正如我们在线性

图上看到的，这种方法更具有普适性——在物品数量有限的情况下，这两种方式都适用。而随着物品数量的增加，方法二的优势就显现出来了（见下图）。虽然这种方法在我们的日常生活中并不是很常用——至少在排序方面——但是从理论上来看这种方法更加实用。[1]



需要注意的是，方法一具有一定的节奏性。查理拿出一摞信封，然后需要看其他摞才能确定这一摞应该放在哪里。然后他再拿出一摞，再将排好序的信封看一遍，重复进行。这个方法是不是有些似曾相识？对，就是那一堆

袜子配对的问题。但这两者之间有一个明显的区别，事实上，查理只需要将排好的信封看一遍，而玛吉则有可能要花很长的时间来找另外一只袜子进行配对。

解决查理这个问题的第一种方法是一个典型的平方时间（quadratic time）[2]算法。每次你需要在一个集合里搜索你的目标对象时，不论是同类型的还是不同类型，你都会发现自己需要整体扫描这个合集中的每一个对象，这其实就是你在执行一个平方时间算法。平方时间算法还有一个很典型的例子，就是从一堆衬衣里选出一件与裤子搭配，或者浏览自己的购物清单，然后在商店的货架上寻找你需要的物品。

在计算中，很多最简单的项目排序方式都以平方时间的计算方式来运行，如同查理运用的方法一，这些方式都是通过比较相邻项的大

小来进行位置切换。事实上，我们可以有把握地说，所有的排序方法都遵循这种比较相邻项的模式，平均按平方时间的方式来运行。换言之，如果说信封的数量为n，我们可以把通过对比相邻信件把它们放回正确位置的函数描述为“上限为n^2”，也就是说，平均来看，我们不会有更快的方式了。例如插入排序（insertion sort）、选择排序（selection sort）和冒泡排序（bubble sort）等。

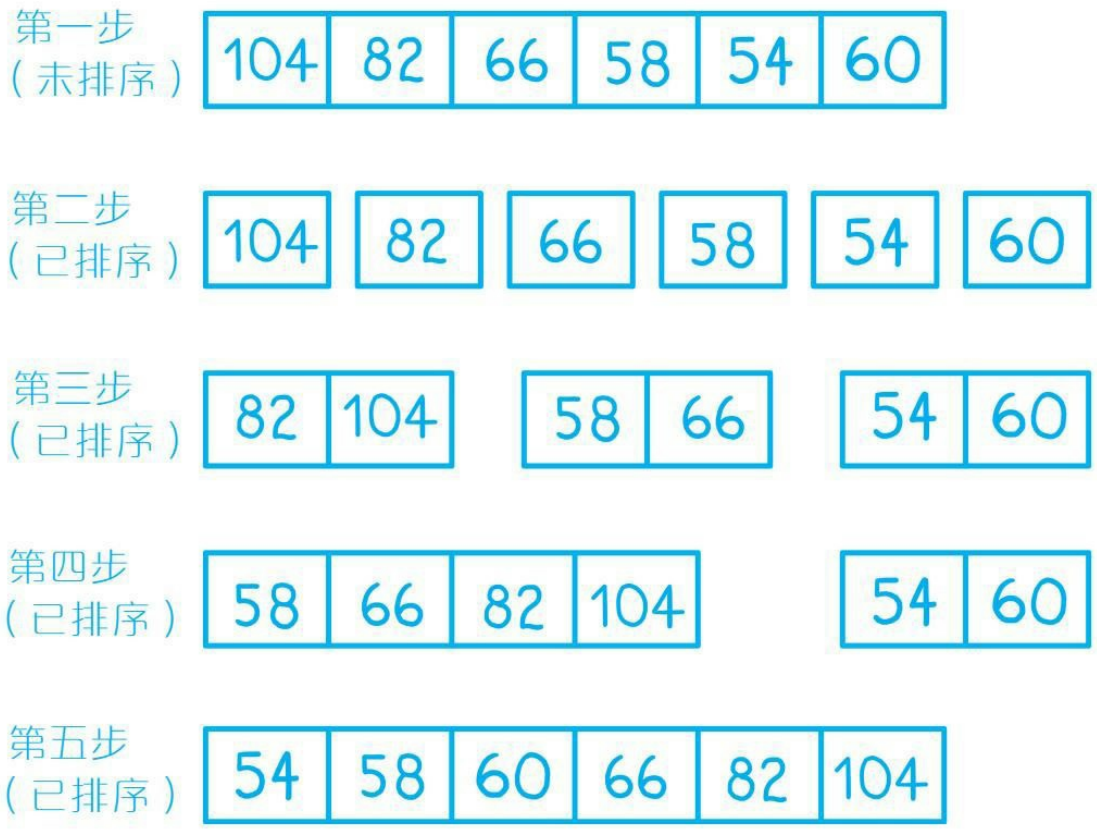
我第一次接触到排序时，还是一个16岁的学生。我记得起初我不太理解还有什么方法能比平方时间算法更有效。本节课两种方法的对比图表明方法二明显快于方法一，由此来看，似乎有一种次平方时间（subquadratic time）的算法可以更快地解决排序问题。

通常来看，次平方排序方法包括了所谓的分和治，也就是将集合拆分成较小的集合并分

别对这些子集进行排序。[3]正如我们在前面看到的那样，将集合分解成两半，这是对数法，我们只需要浏览一遍其中的元素即可。因此，这种排序的方式被普遍认为是线性对数（准线性）关系，所以你可以理解这种解决问题的方法要比平方时间快得多，但会略慢于线性时间。[4]或者说，这种方法可以被称为对数线性（log-linear）或n log n——从拆分集合（log n）到将其重新合并（n）所用的时间就是排序的时长。“线性对数”（linearithmic）这个词是由“线性”（linear）和“对数”（logarithmic）两个词组合而来，由此不难看出，这个概念在初期其实并不复杂，类似于杰德沃德（Jedward）这种合成词。

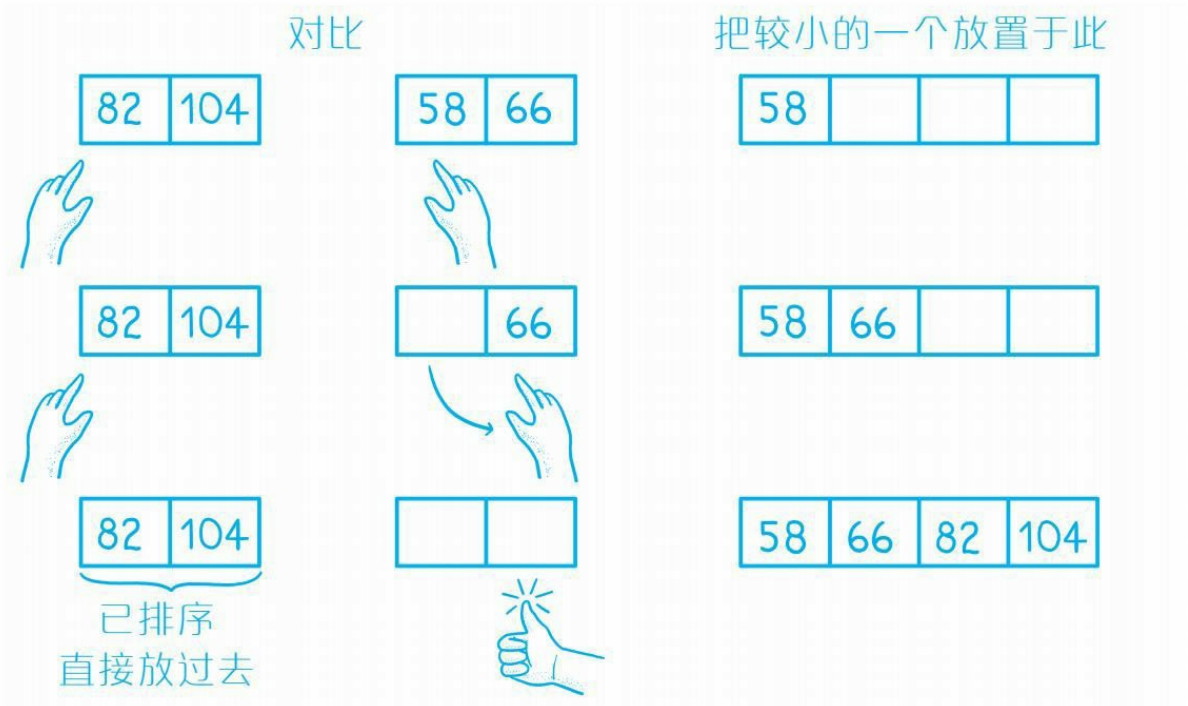
有两个众所周知的线性对数算法，一是由冯·诺依曼于1945年提出的归并排序法（mergesort），另一个则是由托尼·霍尔（Tony Hoare）在1959年提出的快速排序法

（quicksort）。查理提出的方法二类似于归并排序法的原理。将一大堆信件先拆分成小堆，然后通过比较之后再次合并。归并步骤包括了对比和组合。完成第一次合并之后，我们就可以得到一个含有两个有序元素的集合；完成第二次合并之后，集合中的有序元素就会变成四个；以此类推。关于查理的这个故事，我们可以通过下图更为直观地了解其过程：



注：图中数字为门牌号。

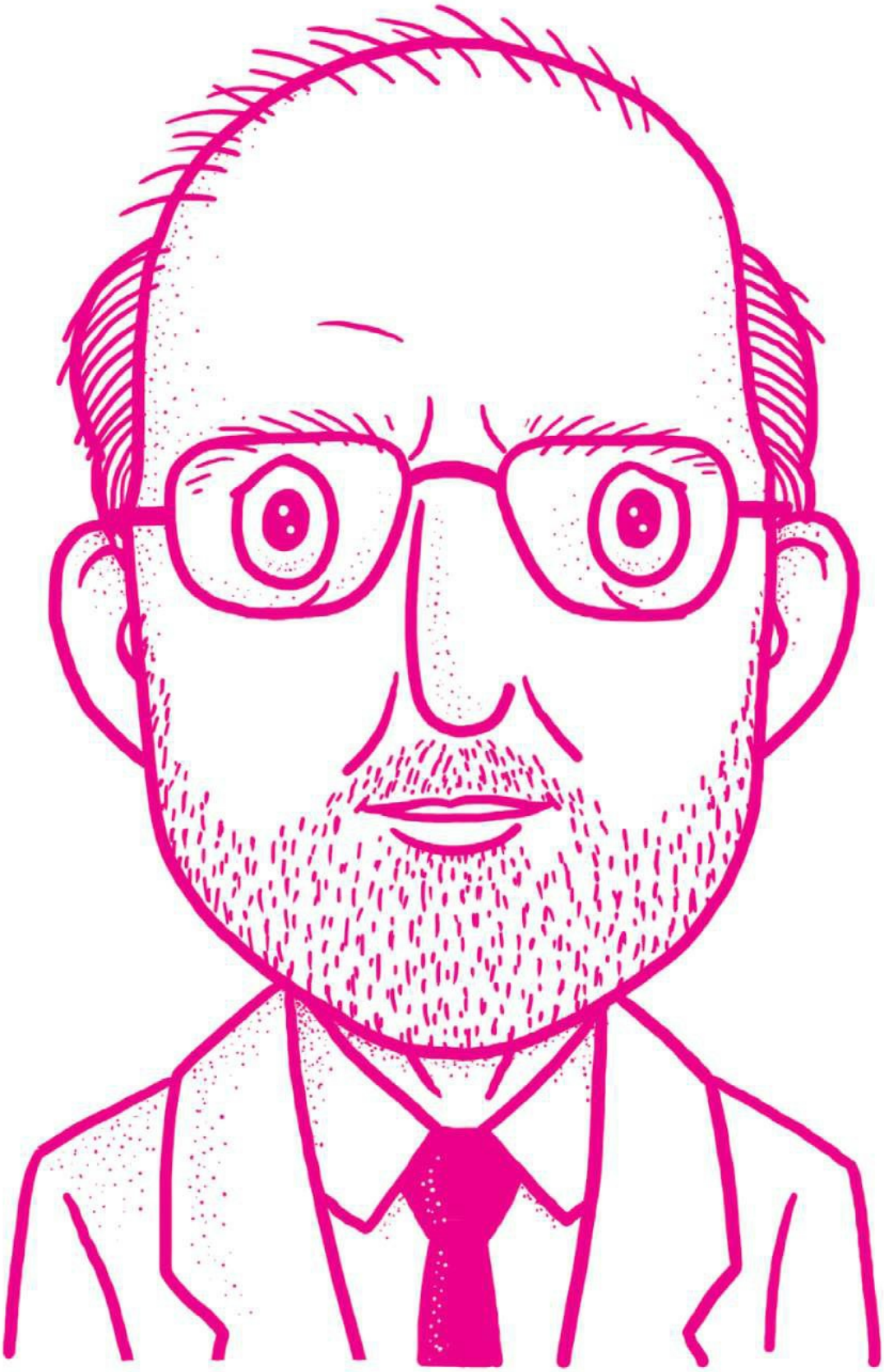
请注意他是如何从第一步完全无序的一堆信件进入第二步的。第二步的分组很重要，虽然在分组时并不需要考虑大小。在后续的每个步骤中，他进一步合并形成较大的信封集，直到最后这个集合包含了所有的信件。如果我们放大其中的某一个步骤来分析，例如第四步，那么我们就能理解归并算法是如何实现的（见下图）。



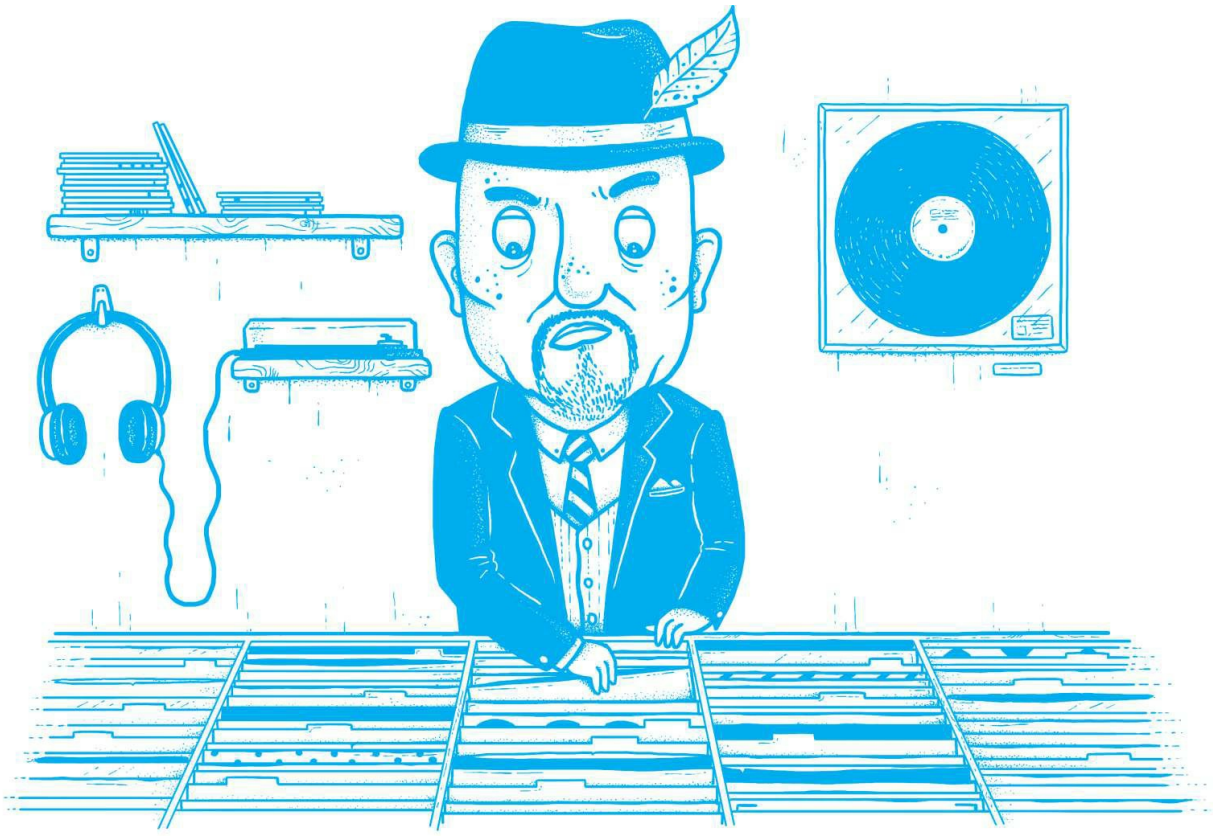
注：图中数字为门牌号。

由此可见，方法二确实是提高执行速度的更优选择。再来看查理的案例，他的优势在于，他需要排序的信件总量是有限的，只有33份。因此，无论用哪种方法，他都可以在很短的时间内解决问题，避免皮肤被烈日灼伤。但如果信件的数量增加，那么方法二的优势就会显现出来了，这时查理无疑就需要了解如何选择最佳的排序方式以节省时间。不用说，现在查理已经完成了他的送件工作了。

当查理完成了自己的送件任务，他一定很高兴自己今天学到了新的东西。伟大的思想无处不在，有时这些想法会出现在最不起眼的地方、最小的琐事中。



“不客气。”——托尼·霍尔，快速排序的发明者



---

[1] 并不是所有的事物都可以用类比法加以解释，事实上也不应如此。

[2] 平方时间是指将信件排序所需要的时间随着信件数量的增加而增加了n2倍，换句话说，将100个信件排序，假设10个信件用时100

秒，100个信件的用时就是10 000秒。

[3] 这个过程其实隐含了一个概念，即递归（recursion），我们在本书中并不会真正涉及这个概念，但是相信各位读者对此并不陌生。

[4] 回忆一下，对数函数的增长曲线是缓慢的。

第6课 解决问题 顺藤摸瓜的方法特别低效

福伊（Foy）最近移民到了美丽的小城阿什兰。他留着山羊胡子，每次出现在公众场合时都要在胳膊下面夹一本最新一期的《纽约客》。当然，这本杂志他从来没有看过，只是会在咖啡厅或者餐厅里漫不经心地拿起来翻一下——对于福伊来说，在这座城市里，他依然是一个局外人。他回答开放式问题最常用的一句话是，“我觉得弥尔顿镇（Milton）人不会喜欢的”。你觉得卡尔·奥韦·克瑙斯高（Karl Ove Knausgaard）的新书怎么样呢，福伊？我只能告诉你，弥尔顿人不会喜欢的。你喜欢阿黛尔（Adele）的新单曲吗，福伊？我只能告诉你，弥尔顿人会不喜欢的。……

福伊很担心自己苦心经营的形象会在这些问题中渐渐坍塌，他最终决定要克服自己的缺

点，成为一个真正了解艺术和文化的人。于是，他打算从世界上最有影响力的音乐着手，沐浴在艺术的长河中。理想很丰满，但现实有点骨感，一上来，福伊就遇到了一个大麻烦。他在阿什兰的唱片店里看到无数的唱片，但一时间不知从何下手。

目标：聆听有影响力[1]的音乐。

方法一：先找到一名有影响力的歌手，去听他/她的音乐；然后再找另外一名歌手，去听他/她的音乐；循环往复。

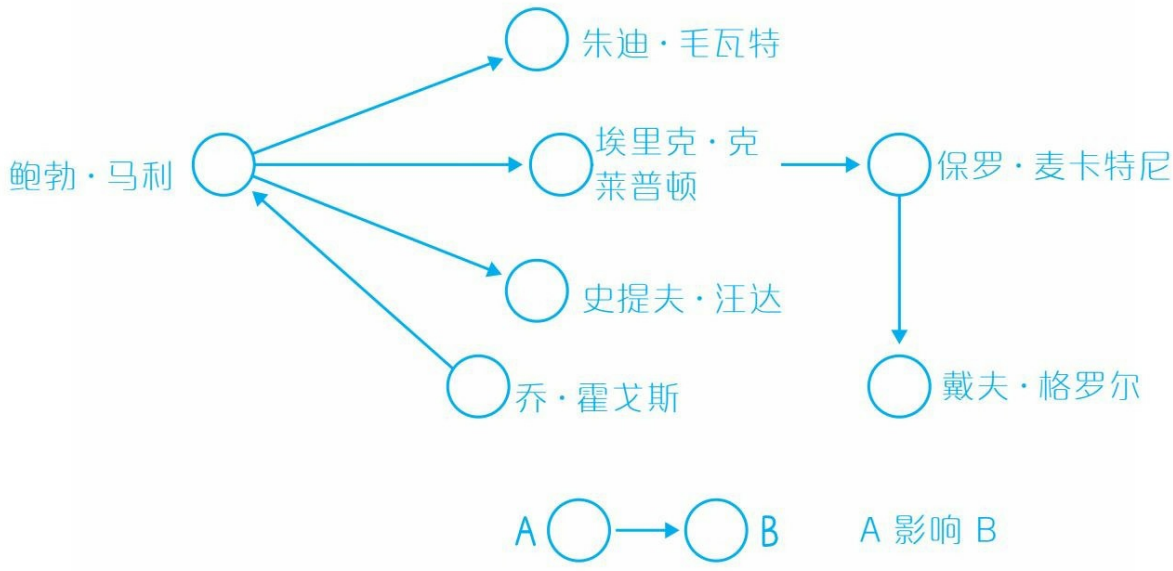
方法二：去一家唱片店，随便挑一些歌曲，开始聆听。

我们先来看一下福伊为了尽快提升自己的文化素养，而开启的这一段音乐知识的新旅途会是什么模样。

很多人会理所当然地选择方法一，因为在网上的推荐引擎很少会为我们呈现我们想要的内容，除非根据我们的偏好而调整。方法一从根本上来讲可以归结为链接分析法（link analysis），也就是说，如果在我们既有的集合中，各个元素之间有部分共同属性，例如歌曲或视频、人物或汽车组件，那么根据这些元素之间的关系——也就是它们之间的链接——我们就可以回答类似“这些东西中哪个最重要？”这样的问题。这正是我最感兴趣的地方。可能大家都很熟悉的一个例子就是引文。通常，一本出版物的引用率，也就是有多少其他作品引用了其内容，是衡量这本出版物影响力最重要的指标。这种根据指向率而衡量项目的方法，被谷歌应用于搜索业务，从而助力谷歌实现了突破性增长——相比于竞争对手，谷歌搜索结果首页所推荐的内容跟用户的偏好更相关。

