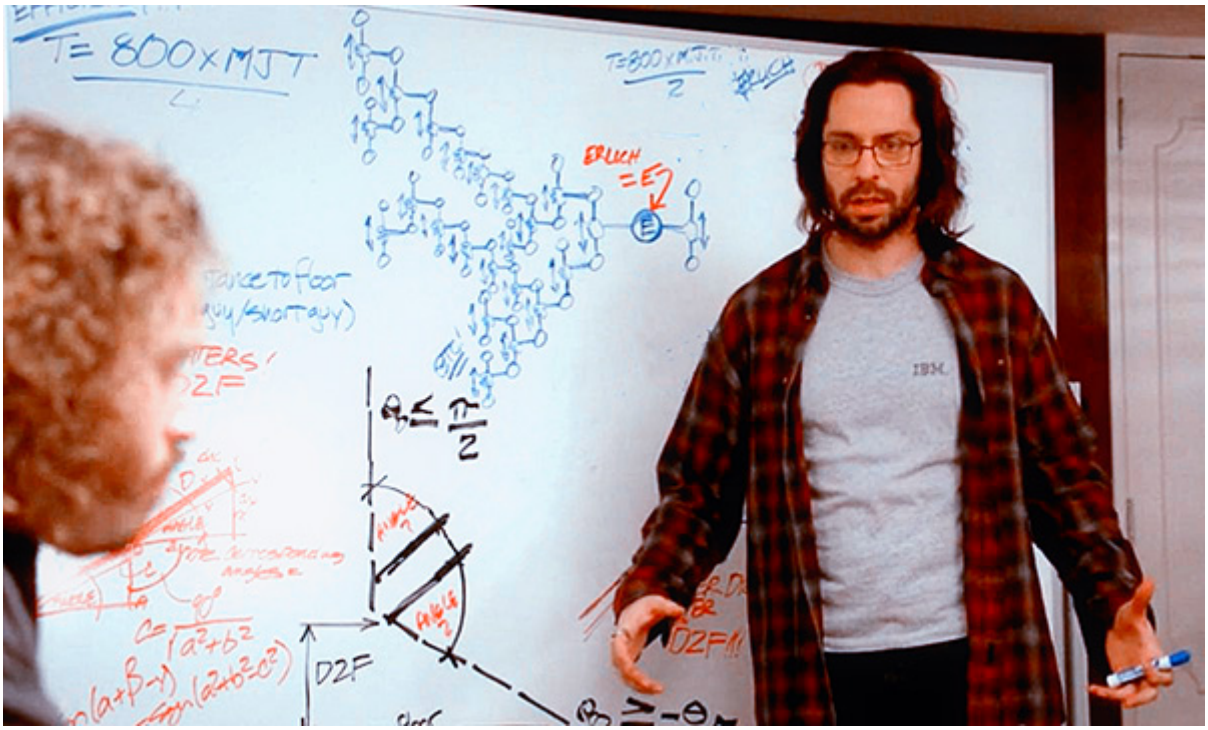
编程面试大学

> 原先我为了成为一个软件工程师而建立这份简单的学习主题清单， 但这份清单随着时间的推移而膨胀成今天这样。在做完这份清单上的每个目标后，我成为了 Amazon 的软件开发工程师! 你或许不需要像我一样学习这么多。但是，让你成为一位称职工程师所需要的知识都在这里了。
>
> 我每天自学8~12小时，这样持续了好几个月。这是我的故事：为什么我为了 Google 面试而自学了8个月。
>
> **请注意:** 你不需要像我一样那么努力学习。我在一些不必要的事情上浪费了很多时间。关于这个问题下面有更多信息。我会帮助你节省宝贵的时间，让你达到目标。 在这份清单内的主题会让你拥有足够的知识去面对几乎每家软件公司的技术面试，包括科技巨头：Amazon、Facebook、Google，以及 Microsoft。
>
> **祝你好运!**

这是?



这是我为成为一家大公司的软件工程师制定的多月学习计划。

要求:

- 一点编程经验（变量、循环、方法/函数等）
- 耐心
- 时间

注意，这是一份关于 **软件工程** 的学习计划，而不是前端工程或全栈开发。 这些职业路径有很多详细的路线图和课程资料可以在其他地方找到（请参阅 https://roadmap.sh/ 获取更多信息）。

在大学计算机科学专业中，有很多知识需要学习，但是只掌握大约75%的内容就足够应对面试了，这也是我在这里涵盖的内容。 如果你想进行完整的自学计算机科学项目，可以参考Kamran Ahmed的计算机科学路线图：https://roadmap.sh/computer-science

---

目录

学习计划

学习的主题

    - 有向图（directed）
    - 无向图（undirected）
    - 邻接矩阵（adjacency matrix）
    - 邻接表（adjacency list）
    - 遍历：广度优先(BFS), 深度优先(DFS)
- 更多知识
    - 递归
    - 动态规划
    - 设计模式
    - 组合 & 概率
    - NP, NP-完全和近似算法
    - 缓存
    - 进程和线程
    - 测试
    - 调度
    - 字符串搜索和操作
    - 字典树（Tries）
    - 浮点数
    - Unicode
    - 字节顺序
    - 网络
- 最终复习

获得工作机会

- 更新你的简历
- 找工作
- 面试流程与一般面试准备
- 当面试来临的时候
- 问面试官的问题
- 当你获得了梦想的职位

---------- **以下所有内容均为可选项** ----------

可选的额外主题和资源

- 额外书籍
- 系统设计、可扩展性和数据处理
- 附加学习
    - 编译器
    - Emacs and vi(m)
    - Unix 命令行工具
    - 信息论
    - 奇偶校验位 & 汉明码 (视频)
    - 系统熵值
    - 密码学
    - 压缩
    - 计算机安全

为何要用到它?

如果你想在一家大公司担任软件工程师，这些是你必须了解的事情。

如果你错过了计算机科学的学位，就像我一样，这将帮助你迎头赶上，并节省四年的时间。

当我开始这个项目时，我对堆栈和堆没有任何了解， 也不知道大O表示法或者关于树的任何东西，也不知道如何遍历图形。 如果让我编写一个排序算法，相信我它会很糟糕。 我曾经使用过的每种数据结构都是内置在语言中的，并且我完全不知道它们在底层是如何工作的。 除非运行中的进程出现"内存不足"错误，否则我从来没有管理过内存，并且那时候就需要找到一种解决方法。 在我的生活中，我使用过一些多维数组和成千上万个关联数组，但从未从头开始创建数据结构。

这是一个漫长的计划，以至于花费了我数月的时间。若你早已熟悉大部分的知识，那么也许能节省大量的时间。

如何使用它

下面所有的东西都只是一个概述。因此，你需要由上而下逐一地去处理它。

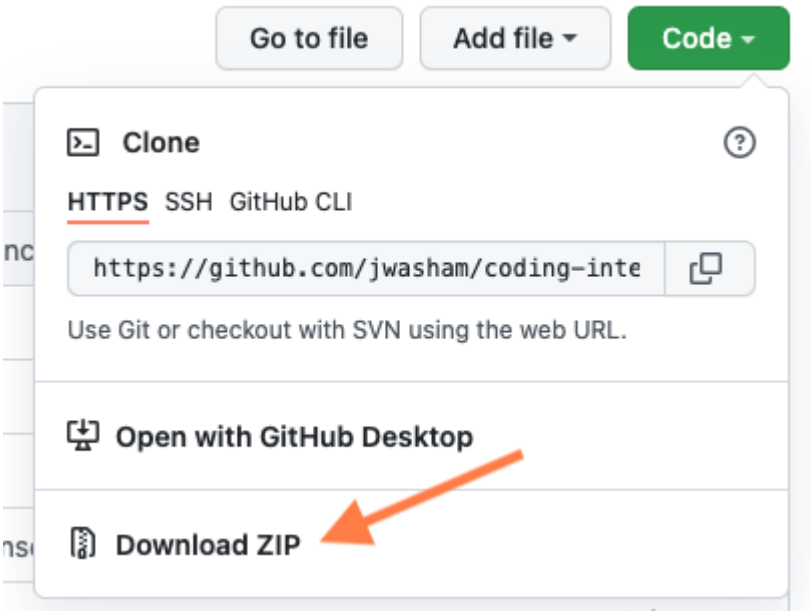
在学习过程中，我使用 GitHub 特殊语法的 Markdown 去检查计划的进展，包括使用包含任务进度的任务列表。

- 更多关于 Github-flavored Markdown 的详情

如果你不想使用 Git

在该页面上，单击顶部附近的 Code 按钮，然后单击“Download ZIP”。解压文件，就可以使用文本文件了。

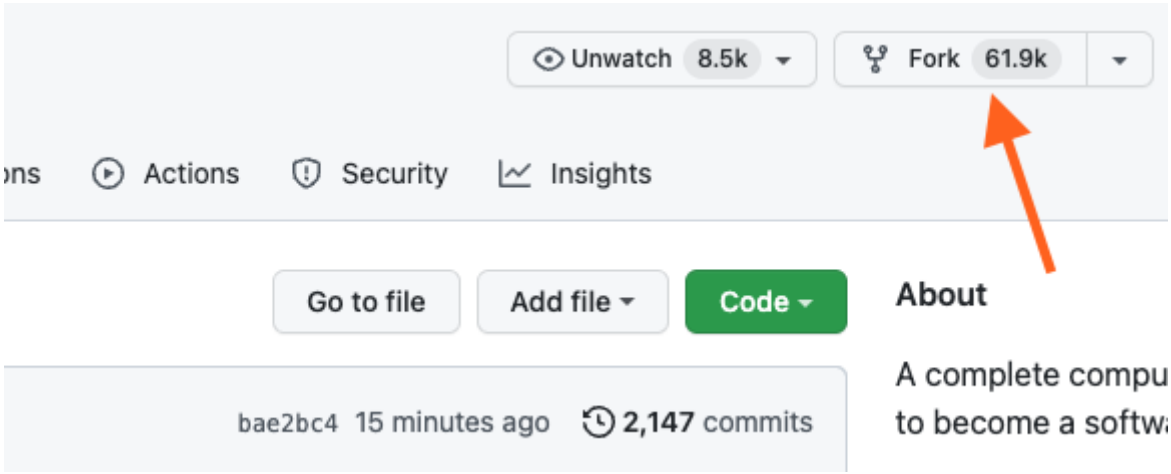
如果你打开一个代码编辑器，你会看到所有格式都很好。



如果你不介意 Git

创建一个新的分支，这样你就可以检查类似这样的项目了，只需在方括号中放入一个x：[x]

1. 在 GitHub 上 Fork 该仓库： 点击 Fork 按钮，将 `https://github.com/jwasham/coding-interview-university` 仓库复制到你的 GitHub 账号中。



2. 克隆项目到本地：

```
git clone git@github.com:<your_github_username>/coding-interview-
university.git
cd coding-interview-university
git remote add upstream https://github.com/jwasham/coding-interview-
university
git remote set-url --push upstream DISABLE  # 这样你就不会将个人进展推回到原始
仓库了。
```

3. 在你完成了一些修改后，在框框中打 x：

```
git commit -am "Marked personal progress"
git pull upstream main  # 将您的分支与原始仓库中的更改保持最新

git push # just pushes to your fork
```

