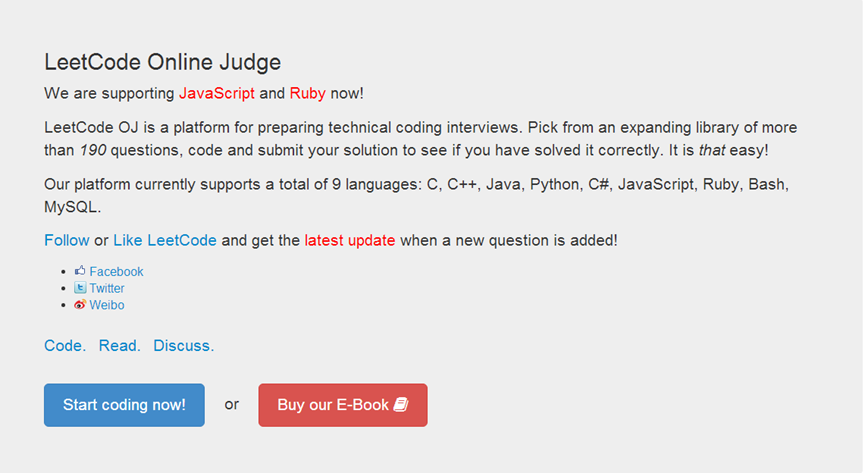
·

菜鸟写给菜鸟的

——LeetCode解题笔记



摘要

Leetcode.com是一个面向面试的oj平台，也是每个求职者及算法初学者必须刷的oj，同时也十分适合编程初学者熟悉各种语言语法、数据结构及编程思维。本解题笔记记录了每道题的简要分析，解题过程及ac源码，不求最佳算法和最优美的代码，但求与读者共同进步。

北京邮电大学 网络技术研究院

网络服务基础研究中心

（一堆高大上的装B头衔之后就是作者）

彩笔初三小学妹

Email: sry2011211535@bupt.edu.cn

## **时间线记录**

2015.12.21 （打出这个日期我才发现，原来是玛雅预言的2012世界末日三周年= =） 开始Leetcode的征程

2015.12.31 在熄灯前最后一分钟，Easy最后一题

2016.1.27 寒假前最后一天。Leetcode百题达成。

2016.2.22 元宵节，一个很2很2的日子。Leetcode200题达成，Middle部分完成。

2016.3.2 解题笔记Easy部分完成。

2016.3.18 解题笔记Middle部分完成。并增补3题。

## **前 言**

本文档仅是本人为纪念解题完毕，对解题思路和解题过程进行统一整理。以便以后遇到类似题目时可以参考，或是多年以后对曾经的学习生活的一段回忆。

1. 笔者也是个算法上的新手（毕竟人家才【初三】嘛，(\*^\_\_^\*) 嘻嘻……），如果代码过于繁琐或算法比较trivial甚至bad，欢迎读者提出宝贵意见，我会向您虚心学习。但千万别喷我。

(p.s. 也许这个报告的读者只有我自己……也没准多年以后再读到此文档的时候，会嘲笑几年前的愚蠢吧……)

1. 因初次刷题，下文中的代码以ac为主要目的，并未考虑follow up中的部分，因此有的可能算法很复杂，或者未实现题目要求（如时间空间复杂度的要求，要求不开辟新空间等等）。更有的题直接水过（例如实现sqrt，实现正则表达式等。直接用Java库函数解决），丑话在先，如果读到这样的代码请不要喷我。
2. 关于选择语言的问题：优先选择Java，但也有其他语言。例如涉及指针较多的（如树、图、链表的题）我也许会考虑C（因为Java没指针写起来不得劲啊！而且C里面非0为true这一性质有时候很节省代码量~~）；如果涉及数组切片我可能考虑Python（因为python对数组的处理实在简洁而强大）其他的如js和ruby因为我也不会，就不写了。
3. 题目的顺序为按难度升序排列，即按Easy,Medium,Hard顺序，同难度按AC率降序。
4. 前140题的大多数代码都是自己写的，限于水平有限，有一部分实在不会的或者想不到的来源于百度（基本在140题后），若侵权请说明。
5. Hard难度暂不收录，以待后期边刷题边整理。第一版中仅有Easy Medium部分。且Easy部分中可能会出现很trivial甚至很stupid 的code，会在日后进一步整理。
6. Beat数、众数统计均截止到截稿。
7. 上述AC代码不确保以后也能AC，反之不能AC的代码也不代表以后不能AC。因为Leetcode中有些测试用例是有问题的。
8. 可能由于Leetcode的评判服务器建立jvm的时间不同、服务器负载量不同等原因，同样的代码的运行时间不一定相同。本解题报告中若出现同代码的不同时间，一律取最小值。
9. 虽然对每一道题都给出了大意，但不是对原文原封不动的直译，且阅读本解题报告仍需要一点英文水平，连这点英文水平都没有也别找工作了……

[Easy-题目1：292. Nim Game 1](#_Toc446778124)

[Easy-题目2：258. Add Digits 1](#_Toc446778125)

[Easy-题目3：104. Maximum Depth of Binary Tree 2](#_Toc446778126)

[Easy-题目4：226. Invert Binary Tree 3](#_Toc446778127)

[Easy-题目5：237. Delete Node in a Linked List 4](#_Toc446778128)

[Easy-题目6：283. Move Zeroes 4](#_Toc446778129)

[Easy-题目7：100. Same Tree 5](#_Toc446778130)

[Easy-题目8：242. Valid Anagram 6](#_Toc446778131)

[Easy-题目9：171. Excel Sheet Column Number 7](#_Toc446778132)

[Easy-题目10：217. Contains Duplicate 7](#_Toc446778133)

[Easy-题目11：169. Majority Element 8](#_Toc446778134)

[Easy-题目12：206. Reverse Linked List 9](#_Toc446778135)

[Easy-题目13：13. Roman to Integer 10](#_Toc446778136)

[Easy-题目14：235.Lowest Common Ancestor of a Binary Search Tree 11](#_Toc446778137)

[Easy-题目15：191. Number of 1 Bits 12](#_Toc446778138)

[Easy-题目16：328. Odd Even Linked List 13](#_Toc446778139)

[Easy-题目17：83. Remove Duplicates from Sorted List 14](#_Toc446778140)

[Easy-题目18：70. Climbing Stairs 14](#_Toc446778141)

[Easy-题目19：263. Ugly Number 15](#_Toc446778142)

[Easy-题目20：202. Happy Number 16](#_Toc446778143)

[Easy-题目21：326. Power of Three 17](#_Toc446778144)

[Easy-题目22：231. Power of Two 18](#_Toc446778145)

[Easy-题目23：21. Merge Two Sorted Lists 19](#_Toc446778146)

[Easy-题目24：110. Balanced Binary Tree 20](#_Toc446778147)

[Easy-题目25：101. Symmetric Tree 21](#_Toc446778148)

[Easy-题目26：198. House Robber 22](#_Toc446778149)

[Easy-题目27：107.Binary Tree Level Order Traversal II 23](#_Toc446778150)

[Easy-题目28：27.Remove Element 24](#_Toc446778151)

[Easy-题目29：26. Remove Duplicates from Sorted Array 25](#_Toc446778152)

[Easy-题目30：66. Plus One 26](#_Toc446778153)

[Easy-题目31：118. Pascal's Triangle 27](#_Toc446778154)

[Easy-题目32：172. Factorial Trailing Zeroes 28](#_Toc446778155)

[Easy-题目33：102. Binary Tree Level Order Traversal 29](#_Toc446778156)

[Easy-题目34：119. Pascal's Triangle II 30](#_Toc446778157)

[Easy-题目35：9. Palindrome Number 31](#_Toc446778158)

[Easy-题目36：112. Path Sum 31](#_Toc446778159)

[Easy-题目37：111. Minimum Depth of Binary Tree 32](#_Toc446778160)

[Easy-题目38：160. Intersection of Two Linked Lists 33](#_Toc446778161)

[Easy-题目39：88. Merge Sorted Array 34](#_Toc446778162)

[Easy-题目40：36. Valid Sudoku 35](#_Toc446778163)

[Easy-题目41：219. Contains Duplicate II 37](#_Toc446778164)

[Easy-题目42：223. Rectangle Area 37](#_Toc446778165)

[Easy-题目43：190. Reverse Bits 38](#_Toc446778166)

[Easy-题目44：19. Remove Nth Node From End of List 39](#_Toc446778167)

[Easy-题目45：58. Length of Last Word 40](#_Toc446778168)

[Easy-题目46：205. Isomorphic Strings 41](#_Toc446778169)

[Easy-题目47：20. Valid Parentheses 42](#_Toc446778170)

[Easy-题目48：299. Bulls and Cows 43](#_Toc446778171)

[Easy-题目49：290. Word Pattern 45](#_Toc446778172)

[Easy-题目50：38. Count and Say 46](#_Toc446778173)

[Easy-题目51：203. Remove Linked List Elements 47](#_Toc446778174)

[Easy-题目52：14. Longest Common Prefix 48](#_Toc446778175)

[Easy-题目53：257. Binary Tree Paths 49](#_Toc446778176)

[Easy-题目54：234. Palindrome Linked List 50](#_Toc446778177)

[Easy-题目55：67. Add Binary 51](#_Toc446778178)

[Easy-题目56：303. Range Sum Query – Immutable 52](#_Toc446778179)

[Easy-题目57：28. Implement strStr() 53](#_Toc446778180)

[Easy-题目58：7. Reverse Integer 55](#_Toc446778181)

[Easy-题目59：204. Count Primes 55](#_Toc446778182)

[Easy-题目60：125. Valid Palindrome 57](#_Toc446778183)

[Easy-题目61：6. ZigZag Conversion 58](#_Toc446778184)

[Easy-题目62：228. Summary Ranges 60](#_Toc446778185)

[Easy-题目63：278. First Bad Version 61](#_Toc446778186)

[Easy-题目64：168. Excel Sheet Column Title 62](#_Toc446778187)

[Easy-题目65：189. Rotate Array 63](#_Toc446778188)

[Easy-题目66：165. Compare Version Numbers 64](#_Toc446778189)

[Easy-题目67：8. String to Integer (atoi) 65](#_Toc446778190)

[Easy-题目68：1. Two Sum（增补1） 66](#_Toc446778191)

[Middle-题目1：136. Single Number 67](#_Toc446778192)

[Middle-题目2：260. Single Number III 67](#_Toc446778193)

[Middle-题目3：238. Product of Array Except Self 69](#_Toc446778194)

[Middle-题目4：122. Best Time to Buy and Sell Stock II 71](#_Toc446778195)

[Middle-题目5：319. Bulb Switcher 71](#_Toc446778196)

[Middle-题目6：268. Missing Number 73](#_Toc446778197)

[Middle-题目7：144. Binary Tree Preorder Traversal 73](#_Toc446778198)

[Middle-题目8：144. Binary Tree Preorder Traversal 74](#_Toc446778199)

[Middle-题目9：318. Maximum Product of Word Lengths 75](#_Toc446778200)

[Middle-题目10：12. Integer to Roman 76](#_Toc446778201)

[Middle-题目11：137. Single Number II 78](#_Toc446778202)

[Middle-题目12：96. Unique Binary Search Trees 79](#_Toc446778203)

[Middle-题目13：35. Search Insert Position 80](#_Toc446778204)

[Middle-题目14：141. Linked List Cycle 81](#_Toc446778205)

[Middle-题目15：108. Convert Sorted Array to Binary Search Tree 82](#_Toc446778206)

[Middle-题目16：230. Kth Smallest Element in a BST 82](#_Toc446778207)

[Middle-题目17：116. Populating Next Right Pointers in Each Node 83](#_Toc446778208)

[Middle-题目18：53. Maximum Subarray 85](#_Toc446778209)

[Middle-题目19：309. Best Time to Buy and Sell Stock with Cooldown 85](#_Toc446778210)

[Middle-题目20：153. Find Minimum in Rotated Sorted Array 87](#_Toc446778211)

[Middle-题目21：62. Unique Paths 87](#_Toc446778212)

[Middle-题目22：89. Gray Code 88](#_Toc446778213)

[Middle-题目23：121. Best Time to Buy and Sell Stock 89](#_Toc446778214)

[Middle-题目24：46. Permutations 90](#_Toc446778215)

[Middle-题目25：24. Swap Nodes in Pairs 92](#_Toc446778216)

[Middle-题目26：216. Combination Sum III 92](#_Toc446778217)

[Middle-题目27：64. Minimum Path Sum 94](#_Toc446778218)

[Middle-题目28：75. Sort Colors 95](#_Toc446778219)

[Middle-题目29：59. Spiral Matrix II 96](#_Toc446778220)

[Middle-题目30：48. Rotate Image 98](#_Toc446778221)

[Middle-题目31：11. Container With Most Water 99](#_Toc446778222)

[Middle-题目32：240. Search a 2D Matrix II 100](#_Toc446778223)

[Middle-题目33：300. Longest Increasing Subsequence 101](#_Toc446778224)

[Middle-题目34：289. Game of Life 103](#_Toc446778225)

[Middle-题目35：77. Combinations 105](#_Toc446778226)

[Middle-题目36：74. Search a 2D Matrix 106](#_Toc446778227)

[Middle-题目37：199. Binary Tree Right Side View 107](#_Toc446778228)

[Middle-题目38：241. Different Ways to Add Parentheses 109](#_Toc446778229)

[Middle-题目39：73. Set Matrix Zeroes 110](#_Toc446778230)

[Middle-题目40：162. Find Peak Element 113](#_Toc446778231)

[Middle-题目41：313. Super Ugly Number 114](#_Toc446778232)

[Middle-题目42/43：274. H-Index && 275. H-Index II 116](#_Toc446778233)

[Middle-题目44：334. Increasing Triplet Subsequence 118](#_Toc446778234)

[Middle-题目45：215. Kth Largest Element in an Array 119](#_Toc446778235)

[Middle-题目46：129. Sum Root to Leaf Numbers 121](#_Toc446778236)

[Middle-题目47：80. Remove Duplicates from Sorted Array II 122](#_Toc446778237)

[Middle-题目48：81. Search in Rotated Sorted Array II 123](#_Toc446778238)

[Middle-题目49：279. Perfect Squares 123](#_Toc446778239)

[Middle-题目50：142. Linked List Cycle II 125](#_Toc446778240)

[Middle-题目51：331. Verify Preorder Serialization of a Binary Tree 126](#_Toc446778241)

[Middle-题目52：78. Subsets 128](#_Toc446778242)

[Middle-题目53：114. Flatten Binary Tree to Linked List 129](#_Toc446778243)

[Middle-题目54：39. Combination Sum 130](#_Toc446778244)

[Middle-题目55：329. Longest Increasing Path in a Matrix 132](#_Toc446778245)

[Middle-题目56：109. Convert Sorted List to Binary Search Tree 133](#_Toc446778246)

[Middle-题目57：90. Subsets II 134](#_Toc446778247)

[Middle-题目58：213. House Robber II 135](#_Toc446778248)

[Middle-题目59：201. Bitwise AND of Numbers Range 137](#_Toc446778249)

[Middle-题目60：120. Triangle 138](#_Toc446778250)

[Middle-题目61：63. Unique Paths II 139](#_Toc446778251)

[Middle-题目62：86. Partition List 140](#_Toc446778252)

[Middle-题目63：147. Insertion Sort List 142](#_Toc446778253)

[Middle-题目64：95. Unique Binary Search Trees II 143](#_Toc446778254)

[Middle-题目65：34. Search for a Range 144](#_Toc446778255)

[Middle-题目66：236. Lowest Common Ancestor of a Binary Tree 145](#_Toc446778256)

[Middle-题目67：16. 3Sum Closest 146](#_Toc446778257)

[Middle-题目68：106. Construct Binary Tree from Inorder and Postorder Traversal 147](#_Toc446778258)

[Middle-题目69：330. Patching Array 148](#_Toc446778259)

[Middle-题目70：103. Binary Tree Zigzag Level Order Traversal 149](#_Toc446778260)

[Middle-题目71：105. Construct Binary Tree from Preorder and Inorder Traversal 151](#_Toc446778261)

[Middle-题目72：17. Letter Combinations of a Phone Number 152](#_Toc446778262)

[Middle-题目73：55. Jump Game 153](#_Toc446778263)

[Middle-题目74：50. Pow(x, n) 154](#_Toc446778264)

[Middle-题目75：113. Path Sum II 154](#_Toc446778265)

[Middle-题目76：47. Permutations II 155](#_Toc446778266)

[Middle-题目77：92. Reverse Linked List II 157](#_Toc446778267)

[Middle-题目78：131. Palindrome Partitioning 158](#_Toc446778268)

[Middle-题目79：40. Combination Sum II 160](#_Toc446778269)

[Middle-题目80：200. Number of Islands 161](#_Toc446778270)

[Middle-题目81：264. Ugly Number II 162](#_Toc446778271)

[Middle-题目82：134. Gas Station 164](#_Toc446778272)

[Middle-题目83：49. Group Anagrams 165](#_Toc446778273)

[Middle-题目84：82. Remove Duplicates from Sorted List II 166](#_Toc446778274)

[Middle-题目85：310. Minimum Height Trees 167](#_Toc446778275)

[Middle-题目86：207. Course Schedule 170](#_Toc446778276)

[Middle-题目87：209. Minimum Size Subarray Sum 172](#_Toc446778277)

[Middle-题目88：31. Next Permutation 173](#_Toc446778278)

[Middle-题目89：229. Majority Element II 175](#_Toc446778279)

[Middle-题目90：306. Additive Number 177](#_Toc446778280)

[Middle-题目91：69. Sqrt(x) 178](#_Toc446778281)

[Middle-题目92：139. Word Break 178](#_Toc446778282)

[Middle-题目93：60. Permutation Sequence 180](#_Toc446778283)

[Middle-题目94：322. Coin Change 181](#_Toc446778284)

[Middle-题目95：222. Count Complete Tree Nodes 183](#_Toc446778285)

[Middle-题目96：187. Repeated DNA Sequences 184](#_Toc446778286)

[Middle-题目97：148. Sort List 185](#_Toc446778287)

[Middle-题目98：316. Remove Duplicate Letters 187](#_Toc446778288)

[Middle-题目99：227. Basic Calculator II 189](#_Toc446778289)

[Middle-题目100：18. 4Sum 190](#_Toc446778290)

[Middle-题目101：332. Reconstruct Itinerary 192](#_Toc446778291)

[Middle-题目102：150. Evaluate Reverse Polish Notation 193](#_Toc446778292)

[Middle-题目103：221. Maximal Square 194](#_Toc446778293)

[Middle-题目104：43. Multiply Strings 196](#_Toc446778294)

[Middle-题目105：93. Restore IP Addresses 196](#_Toc446778295)

[Middle-题目106：5. Longest Palindromic Substring 198](#_Toc446778296)

[Middle-题目107：61. Rotate List 200](#_Toc446778297)

[Middle-题目108：79. Word Search 201](#_Toc446778298)

[Middle-题目109：143. Reorder List 203](#_Toc446778299)

