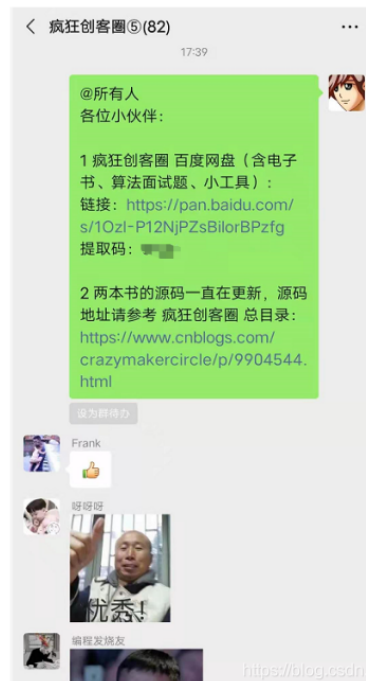


# 史上最全 Java 面试题：算法篇（入群获取资料）



## 刷算法的秘籍

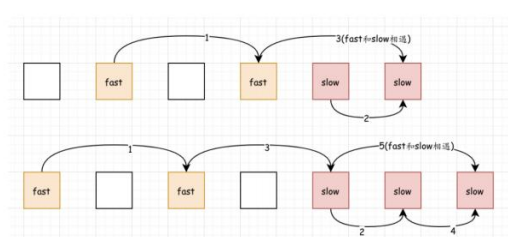
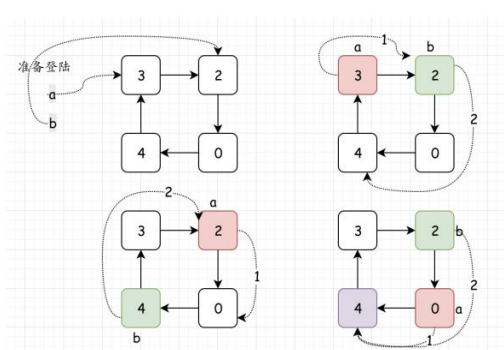
如何刷算法，这块有秘籍，具体找40岁老架构尼恩，微信交流就可以了

## 算法刷题宝典

刷题任务的题目，是根据题目的类型来汇总的，总结了八个类别，每个类别下面也总结了5个左右的题型，帮助大家分门别类的突破，所以刷起来相对会更有重点和针对性。如果从头到尾的刷，每周按顺序刷42题，很容易让自己坚持不下来，也会觉得很枯燥。所以在制定计划的时候可以让这个计划变得更“有趣”和针对性，让它看起来更容易实现一点，才会更容易坚持。