不要觉得自己不够聪明

- 大多数成功的软件工程师都非常聪明，但他们都有一种觉得自己不够聪明的不安全感。
- 下面的视频可以帮助你克服这种不安全感：
  - 天才程序员的神话
  - 不要单打独斗：面对技术中的隐形怪物

相关视频资源

部分视频只能通过在 Coursera 或者 Edx 课程上注册登录才能观看。 这些视频被称为网络公开课程（MOOC）。有时候某些课程需要等待好几个月才能获取，这期间你无法观看这些课程的影片。

很感谢你能帮我把网络公开课程的视频链接转换成公开的，可持续访问的视频源， 比如 YouTube 视频，以代替那些在线课程的视频。 此外，一些大学的讲座视频也是我所青睐的。

选择编程语言

你需要为你做的编程面试选择一种编程语言， 但你也需要找到一种可以用来学习计算机科学概念的语言。

最好是同一种语言，这样你只需精通其中一种。

对于这个学习计划

在这个学习计划中，我主要使用了两种编程语言：C和Python。

- C: 非常底层。它允许你处理指针和内存的分配与释放，因此你能够深入理解数据结构和算法。 在像Python或Java这样的高级语言中，这些细节被隐藏起来。在日常工作中，这是很好的， 但当你学习这些底层数据结构时，感受它们与计算机硬件的联系也是非常有益的。
  - C 语言无处不在。在你学习的过程中，你会在书籍、讲座、视频以及**任何地方**看到C语言的例子。
  - 《C程序设计语言（第2版）》

        - 这是一本简短的书，但它会让你很好地掌握C语言，只要稍微练习一下， 你很快就能熟练使用。理解C语言有助于你了解程序和内存是如何工作的。
        - 你不需要深入研究这本书（甚至不用读完它）。只要阅读到你感觉舒服，并能写一些C语言的代码就可以了。
        - 书中问题的答案
- Python: 现代且非常灵活，我学习它是因为它非常实用，同时在面试中也能让我写更少的代码。

这是我的个人喜好，当然你可以根据自己的偏好来选择。

也许你并不需要，但以下是一些学习新编程语言的网站：

- Exercism
- Codewars
- HackerEarth
- Scaler Topics (Java, C++)
- Programiz PRO Community Challenges)

对于你的编程面试

你可以在编程这一环节，使用一种自己用起来较为舒适的语言去完成编程，但对于大公司，你只有三种固定的选择：

- C++
- Java
- Python

你也可以使用下面两种编程语言，但可能会有某些限制，你需要事先查明：

- JavaScript
- Ruby

这是我写的一篇关于选择面试语言的文章： 为编程面试选择一种语言。 这是我发布帖子所基于的原始文章：Choosing a Programming Language for Interviews

你需要对你所选择的语言感到非常舒适且足够了解。

更多关于语言选择的阅读：

- 选择适合你的编程面试的语言

在此查看相关语言的资源

数据结构和算法的书籍

这本书将为你的计算机科学打下基础。

只需选择一种你感到舒适的语言。你将会进行大量阅读和编码工作。

C

- C语言中的算法，第1-5部分（捆绑包），第3版
    - 基础知识，数据结构，排序，搜索和图算法

Python

- [ ] Python数据结构和算法
  - 作者：Goodrich、Tamassia、Goldwasser
  - 我非常喜爱这本书，它包含了所有东西
  - 很 Python 的代码
  - 我的读书报告：https://startupnextdoor.com/book-report-data-structures-and-algorithms-in-python/

Java

你的选择：

- Goodrich, Tamassia, Goldwasser
  - Java数据结构与算法
- Sedgewick and Wayne:
  - 算法（第4版）
  - 免费Coursera课程，涵盖该书内容（由作者授课！）：
    - 算法I
    - 算法II

C++

你的选择：

- Goodrich, Tamassia, and Mount
  - C++数据结构与算法（第2版）
- Sedgewick and Wayne
  - C++算法（第1-4部分）：基础知识，数据结构，排序，搜索
  - C++算法第5部分：图算法

面试准备书籍

你不需要买一堆这些。老实说，《破解编程面试》可能已经足够了， 但我买了更多来给自己更多的练习。但我总是做得太多。

这两个都是我买的，他们给了我大量的练习。

- Programming Interviews Exposed: Coding Your Way Through the Interview, 4th Edition
  - 提供C++和Java语言的答案
  - 这本书是准备《Cracking the Coding Interview》的很好热身书
  - 难度适中。大多数问题可能比实际面试中遇到的问题要简单（根据我所读的内容）
- Cracking the Coding Interview, 6th Edition
  - 提供Java语言的答案

如果你有很多额外的时间：

选择一个：

- Elements of Programming Interviews (C++ version)

- Elements of Programming Interviews in Python
- Elements of Programming Interviews (Java version) - 配套项目-本书中每个问题的方法存根和测试用例

不要犯我的错误

这个列表在很多个月里不断增长，是的，它变得失控了。

以下是我犯过的一些错误，这样你就能有更好的体验。而且你将节省数月时间。

1. 你不可能把所有的东西都记住

我看了数小时的视频并做了大量笔记，几个月后有很多东西我都不记得了。 我花了三天时间浏览我的笔记并制作闪卡，以便进行复习。其实，并不需要那么多知识。

请阅读以下的文章以免重蹈覆辙:

记住计算机科学知识。

2. 使用抽认卡

为了解决善忘的问题，我制作了一个抽认卡的网页，用于添加两种抽认卡：一般的及带有代码的。每种卡都会有不同的格式设计。 而且，我还以移动设备为先去设计这些网页，以使得在任何地方，我都能通过我的手机及平板去回顾知识。

你也可以免费制作属于你自己的抽认卡网站：

- 抽认卡页面的代码仓库

我不建议使用我的闪卡。它们太多了，而且大部分都是你不需要的琐事。

但是如果你不想听我的话，那就随你吧：

- 我的抽认卡数据库 —— 旧 1200 张
- 我的抽认卡数据库 —— 新 1800 张

有一点需要记住的是，我做事有点过头，以至于卡片都覆盖到所有的东西上，从汇编语言和 Python 的细枝末节，到机器学习和统计都被覆盖到卡片上。 而这种做法，对于要求来说是多余的。

在抽认卡上做笔记： 若你第一次发现你知道问题的答案时，先不要急着把其标注成"已知"。 反复复习这张抽认卡，直到每次都能答对后才是真正学会了这个问题。 反复地问答可帮助你深刻记住该知识点。

这里有个替代我抽认卡的网站 Anki，很多人向我推荐过它。 这个网站用同一个字卡重复出现的方式让你牢牢地记住知识。 这个网站非常容易使用，支持多平台，并且有云端同步功能。在 iOS 平台上收费25美金，其他平台免费。

这是我用 Anki 这个网站里的格式所储存的抽认卡资料库: https://ankiweb.net/shared/info/25173560 （感谢 @xiewenya）。

一些学生提到了关于空白间距的格式问题，可以通过以下方法进行修复：打开卡片组，编辑卡片，点击"卡片"选项，选择"样式"单选按钮，在卡片类中添加成员 "white-space: pre;"。

3. 在学习过程中做编程面试题

这非常重要。

在学习数据结构和算法的同时，开始做编程面试题。

你需要将所学知识应用于解决问题，否则你会忘记。我曾经犯过这个错误。

一旦你学完一个主题，并且对它有了一定的掌握，比如 **链表（linked lists）**：

1. 打开其中一本编程面试书籍（或下方列出的编程问题网站之一）。
2. 请先做2或3个关于链表的问题。
3. 继续学习下一个主题。
4. 稍后，回来再做另外2或3个链表问题。
5. 使用这种方法来学习每个新主题。

在学习这些内容的过程中不断做问题，而不是之后。

你被雇佣的不是因为你的知识，而是因为你知道如何应用这些知识。

下面列出了许多资源供你参考。继续前进吧。

4. 专注

在学习的过程中，往往会有许多令人分心的事占据着我们宝贵的时间。 因此，专注和集中注意力是非常困难的。放点纯音乐能帮上一些忙。

没有包含的内容

有一些熟悉且普遍的技术在此未被谈及到：

- Javascript
- HTML，CSS和其他前端技术
- SQL

日常计划

这门课涵盖了很多主题。每个主题可能需要你几天的时间，甚至可能需要一周或更长时间。这取决于你的日程安排。

每天，按照列表中的下一个主题，观看一些关于该主题的视频， 然后用你选择的语言为这门课程编写该数据结构或算法的实现。

在这里你可以查看到我的代码：

- C
- C++
- Python

你不需要记住每个算法。你只需要能够理解它，以便能够编写自己的实现即可。

编程问题练习

> 这是为什么？我还没有准备好面试。

那就回去阅读这部分。

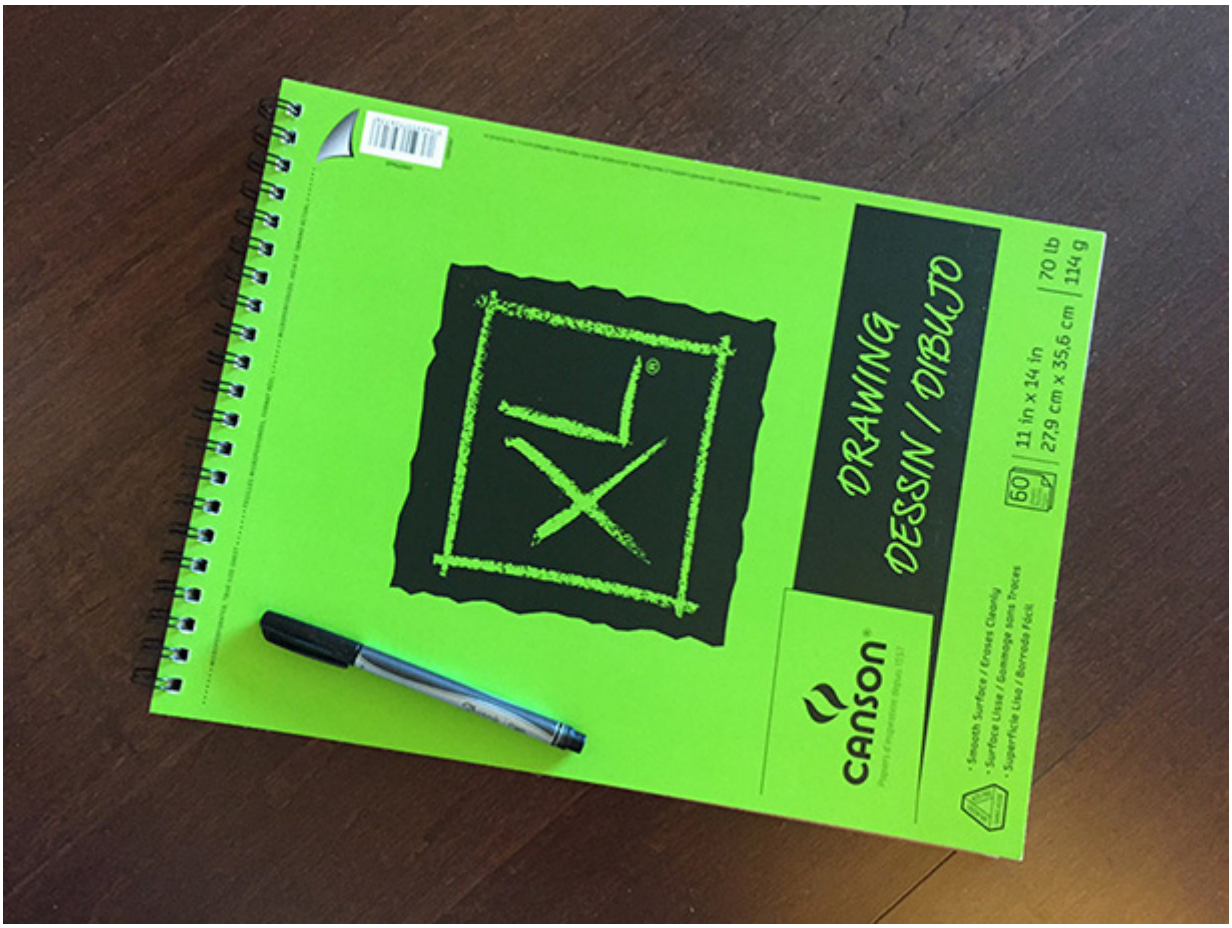
为什么你需要练习编程问题：

- 识别问题，并确定合适的数据结构和算法
- 收集问题的要求
- 像在面试中那样口头表达解决问题的过程
- 在白板或纸上编写代码，而不是在计算机上
- 为您的解决方案确定时间和空间复杂度（参见下文中的大O表示法）。
- 对你的解决方案进行测试