[Middle-题目110：2. Add Two Numbers 203](#_Toc446778300)

[Middle-题目111：54. Spiral Matrix 205](#_Toc446778301)

[Middle-题目112：152. Maximum Product Subarray 207](#_Toc446778302)

[Middle-题目113：324. Wiggle Sort II 208](#_Toc446778303)

[Middle-题目114：3. Longest Substring Without Repeating Characters 210](#_Toc446778304)

[Middle-题目115：71. Simplify Path 211](#_Toc446778305)

[Middle-题目116：224. Basic Calculator 212](#_Toc446778306)

[Middle-题目117：98. Validate Binary Search Tree 212](#_Toc446778307)

[Middle-题目118：210. Course Schedule II 213](#_Toc446778308)

[Middle-题目119：127. Word Ladder 215](#_Toc446778309)

[Middle-题目120：179. Largest Number 217](#_Toc446778310)

[Middle-题目121：15. 3Sum 218](#_Toc446778311)

[Middle-题目122：220. Contains Duplicate III 219](#_Toc446778312)

[Middle-题目123：335. Self Crossing 220](#_Toc446778313)

[Middle-题目124：91. Decode Ways 222](#_Toc446778314)

[Middle-题目125：130. Surrounded Regions 223](#_Toc446778315)

[Middle-题目126：151. Reverse Words in a String 225](#_Toc446778316)

[Middle-题目127：29. Divide Two Integers 226](#_Toc446778317)

[Middle-题目128：166. Fraction to Recurring Decimal 226](#_Toc446778318)

[Middle-题目129：337. House Robber III（增补1） 228](#_Toc446778319)

[Middle-题目130：338. Counting Bits（增补2） 230](#_Toc446778320)

[Hard-题目1：52. N-Queens II 231](#_Toc446778321)

[Hard-题目2：287. Find the Duplicate Number 231](#_Toc446778322)

[Hard-题目3：312. Burst Balloons 232](#_Toc446778323)

[Hard-题目4：145. Binary Tree Postorder Traversal 233](#_Toc446778324)

[Hard-题目5：154. Find Minimum in Rotated Sorted Array II 235](#_Toc446778325)

[Hard-题目6：117. Populating Next Right Pointers in Each Node II 235](#_Toc446778326)

[Hard-题目7：42. Trapping Rain Water 237](#_Toc446778327)

[Hard-题目8：128. Longest Consecutive Sequence 238](#_Toc446778328)

[Hard-题目9：301. Remove Invalid Parentheses 240](#_Toc446778329)

[Hard-题目10：33. Search in Rotated Sorted Array 242](#_Toc446778330)

[Hard-题目11：315. Count of Smaller Numbers After Self 242](#_Toc446778331)

[Hard-题目12：115. Distinct Subsequences 245](#_Toc446778332)

[Hard-题目13：72. Edit Distance 246](#_Toc446778333)

[Hard-题目14：25. Reverse Nodes in k-Group 248](#_Toc446778334)

[Hard-题目15：239. Sliding Window Maximum 249](#_Toc446778335)

[Hard-题目16：164. Maximum Gap 250](#_Toc446778336)

[Hard-题目17：87. Scramble String 251](#_Toc446778337)

[Hard-题目18：51. N-Queens 254](#_Toc446778338)

# Easy-题目1：292. Nim Game

## 题目原文：

You are playing the following Nim Game with your friend: There is a heap of stones on the table, each time one of you take turns to remove 1 to 3 stones. The one who removes the last stone will be the winner. You will take the first turn to remove the stones.

Both of you are very clever and have optimal strategies for the game. Write a function to determine whether you can win the game given the number of stones in the heap.

For example, if there are 4 stones in the heap, then you will never win the game: no matter 1, 2, or 3 stones you remove, the last stone will always be removed by your friend.

## 题目大意：

桌子上有一堆石子，每次拿1-3个石子，拿到最后一个石子的人是赢家。你是第一个拿石子的人，石子的个数满足什么条件你可以保证必胜？

## 题目分析：

若石子的个数是4的倍数则保证先拿的人必输。设先拿的人每轮拿x个石子，则后拿的人拿4-x个石子，那么最后一个石子肯定是后拿的人得到。反之若石子的个数不是4的倍数，则先拿的人必胜。因为可以第一轮拿x个石子使得剩下的石子是4的倍数，再由你的对手开始拿，按上述策略你必胜。因此算法即求n是否为4的倍数。

## 源码：（language： c）

bool canWinNim(int n) {

return n%4;

}

## 成绩：

0ms，beats 0.18% 众数：0ms,99.82%

## Cmershen的碎碎念：

这道题不知道为什么会出现在Leetcode上，也不知道为什么还有0.18%的提交代码不是0ms。难道你们真的进行了模拟？

# Easy-题目2：258. Add Digits

## 题目原文：

Given a non-negative integer num, repeatedly add all its digits until the result has only one digit.

For example:

Given num = 38, the process is like: 3 + 8 = 11, 1 + 1 = 2. Since 2 has only one digit, return it.

Follow up:

Could you do it without any loop/recursion in O(1) runtime?

## 题目大意：

给一个正整数num，重复地计算每一个数位上数字之和，直到只剩一位数。

例如：给数字38，则3+8=11,1+1=2，那么返回2.

你可以不需要循环或递归，在O(1)时间完成它吗？

## 题目分析：

根据Hint中给出的wiki链接，这道题叫做数字根问题。求一个正整数n的数字根有如下公式： \mbox{dr}(n) = 1\ +\ ((n-1)\ {\rm mod}\ 9).\ 因此可以一行解决。

## 源码：（language：c）

int addDigits(int num) {

return 1+(num-1)%9;

}

## 成绩：

4ms,beats56.87% 众数：8ms 52.28%

## Cmershen的碎碎念：

如果不给出hint，我是怎么也推不出这个公式。在这里感叹数学的强大，也希望有人能给出这个公式的证明。

# Easy-题目3：104. Maximum Depth of Binary Tree

## 题目原文：

Given a binary tree, find its maximum depth.

## 题目大意：

给出一个二叉树，求最大高度。

## 题目分析：

若节点为空，则返回0，否则返回左子树和右子树高度的最大值+1（根节点的高度）。依然一行代码解决。

## 源码：（language：java）

public class Solution {

public int maxDepth(TreeNode root) {

return (root==null)?0:Math.max(maxDepth(root.left),maxDepth(root.right))+1;

}

}

## 成绩：

1ms,beats 9.54%,众数1ms,89.37%

# Easy-题目4：226. Invert Binary Tree

## 题目原文：

Invert a binary tree.

4

/ \

2 7

/ \ / \

1 3 6 9

to

4

/ \

7 2

/ \ / \

9 6 3 1

Trivia:

This problem was inspired by this original tweet by Max Howell:

Google: 90% of our engineers use the software you wrote (Homebrew), but you can’t invert a binary tree on a whiteboard so fuck off.

## 题目大意：

翻转一个二叉树。

琐事：

这个问题由HomeBrew软件的开发者Max Howell提出：

谷歌：虽然我们 90% 工程师都在用你写的软件（Homebrew），但你不能在白板上反转二叉树，所以滚蛋。

## 题目分析：

使用递归解法，对空二叉树直接退出，否则递归翻转左右子树并安到对应的位置上。

## 源码：（language：c）

struct TreeNode\* invertTree(struct TreeNode\* root) {

if(!root)

return NULL;

struct TreeNode\* temp = root->left;

root->left = invertTree(root->right);

root->right = invertTree(temp);

return root;

}

## 成绩：

0ms,beats0%.众数：0ms.100%

## Cmershen的碎碎念：

一开始拿到此题时，想想连Homebrew作者都不会的题，我能会么。后来考虑了很多情况，例如叶子节点，只有一个孩子的节点等情况，代码写了很长。但接下来发现所有情况可以归结为空与非空两种情况，所以写出上述的ac代码应该是足够简洁且易理解了。

# Easy-题目5：237. Delete Node in a Linked List

## 题目原文：

Write a function to delete a node (except the tail) in a singly linked list, given only access to that node.

Supposed the linked list is 1 -> 2 -> 3 -> 4 and you are given the third node with value 3, the linked list should become 1 -> 2 -> 4 after calling your function.

## 题目大意：

删除单链表中指定节点。

## 题目分析：

把指定节点的值改成后面的值，并令其后继节点指向原后继节点的后继节点。

## 源码：（language：c）

void deleteNode(struct ListNode\* node) {

if(node) {

node->val=node->next->val;

node->next=node->next->next;

}

}

## 成绩：

4ms,beats 0.52% 众数4ms,99.48%

## Cmershen的碎碎念：

在数据结构课上，我们删除链表节点的方法是令其前驱节点指向后继节点。可是本题中只有当前节点，又是单链表，无法获得其前驱节点。这个问题困扰了我一段时间，但我发现还有value值可以利用，因此得出上述算法。

其实测试用例不完全，根据源码，Node->next是有可能为空的，再引用val和next有可能引发NPD异常。因此上述的ac代码不可以用于删除链表中最后一个节点。（dts大法好！！！(\*^\_\_^\*) 嘻嘻……）

# Easy-题目6：283. Move Zeroes

## 题目原文：

Given an array nums, write a function to move all 0's to the end of it while maintaining the relative order of the non-zero elements.

For example, given nums = [0, 1, 0, 3, 12], after calling your function, nums should be [1, 3, 12, 0, 0].

## 题目大意：

给出一个数组，把所有的0都搬到后面，并且保持原来非0数的顺序不变。

## 题目分析：

使用两个指针temp和i，i每次递增，temp只有在i对应数字非0的时候递增，这样把所有非0的数都搬到前面来，temp记下了第一个0的下标，再把temp以后的数字全部置0。

## 源码：（language：java）

public class Solution {

public void moveZeroes(int[] nums) {

int temp=0;

for(int i=0;i<nums.length;i++) {

if(nums[i]!=0)

nums[temp++]=nums[i];

}

for(;temp<nums.length;temp++) {

nums[temp]=0;

}

}

}

## 成绩：

0ms，beats88.04% 众数1ms,64.54%

# Easy-题目7：100. Same Tree

## 题目原文：

Given two binary trees, write a function to check if they are equal or not.

## 题目大意：

判断两个二叉树是不是全等的。

## 题目分析：

1. 若两树都空，则全等。
2. 若一树空一树不空，则不全等。
3. 若两树的根节点对应的值不同则不全等。
4. 递归判断左子树和右子树是否相等。

## 源码：（language：java）

public class Solution {

public boolean isSameTree(TreeNode p, TreeNode q) {

if(p==null && q== null)

return true;

else if(p==null && q!=null)

return false;

else if(p!=null && q==null)

return false;

else if(p.val != q.val)

return false;

else

return isSameTree(p.left,q.left) && isSameTree(p.right,q.right);

}

}

## 成绩：

0ms,beats14.78%.众数0ms,85.22%

## Cmershen的碎碎念：

其实本题的ac代码可以进一步简化，甚至简化到1行。但为了可读性仍然保持较复杂的代码。

# Easy-题目8：242. Valid Anagram

## 题目原文：

Given two strings *s* and *t*, write a function to determine if *t* is an anagram of *s*.

For example,  
*s* = "anagram", *t* = "nagaram", return true.  
*s* = "rat", *t* = "car", return false.

**Note:**  
You may assume the string contains only lowercase alphabets.

## 题目大意：

给出两个字符串s和t，写出字符串判断s和t是不是anagram。

(百度词典：anagram n. 由颠倒字母顺序而构成的字)

注意：s和t串中只有小写字母。

## 题目分析：

对两个字符串排序，再比较排序后的两个字符数组是否相等。时间复杂度nlogn.,再进行适当判断。

## 源码：（language：java）

public class Solution {

public boolean isAnagram(String s, String t) {

if(s==null||t==null||s.length()!=t.length()){

return false;

}

char[] array1 = s.toCharArray();

char[] array2 = t.toCharArray();

Arrays.sort(array1);

Arrays.sort(array2);

return Arrays.equals(array1, array2);

}

}

## 成绩：

6ms,beats 78.41%.众数：7ms 18.48%

## Cmershen的碎碎念：

一开始想到用HashSet，后来想到字母中可能有重复，就发现自己走了弯路。

# Easy-题目9：171. Excel Sheet Column Number

## 题目原文：

Given a column title as appear in an Excel sheet, return its corresponding column number.

For example:

A -> 1

B -> 2

C -> 3

...

Z -> 26

AA -> 27

AB -> 28

## 题目大意：

把Excel中的列号转换成自然数的列号。

## 题目分析：

递归计算，列号长度为0时对应0，否则为除最后一位的子串对应列号\*26加末尾一位字母的序号。

例如：ABC列对应的自然数是 AB列对应的自然数（28）\*26+C的字母序号（3）。

## 源码：（language：java）

public class Solution {

public int titleToNumber(String s) {

return (s.length()==0)?0:titleToNumber(s.substring(0, s.length()-1))\*26+s.charAt(s.length()-1)-'A'+1;

}

}

## 成绩：

3ms,beats 11.31%,众数3ms,66.06%

# Easy-题目10：217. Contains Duplicate

## 题目原文：

Given an array of integers, find if the array contains any duplicates. Your function should return true if any value appears at least twice in the array, and it should return false if every element is distinct.

## 题目大意：

判断一个整形数组里面是否含有重复(duplicate)的数字.

## 题目分析：

利用HashSet中集合的唯一性。若集合中存在的元素再次被add，则添加失败。此时返回true。

## 源码：（language：java）

public class Solution {

public boolean containsDuplicate(int[] nums) {

HashSet<Integer> set=new HashSet<Integer>();

for(int i:nums) {

if(!set.add(i))

return true;

}

return false;

}

}

## 成绩：

9ms，beats 61.51%,众数16ms，16.44%

# Easy-题目11：169. Majority Element

## 题目原文：

Given an array of size *n*, find the majority element. The majority element is the element that appears more than ⌊ n/2 ⌋ times.

You may assume that the array is non-empty and the majority element always exist in the array.

## 题目大意：

找出一个数组的主元素，即在数组中出现的次数>n/2.

## 分析：

使用repeatTimes变量记录主元素重复次数，每次遇到重复的就+1，不重复的就-1.减到0时就更新当前主元素。（利用主元素出现次数比其他数出现之和多这一性质。）

## 源码：（language：c）

int majorityElement(int\* nums, int numsSize) {

int i,mainElement,repeatTimes = 0;

//每次若不相同则减小这个计数，若相同则增加

for(i = 0;i<numsSize;i++) {

if(repeatTimes == 0){

mainElement = nums[i];

repeatTimes = 1;

}

else{

if(mainElement == nums[i])

repeatTimes++;

else

repeatTimes--;

}

}

return mainElement;

}

## 成绩：

8ms,beats 36.16%.众数：8ms 63.84%

## Cmershen的碎碎念：

此题好像是考研408模拟题中的一道题，听说也经常在面试中出现。一开始我想到的是用HashMap记录每个元素出现的次数，最大的就是主元素。但这样的空间复杂度就是O(n),而上述算法有更优的O(1)空间复杂度。

# Easy-题目12：206. Reverse Linked List

## 题目原文：

Reverse a singly linked list.

## 题目大意：

翻转单链表。

## 题目分析：

使用递归函数，先把头结点后面的链表翻转过来，再把头结点安到末尾即可。

## 源码：（language：c）

struct ListNode\* reverseList(struct ListNode\* head) {

if(!head || !head->next)

return head;

else {

struct ListNode \*temp=head->next;

struct ListNode \*newHead=reverseList(head->next);

temp->next=head;

head->next=NULL;

return newHead;

}

}

## 成绩：

0ms，beats 17.11%.众数0ms,82.89%

## Cmershen的碎碎念：

使用temp指针指向原链表的第二个节点，这样翻转后就到了最后一个节点，把原来的head安到temp后面即可。

# Easy-题目13：13. Roman to Integer

## 题目原文：

Given a roman numeral, convert it to an integer.

Input is guaranteed to be within the range from 1 to 3999.

## 题目大意：

把1-3999之内的罗马数字转换为整数。

## 题目分析：

先用HashMap记录数字与字母的映射关系，再根据罗马数字的换算法则逐位进行转换。遇到40,4等前面数比后面数小的情况时，要将两位看做一个整体进行换算。

## 源码：（language：java）

public class Solution {

public int romanToInt(String s) {

int num=0;

HashMap<Character,Integer> convert=new HashMap<Character,Integer>();

convert.put('I', 1);

convert.put('V', 5);

convert.put('X', 10);

convert.put('L', 50);

convert.put('C', 100);

convert.put('D', 500);

convert.put('M', 1000);

for(int i=0;i<s.length();i++) {

int num1=convert.get(s.charAt(i));

int num2=0;

if(i!=s.length()-1)

num2=convert.get(s.charAt(i+1));

if(num1>=num2)

num+=num1;

else {

num+=(num2-num1);

i++;

}

}

return num;

}

}

## 成绩：

18ms，beats24.69% 众数7ms，15.29%

## Cmershen的碎碎念：

1. 此题成绩很差，可能是HashMap的效率过低造成的，但总体的时间复杂度为O（n）。当然可以暴力穷举，用switch + 3999个case语句完成此题。（这个时间复杂度显然为O(1)但谁会那么无聊呢？）
2. 罗马数字计算方法（摘自百度百科）：
3. 七个基本符号：I（1）、V（5）、X（10）、L（50）、C（100）、D（500）、 M（1000）
4. 相同的数字连写，所表示的数等于这些数字相加得到的数，如 Ⅲ=3；
5. 小的数字在大的数字的右边，所表示的数等于这些数字相加得到的数，如 Ⅷ=8、Ⅻ=12；
6. 小的数字（限于 Ⅰ、X 和 C）在大的数字的左边，所表示的数等于大数减小数得到的数，如 Ⅳ=4、Ⅸ=9；
7. 在一个数的上面画一条横线，表示这个数增值 1,000 倍，如=5000。

# Easy-题目14：235.Lowest Common Ancestor of a Binary Search Tree

## 题目原文：

Given a binary search tree (BST), find the lowest common ancestor (LCA) of two given nodes in the BST.

According to the definition of LCA on Wikipedia: “The lowest common ancestor is defined between two nodes v and w as the lowest node in T that has both v and w as descendants (where we allow a node to be a descendant of itself).”

\_\_\_\_\_\_\_6\_\_\_\_\_\_

/ \

\_\_\_2\_\_ \_\_\_8\_\_

/ \ / \

0 \_4 7 9

/ \

3 5

For example, the lowest common ancestor (LCA) of nodes 2 and 8 is 6. Another example is LCA of nodes 2 and 4 is 2, since a node can be a descendant of itself according to the LCA definition.

