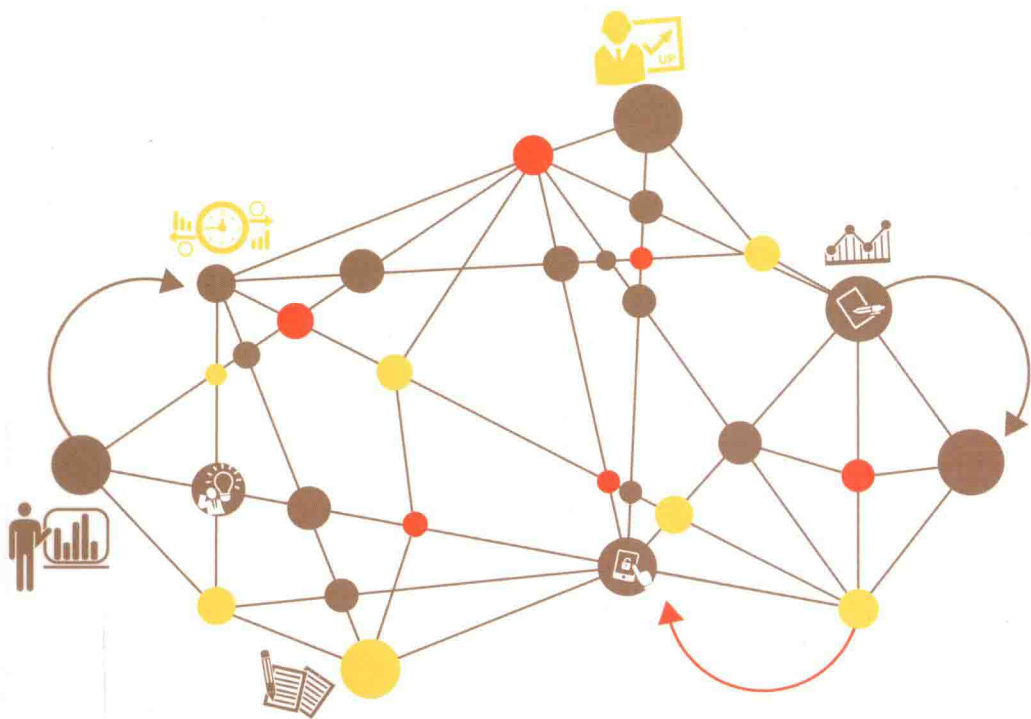


非常白话并且很好理解的算法介绍和实现思路  
温故知新，带你轻松走进互联网企业的大门

Broadview®  
www.broadview.com.cn

# 轻松学算法

互联网算法面试宝典 / 赵焯 编著

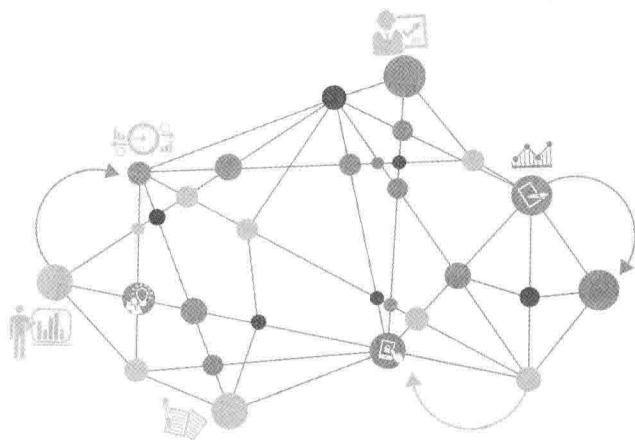


中国工信出版集团

电子工业出版社  
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY  
<http://www.phei.com.cn>