<h3>数组系列</h3> <ul style="list-style-type: none"> <li>两个数组的交集(350)</li> <li>最长公共前缀(14)</li> <li>买卖股票的最佳时机(122)</li> <li>旋转数组(189)</li> <li>原地删除(27)</li> <li>加一(66)</li> <li>两数之和(1)</li> </ul> <h3>链表系列</h3> <ul style="list-style-type: none"> <li>删除链表倒数第N个节点(19)</li> <li>合并两个有序链表(21)</li> <li>环形链表(21)</li> </ul> <h3>动态规划系列</h3> <ul style="list-style-type: none"> <li>爬楼梯(70)</li> <li>最大子序和(53)</li> <li>最长上升子序列(300)</li> <li>三角形最小路径和(120)</li> <li>最小路径和(64)</li> <li>打家劫舍(198)</li> </ul> <h3>字符串系列</h3> <ul style="list-style-type: none"> <li>反转字符串(301)</li> <li>字符串中的第一个唯一字符(387)</li> </ul> <h3>二叉树系列</h3> <ul style="list-style-type: none"> <li>最大深度与DFS(104)</li> <li>层次遍历与BFS(102)</li> <li>BST与其验证(98)</li> <li>BST的查找(700)</li> <li>BST的删除(450)</li> </ul>	<h3>滑动窗口系列</h3> <ul style="list-style-type: none"> <li>滑动窗口最大值(239)</li> <li>无重复字符的最长子串(3)</li> <li>找到字符串中所有字母异位词(438)</li> </ul> <h3>博弈论系列</h3> <ul style="list-style-type: none"> <li>囚徒困境</li> <li>纳什均衡</li> <li>红眼睛和蓝眼睛</li> <li>海盗分金</li> <li>排序类题目</li> <li>按奇偶排序数组(905)</li> </ul> <h3>位运算系列</h3> <ul style="list-style-type: none"> <li>使用位运算求和</li> <li>2的幂(231)</li> <li>返回一个数二进制中1的个数(191)</li> <li>只出现一次的数字(136)</li> <li>只出现一次的数字II(137)</li> <li>缺失数字(268)</li> </ul> <h3>二分法系列</h3> <ul style="list-style-type: none"> <li>爱吃香蕉的珂珂(875)</li> <li>x的平方根(69)</li> <li>第一个错误的版本(287)</li> </ul>	<h3>其他题目</h3> <ul style="list-style-type: none"> <li>螺旋矩阵(54)</li> <li>只有两个键的键盘(650)</li> <li>24点游戏(679)</li> <li>飞机座位分配概率(1227)</li> <li>水分子的产生</li> <li>救生艇(881)</li> <li>救生艇(881)</li> <li>灯泡开关(319)</li> <li>三门问题</li> <li>猜数字游戏(299)</li> <li>LRU缓存机制(146)</li> <li>最小的k个数</li> <li>不同路径</li> <li>不同路径 - 障碍物</li> <li>伪装师卡洛</li> <li>盛最多水的容器</li> <li>扑克牌中的顺子容器</li> <li>整数拆分(343)</li> <li>移动石子直到连续(1033)</li> <li>Nim 游戏(292)</li> </ul>
--	---	---

## 部分内容展示:



所以我们的快指针的步长可以设置为 2。

在上一期中，我们通过题目“最长上升子序列”以及“最大子序和”，学习了DP（动态规划）在递推关系中的分析方法。这种分析方法，也在递推中被称为“线性动态规划”。具体指的是“目标函数为特定变量的线性函数，约束是这些变量的线性不等式或等式，目标是求目标函数的最大值或最小值”。这点大家作为了解即可，不需要死记，更不要死搬硬套！

在本期中，我们将继续分析一道与递推关系于之前的题目，希望可以由此题与之前的题目进行对比论证，进而融会贯通！

## 01、题目分析

### 第120题：三角形最小路径和

给定一个三角形，找出自顶向下的最小路径和。每一步只能移动到下一行中相邻的结点上。

例如，给定三角形：

```
1  [
2  [2],
3  [3,4],
4  [6,5,7],
5  [4,1,8,3]
6  ]
```

则自顶向下的最小路径和为11（即，2+3+5+1=11）。

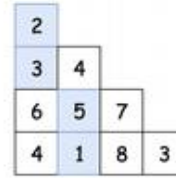
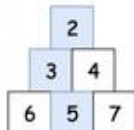
这道题有一定难度！如果没有思路请回顾上一期的学习内容！

不建议直接看题解！

## 02、题目图解

首先我们分析题目，要找的是三角形最小路径和。这是个递推题呢？假设我们有一个三角形：[[2],[3,4],[6,5,7],[4,1,8,3]]。

[[4,1,8,3]]



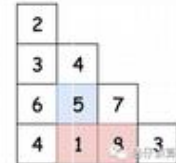
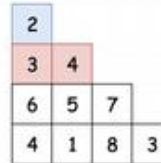
图例