## 题目大意：

给出一个二叉排序树，求两个节点的最近公共祖先。（一个节点的祖先也可以是自己）

## 题目分析：

1. 空节点的祖先是空；
2. 若当前根节点的值比两个节点的值都大，则公共祖先在左子树里，递归向左子树搜索；
3. 若当前根节点的值比两个节点的值都小，则公共祖先在右子树里，递归向右子树搜索；
4. 上述情况均不成立，则当前节点就是公共祖先。

## 源码：（language：c）

struct TreeNode\* lowestCommonAncestor(struct TreeNode\* root, struct TreeNode\* p, struct TreeNode\* q) {

if (!root || !p || !q)

return NULL;

else if (root->val < p->val && root->val < q->val)

return lowestCommonAncestor(root->right, p, q);

else if (root->val > p->val && root->val > q->val)

return lowestCommonAncestor(root->left, p, q);

else

return root;

}

## 成绩：

24ms,beats 22.03%,众数24ms，77.40%

## Cmershen的碎碎念：

注意利用二叉排序树的定义的性质。总结如下：

二叉排序树或者是一棵空树，或者是具有下列性质的二叉树：

（1）若左子树不空，则左子树上所有结点的值均小于它的根结点的值；

（2）若右子树不空，则右子树上所有结点的值均大于它的根结点的值；

（3）左、右子树也分别为二叉排序树；

（4）没有键值相等的节点。

性质：二叉排序树的中序遍历是递增序列。

# Easy-题目15：191. Number of 1 Bits

## 题目原文：

Write a function that takes an unsigned integer and returns the number of ’1' bits it has (also known as the Hamming weight).

For example, the 32-bit integer ’11' has binary representation 00000000000000000000000000001011, so the function should return 3.

## 题目大意：

给出一个整数，求它的二进制形式里面有多少个‘1’。

## 题目分析：

如果n 只有1位（0或1），直接返回n，否则返回n的最后一位加上n右移一位所得数中1的个数（递归计算）。

## 源码：（language：c）

int hammingWeight(uint32\_t n) {

return (n==0||n==1)?n:(hammingWeight(n/2)+n%2);

}

## 成绩：

4ms，beats3.83% 众数4ms，96.17%

## Cmershen的碎碎念：

Java中有一个库函数可以水过此题：Integer.bitCount(n);

# Easy-题目16：328. Odd Even Linked List

## 题目原文：

Given a singly linked list, group all odd nodes together followed by the even nodes. Please note here we are talking about the node number and not the value in the nodes.

You should try to do it in place. The program should run in O(1) space complexity and O(nodes) time complexity.

Example:

Given 1->2->3->4->5->NULL,

return 1->3->5->2->4->NULL.

## 题目大意：

给出一个单链表，把奇数节点放在偶数节点的前面。

## 题目分析：

新开两个头结点oddHead和evenHead，然后扫链表，交替的接在两个新头结点的后面，最后把odd链表的尾节点接到even链表的头结点上。

## 源码：（language：java）

public class Solution {

public ListNode oddEvenList(ListNode head) {

if(head == null)

return head;

ListNode oddHead = head,evenHead =head.next;

ListNode prevOdd = oddHead,prevEven = evenHead;

while(prevOdd.next != null && prevEven.next != null){

prevOdd.next = prevEven.next;

prevOdd = prevOdd.next;

prevEven.next = prevOdd.next;

prevEven = prevEven.next;

}

prevOdd.next = evenHead;

return oddHead;

}

}

## 成绩：

1ms,beats 5.71%,众数1ms,92.88%

# Easy-题目17：83. Remove Duplicates from Sorted List

## 题目原文：

Given a sorted linked list, delete all duplicates such that each element appear only once.

For example,

Given 1->1->2, return 1->2.

Given 1->1->2->3->3, return 1->2->3.

## 题目大意：

给出一个排序的单链表，删除所有重复元素使得每个元素仅出现一次。

## 题目分析：

判断当前节点的值是否等于其后继节点，若等于则删除后继节点（p->next=p->next->next），否则当前节点向右移动(p=p->next)。

## 源码：（language：c）

struct ListNode\* deleteDuplicates(struct ListNode\* head) {

struct ListNode\* p = head;

if(!head)

return;

while(p->next) {

if(p->val == p->next->val)

p->next=p->next->next;

else

p=p->next;

}

return head;

}

## 成绩：

4ms,beats 10.82%,众数4ms,89.18%

# Easy-题目18：70. Climbing Stairs

## 题目原文：

You are climbing a stair case. It takes *n* steps to reach to the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

## 题目大意：

你在爬一个n阶的楼梯，每次可以上1阶也可以上2阶。有多少种不同的爬法？

## 题目分析：

设爬n阶的楼梯不同的方法是f(n),则最后一次可以爬1阶也可以爬2阶。因此f(n)=f(n-1)+f(n-2).这个公式很熟悉吧。。没错此题即是求斐波那契数列的n+1项。为什么是n+1项呢，因为斐波那契数列是1,1,2,3,….而爬楼梯问题的方法数是1,2,3,5…

## 源码：（language：c）

int climbStairs(int n) {

int i,sum1=1,sum2=2,sum=0;

if(n==1||n==2)

return n;

else {

for(i=2;i<n;i++) {

sum=sum1+sum2;

sum1=sum2;

sum2=sum;

}

return sum;

}

}

## 成绩：

0ms，beats0.16% 众数0ms,99.84%.

## Cmershen的碎碎念：

求斐波那契数列切勿用递推公式f(n)=f(n-1)+f(n-2)，会引起超时。因为随着n的增加会大量出现重复计算，使用递推的算法复杂度是NP难的，而本算法是O(n)复杂度的。如果使用斐波那契的通项公式（什么？你不知道斐波那契可以求通项？回去复习高中数学一百遍！）算法复杂度可降低到O(1)，但通项中的系数是无理数，在计算机中不易表示。

斐波那契数列通项公式如下：

http://h.hiphotos.baidu.com/baike/s%3D246/sign=42c5afe59c25bc312f5d069c68df8de7/3ac79f3df8dcd100d591820a778b4710b9122f8f.jpg

# Easy-题目19：263. Ugly Number

## 题目原文：

Write a program to check whether a given number is an ugly number.

Ugly numbers are positive numbers whose prime factors only include 2, 3, 5. For example, 6, 8 are ugly while 14 is not ugly since it includes another prime factor 7.

Note that 1 is typically treated as an ugly number.

## 题目大意：

判断一个数是不是丑陋数。

丑陋数的定义：丑陋数是因子中只包含于2，3，5的正整数。特殊地，1是丑陋数。

## 题目分析：

直观地，用2,3,5分别试除这个数，若可以整除直至得到1，则该数是丑陋数。

## 源码：（language：java）

public class Solution {

public boolean isUgly(int num) {

if(num==0)

return false;

while(num!=1) {

if(num%2==0)

num/=2;

else if(num%3==0)

num/=3;

else if(num%5==0)

num/=5;

else

return false;

}

return true;

}

}

## 成绩：

2ms，beats 18.00%,众数2ms，82.00%

## Cmershen的碎碎念：

此题显然有递归解法，因为n是丑数等价于n/2,n/3,n/5中有一个是丑数。但是提交递归解法会超时。

# Easy-题目20：202. Happy Number

## 题目原文：

Write an algorithm to determine if a number is "happy".

A happy number is a number defined by the following process: Starting with any positive integer, replace the number by the sum of the squares of its digits, and repeat the process until the number equals 1 (where it will stay), or it loops endlessly in a cycle which does not include 1. Those numbers for which this process ends in 1 are happy numbers.

**Example:**19 is a happy number

* 12 + 92 = 82
* 82 + 22 = 68
* 62 + 82 = 100
* 12 + 02 + 02 = 1

## 题目大意：

判断一个数是不是“快乐的”。

快乐数的定义是：把一个正整数每个数位上的数求平方和，得到的数再求平方和……若能够得到1则为快乐数。

## 题目分析：

根据百度百科 任何不快乐数都会陷入4 → 16 → 37 → 58 → 89 → 145 → 42 → 20 → 4 的死循环。因此可以设计算法：不断求当前数的平方和，若出现上述数之一则返回false，若得到1则返回true。

## 源码：（language：java）

public class Solution {

public boolean isHappy(int num) {

while(true) {

if(num==1)

return true;

else if(num==4||num==16||num==37||num==58||num==89||num==145||num==42||num==20)

return false;

else {

int newnum=0;

while(num!=0) {

newnum=newnum+(num%10)\*(num%10);

num=num/10;

}

num=newnum;

}

}

}

}

## 成绩：

2ms，beats 85.98%，众数6ms，25.67%

## Cmershen的碎碎念：

Trivial的方法是，使用一个hashset记录迭代过程中出现的数字，如果出现重复的则不是快乐数。但本算法中利用了重复数字的规律可以减少迭代次数和在hashset中的比较开销，在此第二次感叹数学的强大。

# Easy-题目21：326. Power of Three

## 题目原文：

Given an integer, write a function to determine if it is a power of three.

**Follow up:**  
Could you do it without using any loop / recursion?

## 题目大意：

给出一个整数，判断是不是3的幂。

你能否不使用循环或者递归？

## 题目分析：

如果使用循环或者递归那就很简单了，如果不使用，则判断能否被3^19整除。（为什么是19？因为3^20就超过了int的上限。）

## 源码：（language：c）

使用递归：

bool isPowerOfThree(int n) {

return (n==0||n==1)?n:((n%3)?false:isPowerOfThree(n/3));

}

不使用递归：

bool isPowerOfThree(int n) {

return ( n>0 && 1162261467%n==0);

}

## 成绩：

使用递归：100ms,beats 97.81%

不使用递归：112ms beats 70.84%

众数：112ms,13.13%

## Cmershen的碎碎念：

本题一个不理解的地方是，为什么递归比非递归快。似乎递归的算法是O(logn)的，而非递归的算法是O(1)的。难道是因为计算1162261467 % n是很费时间的？这个问题希望有人给予解释。

# Easy-题目22：231. Power of Two

## 题目原文：

Given an integer, write a function to determine if it is a power of two.

## 题目大意：

给出一个整数，判断是不是2的幂。

## 题目分析：

跟上题如出一辙，但还有一种方法是利用位运算，即2的幂的最高位是1，其余位全是0这一性质，判断n和n-1按位与是否等于0来解决。@15-信安-L。

## 源码：（language：c）

递归解法：

bool isPowerOfTwo(int n) {

return (n==0||n==1)?n:(n%2?0:isPowerOfTwo(n/2));

}

按位与解法：

bool isPowerOfTwo(int n) {

return (n&n-1)==0&&n>0;

}

暴力枚举：

bool isPowerOfTwo(int n) {

return (n==1||n==2||n==4||n==8||n==16||n==32||n==64||n==128||n==256||n==512||n==1024||n==2048||n==4096||n==8192||n==16384||n==32768||n==65536||n==131072||n==262144||n==524288||n==1048576||n==2097152||n==4194304||n==8388608||n==16777216||n==33554432||n==67108864||n==134217728||n==268435456||n==536870912||n==1073741824);

}

## 成绩：

三种算法都是4ms，beats44.98%，众数4ms，55.02%。

# Easy-题目23：21. Merge Two Sorted Lists

## 题目原文：

Merge two sorted linked lists and return it as a new list. The new list should be made by splicing together the nodes of the first two lists.

## 题目大意：

把两个排序的链表合并成一个链表并仍然保持有序。

## 题目分析：

1. 如果一个链表是空的，则返回另一个。
2. 令两个指针l1和l2分别指向A、B两个链表的头，扫两个链表，对应节点谁较小谁就接到新链表的尾部，若相等两个都接上。
3. 当其中一个链表扫完，另一个未扫完的时候，把未扫完的部分接到新链表尾部。

## 源码：（language：java）

public class Solution {

public ListNode mergeTwoLists(ListNode l1, ListNode l2) {

if(l1==null)

return l2;

else if(l2==null)

return l1;

else {

ListNode head=new ListNode(Math.min(l1.val,l2.val));

ListNode temp=head;

if(temp.val==l1.val && temp.val==l2.val) {

head.next=new ListNode(l1.val);

temp=temp.next;

}

if(temp.val==l1.val)

l1=l1.next;

if(temp.val==l2.val)

l2=l2.next;

while(l1!=null && l2!=null) {

if(l1.val<=l2.val) {

ListNode p=new ListNode(l1.val);

temp.next=p;

temp=p;

l1=l1.next;

}

else {

ListNode p=new ListNode(l2.val);

temp.next=p;

temp=p;

l2=l2.next;

}

}

if(l1==null)

temp.next=l2;

else if(l2==null)

temp.next=l1;

return head;

}

}

}

## 成绩：

1ms，beats 12.13%，众数1ms，87.87%

# Easy-题目24：110. Balanced Binary Tree

## 题目原文：

Given a binary tree, determine if it is height-balanced.

## 题目大意：

判断一个树是不是二叉平衡树。

## 题目分析：

二叉平衡树可递归定义如下：

1. 空树是平衡二叉树。
2. 平衡二叉树的左子树和 右子树高度差不超过1，且左右子树都是平衡二叉树。

严格按此定义设计算法即可。

## 源码：（language：java）

public class Solution {

public boolean isBalanced(TreeNode root) {

if(root==null)

return true;

else if(Math.abs(getHeight(root.left)-getHeight(root.right))<=1 && isBalanced(root.left) && isBalanced(root.right))

return true;

else

return false;

}

private int getHeight(TreeNode root)

{

if(root==null)

return 0;

else

return Math.max(getHeight(root.left),getHeight(root.right))+1;

}

}

## 成绩：

2ms，beats 23.38%，众数2ms，64.97%

# Easy-题目25：101. Symmetric Tree

## 题目原文：

Given a binary tree, check whether it is a mirror of itself (ie, symmetric around its center).

## 题目大意：

给一个二叉树，判断它是不是沿中线轴对称的。

## 题目分析：

1. 空树是轴对称的。
2. 轴对称的两棵子树tree1和tree2应满足以下递归条件：
   1. 对应值相等；
   2. Tree1的右子树和tree2的左子树是轴对称的；
   3. Tree1的左子树和tree2的右子树的轴对称的。

## 源码：（language：java）

public class Solution {

public boolean isSymmetric(TreeNode root) {

if(root== null)

return true;

return isSymmetric(root.left, root.right);

}

public boolean isSymmetric(TreeNode tree1, TreeNode tree2){

if(tree1==null && tree2==null)

return true;

else if(tree1 == null || tree2 == null || tree1.val != tree2.val)

return false;

else

return isSymmetric(tree1.left, tree2.right) && isSymmetric(tree1.right, tree2.left);

}

}

## 成绩：

1ms，beats21.41%，众数1ms，78.59%

## Cmershen的碎碎念：

这道题很基础，但当时却没有想到。这种题应该予以重视，以免面试时面对如此简单题却无能为力。

# Easy-题目26：198. House Robber

## 题目原文：

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security system connected and it will automatically contact the police if two adjacent houses were broken into on the same night.

Given a list of non-negative integers representing the amount of money of each house, determine the maximum amount of money you can rob tonight without alerting the police.

## 题目大意：

你是一个专业的小偷，计划偷取一条街上的房子。每一个房子内有确定数量的钱，但每两间房子之间连有警报器，如果两个相邻的房子都被偷了，则警报器报警。

用一个数组代表每间房子里拥有的钱数，求出小偷在不引发警报的情况下获得的最大钱数。

## 题目分析：

一个很简单的DP问题。设dp[n]为从头开始偷到第n家获得的钱数，则转移方程如下：

dp[n]=max(dp[n-1],dp[n-2]+money[n])

因为如果不偷第n家，则获得的最大钱数与偷前n-1家获得的钱数相同，如果偷第n家，则第n-1家不能偷，获得的钱数就是前n-2家的最大收益加上第n家里有的钱。

而编码过程中因为dp[n]只与n-1和n-2两项有关系，所以只需要保留两个变量，而不需开一个长度为n的数组。

## 源码：（language：c）

int rob(int\* nums, int numsSize) {

if(numsSize==0)

return 0;

if(numsSize==1)

return nums[0];

else if(numsSize==2)

return nums[0]>nums[1]?nums[0]:nums[1];

else

{

int i;

int dp1,dp2,dp;

dp1=nums[0];

dp2=nums[0]>nums[1]?nums[0]:nums[1];

for(i=2;i<numsSize;i++)

{

dp=dp2>dp1+nums[i]?dp2:dp1+nums[i];

dp1=dp2;

dp2=dp;

}

return dp;

}

}

## 成绩：

0ms，beats3.60%，众数0ms，96.40%

## Cmershen的碎碎念：

本题应该是从第一题以来第一道用dp解决的题目。以后遇到dp的题要多注意，多考虑dp数组构造的方法以及转移方程的确立。

# Easy-题目27：107.Binary Tree Level Order Traversal II

## 题目原文：

Given a binary tree, return the bottom-up level order traversal of its nodes' values. (ie, from left to right, level by level from leaf to root).

## 题目大意：

给出一个二叉树，求层次遍历结果。其中，层次靠后的先输出。

## 题目分析：

与普通的层次遍历不同，本题要求输出为二维数组，其中第一层维度代表层数。因此使用了两个队列，分别记录节点和层数。每次层数发生变化时，将当前数组（第二层数组）加入第一层。

## 源码：（language：java）

public class Solution {

public List<List<Integer>> levelOrderBottom(TreeNode root) {

Queue<TreeNode> queue=new LinkedList<TreeNode>();

Queue<Integer> levelqueue=new LinkedList<Integer>();

queue.add(root);

levelqueue.add(1);

List<List<Integer>> result=new ArrayList();

List<Integer> temp=new ArrayList<Integer>();

if(root==null)

return result;

while(!queue.isEmpty()) {

TreeNode current=queue.remove();

int curLevel=levelqueue.remove();

if(curLevel==result.size()) {

temp.add(current.val);

}

else {

result.add(temp);

temp=new ArrayList<Integer>();

temp.add(current.val);

}

if(current.left!=null) {

queue.add(current.left);

levelqueue.add(curLevel+1);

}

if(current.right!=null) {

queue.add(current.right);

levelqueue.add(curLevel+1);

}

}

result.add(temp);

result.remove(0);

Collections.reverse(result);

return result;

}

}

## 成绩：

4ms,beats 10.19%,众数3ms,46.84%

## Cmershen的碎碎念：

本题成绩较差，代码逻辑也很混乱，应该是因为处理的过程中流程过于复杂，且最后一步将数组翻转比较浪费时间。应考虑使用堆栈和队列两种数据结构存储数组。

# Easy-题目28：27.Remove Element

## 题目原文：

Given an array and a value, remove all instances of that value in place and return the new length.

The order of elements can be changed. It doesn't matter what you leave beyond the new length.