我们来研究两种类型的相关性：事物之间的关联度（degree），以及这些相关事物之间的相似度（similar）。

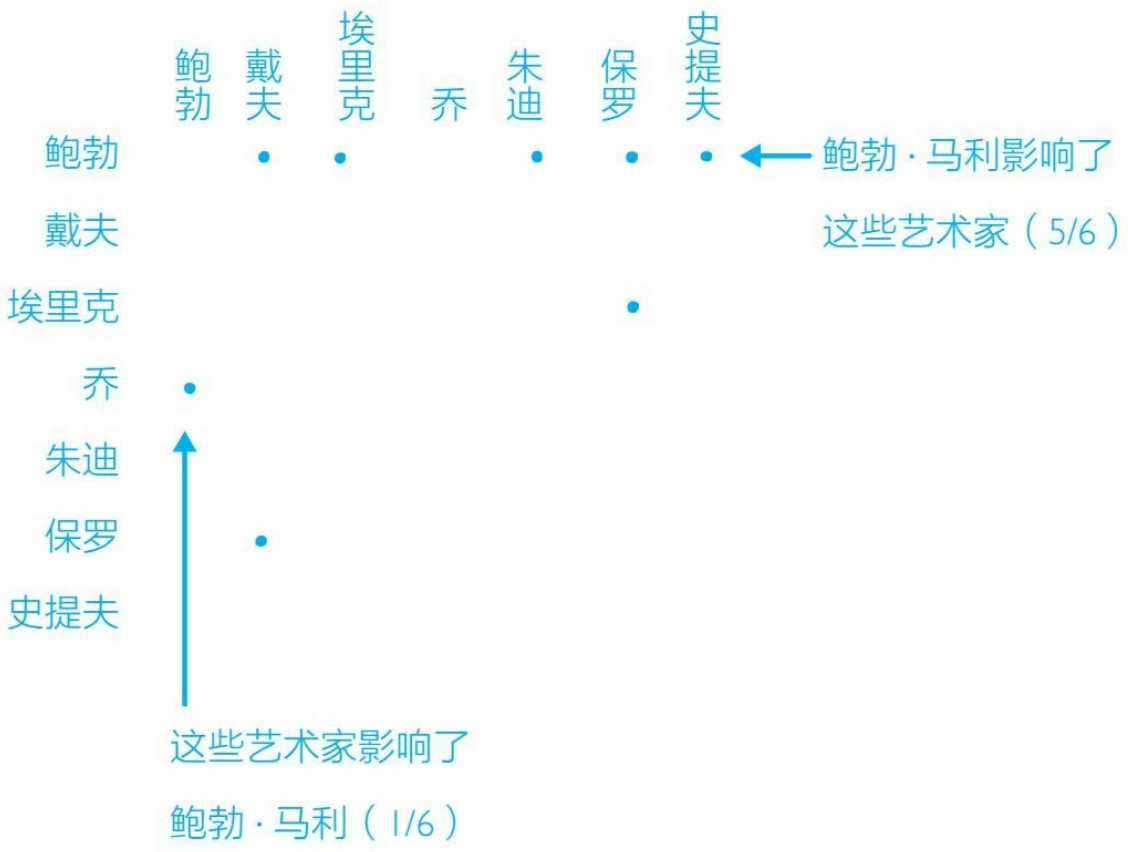
关联度：假设我们收录了世界上所有的歌曲，并且我们已知或者我们可以分辨出各组歌曲之间的关系——某一首歌的创作者受到了哪些音乐人的影响。我们可以通过查询人物之间的翻唱关系或者援引某些公开数据来做出判断。这个图谱的最初形态可能如下图所示。



如果我们把所有的歌曲都做这种判断，那

么最终将会得到很多圈子和无数链接（或者网络）。然而该网络中缺少了一个重要的信息，即间接链接。如果鲍勃·马利（Bob Marley）影响了埃里克·克莱普顿（Eric Clapton），而埃里克·克莱普顿又影响了保罗·麦卡特尼（Paul McCartney），那么我们会认为鲍勃·马利的影响力也延伸到了保罗·麦卡特尼身上。

为了捕捉这些间接链接，我们可以采用“矩阵乘法”（matrix multiplication）的解决方案，在这种解法中，我们将这些音乐家全部放置在一个矩阵中，从左往右，从上到下依次排列，纵向的每一位音乐家依次对应横向的音乐家，如果曾经产生过影响则标注一个点，以此类推，直到纵向的名单全部完成，也就是我们常说的完成了矩阵的传递闭包（matrix's transitive closure）。我们对这些矩阵求和，就可以得到类似下面的图。

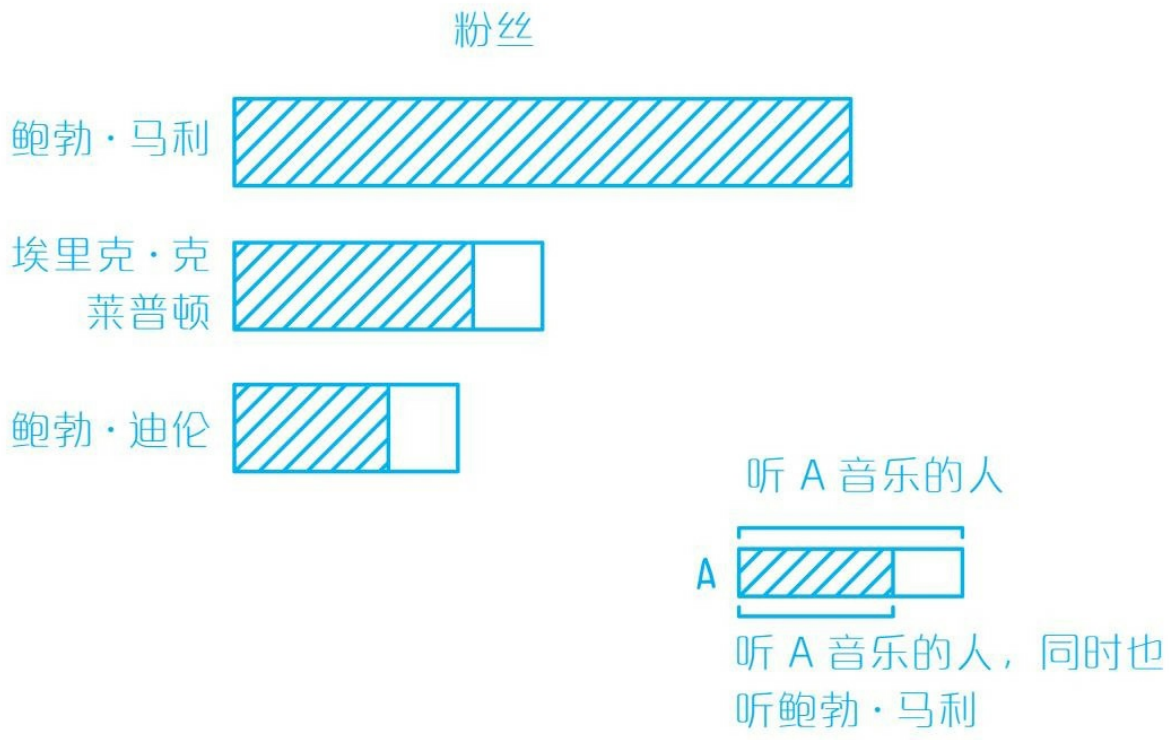


通过计算每个音乐家每一行的点数并排序，每个音乐家拥有的点数就代表了他／她曾经影响过多少人，我们就可以找出那些史上最具影响力的音乐家及其歌曲。这个程序适用范围是不确定的——我们可以把完全相同的步骤应用到汽车零件上，只需简单地改变一下链接的含义就可以了。就汽车零件而言，每一个点代表了物理依赖性，就好比车轮依赖于车轴，

车轴依赖于底盘一样。事实上，链接分析可以用来解决类似这样的问题：“我们接到了很多投诉，我们认为这是由于汽车模型过度复杂造成的。你能否给我一个零件关联清单，把连接点最多的零件识别出来？”车轮直接或者间接地连接到另外两个部分，有没有其他的零件连接到了其他4个部分呢？甚至是5个或者10个？在第一个案例中，对于音乐家来说，连接的点越多说明这位音乐家的影响力越高，但是在汽车零件的案例中，连接的点太多就不是件好事儿了，这有可能意味着这款车型在设计上是失败的，容易出现故障。

当然了，这可以作为一个起点。如果排除流派之分的因素，我们可能会选择从最具影响力的音乐家开始，然后次之。而如果我们更喜欢从一首歌到另外一首风格类似的歌曲时，我们就需要采取其他的方式了。

相似度：确定哪些歌曲可能跟某首特定的歌曲相似的方法有很多，其中一种是查找这些歌曲背后的艺术家。确认音乐人是否类似的方式之一是考虑听众。以鲍勃·马利为例，有多少人在听埃里克·克莱普顿的同时也在听鲍勃·马利？有多少史提夫的音乐迷也在听鲍勃·马利的音乐？等等。如果我们把所有的音乐家全部考虑进来，然后将结果从最高到最低排序，我们就会了解哪些音乐家可能跟鲍勃·马利的风格最相似（见下图）。这个方式可能会有一些细微的偏差，但是可以通过一些更高级的方法来改进，例如，采用杰卡德系数（Jaccard index）而非简单的加总，从而避免因为某些音乐人的粉丝数量巨大而引起的结果倾斜。

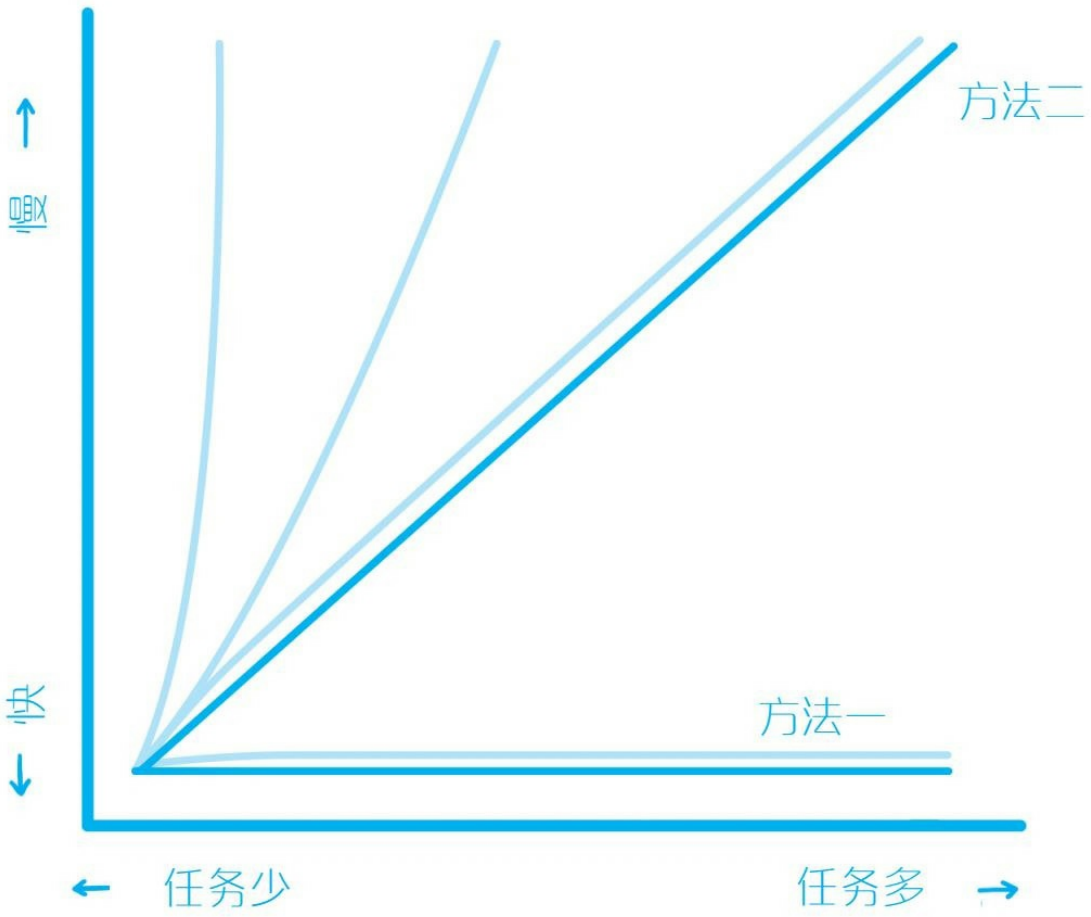


当你打开一个搜索引擎的结果页，或者在社交媒体上显示的新闻提要中，或在购物网站浏览时显示的“您可能想要购买的商品”推荐，抑或在职场社交网站上出现“您可能想要关注的人”，你就会看到这种分析形式带来的结果。报纸也会这么做，通过对一篇文章进行建模，例如文章所涉及的主题，然后考虑该文同其他文章的相似程度。而视频流服务的竞争优势在于准确预测用户喜好，并推送相关内容。最近的

一篇博客文章揭示了网飞（Netflix）在向用户推荐电影和电视剧时会考虑的信息，其中不仅包括内容类型——“您正在观看科幻内容，您可能还会喜欢这些科幻内容”——区域因素也会被考虑在内——“您正在观看烹饪节目，而您的位置位于印度，所以您可能会喜欢以下宝莱坞电影作品。”据估计，网飞80%的内容浏览量都是推荐引擎带来的。正如我们在第4课提到的，这种连接与正确的分析相结合可以带来卓越的精准洞察。[2]

方法二的结果是显而易见的，因为它本质上是一个无意识的选择。如果我们在一家唱片商店，我们可能会走到一个CD箱前挑选一些唱片。就像所有无意识的选择一样，我们无从知晓每一个选择距离我们的预期有多远。即使我们偶然选择了一些很有影响力的音乐，我们可能也不知道。

通过利用链接分析的结果，我们不再需要在我们感兴趣而又陌生的领域中随机猜想该从何做起。如果福伊愿意利用技术进步带来的福祉，我们可以将这两种方法以下图的方式展示给他：在最坏的情况下，方法二占用的时间是线性的，而方法一则是恒定的。方法二呈线性，是因为福伊可能要试听一下所有的歌曲才能找到他喜欢的那一种类型；而方法一是不变的，因为无论世界上有多少歌曲，福伊都会在最有影响力的歌曲中展开他的音乐之旅。



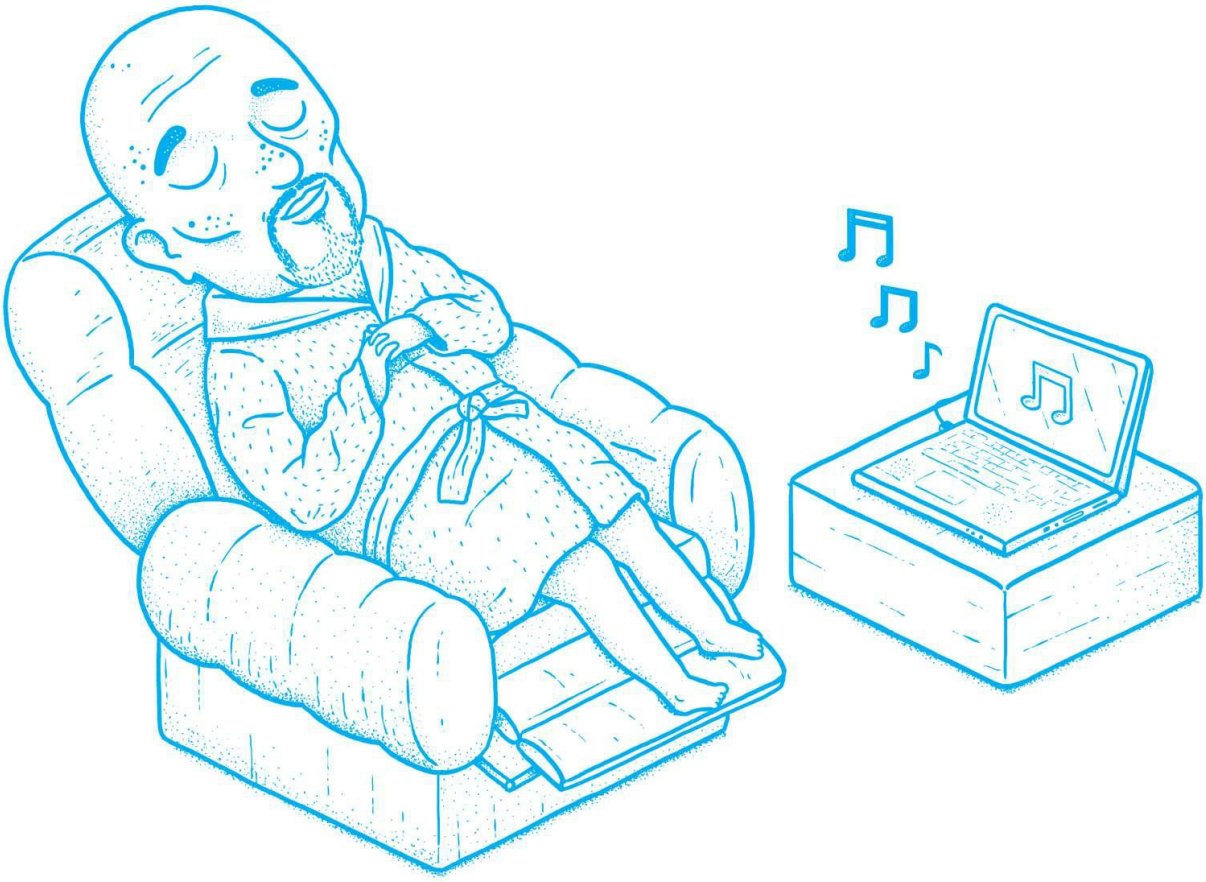
福伊的故事其实非常具有普遍性，我们可以用另外一个完全不同领域的例子来说明——政治。直到19世纪，美国的政治仍旧与现在大相径庭。在选举中，街头挤满了男性（女性直到1920年才可以参与投票），他们游行、喝酒、投票。那时，投票已不再是一种社会活动，这就意味着政治家不得不走出来拉选票。

19世纪90年代，威廉·詹宁斯·布莱恩（William Jennings Bryan）创造了可能是第一个支持者邮件列表——一个数据库，这个数据库陪伴了他此后的政治生涯。20世纪，这类数据库变得无处不在。到了21世纪，数据库在政党层面开始精简，并获得了基于购买行为等细微特征的人群定向能力。

这个跨越数百年的发展向我们揭示了一个道理，为了让政党的拉票变得更加有效，节省时间和金钱，他们必须先弄清楚选举从哪里着手。他们发现将目标锁定在那些更有可能与之互动的人群上，要比把信息笼统地在全国范围内进行传播更行之有效。这是一个具有普遍性的问题，其解决方法同样可被推广。事实上，这种方法非常重要，它几乎影响了我们目前使用的每个主流网站和服务。

而这对福伊来说意味着什么呢？现在他是

否在文化修养方面达到了自己的预期呢？我们无从知晓，但是我们的目标重在授之以渔而非授之以鱼。学习新鲜事物的陷阱在于从错误的地方起步，一旦开头错了很容易遭遇挫折，导致梦想幻灭或者中途放弃。链接分析的结果是一种由互联网推动的创新，也许在不久的将来，借助家庭和办公的物理设备，那些像福伊一样好学的人就可以轻松获得新技能，而不会无从下手或者走弯路。在福伊的案例中，他选择放弃卢德分子的做法而拥抱新技术，分析了数以百万计的歌曲，这帮助他更快地融入新的文化阶层。目前，他已经鼓起勇气报名参加了一些当地音乐爱好者的聚会，看起来他的情况还不错。