这样相当于我们将整个三角形进行了拉伸，这时候，我们根据题目中给出的条件：每一步只能移动到下一行中相邻

的结点上，其实也是等同于，每一步我们只能往下移动一格或者右下移动一格。将其转化成代码，假如2所在的位置

位置为(0,0)，那我们往下移动就只能移动到(1,0)或者(1,1)的位置上，假如(0,0)所在的位置为(2,1)，同样也只能移动

到(3,1)或(3,2)的位置上，如下图所示：



图例

题目明确了之后，现在我们可以开始进行分析，题目很明显是一个找最优解的问题，并且可以从子问题的最优解

构建，所以我们通过动态规划进行求解，首先，我们定义状态：

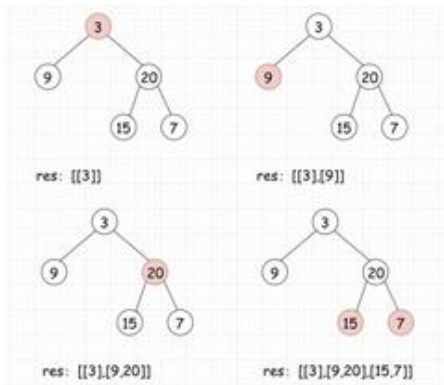
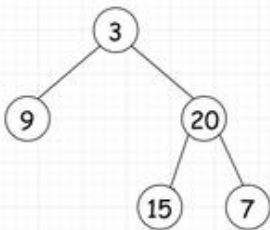
dp[i][j]：表示包含第i行到第j列元素的最小路径和

我们很容易想到可以从自顶向下进行分析，并且，无论最后的路径是哪一条，它一定需要经过最顶层的元素，即(0,0)，所以我们需要对dp[0][0]进行初始化。

## 03、递归求解

同样，我们先考虑题目的递归解法。想到递归，我们一般先想到DFS，我们可以对二叉树进行先序遍历（根左右的顺序），同时，记录节点所在的层数level，并且对每一层都定义一个数组，然后将访问到的节点值放入对应的数组中。

假设给定二叉树为[[3,9,20,null,null,15,7]]，图解如下：



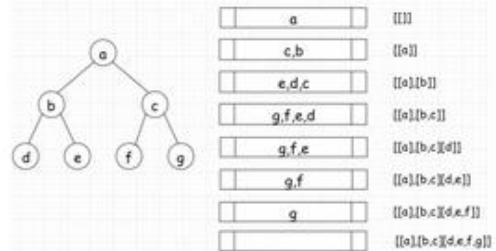
根据以上分析，代码如下：

```
1  func dfs(root *TreeNode, level int, res [][]int) [][]int {
2      if root == nil {
3          return res
4      }
5      if len(res) == level {
6          res = append(res, []int{root.Val})
7      } else {
8          res[level] = append(res[level], root.Val)
9      }
10     res = dfs(root.Left, level+1, res)
11     res = dfs(root.Right, level+1, res)
12     return res
13 }
```

## 04、BFS求解

上面的解法，其实相当于用DFS的方法实现了二叉树的BFS，那我们能不能直接使用BFS的方式进行求解呢？当然，我们可以使用Queue的数据结构，我们将root节点初始化进队列，通过消耗队列，插入头部的元素来完成BFS。

具体步骤如下：



根据以上分析，代码如下：

```
1  func levelOrder(root *TreeNode) [][]int {
2      var result [][]int
3      if root == nil {
4          return result
5      }
6      // 定义一个队列
7      queue := []int{root.Val}
```

数组	双端队列	result数组
1 3 -1 -3 5 3 6 7	1	
1 3 -1 -3 5 3 6 7	3	
1 3 -1 -3 5 3 6 7	3,1	3
1 3 -1 -3 5 3 6 7	3,-1,3	3 3
1 3 -1 -3 5 3 6 7	5	3 3 5
1 3 -1 -3 5 3 6 7	5,3	3 3 5 5
1 3 -1 -3 5 3 6 7	6	3 3 5 5 6
1 3 -1 -3 5 3 6 7	7	3 3 5 5 6 7

(小话os: 我觉得自己画的这个图对于双端队列的解法还是介绍的比较清晰的。大家好好看一下哦。这样我的努力也就没有白费啦。)