## 题目大意：

给一个数组，删除指定元素，并且返回剩下的数组长度。原数组的顺序可以改变。新长度以后的数据是无关紧要的。

## 题目分析：

使用Java STL中的ArrayList存储数组，遇到指定元素则跳过一次，再遍历ArrayList，把删除后的数组存回原数组，返回ArrayList的长度。

## 源码：（language：java）

public class Solution {

public int removeElement(int[] nums, int val) {

ArrayList<Integer> list=new ArrayList<Integer>();

for(int i=0;i<nums.length;i++) {

if(nums[i]!=val)

list.add(nums[i]);

}

for(int i=0;i<list.size();i++)

nums[i]=list.get(i);

return list.size();

}

}

## 成绩：

2ms,beats 0.74%,众数1ms,67.19%

## cmershen的碎碎念：

这道题也得到了一个很差的成绩，原因是使用了开销比较大的STL，且重复了两次写数组。可以通过two pointer的思想改进一下，一个指针指向待存位置，一个指针指向正在扫描的位置，这样仅需一次读写数组。此外，该题的tag中提示了two pointer，以后在做题时tag的信息应予以注意，因为是很好的提示。

# Easy-题目29：26. Remove Duplicates from Sorted Array

## 题目原文：

Given a sorted array, remove the duplicates in place such that each element appear only once and return the new length.

Do not allocate extra space for another array, you must do this in place with constant memory.

For example,

Given input array nums = [1,1,2],

Your function should return length = 2, with the first two elements of nums being 1 and 2 respectively. It doesn't matter what you leave beyond the new length.

## 题目大意：

给一个按序排列好的数组，在原位上删除相同元素，只留下一个，并返回新的长度。

不要使用新空间。

例如，原数组为[1,1,2]则返回2，并把原数组修改为[1,2].在新长度以后的数字是不用考虑的。

## 题目分析：

吸取上题经验之后，使用双指针，其中“指针”i代表当前扫描的位置，“指针”count指向要插入的位置。每次比较i和i-1下标对应元素是否相同，若相同则i是冗余的，直接跳过，否则count指针对应元素记录下来当前数值，并右移count。i扫完后count就代表不重复数字的个数。

## 源码：（language：java）

public class Solution {

public int removeDuplicates(int[] A) {

int len = A.length;

if (len == 0)

return 0;

int count = 1;

for (int i = 1; i < len; i++) {

if (A[i] != A[i - 1])

A[count++] = A[i];

}

return count;

}

}

## 成绩：

1ms,beats 54.30%,众数2ms,46.31%

## cmershen的碎碎念：

本题的i“指针”是与i-1作比较，因此i从1开始遍历以防止数组越界。

严格而言，i和count是int类型，而不是指针（int\*类型），且java中没有指针。但i和count起到了类似于C语言指针的作用。

# Easy-题目30：66. Plus One

## 题目原文：

Given a non-negative number represented as an array of digits, plus one to the number.

The digits are stored such that the most significant digit is at the head of the list.

## 题目大意：

对用各数位的数组表示一个非负数做+1运算。数组的头部代表数字的高位。

（例如，数字123用数组[1,2,3]表示，加1运算后返回数组[1,2,4]）。

## 题目分析：

分两种情况考虑。

1. 这个数字全是9，这是一种最特殊的情况，因为新数组的长度跟原来数组不一样。
2. 其他情况，则仿照竖式加法，从最后一位开始，寻找第一个不是9的数位（一定存在，否则在情况1中已经解决），这一位加1，后面的数位全部置为0.

## 源码：（language：java）

public class Solution {

public int[] plusOne(int[] digits) {

int len=digits.length;

if(len==0)

return null;

boolean all9=true;

for(int i=0;i<len;i++) {

if(digits[i]!=9) {

all9=false;

break;

}

}

if(all9) { //this num has the form of 999......... , which will be 1000.... after added 1.

int[] result=new int[len+1];

result[0]=1;

return result;

}

else {

if(digits[len-1]<9)

digits[len-1]++;

else {

digits[len-1]=0;

int i=len-2;

while(digits[i]==9)

digits[i--]=0;

digits[i]++;

}

return digits;

}

}

}

## 成绩：

1ms,beats 4.41%，众数：0ms,58.86%

## cmershen的碎碎念：

本题巧用了Java的变量初始化，对整形及整形数组的元素，未赋值的均初始化为0.（不像c中还有什么UVF……）

# Easy-题目31：118. Pascal's Triangle

## 题目原文；

Given numRows, generate the first numRows of Pascal's triangle.

## 题目大意：

输入行数n，输出Pascal三角形（杨辉三角）的前n行。

## 题目分析：

如果逐一用组合数公式求二项式系数，肯定会超时。因此利用组合数的性质公式从上一行递推出本行的系数。（这个公式你不会？？回去复习高中数学10000遍……）

## 源码：（language：java）

public class Solution {

public List<List<Integer>> generate(int numRows) {

List<List<Integer>> triangle=new ArrayList<List<Integer>>();

for(int i=0;i<numRows;i++) {

List<Integer> line=new ArrayList<Integer>();

for(int j=0;j<=i;j++)

line.add((j==0||j==i)?1:triangle.get(i-1).get(j-1)+triangle.get(i-1).get(j));

triangle.add(line);

}

return triangle;

}

}

## 成绩：

1ms,35.14%,众数1ms,64.86%.

## cmershen的碎碎念：

使用公式递推的时候，仍要注意数组越界的问题。

# Easy-题目32：172. Factorial Trailing Zeroes

## 题目原文：

Given an integer *n*, return the number of trailing zeroes in *n*!.

## 题目大意：

给出正整数n，求n!末尾有几个0.

## 题目分析：

trivial的做法是把n！求出来，然后除10直到余数不为0，但开销太大（使用BigInteger）

那么分析一下：

（1）0的来源是2\*5=10，那么将n！因式分解，有几个2\*5就有几个0；

（2）只有末尾是5和0的数里面才有因子5，所以无论如何，n！中因子5都比因子2多，因此数n!里面有几个因子5即可。

（3）接下来比较难以理解，使用分治算法逐步降低问题规模，其递推公式为：

令f(x)表示正整数x中因子5的个数

当0 < n < 5时，f(n!) = 0;

当n >= 5时，f(n!) = k + f(k!), 其中 k = n / 5（取整）。

## 源码：（language：c）

int trailingZeroes(int n) {

int ret = 0;

while(n) {

ret += n/5;

n /= 5;

}

return ret;

}

## 成绩：

0ms,beats 62.92%,众数4ms,59.55%.

## cmershen的碎碎念::

代码实现仅有6行，但其中的数学背景却不是很好理解。该递推关系的数学推导见<http://blog.csdn.net/niushuai666/article/details/6695790>

# Easy-题目33：102. Binary Tree Level Order Traversal

## 题目原文：

Given a binary tree, return the *level order* traversal of its nodes' values. (ie, from left to right, level by level).

## 题目大意：

按层次遍历二叉树。

## 题目分析：

与前面的[Easy第27题](#_Easy-题目27：107.Binary_Tree_Level)类似，只是不用翻转最后的结果数组。

## 源码：（language：java）

public class Solution {

public List<List<Integer>> levelOrder(TreeNode root) {

Queue<TreeNode> queue=new LinkedList<TreeNode>();

Queue<Integer> levelqueue=new LinkedList<Integer>();

queue.add(root);

levelqueue.add(1);

List<List<Integer>> result=new ArrayList();

List<Integer> temp=new ArrayList<Integer>();

if(root==null)

return result;

while(!queue.isEmpty()) {

TreeNode current=queue.remove();

int curLevel=levelqueue.remove();

if(curLevel==result.size())

temp.add(current.val);

else {

result.add(temp);

temp=new ArrayList<Integer>();

temp.add(current.val);

}

if(current.left!=null) {

queue.add(current.left);

levelqueue.add(curLevel+1);

}

if(current.right!=null) {

queue.add(current.right);

levelqueue.add(curLevel+1);

}

}

result.add(temp);

result.remove(0);

return result;

}

}

## 成绩：

3ms,beats 14.37%,众数3ms,46.37%

# Easy-题目34：119. Pascal's Triangle II

## 题目原文：

Given an index *k*, return the *k*th row of the Pascal's triangle.

## 题目大意：

给出整数k，输出杨辉三角的第k行。

## 题目分析：

在[Easy第31题](#_Easy-题目31：118._Pascal's_Triangle)上修改，求出前n行，但只返回最后一行。

## 源码：（language：java）

public class Solution {

public List<Integer> getRow(int rowIndex) {

List<List<Integer>> triangle=new ArrayList();

for(int i=0;i<=rowIndex;i++)

{

List<Integer> line=new ArrayList<Integer>();

for(int j=0;j<=i;j++)

line.add((j==0||j==i)?1:triangle.get(i-1).get(j-1)+triangle.get(i-1).get(j));

triangle.add(line);

}

return triangle.get(rowIndex);

}

}

## 成绩：

4ms,beats 3.88%,众数3ms,51.41%

## cmershen的碎碎念：

本题可能有直接生成不依赖第k-1行的算法，如果依赖前一行的推导，那么后一行可以覆盖前一行，这样只需(k-1)个额外空间。满足Follow up中 o(n)空间复杂度的要求。

# Easy-题目35：9. Palindrome Number

## 题目原文：

Determine whether an integer is a palindrome. Do this without extra space.

## 题目大意：

判断一个数是不是回文整数，要求不使用额外空间。

## 题目分析：

没想到不使用额外空间的办法，两种程序都是判断原数和翻转数是否相等来判断。一个是在字符串上的判断，一个是在数上的判断。

## 源码：（language：c/python）

方法一：

int isPalindrome(int x) {

int n=0,x2=x;

if(x<0)

return false;

while(x) {

n=n\*10+x%10;

x/=10;

}

return n==x2;

}

方法二：

class Solution(object):

def isPalindrome(self, x):

return str(x)==str(x)[::-1]

## 成绩：

c程序：56ms,beats 94.01%,众数60ms,18.43%

python程序：268ms,beats 70.58%,众数288ms,5.55%

# Easy-题目36：112. Path Sum

## 题目原文：

Given a binary tree and a sum, determine if the tree has a root-to-leaf path such that adding up all the values along the path equals the given sum.

For example:

Given the below binary tree and sum = 22,

5

/ \

4 8

/ / \

11 13 4

/ \ \

7 2 1

return true, as there exist a root-to-leaf path 5->4->11->2 which sum is 22.

## 题目大意：

给出一个二叉树和一个和值，判断是否存在一条从根节点到叶节点路径，使得路径上所有值之和等于给定的和值。

## 题目分析：

1. 若该节点为空，则不存在；
2. 若该节点为叶子节点，则判断节点值是否等于当前和值；
3. 若不是叶子节点，则向左右子树分别递归搜索下去。

## 源码：（language：c）

bool hasPathSum(struct TreeNode\* root, int sum) {

if(!root)

return 0;

else if (!root->left && !root->right)

return root->val==sum;

else

return hasPathSum(root->left,sum-root->val) || hasPathSum(root->right,sum-root->val);

}

## 成绩：

4ms,beats 73%,众数4ms,27%

# Easy-题目37：111. Minimum Depth of Binary Tree

## 题目原文：

Given a binary tree, find its minimum depth.

The minimum depth is the number of nodes along the shortest path from the root node down to the nearest leaf node.

## 题目大意：

给一个二叉树，求最小深度。

最小深度指的是沿着最短的路径从根节点走到最近的叶子节点经过的节点数。

## 题目分析：

1. 空树最小深度是0；
2. 一边子树为空的情况下，最小深度为1+非空子树最小深度；
3. 两边均有子树的情况下，最小深度为1+左右子树最小深度的最小值。

据此很容易写出递归算法。

## 源码：（language：c）

int minDepth(struct TreeNode\* root) {

if(!root)

return 0;

else if(!root->right)

return 1+minDepth(root->left);

else if(!root->left)

return 1+minDepth(root->right);

else

return 1+min(minDepth(root->left),minDepth(root->right));

}

int min(int a,int b) {

return a<b?a:b;

}

## 成绩：

4ms,beats 37.3%,众数4ms,62.7%.

## cmershen的碎碎念：

第二和第三个if不能省略，否则对一边子树为空的情况则返回1.（根节点到空树）。

# Easy-题目38：160. Intersection of Two Linked Lists

## 题目原文：

Write a program to find the node at which the intersection of two singly linked lists begins.

For example, the following two linked lists:

A: a1 → a2

↘

c1 → c2 → c3

↗

B: b1 → b2 → b3

begin to intersect at node c1.

## 题目大意：

求两个单链表的交点。

## 题目分析：

考研数据结构复习的时候见过此题，应该是链表处理的题目中比较经典的一道。解法是求出两个链表的长度，然后两个指针分别指向两个链表的头结点，先让较长的链表的指针走过差值的步数，这样两个指针距离尾节点的距离就相同了，让两个节点并行走直到遇到交点。

## 源码：（language：c）

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* struct ListNode \*next;

\* };

\*/

struct ListNode \*getIntersectionNode(struct ListNode \*headA, struct ListNode \*headB) {

if(!headA||!headB)

return NULL;

struct ListNode \*p;

int lengtha,lengthb;

for(p=headA,lengtha=0;p;p=p->next,lengtha++);

for(p=headB,lengthb=0;p;p=p->next,lengthb++);

if(lengtha>lengthb) {

for(int i=0;i<lengtha-lengthb;i++)

headA=headA->next;

}

else if (lengthb>lengtha) {

for(int i=0;i<lengthb-lengtha;i++)

headB=headB->next;

}

while(headA!=headB) {

headA=headA->next;

headB=headB->next;

}

return headA;

}

## 成绩：

32ms，beats 18.75%，众数32ms,78.91%

## cmershen的碎碎念：

这道题比较经典，但如果不给出这个算法，自己是很难想到的。

# Easy-题目39：88. Merge Sorted Array

## 题目原文：

Given two sorted integer arrays *nums1* and *nums2*, merge *nums2* into *nums1* as one sorted array.

## 题目大意：

给出两个排序好的数组nums1和nums2，把nums2合并到nums1里面成为一个有序的数组。

## 题目分析：

从后往前思考，同时从最后面往前遍历两个数组，每次把nums1和nums2数组中比较大的值依次放入nums1数组的最后面（用一个从n+m-1开始的下标记录），直到nums1或nums2数组有一个扫完，这时把另一个数组的所有数字都从后往前装到nums1里面。

## 源码：（language：java）

public class Solution {

public void merge(int[] nums1, int m, int[] nums2, int n) {

int len = m + n;//排序后总长度

while(m > 0 && n > 0){

if(nums1[m-1] > nums2[n-1]){

nums1[--len] = nums1[--m];

}else{

nums1[--len] = nums2[--n];

}

}

//余下的数字

while(m > 0){

nums1[--len] = nums1[--m];

}

while(n > 0){

nums1[--len] = nums2[--n];

}

}

}

## 成绩：

0ms,beats 46.18%,众数0ms,53.82%

## Cmershen的碎碎念：

有时候一个问题从前往后考虑比较难的时候，可以考虑从后向前思考。

# Easy-题目40：36. Valid Sudoku

## 题目原文：

Determine if a Sudoku is valid, according to: [Sudoku Puzzles - The Rules](http://sudoku.com.au/TheRules.aspx).

The Sudoku board could be partially filled, where empty cells are filled with the character '.'.

**Note:**  
A valid Sudoku board (partially filled) is not necessarily solvable. Only the filled cells need to be validated.

## 题目大意：

判断一个数独是不是合理的。

注意：一个部分填好的合理数独不一定可解，但只需要验证是否合理即可。（@15-信安-L 之前我们争论此题要不要求解数独，那么读完题意你明白了么？）

## 题目分析：

利用HashSet的性质，分别判断每个行、列、九宫格中有没有重复元素。很trivial的解法。

## 源码：（language：java）

public class Solution {

public boolean isValidSudoku(char[][] board) {

HashSet<Character> set=new HashSet<Character>();

for(int i=0;i<9;i++) {

for(int j=0;j<9;j++) {

if(board[i][j]>='0' && board[i][j]<='9') {

if(!set.add(board[i][j]))

return false;

}

}

set.clear();

}

for(int j=0;j<9;j++) {

for(int i=0;i<9;i++) {

if(board[i][j]>='0' && board[i][j]<='9') {

if(!set.add(board[i][j]))

return false;

}

}

set.clear();

}

for(int i=0;i<3;i++) {

for(int j=0;j<3;j++) {

for(int s=0;s<3;s++) {

for(int t=0;t<3;t++) {

if(board[i\*3+s][j\*3+t]>='0' && board[i\*3+s][j\*3+t]<='9') {

if(!set.add(board[i\*3+s][j\*3+t]))

return false;

}

}

}

set.clear();

}

}

return true;

}

}

## 成绩：

7ms,beats 35.54%,众数4ms,23.99%

## Cmershen的碎碎念：

本题只根据定义判断，应该是用了最为trivial的解法，因此成绩不好。

# Easy-题目41：219. Contains Duplicate II

## 题目原文：

Given an array of integers and an integer *k*, find out whether there are two distinct indices *i* and *j* in the array such that **nums[i] = nums[j]** and the difference between *i* and *j*is at most *k*.

## 题目大意：

给一个数组nums和整数K，问是否存在两个下标i和j，使得nums[i]=nums[j]，并且|i-j|≤k。

## 题目分析：

使用HashMap记录一组key-value对，其中key为数组中元素的值，value为所在下标。

当key不重复时加入HashMap即可，如果发现了重复的key，则判断当前数组下标和value值（分别对应题干中的i和j）绝对值是否≤k。

## 源码：（language：java）

public class Solution {

public boolean containsNearbyDuplicate(int[] nums, int k) {

HashMap<Integer,Integer> map=new HashMap<Integer,Integer>();

for(int i=0;i<nums.length;i++) {

if(map.containsKey(nums[i]) && Math.abs(i-map.get(nums[i]))<=k)

return true;

else

map.put(nums[i],i);

}

return false;

}

}

## 成绩：

16ms，beats 5.31%，众数14ms，30.23%

## cmershen的碎碎念：

使用了trivial的解法因此成绩低于平均值（但不算太差），可能是HashMap浪费了开销，discuss中似乎有基于快排的解法，有待日后研读。

# Easy-题目42：223. Rectangle Area

## 题目原文：

Find the total area covered by two rectilinear rectangles in a 2D plane.

Each rectangle is defined by its bottom left corner and top right corner as shown in the figure.



Assume that the total area is never beyond the maximum possible value of int.

## 题目大意：

给出(A,B),(C,D),(E,F),(G,H)四个点，如图所示求围成图形的面积。

## 题目分析：

在纸上多画几种情况可以发现，A>=G、B>=H、C<=E、D<=F这四种情况下，两个矩形是不重叠的，因此直接计算面积之和，否则要减去重叠部分。

## 源码：（language：cpp）

class Solution {

public:

int computeArea(int A, int B, int C, int D, int E, int F, int G, int H) {

int area = (C-A)\*(D-B) + (G-E)\*(H-F);

if (A >= G || B >= H || C <= E || D <= F)

return area;

int top = min(D, H);

int bottom = max(B, F);

int left = max(A, E);

int right = min(C, G);

return area - (top-bottom)\*(right-left);

}

};

## 成绩：

36ms,beats 20.05%，众数36ms,37.38%.

# Easy-题目43：190. Reverse Bits

## 题目原文：

Reverse bits of a given 32 bits unsigned integer.

For example, given input 43261596 (represented in binary as 00000010100101000001111010011100), return 964176192 (represented in binary as 00111001011110000010100101000000).