尽情享受吧，福伊。可能这在那位长着浓密胡子的德国哲学家看来，成功不可能轻而易举，但付出总有回报。

[1] 所谓“有影响力”，这里指的是那些可以鼓舞人心或者启发了其他艺术家灵感的音乐，也可以指更通俗的，比如流行音乐。正如我们会在后文中看到的，我们最关注的问题其实是“这些音乐中哪一首最重要？”

[2] 考虑向某人推荐相似事物的长期实际效果是很重要的。对于福伊来说，就他的目标和计划而言，这样做是有道理的，但总看同一

种类型的节目、读同一种类型的书或者仅取一家之言真的是件好事儿吗？这不会妨碍一个人体验生命的全部吗？算法就是背后的人的反映。我们应始终清醒地认识到人类存在的偏见，这种偏见不仅存在于我们的言行之中，更存在于我们创造出来的事物之中。

第7课 高效表达 让别人读懂你的言“外”之意

德韦恩（Dwayne）正在加拿大落基山脉徒步旅行。清澈的绿松石湖泊水平如镜，水面倒映着壮丽的山峰和青翠树木。鸟儿在晴朗的天空下欢快地掠过，一阵微风吹向东方。在那一刻，这里仿佛是世界上最宁静之地，没有血流成河，没有气候变化，没有贫穷，也没有坎耶·维斯特（Kanye West）。“理想主义者是对的。”一团转瞬即逝的云朵在地平线上悄悄地说。“他们说得对啊。”虽然德韦恩这么说，但是他的思绪却飘在了别处。你看，那天早上，就在他的团队从温哥华出发的时候，一个奇特的景象吸引了他的目光，他看到一只野鸭子在来回摇摆，好像在跳伦巴舞一样。从那时开始他就一直在绞尽脑汁，努力地想要找到一个诙谐而简洁的句子来描述那个欢快的场景，并发布在自己的社交平台上。鉴于140字的限制，这

段文字并不容易想，而这个限制正是他用来释放自己的社交工具施加的。他可不想让那些他一直依赖的陌生人失望。

正如神经系统学说里提到的，大脑具有灵敏的探测能力。当你身处一个安静的房间，听到任何一点声音，你的大脑都会接收到。而如果你在一个嘈杂的环境中，听到一些与众不同的声音，那么你的大脑也会把它记下来。从某种意义上说，频发的信息通常会被认为没有意义，这也是大脑知道如何过滤信息的原因。

有的人发文本信息时，常常会省略一些字母（如元音），这种方式符合信息论的观点：“只有重要的信息才会被接收。”鉴于英语自身的特点，尽管省略了部分字母，句子的意义也完全可以被接收方理解，因为被省略的字母是可以被猜想出来的，因而可以被省略。正因如此，当我们面对一段缩略的文字时，依然

可以理解其中的含义。对于德韦恩来说，采用这个方法并不是一个坏主意。事实上，在预测性文本出现之前我们就已经开始这样做了。[1]为了节省空间，这种方法确实会导致数据丢失，尽管是非必要的数据。请注意，到目前为止，我们讨论过的主题都围绕处理问题的速度，而在这一课，我们要探讨的问题是如何减少空间的占用。这种平衡反映了我们在实践中看到的评估解决问题的不同方法时的典型平衡——通常，计算机科学家们对比各种方法的相对速度［也称为运行时复杂度（run-time complexity）］，但偶尔也会基于不同方法需要占用多少内存来进行比较［这被称为空间复杂度（space complexity）］。

目标：重写一篇诙谐的博文，不超过140个字符。

方法一：用短单词替换长单词，牺牲文字

的表现力。

方法二：在部分单词中省略部分字母，如元音。

有趣的是，方法二的算法中有一个模拟值。1952年，计算机科学家哈夫曼（David A. Huffman）提出了一种减少存储数据所需空间的方法。与前面的例子不同，哈夫曼的方法不用删除数据，而专注于优化，正如我们在后文可以看到的。计算机存储数据（如单词）的方式是将字母表中的字母（以及数字和其他字符）映射到一组数值，然后使用计算机可以识别的二进制表示法来存储这些值。每个字符都采用二进制代码表示，二进制可能由7位组成。例如，字母“a”被映射到值97，在二进制中的表示为：

1 100001

而字母“b”被映射到值98，在二进制中表示为：

1 100010

如果我们想用二进制来表示单词“hans”，每个字母占用7位，共28位，则可以表示为：

1 101000 1 100001 1 101 1 10 1 1 1001 1

具有相同长度的二进制代码（在本例中为7位）的字符的好处在于，它使解码二进制字符串变得容易。我们要做的就是每读取7位数据，就可以通过映射表将其解码为英文。

不过哈夫曼是个特立独行的人。他看了看那些7位的代码说道：“当然，肯定是有办法来压缩数据的。”他的朋友却持反对意见。“不，哈夫曼，”他们说，“这是不可能的事情，哈夫曼。别对自己太苛刻，哈夫曼。不要逞英雄，

哈夫曼。”但是哈夫曼却不在乎，如果这意味着他可能会为一组字符想出一种最佳的二进制表示法，他愿意将自己置于未知的世界。

哈夫曼并没有采用固定长度的二进制码，而是选择了可变长度的代码。他利用这样一个洞察：将句子中出现频率比其他字符更高的字母映射到较小的值，把出现频次较低的字母映射到较大的值，从而进一步缩短了二进制代码。例如，假设我们确定了某一组数据，字符出现的频率分布如下图所示：

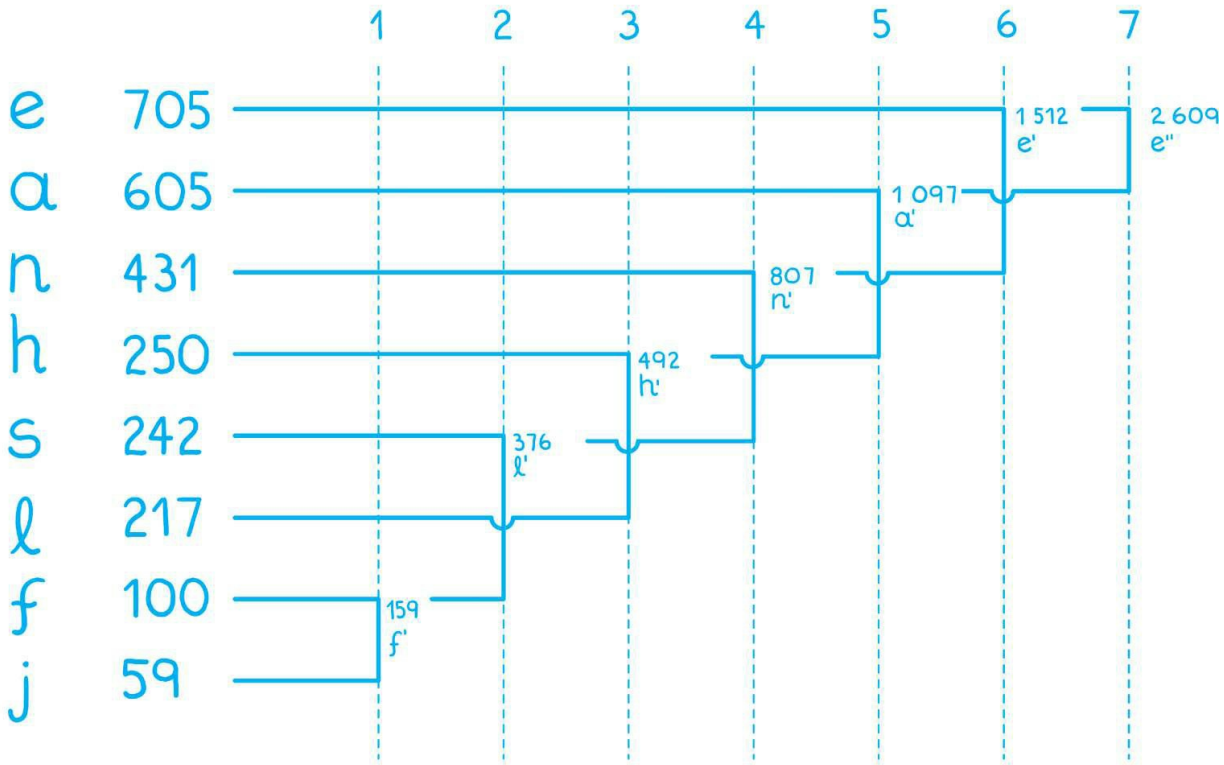
e	705
a	605
n	431
h	250
s	242
l	217
f	100
j	59

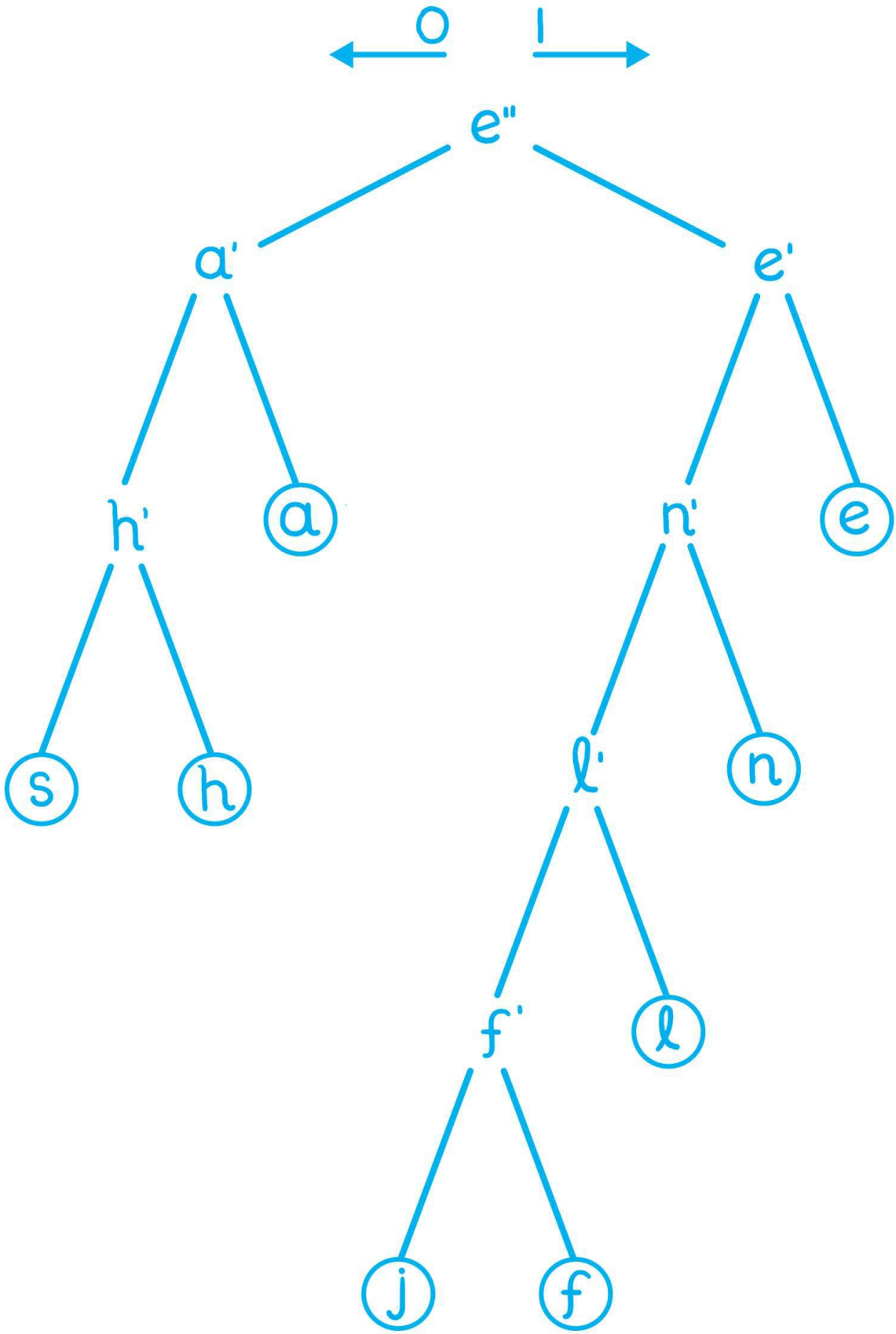
从上图可以看出，字母“e”出现了705次，字母“a”出现了605次，依此类推。请注意，字母将按照出现频率的高低进行排列。哈夫曼的方法是采用一对出现频率最小的字符，对它们的值求和，把结果存储在一个新的临时字符中，然后对该集合进行排序。重复该过程直至最后。

我们最终得到的基本是一棵树，其中每个节点（一个字符）连接到两侧枝的节点。如果我们用上面的字符集合为例，最终会得到如下图的结果。在这里我们首先将“f”和“j”配对，然后将结果与“l”相匹配，以此类推。每一列（从第二列开始）就是算法的一个步骤。

当我们把图表重新排列成树状时，就会变得很清晰。一个经过优化的字符的二进制代码就是我们从树状图的最顶端节点，也就是“根节

点”，[2]向下读到该字符节点时得到的字符串。因此，在下面的树状图中，每当我们往左移动，就要在该二进制代码中添加一个0，当我们向右移动时，则需要添加一个1。这也就是为什么字母“e”得到的是一个两位的二进制代码11，而“f”则是一个五位的二进制代码10001。在哈夫曼树中，为节点的子节点赋值1或0是任意的，即“e”可能被编码为01而不是1 1。虽然二进制码不能保证是唯一的，但可以确保是最优的。在任何情况下，哈夫曼树会与信息一起发送给接收者，以便接收方了解如何解码文本。


1
2
3
4
5
6
7
e
705
a
605
n
431
h
250
s
242
l
217
f
100
j
59
159
f'
376
l'
492
h'
807
n'
1 097
a'
1 512
e'
2 609
e''


0
1
e"
a'
e'
h'
a
n'
e
s
h
l'
n
f'
l
j
f

下图是我们优化的二进制代码列表。请注意，出现频次越高的字母代码位数越少。

e	a	n	h	s	l	f	j
11	01	101	001	000	1001	10001	10000

那么在二进制中，单词“hans”是如何表示的呢？答案是：

001 01 101 000

事实上，我们表示这个单词，只需要1 1位而非28位。这种洞察让人想起了近一个世纪前的一种创新——传递电报的最佳方式。那时，塞缪尔·摩尔斯（Samuel Morse）也是基于英语中出现的字母的频率来决定用什么代码来表示。有趣的是，摩尔斯并不是通过与专家交谈，也不是进行研究或分析数据来确定字母出

现的频率，而是通过计算打印机的铅字盒中的铅字数量来确定的。所以下一次如果有学者来质疑你的研究方法时，别怕。

像哈夫曼编码这样的数据压缩技术在现实世界中非常重要。充分优化空间利用意味着网站的加载速度会更快——网站服务器可以在文件通过网络发送之前进行压缩，现代网络浏览器可以对其进行解压。当带宽不足时，任何类似这样的速度增加都显得至关重要。压缩也意味着储存电影（如MPEG–2）、图片（如JPEG）和歌曲（如MP3）所占用的空间更少，从而节省存储和传输的费用。像MP3这样的音频格式非常有趣，其压缩模式依赖于音频构成，而这部分构成是人耳因生物及神经系统的局限而无法听到的。例如，人耳无法听到高于20 000赫兹的声音频率。

下一次，当你与其他人进行稳定、流畅的

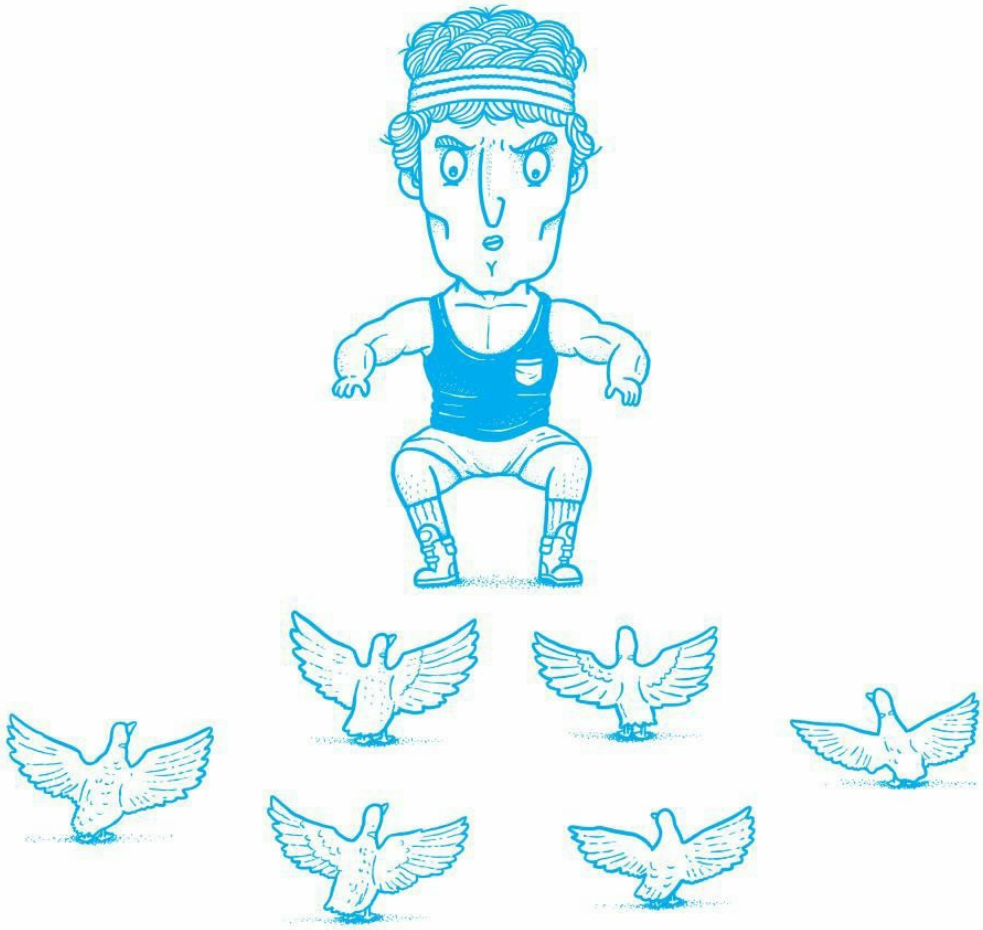
语音或视频对话时，你就明白压缩是一件多么重要的事情。如今，技术已经足够成熟，你的请求只需要通过网络发送一些数据包，在网络的另一端就可以推断或者重构这些数据。事实上，压缩有助于降低技术应用的障碍。

这一切似乎都很美好。但是德韦恩呢？德韦恩和他的粉丝是不是都满意了呢？幸运的是，答案是肯定的。这个男孩成功了。他成功地将那个有趣的场景表达了出来。通过他诙谐幽默的描述，野鸭的形象活灵活现地出现在了粉丝眼前。这就是文字的力量。这就是威廉·廷代尔（William Tyndale）为之牺牲的东西。德韦恩更新了他的社交状态，数十万粉丝翘首以待，他们的等待没有被辜负。

> 黎明时分开始了一段梦幻般的徒步，最赞的是偶遇一场野鸭子的舞蹈演出，还不收门票呢。

人们希望德韦恩的这种在不丢失任何重要信息的同时尽可能压缩内容的传递方式，总是能够带给我们类似的快乐结果。不然，这群可爱的小鸭子又怎么能活灵活现地出现在读者面前呢？

用户留存培训



---

[1] 在热门视频分享网站上的评论中，怀旧的人会把这种情怀传递给后代。

[2] 在计算机运算中，树的根在顶部，树枝向下延伸，这跟真实的树正好相反。

第8课 推进项目 怎么在上班时间完成所有工作

郭诺亚（Kwee Noah）在农业生物技术公司Hyperbowl的圣路易斯办公室工作，该公司专门从事转基因改良种子的销售。她是一位行事低调的秘书，而本周老板给了她一大堆不可能完成的工作任务，更过分的是还要求她周五汇报，而今天已经是周三了。如果郭诺亚没有按时完成任务，那么她就不能参加本周六举办的公司年会了，这场年会是Hyperbowl举办的唯一一场高管与员工交流的活动。这是郭诺亚抛头露面的最好机会，如果幸运的话，还有可能帮她得到一直希望的晋升机会。那么郭诺亚应该怎么做呢？