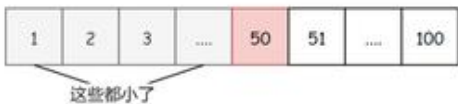
根据分析，得出代码：

```
1 func maxSlidingWindow(nums []int, k int) []int {
2     if len(nums) == 0 {
3         return []int{}
4     }
5     //初始化队列-一个双端队列
6     queue := []int{}
7     result := []int{}
8     for i := range nums {
9         for i > 0 && (len(queue) > 0 && nums[i] > queue[len(queue)-1]) {
10             //用比当前元素小的元素清空
11             queue = queue[:len(queue)-1]
12         }
13         //将当前元素放入queue中
14         queue = append(queue, nums[i])
15         if i >= k && nums[i-k] == queue[0] {
16             //维护队列，保证其头元素为当前窗口最大值
17             queue = queue[1:]
18         }
19         if i >= k-1 {
20             //加入结果数组
21             result = append(result, queue[0])
22         }
23     }
24     return result
25 }
```

在最简单的形式中，二分查找对具有固定索引或右索引的连续序列进行操作。我们称之为**查找空间**。二分查找维护查找空间的左、右和中间指示符，并比较查找目标；如果条件不满足或不相等，则清除目标不可能存在的那一半，并在剩下的一半上继续查找，直到成功为止。



举例说明：比如你需要找1-100中的一个数字，你的目标是**用最少**的次数猜到这个数字。你每次猜测后，我会说大了或者小了，而你只需要每次猜到中间的数字，就可以将余下的数字排除一半。



不管我心里想的数字如何，你在7次之内都能猜到。这就是一个典型的二分查找。每次将选择一半数据，所以我们也称之为**折半查找**。一般而言，对于包含n个元素的列表，用二分查找最多需要log2n步。



当然，一般题目不太可能给你一个如此规整的题型，让你上手就可以使用二分，所以我们需要思考，如何来构造一个成功的二分查找。大部分的二分查找，基本都由以下三步组成：

- 预处理过程（大部分场景就是对未排序的集合进行排序）
- 二分查找过程（找到合适的循环条件，每一次将查找空间一分为二）
- 后处理过程（在剩余的空间中，找到合适的目标值）

了解了二分查找的过程，我们对二分查找进行**一般实现**（这里给出一个Java版本，比较正规的代码，没

执行用时：20 ms , 在所有 Go 提交中击败了 98.41 的用户  
内存消耗：6.3 MB , 在所有 Go 提交中击败了 88.18% 的用户

Perfect提交地址：戳着一下子超越百分之99的用户，是不是感觉爽爽爽

## 无重复字符的最长子串（3）

在上一节中，我们使用双端队列完成了滑动窗口的一道极为困难的题目，以此展示了什么是滑动窗口。在本节中我们将继续深入分析，探索滑动窗口题型一些具有模式性的解法。

### 01. 滑动窗口介绍

对于大部分滑动窗口类型的题目，一般是考察字符串的匹配，比较标准的题目，会给出一个模式串B，以及一个目标串A，然后提出问题，找到A中符合对B一些限定规则的子串或者对A一些限定规则的结果，最终再根据求出的子串完成题目中要求的组合或者其他。

比如：给定一个字符串s和一个非空字符串p，找到s中所有是p的字母异位词的子串，返回这些子串的起始索引。

又或者：给你一个字符串S、一个字符串T，请在字符串S里面找出：包含T所有字母的最小子串。

再如：给定一个字符串s和一些长度相同的单词 words，找出 s 中恰好可以由 words 中所有单词串联形成的子串的起始位置。

都是属于这一类的标准题型。而对于这一类题目，我们常用的解题思路，是去维护一个可变长度的滑动窗口。无论是使用双指针，还是使用双端队列，又或者用游标等其他奇技淫巧，目的都是一样的。

Now，我们通过一道题目来进行具体学习吧

### 02. 题目分析

```
5     int mid = low + (high - low) / 2;
6     if (des == array[mid]) {
7         return mid;
8     } else if (des < array[mid]) {
9         high = mid - 1;
10    } else {
11        low = mid + 1;
12    }
13 }
14 return -1;
15 }
```