## 题目大意：

对一个32位的无符号整数，翻转它的比特位。

## 题目分析：

用一个新数t从0开始记录n的最后一位，然后令n右移，t左移。

## 源码：（language：c）

uint32\_t reverseBits(uint32\_t n) {

int t=0;

for(int i=0;i<32;i++) {

t=t\*2+n%2;

n/=2;

}

return t;

}

## 成绩：

4ms,beats 10.63%,众数4ms,89.37%

# Easy-题目44：19. Remove Nth Node From End of List

## 题目原文：

Given a linked list, remove the nth node from the end of list and return its head.

For example,

Given linked list: 1->2->3->4->5, and n = 2.

After removing the second node from the end, the linked list becomes 1->2->3->5.

## 题目大意：

给出一个单链表，删除倒数第n个节点。

## 题目分析：

用两个指针，第一个指针p1指向头结点，然后第二个节点p2指向p1之后第n个节点（向右滑动n次）。然后并行向右滑动两个指针，直到p2为最后一个节点（p2->next==NULL），此时p1指向待删除节点的前一个节点，令p1=p1->next即可。

## 源码：（language：c）

struct ListNode\* removeNthFromEnd(struct ListNode\* head, int n) {

if (!head)

return NULL;

struct ListNode \*p1 = head;

struct ListNode \*p2 = head;

for(int i = 0; i < n; i++)

p2 = p2->next;

if(!p2)

return head->next;

while(p2->next) {

p1 = p1->next;

p2 = p2->next;

}

p1->next=p1->next->next;

return head;

}

## 成绩：

0ms,beats 19.70%,众数0ms,80.30%

## Cmershen的碎碎念：

本题有一个需要讨论的地方，就是在p2向右滑动n步的时候，有可能滑到最后一个节点的后面（即null），此时说明要删除的是头节点，直接返回head->next即可。

# Easy-题目45：58. Length of Last Word

## 题目原文：

Given a string *s* consists of upper/lower-case alphabets and empty space characters ' ', return the length of last word in the string.

If the last word does not exist, return 0.

## 题目大意：

给出一个字符串s，求空格分隔的最后一个单词的长度

## 题目分析：

使用Java的split这个api，按空格分割成字符串数组，返回最后一个字符串的长度。

## 源码：（language：java）

public class Solution {

public int lengthOfLastWord(String s) {

String[] words= s.split(" ");

if(words.length==0)

return 0;

return words[words.length-1].length();

}

}

## 成绩：

3ms，beats 12.92%，众数0ms，47.42%

## cmershen的碎碎念：

这道题还可以用trim修剪字符串两端的空格，再从后向前搜索，这样的时间应该会短一些，因为split的底层是依据正则表达式实现的，开销很大。

# Easy-题目46：205. Isomorphic Strings

## 题目原文：

Given two strings s and t, determine if they are isomorphic.

Two strings are isomorphic if the characters in s can be replaced to get t.

All occurrences of a character must be replaced with another character while preserving the order of characters. No two characters may map to the same character but a character may map to itself.

For example,

Given "egg", "add", return true.

Given "foo", "bar", return false.

Given "paper", "title", return true.

Note:

You may assume both s and t have the same length..

## 题目大意：

给出字符串s和t，判断他们是不是同构(isomorphic)的。

同构的定义是s和t具有相同的结构，即可用t串一个字符代替s串的一个字符，而其他字符不变。

例如，”egg”和 ”add”是同构的；”foo”和”bar”不是同构的;”paper”和”title ”是同构的。

保证s串和t串的长度相等。

## 题目分析：

使用一个HashMap记录一组key-value对，key为s串中每个字符,value为t串中每个字符。如果出现相同的key则不是同构的。

例如s=”foo”，t=”bar”,则存在o-b和o-r两个同key的映射。

但这样还不够，还要再按相同方法建立t串到s串的map，因为若s="bar",t=”foo”则不存在s串到t串中的相同key，但存在t串到s串的相同key。

## 源码：（language：java）

public class Solution {

public boolean isIsomorphic(String s, String t) {

int slen=s.length();

int tlen=t.length();

if(slen!=tlen)

return false;

else

{

HashMap<Character,Character> map=new HashMap();

for(int i=0;i<slen;i++)

{

if(!map.containsKey(s.charAt(i)))

map.put(s.charAt(i),t.charAt(i));

else

{

if(map.get(s.charAt(i)) != t.charAt(i))

return false;

}

}

map.clear();

for(int i=0;i<slen;i++)

{

if(!map.containsKey(t.charAt(i)))

map.put(t.charAt(i),s.charAt(i));

else

{

if(map.get(t.charAt(i)) != s.charAt(i))

return false;

}

}

return true;

}

}

}

## 成绩：

34ms,beats 30.09%,众数26ms，8.93%

# Easy-题目47：20. Valid Parentheses

## 题目原文：

Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

The brackets must close in the correct order, "()" and "()[]{}" are all valid but "(]" and "([)]" are not.

## 题目大意：

给出一个括号字符串，判断括号是否正确匹配。

## 题目分析：

本题考查堆栈的使用，遇到左边的括号全部入栈，遇到右边的括号则看栈顶的括号是不是对应的左括号，如是则弹出，如不是则返回错误。如果整个括号字符串扫完之后栈恰好为空，则是合法匹配。

## 源码：（language：python）

class Solution(object):

def isValid(self, s):

"""

:type s: str

:rtype: bool

"""

stack=[]

for c in s:

if c in ['(','[','{']:

stack.append(c)

else:

if(len(stack)==0):

return False

top=stack[len(stack)-1]

if ((c == ')' and top=='(') or (c == ']' and top=='[') or (c == '}' and top == '{')):

stack.pop()

else:

return False

return len(stack)==0

## 成绩：

43ms,beats 55.57%,众数40ms,27.91%

# Easy-题目48：299. Bulls and Cows

## 题目原文：

You are playing the following Bulls and Cows game with your friend: You write down a number and ask your friend to guess what the number is. Each time your friend makes a guess, you provide a hint that indicates how many digits in said guess match your secret number exactly in both digit and position (called "bulls") and how many digits match the secret number but locate in the wrong position (called "cows"). Your friend will use successive guesses and hints to eventually derive the secret number.

For example:

Secret number: "1807"

Friend's guess: "7810"

Hint: 1 bull and 3 cows. (The bull is 8, the cows are 0, 1 and 7.)

Write a function to return a hint according to the secret number and friend's guess, use A to indicate the bulls and B to indicate the cows. In the above example, your function should return "1A3B".

Please note that both secret number and friend's guess may contain duplicate digits, for example:

Secret number: "1123"

Friend's guess: "0111"

In this case, the 1st 1 in friend's guess is a bull, the 2nd or 3rd 1 is a cow, and your function should return "1A1B".

You may assume that the secret number and your friend's guess only contain digits, and their lengths are always equal.

## 题目大意：

你在和朋友玩Bulls and Cows游戏：你写出一串数字，然后你的朋友猜这个数字，如果数位和数字完全对应，则称为bulls（用A表示），如果数字正确而数位错误，则称为cows（用B表示）。

输入你的答案和朋友的猜测，判断有几个bulls和几个cows（用xAyB）表示。

例如答案=1807，猜测=7810，则数位8正确，1，0,7数字正确但位置错误，所以输出1A3B。

## 题目分析：

使用scount和gcount两个二位数组统计各数字所在的下标，s和g两个数组记录每个数字出现了多少次，然后从0到9全部遍历，先看s[i]和g[i]，即数字i在两个串中分别出现了多少次，取较小值，先全部加到B上，再扫scount[i]和gcount[i]数组去比较i出现的数位，如果找到了相同数位，则b递减a递增，直到扫完10个数字。时间复杂度和空间复杂度都是O（n）。

## 源码：（language：java）

public class Solution {

public String getHint(String secret, String guess) {

int slen=secret.length();

boolean[][] scount=new boolean[10][slen];

boolean[][] gcount=new boolean[10][slen];

int[] s=new int[10];

int[] g=new int[10];

int a = 0,b = 0;

for(int i=0;i<slen;i++) {

scount[secret.charAt(i)-'0'][i]=true;

s[secret.charAt(i)-'0']++;

gcount[guess.charAt(i)-'0'][i]=true;

g[guess.charAt(i)-'0']++;

}

for(int i=0;i<10;i++) {

int binc=Math.min(s[i], g[i]);

if(binc==0)

continue;

b+=binc;

for(int j=0;j<slen;j++) {

if(scount[i][j] && gcount[i][j]) {

b--;

a++;

}

}

}

return new String(a+"A"+b+"B");

}

}

## 成绩：

8ms,beats 39.5%,众数4ms,21.65%

## cmershen的碎碎念：

这个游戏小时候在电子词典上都玩过，当时做的限定比较多，要求只能是4位数，数字不重复，且当年最大的吐槽是为何只能猜八次。希望以后可以设计一个ai算法解决任意4位猜数字问题。

# Easy-题目49：290. Word Pattern

## 题目原文：

Given a pattern and a string str, find if str follows the same pattern.

Here follow means a full match, such that there is a bijection between a letter in pattern and a non-empty word in str.

Examples:

pattern = "abba", str = "dog cat cat dog" should return true.

pattern = "abba", str = "dog cat cat fish" should return false.

pattern = "aaaa", str = "dog cat cat dog" should return false.

pattern = "abba", str = "dog dog dog dog" should return false.

## 题目大意：

给出一个模式和字符串。判断字符串时候满足模式。这里模式和对应字符串要求是一一映射(bijection)的。（如果不知道啥叫一一映射，继续滚回去复习高中数学！）

## 题目分析：

这题和[Easy第46题](#_Easy-题目46：125._Valid_Palindrome)类似，使用两个HashMap维护双向的key-value对，一旦发现有重复的key返回false。

## 源码：（language：java）

public class Solution {

public boolean wordPattern(String pattern, String str) {

HashMap<Character,String> map=new HashMap();

HashMap<String,Character> map2=new HashMap();

String[] words=str.split(" ");

if(pattern.length()!=words.length)

return false;

for(int i=0;i<pattern.length();i++) {

Character key=pattern.charAt(i);

String word=words[i];

if(!map.containsKey(key)) {

if(!map2.containsKey(word))

{

map.put(key, word);

map2.put(word, key);

}

else

return false;

}

else if (!(map.get(key).equals(word) && map2.get(word).equals(key)))

return false;

}

return true;

}

}

## 成绩：

3ms，beats 18.29%，众数3ms，51.82%

# Easy-题目50：38. Count and Say

## 题目原文：

The count-and-say sequence is the sequence of integers beginning as follows:

1, 11, 21, 1211, 111221, ...

1 is read off as "one 1" or 11.

11 is read off as "two 1s" or 21.

21 is read off as "one 2, then one 1" or 1211.

Given an integer n, generate the nth sequence.

## 题目大意：

按规律填空：1,11,21,1211,111221…….

规律是这样的：

1读作“1个1”并写为11；

11读作“2个1”并写为21；

21读作“1个2,1个1”并写为1211

……

求出这个数列的第n项

## 题目分析：

首先求出第n-1项，然后从头扫第n-1项的字符串，依次统计连续相同字符的个数。

## 源码：(language：java)

public class Solution {

public String countAndSay(int n) {

if(n==1)

return "1";

else

{

String str=countAndSay(n-1);

int i=0;

String answer="";

while(i<str.length())

{

int j=i;

while(j<str.length() && str.charAt(i)==str.charAt(j))

j++;

if(j>i)

{

answer+=Integer.toString(j-i);

answer+=str.charAt(i);

i=j;

}

else

i++;

}

return answer;

}

}

}

## 成绩：

36ms,beats 3.03%,众数5ms,16.44%.

## cmershen的碎碎念：

本题成绩较差，是因为使用了递归？不过如果不用递归难道不从n-1项推出n项吗？

本题的数列可在OEIS的[A005150](http://oeis.org/A005150)查到。

# Easy-题目51：203. Remove Linked List Elements

## 题目原文：

Remove all elements from a linked list of integers that have value val.

Example

Given: 1 --> 2 --> 6 --> 3 --> 4 --> 5 --> 6, val = 6

Return: 1 --> 2 --> 3 --> 4 --> 5

## 题目大意：

删除单链表中指定节点。

## 题目分析：

用两个指针p和q（p始终是q前驱节点）并行向后推，若找到q是待删除节点(q->val==val)则删除q，但还没有处理头结点，故看一下head->val与val是否相等。

## 源码：（language：cpp）

class Solution {

public:

ListNode\* removeElements(ListNode\* head, int val) {

ListNode \*p=head;

if(!p || (!p->next && p->val==val))//p is null or (length of linklist=1 and this unique node is going to be deleted)

return NULL;

else {

ListNode \*q=head->next;

while(q) {

if(q->val==val) {// the first node is going to be deleted

p->next=p->next->next;

q=q->next;

continue;

}

else {

p=p->next;

q=q->next;

}

}

if(head->val!=val)

return head;

else

return head->next;

}

}

};

## 成绩：

32ms,beats 54.36%,众数36ms,45.72%.

# Easy-题目52：14. Longest Common Prefix

## 题目原文：

Write a function to find the longest common prefix string amongst an array of strings.

## 题目大意：

写一个函数，求出一个字符串数组中所有字符的最长公共前缀。

## 题目分析:

没什么难的，贪心的一个一个比，直到其中一个字符串比完，或者遇到了不同字符为止。

## 源码：（language：java）

public class Solution {

public String longestCommonPrefix(String[] strs) {

if(strs.length==0)

return "";

else if(strs.length==1)

return strs[0];

else

{

int sublen=0;

while(true)

{

if(strs[0].length()==0)

return "";

if(sublen==strs[0].length())

return strs[0];

char prefix=strs[0].charAt(sublen);

for(String str:strs)

{

if(str.length()==sublen || str.charAt(sublen)!=prefix)

return str.substring(0,sublen);

}

sublen++;

}

}

}

}

## 成绩：

3ms,beats 40.93%,众数3ms,23.44%

# Easy-题目53：257. Binary Tree Paths

## 题目原文：

Given a binary tree, return all root-to-leaf paths.

For example, given the following binary tree:

1

/ \

2 3

\

5

All root-to-leaf paths are:

["1->2->5", "1->3"]

## 题目大意：

给出一个二叉树，输出所有从根节点到叶子节点的路径。

## 题目分析：

遍历即可，注意每次向左右子树搜索的时候，把当前路径一直带在函数参数里面。

## 源码：（language：java）

public class Solution {

List<String> res = new ArrayList<String>();

public List<String> binaryTreePaths(TreeNode root) {

if(root != null) findPaths(root,String.valueOf(root.val));

return res;

}

private void findPaths(TreeNode n, String path){

if(n.left == null && n.right == null) res.add(path);

if(n.left != null) findPaths(n.left, path+"->"+n.left.val);

if(n.right != null) findPaths(n.right, path+"->"+n.right.val);

}

}

## 成绩：

3ms,beats 27.22%,众数3ms,47.28%.

# Easy-题目54：234. Palindrome Linked List

## 题目原文：

Given a singly linked list, determine if it is a palindrome.

**Follow up:**  
Could you do it in O(n) time and O(1) space?

## 题目大意：

给一个单链表，判断它是不是回文的。

你能实现O(n)时间复杂度和O(1)空间复杂度吗？

## 题目分析：

如果用O(1)空间复杂度的话，可以求出链表的中点，然后翻转前半个链表再比较两个“半链表”是否相同。但是，我觉得代码太麻烦，就改用O(n)的空间复杂度了，这样的算法很简单：把链表转换成线性表，因为是随机存储的很容易判断是否是回文。

## 源码：（language：cpp）

class Solution {

public:

bool isPalindrome(ListNode\* head) {

int length=0;

for(ListNode\* p=head;p;p=p->next,length++);//length is the length of linklist

int\* array=new int[length];

int i=0;

for(ListNode\* p=head;p;p=p->next)

array[i++]=p->val;

return isPalindromeArray(array,length);

}

bool isPalindromeArray(int\* array,int length)

{

if(length==0||length==1)

return true;

else if(\*array!=\*(array+length-1))

return false;

else

return isPalindromeArray(array+1,length-2);

}

};

## 成绩：

28ms,beats 9.66%,众数28ms，58.57%

# Easy-题目55：67. Add Binary

## 题目原文：

Given two binary strings, return their sum (also a binary string).

## 题目大意：

给出两个二进制数的字符串，求出它们的和，仍用字符串表示。

## 题目分析：

调用python中的api :int(str,n),按n进制解析字符串str。最后会输出字符串0b….，因此再截取一下从第2位开始的子串即可。

## 源码：（language：python）

class Solution(object):

def addBinary(self, a, b):

"""

:type a: str

:type b: str

:rtype: str

"""

return str(bin(int(a,2)+int(b,2)))[2:]

## 成绩：

64ms,beats 44.73%,众数60ms,14.36%

# Easy-题目56：303. Range Sum Query – Immutable

## 题目原文：

Given an integer array *nums*, find the sum of the elements between indices *i* and *j* (*i* ≤ *j*), inclusive.

Note:

You may assume that the array does not change.

There are many calls to sumRange function.

## 题目大意：

给出一个数组，计算从下标i到下标j的所有元素之和。

注意：

数组是不会改变的，且sumRange函数会多次调用。

## 题目分析：

构造一个类，里面有一个辅助数组Nums[0…n],其中nums[i]记录0-n的总和，这样只需在构造函数里累加一次各元素之和，以后每次调用sumRange方法的时候只需求一次减法即可。

## 源码：（language：java）

public class NumArray {

private int[] nums;

public NumArray(int[] nums) {

this.nums=new int[nums.length+1];

this.nums[0]=0;

for(int i=0;i<nums.length;i++)

{

this.nums[i+1]=this.nums[i]+nums[i];

}

}

public int sumRange(int i, int j) {

if(nums.length==1)

return 0;

else

return nums[j+1]-nums[i];

}

}

## 成绩：

3ms，beats 26.34%，众数3ms，73.62%

## cmershen的碎碎念：

本题似乎在实际开发中很有价值，这样的问题应予以重视。

# Easy-题目57：28. Implement strStr()

## 题目原文：

Implement strStr().

Returns the index of the first occurrence of needle in haystack, or -1 if needle is not part of haystack.

## 题目大意：

实现strStr()函数。返回needle字符串在haystack字符串中首次出现的下标，如果needle不是haystack的子串则返回-1.

## 题目分析：

c99中strstr函数的接口不对，因此使用Java中的indexOf()方法水过。

## 源码：（language：java）

public class Solution {

public int strStr(String haystack, String needle) {

return haystack.indexOf(needle);

}

}

## 成绩：

1ms,beats 69.31%,众数1ms,29.29%.