# 轻松学算法

互联网算法面试宝典 / 赵焯 编著



电子工业出版社  
Publishing House of Electronics Industry  
北京·BEIJING

## 内 容 简 介

本书共分为 12 个章节，首先介绍了一些基础的数据结构，以及常用的排序算法和查找算法；其次介绍了两个稍微复杂一些的数据结构——树和图，还介绍了每种数据结构和算法的适用场景，之后是一些在工作与面试中的实际应用，以字符串、数组、查找等为例介绍了一些常见的互联网面试题及分析思路，便于读者了解这些思路，顺利地通过互联网公司的面试；最后介绍了一些常见的算法思想，便于读者对今后遇到的算法问题更轻易地想出解决方案。

本书的讲解轻松有趣，易于读者把烦琐、枯燥的算法学习变为有趣、愉快的学习，把被动学习变为主动学习。本书也介绍了一些会在工作面试中用到的算法。对于一些正在学习算法的人来说，本书绝对是可以帮你轻松掌握算法的辅助资料；对于已经了解算法的人来说，可以从本书中了解到这些算法是如何在实际工作中使用的。

本书适合即将毕业的学生、初入职场的工程师及想补充基础算法知识的人学习，也适合作为一本互联网公司面试的参考书，更是一本不可多得的便于读者时常补习算法知识的收藏宝典。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。

版权所有，侵权必究。

### 图书在版编目（CIP）数据

轻松学算法：互联网算法面试宝典 / 赵焯编著. —北京：电子工业出版社，2016.7

ISBN 978-7-121-29194-4

I. ①轻… II. ①赵… III. ①算法语言 IV. ①TP312

中国版本图书馆 CIP 数据核字（2016）第 143464 号

策划编辑：董 英

责任编辑：徐津平

印 刷：三河市双峰印刷装订有限公司

装 订：三河市双峰印刷装订有限公司

出版发行：电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本：787×980 1/16 印张：25.5 字数：500 千字

版 次：2016 年 7 月第 1 版

印 次：2016 年 7 月第 1 次印刷

印 数：3000 册 定价：69.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 [zltz@phei.com.cn](mailto:zltz@phei.com.cn)，盗版侵权举报请发邮件至 [dbqq@phei.com.cn](mailto:dbqq@phei.com.cn)。

本书咨询联系方式：010-51260888-819 [faq@phei.com.cn](mailto:faq@phei.com.cn)。

# 前 言

互联网越来越热门了，相信每个人都或多或少地在体验如今的互联网带给我们的各种便利。借助之前非常火热的电商，我们可以足不出户地购买衣服和日常用品；现在，就算是一个超级宅的人，也可以不出门便可完成订餐、在线交水电费、通过各种到家服务进行保洁和按摩、通过应用将在超市买的東西轻松配送到家，等等。在线办公在一些行业中也开始流行，视频会议更是可以轻松地实现异地交流。之前我们还需要见面签约，现在越来越多地在进行在线签约。

不得不感谢互联网带给我们的各种便利，但这背后是由很多产品、运营和技术人员的努力在支撑的。

我最初所在的公司属于传统行业，但也是一家服务与互联网公司。相信任何行业的技术员工都想进入互联网公司，这其中的好处有太多：技术的挑战、大用户量、大数据、高并发，这些都是我们所渴望的。

经常听到身边有很多人在抱怨算法不好学、学会了记不住、记住了不知道怎么用等，所以我决定写本书，结合自己的经验讲解一些算法的实际应用及适用场景，希望通过本书帮助更多的朋友进入互联网公司。

其实很多人怯场时无非担心的是自己的算法太差、技术太烂、别人会瞧不起，等等。本书可以帮助读者解决一些基础、常见的算法问题，当然，在技术上仍需自己努力，若再有一点运气，则一定可以找到理想的公司。不要害怕，很多时候就算没有面试成功，也应

该总结一下，等过段时间后便能感悟到自己的成长。

算法有很多，而且不停地有新的算法出现。本书将介绍其中一些比较基础且常用的算法，当然，会先简单介绍几个基础的数据结构作为学习算法的铺垫；之后会介绍一些在工作中可能用到的算法；最后会介绍一些新兴的算法，以拓展读者的思路。