对于一个成年人来说，完成任务是最重要的责任。想一想，你看过多少关于如何高效完成工作、如何避免拖延以及更好地管理工作负

担的文章和书籍。勤奋的郭诺亚深知，在工作中遇到不合理的交付时间，会带来很大的压力。那么让我们一起来看看有什么方法，可以让她完成工作目标。

目标：在本周内完成所有工作任务。

方法一：先完成一项任务的一部分，然后做第二项任务的一部分，再回到第一项任务中。循环往复。

方法二：将这些任务按照从简到难排序。从最简单的任务着手，完成一项任务后，再开始下一项任务。

方法三：将任务按照优先级排序。从最紧要的任务着手，每完成一项任务，再开始下一项任务。

你应该对这些方法都感到很熟悉。对于方

法一来说，我们使用时间段作为决定何时从一项任务切换到另一项任务的触发器。比如说，我们有三个问题需要解决，分属三个不同的类别，并且需要在周末前完成。我们可能会在上午做第一个，下午做第二个，晚上做第三个。然后，在第二天早上继续进行第一项任务，直到完成所有任务。这种时间分割法实际上是现代操作系统处理多个应用程序的方式，有时也被称为上下文切换（context switching）。调度程序查看当前正在运行的程序[1]，并为它们分配时间片，确保每个程序按照分配的时间运行。因此，进程之间的切换是无缝隙的，以至于操作系统给人一种进程是并行运行的错觉。如今，有了多核处理器，进程确实可以并行。一个有四个内核的处理器可以并行四个程序，并且只有运行的程序多于可用处理器内核时才需要上下文切换。这让人想起了一个并行处理在现实生活中的有趣应用——流水线。把一组

互相关联的任务分解，然后以一种能够优化可用资源的方式执行。假设你和两个朋友刚刚想起还没有人为即将结束的派对准备礼物袋，为了在单位时间内最大限度地完成包装礼物袋的任务，采用流水线的方式可能会更为有效——你可以负责在礼物袋上写祝福语，你的一个朋友负责将礼物装到袋子里，另一个朋友在礼物袋上系丝带。否则，如果等你把所有的祝福语都写好再装袋，会导致你一个人很忙碌，而同时你的朋友却是空闲的。将任务分解是提高效率的重要方式，但是在某种意义上，正如弗雷德·布鲁克斯（Fred Brooks）所说，让9位女性在一个月内孕育出一个孩子是不可能的。

鉴于目前的硬件水平，我们并不能真正感受到上下文切换的费用有多大。但事实上，每次操作系统进行上下文切换时，都必须挂起最后一个进程的状态，清理寄存器和瞬态数据，然后加载到新进程状态。[2]对于人类来说，认

识到这类切换存在的问题具有重要意义。一次又一次，我们发现生产力的最大障碍之一就是：为了响应紧急需求而不得不停止正在做的事情，去处理其他的事情，然后再回到之前做的事情上。同理，操作系统也是如此，中断处理程序能够暂停正在运行的程序，例如，让程序中的某些数据处于等待状态，或降级优先顺序。

如果中断时间足够长，那么我们在试图回到之前任务的状态时可能会挣扎一下。另一方面，上下文切换也确实有其优势，可以确保我们每个任务至少都完成了一部分。同时，当时间紧、任务重的时候，“没有任务掉队”原则就成了一种理想主义，它会导致普遍的失望和民众被剥夺公民权的现象。

至于方法二，则是拖延症患者所熟悉的一种方法。它把最艰难的任务推迟到最后，而优

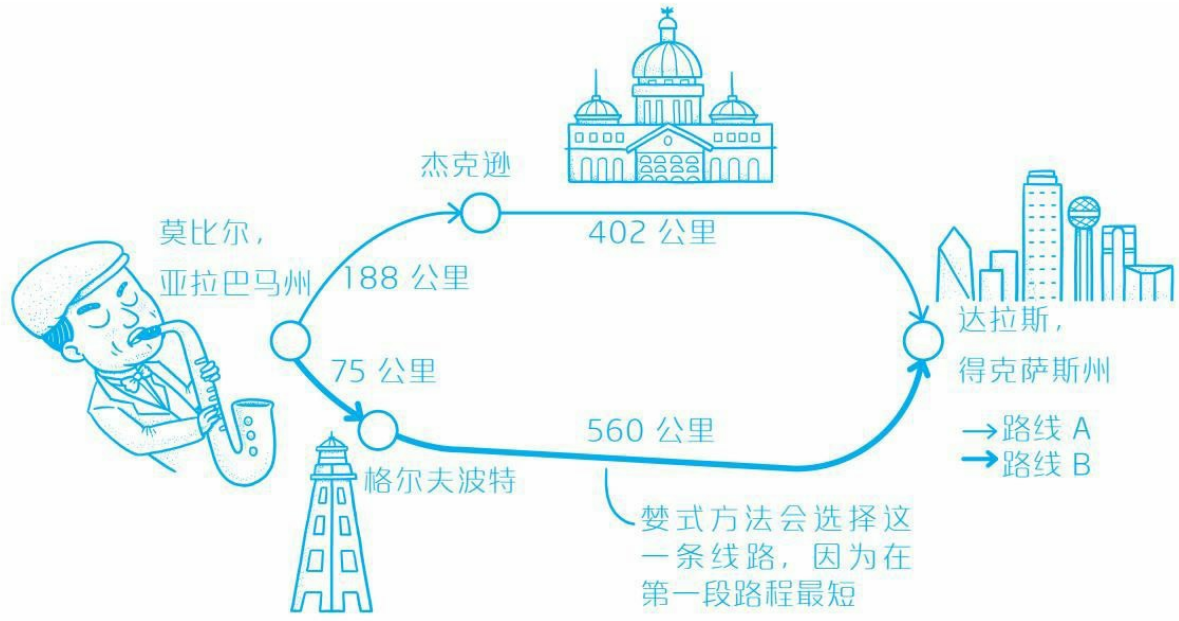
先选择解决简单的任务。这有点机会主义，其优点是可以获得大范围的快速胜利，但后期可能需要付出较大的代价。这种方法有时被称为婪式方法（greedy approach），这个术语其实并不带有贬低或轻蔑的意思，而是强调这种方法能够通过最少的付出完成尽可能多的任务。

婪式方法的一个实际应用是试图在尽可能短的时间内从一个点到另一个点，比如从一个城镇到另一个城镇。利用婪式方法，我们可能会问：“离这里最近的城镇是哪个？”这是一个快速且容易做出的决策，尽管这可能会阻碍我们考虑那些整体偏短的路径，因为这种方法偏向于考虑初期的较短路径，而忽略了后期。其实我们早在第4课中看到过另一个版本，虽然伊奥尼斯只关心如何走出去，而并不关心走出去要花多久。目前已经有很多关于最短路径算法的文章，并且存在一些互相竞争的算法。其中一种著名的方法——迪克斯特拉算法

（Dijkstra's algorithm），是由荷兰计算机科学家艾兹格·W. 迪克斯特拉（Edsger W. Dijkstra）于1959年提出的。通常，这类算法被称为图搜索算法（graph-search algorithm）。

值得一提的是，婪式方法的另一个应用是模式匹配：一系列字符（模式）被用于搜索文本主体，以寻找潜在的匹配对象，通常是为了替换现有对象。假设一段文本包含“Jessa Jessica”一行，而且我们对合并这个名字的所有不同拼写方式感兴趣。给定一个婪式方法匹配，“搜索以‘a’结尾、前缀为‘Jess’的所有单词”，那么返回的字符行将是一整段（“Jessa Jessica”），而非独立的名字（“Jessa”，“Jessica”）。这是由于匹配器会在最后一个“a”上停止，而非第一个。有时这种方式是可取的，但在下面这种情况下却并非如此。

在一条你很陌生的道路上，做出这样的选择其实很容易理解。比如你在州际公路上驾车，有一个路标告诉你距离最近的三个城镇分别有多远，你极有可能会选择最近的那一个（见下图）。



非婪式方法无疑会更加复杂，但是往往会带来更好的结果。我们在军事演习中可以看到这个有趣的模拟。例如在首都保卫战中，为了获得更大的胜利而放弃第一战。是不是想到了俄国在1812年与拿破仑大军的战役？因此这种

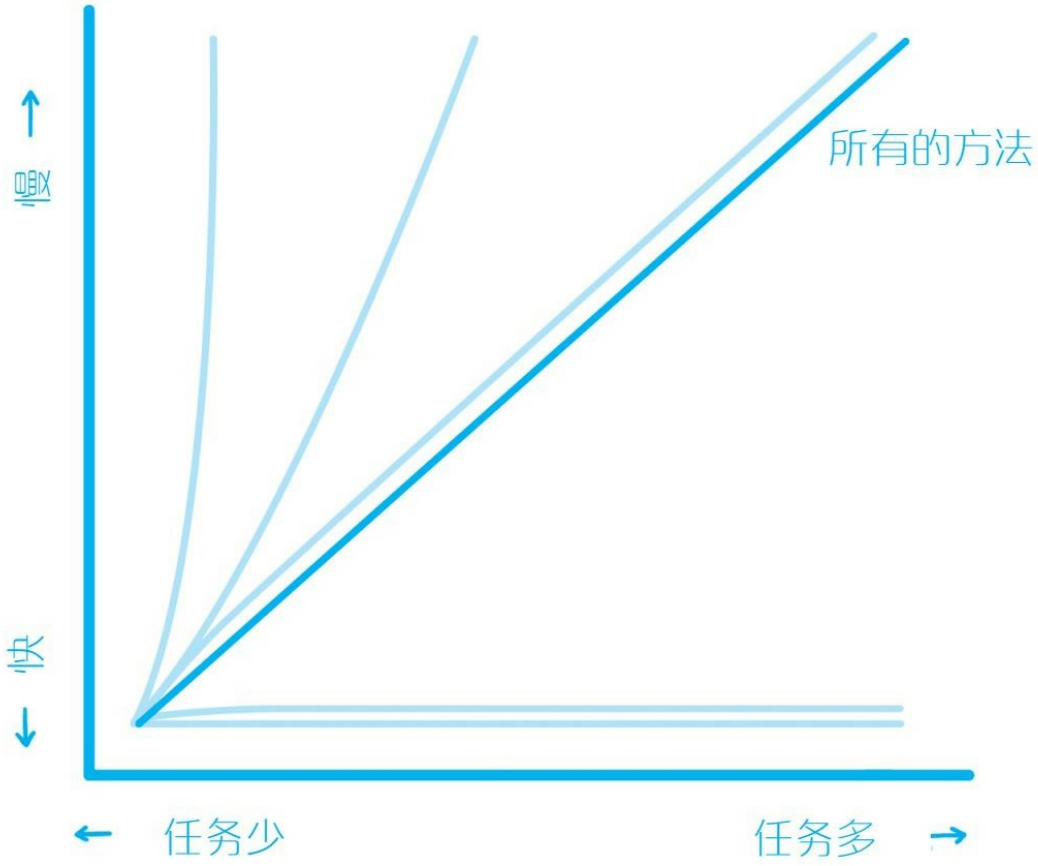
方式也称为打持久战。《华尔街日报》最近发表了一篇关于尼日利亚在拼字冠军赛中崛起的文章，文章认为他们获胜的缘由可能并非是由于词汇量，而是选择短词的反直觉策略。尼日利亚的拼字游戏玩家发现，相比于七八个字母的单词来说，四五个字母的单词可以带来更好的总体策略，因为玩家可以将最有用的字母留到后面几轮，并且减少了抽到不好用的字母的机会。这篇文章将这种方式描述得很精妙：

> 一名英国人以一个8个字母的宾果游戏取得了86分而领先，但是在接下来的五轮比赛中，他一直在摆弄一些不太顺手的字谜，得分只有二三十分。随着金格黑尔先生（Mr. Jighere）用一个4个字母的单词得到了93分的高分，他一下子领先了。最终的比分是449:432，尼日利亚队获胜。冠军队举着奖杯在房间里欢呼雀跃，唱着尼日利亚流行歌曲：“我们赢了。”

方法三专注于那些实际上更关键的任务，而不是那些涉及较少、影响较小的任务，从而避免了方法二会出现的问题。我们将任务或者事件都列出来，按照优先级排序。请注意，我们这里提到的“优先级”事实上可能会根据完成时间来确定，这就是为什么可以说方法二也是按照优先级排序的原因。一个类似的能让你有所体会的应用场景可能是打印机。如果一台打印机有10个50页的任务要执行，然后后面跟着一个1页的任务，那么打印机可能会优先执行后面的单页文件打印任务，而不是等到打印完了500页之后再执行下一个任务。在这里，我们区分两种方法是为了强调优先级可以依赖于时间之外的其他属性。

如果有新的任务进来，我们可能会因为它的优先级而把这个任务夹在中间，而不是排在最后。可以想象一下，在已经排好的优先级列表中添加任务时，如果为了腾出空间而删除其

他任务，反而可能会占用更多的时间。在第12课中，我们将会讲到计算机如何存储这样一个列表，通常这个列表被我们称为“优先队列”（priority queue），以便更快速地插入任务。通常在生活中，这种方法是最高效的。下面这张图就展示了三种方法的差别：



假设任务之间彼此独立，也就是说我们进

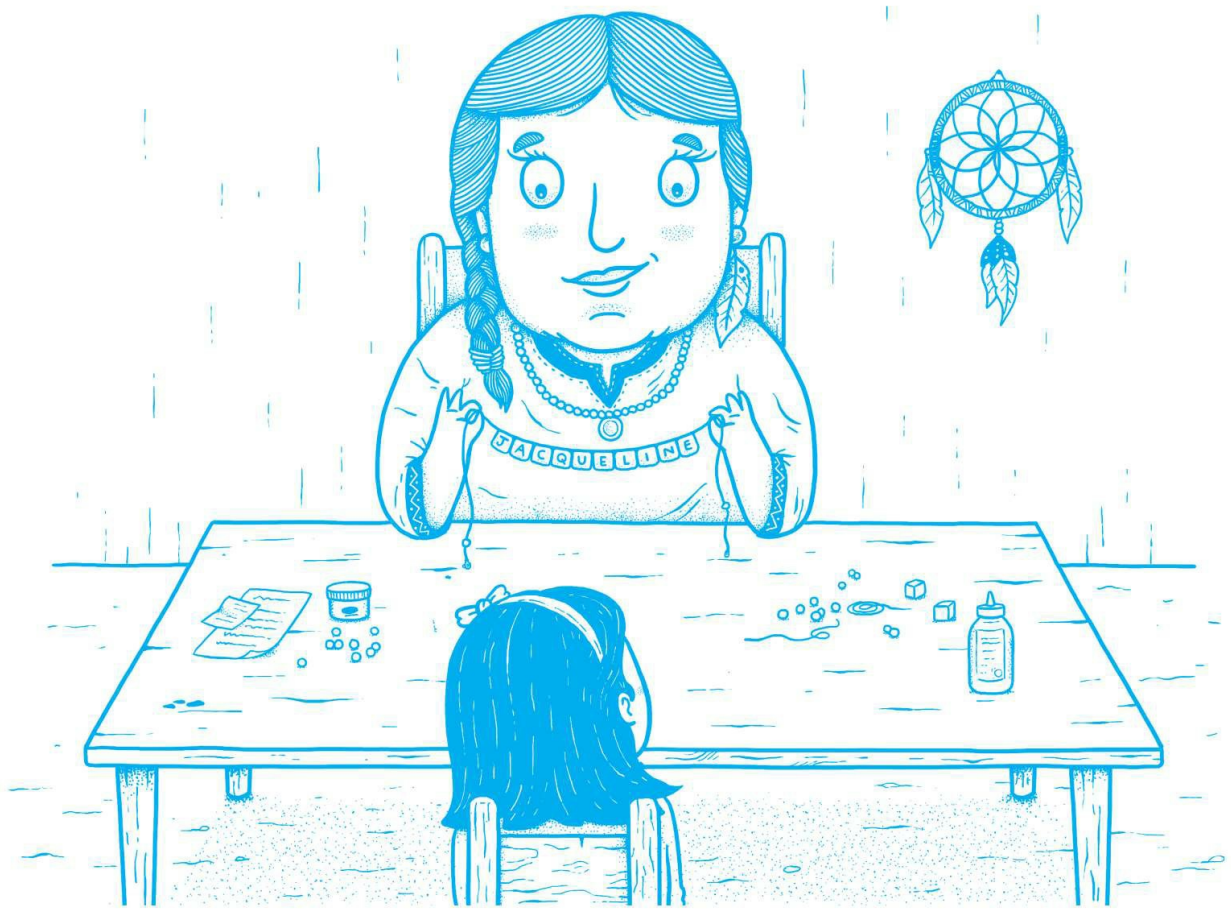
行早期任务排序对于完成后续任务所花费的时间是没有影响的，那么这三种方法对郭诺亚来说都是一样的。[3]如第1课所述，我们正在对三种方法的基本操作进行比较，看看郭诺亚要花多长时间来完成这些任务。如果我们反过来考虑郭诺亚用来构建和维护任务的时间，那么我们可能会说方法一所有的时间是恒定的，也就是0，而方法二和方法三在最坏的情况下会呈现对数时间。既然如此，为什么还要考虑方法二或方法三呢？可能是因为考虑到帮助郭诺亚完成工作任务所涉及的成本，我们认为方法一与另外两种方法相比，可能导致更高的总体成本。也就是说，在实践中，构建和维护一组任务并不重要。[4]更多资料可参考第10课和第12课。

以下场景是一个很好的例子：当只看相对量级已经不足以说明问题时，考虑任务的实际结果就显得至关重要。这同样适用于恒定时间

法。想象一下，一名代客泊车服务员，希望在停车场有限的地方停进更多车辆，以最大化地利用空间。假设你帮客户取车的时间是一定的，但客户的体验仍旧会存在很大差异。如果停车场塞得太满，没有行车道时，那么服务员可能就需要挪动其他8辆车才能取回客户的车。而如果留一个行车道出来，那么在最坏的情况下，可能也只需要挪动3辆车即可。而如果通过诸如“禁止进出特权”之类的政策来强行限制车辆何时可停放、何时可取回，结果则可能不需要挪动任何车辆就可以取回客户的车。

在无法达成整体目标的情况下，对郭诺亚来说，能达成的最好结果是什么呢？如果碰巧她所关心的是高优先级的任务，那么最开始恰好碰到了一项她既不擅长又很麻烦的任务，那么她这一周是不是都卡壳了呢？或许我们可以将优先级队列方法与上下文切换法结合起来？[5]当任务要求不清晰或遇到自己擅长处理

的情况时，像这样提出疑问就会找到一些新颖的解决方案。顺便提一句，在2005年东野圭吾的小说《嫌疑人X的献身》中，一位数学老师提到了将一道几何题设计得像代数题，从而发现学生们对自己的盲点有多强的敏感度。这是一个令人着迷的提醒，提醒我们对事物只做简单解读而不对假设提出质疑是多么容易。我们倾向于用我们已然接受的方式来解读新信息。这就是计算机科学家艾伦·凯（Alan Kay）所称的相对化（relativizing）。当然，这种倾向可能是一种恶习，也可能是一种美德，取决于人们如何利用。



---

[1] 运行的每个应用程序都会生成一个或多个进程，然后由操作系统来管理。

[2] 请注意，在大多数浏览器中，每个选项卡作为单独的进程运行，这就是为什么我们可以在iMac（苹果电脑）的活动监视器或者PC（个人电脑）的任务管理器上看到同一个浏览器的多个条目的原因。

[3] 这个情况不会总是存在。

[4] 就事论事，这个情况可能存在，也可能不存在。试想这样一个任务，在网球练习赛结束之后，把球场上的所有网球捡起来。对你而言，是你走路的总距离更重要还是你弯腰的次数更重要？对某些人来说，可能是前者，在这种情况下，他们可能认为找到最短路径把所有的球都收起来是根本任务。对另外一些人来说，可能他们腰不好，那么他们会认为后者更重要，在这种情况下，他们可能会选择另外一种方法，比如把球打向球网，然后立刻把球收起来。

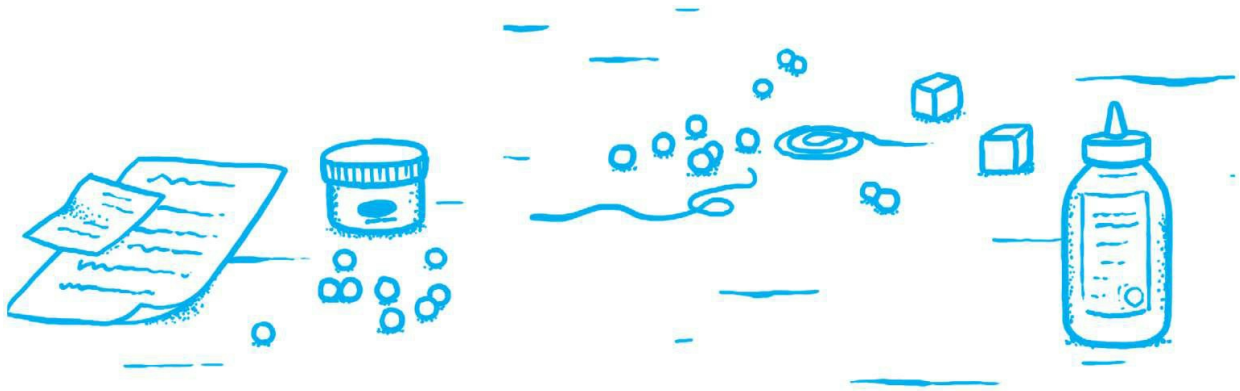
[5] 计算机鼠标的发明者道格拉斯·卡尔·恩格尔巴特（Douglas Carl Engelbart）曾经写道，我们不仅需要提高完成一组任务的效率，还要从一开始就懂得质疑那些任务是否必不可少。