## cmershen的碎碎念：

附C99中strstr()函数源码：

char \*strstr(constchar\*s1,constchar\*s2)

{

int n;

if(\*s2)

{

while(\*s1)

{

for(n=0;\*(s1+n)==\*(s2+n);n++)

{

if(!\*(s2+n+1))

return(char\*)s1;

}

s1++;

}

return NULL;

}

else

return (char\*)s1;

}

但是稍加改动（简单改动了一下return的部分）之后放入OJ居然会提示TLE！也许C99的源码存在问题有待思考。

再附jdk中indexOf的底层实现供研究：

static int indexOf(char[] source, int sourceOffset, int sourceCount,

char[] target, int targetOffset, int targetCount,

int fromIndex) {

if (fromIndex >= sourceCount) {

return (targetCount == 0 ? sourceCount : -1);

}

if (fromIndex < 0) {

fromIndex = 0;

}

if (targetCount == 0) {

return fromIndex;

}

char first = target[targetOffset];

int max = sourceOffset + (sourceCount - targetCount);

for (int i = sourceOffset + fromIndex; i <= max; i++) {

/\* Look for first character. \*/

if (source[i] != first) {

while (++i <= max && source[i] != first);

}

/\* Found first character, now look at the rest of v2 \*/

if (i <= max) {

int j = i + 1;

int end = j + targetCount - 1;

for (int k = targetOffset + 1; j < end && source[j]

== target[k]; j++, k++);

if (j == end) {

/\* Found whole string. \*/

return i - sourceOffset;

}

}

}

return -1;

}

# Easy-题目58：7. Reverse Integer

## 题目原文：

Reverse digits of an integer.

Example1: x = 123, return 321

Example2: x = -123, return -321

## 题目大意：

翻转一个整数。（若溢出返回0）

## 题目分析：

用类似于[Easy第35题](#_Easy-题目35：9._Palindrome_Number)的思路，但此处可能出现溢出，所以用long存储翻转后的数字，如果超过了上限则返回0.

## 源码：（language：c）

int reverse(int x) {

long n=0;

while(x) {

n=n\*10+x%10;

x/=10;

}

return n==(int)n?(int)n:0;

}

## 成绩：

4ms,beats 61.68%，众数5ms,51.40%.

## cmershen的碎碎念：

本题似乎在js和python下会出现bug。（跟溢出有关）

# Easy-题目59：204. Count Primes

## 题目原文：

Count the number of prime numbers less than a non-negative number, ***n***.

## 题目大意：

问＜n的素数有多少个？

## 题目分析：

使用埃拉托色尼筛法。

该算法描述如下：(Translate from wikipaedia.)

1. 列出从2到n的序列；
2. 初始化p为序列第一个数（目前为2，这是废话，2当然是素数了）；
3. 划掉所有p的倍数；
4. 如果序列第一个数的平方>=n，则剩下的数都是素数，否则goto 2.

## 源码：（language：java）

public class Solution {

public int countPrimes(int n) {

if( n <=2)

return 0;

int count = 1;

boolean isNotPrime[] = new boolean[n+1];

for(int i=3;i\*i<=n;i=i+2) {

if(!isNotPrime[i]) {

for(int j= i\*i ;j<=n;j=j+2\*i)

isNotPrime[j] = true;

}

}

for(int i = 3;i<n;i=i+2) {

if(!isNotPrime[i])

count++;

}

return count;

}

}

## 成绩：

11ms,beats 99.20%,众数28ms,6.70%

## cmershen的碎碎念：

可能大多数提交的代码都用的是朴素解法，所以埃拉托色尼筛法可以击败这么大比例的提交代码……（暴力从2-n尝试，时间复杂度为）

附朴素解法(现在提交这个好像会超时)：

public int countPrimes(int n) {

int count = 0;

for (int i = 1; i < n; i++) {

if (isPrime(i)) count++;

}

return count;

}

private boolean isPrime(int num) {

if (num <= 1) return false;

// Loop's ending condition is i \* i <= num instead of i <= sqrt(num)

// to avoid repeatedly calling an expensive function sqrt().

for (int i = 2; i \* i <= num; i++) {

if (num % i == 0) return false;

}

return true;

}

# Easy-题目60：125. Valid Palindrome

## 题目原文：

Given a string, determine if it is a palindrome, considering only alphanumeric characters and ignoring cases.

## 题目大意：

给一个字符串，只考虑字母数字且忽略大小写的情况下，判断是不是回文的。

## 题目分析：

在普通判断回文的算法基础上简单加以改进即可，遇到大写字母转换为小写字母，遇到非字母和数字跳过，遇到两边指针指向的字符不相等直接返回false跳出去。

## 源码：（language：java）

public class Solution {

public boolean isPalindrome(String s) {

int len=s.length();

if(len==0||len==1)

return true;

int i=0,j=len-1;

while(i<j) {

char ch1=s.charAt(i);

char ch2=s.charAt(j);

if(ch1>='A'&&ch1<='Z')

ch1+=32;

if(ch2>='A'&&ch2<='Z')

ch2+=32;

if (!((ch1>='a'&&ch1<='z') || (ch1>='0' && ch1<='9'))) {

i++;

continue;

}

if (!((ch2>='a'&&ch2<='z') || (ch2>='0' && ch2<='9'))) {

j--;

continue;

}

if(ch1!=ch2)

return false;

else {

i++;

j--;

}

}

return true;

}

}

## 成绩：

5ms,beats 97.26%,众数9ms,15.18%

## cmershen的碎碎念

其实本题的思路不难想，但我也想不到为什么会击败这么多提交代码……，可能是计算机中加32比较好操作吧（只需改动1bit）

# Easy-题目61：6. ZigZag Conversion

## 题目原文：

The string "PAYPALISHIRING" is written in a zigzag pattern on a given number of rows like this: (you may want to display this pattern in a fixed font for better legibility)

P A H N

A P L S I I G

Y I R

And then read line by line: "PAHNAPLSIIGYIR"

Write the code that will take a string and make this conversion given a number of rows:

string convert(string text, int nRows);

convert("PAYPALISHIRING", 3) should return "PAHNAPLSIIGYIR".

## 题目大意：

把一个字符串排成n行的锯齿，再按正常顺序（从左到右，从上到下）输出。

## 题目分析：

首先看一下规律：（例如输入字母a-z，nRows=3）,则生成的锯齿如下：



(只能截图了，word 的排版总是乱乱的我也不知道怎么让他整齐起来)

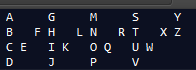
观察每行中字母的下标：（A对应下标0，以此类推）

第一行：0,4,8,12…

第二行：1,3,5,7…

第三行：2,6,10,14…

发现都是等差数列！那么再看一下nRows=4的情况吧！（毕竟这样的规律不是做数学题，不需要用严格的数学归纳法证明，只需大概看出规律即可推广到任意nRows）



观察下标：

第一行：0,6,12,18…

第二行：1,5,7,11,13…

第三行：2,4,8,10…

第四行：3,9,15,21

观察到第二行和第三行不再是等差数列，而是差值分别为2,4交替出现。

由此得出规律：每一行都是两个差值Step1和Step2交替出现，每读完一行Step1递减2，Step2递增2。但有两种特殊情况：一是首末两行差值有一个为0，二是nRows为奇数时最中间一行的Step1=Step2.

结合具体实例来看：

nRows=3时：

第一行Step1=0，Step2=4；

第二行Step1=2，Step2=2；

第三行Step1=4，Step2=0。

nRows=4时：

第一行Step1=0，Step2=6；

第二行Step1=2，Step2=4；

第三行Step1=4，Step2=2；

第四行Step1=6，Step2=0。

据此设计算法，用一个类似于开关的布尔变量记录应该加step1还是加step2，然后逐行读取。

## 源码：（language：java）

public class Solution {

public String convert(String s, int numRows)

{

if(numRows==1)

return s;

else {

String answer="";

int step1=(numRows-1)\*2;

int step2=0;

for(int i=0;i<numRows;i++) {

String line="";

int j=i;

boolean isStep1=true;

while(j<s.length()) {

if(step2==0) {

line+=s.charAt(j);

j+=step1;

}

else if(step1==0) {

line+=s.charAt(j);

j+=step2;

}

else {

line+=s.charAt(j);

if(isStep1)

j+=step1;

else

j+=step2;

isStep1=!isStep1;

}

}

answer=answer+line;

step1-=2;

step2+=2;

}

return answer;

}

}

}

## 成绩：

33ms,beats 25.35%,众数8ms,15.13%

## cmershen的碎碎念：

本题成绩较差，可能是存在更简洁的规律，或者从stl的函数调用上试图优化。毕竟数量级为O（n）已经没法再缩小了。

# Easy-题目62：228. Summary Ranges

## 题目原文：

. Given a sorted integer array without duplicates, return the summary of its ranges.

For example, given [0,1,2,4,5,7], return ["0->2","4->5","7"].

## 题目大意：

给出一个排序的数组，没有重复元素，用区间的字符串表示。

例如给出[0,1,2,4,5,7], 返回["0->2","4->5","7"].

## 题目分析：

使用两个指针p1和p2，类似于滑动窗口，固定指针p1，一直滑动p2，直到p2的后继元素不等于p2所指元素+1（即出现间断），那么列表中增加一个字符串，并令p1指向p2的下一个节点，再滑动。如果p1就是一个间断的数字，那么直接把p1对应的字符串输入到列表中。

## 源码：（language：java）

public class Solution {

public List<String> summaryRanges(int[] nums) {

int len=nums.length;

List<String> answer=new ArrayList<String>();

if(len==0)

return answer;

int start=0,end=0;

while(start<len) {

String range;

while(end+1<len && nums[end+1]==nums[end]+1)

end++;

if(end!=start)

range=new Integer(nums[start]).toString()+"->"+new Integer(nums[end]).toString();

else

range=new Integer(nums[start]).toString();

answer.add(range);

end++;

start=end;

}

return answer;

}

}

## 成绩：

1ms,beats 3.19%,众数1ms,94.54%

# Easy-题目63：278. First Bad Version

## 题目原文：

You are a product manager and currently leading a team to develop a new product. Unfortunately, the latest version of your product fails the quality check. Since each version is developed based on the previous version, all the versions after a bad version are also bad.

Suppose you have n versions [1, 2, ..., n] and you want to find out the first bad one, which causes all the following ones to be bad.

You are given an API bool isBadVersion(version) which will return whether version is bad. Implement a function to find the first bad version. You should minimize the number of calls to the API.

## 题目大意：

你是一个PM，现在率领一个团队开发一个新的产品。不幸的是，其中一个版本没有通过质量检测。而后面的版本由于是基于前面的版本的，所以都是坏的产品。

假设你有n个版本[1,2,…n],你希望找出第一个坏的版本。

现在有一个API bool isBadVersion(version)会返回这个版本是不是坏的。实现一个函数来找出第一个坏的版本。要求这个api尽可能少调用。

## 题目分析：

最直观不能再直观的二分查找了。如果还不会写二分查找，只能说你这大学白念了……

## 源码：（language：cpp）

// Forward declaration of isBadVersion API.

bool isBadVersion(int version);

class Solution {

public:

int firstBadVersion(int n) {

int low = 1;

int high = n;

int ver = 0;

while (low < high) {

ver = low + (high - low) / 2;

if (isBadVersion(ver)) {

high = ver;

} else {

low = ver + 1;

}

}

return low;

}

};

## 成绩：

0ms,beats 3.21%,众数0ms，96.79%。

## cmershen的碎碎念：

像二分查找，包括后面的二叉树遍历，图的dfs，拓扑排序等最最最基础的问题，反而应当予以高度重视，做到倒背如流，切不可因为简单就不去强化练习。

# Easy-题目64：168. Excel Sheet Column Title

## 题目原文：

Given a positive integer, return its corresponding column title as appear in an Excel sheet.

For example:

1 -> A

2 -> B

3 -> C

...

26 -> Z

27 -> AA

28 -> AB

## 题目大意：

给一个正整数，求出Excel中对应的列号。

## 题目分析：

与[Easy第9题](#_Easy-题目9：171._Excel_Sheet)类似，一直模26求余数对应字母即可。需注意余0是一种特殊情况。此时末位为Z。

## 源码：（language：java）

public class Solution {

public String convertToTitle(int n) {

if(n<27)

return new Character((char) (n+64)).toString();

else {

if(n%26!=0)

return convertToTitle(n/26)+new Character((char) ((n%26)+64)).toString();

else

return convertToTitle(n/26-1)+new Character('Z').toString();

}

}

}

## 成绩：

0ms,beats 6.39%，众数0ms,93.61%.

# Easy-题目65：189. Rotate Array

## 题目原文：

Rotate an array of n elements to the right by k steps.

For example, with n = 7 and k = 3, the array [1,2,3,4,5,6,7] is rotated to [5,6,7,1,2,3,4].

## 题目大意：

给一个数组，循环右移k位。

## 题目分析：

这题好像也是考研408复习资料里面出现过的一道题，（还是笔试原题？我忘了）。如果不给出答案我也是想不到的。用三步翻转实现：

1. 先翻转前n-k位；
2. 再翻转后k位；
3. 再从头到尾翻转。

但本题中k有可能大于数组长度n，这种情况下相当于循环移动了n%k次（剩下的因为是整除的，相当于循环移动了n的整数倍，等于没移）

## 源码：（language：c）

void rotate(int\* nums, int numsSize, int k) {

if(k>=numsSize)

k=k%numsSize;

reverse(nums,numsSize-k);

reverse(nums+numsSize-k,k);

reverse(nums,numsSize);

}

void reverse(int\* a,int n)

{

int i,t;

for(i=0;i<n/2;i++) {

t=\*(a+i);

\*(a+i)=\*(a+n-i-1);

\*(a+n-i-1)=t;

}

}

## 成绩：

8ms,beats 9.23%，众数8ms，90.77%

# Easy-题目66：165. Compare Version Numbers

## 题目原文：

Compare two version numbers version1 and version2.

If version1 > version2 return 1, if version1 < version2 return -1, otherwise return 0.

You may assume that the version strings are non-empty and contain only digits and the . character.

The . character does not represent a decimal point and is used to separate number sequences.

For instance, 2.5 is not "two and a half" or "half way to version three", it is the fifth second-level revision of the second first-level revision.

## 题目大意：

比较两个软件的版本号version1和version2.如果version1＞version2则返回1，version1<version2返回-1，相等则返回0.

版本号中的点与数学上的小数点不同，它用来分割软件的子版本。例如2.5不是数学中的2.5，而是第2个一级版本中的第5个二级版本。

## 题目分析：

分离各级版本，并逐级比较。

## 源码：（language：java）

public class Solution {

public int compareVersion(String version1, String version2) {

String[] v1=version1.split("\\.");

String[] v2=version2.split("\\.");

int len1=v1.length;

int len2=v2.length;

int i;

for(i=0;i<Math.min(len1,len2);i++) {

int temp1=Integer.parseInt(v1[i]);

int temp2=Integer.parseInt(v2[i]);

if(temp1!=temp2)

return temp1<temp2?-1:(temp1>temp2?1:0);

}

if(len1>len2) {

for(int j=i;j<len1;j++)

if(Integer.parseInt(v1[j])!=0)

return 1;

return 0;

}

else if(len1<len2) {

for(int j=i;j<len2;j++)

if(Integer.parseInt(v2[j])!=0)

return -1;

return 0;

}

else

return 0;

}

}

## 成绩：

3ms,beats 42.39%，众数3ms,40.18%

## cmershen的碎碎念：

本题个人感觉测试用例存在bug，它认为版本号1.0.0和1.0是相等的，即末尾的0可以约掉，但个人认为1.0.0这个版本号是不规范的。在testcase中比较1.0.0和1.0是没意义的。可能这就是本题ac率过低的原因。

# Easy-题目67：8. String to Integer (atoi)

## 题目原文：

Implement atoi to convert a string to an integer.

## 题目大意：

实现atoi函数。其中如果出现溢出了，则返回int的最大值和最小值。

## 题目分析：

使用atol中转一下，如果出现了int范围的溢出，判断一下即可。

## 源码：（language：c）

int myAtoi(char\* str) {

long l = atol(str);

if(l>2147483647)

return 2147483647;

else if(l<-2147483648)

return -2147483648;

else

return (int)l;

}

## 成绩：

16ms,beats 5.20%,众数8ms,48.70%

## cmershen的碎碎念：

直接用库函数水过居然会被近95%的提交代码击败，且不加判断还返回wa，可见atol函数的底层实现过程似乎可以优化。此外还有一个用BigInteger中转的算法，但if-else太多了不方便贴出。而atol的底层不是开源的，因此也没能查到实现过程。

# Easy-题目68：1. Two Sum（增补1）

## 题目原文：

Given an array of integers, return indices of the two numbers such that they add up to a specific target.

You may assume that each input would have exactly one solution.

Example:

Given nums = [2, 7, 11, 15], target = 9,

Because nums[0] + nums[1] = 2 + 7 = 9,

return [0, 1].

## 题目大意：

给出一个数组和一个目标值target，寻找两个**下标i和j，**使得nums[i]+nums[j]=target。

## 题目分析：

这道题先无脑水过去。复杂度O(n2)

## 源码：（language：java）

public class Solution {

public int[] twoSum(int[] nums, int target) {

for(int i = 0;i<nums.length;i++) {

for(int j=i+1;j<nums.length;j++) {

if(nums[i]+nums[j]==target)

return new int[]{i,j};

}

}

return new int[]{-1,-1};

}

}

## 成绩：

42ms,beats 19.97%,众数6ms,35.54%

## Cmershen的碎碎念：

这道题不知道何时被改成Easy了，其实可以用HashMap做，应该能在线性时间解决。

# Middle-题目1：136. Single Number

## 题目原文：

Given an array of integers, every element appears *twice* except for one. Find that single one.

## 题目大意：

给出一个数组，除了一个元素出现一遍以外，所有元素都出现了两遍。找到这个单个的元素。

## 题目分析：

把所有元素都按位异或起来，最终的异或值就是那个单独的元素。

原因：因为1⊕1=0⊕0=0,1⊕0=0⊕1=1，所以两个相同的数异或起来必为0，而0异或任何数都是它本身。

## 源码：（language：c）

public class Solution {

public int singleNumber(int[] nums) {

int single=0;

for(int i=0;i<nums.length;i++)

single=single^nums[i];

return single;

}

}

## 成绩：

1ms,38.79%,众数1ms,61.21%

## Cmershen的碎碎念：

如果第一次接触到这种类型题，想到按位异或是不太容易的，我一开始的想法是用HashSet记，如果集合中有就remove掉，没有就add。其实和异或算法是异曲同工的，但计算机中比特运算比哈希表快多了。

# Middle-题目2：260. Single Number III

## 题目原文：

Given an array of numbers nums, in which exactly two elements appear only once and all the other elements appear exactly twice. Find the two elements that appear only once.