我会尽量以轻松、愉快的方式介绍每一个算法，由浅入深地介绍如何在工作中使这些算法变成我们的代码，让我们在开发应用时更高效。

# 目 录

第 1 章 数组、集合和散列表 .....	1
1.1 要用就要提前想好的数据结构——数组 .....	2
1.1.1 什么是数组 .....	2
1.1.2 数组的存储结构 .....	3
1.1.3 数组的特点 .....	6
1.1.4 数组的适用场景 .....	7
1.2 升级版数组——集合 .....	8
1.2.1 什么是集合 .....	8
1.2.2 集合的实现 .....	8
1.2.3 集合的特点 .....	13
1.2.4 集合的适用场景 .....	13
1.2.5 数组与变长数组的性能 .....	14
1.3 数组的其他应用——散列表 .....	14
1.3.1 什么是散列表 .....	15
1.3.2 对散列表函数产生冲突的解决办法 .....	16
1.3.3 散列表的存储结构 .....	17
1.3.4 散列表的特点 .....	18
1.3.5 散列表的适用场景 .....	20
1.3.6 散列表的性能分析 .....	21

1.4	小结	28
<b>第2章</b>	<b>栈、队列、链表</b>	<b>29</b>
2.1	汉诺塔游戏——栈	30
2.1.1	什么是汉诺塔	30
2.1.2	什么是栈	31
2.1.3	栈的存储结构	31
2.1.4	栈的特点	36
2.1.5	栈的适用场景	36
2.2	火爆的奶茶店——队列	37
2.2.1	什么是队列	37
2.2.2	队列的存储结构	38
2.2.3	队列的特点	43
2.2.4	队列的适用场景	44
2.3	用栈实现队列	45
2.3.1	用两个栈实现队列	46
2.3.2	两个队列实现栈	50
2.4	链表	53
2.4.1	什么是链表	54
2.4.2	链表的存储结构	54
2.4.3	链表的操作	55
2.4.4	链表的特点	66
2.4.5	链表的适用场景	66
2.4.6	链表的性能分析	67
2.4.7	面试举例：如何反转链表	68
2.5	链表其实也可以用数组模拟	69
2.5.1	静态链表	70
2.5.2	静态链表的实现	70
2.5.3	静态链表的特点	80
2.6	再谈汉诺塔	81
2.6.1	汉诺塔的移动原理	81
2.6.2	汉诺塔的递归实现	82

第 3 章 排序算法	84
3.1 算法基础	85
3.1.1 时间复杂度	85
3.1.2 空间复杂度	88
3.1.3 稳定性	88
3.2 快而简单的排序——桶排序	89
3.2.1 举个例子	89
3.2.2 什么是桶排序	90
3.2.3 桶排序的实现	90
3.2.4 桶排序的性能及特点	92
3.2.5 桶排序的适用场景	93
3.3 咕嘟咕嘟的冒泡排序	94
3.3.1 什么是冒泡排序	94
3.3.2 冒泡排序的原理	94
3.3.3 冒泡排序的实现	96
3.3.4 冒泡排序的特点及性能	99
3.3.5 冒泡排序的适用场景	99
3.3.6 冒泡排序的改进方案	100
3.4 最常用的快速排序	100
3.4.1 什么是快速排序	101
3.4.2 快速排序的原理	101
3.4.3 快速排序的实现	105
3.4.4 快速排序的特点及性能	107
3.4.5 快速排序的适用场景	108
3.4.6 快速排序的优化	108
3.5 简单的插入排序	109
3.5.1 什么是插入排序	110
3.5.2 插入排序的原理	110
3.5.3 插入排序的实现	112
3.5.4 插入排序的特点及性能	114
3.5.5 插入排序的适用场景	115