第9课 重组信息 据说记忆大师都是这么记东西的

乔（Jo）是新墨西哥州工艺品市场一名独立的手工艺者，正如我们常在40号州际公路上的广告牌上所看到的，这个市场便是印第安市场。近几年，乔一直饱受类风湿性关节炎的困扰，这让她越来越难以谋生。残酷的命运之手让她不得不从事一份出售手工个性化字母珠项链的工作。乔的摊位就摆在这个市场的入口处，因此，她尽一切努力说服游客购买一条项链送给家人，让他们相信这个个性化的礼物是最佳选择。一个小女孩向着乔走过来。“我叫杰奎琳（Jacqueline），拜托了。”乔开始工作了，她将串珠穿到一条不起眼的麻线上，然后在两端粘上扣子。她把项链递给小女孩，但是小女孩的脸上充满了不悦：“不好意思，我的名字中‘Q’后面有一个‘X’，只是这个字母不发

音。那些时髦的孩子都有自己独特的名字，你不知道吗？”哦，可怜的乔。

在第1课中，我们讨论了数组可以作为一种存储方式，存储的数据集合可以以线性时间被扫描。在第3课中，我们又看到无论任务量有多大，我们都可以找到相应的方式来解决这些问题，虽然花费的时间会比较长。在本课中，我们将讨论另外一种方法，这种方法并非聚焦在扫描一个完整的集合，而是如何在一个集合中的任意节点增加或者删除某些元素。首先，我们来看一下乔修改女孩项链时可以采用的两种方式。



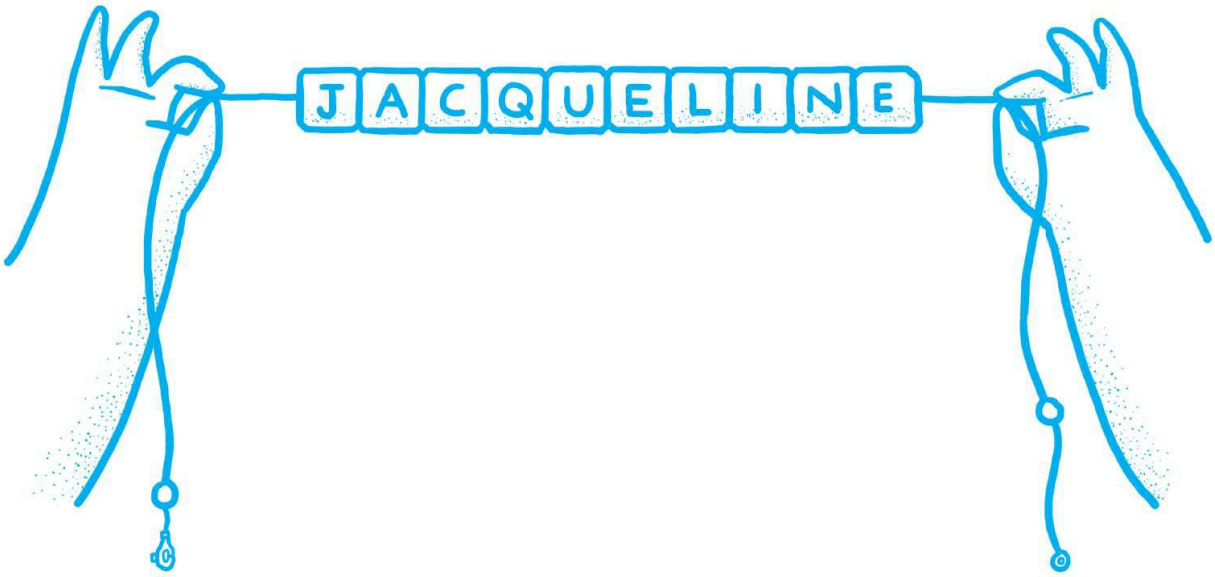
目标：在一串项链中增加遗漏的字母串珠。

方法一：拆解项链，逐个取下串珠，直到“Q”或“U”字母串珠，增加漏掉的“X”字母。然后将其他的串珠依次穿好。

方法二：在串珠“Q”和“U”之间，将项链剪断，将“X”插入绳子的任意一端，然后用胶布粘好绳子。

数组的一大缺点是它们将数据存储在连续的模块中，也就是说，彼此相邻的事物在存储器中的存储位置也是彼此相邻的。因此，如果碰巧我们需要在数组中的两个元素之间添加一些新的对象，我们就不能简单粗暴地直接添加——我们必须将该点之后的所有对象移动位置，才能为新的对象腾出空间。这就是方法一的解决方案。乔需要把串珠一个个拆下来，才

能把新的串珠插进去。可以想象这个过程花费的时间可能是正常情况下一次性穿好这个名字的两倍长。[1]

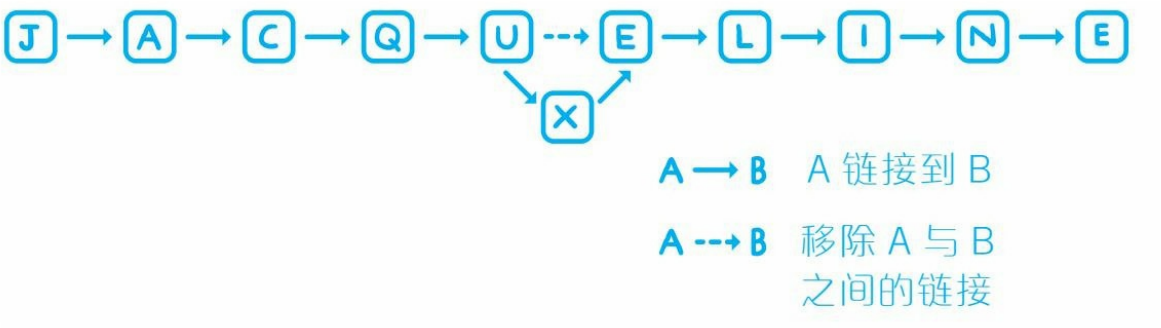


方法二的创新之处在于，它利用了这样一个事实：乔用一根绳子将串珠固定在一起，可以在任意节点切断然后用结扣或者织物胶固定。这是字符串的一个重要属性。正如我们将会在后文中看到的那样，字符串完全可以解决数组存在的问题，即添加或删除数据会花费的高昂成本。至此，我们可以注意到，如果需要

移除三个或者四个串珠的时候，方法一可能会更合适。不妨设想，如果我们需要移除大量的串珠时，方法一还那么好用吗？

在计算机科学中，恰好有一个结构可以精确地表示这个属性：

注意，我们还有一个集合，但现在我们已经不再受限于需要将数据连续存储在一起了。相反，集合中的每一个字符都只是指向后一个。这种每对字符之间的联系（或者链接），同乔的字符串问题类似。因此，如果当我们希望在某个地方增加一个项目时，就无须考虑如何腾出空间，我们只要简单地修改这些链接即可。删除其中的某一个项目也是如此。

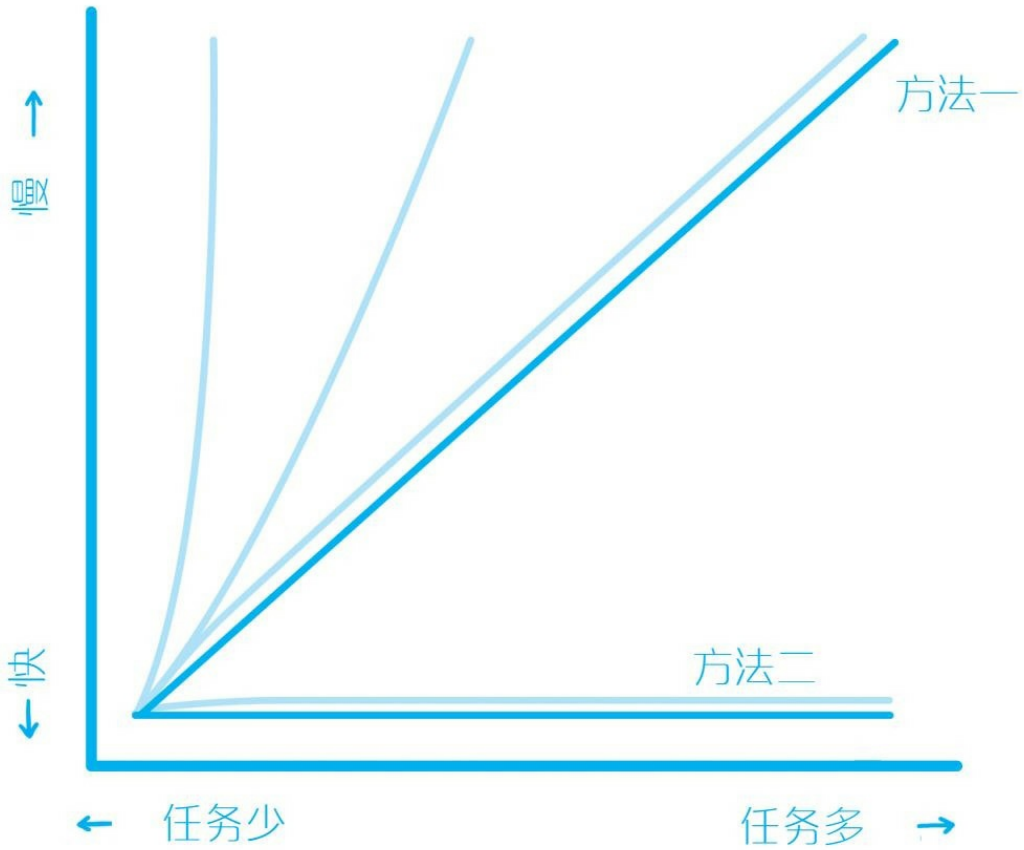


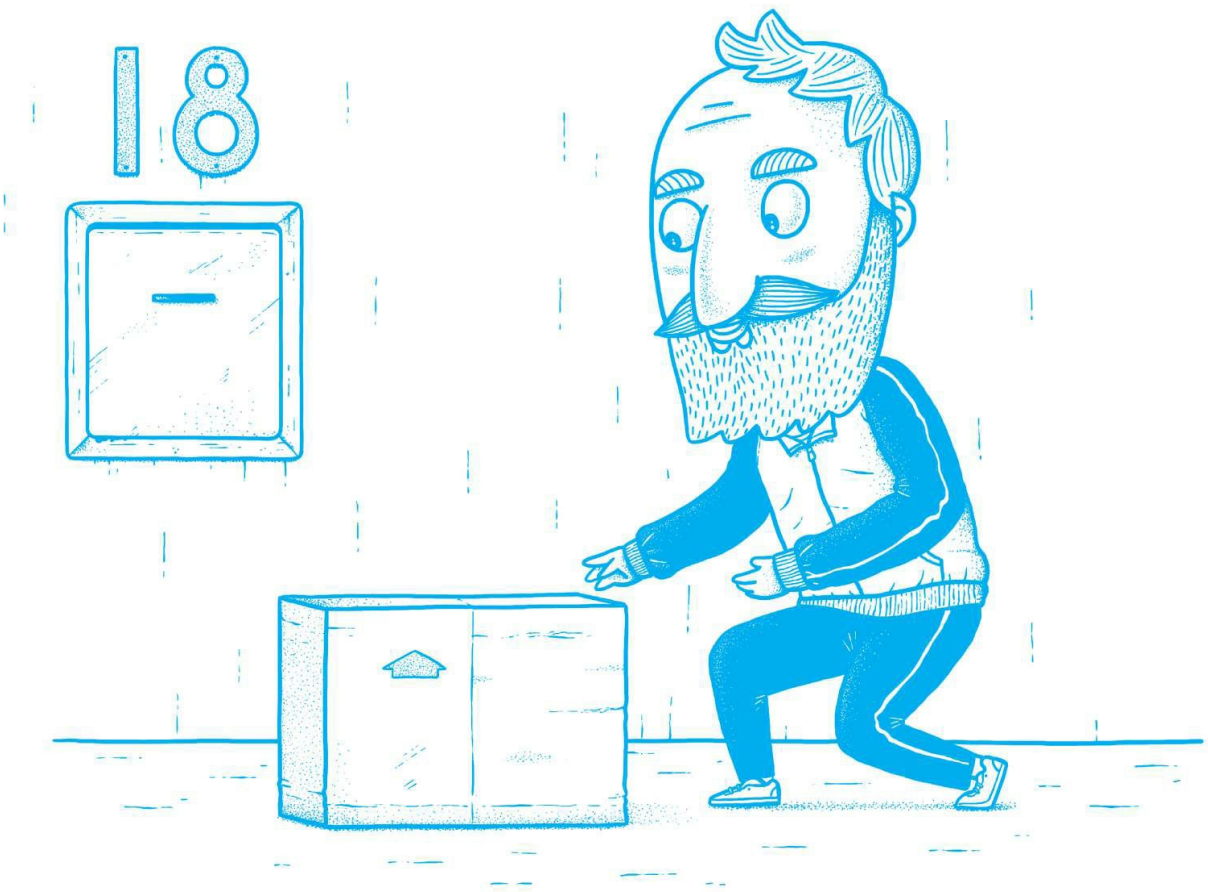
这种结构最早出现于20世纪50年代中期，人们把它称为链表（linke dlist），它是很多计算应用程序的基础，因为它可以有效地处理集合中在任意点的插入和删除操作。例如，我们在第8课中提到了打印机可能会使用队列来存储作业，并且可以决定在较大的作业之间插入较小的作业。而实现队列的一种有效的方式就是使用链表。请记住，我们关注的是基本操作，也就是在集合中添加或者删除项目的成本，这在前文已经提到过。同时，还有一些其他的操作，比如寻找我们想要添加的新项目。大多数情况下，在数组和链表中查找项目花费的时间可能是相同的。

另一种应用是文本编辑器，它可以通过在链表中存储行来表示文本文档，这样，当用户在不同的行之间移动或者添加行时，文本编辑器必须做的是修改这些行的链接，而不是物理地移动内存中的行。在这里，我们忽略的一个细节就是这些链接都是单向的。在某些情况下——正如使用文本编辑器的例子——行在事实上所带来的好处不仅仅是让我们知道了后面接的是什么，同时也让我们知道前面一行是什么。这样一来，如果我们的光标停留在某一行时，只要将光标前移一行，文本编辑器可以轻易地追踪到前一行的链接，而无须返回到链表的开头位置，再通过一个一个的节点移动实现。这种对链表的修改产生了新的结构，即双向链表结构。这个名字跟对讲机有着异曲同工之妙。

乔绝不是一个轻言放弃的人。即便如此，毫无疑问，在了解了修改项链的最快方式后，

她一定会从中受益。她用来添加串珠的方式，不由让人想起了在文字处理软件出现之前，剧本编剧是如何修改自己的剧本的——把打印出来的文字裁剪下来贴到正确的位置。乔的两种方法如下图所示：





---

[1] 用时翻倍的情况发生在遗漏的字母恰好在中间位置时。

第10课 提升效率 寄快递时怎样快速找到合适的箱子

路德维克（Ludwik）在教会街（Mission Street）开了一家出售电脑设备的商店。他住在附近一座42层公寓楼的第14层，公寓每一层的公共空间都装有闭路监控。住房租金不断上涨，路德维克需要赚更多的钱来谋生。他所居住的公寓楼每一层都有一间回收室，于是他常常在回收室里找一部分纸箱作为快递盒向国外的客户运送内存模块。今天，他遇到了一个难题，因为邮局在15分钟之后就要关门了，而他有一个订单需要在当天完成，那么他得马上找到他需要的包装盒。

目标：尽快找到空盒子，走越少的楼层越好。

方法一：一层楼一层楼地寻找空盒子。

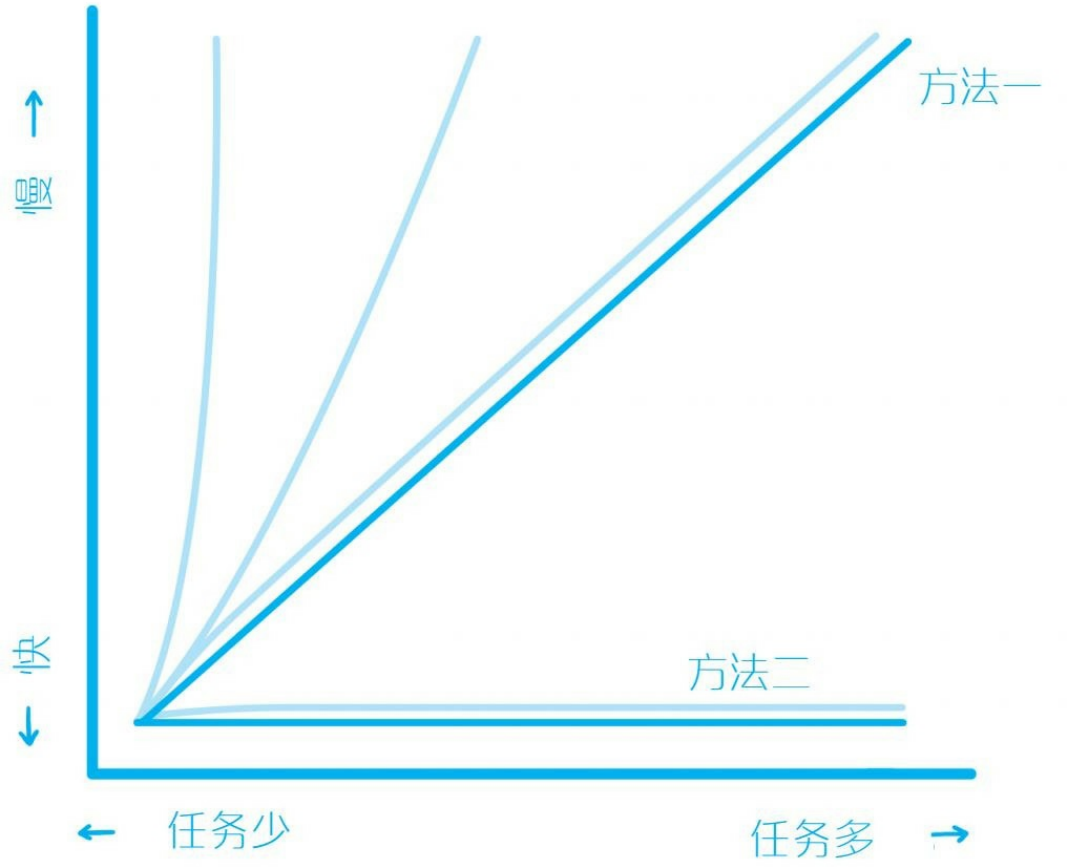
方法二：询问前台工作人员能否查到每间回收室的情况。

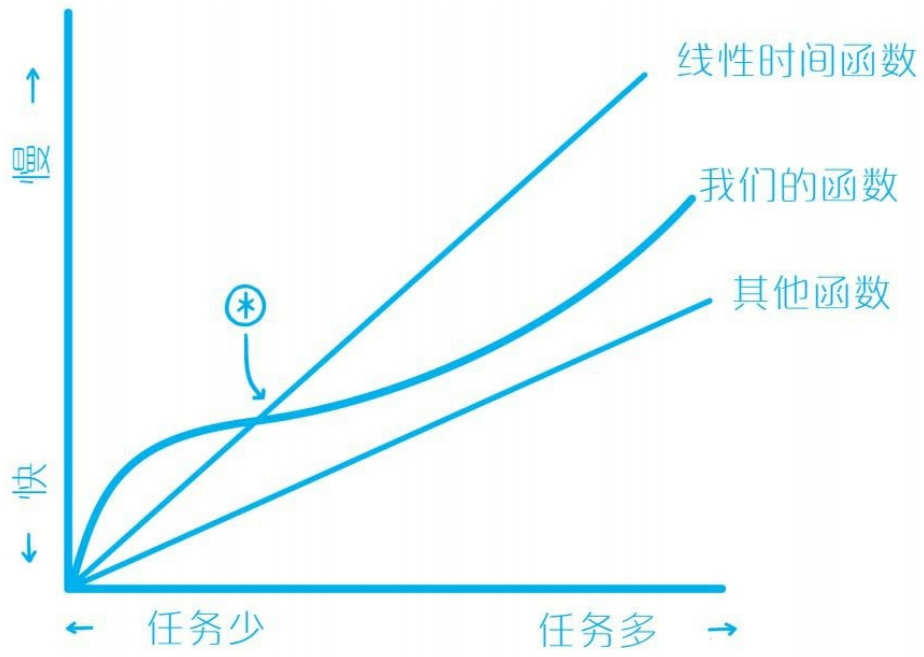
我们来描述一下路德维克如何完成在这座大楼里找到空盒子的目标。

方法一是路德维克的本能想法——从最高层开始一层一层往下找。如果有一个朋友帮忙检查单数楼层，路德维克检查双数楼层，那么尽管采用这个方法所用的时间将会减半，这种方法依然可被描述为线性时间，原因我们稍后会讨论。方法二确实是个好主意，路德维克可以通过让前台工作人员浏览监控屏幕上的实况来确定哪间回收室会有空箱子。根据他的目标，这种方法可能会让他在一个常数时间内找到空箱子，而不存在线性时间的风险，因为他只需要去一层楼即可。因此，打电话到前台就是路德维克为避免线性时间增长风险而付出的固定时间成本。两种方法的比较见下图。