在面试中，有一种方法论的、有交流的问题解决方法。你可以从编程面试书籍中了解这些， 但我发现下面这个网站也非常出色： 算法设计画布

在白板或纸上写代码，而不是在计算机上。使用一些样例输入进行测试。然后在计算机上键入并进行测试。

如果家里没有白板，请从艺术用品店购买一个大型的绘图本。 你可以坐在沙发上练习。这就是我的"沙发白板"。照片中我加了一支笔来衡量尺寸。如果你使用钢笔，你会希望能擦除。 会很快变得凌乱， **我用铅笔和橡皮擦。**



编程问题练习并不是为了记住解决编程问题的答案。

编程问题

别忘了参考你的主要编程面试书籍这里.

解决问题：

- 如何找到解决方案
- 如何分析Topcoder问题陈述

编程面试问题视频：

- IDeserve（88个视频）
- Tushar Roy（5个播放列表）
  - 非常适合问题解决方案的演示
- Nick White - LeetCode解答（187个视频）
  - 解释解决方案和代码的很好
  - 你可以在短时间内观看多个视频
- FisherCoder - LeetCode解答

挑战/练习网站：

- LeetCode
  - 我最喜欢的编程问题网站。对于你准备的1-2个月时间，订阅会费是值得的。
  - 观看上面提到的Nick White和FisherCoder的视频，可以帮助你理解代码解决方案。
- HackerRank
- TopCoder
- Codeforces
- Codility
- Geeks for Geeks
- AlgoExpert
  - 由谷歌工程师创建，也是提高你技能的优秀资源。
- Project Euler
  - 主要关注数学问题，并不完全适合编程面试。

让我们开始吧

好了，说得够多了，让我们学习吧!

但在学习的同时，不要忘记做上面的编码问题！

算法复杂度 / Big-O / 渐进分析法

- 这里没有什么需要实施的，你只是在观看视频并记笔记！耶！
- 这里有很多视频，只要看到你理解为止就好了，你随时可以回来复习。
- 如果你不理解背后的所有数学，不要担心。
- 你只需要理解如何用大O表示法来表达算法的复杂度。
- ☐ 哈佛大学CS50 - 渐进符号（视频）
- ☐ 大O符号（通用快速教程）（视频）
- ☐ 大O符号（以及Ω和Θ）- 最佳数学解释（视频）
- ☐ Skiena（视频）
- ☐ 加州大学伯克利分校关于大O符号（视频）
- ☐ 摊还分析（视频）

- [ ] TopCoder（包括递归关系和主定理）：
  - 计算复杂性：第1部分
  - 计算复杂性：第2部分
- [ ] 速查表
- [ ] [回顾] 在 18 分钟内分析算法（播放列表）（视频）

好吧，差不多就到这里了。

当你阅读《破解编程面试》时，有一个章节专门讲述此事，并在最后进行了一次测验， 以测试你是否能够确定不同算法的运行时间复杂度。这是一个非常全面的复习和测试。

数据结构

- 数组（Arrays）
  - [ ] 介绍：
    - 数组 CS50 哈佛大学
    - 数组（视频）
    - 加州大学伯克利分校CS61B - 线性和多维数组（视频）（从15分32秒开始）
    - 动态数组（视频）
    - 嵌套数组（视频）
  - [ ] 实现一个动态数组（可自动调整大小的可变数组）：
    - [ ] 练习使用数组和指针去编码，并且指针是通过计算去跳转而不是使用索引
    - [ ] 通过分配内存来新建一个原生数据型数组
      - 可以使用 int 类型的数组，但不能使用其语法特性
      - 从大小为16或更大的数（使用2的倍数 —— 16、32、64、128）开始编写
    - [ ] size() —— 数组元素的个数
    - [ ] capacity() —— 可容纳元素的个数
    - [ ] is_empty()
    - [ ] at(index) —— 返回对应索引的元素，且若索引越界则愤然报错
    - [ ] push(item)
    - [ ] insert(index, item) —— 在指定索引中插入元素，并把后面的元素依次后移
    - [ ] prepend(item) —— 可以使用上面的 insert 函数，传参 index 为 0
    - [ ] pop() —— 删除在数组末端的元素，并返回其值
    - [ ] delete(index) —— 删除指定索引的元素，并把后面的元素依次前移
    - [ ] remove(item) —— 删除指定值的元素，并返回其索引（即使有多个元素）
    - [ ] find(item) —— 寻找指定值的元素并返回其中第一个出现的元素其索引，若未找到则返回 -1
    - [ ] resize(new_capacity) // 私有函数
      - 若数组的大小到达其容积，则变大一倍
      - 获取元素后，若数组大小为其容积的1/4，则缩小一半
  - [ ] 时间复杂度
    - 在数组末端增加/删除、定位、更新元素，只允许占 O(1) 的时间复杂度（平摊（amortized）去分配内存以获取更多空间）
    - 在数组任何地方插入/移除元素，只允许 O(n) 的时间复杂度
  - [ ] 空间复杂度
    - 因为在内存中分配的空间邻近，所以有助于提高性能

        - 空间需求 = （大于或等于 n 的数组容积）* 元素的大小。即便空间需求为 2n，其空间复杂度仍然是 O(n)

- 链表（Linked Lists）
    - ☐ 介绍：
        - ☐ 链表 CS50 哈佛大学 - 这样建立了直观感。
        - ☐ 单链表（视频）
        - ☐ CS 61B - 链表1（视频）
        - ☐ CS 61B - 链表 2（视频）
        - ☐ [复习] 4分钟了解链表（视频）
    - ☐ C代码（视频） - 不是整个视频，只是关于Node结构和内存分配的部分。
    - ☐ 链表 vs 数组：
        - 核心链表与数组（视频）
        - 在现实世界中，链表与数组的比较（视频）
    - ☐ 为什么你需要避免使用链表（视频）
    - ☐ 的确：你需要关于"指向指针的指针"的相关知识：（因为当你传递一个指针到一个函数时， 该函数可能会改变指针所指向的地址）该页只是为了让你了解"指向指针的指针"这一概念。 但我并不推荐这种链式遍历的风格。因为，这种风格的代码，其可读性和可维护性太低。
        - 指向指针的指针
    - ☐ 实现（我实现了使用尾指针以及没有使用尾指针这两种情况）：
        - ☐ size() —— 返回链表中数据元素的个数
        - ☐ empty() —— 若链表为空则返回一个布尔值 true
        - ☐ value_at(index) —— 返回第 n 个元素的值（从0开始计算）
        - ☐ push_front(value) —— 添加元素到链表的首部
        - ☐ pop_front() —— 删除首部元素并返回其值
        - ☐ push_back(value) —— 添加元素到链表的尾部
        - ☐ pop_back() —— 删除尾部元素并返回其值
        - ☐ front() —— 返回首部元素的值
        - ☐ back() —— 返回尾部元素的值
        - ☐ insert(index, value) —— 插入值到指定的索引，并把当前索引的元素指向到新的元素
        - ☐ erase(index) —— 删除指定索引的节点
        - ☐ value_n_from_end(n) —— 返回倒数第 n 个节点的值
        - ☐ reverse() —— 逆序链表
        - ☐ remove_value(value) —— 删除链表中指定值的第一个元素
    - ☐ 双向链表
        - 介绍（视频）
        - 并不需要实现

- 堆栈（Stack）
    - ☐ 堆栈（视频）
    - ☐ [Review] Stacks in 3 minutes (video)
    - ☐ 可以不实现，因为使用数组来实现是微不足道的事

- 队列（Queue）
    - ☐ 队列（视频）

- [ ] 原型队列/先进先出（FIFO）
- [ ] [Review] Queues in 3 minutes (video)
- [ ] 使用含有尾部指针的链表来实现:
  - enqueue(value) —— 在尾部添加值
  - dequeue() —— 删除最早添加的元素并返回其值（首部元素）
  - empty()
- [ ] 使用固定大小的数组实现：
  - enqueue(value) —— 在可容的情况下添加元素到尾部
  - dequeue() —— 删除最早添加的元素并返回其值
  - empty()
  - full()
- [ ] 花销：
  - 在糟糕的实现情况下，使用链表所实现的队列，其入列和出列的时间复杂度将会是 O(n)。因为，你需要找到下一个元素，以致循环整个队列
  - enqueue：O(1)（平摊（amortized）、链表和数组 [探测（probing）]）
  - dequeue：O(1)（链表和数组）
  - empty：O(1)（链表和数组）

- 哈希表（Hash table）
  - [ ] 视频：
    - [ ] 链式哈希表（视频）
    - [ ] Table Doubling 和 Karp-Rabin（视频）
    - [ ] Open Addressing 和密码型哈希（Cryptographic Hashing）（视频）
    - [ ] PyCon 2010：强大的字典（视频）
    - [ ] PyCon 2017：字典更强大（视频）
    - [ ] (高级) 随机化：通用和完美哈希（视频）
    - [ ] (进阶)完美哈希（Perfect hashing）（视频）
    - [ ] [复习]4分钟了解哈希表（视频）
  - [ ] 在线课程：
    - [ ] 核心哈希表（视频）
    - [ ] 数据结构（视频）
    - [ ] 电话簿问题（视频）
    - [ ] 分布式哈希表：
      - Dropbox中的即时上传和存储优化（视频）
      - 分布式哈希表（视频）
  - [ ] 使用线性探测法的数组实现
    - hash(k, m) - m是哈希表的大小
    - add(key, value) - 如果键已存在，则更新值
    - exists(key) - 检查键是否存在
    - get(key) - 获取给定键的值
    - remove(key) - 删除给定键的值