3.6	直接插入的改进——希尔排序	115
3.6.1	什么是希尔排序	116
3.6.2	希尔排序的原理	116
3.6.3	希尔排序的实现	118
3.6.4	希尔排序的特点及性能	120
3.6.5	希尔排序的适用场景	121
3.7	简单选择排序	121
3.7.1	什么是选择排序	122
3.7.2	简单选择排序的原理	122
3.7.3	简单选择排序的实现	123
3.7.4	选择排序的特点及性能	125
3.7.5	简单选择排序的优化	125
3.7.6	选择排序的适用场景	126
3.8	小结	126
<b>第4章</b>	<b>搜索，没那么难</b>	<b>128</b>
4.1	最先想到的——顺序查找	129
4.1.1	最先想到的	129
4.1.2	顺序查找的原理与实现	129
4.1.3	顺序查找的特点及性能分析	131
4.1.4	顺序查找的适用场景	132
4.2	能不能少查点——二分查找	133
4.2.1	某些特殊情况的查找需求	133
4.2.2	二分查找的原理及实现	133
4.2.3	二分查找的优化	137
4.2.4	二分查找的特点及性能分析	138
4.2.5	二分查找的适用场景	139
4.2.6	我是来还债的——之前欠的二分插入排序	139
4.3	行列递增的矩阵查找——二分查找思维拓展	141
4.3.1	一道题	142
4.3.2	几个解法	142
4.3.3	其他拓展	153

4.4	分块查找	154
4.4.1	每次插入元素都要有序吗	154
4.4.2	什么是分块查找	155
4.4.3	分块查找的原理及实现	155
4.4.4	分块查找的特点与性能分析	159
4.4.5	分块查找的适用场景	160
4.5	查找算法小结	161
4.6	搜索引擎与倒排索引	162
4.6.1	什么是搜索引擎	162
4.6.2	倒排索引	162
4.6.3	索引实例	163
4.6.4	倒排索引的关键字提取	164
4.6.5	商业索引的其他拓展	164
<b>第 5 章</b>	<b>树</b>	<b>166</b>
5.1	树的定义及存储结构	167
5.1.1	什么是树	167
5.1.2	其他相关术语	168
5.1.3	都有哪些树	170
5.1.4	树的存储结构及实现	170
5.2	二叉树	173
5.2.1	什么是二叉树	173
5.2.2	还有两种特殊的二叉树	173
5.2.3	二叉树的实现	175
5.2.4	二叉树的遍历	185
5.2.5	完全二叉树	187
5.3	二叉树的查找算法	188
5.3.1	二叉查找树	188
5.3.2	平衡二叉树	198
5.4	B-树、B+树	202
5.4.1	什么是 B-树	203
5.4.2	什么是 B+树	204

5.4.3	B-树、B+树的特点及性能分析	205
5.5	在 MySQL 数据库中是如何应用 B+树的	206
5.6	哈夫曼树	209
5.6.1	几个术语	209
5.6.2	什么是哈夫曼树	209
5.6.3	哈夫曼树的构造	211
5.6.4	哈夫曼编码及解码	213
5.7	堆	215
5.7.1	什么是堆	215
5.7.2	堆的基本操作	216
5.7.3	堆的性能分析	221
5.7.4	堆排序	222
5.8	红黑树	224
5.8.1	什么是红黑树	224
5.8.2	红黑树的特点与优势	224
<b>第 6 章</b>	<b>图</b>	<b>226</b>
6.1	图的定义及相关术语	227
6.1.1	什么是图	227
6.1.2	图的分类	227
6.1.3	图的相关术语	228
6.2	图的表示与存储方式	229
6.2.1	邻接矩阵	229
6.2.2	邻接表	234
6.3	更多的图	237
6.3.1	连通图	238
6.3.2	强连通图	238
6.4	深度优先遍历与广度优先遍历	238
6.4.1	深度优先遍历	239
6.4.2	广度优先遍历	248
6.4.3	两种遍历方法的对比	253