对于我们正在讨论的衡量增长率的方式来说，这可能是一个很好的观点。在本书中，我们有意放弃一些严谨的方式来寻求更多洞察。即便如此，重要的是要认识到，我们可以用不同的方式来描述特定方法或函数的增长率。其中一种方式称为“big-θ表示法”，它用一组上下限表示一个算法区间——对于数量级较大的任务来说，这种方法表明我们的函数不会比一些简单的函数，比如线性函数（n）或对数函数（log n）的增长速度更快，也不会比其他的函数慢（见下图）。[1]我们的函数因此受到其他函数的约束，这就是为什么我们会认为“二分查找优于线性查找，因为在最坏的情况下它消耗的时间成本是呈现对数型的”。正如我们在第2课中看到的，二分查找（对数时间法）可以让我们在7步之内从放有100件衬衣的衣架上找到我们所需要的那一件，或者在10步左右从1 000件衬衣中找到正确的那一件。在线性查找的情

况下，解决这两个问题则需要100步和1000步。





注：超越⊛这个点之后，我们的函数总是介于线性时间函数和其他函数之间。

big-θ表示法设定了两个假设。首先，它忽略了系数。理由是随着任务数量的增加，它们的值变得无关紧要。[2]因此，对于路德维克来说，n或n/2的增长率都可以用线性时间来表示，记作θ（n），读作“n的大θ”。其次，big-θ只考虑函数中的主项，因为它假定主项对函数的输出相对于其他项来说影响更大。我们一直把主项称为基本操作。计算机科学教授马克·韦

斯（Mark Weiss）通过一个例子阐明了这一点：

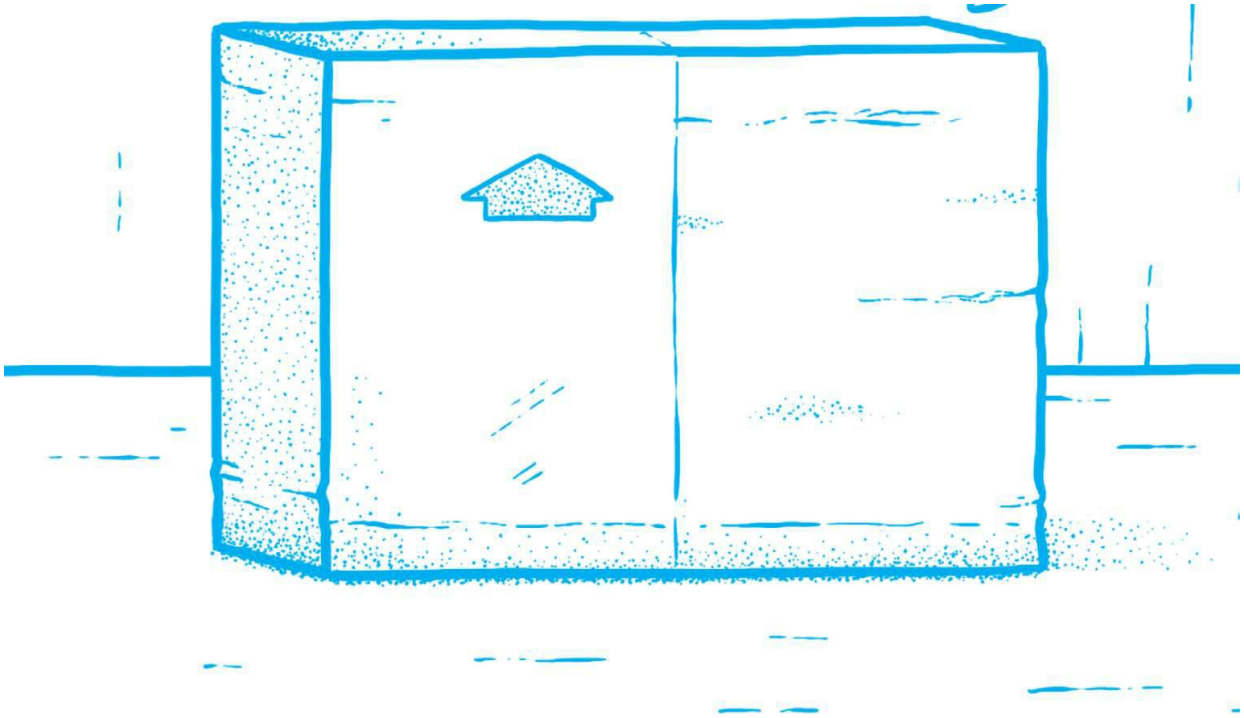
在函数$10N^3+N^2+40N+80$中，N=1 000，那么该函数的值为10 001 040 080，其中10 000 000 000是由$10N^3$这个项来决定的。

因此，如果路德维克的方法一包含了这样的情况，比如他探查了两次自己所在的楼层，那么我们可以将他在最坏的情况下实现目标所需要的时间表述为t（n）=n+1，其中“+1”表示额外到访同一楼层的次数，并写入big-θ，记作θ（n）。必须要指出的是，这个假设伴随着一系列的警告。例如，在某些情况下，非主项可能会对函数产生显著影响。回想一下在第9课中乔和串珠的故事。我们将重点放在了添加串珠上，并认为第一种方法消耗的是线性时间，第二种方法消耗的是常数时间，因此，我们认为第二种方法更具吸引力。在这个过程中，我们

把用胶水黏合串珠看作一个简单任务，但是如果黏合串珠这个动作需要5分钟呢？这会影响她的选择吗？这种常数值在我们处理少量任务时的优势非常明显，但是这也要求我们至少要注意到这个问题。

在相同的假设条件下，有两个符号可以作为big-θ的补充：（1）big-omega，它为函数提供了一个足够大的下限值n，也就是说，用这种表示法，我们的函数增长率不会低于该下限值，这是一种最好情况的度量法则；（2）big-o，它为函数提供了一个足够大的上限值n，也就是我们的函数增长率不会高于该上限值，这是一个最坏情况的度量法则。当然，在现实中，函数的增长速度可能会低于上限值，从而更具吸引力，但那是big-o，是悲观主义者的化身、索德定律（Sod's Law）[3]的体现。在本书中，当我们谈论某一种方法的增长速度时，我们通常指的是在某种严格的约束下，也就是最

坏情况下的表现。也就是说，这就是big-θ的估值。请注意，任何间接方式，正如我们在本书中所介绍的案例，都会被引入权衡。虽然我们正在这种权衡中获利，但是更重要的是，我们需要认识到现实是更加微妙的。就路德维克的难题而言，这种方法之间的现实差异是相当明显的，因此对他而言做出选择非常轻松。



---

[1] 回想一下，函数增长越慢，就越具有吸引力。

[2] 当然，情况可能并非如此。很多常量对函数的影响总是不可忽略的。

[3] 索德定律的意思是：坏运气会在最坏的情况下发生。——编者注

第11课 设定目标 这样给文档分类能节约大把时间

特里（Terry）是一名高二学生，就读于加利福尼亚州比佛利山庄一家名为梅德洛克高中（Medlock High）的私立学校。目前，他正在学校接受惩罚，因为他在社会学课堂上挑起了一个不必要的带有挑衅性的话题——“并不是所有的东西都放牛油果和羽衣甘蓝。”[1]作为惩罚，校长要求他去学校图书馆整理图书，把新买的书放在同一个书架上替换旧书，有接近250本书捆好放在地板上。特里要做的就是把这些新书按照作者姓氏字母顺序摆放到书架上。特里晚上约了几个朋友看电影，所以他迫切希望可以在此之前把这些书摆放好。虽然这不是个大工程，但是他却不想浪费一点儿时间。

目标：根据作者姓氏字母排序把书籍摆放

在书架上。

方法一：选择其中一本书放在书架上，然后再拿起另外一本，根据第一本的字母顺序决定第二本的顺序，以此类推。

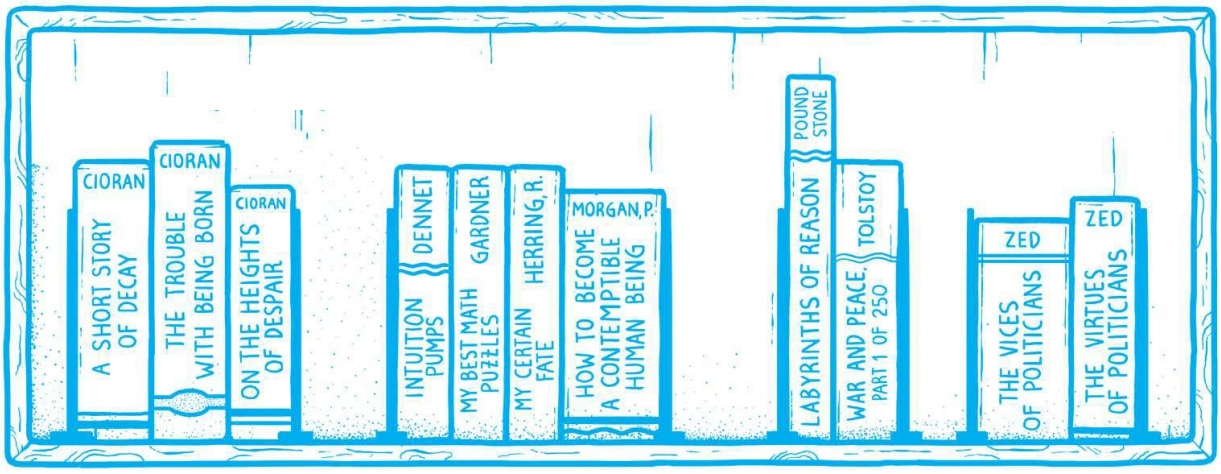
方法二：利用书挡为每个字母创建独立空间，然后将每本书放在对应的空间，根据需要调整书挡。

让我们从特里如何执行他的任务开始。

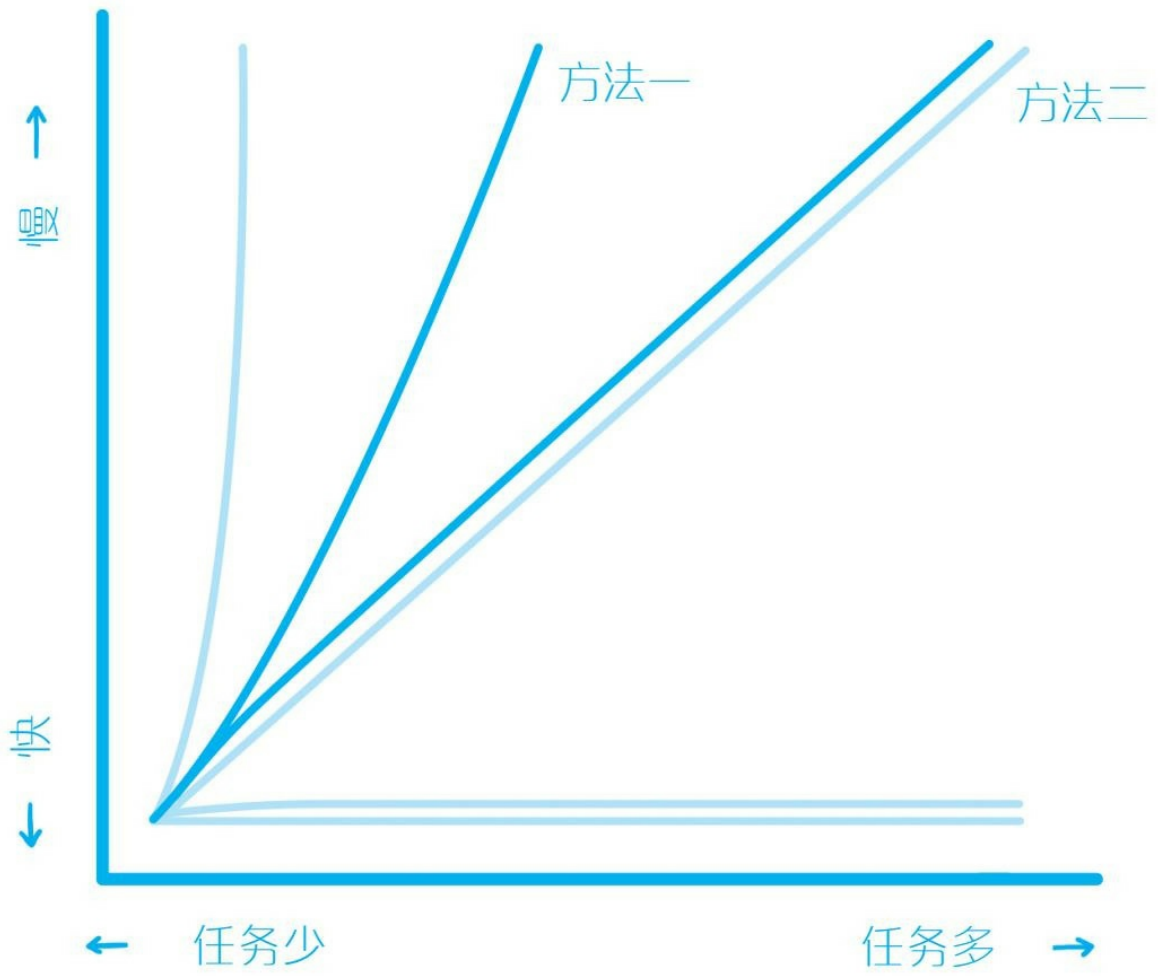
我们在第5课了解了如何通过比较相邻项来决定大小，并取平方时间来排序。我们认为这种方法的实践案例包括插入排序、选择排序和冒泡排序，所有这些操作都有细微差别。本课的方法一事实上跟第5课中提到的方法一完全相同。

有趣的是，特里的改进方法跟查理采用的分而治之的方法不同，这种方法改变了原来的平方时间法，通过一个简单的创新即可实现。如果我们采用诸如插入排序之类的方法，在正确的位置插入项是最耗时的步骤，因为每插入一项，我们就需要把后面的所有项全部顺次移动。本课的方法二，则是通过事先在书架上均匀地布点来区隔空间。这样，当特里找到一本书的正确位置时，他只需要移动几本书就可以了。这个方法，其实是通过空间换时间，当他把每个空间预留得越大或者空间数量越多时，

每次移动的书本数量就越少。这种对插入排序的改进，将其从平方时间变为线性时间，让每次排序都有成功的“高概率”，我们将这种方法称为图书馆排序或缺口插入排序。回顾第10课中提到的解法，将在书架上设置书挡作为初始设置成本，当需要排序的书本数量很少时，这种方法的耗时成本就比较高，而随着书本数量的增加，这种方法就会更高效。



这两种方法可以用下图进行呈现：



如果我们考虑另外一种更具约束性的场景，或许可以更好地理解这种方法的价值所在。试想下面一个极端的场景：科技行业没落了，整个世界对科技的发展开始厌烦。因此，成千上万的贫困科技工作者到世界上某一个地方寻求避难，在那里，这些天赋极高的破产男女可以获得免费比萨和调味品。人才的涌入使

得各个大学欣喜若狂。每个人都很兴奋，除了从梅德洛克高中毕业的收发室工作人员。是的，这就是15年后的特里的境遇。他在管理一系列壁柜，这些壁柜按照字母顺序标注该部门所有研究生的名字。每当一名新学生入校，他就需要把这个人的名字加到墙上去，有可能需要替换现有的人名，然后依次移动其他的标签（见下图）。

如今，移动这些壁柜标签的频率急剧上升，特里想起了15年前所采用的方法。他意识到当时用来整理图书馆书架的方法可以极大地减轻他的工作压力，但是这种方法却需要一些额外的空间。因此他在已占用的壁柜之间腾空了几个，起到了书挡的作用。[2]

前面提到的“高概率”为我们带来了一个重要话题，也是本节课的一个重要内容和收获。迄今为止，我们提到了很多解决“最坏情况”和“一般情况”的不同方法。这些限定符本质上是一些指标，可以让我们了解需要花费多长时间才能完成特定的所有输入。采用一种方法实际需要多长时间才能完成任务取决于很多因素。例如，在代码中，我们可能有些逻辑分支（例如，如果发生这种情况则执行，否则跳过），并且根据特定运行期间遵循的路径，运行时间可能会有所不同。输入如何影响运行时间的一个很好案例是快速排序，我们曾在第5课中提到过。快速排序通过一次排序将要排序的数据分割成独立的两部分，其中一个部分的所有数据比另外一部分的所有数据都小，然后再按照此方法对这两部分数据分别进行快速排序，整个排序过程可以递归进行，以此达到整个数据变成序列的目的。通过随机选择一个项

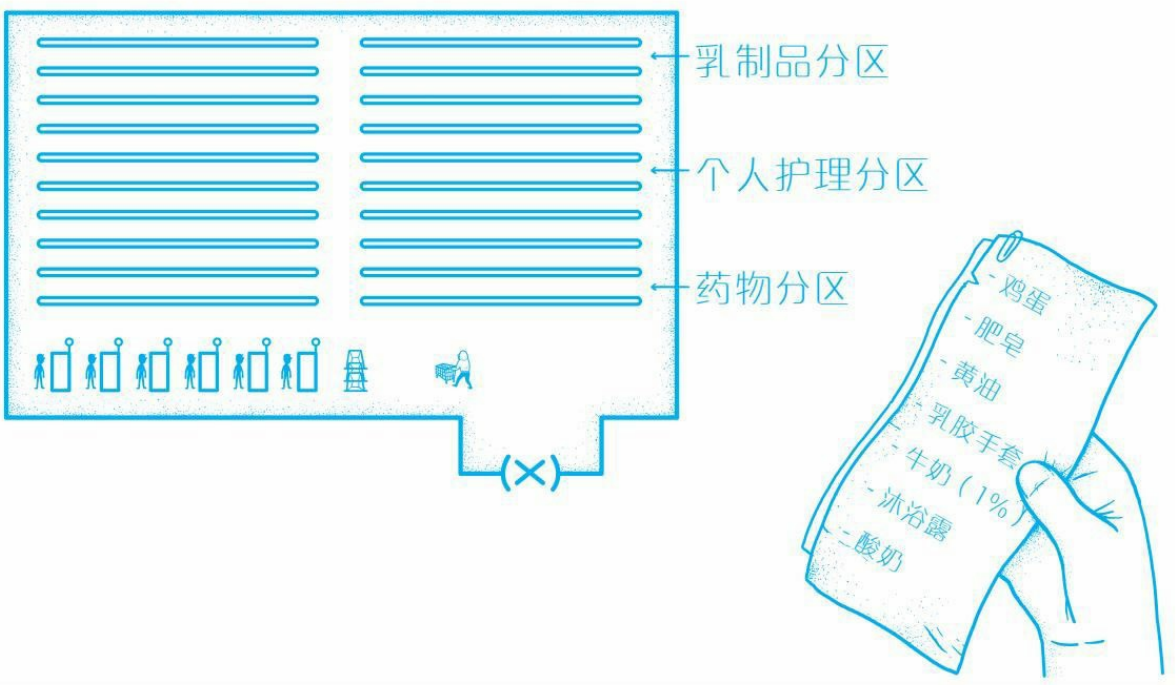
作为枢轴，快速排序在一般情况下可以按照线性时间运行。如果枢轴过大或者过小，快速排序会在最坏的情况下按照平方时间运行。这种表现上的戏剧性转变是因为当枢轴在每一步没有很好地把项目集合数量减半时，我们最终不得不检查每一步中的每一项，正如我们在前面几课中提到的，这是一个典型的平方时间方法。

在快速排序的情况下，分析让我们相信，该算法按线性时间运行的可能性很高。我们知道需要做些什么来实现这一目标——保证枢轴永远不是最低值或者最高值——因此，对于所有实际目的来说，通常情况下，快速排序都将在线性时间内运行。

有时，概率保证还不够。如果碰巧我们的方法影响了关键任务程序，例如控制航天飞机或者生命攸关的应用（如调节输液泵或除颤

器），那么一种方法在最坏情况下的指标可能就是我们最关心的。即使对于日常生活方面的应用，这种对潜在结果的分类也是有效的。

对特里来说，尽管事实上他的方法在最坏的情况下以平方时间运行，但一般情况下线性时间的回报更大。现在他没有什么好担心的。只要他能够找到足够多的书挡，或者自己造一些出来，那么他一定会准时赶到电影院的。



---

[1] 有些人的胆量是不可估量的。

[2] 我用的第一种编程语言是BASIC。在最近与一位朋友的一次谈话中，我忽然想起了一件快被遗忘的事情——编程语言中的行数之间通常都有一些“间隙”。第二行不是跟着第一行的，而第二十行可能跟着第十行，以此类推。这种约定可以允许程序员在现有的行之间添加新行，同时避免了必须重新编号的烦琐工作。