更多的知识

- 二分查找（Binary search）
  - [ ] 二分查找（视频）
  - [ ] 二分查找（视频）
  - [ ] 详情
  - [ ] 蓝图
  - [ ] 【复习】四分钟二分查找(视频)
  - [ ] 实现：
    - 二分查找（在一个已排序好的整型数组中查找）
    - 迭代式二分查找
- 按位运算（Bitwise operations）
  - [ ] Bits 速查表 — 你需要知道大量2的幂数值（从2^1 到 2^16 及 2^32）
  - [ ] 好好理解位操作符的含义：&、|、^、~、>>、<<
    - [ ] 字码（words）
    - [ ] 好的介绍： 位操作（视频）
    - [ ] C 语言编程教程 2-10：按位运算（视频）
    - [ ] 位操作
    - [ ] 按位运算
    - [ ] Bithacks
    - [ ] 位元抚弄者（The Bit Twiddler）
    - [ ] 交互式位元抚弄者（The Bit Twiddler Interactive）
    - [ ] 位操作技巧（Bit Hacks）（视频）
    - [ ] 练习位操作
  - [ ] 一补数和补码
    - 二进制：利 & 弊（为什么我们要使用补码）（视频）
    - 一补数（1s Complement）
    - 补码（2s Complement）
  - [ ] 计算置位（Set Bits）
    - 计算一个字节中置位（Set Bits）的四种方式（视频）
    - 计算比特位
    - 如何在一个 32 位的整型中计算置位（Set Bits）的数量
  - [ ] 交换值：
    - 交换（Swap）
  - [ ] 绝对值：
    - 绝对整型（Absolute Integer）

树（Trees）

- 树-介绍
  - [ ] 树的介绍（视频）
  - [ ] 树遍历（视频）
  - [ ] BFS（广度优先搜索）和DFS（深度优先搜索）（视频）
    - BFS 笔记
      - 层次遍历（BFS，使用队列）

        - 时间复杂度： O(n)
        - 空间复杂度：最佳情况：O(1)，最坏情况：O(n/2)=O(n)
    - DFS 笔记：
        - 时间复杂度：O(n)
        - 空间复杂度：
            - 最好情况：O(log n) - 树的平均高度
            - 最坏情况：O(n)
        - 中序遍历（DFS：左、节点本身、右）
        - 后序遍历（DFS：左、右、节点本身）
        - 先序遍历（DFS：节点本身、左、右）
- [ ] [复习]4分钟内的广度优先搜索（视频）
- [ ] [复习] 4分钟内的深度优先搜索（视频）
- [ ] [复习]11分钟内的树遍历（播放列表）（视频）

- 二叉查找树（Binary search trees）：BSTs
    - [ ] 二叉搜索树复习（视频）
    - [ ] 介绍（视频）
    - [ ] MIT（视频）
    - C/C++:
        - [ ] 二叉查找树 —— 在 C/C++ 中实现（视频）
        - [ ] BST 的实现 —— 在堆栈和堆中的内存分配（视频）
        - [ ] 在二叉查找树中找到最小和最大的元素（视频）
        - [ ] 寻找二叉树的高度（视频）
        - [ ] 二叉树的遍历 —— 广度优先和深度优先策略（视频）
        - [ ] 二叉树：层序遍历（视频）
        - [ ] 二叉树的遍历：先序、中序、后序（视频）
        - [ ] 判断一棵二叉树是否为二叉查找树（视频）
        - [ ] 从二叉查找树中删除一个节点（视频）
        - [ ] 二叉查找树中序遍历的后继者（视频）
    - [ ] 实现：
        - [ ] insert // 将值插入树中
        - [ ] get_node_count // 查找树上的节点数
        - [ ] print_values // 从小到大打印树中节点的值
        - [ ] delete_tree
        - [ ] is_in_tree // 如果值存在于树中则返回 true
        - [ ] get_height // 以节点为单位返回高度（单个节点的高度为1）
        - [ ] get_min // 返回树上的最小值
        - [ ] get_max // 返回树上的最大值
        - [ ] is_binary_search_tree
        - [ ] delete_value
        - [ ] get_successor // 返回给定值的后继者，若没有则返回-1

- 堆（Heap）/ 优先级队列（Priority Queue）/ 二叉堆（Binary Heap）
    - 以树形结构可视化，但通常在存储上是线性的（数组、链表）
    - [ ] 堆（Heap）

- 堆简介（视频）
- 二叉树（视频）
- 树高度备注（视频）
- 基本操作（视频）
- 完全二叉树（视频）
- 伪代码（视频）
- 堆排序 - 跳转到开始部分（视频）
- 堆排序（视频）
- 构建堆（视频）
- MIT：堆和堆排序（视频）
- CS 61B Lecture 24：优先队列（视频）
- 线性时间构建堆（大顶堆）
- [复习] 13分钟了解堆（视频）
- 实现一个大顶堆：
  - insert
  - sift_up —— 用于插入元素
  - get_max —— 返回最大值但不移除元素
  - get_size() —— 返回存储的元素数量
  - is_empty() —— 若堆为空则返回 true
  - extract_max —— 返回最大值并移除
  - sift_down —— 用于获取最大值元素
  - remove(i) —— 删除指定索引的元素
  - heapify —— 构建堆，用于堆排序
  - heap_sort() —— 拿到一个未排序的数组，然后使用大顶堆或者小顶堆进行就地排序

排序（Sorting）

- 笔记:
  - 实现各种排序，知道每种排序的最坏、最好和平均的复杂度分别是什么场景:
    - 不要用冒泡排序 - 效率太差 - 时间复杂度 O(n^2), 除非 n <= 16
  - 排序算法的稳定性 ("快排是稳定的么?")
    - 排序算法的稳定性
    - 排序算法的稳定性
    - 排序算法的稳定性
    - 排序算法 - 稳定性
  - 哪种排序算法可以用链表？哪种用数组？哪种两者都可？
    - 并不推荐对一个链表排序，但归并排序是可行的.
    - 链表的归并排序
- 关于堆排序，请查看前文堆的数据结构部分。堆排序很强大，不过是非稳定排序。
- Sedgewick —— 归并排序（5个视频）
  - 1. 归并排序（Mergesort）
  - 2. 自底向上的归并排序（Bottom up Mergesort）
  - 3. 排序复杂性（Sorting Complexity）
  - 4. 比较器（Comparators）

    - [ ] 5. 稳定性（Stability）
- [ ] Sedgewick — 快速排序（4个视频）
    - [ ] 1. 快速排序（Quicksort）
    - [ ] 2. 选择排序（Selection）
    - [ ] 3. 重复键（Duplicate Keys）
    - [ ] 4. 系统排序（System Sorts）
- [ ] 加州大学伯克利分校：
    - [ ] CS 61B Lecture 29：排序 I（视频）
    - [ ] CS 61B Lecture 30：排序 II（视频）
    - [ ] CS 61B Lecture 32：排序 III（视频）
    - [ ] CS 61B Lecture 33：排序 V（视频）
    - [ ] CS 61B 2014-04-21：基数排序（视频）
- [ ] 冒泡排序（视频）
- [ ] 冒泡排序分析（视频）
- [ ] 插入排序 & 归并排序（视频）
- [ ] 插入排序（视频）
- [ ] 归并排序（视频）
- [ ] 快排（视频）
- [ ] 选择排序（视频）
- [ ] 归并排序代码：
    - [ ] 使用外部数组（C语言）
    - [ ] 使用外部数组（Python语言）
    - [ ] 对原数组直接排序（C++）
- [ ] 快速排序代码：
    - [ ] 实现（C语言）
    - [ ] 实现（C语言）
    - [ ] 实现（Python语言）
- [ ] [Review] Sorting (playlist) in 18 minutes
    - [ ] Quick sort in 4 minutes (video)
    - [ ] Heap sort in 4 minutes (video)
    - [ ] Merge sort in 3 minutes (video)
    - [ ] Bubble sort in 2 minutes (video)
    - [ ] Selection sort in 3 minutes (video)
    - [ ] Insertion sort in 2 minutes (video)
- [ ] 实现:

    - [ ] 归并：平均和最差情况的时间复杂度为 O(n log n)。
    - [ ] 快排：平均时间复杂度为 O(n log n)。
    - 选择排序和插入排序的最坏、平均时间复杂度都是 O(n^2)。
    - 关于堆排序，请查看前文堆的数据结构部分。

- [ ] 有兴趣的话，还有一些补充，但并不是必须的:

    - Sedgewick—基数排序 (6个视频)
        - [ ] 1. Java 中的字符串
        - [ ] 2. 键值索引计数（Key Indexed Counting）
        - [ ] 3. Least Significant Digit First String Radix Sort
        - [ ] 4. Most Significant Digit First String Radix Sort
        - [ ] 5. 3中基数快速排序
        - [ ] 6. 后继数组（Suffix Arrays）
    - [ ] 基数排序
    - [ ] 基数排序（视频）
    - [ ] 基数排序, 计数排序 (线性时间内)（视频）
    - [ ] 随机算法: 矩阵相乘, 快排, Freivalds' 算法（视频）
    - [ ] 线性时间内的排序（视频）

总结一下，这是15种排序算法的可视化表示。 如果你需要有关此主题的更多详细信息，请参阅“一些主题的额外内容”中的“排序”部分。

图（Graphs）

图表可以用来表示计算机科学中的许多问题，所以这一部分很长，就像树和排序一样。

- 笔记:

    - 有4种基本方式在内存里表示一个图:
        - 对象和指针
        - 邻接矩阵
        - 邻接表
        - 邻接图
    - 熟悉以上每一种图的表示法，并了解各自的优缺点
    - 广度优先搜索和深度优先搜索：知道它们的计算复杂度和设计上的权衡以及如何用代码实现它们
    - 遇到一个问题时，首先尝试基于图的解决方案，如果没有再去尝试其他的。

- MIT（视频）：

    - 广度优先搜索
    - 深度优先搜索

- [ ] Skiena 教授的课程 - 很不错的介绍:

    - [ ] CSE373 2012 - 课程 11 - 图的数据结构（视频）
    - [ ] CSE373 2012 - 课程 12 - 广度优先搜索（视频）
    - [ ] CSE373 2012 - 课程 13 - 图的算法（视频）
    - [ ] CSE373 2012 - 课程 14 - 图的算法 (1)（视频）
    - [ ] CSE373 2012 - 课程 15 - 图的算法 (2)（视频）