注意：上面的代码，mid 使用 low + (high - low)/2 的目的，是防止 high-low 溢出内存。

为什么说是一般实现？

1. 根据边界的不同（开区间闭区间），有时需要弹性调整low与high的值，以及循环的终止条件。
2. 根据元素是否有重复值，以及是否需要找到重复值区间，有时需要对原算法进行改进。

那上面我们说了，一般二分查找的过程分为：预处理-二分查找-后处理。上面的代码，就没有后处理的过程，因为在每一步中，你都检查了元素，如果到达末尾，也已知知道没有找到元素。

总结一下一般实现的几个条件：

- 初始条件：left = 0, right = length-1
- 终止：left > right
- 向左查找：right = mid-1
- 向右查找：left = mid+1

请大家记住这个模板图形，在后面的系列中，我们将介绍二分查找其他的模板类型。

### 03. 题目分析

简单复习了二分查找，我们来看本题。

注意，绝大部分『在递增递减区间中搜索目标值』的问题，都可以转化为二分查找问题。并且，二分查找的题目，基本逃不出三种：找特定值，找大于特定值的元素（上界），找小于特定值的元素（下

继续向上调整

继续向上调整

继续向上调整

建堆 调整的代码大概就是这样: (翻java牌子)

```

1 //建堆, 对于每一个还没有经过调整的, 从根的最右一个节点的父节点开始进行调整。
2 private void buildHeap(int[] nums) {
3     //最后一个节点
4     int lastNode = nums.length - 1;
5     //记住, 父节点 = (i + 1) / 2 左节点 = 2 * i + 1 右节点 = 2 * i + 2;
6     //最后一个节点的父节点
7     int startHeapify = (lastNode + 1) / 2;
8     while (startHeapify >= 0) {
9         //不断调整堆的过程
10        heapify(nums, startHeapify--);
11    }
12 }
13 //调整大根堆的过程
14 private void heapify(int[] nums, int i) {
15     //和当前节点的左右节点比较, 如果节点中有更大的数, 那么交换, 并继续对交换后的节点进行调整
16

```

```

21 //假定当前节点最大
22 int max = i;
23 //如果左子节点比父大, 更新max = c1;
24 if (c1 > 1000000000) max = c1;
25 //如果右子节点比父大, 更新max = c2;
26 if (c2 > 1000000000) max = c2;
27 //如果最大的数不是当前节点, 那么heapify(nums, max), 即调整节点i的子树。
28 if (max != i) {
29     swap(nums, max, i);
30     //递归调整
31     heapify(nums, max);
32 }
33 }
34 private void swap(int[] nums, int i, int j) {
35     nums[i] = nums[i] + nums[j];
36     nums[j] = nums[i] - nums[j];
37     nums[i] = nums[i] - nums[j];
38 }

```

然后我们从下往上继续开始依次调整剩余的节点。如果元素小于堆顶元素, 那么取出堆顶元素, 将堆顶元素入堆。在上例的分例中, 因为2小于堆顶元素6, 所以将2入堆, 我们发现现在的完全二叉树不满足大根堆, 所以对其进行调整。

调整前

调整后

继续重复上述步骤, 依次将7,3,8入堆。这里因为7和8都大于堆顶元素5, 所以只有3会入堆。

调整前

## LeetCode (520 道题)

除此之外, 这里再跟大家推荐一本前不久火爆 GitHub 的 LeetCode 中文刷题手册, 这本小册里面共包含刷 LeetCode 后整理的 520 道题, 每道题均附有详细题解过程。自发布以后, 受到技术圈内广大开发者的赞赏, 建议大家收藏阅读。目录如下:

部分目录展示:

### 11. Container With Most Water

**题目**

Given  $n$  non-negative integers  $a_1, a_2, \dots, a_n$ , where each represents a point at coordinate  $(i, a_i)$ .  $n$  vertical lines are drawn such that the two endpoints of line  $i$  is at  $(i, a_i)$  and  $(i, 0)$ . Find two lines, which together with x-axis forms a container, such that the container contains the most water.