第12课 运用逻辑 生活中的每个场景都能找到逻辑

沃兹玛·莫奈（Wurzma Monet）一直是一名对冲基金经理，但他渴望自己能够成为一名说唱歌手，这个愿望是他最近在给一群10岁的学生进行的一场演讲中萌发的。这个对话的主题是“我爸爸的日常生活”。由此，他联想到了自己单调的生活。他下定决心，要过一种更有趣的生活。

沃兹玛每两周去一次当地的超市，他发现自己每次都要在货架间来来回回很多次，以寻找购物清单上的商品，这让他觉得购物就是一件没完没了的事情，浪费了他大量的时间。更糟糕的是，他走路的方式有些笨拙，不是那种“黑帮走”（gangsta nalk）的感觉，而有一种“又要错过跟理疗师的预约”的焦虑。其他说

唱歌手开始注意到这一点，沃兹玛脆弱的“街头信用”随之崩溃。他要想办法阻止自己变得像一个业余选手而被嘲笑终生。

这对沃兹玛来说是一个重要的转折点。好消息是，我们可能已经知道如何来帮助他了，因此在本节课中，我们将详细地阐述前面几节课提到的一些概念。首先，沃兹玛可以用两种方法来完成购物清单上的采购工作。

目标：把在超市中行走的区域最小化。

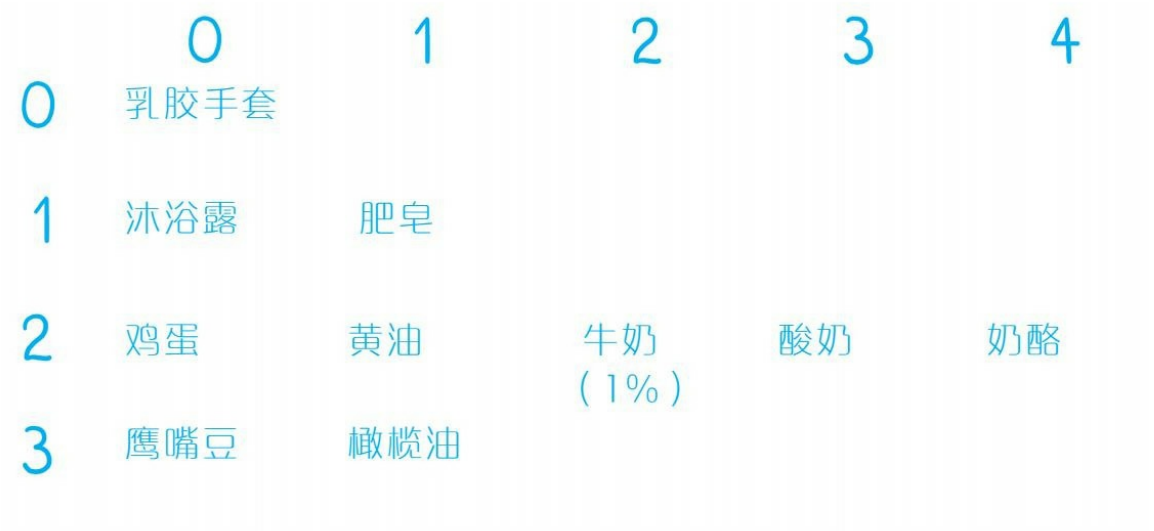
方法一：按照购物清单上的物品数量一一寻找商品。

方法二：提前准备好购物清单，按照商品的种类进行分组，在购物过程中按照商品分类进行采购。

看，我在你的数组里置入了另一个数组，

这样你就不会误入歧途了。

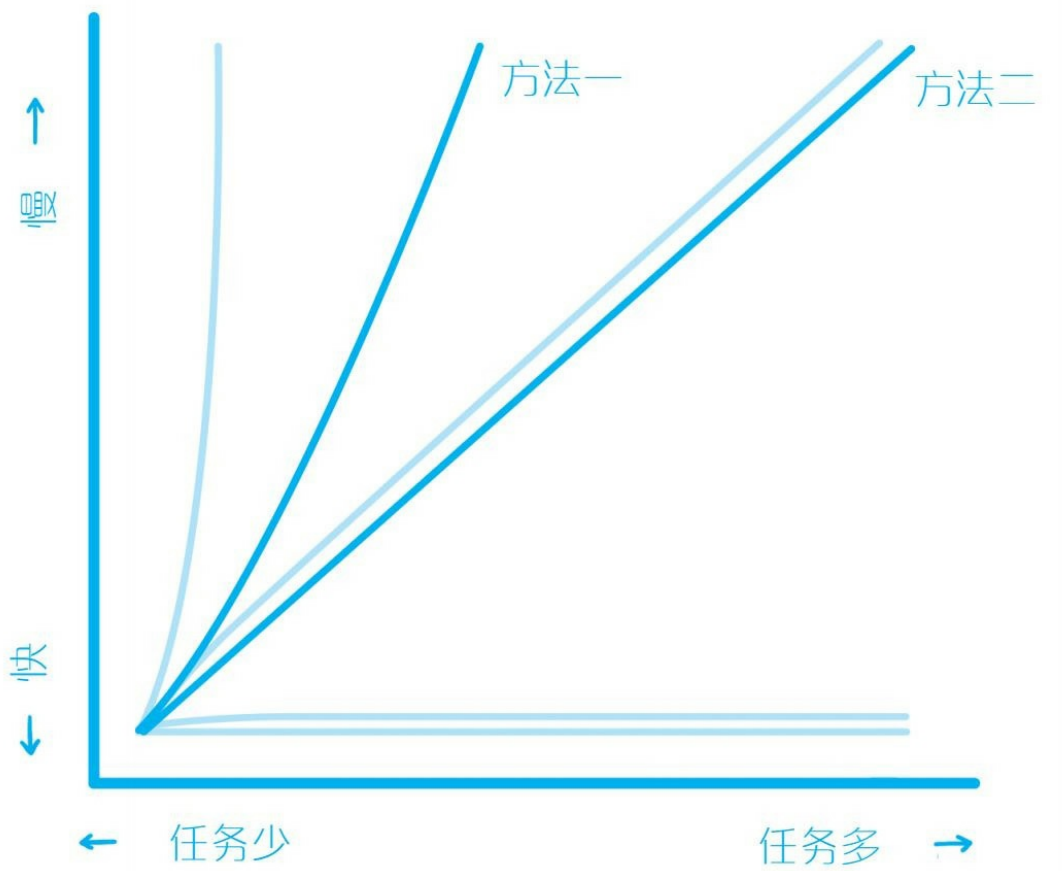
到目前为止，我们讨论过了数组作为存储项目集合的基本类型结构。在第6课中，我们介绍了另一种被称为矩阵的有效结构，同样是存储项目集合的结构，但是与数组不同，数组的存储形态是一维存储，而矩阵的存储是二维的。事实上，数组可以转化为矩阵，只要将存储的文本（数字、字母、单词）替换为数组即可。一个包含数组的数组便是二维数组，也称多维数组（multi-dimensional array），如下图所示。



在沃兹玛的故事中，多维数组允许我们在二维空间上存储他的购物清单，从而允许他基于超市的分类遍历每一个子数组。因此，他的购物清单就不再是一个简单的购物清单，而是一个类目列表，而每一个类目又是一个商品清单。[1]由于超市的商品通常会按照类别陈列，所以沃兹玛可以直接走到目标货架处，例如个人护理类商品，他就可以直接走到“个人护理”货架然后从上面找到所需要的商品，以此类推。这便是方法二的原理。如果沃兹玛只是采用了一个列表，即一个数组，正如方法一提到的，那么他最终可能会浪费13分钟在货架过道之间穿梭，因为在最坏的情况下，沃兹玛寻找每一个商品都需要走遍超市的所有货架过道。如果n代表货架过道的数量，m是购物清单上的商品数量，用算式来表示就是：n/2x m=（n x m/2）。当沃兹玛的购物清单上有20件商品，超市有40个过道时，他需要20次经过20条过

道。假设每经过一条过道需要2秒钟，那么整体就需要13分钟。鉴于现在他穿梭在过道间的时间不到1分钟，由于他每次可以选择不止一条过道，在最坏的情况下，沃兹玛需要走过所有的过道，也就是n/2。所以按照20件货品和40条过道来算，沃兹玛从一条过道到另一条过道所用的时间还不到1分钟。

请注意，虽然方法一并不像我们在第5课和第11课中提到的二次排序算法那样，但是遵循的蓝图却是相同的，这是因为在最坏的情况下，这种方法可能会导致沃兹玛走遍超市所有的过道才能完成采购。下图是两种方法的比较。



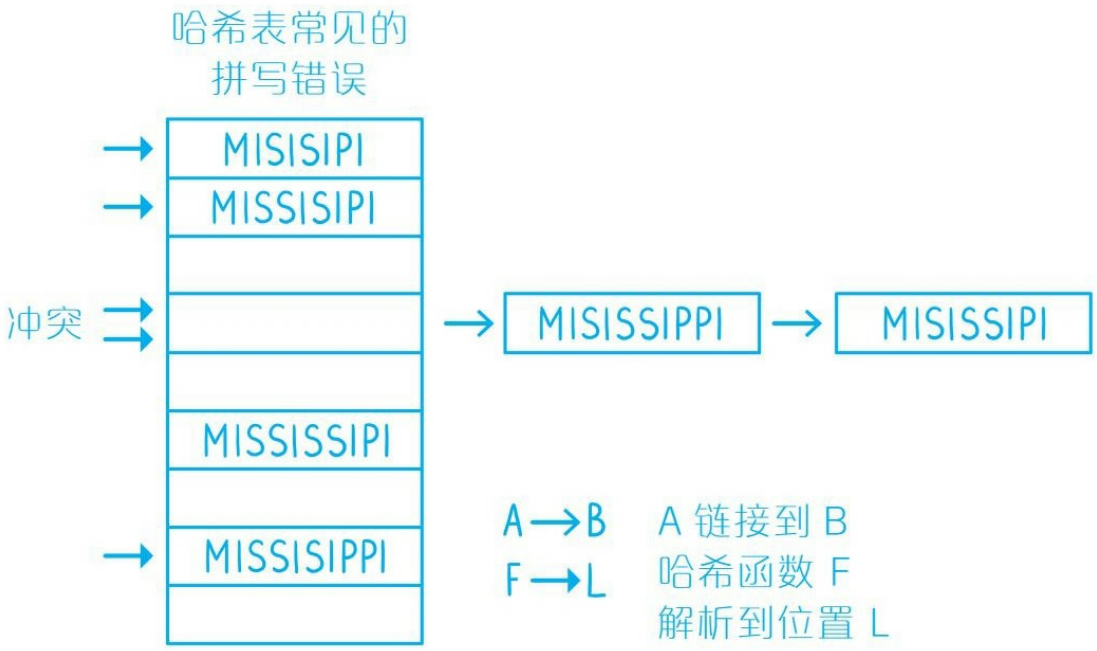
碰撞，碰撞，碰撞。如果你常这么做，那么你就一定会后悔，就像村上春树故事里的人物一样。

在第1课中，我们讨论了哈希表，以及我们在忽略顺序的快速查找中它可以起到的作用。这里关于多维数组的探讨就是我们拓展对哈希表认识的一个重要原因。我们之前讨论过，哈

希函数是将一个项目映射到哈希表的某个特殊位置，这就是这种方法可以保证常数时间查找的原因。然而事实上，哈希函数可能会遇到冲突，在哈希表中解析的位置已经被其他项目所占用。这可能因为哈希函数并不完美，也就是说它不能很好地均匀分配散列值，或者因为我们要存储的项比哈希表空间的存储项目要多。被占用的哈希表的比例被称为负载因子（loadfactor），当哈希表为空时，负载因子为零，当哈希表为满时，负载因子为1。

在这种情况下，我们可以用于解决冲突的方法之一，便是链接。通过链接，我们将拥有一个包含项目集合的哈希表，而非一个包含多个项目的哈希表。这样，当冲突产生时，冲突项会被推到集合的末端，并且不会有数据被意外覆盖。因此，我们最终得到的哈希表，是一个集合的集合。每当哈希函数解析到具有多个项的位置时，我们就必须遍历这些项，直到找

到我们需要的项。当然，这一切对用户来说都是完全透明的。



对下一步感到迷茫？别焦虑，让我告诉你该往哪里走，快点动起来。

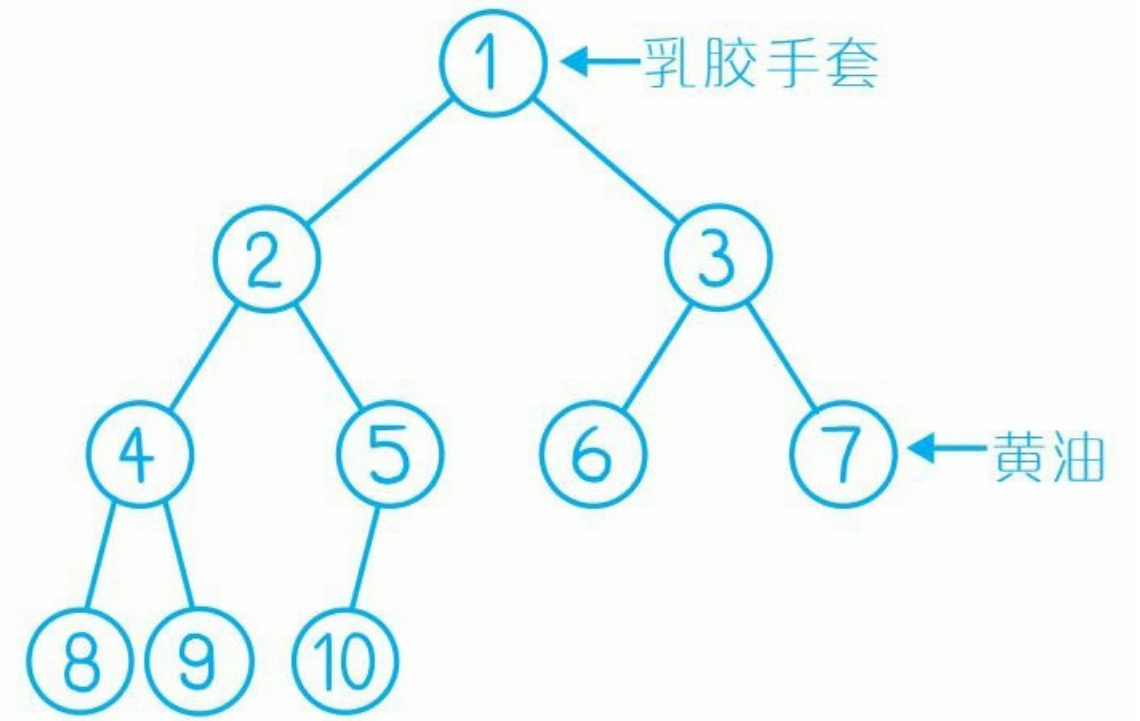
还有一点需要指出的是，之前提到的多维数组，事实上，我们已经考虑了优先级，即从商店入口到物品所在区域的距离。也许，这可以帮助我们更好地理解在第8课中提到的优先级队列。在第8课中，我们提到了一种情况，如果

你草草地列出了项目的先后次序，那么当需要添加一个新的项目进去时，你就得清除一部分项目以腾出足够的空间。很快，事情没准儿还会变得一团糟，你很有可能不得不写一个新的列表。那么一台机器如何才能管理这样的一个列表以满足我们对效率的期许呢？我们已经了解了针对扫描（数组）和在任意点（链接）进行插入操作的优化结构，现在我们将展开谈一下优先级队列（priority queue）。优先级队列是针对对数时间集合中的高优先级项目[2]进行优化的解决方案。它被称为队列，却并非我们通常理解的队列，通常情况下，队列是“先到先得”，[3]而优先级队列却不是。你可以把它想象成一组植物的种子，每次只会发一个芽供路人采摘。

当我们选出最高优先级的任务时，树就会自动重新平衡，弹出次高优先级的任务，以此类推。这种特殊的描述优先级队列的方式被称

为堆（heap）。我们无法用类比的方法来真正解释堆，但这个结构着实迷人，而且值得我们花一点时间来理解它是如何在对数时间内重新平衡自身，并准确地调整优先级以及如何确保项的插入也发生在对数时间内的。

让我们先来看一下如何用堆来表示沃兹玛的优先级列表（见下图）。



1. 乳胶手套

2. 沐浴露

3. 肥皂

4. 鹰嘴豆

5. 橄榄油

6. 鸡蛋

7. 黄油

8. 酸奶

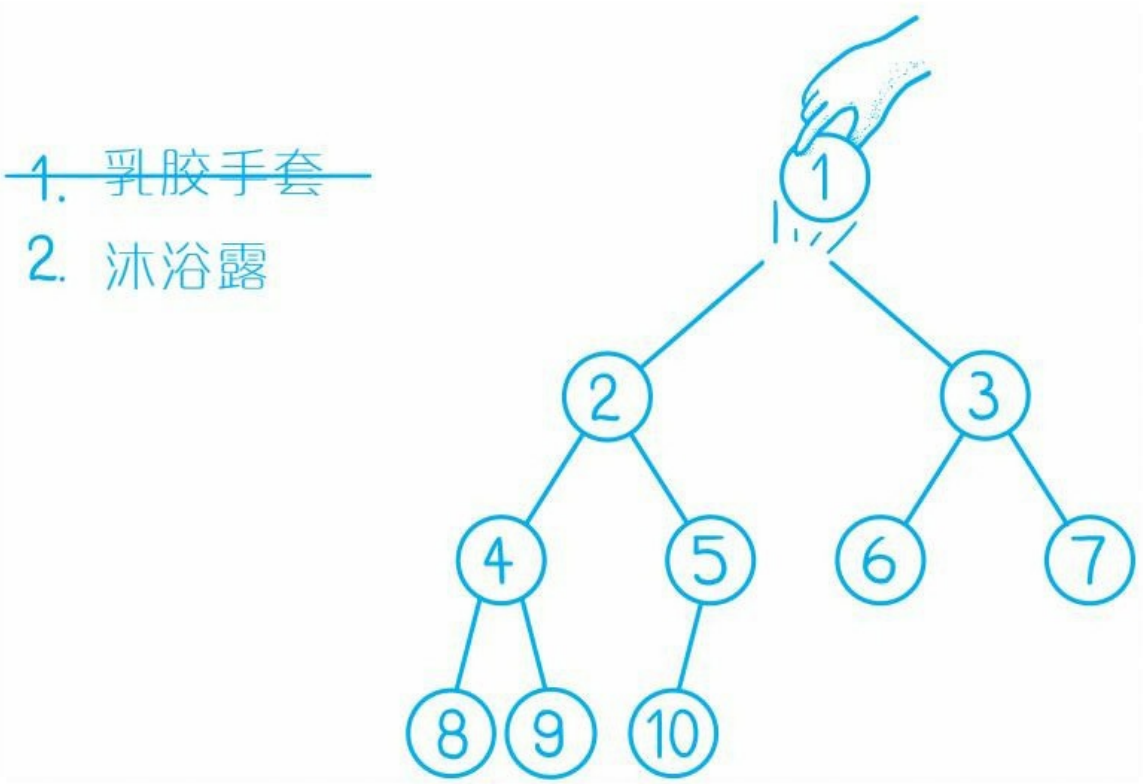
9. 奶酪

10. 牛奶（1%）

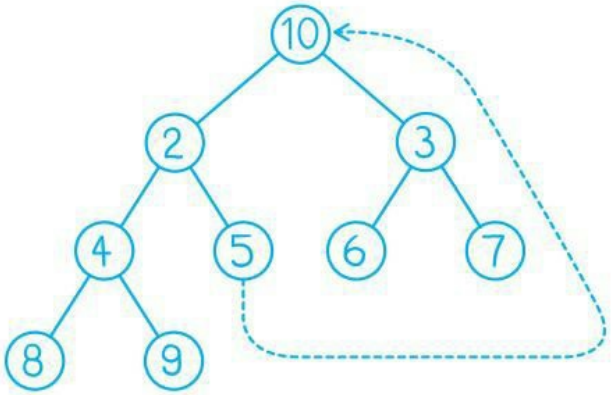
注意，事实上，堆便是节点树，并具有两个属性。第一个属性是每个子节点的优先级要低于父节点的优先级。[4]“这也就是为什么在商场入口处的商品优先级最高，处在最高节点。那么其他节点的顺序就很难推断了，比如