    - [ ] CSE373 2012 - 课程 16 - 图的算法 (3)（视频）

- [ ] 图 (复习和其他):
    - [ ] 6.006 单源最短路径问题（视频）
    - [ ] 6.006 Dijkstra算法（视频）
    - [ ] 6.006 Bellman-Ford算法（视频）
    - [ ] 6.006 加速Dijkstra算法（视频）
    - [ ] Aduni：图算法 I - 拓扑排序，最小生成树，Prim算法 - 讲座6（视频）
    - [ ] Aduni：图算法 II - DFS，BFS，Kruskal算法，Union Find数据结构 - 讲座7（视频）
    - [ ] Aduni：图算法 III：最短路径 - 讲座8（视频）
    - [ ] Aduni：图算法 IV：几何算法入门 - 讲座9（视频）
    - [ ] CS 61B 2014：加权图（视频）
    - [ ] 贪婪算法：最小生成树（视频）
    - [ ] 强连通分量Kosaraju算法图算法（视频）
    - [ ] [复习] 最短路径算法（播放列表）16分钟（视频）
    - [ ] [复习] 最小生成树（播放列表）4分钟（视频）
- 完整的 Coursera 课程:
    - [ ] 图的算法（视频）
- 我会实现:
    - [ ] DFS 邻接表 (递归)
    - [ ] DFS 邻接表 (栈迭代)
    - [ ] DFS 邻接矩阵 (递归)
    - [ ] DFS 邻接矩阵 (栈迭代)
    - [ ] BFS 邻接表
    - [ ] BFS 邻接矩阵
    - [ ] 单源最短路径问题 (Dijkstra)
    - [ ] 最小生成树
    - 基于 DFS 的算法 (根据上文 Aduni 的视频):
        - [ ] 检查环 (我们会先检查是否有环存在以便做拓扑排序)
        - [ ] 拓扑排序
        - [ ] 计算图中的连通分支
        - [ ] 列出强连通分量
        - [ ] 检查双向图

更多知识

- 递归（Recursion）
    - [ ] Stanford 大学关于递归 & 回溯的课程:
        - [ ] 课程 8 | 抽象编程（视频）
        - [ ] 课程 9 | 抽象编程（视频）
        - [ ] 课程 10 | 抽象编程（视频）
        - [ ] 课程 11 | 抽象编程（视频）
    - 什么时候适合使用

    - 尾递归会更好么?
        - [ ] 什么是尾递归以及为什么它如此糟糕?
        - [ ] 尾递归（视频）
    - [ ] 解决任何递归问题的5个简单步骤（视频）

    回溯蓝图: Java Python

- 动态规划（Dynamic Programming）

    - 在你的面试中或许没有任何动态规划的问题， 但能够知道一个题目可以使用动态规划来解决是很重要的。
    - 这一部分会有点困难，每个可以用动态规划解决的问题都必须先定义出递推关系，要推导出来可能会有点棘手。
    - 我建议先阅读和学习足够多的动态规划的例子，以便对解决 DP 问题的一般模式有个扎实的理解。
    - [ ] 视频:
        - [ ] Skiena：CSE373 2020 - 讲座19 - 动态规划简介（视频）
        - [ ] Skiena：CSE373 2020 - 讲座20 - 编辑距离（视频）
        - [ ] Skiena：CSE373 2020 - 讲座20 - 编辑距离（续）（视频）
        - [ ] Skiena：CSE373 2020 - 讲座21 - 动态规划（视频）
        - [ ] Skiena：CSE373 2020 - 讲座22 - 动态规划和复习（视频）
        - [ ] Simonson：动态规划 0（从59:18开始）（视频）
        - [ ] Simonson：动态规划 I - 第11讲（视频）
        - [ ] Simonson：动态规划 II - 第12讲（视频）
        - [ ] 单独的动态规划问题列表（每个都很短）: 动态规划（视频）
    - [ ] 耶鲁课程笔记:
        - [ ] 动态规划
    - [ ] Coursera 课程:
        - [ ] RNA 二级结构问题（视频）
        - [ ] 动态规划算法（视频）
        - [ ] DP 算法描述（视频）
        - [ ] DP 算法的运行时间（视频）
        - [ ] DP vs 递归实现（视频）
        - [ ] 全局成对序列排列（视频）
        - [ ] 本地成对序列排列（视频）

- 设计模式

    - [ ] UML 统一建模语言概览 (视频)
    - [ ] 主要有如下的设计模式:
        - [ ] 策略模式（strategy）
        - [ ] 单例模式（singleton）
        - [ ] 适配器模式（adapter）
        - [ ] 原型模式（prototype）
        - [ ] 装饰器模式（decorator）
        - [ ] 访问者模式（visitor）
        - [ ] 工厂模式，抽象工厂模式（factory, abstract factory）
        - [ ] 外观模式（facade）

        - [ ] 观察者模式（observer）
        - [ ] 代理模式（proxy）
        - [ ] 委托模式（delegate）
        - [ ] 命令模式（command）
        - [ ] 状态模式（state）
        - [ ] 备忘录模式（memento）
        - [ ] 迭代器模式（iterator）
        - [ ] 组合模式（composite）
        - [ ] 享元模式（flyweight）
    - [ ] 系列视频（27个）
    - [ ] 书籍：《Head First设计模式》
        - I know the canonical book is "Design Patterns: Elements of Reusable Object-Oriented Software", but Head First is great for beginners to OO.
    - Handy reference: 101 Design Patterns & Tips for Developers

- 组合（Combinatorics）（n 中选 k 个) & 概率（Probability）

    - [ ] 数据技巧: 如何找出阶乘、排列和组合(选择)（视频）
    - [ ] 来点学校的东西: 概率（视频）
    - [ ] 来点学校的东西: 概率和马尔可夫链（视频）
    - [ ] 可汗学院:
        - 课程设置:
            - [ ] 概率理论基础
        - 只有视频 - 41 (每一个都短小精悍):
            - [ ] 概率解释（视频）

- NP, NP-Completeness和近似算法

    - 知道最经典的一些 NP-Completeness 问题，比如旅行商问题和背包问题， 而且能在面试官试图忽悠你的时候识别出他们。
    - 知道 NP-Completeness 是什么意思.
    - [ ] 计算复杂度（视频）
    - [ ] Simonson:
        - [ ] 贪心算法. II & 介绍 NP-Completeness（视频）
        - [ ] NP-Completeness II & 归约（视频）
        - [ ] NP-Completeness III（视频）
        - [ ] NP-Completeness IV（视频）
    - [ ] Skiena:
        - [ ] CSE373 2012 - 课程 23 - 介绍 NP-Completeness IV（视频）
        - [ ] CSE373 2012 - 课程 24 - NP-Completeness证明（视频）
        - [ ] CSE373 2012 - 课程 25 - NP-Completeness挑战（视频）
        - [ ] CSE373 2020年 - 第26讲 - NP-Completeness挑战（视频）
    - [ ] 复杂度: P, NP, NP-完全性, 规约（视频）
    - [ ] 复杂度: 近视算法 Algorithms（视频）
    - [ ] 复杂度: 固定参数算法（视频）
    - Peter Norvik 讨论旅行商问题的近似最优解:
        - Jupyter 笔记本

    - 《算法导论》（CLRS）的第 1048 - 1140 页。

- 计算机如何处理程序

    - ☐ CPU如何执行程序（视频）
    - ☐ 计算机如何进行计算 - 算术逻辑单元（视频）
    - ☐ 寄存器和RAM（视频）
    - ☐ 中央处理器（CPU）（视频）
    - ☐ 指示和程序（视频）

- 缓存（Cache）

    - ☐ LRU 缓存:
        - ☐ LRU 的魔力 (100 Days of Google Dev)（视频）
        - ☐ 实现 LRU（视频）
        - ☐ LeetCode - 146 LRU Cache (C++)（视频）
    - ☐ CPU 缓存:
        - ☐ MIT 6.004 L15: 存储体系（视频）
        - ☐ MIT 6.004 L16: 缓存的问题（视频）

- 进程（Processe）和线程（Thread）

    - ☐ 计算机科学 162 - 操作系统 (25 个视频):
        - 视频 1-11 是关于进程和线程
        - 操作系统和系统编程（视频）
    - 进程和线程的区别是什么?
    - 涵盖了:
        - 进程、线程、协程
            - 进程和线程的区别
            - 进程
            - 线程
            - 锁
            - 互斥
            - 信号量
            - 监控
            - 他们是如何工作的
            - 死锁
            - 活锁
        - CPU 活动, 中断, 上下文切换
        - 现代多核处理器的并发式结构
        - 分页（paging），分段（segmentation）和虚拟内存（视频）
        - 中断（视频）
        - 进程资源需要（内存：代码、静态存储器、栈、堆、文件描述符、I/O）
        - 线程资源需要（在同一个进程内和其他线程共享以上（除了栈）的资源，但是每个线程都有独立的程序计数器、栈计数器、寄存器和栈）
        - Fork 操作是真正的写时复制（只读），直到新的进程写到内存中，才会生成一份新的拷贝。
        - 上下文切换

        - 操作系统和底层硬件如何启动上下文切换?
    - C++ 的线程 (系列 - 10 个视频)
    - CS 377 春季'14：马萨诸塞大学的操作系统
    - Python 的并发 (视频):
        - 线程系列
        - Python 线程
        - 理解 Python 的 GIL (2010)
            - 参考
        - David Beazley - Python 协程 - PyCon 2015
        - Keynote David Beazley - 兴趣主题 (Python 异步 I/O)
        - Python 中的互斥

- 测试
    - 涵盖了:
        - 单元测试是如何工作的
        - 什么是模拟对象
        - 什么是集成测试
        - 什么是依赖注入
    - James Bach 讲敏捷软件测试（视频）
    - James Bach 软件测试公开课（视频）
    - Steve Freeman - 测试驱动的开发（视频）
        - slides
    - 依赖注入:
        - 视频
        - 测试之道
    - 如何编写测试

- 字符串搜索和操作
    - Sedgewick——后缀数组（Suffix Arrays）（视频）
    - Sedgewick——子字符串搜寻（视频）
        - 1. 子字符串搜寻导论
        - 2. 子字符串搜寻——暴力法
        - 3. KMP算法
        - 4. Boyer-Moore算法
        - 5. Rabin-Karp算法
    - 文本的搜索模式（视频）

如果你需要有关此主题的更多详细信息，请参阅“一些主题的额外内容”中的“字符串匹配”部分。

- 字典树（Tries）
    - 需要注意的是，字典树各式各样。有些有前缀，而有些则没有。有些使用字符串而不使用比特位来追踪路径。
    - 阅读代码，但不实现。
    - Sedgewick——字典树（3个视频）
        - 1. R Way字典树

        - [ ] 2. 三元搜索树
        - [ ] 3. 基于字符串的操作
    - [ ] 数据结构笔记及编程技术
    - [ ] 短课程视频：
        - [ ] 对字典树的介绍（视频）
        - [ ] 字典树的性能（视频）
        - [ ] 实现一棵字典树（视频）
    - [ ] 字典树：一个被忽略的数据结构
    - [ ] TopCoder —— 使用字典树
    - [ ] 标准教程（现实中的用例）（视频）
    - [ ] MIT，高阶数据结构，字符串（视频中间有点困难）（视频）

- 浮点数
    - [ ] 简单的8位: 浮点数的表示 - 1（视频 - 计算中有错误 - 请查看视频描述）

- Unicode
    - [ ] 每一个软件开发者的绝对最低限度，必须要知道的关于 Unicode 和字符集知识
    - [ ] 关于处理文本需要的编码和字符集，每个程序员绝对需要知道的知识

- 字节序（Endianness）
    - 大/小端序
    - 大端序 Vs 小端序（视频）
    - 由里入内的大端序与小端序（视频）
        - 对于内核开发非常具有技术性，如果大多数的内容听不懂也没关系。
        - 前半部就已经足够了。