6.5	最短路径	254
6.5.1	带权图	254
6.5.2	Dijkstra 算法	255
6.5.3	Floyd 算法	269
<b>第 7 章</b>	<b>字符串</b>	<b>272</b>
7.1	字符及字符串简介	273
7.1.1	什么是字符	273
7.1.2	什么是字符串	273
7.2	字符的全排列	275
7.2.1	问题描述及分析	275
7.2.2	最先想到的	275
7.2.3	利用字典序排列	278
7.3	反转字符串	283
7.3.1	问题描述及分析	283
7.3.2	最先想到的	283
7.3.3	对换反转法	285
7.3.4	拓展——旋转字符串	287
7.4	判断回文	288
7.4.1	问题描述及分析	288
7.4.2	对半判断	289
7.5	寻找最大的回文子串	290
7.5.1	问题描述及分析	290
7.5.2	遍历实现	291
7.6	将字符串转换为数字	293
7.6.1	问题描述及分析	293
7.6.2	解决	293
7.7	判断字符串包含的问题	297
7.7.1	问题描述及分析	297
7.7.2	非常简单的解决思路	297
7.7.3	利用排序进行优化	299
7.7.4	投机取巧的素数方案	302

7.7.5	用散列表进行实现	304
7.7.6	用位运算进行实现	305
<b>第 8 章</b>	<b>数组还有好多玩法</b>	<b>308</b>
8.1	从数组中找出其和为指定值的两个数	309
8.1.1	问题描述及分析	309
8.1.2	最简单的办法	309
8.1.3	一切为了速度	310
8.1.4	排序总是好的选择	311
8.1.5	还能更好	313
8.2	找出连加值最大的子数组	315
8.2.1	问题描述及分析	315
8.2.2	暴力穷举法	316
8.2.3	动态规划法	319
8.2.4	问题拓展	321
8.3	数组正负值排序	323
8.3.1	问题描述及分析	323
8.3.2	最直观的解法	324
8.3.3	借鉴简单插入排序	325
8.3.4	借鉴快速排序	327
8.3.5	拓展	329
8.4	将数组随机打乱顺序	329
8.4.1	问题描述及分析	329
8.4.2	随便糊弄一下	330
8.4.3	插入排序的思想又来了	331
8.5	数组赋值	333
8.5.1	问题描述及分析	333
8.5.2	分解计算	333
8.6	寻找旋转数组的拐点	335
8.6.1	问题描述及分析	335
8.6.2	最简单的方法	336
8.6.3	利用“有序”	337

8.7 荷兰国旗问题 .....	339
8.7.1 问题描述及分析 .....	339
8.7.2 排序法 .....	340
8.7.3 快速排序带给人的灵感 .....	340
<b>第 9 章 查找又来了 .....</b>	<b>344</b>
9.1 出现次数超过一半的数字 .....	345
9.1.1 问题描述及分析 .....	345
9.1.2 排序法 .....	345
9.1.3 散列表 .....	347
9.1.4 删除法 .....	349
9.1.5 更优的解法 .....	349
9.2 寻找缺少数字 .....	352
9.2.1 问题描述及分析 .....	352
9.2.2 借助快速排序 .....	352
9.2.3 借助散列表实现 .....	354
9.2.4 投机取巧法 .....	355
9.2.5 思路拓展 .....	357
9.3 在 10 亿个数中找出最大的 1 万个数 .....	357
9.3.1 问题描述及分析 .....	357
9.3.2 拍脑袋想问题 .....	358
9.3.3 借助快速排序 .....	358
9.3.4 不想都放入内存 .....	358
9.3.5 传说中的大顶堆 .....	359
9.3.6 拓展——找出数组中第 $k$ 大的数 .....	359
<b>第 10 章 更多 .....</b>	<b>363</b>
10.1 不使用额外的空间交换两个数 .....	364
10.1.1 问题描述 .....	364
10.1.2 分析问题 .....	364
10.1.3 解决问题 .....	364