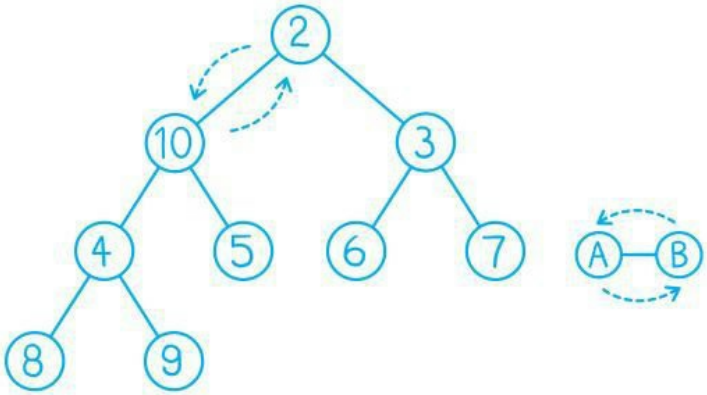
兄弟节点，其优先级处于同一水平。还有一些其他结构可以确保树状结构中的所有节点都是有先后顺序的，例如二叉搜索树（binary search tree），这对于解决第2课中提出的问题非常有效。第二个属性是每个节点都有两个子节点，其中的例外就是最低级的节点没有子节点。这种结构保证了堆的高度，也就是说，即使是最长的路径，也不会比log n更深，而n便是堆中的项目数。

一个情况是，当沃兹玛找到了离商店入口最近的橡胶手套后，就需要确定接下来要找的是什么商品了。

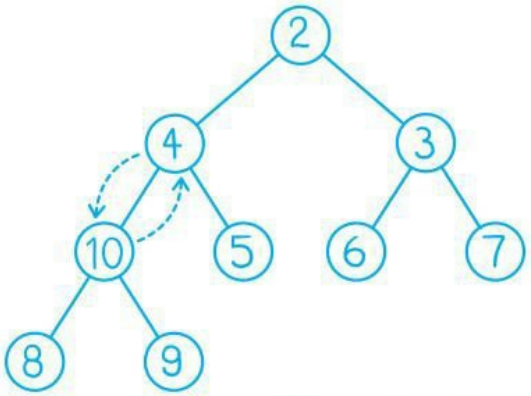


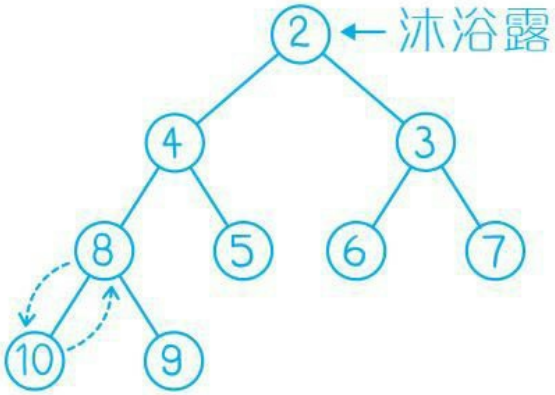
一旦我们删除了根节点，堆的第一个属性就被破坏了，因此会重新调用它的平衡算法，这个算法涉及用堆中的最后一个节点替换空的根节点，然后检查新的根节点与最小的子节点，来判断是否需要替换。检查和替换是在父节点和其最小节点之间进行的，一直到堆的最后一个节点。请注意，重新平衡堆的这个过程需要对数时间[5]来完成，结果就是下一个最接近的项出现在顶部。



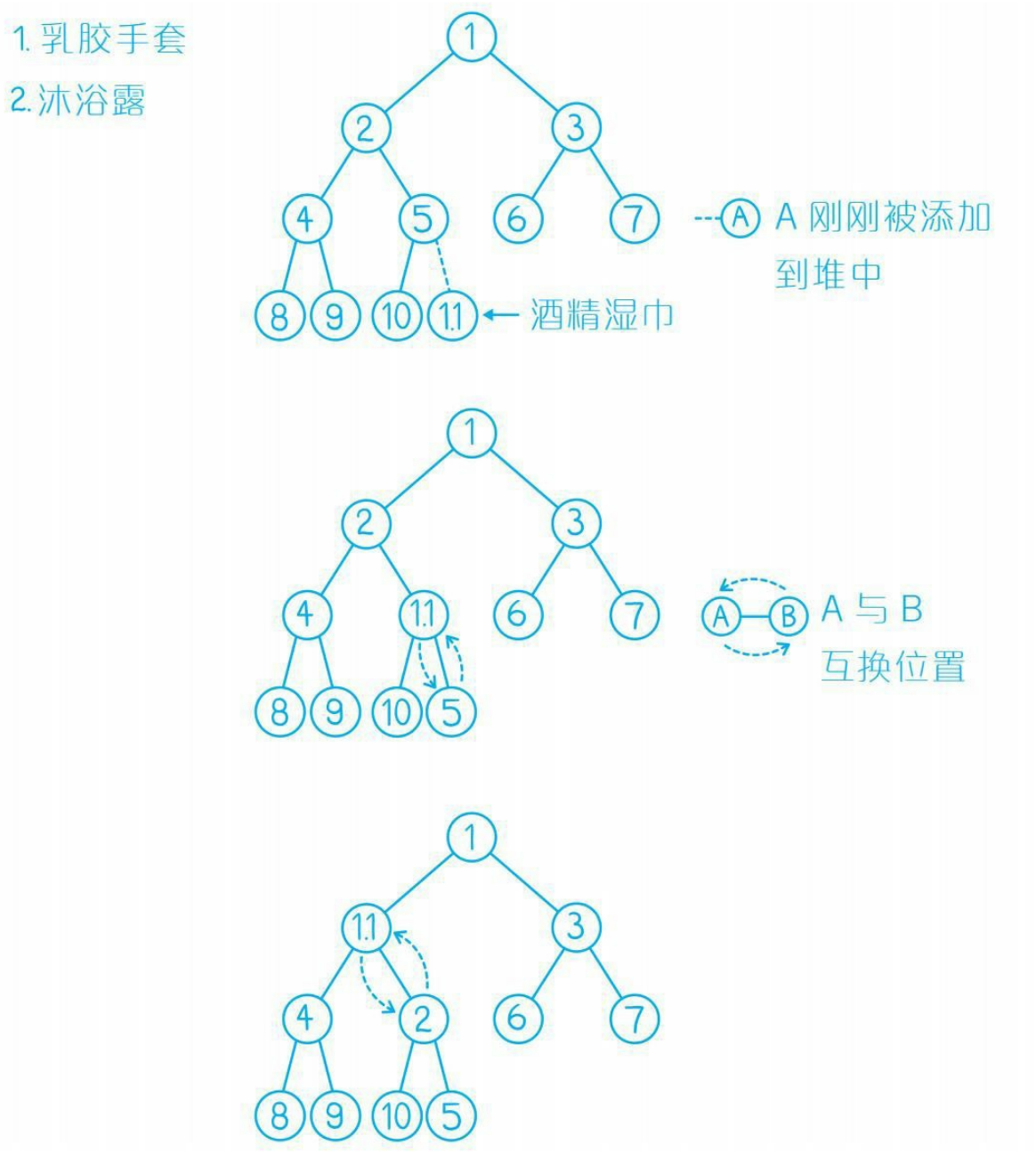


A 与 B
互换位置





同样，如果我们要将一个新的优先项插入列表中，那么超过log n数量级的移动，则不需要删除和移动其他的项目。我们将新项目添加到堆的末端，如果父节点大于新项目，则将其与父节点交换；如果有必要，我们就继续交换，直到根节点。



我们在本书的最后几页讨论了计算机可能会在解决问题时用到一种优雅的方式。在书的最后一节课中提到这种方法，是为了强调这样

一个事实：算法，就如同科学中的方程式一样，拥有属于艺术学科的所有属性——美、优雅和高贵。如果你碰巧在算法的更高级领域探索其扮演的重要角色，那么不仅要注意到这些算法能带来的结果和属性，也需要了解它们如何建模。不管怎样，冒险进入这些领域，你可能会发现自己置身一个人工智能的世界里，在这里，算法的实际应用包括了医院快速诊断、挽救病人生命，或者帮助研究人员更好地理解人类基因组的复杂性等问题。你也可能会发现自己置身一个博弈论的世界里，在这里，算法的应用包括了为共享汽车公司提供决策，例如如何在既定时间匹配合适的乘客。你还可能会发现自己置身一个计算机视觉的世界中，思考如何让自动驾驶汽车变得足够靠谱以成为普及的交通工具；或者说在图像处理的世界里，有更高层次的思考，不是依赖简单的变换，如亮度和对比度，而是依赖先进的算法进行权衡，

例如如何更好地模糊、锐化、镜头校正或色彩校正。算法的应用场景真的是十分广泛。

至于沃兹玛，他的先锋派说唱生涯即将迎来兴旺发展期。没有人真正想过用说唱来表现数组、哈希表和优先级队列。那到底有多酷？他还没有告诉他的儿子，他计划下周在这个可怜的男孩11岁生日的派对上举办一场说唱大赛。这样做没有任何问题。更重要的是，沃兹玛购物速度的加快，将会让他在说唱时变得更加自信，这对他大有裨益。

不再玩游戏，不再开玩笑，我浴火重生，涅槃而出

从灰烬中回归，像比尔·希克斯一样自由

你无法相信我有多么快速，从果酱到舒洁纸巾

面包、牛奶和蜂蜜，有机糖，然后是布朗尼

我看到你在盯着我，眼里写满了惊讶和惊叹，

为我神魂颠倒

然而你却羞于承认

没关系

你看，你是批判家

而我是迈克尔·贝的男主角

---

[1] 您能想到的任何有趣的数据集都可能类似多维数组，鉴于它存在多个列，统计学家就会利用这些列对行进行多重分析（观察）。

[2] 实际上，它表示最小（或最大）项，

但在这个案例中，我们使用该属性表示最高优先级项。

[3] 尽管如此，满足这种先进先出（FIFO）特性的结构确实存在，甚至在日常生活中也有类似的事物。想想杂货店把易腐坏的物品放在货架上的方式：旧的放在前面，新的放在后面。

[4] 除了最上面的节点，根结点是没有父节点的。

[5] 回想一下，n=10，$\log_2 10 \approx 3$。

结语 常规方法之外，还有另一种解决问题的方法

每当有人问费曼（Feynman）他是如何培养出自己如此迅速解决问题的能力时，他总是说，那是因为他用多种方式对这些问题进行模拟，也就是说，他会从不同的角度来看这些问题。类比是研究问题的一种方法，方程式是一种方法，同样，辩证法也是一种方法。对费曼而言，一个让观众大吃一惊的虚构故事，同样是一种方法。

在艺术领域，也有一种类似的方法类型，即从不同视角绘制场景而不损失细节，这种能力被认为是一种美德。有趣的是，这是文艺复兴时期布鲁内莱斯基（Brunelleschi）做出的创新并使这一方法成为可能。通过设计一种叫编纂线性透视法的数学方法，布鲁内莱斯基得以

精准地绘制出佛罗伦萨圣约翰洗礼堂（Baptistery of Saint John）的图纸。近处的物体在视觉上显得更大，远处的物体则看起来更小，洗礼堂的瓷砖线条以符合现实情况的方式汇聚。那些著名的壁画，例如彼得·佩鲁吉娜（Pietro Perugino）的作品《基督将钥匙交给彼得》（*Delivery of the Keys*）以及拉斐尔（Raphael）的作品《雅典学院》（*School of Athens*）都是这种方法的优秀代表作品。

我们首先探讨了聚焦相对量级和日常任务如何帮助我们把复杂的主题变得更加相关。在某种程度上看，它们有助于在我们的脑海中创造触发器，提醒我们不断思考各种选择。下一次当你看到有一堆需要分类的东西时，也许你会想，“嗯，或好或坏，或快或慢，或平方或线性”。抑或相反，比如你的儿子、女儿、侄女可能会问你二分查找是什么，你会想，“嗯，自由，威廉·华莱士（William Wallace），艾培·托

马（Eppy Toam），架子上的衬衫”。这种联想很容易回忆起来，也很有趣。通过了解特定任务的选择目录，可以在很大程度上帮助你辨别好的和坏的选择。

如今，“算法”已经成为很多人挂在嘴边的词，就像几年前的“大数据”、在未来数年里的“深度学习”一样。笔者希望读者能从本书中了解“算法”这个概念，而不是作为潮流去追赶。正如我们在早期讨论古巴比伦石碑一样，它将深深扎根于历史。因为它是一个永恒的概念，值得探讨，并不断被充实，最重要的是如何将其作为一种工具应用于智慧思维。

名词解释

在本书中，最核心的主旨之一便是对比不同的方式来完成相同的任务。在每一节课中，我们都运用了增长率图示来进行比较。有些图标中的线条有意未做标记。在这里我们对这些增长率的概念做一下总结，从最慢（最好）到最快（最坏）进行排序。

常数时间（constant time）：给定多个项目，如果我们将项目数量加倍，任务完成所需时间保持不变。

对数时间（logarithmic time）：当项目数量足够庞大时，如果我们把项目的数量加倍，任务完成所需的时间大约按对数量级增加。

线性时间（linear time）：当项目数量足够庞大时，如果我们将项目的数量加倍，任务完

成所需的时间将近乎翻倍。

线性对数时间（line arithmic time）：当项目数量足够庞大时，如果我们将项目的数量加倍，任务完成所需的时间将近乎翻倍并按对数量级增加。

平方时间（quadratic time）：当项目数量足够庞大时，如果我们将项目的数量加倍，任务完成所需的时间将大约是原先的四倍。

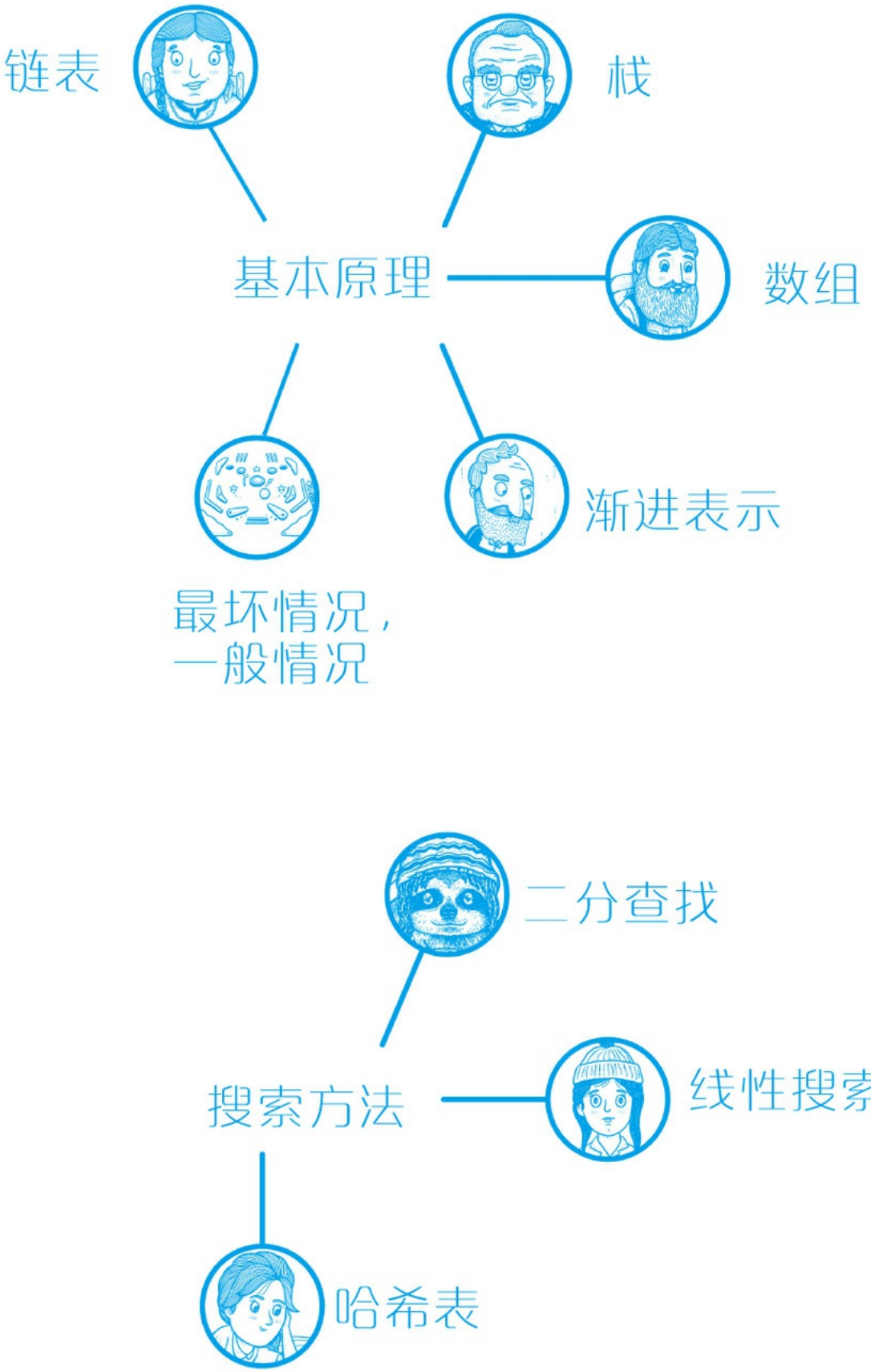
指数时间（exponential time）：当项目数量足够庞大时，如果我们每次增加一个项目，那么完成任务所需的时间需要翻倍。本书中，每个图示最左边淡出的线代表的就是指数时间。

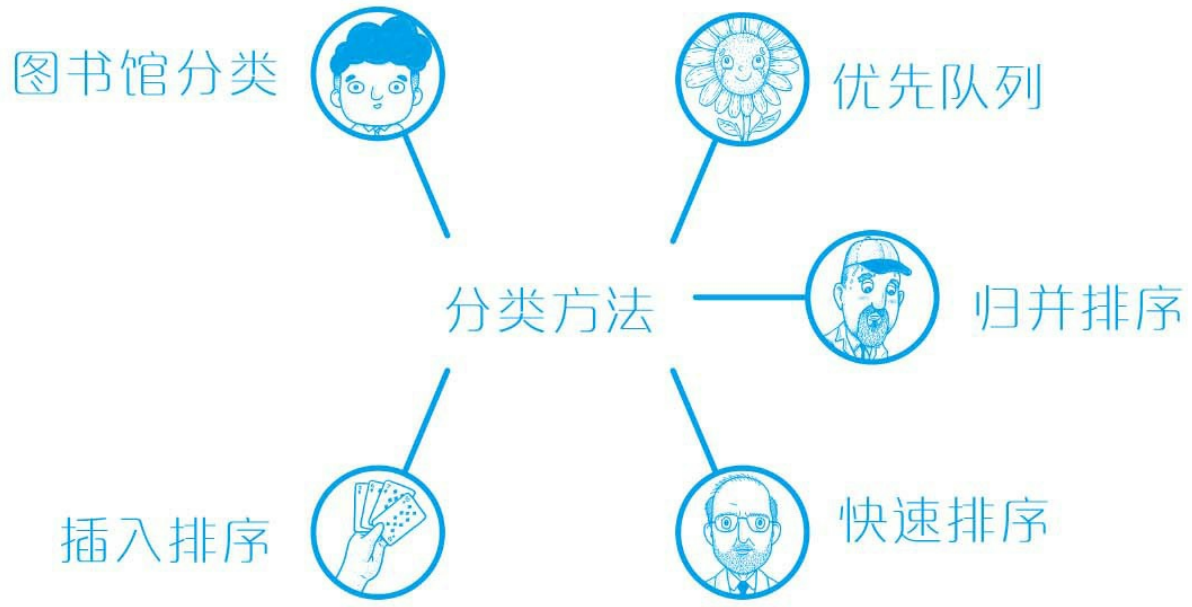
鸣谢

感谢每一位关注本书的朋友，因为你们让本书变得更加完善。我向塞思·菲什曼（Seth Fishman）致以感谢，用这么短的时间让这个项目落地。同样，我也要感谢梅拉尼·托托罗利（Melanie Tortoroli），她用自己专业的指导、洞察和编辑技巧为本书撰写了引人入胜的标题；感谢乔治娜·莱科克（Georgina Laycock）对本书的编辑以及提出的创意和建议；感谢维京（Viking）和约翰·墨里（John Murray）的授权。还要特别感谢同我合作了三次的天才亚历杭德罗·吉拉尔多（Alejandro Giraldo），为本书奉献了他的艺术作品。感谢萨姆·彭罗斯（Sam Penrose）、伊莲娜·格拉斯曼（Elena Glassman）和马克·里德（Mark Reid）花了大量时间来帮忙审阅手稿并提供了很多有见地的反馈，在第11课中提到的BASIC语言的链接便

是由马克·里德帮忙提供的。我要感谢马克·瑟曼（Mark Surman），是他在早期支持了这个项目；感谢彼得·诺米格（Peter Norvig），是他分享了对初期版本的意见；感谢伊莲娜·格拉斯曼（Elena Glassman），是她向我介绍了彼得。正是有了以上各位的帮助，这本书才得以出版。

最后，我还要感谢我的妻子达娜（Danah）以及我的父母。谢谢。


链表
栈
基本原理
数组
渐进表示
最坏情况，
一般情况
二分查找
搜索方法
线性搜索
哈希表


图书馆分类
优先队列
分类方法
归并排序
插入排序
快速排序


字符串
图表方法
迷宫解法
链接分析
哈夫曼编码