- 网络（视频）
    - **如果你具有网络经验或想成为可靠性工程师或运维工程师，期待你的问题**
    - 知道这些有益无害，多多益善!
    - [ ] 可汗学院
    - [ ] UDP 和 TCP：网络传输协议中的数据压缩（视频）
    - [ ] TCP/IP 和 OSI 模型解释！（视频）
    - [ ] 互联网上的数据包传输。网络和 TCP/IP 教程。（视频）
    - [ ] HTTP（视频）
    - [ ] SSL 和 HTTPS（视频）
    - [ ] SSL/TLS（视频）
    - [ ] HTTP 2.0（视频）
    - [ ] 视频系列（21个视频）
    - [ ] 子网络解密 - 第五部分 经典内部域名指向 CIDR 标记（视频）
    - [ ] 套接字（Sockets）：
        - Java——套接字——介绍（视频）
        - 套接字编程（视频）

最终复习

> 本节将包含一系列短视频，您可以迅速观看，以便复习大部分重要概念。
> 如果您经常需要温习知识，这会很有帮助。

- [ ] 一系列2-3分钟的短主题视频（共23个视频）
  - 视频链接
- [ ] 一系列2-5分钟的短主题视频 - Michael Sambol（共48个视频）：
  - 视频链接
  - 代码示例
- [ ] Sedgewick的算法课程视频 - 算法I
- [ ] Sedgewick的算法课程视频 - 算法II

更新你的简历

- 在书籍《Cracking The Coding Interview》和《Programming Interviews Exposed》中查看简历准备信息。
- “这就是一个优秀简历的样子” by Gayle McDowell（《Cracking the Coding Interview》的作者），
  - 作者备注：“这是针对美国的简历。印度和其他国家的简历有不同的期望，尽管许多要点是相同的。”
- “逐步简历指南” by Tech Interview Handbook
  - 详细指南，教您如何从零开始设置您的简历，编写有效的简历内容，优化它，并测试您的简历。

面试流程与一般面试准备

- [ ] 如何在2021年通过工程师面试
- [ ] 揭秘技术招聘过程
- [ ] 如何在四大科技巨头公司中找到工作：
  - [ ] 如何在四大科技巨头公司中找到工作 - 亚马逊、Facebook、谷歌和微软（视频）
  - [ ] 如何在四大科技巨头公司中找到工作1（后续视频）
- [ ] 《破解编程面试》第一集：
  - [ ] Gayle L McDowell - Cracking The Coding Interview（视频）
  - [ ] 与作者Gayle Laakmann McDowell一起破解编程面试（视频）
- [ ] 破解Facebook编程面试：
  - [ ] 方法论
  - [ ] 问题演示
- 面试准备课程：
  - 数据结构、算法和面试的Python课程（付费课程）：：
    - 以Python为中心的面试准备课程，涵盖数据结构、算法、模拟面试等内容。
  - 使用Python的数据结构和算法简介（Udacity免费课程）：：
    - 一个免费的以Python为中心的数据结构和算法课程。
  - 数据结构和算法纳米学位！（Udacity付费纳米学位）：：
    - 提供超过100个数据结构和算法练习的实际操作体验，并得到专属导师的指导，以帮助您为面试和实际工作做准备。
  - Grokking行为面试（Educative免费课程）：：

        - 很多时候，阻碍您获得梦想工作的不是您的技术能力，而是您在行为面试中的表现。
    - AlgoMonster（付费课程，提供免费内容）：):
        - LeetCode的速成课程。涵盖了从成千上万的问题中提炼出的所有模式。

模拟面试：

- Gainlo.co：来自大公司的模拟面试官 - 我用过这个，帮助我放松进行电话和现场面试。
- Pramp：与同行进行模拟面试 - 同行模式的实践面试。
- interviewing.io：与资深工程师进行模拟面试 - 匿名算法/系统设计面试，与FAANG公司的资深工程师进行。
- Meetapro：与顶级FAANG面试官进行模拟面试 - 类似Airbnb的模拟面试/指导平台。
- Hello Interview：与专家教练和人工智能模拟面试 - 直接与人工智能或 FAANG 员工工程师和经理面试。
- Codemia：通过人工智能或社区解决方案和反馈来练习系统设计问题 - 通过AI练习工具来解决系统设计问题。与社区分享你的解决方案，以获得反馈。.

当面试来临的时候

随着下面列举的问题思考下你可能会遇到的 20 个面试问题，每个问题准备 2-3 种回答。 准备点故事，不要只是摆一些你完成的事情的数据，相信我，人人都喜欢听故事。

- 你为什么想得到这份工作?
- 你解决过的最有难度的问题是什么?
- 面对过的最大挑战是什么?
- 见过的最好或者最坏的设计是怎么样的?
- 对某个产品提出改进建议。
- 你作为一个个体同时也是团队的一员，如何达到最好的工作状态?
- 你的什么技能或者经验是你的角色中不可或缺的，为什么?
- 你在某份工作或某个项目中最享受的是什么?
- 你在某份工作或某个项目中面临过的最大挑战是什么?
- 你在某份工作或某个项目中遇到过的最硬的 Bug 是什么样的?
- 你在某份工作或某个项目中学到了什么?
- 你在某份工作或某个项目中哪些地方还可以做的更好?

问面试官的问题

我会问的一些：(可能我已经知道了答案但我想听听面试官的看法或者了解团队的前景):

- 团队多大规模?
- 开发周期是怎样的? 会使用瀑布流/极限编程/敏捷开发么?
- 经常会为截止日期（deadlines）加班么? 或者是有弹性的?
- 团队里怎么做技术选型?
- 每周平均开多少次会?
- 你觉得工作环境有助于员工集中精力吗?
- 目前正在做什么工作?
- 喜欢这些事情吗?
- 工作期限是怎么样的?
- 工作生活怎么平衡?

当你获得了梦想的职位

恭喜你!

继续学习。

活到老，学到老。

```
**********************************************************************************
********************
**********************************************************************************
********************
```

下面的内容都是可选的。
通过学习这些内容，你将会得到更多的有关 CS 的概念，并将为所有的软件工程工作做更好的准备。
你将会成为一个更全面的软件工程师。

```
**********************************************************************************
********************
**********************************************************************************
********************
```

额外书籍

你可以从以下的书单挑选你有兴趣的主题来研读。

- UNIX环境高级编程
  - 老，但却很棒
- Linux 命令行大全
  - 现代选择
- TCP-IP详解系列
- Head First 设计模式
  - 设计模式入门介绍
- 设计模式：可复用面向对象软件的基础
  - 也被称为"四人帮"（Gang of Four(GOF)）
  - 经典设计模式书籍
- 算法设计手册（Skiena）
  - 作为复习以及问题辨别
  - 这本书中算法的部分难度已经超过面试会出现的
  - 本书分为两个部分:
    - 数据结构和算法课本
      - 优点:
        - 跟其他算法课本一样是个很棒的复习素材
        - 包含作者以往解决工业及学术上问题的经验的故事
        - 含C语言代码示例

        - 缺点:
            - 某些地方跟《算法导论》（CLRS）一样艰深，但在某些主题，算法导论或许是更好的选择。
            - 第7、8、9章有点难以消化，因为某些地方并没有解释得很清楚，或者根本上我就是个学渣
            - 别会错意了，我很喜欢 Skiena 的教学方法以及他的风格。
        - 算法目录:
            - 这个部分是买这本书的最大原因
            - 我即将着手进行这部分，一旦完成这部分我会再更新上来
    - 可以在 kindle 上租
    - 解答:
        - 解答
    - 勘误表
- 算法 (Jeff Erickson)
- 编程卓越之道（第一卷）：深入理解计算机
    - 该书于2004年出版，虽然有些过时，但是对于简单了解计算机而言，这是一个了不起的资源
    - 作者发明了高阶组合语言 HLA，所以提到，并且举了一些HLA的例子。里面没有用到很多，但都是很棒的组合语言的例子。
    - 这些章节值得阅读，为你提供良好的基础:
        - 第2章——数字表示
        - 第3章——二进制算术和位运算
        - 第4章——浮点表示
        - 第5章——字符表示
        - 第6章——内存组织和访问
        - 第7章——组合数据类型和内存对象
        - 第9章——CPU体系结构
        - 第10章——指令集架构
        - 第11章——内存体系结构和组织
- 算法导论
    - **重要提示**：读这本书的价值有限。本书很好地回顾了算法和数据结构，但不会教你如何编写良好的代码。你必须能够有效地编写一个不错的解决方案
    - 又称 CLR，有时是 CLRS，因为 Stein 最后才加入
- 计算机体系结构，第六版：定量方法
    - 对于更丰富、更时新（2017年）但较长的处理方式

系统设计、可扩展性和数据处理

如果您有4年以上的工作经验，可以预期会遇到系统设计问题。

- 可扩展性和系统设计是一个非常广泛的主题，涵盖了许多内容和资源， 因为在设计一个可以扩展的软件/硬件系统时需要考虑很多因素。 预计需要花费相当多的时间来学习这方面的知识。
- 考虑要点:
    - 可扩展性
        - 将大数据集归纳为单一值
        - 将一个数据集转换为另一个数据集
        - 处理海量数据
    - 系统设计

        - 功能集
        - 接口
        - 类层次结构
        - 在特定约束下设计系统
        - 简单性和鲁棒性
        - 权衡
        - 性能分析和优化
- [ ] **从这里开始**: The System Design Primer
- [ ] HiredInTech的系统设计
- [ ] 如何准备回答技术面试中的设计问题?
- [ ] 通过8个步骤掌握系统设计面试
- [ ] 数据库规范化 - 第一范式、第二范式、第三范式和第四范式（视频）
- [ ] 系统设计面试 - 这个资源有很多内容。浏览文章和示例。我列出了一些示例在下面。
- [ ] 如何在系统设计面试中脱颖而出
- [ ] 每个人都应该了解的数字
- [ ] 进行上下文切换需要多长时间?
- [ ] 跨数据中心的事务（视频）
- [ ] CAP定理的简明英文介绍
- [ ] MIT 6.824：分布式系统，2020年春季（20个视频）
- [ ] 共识算法:
    - [ ] Paxos - Paxos协议 - Computerphile（视频）
    - [ ] Raft - Raft分布式共识算法简介（视频）
        - [ ] 易于理解的论文
        - [ ] 信息图
- [ ] 一致性哈希
- [ ] NoSQL模式
- [ ] 可扩展性:
    - 您不需要掌握所有这些内容，只需选择一些您感兴趣的。
    - [ ] 优秀的概述（视频）
    - [ ] 短系列:
        - 克隆
        - 数据库
        - 缓存
        - 异步性
    - [ ] 可扩展的Web架构和分布式系统
    - [ ] 分布式计算的谬误解释
    - [ ] Jeff Dean - 在Google构建软件系统以及吸取的教训（视频）
    - [ ] 架构师为规模而设计的介绍
    - [ ] 缩放移动游戏以面向全球受众使用App Engine和Cloud Datastore（视频）
    - [ ] 谷歌是如何进行面向全球基础设施的大规模工程的（视频）
    - [ ] 算法的重要性
    - [ ] 分片
    - [ ] 针对长期目标的工程 - Astrid Atkinson主题演讲（视频）
    - [ ] 在30分钟内了解YouTube 7年的可扩展性经验
        - 视频
    - [ ] PayPal如何使用仅8台VM每天处理数十亿次交易