## 题目大意：

给一个数组，里面有两个元素只出现了一次，而其他数都出现了两次，找出这两个元素。

## 题目分析：

这道题是[Middle-题目1](#_Middle-题目1：136._Single_Number)的变形。

朴素解法：

用HashSet存储每一个元素，如果元素存在于集合内就remove掉，否则add进集合内，这样遍历完一个数组就set里面只剩下两个元素。

使用位运算的解法：

设两个单独的数是a和b，先把所有数都异或起来，得到a⊕b，记这个数是r，而因为a≠b，所以r≠0，r对应的二进制数一定有1。再令mask=r∧￢(r-1)，得到一个比特串，mask串一定只有一位是1，其他位都是0，这个1即是r中最低位的1.既然这一位为1，说明a和b中对应位必然一个是0一个是1。再遍历一遍这个数组，把每个数和mask求一次与，再分别异或起来，这就得到了a和b。（因为相当于分成mask所在位为0和1的两个子数组，把这两个子数组都异或起来自然得到了a和b。）

## 源码：（language：java）

朴素解法：

public class Solution {

public int[] singleNumber(int[] nums) {

HashSet<Integer> set=new HashSet<Integer>();

for(int num:nums) {

if(set.contains(num))

set.remove(num);

else

set.add(num);

}

int[] array=new int[2];

int i=0;

for(int num:set)

array[i++]=num;

return array;

}

}

位运算解法：

public class Solution {

public int[] singleNumber(int[] nums) {

int[] res = new int[2];

int r = 0;

for(int i=0;i<nums.length;i++) {

r = r^nums[i];

}

res[0] = 0;

res[1] = 0;

int mask = r & (~(r-1));

for(int i=0;i<nums.length;i++){

if((mask & nums[i])!=0){

res[0] = res[0] ^ nums[i];

}else {

res[1] = res[1] ^ nums[i];

}

}

return res;

}

}

## 成绩：

朴素解法：12ms,beats 14.33%,众数2ms,62.30%

位运算解法：2ms,beats 37.32%

# Middle-题目3：238. Product of Array Except Self

## 题目原文：

Given an array of n integers where n > 1, nums, return an array output such that output[i] is equal to the product of all the elements of nums except nums[i].

Solve it without division and in O(n).

For example, given [1,2,3,4], return [24,12,8,6].

## 题目大意：

给一个数组nums，输出一个数组output，其中output[i]等于nums数组中除了nums[i]外所有数的乘积。

要求不使用除法，且复杂度O（n），

## 题目分析：

1. 朴素解法

既然是oj题，AC就好，你说不用除就不用除了？那么问题来了，如果nums[i]是0怎么办？

这时要分两种情况：

1. nums数组里面有两个以上的0，那么output数组必然全是0.
2. nums数组里面只有一个0，那么0这一位对应是其他所有非0数的乘积，其他位都是0.

按这个思路，先统计数组里有几个0，作为两种特殊情况。其他的就都乘起来分别算即可。

1. 真的不用除的算法：

考虑output[i]=(nums[0]\*nums[1]\*…\*nums[i-1])\*(nums[i+1]\*…\*nums[n-1])，那么遍历两次数组，第一轮正向遍历，计算出nums[0]\*nums[1]\*…\*nums[i-1]的积放到output[i]里面，第二轮反向遍历，计算出nums[i+1]\*…\*nums[n-1]的积再乘进去即可。

## 源码：（language：java）

1. 朴素解法：

public class Solution {

public int[] productExceptSelf(int[] nums) {

int product=1,product2=1;

int zerocount=0;

for(int num:nums)

{

if(num==0)

zerocount++;

if(zerocount==2)

break;

product\*=num;

product2\*=(num==0)?1:num;

}

if(zerocount==0){

for(int i=0;i<nums.length;i++)

nums[i]=product/nums[i];

return nums;

}

else if(zerocount==1)

{

int[] array=new int[nums.length];

for(int j=0;j<nums.length;j++)

array[j]=(nums[j]==0)?product2:0;

return array;

}

else

return new int[nums.length];

}

}

1. 不用除的巧妙算法：

public class Solution {

public int[] productExceptSelf(int[] nums) {

int[] res = new int[nums.length];

res[res.length-1] = 1;

for(int i=nums.length-2; i>=0; i--) {

res[i] = res[i+1] \* nums[i+1];

}

int left = 1;

for(int i=0; i<nums.length; i++) {

res[i] \*= left;

left \*= nums[i];

}

return res;

}

}

## 成绩：

朴素解法：3ms,beats 11.83%,

巧妙解法：2ms,beats 46.07%,众数2ms,53.93%

## Cmershen的碎碎念：

Middle前期的很多道题看似简单，但却有很巧妙的算法，应予以注意。

# Middle-题目4：122. Best Time to Buy and Sell Stock II

## 题目原文：

Say you have an array for which the *i*th element is the price of a given stock on day *i*.

Design an algorithm to find the maximum profit. You may complete as many transactions as you like (ie, buy one and sell one share of the stock multiple times). However, you may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

## 题目大意：

已知一个数组，第i个元素代表一支股票第i天的价格。设计一个算法，求出买卖这只股票的最大利润。你可以任意的买卖，但每天只能进行一个操作。

## 题目分析：

贪心算法，因为最优的操作是在最低值点买，最高值点卖，所以只要后一天比前一天价格高，就加到利润里面。

## 源码：（language：java）

public class Solution {

public int maxProfit(int[] prices) {

int profit=0;

int l=prices.length-1;

for(int i=0;i<l;i++) {

if(prices[i+1]>=prices[i])

profit+=prices[i+1]-prices[i];

}

return profit;

}

}

## 成绩：

2ms,beats 15.31%，众数2ms,76.17%

## Cmershen的碎碎念：

一开始想到的是模拟买卖过程，用一个布尔形变量表示有没有股票，如果有股票，且“明天”价格比“今天”低，则卖掉，maxProfit加上今天股价，反之买入，maxProfit减今天股价。但其实本质是一样的，上述算法更为简洁。

# Middle-题目5：319. Bulb Switcher

## 题目原文：

There are n bulbs that are initially off. You first turn on all the bulbs. Then, you turn off every second bulb. On the third round, you toggle every third bulb (turning on if it's off or turning off if it's on). For the ith round, you toggle every i bulb. For the nth round, you only toggle the last bulb. Find how many bulbs are on after n rounds.

Example:

Given n = 3.

At first, the three bulbs are [off, off, off].

After first round, the three bulbs are [on, on, on].

After second round, the three bulbs are [on, off, on].

After third round, the three bulbs are [on, off, off].

So you should return 1, because there is only one bulb is on.

## 题目大意：

起初有n个灯泡，都是关闭的，你进行n轮操作：

第1轮，你把每个灯泡的状态都改变（打开的关闭，关闭的打开）；

第2轮，你把每个灯泡序号是2的倍数的灯泡都改变状态（灯泡序号从1开始）；

……

第i轮，你把每个灯泡序号是i的倍数的灯泡都改变状态（灯泡序号从1开始）；

……

第n轮，你把最后一个灯泡改变状态（因为1~n里面n的倍数只有n）

求n轮过后，还有几个灯泡亮着？

## 题目分析：

若一个灯泡在n轮过后还是亮的，说明这个灯泡被改变了奇数次，因为只有被轮数整除的时候才能变状态，所以[1,k]能整除这个灯泡序号k的正整数是奇数个。

以下证明k是个完全平方数。

首先，若灯泡的序号i是平方数，则i的因数一定是奇数个，证明：

因为i是平方数，所以i=k\*k,其中k∈Z，而1能整除i，所以i的因数个数为k的非1因子个数\*2+1，一定是一个奇数。

那么若i不是平方数，i的因数必然是偶数个，证明：

设所有＜ 的因子构成的集合为A，>的因子构成的集合为B，显然A∪B即为i的所有因子构成的集合，而对任意a∈A，必存在唯一的i/a∈B，反之亦然，所以A和B两个集合的元素个数相等。

综上证明，只有≤n的完全平方数最后是亮的。那么求的取整即可。一行代码解决。

## 源码：(language : c)

int bulbSwitch(int n) {

return sqrt(n);

}

## 成绩：

0ms,beats 6.36%,众数0ms,93.14%

## Cmershen的碎碎念：

这道题起初只想到进行模拟，但看到discuss中给出的答案是求sqrt(n)，遂想到了上述证明过程。

# Middle-题目6：268. Missing Number

## 题目原文：

Given an array containing n distinct numbers taken from 0, 1, 2, ..., n, find the one that is missing from the array.

For example,

Given nums = [0, 1, 3] return 2.

## 题目大意：

给出从自然数列0,1,2….n中去掉一个数之后的数组，求出去掉的数。

## 题目分析：

求这个数组的和，然后用等差数列前n项和Sn=n(n+1)/2减它，得到的就是缺的数。

## 源码：（language：java）

public class Solution {

public int missingNumber(int[] nums) {

int sum=0,n=nums.length;

for(int num:nums)

sum+=num;

return n\*(n+1)/2-sum;

}

}

## 成绩：

1ms,beats 37.49%,众数1ms,62.51%

## Cmershen的碎碎念：

有10个箱子，每箱里有很多个产品，每个产品质量是10g，其中有一箱是次品，次品每个质量是9g，那么使用一个秤，只称一次，如何找出哪箱装的是次品？

这道大家耳熟能详的小学脑筋急转弯题与本题似有异曲同工之妙。

# Middle-题目7：144. Binary Tree Preorder Traversal

## 题目原文：

Given a binary tree, return the *preorder* traversal of its nodes' values.

## 题目大意：

给出一个二叉树，求前序遍历。

## 题目分析：

前序遍历是先访问根节点，然后前序遍历左子树和右子树。

## 源码：（language：java）

public class Solution {

public List<Integer> preorderTraversal(TreeNode root) {

List<Integer> result = new ArrayList<Integer>();

traverse(root,result);

return result;

}

public void traverse(TreeNode node,List<Integer> list) {

if(node!=null) {

list.add(node.val);

traverse(node.left,list);

traverse(node.right,list);

}

return;

}

}

## 成绩：

1ms,beats 56.04%,众数2ms,52.79%

## Cmershen的碎碎念：

本题放在Middle里面应该是要求写出非递归。

# Middle-题目8：144. Binary Tree Preorder Traversal

## 题目原文：

Given a binary tree, return the *inorder* traversal of its nodes' values.

## 题目大意：

给出一个二叉树，求中序遍历。

## 题目分析：

先中序遍历左子树，再访问根节点，再中序遍历右子树。

## 源码：（language：java）

public class Solution {

public List<Integer> inorderTraversal(TreeNode root) {

List<Integer> result = new ArrayList<Integer>();

traverse(root,result);

return result;

}

public void traverse(TreeNode node,List<Integer> list) {

if(node!=null) {

traverse(node.left,list);

list.add(node.val);

traverse(node.right,list);

}

return;

}

}

## 成绩：

1ms,beats 62.04%,众数2ms,58.69%

# Middle-题目9：318. Maximum Product of Word Lengths

## 题目原文：

Given a string array words, find the maximum value of length(word[i]) \* length(word[j]) where the two words do not share common letters. You may assume that each word will contain only lower case letters. If no such two words exist, return 0.

Example 1:

Given ["abcw", "baz", "foo", "bar", "xtfn", "abcdef"]

Return 16

The two words can be "abcw", "xtfn".

Example 2:

Given ["a", "ab", "abc", "d", "cd", "bcd", "abcd"]

Return 4

The two words can be "ab", "cd".

Example 3:

Given ["a", "aa", "aaa", "aaaa"]

Return 0

No such pair of words.

## 题目大意：

给一个字符串数组，字符串都是由小写字母组成的。求出两字符没有交集的字符串的长度的最大积。若不存在没有交集的字符串，则返回0.

## 题目分析：

对每一个字符串进行编码，编码成一个26位的比特串，其中比特串的第i位代表字符串中有第i个字母。这样得到一个编码数组，再O(n2)暴力扫描编码数组，如果存在两个编码按位或的值为0，说明这两个编码对应的字符串没有交集，这样更新一次长度乘积的最大值。

## 源码：（language：java）

public class Solution {

public int maxProduct(String[] words) {

int maxPdc=0;

int[] letters=new int[words.length];

for(int i=0;i<words.length;i++)

for(int j=0;j<words[i].length();j++)

letters[i]=letters[i]|1<<(words[i].charAt(j)-'a');

for(int i=0;i<words.length;i++) {

for(int j=i+1;j<words.length;j++) {

if((letters[i]&letters[j]) == 0) {

int tempPdc=words[i].length()\*words[j].length();

if(tempPdc>maxPdc)

maxPdc=tempPdc;

}

}

}

return maxPdc;

}

}

## 成绩：

37ms,beats 63.38%,众数39ms,6.29%.

## Cmershen的碎碎念：

letters[i]=letters[i]|1<<(words[i].charAt(j)-'a');这行代码起初写成了Math.pow(2,…)结果提交后成绩为100ms，可见移位比幂运算的速度快很多。

# Middle-题目10：12. Integer to Roman

## 题目原文：

Given an integer, convert it to a roman numeral.

Input is guaranteed to be within the range from 1 to 3999.

## 题目大意：

把1-3999的整数转换成对应的罗马数字。

## 题目分析：

使用switch-case加递归，考虑所有特殊情况（遇到5,10,9等数字的时候）。

## 源码：（language：java）

public class Solution {

public String intToRoman(int num) {

if(num >= 1 && num <= 9) {

switch (num) {

case 1:{return "I";}

case 2:case 3:{return "I"+intToRoman(num-1);}

case 4:case 9:{return "I"+intToRoman(num+1);}

case 5:{return "V";}

default:{return "V"+intToRoman(num-5);}//6,7,8

}

}

else if (num >= 10 && num <= 99) {

switch (num) {

case 10:{return "X";}

case 50:{return "L";}

case 40:case 90:{return "X"+intToRoman(num+10);}

default: {

if(num > 90)

return "X"+intToRoman(num+10);

else if(num > 50)

return "L"+intToRoman(num-50);

else if(num > 40 && num < 50) //XL

return "X"+intToRoman(num+10);

else

return "X"+intToRoman(num-10);

}

}

}

else if (num >= 100 && num <= 999) {

switch (num) {

case 100:{return "C";}

case 500:{return "D";}

case 400:case 900:{return "C"+intToRoman(num+100);}

default:{

if(num > 900)

return "C"+intToRoman(num+100);

else if(num > 500)

return "D"+intToRoman(num-500);

else if(num > 400 && num < 500)

return "C"+intToRoman(num+100);

else

return "C"+intToRoman(num-100);

}

}

}

else { //1000~3999

switch (num) {

case 1000:{return "M";}

default:{return "M"+intToRoman(num-1000);}

}

}

}

}

## 成绩：

11ms，beats24.85%，众数8ms，23.08%

## Cmershen的碎碎念：

代码很长也很繁琐，但感觉有一些情况是可以合并的，这样应该可以节约几ms。

此外注意，因为switch-case的每个分支都是return，故省略了break，但在其他情况下写switch一定不要忘记break。

# Middle-题目11：137. Single Number II

## 题目原文：

Given an array of integers, every element appears *three* times except for one. Find that single one.

## 题目大意：

给出一个数组，除了一个元素出现了一次外都出现了三次，找出这个单独的元素。

## 题目分析：

1. 朴素解法：这回使用HashMap统计每个元素出现次数，然后返回那个次数为1的。
2. 位运算解法：（参考[higerzhang大神的博客](http://www.cnblogs.com/higerzhang/p/4159330.html)）遍历32次每次记录某位的出现的次数，如果不能被三整除，说明那个出现一次的就在该位有值，那么ans 或该位一下就可以了。

## 源码：（language：java/c）

1. 朴素解法：

public class Solution {

public int singleNumber(int[] nums) {

HashMap<Integer,Integer> count=new HashMap<Integer,Integer>();

for(int num:nums) {

if(count.containsKey(num))

count.put(num, count.get(num)+1);

else

count.put(num, 1);

}

for(Integer key:count.keySet())

if(count.get(key)==1)

return key;

return 0;

}

}

1. 位运算解法：

int singleNumber(int nums[], int numsSize) {

int single = 0;

for (int i = 0; i < 32; i++) {

int cnt = 0, bit = 1 << i;

for (int j = 0; j < numsSize; j++)

if (nums[j] & bit) cnt++;

if (cnt % 3 != 0)

single |= bit;

}

return single;

}

## 成绩：

朴素解法：15ms,beats 10.87%，众数1ms,23.02%

位运算解法：12ms,beats 8.33%,众数8ms，48.61%

## Cmershen的碎碎念：

HigherZhang的博客中还提到了第三种算法，我理解能力有限,一时间未能完全看明白，以后可以点击上面的链接再研究一番。

# Middle-题目12：96. Unique Binary Search Trees

## 题目原文：

Given n, how many structurally unique BST's (binary search trees) that store values 1...n?

For example,

Given n = 3, there are a total of 5 unique BST's.

1 3 3 2 1

\ / / / \ \

3 2 1 1 3 2

/ / \ \

2 1 2 3

## 题目大意：

n个节点的二叉排序树有几种？

## 题目分析：

等价于求Catalan数。（证明我也不知道。求大神给出证明。）

## 源码：（language：java）

public class Solution {

public int numTrees(int n) {

int[] catalan=new int[]{1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, 35357670, 129644790, 477638700, 1767263190};

return catalan[n];

}

}

## 成绩：

0ms,beats 2.75%,众数0ms,92.25%.

## Cmershen的碎碎念：

关于Catalan数的性质见<https://zh.wikipedia.org/wiki/%E5%8D%A1%E5%A1%94%E5%85%B0%E6%95%B0>。

# Middle-题目13：35. Search Insert Position

## 题目原文：

Given a sorted array and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You may assume no duplicates in the array.

Here are few examples.

[1,3,5,6], 5 → 2

[1,3,5,6], 2 → 1

[1,3,5,6], 7 → 4

[1,3,5,6], 0 → 0

## 题目大意：

给出一个排序好的数组，和一个目标值，求出把目标值插到原数组保持升序的情况下，目标值所在的数组下标。

## 题目分析：

在二分查找的基础上简单变形即可，需要注意目标值在两端的特殊情况。

## 源码：（language：java）

public class Solution {

public int searchInsert(int[] nums, int target) {

int start = 0,end = nums.length - 1;

int mid = 0;

while(start <= end) {

mid = (start + end) / 2;

if(nums[mid] == target)

return mid;

else if(nums[mid] > target) { //target lies between start and mid

if(mid == 0 || nums[mid - 1] < target)

return mid;

else

end=mid-1;

}

else { //target lies between mid and end

if(mid == nums.length-1 || nums[mid + 1] > target)

return mid+1;

else

start=mid+1;

}

}

return mid;

}

}

## 成绩：

0ms，beats 19.33%，众数0ms,90.67%

# Middle-题目14：141. Linked List Cycle

## 题目原文：

Given a linked list, determine if it has a cycle in it.

## 题目大意：

判断单链表里面有没有环。

## 题目分析：

1. 朴素解法：用一个HashSet存储所有节点，如果遇到重复的则有环。
2. Two Pointer解法：用两个指针p1和p2，p1每次前进一步，p2每次前进两步，如果有环则p2一定能追上p1.