**Note:** You may not slant the container and  $n$  is at least 2.

The above vertical lines are represented by array  $[1, 8, 6, 2, 5, 4, 8, 3, 7]$ . In this case, the max area of water (blue section) the container can contain is 49.

**Example 1:**

Input:  $[1, 8, 6, 2, 5, 4, 8, 3, 7]$   
Output: 49

**题目大意**

给出一个非负整数数组  $a_1, a_2, a_3, \dots, a_n$ , 每个整数标识一个建立在坐标轴  $x$  位置的一堵高度为  $a_i$  的墙。选择两堵墙, 和  $x$  轴构成的容器可以容纳最多的水。

**解题思路**

这一题也是对撞指针的思路。首先分别 2 个指针, 每次移动以后都分别判断长度的乘积是否最大。

**代码**

```

package Testcode

func maxArea(height []int) int {
    max, start, end := 0, 0, len(height)-1
    for start < end {
        width := end - start
        high := 0
        if height[start] < height[end] {
            high = height[start]
            start++
        } else {
            high = height[end]
            end--
        }
        temp := width * high
        if temp > max {
            max = temp
        }
    }
    return max
}

```

### 13. Roman to Integer

**题目**

Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M.

Symbol	Value
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

For example, two is written as II in Roman numeral, just two one's added together. Twelve is written as, XII, which is simply X + II. The number twenty seven is written as XXVII, which is XX + V + II.

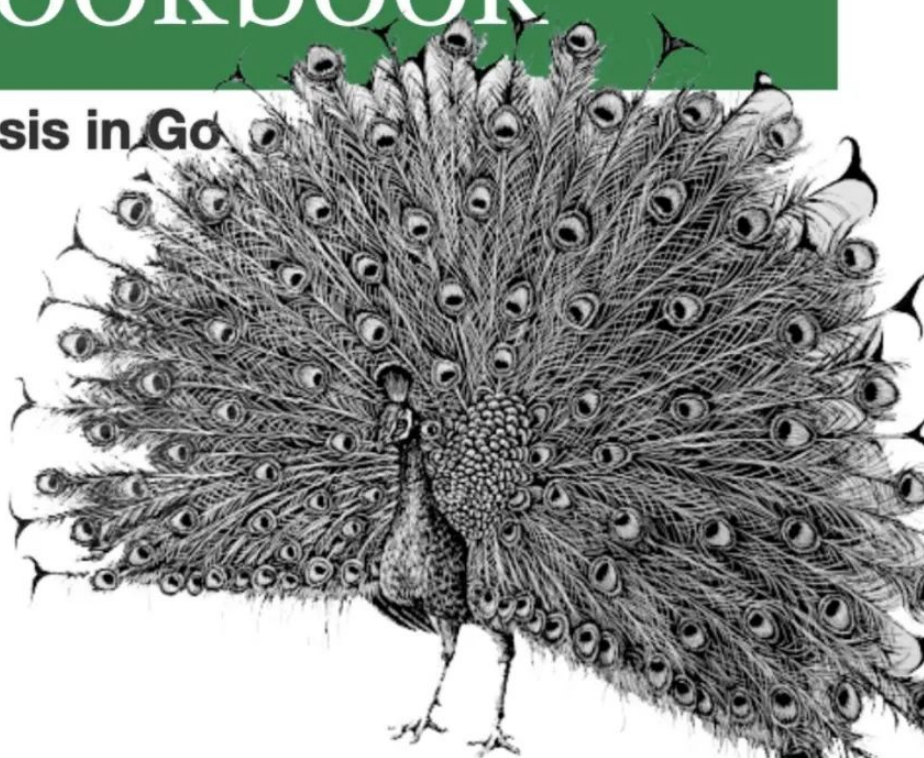


*A Programmer's Advanced Cookbook*

How to improve algorithm skills

# LeetCode Cookbook

Analysis in Go



HALFROST©

halfrost 著

フロストランド 訳

知乎 @程序员吴师兄

**力扣** Cookbook是@halfrost（中文名：**霜神**）去年刷的**力扣**整理出的 520 题，每道题都写了解题思路，并且每题都 runtime beats 100% 了。

## 至于为什么要求每题都 runtime beats 100%?

霜神是这样回复的：优化到 beats 100% 才算是把这题做出感觉了。有好几道 Hard 题，可以用暴力解法 AC 了，但只 beats 了 5%，这题就如同没做一样；