    - [ ] 如何在大型数据集中去重
    - [ ] 通过Jon Cowie深入了解Etsy的规模和工程文化（视频）
    - [ ] Amazon是如何转向自己的微服务架构的
    - [ ] 压缩还是不压缩，这是Uber面临的问题
    - [ ] 何时应使用近似查询处理?
    - [ ] 谷歌从单一数据中心到故障转移再到本地多家数据中心架构的转变
    - [ ] 为每天处理数百万请求的图像优化技术
    - [ ] Patreon架构简介
    - [ ] 如何在Instagram庞大的推荐引擎中决定您将看到谁?
    - [ ] 现代缓存设计
    - [ ] 在Facebook规模下进行直播视频流
    - [ ] 在亚马逊AWS上如何扩展到1100万以上的用户
    - [ ] 全面了解Netflix整个堆栈
    - [ ] 延迟无处不在，而且它会让您丧失销售机会 - 如何应对
    - [ ] Instagram的动力：数百个实例，几十种技术
    - [ ] Salesforce架构 - 如何处理每天13亿次交易
    - [ ] ESPN规模上的架构 - 每秒操作10万次“嘟嘟噜嘟嘟噜”
    - [ ] 在下面的“消息、序列化和队列系统”部分查看一些将服务连接在一起的技术信息
    - [ ] Twitter:
        - O'Reilly MySQL CE 2011: Jeremy Cole, "Big and Small Data at @Twitter" (视频)
        - 时间轴扩展
    - 欲知更多信息，请参阅Video Series 部分中的“Mining Massive Datasets”视频系列
- [ ] 练习系统设计过程：以下是一些建议您在纸上尝试的想法，每个想法都有一些关于如何在现实世界中处理的文档：
    - 复习: The System Design Primer
    - HiredInTech的系统设计
    - 速查表
    - 流程：
        1. 理解问题和范围：
            - 定义用例，与面试官的帮助
            - 提出额外的功能
            - 移除面试官认为超出范围的项目
            - 假设需要高可用性，并将其添加为用例
        2. 考虑限制：
            - 询问每月有多少个请求
            - 询问每秒有多少个请求（他们可能会主动提供或让您计算）
            - 估计读取与写入的百分比
            - 保持估计时考虑80/20法则
            - 每秒写入多少数据
            - 在5年内所需的总存储量
            - 每秒读取多少数据
        3. 抽象设计：
            - 层（服务、数据、缓存）
            - 基础架构：负载均衡、消息传递
            - 驱动服务的任何关键算法的粗略概述
            - 考虑瓶颈并确定解决方案

    - 练习：
        - 设计一个随机唯一ID生成系统
        - 设计一个键值数据库
        - 设计一个图片分享系统
        - 设计一个推荐系统
        - 设计一个URL缩短系统：来自上面的复制
        - 设计一个缓存系统

附加学习

> 我把它们加进来是为了让你成为更全方位的软件工程师，并且留意一些技术以及算法，让你拥有更大的工具箱。

- 编译器
    - 编译器的工作方式，约1分钟（视频）
    - Harvard CS50-编译器（视频）
    - C ++（视频）
    - 了解编译器优化（C ++）（视频）
- Emacs and vi(m)
    - 熟悉基于 unix 的代码编辑器
    - vi(m):
        - 使用 vim 进行编辑 01 - 安装, 设置和模式 (视频)
        - VIM 的冒险之旅
        - 4 个视频集:
            - vi/vim 编辑器 - 课程 1
            - vi/vim 编辑器 - 课程 2
            - vi/vim 编辑器 - 课程 4
            - vi/vim 编辑器 - 课程 3
        - 使用 Vi 而不是 Emacs
    - emacs:
        - 基础 Emacs 教程 (视频)
        - 3 个视频集:
            - Emacs 教程 (初学者) -第 1 部分- 文件命令, 剪切/复制/粘贴, 自定义命令
            - Emacs 教程 (初学者 -第 2 部分- Buffer 管理, 搜索, M-x grep 和 rgrep 模式
            - Emacs 教程 (初学者 -第 3 部分- 表达式, 声明, ~/.emacs 文件和包机制
        - Evil 模式: 或许, 我是怎样对 Emacs 路人转粉的 (视频)
        - 使用 Emacs 开发 C 程序
    - Emacs 绝对初学者指南（David Wilson的视频）
    - Emacs 绝对初学者指南（David Wilson 批注）
- Unix 命令行工具
    - 下列内容包含优秀工具

    - bash
    - cat
    - grep
    - sed
    - awk
    - curl or wget
    - sort
    - tr
    - uniq
    - strace
    - tcpdump

- 信息论 (视频)

    - Khan Academy 可汗学院
    - 更多有关马尔可夫的内容:
        - 马尔可夫内容生成（Core Markov Text Generation）
        - Core Implementing Markov Text Generation马尔可夫内容生成实现
        - 一个马尔可夫内容生成器的项目（Project = Markov Text Generation Walk Through）
    - 关于更多信息，请参照下方 MIT 6.050J 信息和系统复杂度的内容。

- 奇偶校验位 & 汉明码 (视频)

    - 入门
    - 奇偶校验位
    - 汉明码(Hamming Code):
        - 发现错误
        - 修正错误
    - 检查错误

- 系统熵值（Entropy）

    - 请参考下方视频
    - 观看之前，请先确定观看了信息论的视频
    - 信息理论, 克劳德·香农, 熵值, 系统冗余, 数据比特压缩 (视频)

- 密码学

    - 请参考下方视频
    - 观看之前，请先确定观看了信息论的视频
    - 可汗学院
    - 密码学: 哈希函数
    - 密码学: 加密

- 压缩

    - 观看之前，请先确定观看了信息论的视频
    - Computerphile (视频)：

        - 压缩
        - 压缩熵值
        - 由上而下的树 (霍夫曼编码树)
        - 额外比特 - 霍夫曼编码树
        - 优雅的压缩数据 (无损数据压缩方法)
        - Text Compression Meets Probabilities
    - 数据压缩的艺术
    - (可选) 谷歌开发者：GZIP 还差远了呢!

- 计算机安全

    - MIT（23个视频）
        - 威胁模型：入门
        - 控制劫持攻击
        - 缓冲区溢出漏洞攻击和防御
        - 优先权区分
        - 能力
        - 在沙盒中运行原生代码
        - 网络安全模型
        - 网络安全应用
        - 标志化执行
        - 网络安全
        - 网络协议
        - 旁路攻击

- 垃圾回收

    - [ ] Python 中的垃圾回收 (视频)
    - [ ] 深度解析：论垃圾回收在 JAVA 中的重要性
    - [ ] 深度解析：论垃圾回收在 Python 中的重要性(视频)

- 并行编程

    - [ ] Coursera (Scala)
    - [ ] 用于高性能并行计算的高效Python（视频）

- 消息传递，序列化和队列系统

    - Thrift
        - 教程
    - 协议缓冲
        - 教程
    - gRPC
        - gRPC 对于JAVA开发者的入门教程（视频）
    - Redis
        - 教程
    - Amazon的 SQS 系统 (队列)
    - Amazon的 SNS 系统 (pub-sub)

    - RabbitMQ
        - 入门教程
    - Celery
        - Celery入门
    - ZeroMQ
        - 入门教程
    - ActiveMQ
    - Kafka
    - MessagePack
    - Avro

- A*搜索算法

    - A 搜索算法
    - A* 路径搜索（E01：算法解释）（视频）

- 快速傅里叶变换

    - 傅立叶变换的交互式指南
    - 什么是傅立叶变换？论傅立叶变换的用途
    - 什么是傅立叶变换？（视频）
    - 分而治之：FFT（视频）
    - FTT 是什么

- 布隆过滤器

    - 给定布隆过滤器m比特位和k个哈希函数，插入和成员检测都会是 O(k)。
    - 布隆过滤器（视频）
    - 布隆过滤器 | 数据挖掘 | Stanford University（视频）
    - 教程
    - 如何写一个布隆过滤器应用

- HyperLogLog

    - 如何仅使用1.5KB内存计算十亿个不同的对象

- 局部敏感哈希

    - 用于确定文件的相似性
    - MD5 或 SHA 的反义词，用于确定2个文档/字符串是否完全相同
    - Simhashing（希望如此）变得简单

- van Emde Boas 树

    - 分而治之：van Emde Boas 树 (视频)
    - MIT课堂笔记

- 增强数据结构

    - CS 61B 第 39 课: 增强数据结构

- 平衡查找树（Balanced search trees）

    - 掌握至少一种平衡查找树（并懂得如何实现）：

    - “在各种平衡查找树当中，AVL 树和2-3树已经成为了过去，而红黑树（red-black trees）看似变得越来越受人青睐。 这种令人特别感兴趣的数据结构，亦称伸展树（splay tree）。 它可以自我管理，且会使用轮换来移除任何访问过根节点的键。” —— Skiena

    - 因此，在各种各样的平衡查找树当中，我选择了伸展树来实现。 虽然，通过我的阅读，我发现在面试中并不会被要求实现一棵平衡查找树。 但是，为了胜人一筹，我们还是应该看看如何去实现。在阅读了大量关于红黑树的代码后， 我才发现伸展树的实现确实会使得各方面更为高效。

        - 伸展树：插入、查找、删除函数的实现，而如果你最终实现了红黑树，那么请尝试一下：
        - 跳过删除函数，直接实现搜索和插入功能

    - 我希望能阅读到更多关于 B 树的资料，因为它也被广泛地应用到大型的数据集当中。

    - 自平衡二叉查找树

    - **AVL 树**

        - 实际中： 我能告诉你的是，该种树并无太多的用途，但我能看到有用的地方在哪里： AVL 树是另一种平衡查找树结构。其可支持时间复杂度为 O(log n) 的查询、插入及删除。 它比红黑树严格意义上更为平衡，从而导致插入和删除更慢，但遍历却更快。正因如此，才彰显其结构的魅力。 只需要构建一次，就可以在不重新构造的情况下读取， 适合于实现诸如语言字典（或程序字典，如一个汇编程序或解释程序的操作码）。
        - MIT AVL 树 / AVL 树的排序（视频）
        - AVL 树（视频）
        - AVL 树的实现（视频）
        - 分离与合并
        - [Review] AVL Trees (playlist) in 19 minutes (video)

    - **伸展树**

        - 实际中： 伸展树一般用于缓存、内存分配者、路由器、垃圾回收者、数据压缩、ropes（字符串的一种替代品，用于存储长串的文本字符）、 Windows NT（虚拟内存、网络及文件系统）等的实现。
        - CS 61B：伸展树（Splay trees） （视频）
        - MIT 教程：伸展树（Splay trees）：
            - 该教程会过于学术，但请观看到最后的10分钟以确保掌握。
            - 视频