10.2	拿乒乓球的问题	365
10.2.1	问题描述	365
10.2.2	分析问题	365
10.2.3	解决问题	365
<b>第 11 章</b>	<b>实现一些集合类</b>	<b>367</b>
11.1	栈 (Stack) 的实现	368
11.1.1	实现前的思考	368
11.1.2	实现栈	368
11.1.3	参考 JDK 的实现	372
11.2	变长数组 (ArrayList) 的实现	372
11.2.1	实现前的思考	372
11.2.2	实现变长数组	373
11.2.3	参考 JDK 的实现	380
11.3	散列表 (HashMap) 的实现	381
11.3.1	实现前的思考	381
11.3.2	实现散列表	381
11.3.3	参考 JDK 的实现	389
<b>第 12 章</b>	<b>方向</b>	<b>390</b>
12.1	算法的一些常用思想	391
12.1.1	分治法	391
12.1.2	动态规划	391
12.1.3	贪心算法	391
12.1.4	回溯法	392
12.1.5	分支限界法	392
12.2	新兴算法	392
12.2.1	加密算法	392
12.2.2	商业算法	393
12.3	其他算法	393
12.3.1	基数估计算法	393
12.3.2	蚁群算法	394

# 数组、集合和散列表

我们将在本章中学习最基础、最简单的数据结构及其延伸。

数组，作为数据结构中最基础的一个存储方式，是我们学习一切数据结构、算法的基石。大部分数据结构都可以用数组来实现。本章会介绍数组的概念、存储结构、特点及适用场景。

集合这个结构其实并不存在于我们在学校学习的数据结构知识中，而是在一些高级语言中出现的，算是升级版的数组。本章会介绍集合的特点、实现及它的适用场景。

散列表，又叫哈希表（Hash Table），其实在很多高级语言里是在数组的基础上实现的，当然散列表也有其他实现形式。本章会详细介绍散列表的基本概念、实现方式，并结合实现方式介绍其对应的存储结构、特点及适用场景。



## 1.1 要用就要提前想好的数据结构——数组

要用就要提前想好？为什么？这其实是由数组的一个特点决定的，那就是对于数组这个数据结构，在用它之前必须提前想好它的长度；有了长度，才能知道该为这个存储结构开辟多少空间；而在决定了长度之后，不管我们最后往里面填充的数据够不够长，没有用到的空间也就都浪费了；如果我们想往这个数组中放入的数据超过了提前设定好的长度，那么是不可行的，因为空间只有这么大。

### 1.1.1 什么是数组

数组（Array），就是把有限个数据类型一样的元素按顺序放在一起，用一个变量命名，然后通过编号可以按顺序访问指定位置的元素的一个有序集合。

其实简单来说，就是为了方便而把这些元素放在一起。我们通过编号去获取每个元素，这个编号叫作下标或者索引（Index），一般的语言是从 0 开始的。

我们常说的数组一般指一维数组，当然还有多维数组，虽然多维数组并不常用。

多维的实现其实是数组的某些元素本身也是一个数组，这里以一个标准的二维数组为例进行介绍。其实，二维数组相当于每个元素的长度都一样的一个一维数组（也就是我们常说的数组）。可以想象一下矩阵（若不了解矩阵，则请阅读 1.1.2 节中标准二维数组的存储结构示例），其实它和数学中的矩阵类似。

在很多弱语言中，并不要求每个元素的长度都一样，可以某些元素是数组（长度可以不一样），某些元素不是数组，甚至每个元素的数据类型都不同。这里讲的二维数组指的是标准的二维数组。

注：弱类型语言也叫作弱类型定义语言，简称弱语言。弱语言一般对语言的标准没有特别的要求。比如在 JavaScript 中用 var 声明变量，不会指定该变量是哪种类型。如果想更多地了解弱语言，则请参考 JavaScript，该语言主要用于前端开发。强语言对编写规则比较