而且面试中如果给了暴力的答案，面试官也不会满意，通过自己的思考给出更优解，面试官也会更满意一些。

所以如果你把这些题解都摸透，相信在面试环节你可以从容的回答“还有没有更优解”这个问题。

现在就把这本电子书分享给大家，希望能帮助大家克服刷题的恐惧，顺利拿到大厂 offer。

# Linux 归纳笔记

此外这里还有一份华为大牛总结的 Linux 归纳笔记，一并分享给大家。

这份资料非常全面且详细，从 **Linux 常用命令**到 **Linux 常用操作**，再到**网络管理**、**性能优化**，几乎覆盖了 Linux 基础学习的方方面面，非常适合初学者入门！

资料也按目录进行编排，每一章下面都有更具体的内容：

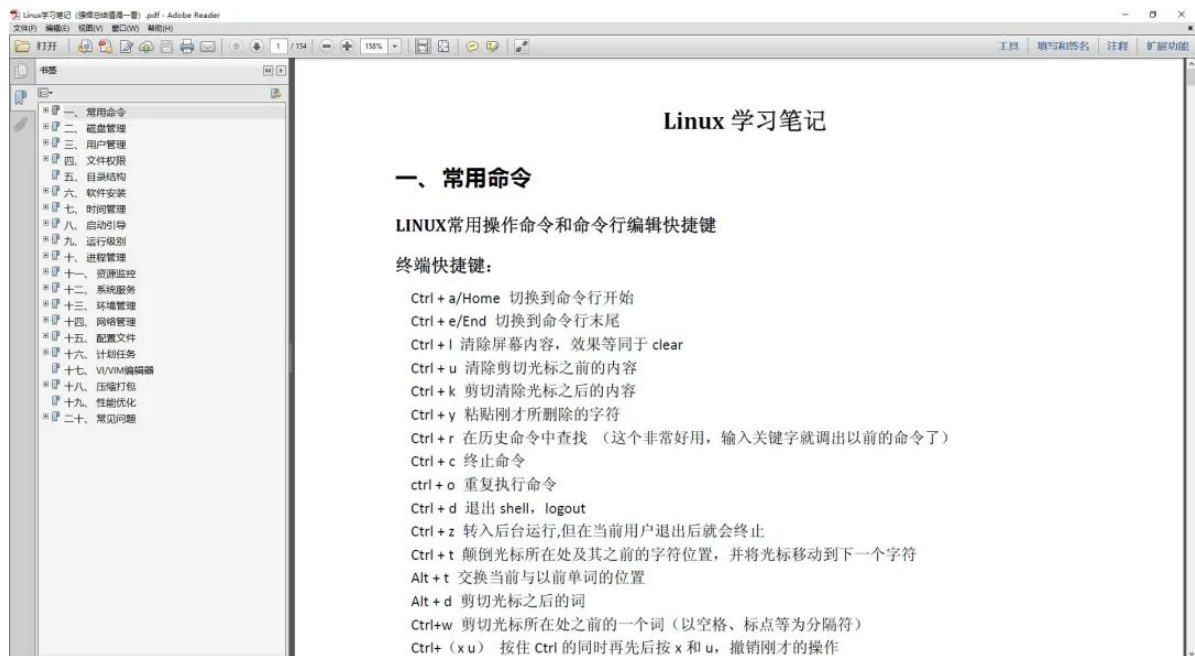
## 非常详细的目录



## 每章都有更细分的内容



而且，这份资料不是扫描版的，里面的文字都可以直接复制，非常便于我们学习：



## 获取方式

通过疯狂创客圈网盘，统一获取，进群后请参见群公告，加尼恩微信即可，具体的微信二维码，请参见[博客园总入口](#)

另外附带n本电子书。