    - **红黑树**

        - 这些是2-3棵树的翻译（请参见下文）。
        - 实际中：红黑树提供了在最坏情况下插入操作、删除操作和查找操作的时间保证。 这些时间值的保障不仅对时间敏感型应用有用，例如实时应用， 还对在其他数据结构中块的构建非常有用， 而这些数据结构都提供了最坏情况下的保障； 例如，许多用于计算几何学的数据结构都可以基于红黑树， 而目前 Linux 内核所采用的完全公平调度器（the Completely

Fair Scheduler）也使用到了该种树。 在 Java 8中，Collection HashMap也从原本用Linked List实现， 储存特定元素的哈希码，改为用红黑树实现。

- Aduni —— 算法 —— 课程4（该链接直接跳到开始部分）（视频）
- Aduni —— 算法 —— 课程5（视频）
- 黑树（Black Tree）
- 二分查找及红黑树的介绍
- [Review] Red-Black Trees (playlist) in 30 minutes (video)

- **2-3查找树**

  - 实际中： 2-3树的元素插入非常快速，但却有着查询慢的代价（因为相比较 AVL 树来说，其高度更高）。
  - 你会很少用到2-3树。这是因为，其实现过程中涉及到不同类型的节点。因此，人们更多地会选择红黑树。
  - 2-3树的直感与定义（视频）
  - 2-3树的二元观点
  - 2-3树（学生叙述）（视频）

- **2-3-4树 (亦称2-4树)**

  - 实际中： 对于每一棵2-4树，都有着对应的红黑树来存储同样顺序的数据元素。 在2-4树上进行插入及删除操作等同于在红黑树上进行颜色翻转及轮换。 这使得2-4树成为一种用于掌握红黑树背后逻辑的重要工具。 这就是为什么许多算法引导文章都会在介绍红黑树之前，先介绍2-4树，尽管**2-4树在实际中并不经常使用。**
  - CS 61B Lecture 26：平衡查找树（视频）
  - 自底向上的2-4树（视频）
  - 自顶向下的2-4树（视频）

- **N 叉树（K 叉树、M 叉树）**

  - 注意：N 或 K 指的是分支系数（即树的最大分支数）：
  - 二叉树是一种分支系数为2的树
  - 2-3树是一种分支系数为3的树
  - K 叉树

- **B 树**

  - 有趣的是：为啥叫 B 仍然是一个神秘。因为 B 可代表波音（Boeing）、平衡（Balanced）或 Bayer（联合创造者）
  - 实际中： B 树会被广泛适用于数据库中，而现代大多数的文件系统都会使用到这种树（或变种）。 除了运用在数据库中，B 树也会被用于文件系统以快速访问一个文件的任意块。但存在着一个基本的问题， 那就是如何将文件块 i 转换成一个硬盘块（或一个柱面-磁头-扇区）上的地址。
  - B 树
  - B 树数据结构
  - B 树的介绍（视频）
  - B 树的定义及其插入操作（视频）
  - B 树的删除操作（视频）
  - MIT 6.851 —— 内存层次模块（Memory Hierarchy Models）（视频）

        - 覆盖有高速缓存参数无关型（cache-oblivious）B 树和非常有趣的数据结构
        - 头37分钟讲述的很专业，或许可以跳过（B 指块的大小、即缓存行的大小）
    - [Review] B-Trees (playlist) in 26 minutes (video)

- k-D树
    - 非常适合在矩形或更高维度的对象中查找点数
    - 最适合k近邻
    - kNN K-d树算法（视频）
- 跳表
    - "有一种非常迷幻的数据类型" - Skiena
    - 随机化: 跳表 (视频)
    - 更生动详细的解释
- 网络流
    - 5分钟简析 Ford-Fulkerson——一步步示例 (视频)
    - Ford-Fulkerson 算法 (视频)
    - 网络流 (视频)
- 不相交集 & 联合查找
    - UCB 61B - 不相交集；排序 & 选择(视频)
    - Sedgewick算法——Union-Find（6视频）
- 快速处理的数学
    - 整数运算, Karatsuba 乘法 (视频)
    - 中国剩余定理 (在密码学中的使用) (视频)
- 树堆 (Treap)
    - 一个二叉搜索树和一个堆的组合
    - 树堆
    - 数据结构：树堆的讲解（视频）
    - 集合操作的应用(Applications in set operations)
- 线性规划（Linear Programming）（视频）
    - 线性规划
    - 寻找最小成本
    - 寻找最大值
    - 用 Python 解决线性方程式——单纯形算法
- 几何：凸包（Geometry, Convex hull）（视频）
    - Graph Alg. IV: 几何算法介绍 - 第 9 课
    - Graham & Jarvis: 几何算法 - 第 10 课

    - 分而治之: 凸包, 中值查找

- 离散数学

    - 计算机科学70，001 - 2015年春季 - 离散数学与概率论
    - 离散数学由Shai Simonson（19个视频）
    - 离散数学由印度理工学院罗帕尔分校NPTEL提供

一些主题的额外内容

> 我添加了这些内容来加强上面已经提出的一些观点，但是不想把它们放在上面，因为那样会太多。
> 对于一个主题来说，过度处理很容易。
> 你希望在本世纪被雇佣吗?

- **SOLID**

    - [ ] Bob Martin SOLID Principles of Object Oriented and Agile Design (视频)
    - [ ] S - 单一职责原则 | 每个对象负责一个单一职责 | Single responsibility to each Object
        - 更多解释
    - [ ] O - 开闭原则 | 在生产级别上，对象应准备好进行扩展，但不进行修改
        - 更多解释
    - [ ] L - 里氏替换原则 | 基类和派生类遵循'是一个'原则
        - 更多解释
    - [ ] I - 接口隔离原则 | 客户端不应被强制实现不使用的接口
        - 5分钟内的接口隔离原则（视频）
        - 更多解释
    - [ ] D -依赖反转原则 | 在对象的组合中减少依赖
        - 为何依赖反转原则如此重要
        - 更多解释

- **Union-Find**

    - 概览
    - 初级实践
    - 树状结构
    - 合并树状结构
    - 路径压缩
    - 分析选项

- **动态规划的更多内容** (视频)

    - 6.006: 动态规划 I: 斐波那契数列, 最短路径
    - 6.006: 动态规划 II: 文本匹配, 二十一点/黑杰克
    - 6.006: 动态规划 III: 最优加括号方式, 最小编辑距离, 背包问题
    - 6.006: 动态规划 IV: 吉他指法，拓扑，超级马里奥.
    - 6.046: 动态规划: 动态规划进阶
    - 6.046: 动态规划: 所有点对最短路径

    - 6.046: 动态规划: 更多示例

- **图形处理进阶** (视频)
    - 异步分布式算法: 对称性破缺，最小生成树
    - 异步分布式算法: 最小生成树

- MIT **概率论** (过于数学，进度缓慢，但这对于数学的东西却是必要之恶) (视频):
    - MIT 6.042J - 概率论概述
    - MIT 6.042J - 条件概率 Probability
    - MIT 6.042J - 独立
    - MIT 6.042J - 随机变量
    - MIT 6.042J - 期望 I
    - MIT 6.042J - 期望 II
    - MIT 6.042J - 大偏差
    - MIT 6.042J - 随机游走

- Simonson: 近似算法 (视频)

- **字符串匹配**
    - Rabin-Karp（视频）
        - Rabin Karps 算法
        - 预计算
        - 优化：实施和分析
        - 表翻倍，Karp-Rabin
        - 滚动哈希，摊销分析
    - Knuth-Morris-Pratt (KMP)：
        - Knuth-Morris-Pratt（KMP）字符串匹配算法
    - Boyer–Moore 字符串搜索算法
        - Boyer–Moore 字符串搜索算法
        - 高级字符串搜索Boyer-Moore-Horspool算法（视频）
    - Coursera：字符串算法
        - 刚开始时很棒，但是当它超过 KMP 时，它变得比需要复杂得多
        - 很好的字典树解释
        - 可以跳过

- **排序**
    - 斯坦福大学关于排序算法的视频:
        - 课程 15 | 编程抽象（视频）
        - 课程 16 | 编程抽象（视频）
    - Shai Simonson 视频，Aduni.org:
        - 算法 - 排序 - 第二讲（视频）
        - 算法 - 排序2 - 第三讲（视频）
    - Steven Skiena 关于排序的视频:
        - CSE373 2020 - 归并排序/快速排序（视频）
        - CSE373 2020 - 线性排序（视频）

- NAND 到 Tetris: 从第一原理构建现代计算机

视频系列

坐下来，尽情享受。

- 个人的动态规划问题列表 (都是短视频)
- x86 架构，汇编，应用程序 (11 个视频)
- MIT 18.06 线性代数，2005 年春季 (35 个视频)
- 绝妙的 MIT 微积分：单变量微积分
- Skiena讲座来自《算法设计手册》- CSE373 2020 - 算法分析（26个视频）
- UC Berkeley 61B (2014 年春季): 数据结构 (25 个视频)
- UC Berkeley 61B (2006 年秋季): 数据结构 (39 个视频)
- UC Berkeley 61C：计算机结构 (26 个视频)
- OOSE：使用 UML 和 Java 进行软件开发 (21 个视频)
- MIT 6.004: 计算结构 (49 视频)
- 卡内基梅隆大学 - 计算机架构讲座 (39 个视频)
- MIT 6.006：算法介绍 (47 个视频)
- MIT 6.033：计算机系统工程 (22 个视频)
- MIT 6.034：人工智能，2010 年秋季 (30 个视频)
- MIT 6.042J：计算机科学数学, 2010 年秋季 (25 个视频)
- MIT 6.046：算法设计与分析 (34 个视频)
- MIT 6.050J：信息和熵, 2008 年春季 (19 个视频)
- MIT 6.851：高等数据结构 (22 个视频)
- MIT 6.854: 高等算法, 2016 年春季 (24 个视频)
- Harvard COMPSCI 224：高级算法（25个视频）
- MIT 6.858：计算机系统安全, 2014 年秋季
- 斯坦福: 编程范例 (27 个视频)
- 密码学导论，Christof Paar
    - 课程网站以及幻灯片和问题集
- 大数据 - 斯坦福大学 (94 个视频)

- 图论，Sarada Herke（67个视频）

计算机科学课程

- 在线 CS 课程目录
- CS 课程目录 (一些是在线讲座)

算法实现

- 普林斯顿大学的多算法实现

论文

- 喜欢经典的论文?
- 1978: 通信顺序处理
  - Go 实现
- 2003: The Google 文件系统
  - 2012 年被 Colossus 取代了
- 2004: MapReduce: Simplified Data Processing on Large Clusters
  - 大多被云数据流取代了?
- 2006年：Bigtable：结构化数据的分布式存储系统
- 2006年：针对松散耦合的分布式系统的Chubby Lock服务
- 2007年：Dynamo：亚马逊的高可用键值存储
  - Dynamo论文启动了NoSQL革命
- 2007: 每个程序员都应该知道的内存知识 (非常长，作者建议跳过某些章节来阅读)
- 2012: AddressSanitizer: 快速的内存访问检查器:
  - 论文
  - 视频
- 2013: Spanner: Google 的分布式数据库:
  - 论文
  - 视频
- 2015: Google的持续流水线
- 2015: 大规模高可用性：构建Google广告数据基础设施
- 2015: 开发人员如何搜索代码：一个案例研究
- 更多论文： 1,000篇论文

LICENSE