## 源码：（language：java/cpp）

朴素解法：

public class Solution {

public boolean hasCycle(ListNode head) {

HashSet<ListNode> nodes=new HashSet<ListNode>();

for(ListNode node=head;node!=null;node=node.next){

if(!nodes.add(node))

return true;

}

return false;

}

}

Two Pointer解法：

class Solution {

public:

bool hasCycle(ListNode \*head) {

ListNode\* p1=head;

ListNode\* p2=head;

while(p1 && p2 && p2->next) {

p1 = p1->next;

p2 = p2->next->next;

if(p1 == p2)

return true;

}

return false;

}

};

## 成绩：

朴素解法：9ms，beats 3.91%，众数1ms，90.82%

Two Pointer解法：12ms,beats 13.53%,众数12ms，86.39%

## Cmershen的碎碎念：

找一个链表的N等分点用的也是这种思路。

# Middle-题目15：108. Convert Sorted Array to Binary Search Tree

## 题目原文：

Given an array where elements are sorted in ascending order, convert it to a height balanced BST.

## 题目大意：

把一个升序排列的数组构建成BST（二叉排序树）。

## 题目分析：

数组的中点是二叉排序树的根节点，然后递归把左半边和右半边递归构建成左子树和右子树。

## 源码：（language：java）

public class Solution {

public TreeNode sortedArrayToBST(int[] nums) {

if(nums.length==0)

return null;

TreeNode bst=createBST(nums,0,nums.length-1);

return bst;

}

public TreeNode createBST(int[] nums,int start,int end) {

int mid = (start + end) / 2;

TreeNode root = new TreeNode(nums[mid]);

if(start < mid )

root.left = createBST(nums,start,mid-1);

if(mid < end)

root.right = createBST(nums,mid+1,end);

return root;

}

}

## 成绩：

1ms,beats 7.24%,众数1ms,92.62%

# Middle-题目16：230. Kth Smallest Element in a BST

## 题目原文：

Given a binary search tree, write a function kthSmallest to find the kth smallest element in it.

## 题目大意：

求一个BST的第k小的节点。

## 题目分析：

根据二叉排序树的性质，中序遍历序列第k个元素即为所求。

## 源码：（language：java）

public class Solution {

public int kthSmallest(TreeNode root, int k) {

List<Integer> list=new ArrayList<Integer>();

traverse(root,list);

return list.get(k-1);

}

public void traverse(TreeNode node,List<Integer> list) {

if(node!=null) {

traverse(node.left,list);

list.add(node.val);

traverse(node.right,list);

}

return;

}

}

## 成绩：

2ms，beats 16.50%，众数1ms，49.50%

# Middle-题目17：116. Populating Next Right Pointers in Each Node

## 题目原文：

Given a binary tree

struct TreeLinkNode {

TreeLinkNode \*left;

TreeLinkNode \*right;

TreeLinkNode \*next;

}

Populate each next pointer to point to its next right node. If there is no next right node, the next pointer should be set to NULL.

For example,  
Given the following perfect binary tree,

1

/ \

2 3

/ \ / \

4 5 6 7

After calling your function, the tree should look like:

1 -> NULL

/ \

2 -> 3 -> NULL

/ \ / \

4->5->6->7 -> NULL

## 题目大意：

给一个二叉树加一个Next指针，指向右邻节点。

## 题目分析：

对每个根节点，其左孩子的next是右孩子，右孩子的next是next的左孩子。再递归对左子树和右子树调用这个函数，注意一些边界情况（叶子节点，空节点等）即可。

## 源码：（language：java）

public class Solution {

public void connect(TreeLinkNode root) {

if(root==null)

return;

else if(root.left == null && root.right == null && root.next == null) { // root is a leaf node

root.next=null;

return;

}

else {

if(root.left != null)

root.left.next = root.right;

if(root.right != null && root.next != null)

root.right.next = root.next.left;

if(root.right != null && root.next == null)

root.right.next = null;

connect(root.left);

connect(root.right);

}

}

}

## 成绩：

0ms，beats 81.60%，众数1ms，52.14%

# Middle-题目18：53. Maximum Subarray

## 题目原文：

Find the contiguous subarray within an array (containing at least one number) which has the largest sum.

For example, given the array [−2,1,−3,4,−1,2,1,−5,4],

the contiguous subarray [4,−1,2,1] has the largest sum = 6.

## 题目大意：

给一个数组，求和最大的连续的子数组。

## 题目分析：

用dp[n]代表结尾为n的连续子数组的和最大值，则有如下转移方程：

dp[n] = dp[n-1]+a[n] dp[n-1]>0

a[n] others

即如果以n-1为结尾的子数组和是>0的，则前n-1项对n为结尾是“有贡献”的，那么以n为结尾的子列和就带上前面那个数组。否则从第n项开始。最后返回dp[n]里面最大值即可。因为dp[n]只跟dp[n-1]有关，故维护dp[n-1]一个变量就可以了。

## 源码：（language：java）

public class Solution {

public int maxSubArray(int[] nums) {

int maxsum = -9999999;

int sumk = 0;

for(int num : nums) {

sumk = (sumk + num > num) ? sumk + num : num;

maxsum = (maxsum > sumk) ? maxsum : sumk;

}

return maxsum;

}

}

## 成绩：

1ms,beats 69.81%,众数2ms,55.83%

## Cmershen的碎碎念：

这是DP问题里面最经典的题之一，此算法叫做Kadane算法。这个优美的算法由Carnegie Mellon University的统计学家Jay Kadane于1984年给出。

# Middle-题目19：309. Best Time to Buy and Sell Stock with Cooldown

## 题目原文：

Say you have an array for which the ith element is the price of a given stock on day i.

Design an algorithm to find the maximum profit. You may complete as many transactions as you like (ie, buy one and sell one share of the stock multiple times) with the following restrictions:

You may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

After you sell your stock, you cannot buy stock on next day. (ie, cooldown 1 day)

## 题目大意：

与[Middle-题目4](#_Middle-题目4：122._Best_Time)相同，区别是卖出股票之后会有一天的CD（冷却期），而在CD的时候是不能买入的。求最大利益。

## 题目分析：

维护两个dp数组buy和sell，其中buy代表第n天时手里有股票的收益，sell代表第n天时手里无股票的收益。

接下来的转移方程不是很好想到：

buy[i] = max(sell[i-2]-price[i],buy[i-1])

第i天时手里有股票，这时候看第i-1天在做什么：

1. 如果是买入，则第i天要想手里有股票，什么也不能做，此时buy[i]=buy[i-1].
2. 如果是卖出，则第i天是CD，啥也干不了，手里一定是没有股票的，这种情况无解。
3. 如果什么也没做，则第i-2天把股票卖了，手里没股票，并在第i天买入。此时buy[i]=sell[i-2]+price[i]

Sell[i] = max(buy[i-1]+price[i],sell[i-1])

故buy[i]是两者的最大值。

第i天时手里没有股票，这时候还是看第i-1天在做什么：

1. 如果是买入，则第i-1天是买入，第i天卖掉，此时sell[i]=buy[i-1]+price[i]
2. 如果是卖出，则第i天是cd，此时sell[i]=sell[i-1]
3. 如果啥也没做，而手里有股票，还是sell[i]=sell[i-1]

同理sell[i]也是两者最大值。

最后的最大利益是sell[i]的值。因为到最后必须把股票卖了。

因为只跟前两项有关系，所以也是维护两项即可。

## 源码：（language：java）

public class Solution {

public int maxProfit(int[] prices) {

int buy=-999999,preBuy=0,sell=0,preSell=0;

for(int price : prices) {

preBuy = buy;

buy = Math.max(preSell - price, preBuy);

preSell = sell;

sell = Math.max(preBuy + price, preSell);

}

return sell;

}

}

## 成绩：

1ms,beats 87.89%,众数2ms，40.16%

## Cmershen的碎碎念：

DP问题的关键是找出转移数组和转移方程。而这道题的转移方程对我来说比较难想，应多注意。

# Middle-题目20：153. Find Minimum in Rotated Sorted Array

## 题目原文：

Suppose a sorted array is rotated at some pivot unknown to you beforehand.

(i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2).

Find the minimum element.

You may assume no duplicate exists in the array.

## 题目大意：

对一个移位后的数组，求最小值。

## 题目分析：

遍历一遍数组无脑找。

## 源码（略）

## 成绩：

1ms,beats 2.14%，众数1ms,92.29%

# Middle-题目21：62. Unique Paths

## 题目原文：

A robot is located at the top-left corner of a m x n grid (marked 'Start' in the diagram below).

The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid (marked 'Finish' in the diagram below).



How many possible unique paths are there?

## 题目大意：

一个机器人从一个m\*n的网格的左上角开始走，每次只能向下或向右走一步，问有几种走到右下角的不同走法？

## 题目分析：

使用dp，设dp[i][j]是从(0,0)点走到(i,j)点的不同走法数，则dp[i][j]=dp[i-1][j]+dp[i][j-1],初始化第一行和第一列为1,（因为只有一行或一列的时候，是唯一走法）

## 源码：（language：java）

public class Solution {

public int uniquePaths(int m, int n) {

int[][] dp=new int[m][n];

for(int i=0;i<m;i++)

dp[i][0]=1;

for(int i=0;i<n;i++)

dp[0][i]=1;

for(int i=1;i<m;i++)

for(int j=1;j<n;j++)

dp[i][j]=dp[i-1][j]+dp[i][j-1];

return dp[m-1][n-1];

}

}

## 成绩：

1ms,beats 5.98%,众数1ms,77.94%

## cmershen的碎碎念：

小学学过奥数的人会对“数最短路线”的算法很熟悉，其实本质是一样的。可是那时候不懂编程，更不懂dp，就知道在网格上累加。

# Middle-题目22：89. Gray Code

## 题目原文：

The gray code is a binary numeral system where two successive values differ in only one bit.

Given a non-negative integer n representing the total number of bits in the code, print the sequence of gray code. A gray code sequence must begin with 0.

For example, given n = 2, return [0,1,3,2]. Its gray code sequence is:

00 - 0

01 - 1

11 - 3

10 – 2

## 题目大意：

在一组数的编码中，若任意两个相邻的代码只有一位二进制数不同，则称这种编码为格雷码。

求n位的格雷码。

## 题目分析：

使用门电路中生成格雷码的公式计算：

http://b.hiphotos.baidu.com/baike/s%3D197/sign=dc278534ae51f3dec7b2bd6da3eff0ec/241f95cad1c8a786882607f06609c93d70cf503b.jpg，其中Gi表示格雷码的第i位，Bi表示二进制数的第i位。（因此生成格雷码的电路是异或门。）

## 源码：（language：java）

public class Solution {

public List<Integer> grayCode(int n) {

List<Integer> gray = new ArrayList<Integer>();

for(int i = 0 ; i < 1<<n ; i++) {

int temp = i>>1;

gray.add(i^temp);

}

return gray;

}

}

## 成绩：

1ms,beats 58.59%,众数2ms,40.77%

## cmershen的碎碎念：

本题的tags是BackTrack，可能是根据n-1位格雷码推n位格雷码，这也是可以的，但本题最简洁的算法应该是Bit manipulation.（by the way，简单的门能搭出计算格雷码的电路，我觉得我也能搭出来，但能实现回溯吗？）

# Middle-题目23：121. Best Time to Buy and Sell Stock

## 题目原文：

Say you have an array for which the ith element is the price of a given stock on day i.

If you were only permitted to complete at most one transaction (ie, buy one and sell one share of the stock), design an algorithm to find the maximum profit.

## 题目大意：

还是买卖股票，这回限定只能买卖一次，求最大利益。

## 题目分析：

本题最为简单，最低点买入最高点卖出即可，维护两个变量，一个记录最小值，一个记录当前价格减去最小值的最大值（即以今天价格卖出的profit）。

## 源码：（language：java）

public class Solution {

public int maxProfit(int[] prices) {

if(prices.length == 0)

return 0;

int min = prices[0];

int profit = 0;

for(int price:prices) {

if(price < min)

min = price;

if(price - min > profit)

profit = price - min;

}

return profit;

}

}

## 成绩：

2ms,beats 60.20%,众数3ms,44.72%

## cmershen的碎碎念：

一开始我误以为直接找数组的最大值和最小值相减，但是如果最小值出现在最大值后面就错了。所以维护的不是整个数组的最大值，而且当前值减去min的最大值。

# Middle-题目24：46. Permutations

## 题目原文：

Given a collection of distinct numbers, return all possible permutations.

For example,

[1,2,3] have the following permutations:

[1,2,3], [1,3,2], [2,1,3], [2,3,1], [3,1,2], and [3,2,1].

## 题目大意：

给一个没有重复数字的数组，求所有可能的排列情况。

## 题目分析：

使用backtrack(List<List<Integer>> list,List<Integer> sublist,int[] nums,boolean[] visited,int visitCount)函数递归分析，其中第一个参数是要返回的数组，第二个参数sublist记录当前搜索情况（即搜索树的路径），nums是输入的数组，visited记录了原数组中哪个数已经加入了搜索树，visitCount记录了visited数组中false的个数。如果为0说明所有数都访问过了，就把sublist加入list。其实也可以省略这个变量，而用sublist的长度和nums的长度去比，但每次都调用STL库函数会变慢。

此处需要注意的是，visitCount=0时把当前子列加入list不能写list.add(sublist);因为加入的是sublist这个对象的引用，而再次搜索的时候改变sublist的值会导致已经加入的结果修改。所以必须写list.add(new ArrayList<Integer>(sublist));构造一个和sublist内容相等但不同引用的数组。

## 源码：（language：java）

public class Solution {

public List<List<Integer>> permute(int[] nums) {

List<List<Integer>> list=new ArrayList<List<Integer>>();

boolean[] visited = new boolean[nums.length];

backtrack(list,new ArrayList<Integer>(),nums,visited,nums.length);

return list;

}

private void backtrack(List<List<Integer>> list,List<Integer> sublist,int[] nums,boolean[] visited,int visitCount)

{

if(visitCount == 0)

list.add(new ArrayList<Integer>(sublist));

else

{

for(int i=0;i<nums.length;i++) {

if(visited[i] == false) {

visited[i] = true;

sublist.add(nums[i]);

backtrack(list,sublist,nums,visited,visitCount-1);

sublist.remove(sublist.size()-1);

visited[i] = false;

}

}

}

}

}

## 成绩：

3ms，beats 67.81%，众数4ms，33.12%

## Cmershen的碎碎念：

在大一学习计导课的时候，回溯法讲了三道例题，分别是排列数（即本题），马踏棋盘，八皇后。当时觉得八皇后很难，甚至当时的同学们之间把能不能独立写出八皇后代码看做衡量编程水平的分水岭，当然这道题现在对我来说也很难（毕竟N皇后问题在leetcode里面是hard）。排列数作为回溯法里面最基本的一道题，但也体现了回溯法最基本的要领：（1）循环+递归；（2）记录下当前搜索信息之后再递归向下搜索，搜完之后把信息删除掉，否则每次调用backtrack的时候不能维持【当前】搜索信息。

# Middle-题目25：24. Swap Nodes in Pairs

## 题目原文：

Given a linked list, swap every two adjacent nodes and return its head.

For example,

Given 1->2->3->4, you should return the list as 2->1->4->3.

## 题目大意：

给一个单链表，两两交换相邻节点。

## 题目分析：

用Two Pointers，分别指向头结点和第二个节点，每次移动两步，注意处理结点个数为奇数（最后一次的时候p2->next为空，再取next会报错）的特殊情况。

## 源码：（language：java）

public class Solution {

public ListNode swapPairs(ListNode head) {

if(head == null || head.next == null) // This list has only 0 or 1 node.

return head;

ListNode l1=head,l2=head.next;

while(l1!=null && l2!=null) {

int temp=l1.val;

l1.val=l2.val;

l2.val=temp;

l1=l1.next.next;

l2=l2.next;

if(l2==null)

break;

else

l2=l2.next;

}

return head;

}

}

## 成绩：

0ms,beats 13.35%,众数0ms，86.65%

## Cmershen的碎碎念：

这题很水，不知道为什么出现在Middle里面。

# Middle-题目26：216. Combination Sum III

## 题目原文：

Find all possible combinations of k numbers that add up to a number n, given that only numbers from 1 to 9 can be used and each combination should be a unique set of numbers.

Ensure that numbers within the set are sorted in ascending order.

Example 1:

Input: k = 3, n = 7

Output:

[[1,2,4]]

Example 2:

Input: k = 3, n = 9

Output:

[[1,2,6], [1,3,5], [2,3,4]]

## 题目大意：

给出k和n，求把n拆成k个在1~9之间不重复整数之和的所有方式。

例如k=3，n=9，可以拆成9=1+2+6,9=1+2+4,9=2+3+4.

## 题目分析：

因为拆掉一个之后的子问题和原来是相似的（逻辑相同只是参数不同），故还是使用回溯法求解，函数写成如下形式：

backtrack(List<List<Integer>> list,List<Integer> sublist,int k,int n,int start)

其中list表示返回的数组，sublist表示当前搜索路径，k和n均为当前子问题的两个参数，start为新增的参数，代表从几开始找，问题开始时显然从1开始找。

那么递归关系是：每次找到一个i，则k=k-1,n=n-i,start=i+1

## 源码：（language：java）

public class Solution {

public List<List<Integer>> combinationSum3(int k, int n) {

List<List<Integer>> list=new ArrayList<List<Integer>>();

backtrack(list,new ArrayList<Integer>(),k,n,1);

return list;

}

private void backtrack(List<List<Integer>> list,List<Integer> sublist,int k,int n,int start) {

if(k == 1 ) {

if(n < 10) {

List<Integer> sublist2=new ArrayList<Integer>(sublist);

sublist2.add(n);

list.add(sublist2);

return;

}

}

else {

for(int i=start;i<(double)n/(double)k;i++) {

sublist.add(i);

backtrack(list,sublist,k-1,n-i,i+1);

sublist.remove(sublist.size()-1);

}

}

}

}

## 成绩：

1ms，beats 52.84%，众数1ms，47.16%

## Cmershen的碎碎念：

本题的代码第17行for(int i=start;i<(double)n/(double)k;i++)需要注意，如果不转换为double，会导致解空间变小。例如n=7,k=2时如果不转换相当于i<3,如果加double转换相当于i<3.5,前者会遗漏掉i=3的情况。同理如果改为i<=n/k会导致解空间变大。

# Middle-题目27：64. Minimum Path Sum

## 题目原文：

Given a *m* x *n* grid filled with non-negative numbers, find a path from top left to bottom right which *minimizes* the sum of all numbers along its path.

**Note:** You can only move either down or right at any point in time.

## 题目大意：

有一个m \* n的网格，每个格内填写一个非负整数，求出从左上角走到右下角的所有路径中的最小和。

注意：你每次只能向下或者向右走。

## 题目分析：

设dp[i][j]为从(0,0)点开始走到(i,j)点的最小和，则有：

dp[i][j] = min(dp[i-1][j],dp[i][j-1])+grid[i][j] (i>0,j>0)

很好解释，因为(i,j)这一点要么是从上面来的，要么是从左面来的，哪面来的和值小就取哪面来的。

初始化第一行和第一列为从原点到第一行（列）的点一直走的和。因为路径是唯一的。

dp数组计算后，显然dp[m-1][n-1]即为所求。

## 源码：（language：c）

int minPathSum(int\*\* grid, int gridRowSize, int gridColSize) {

int dp[gridRowSize][gridColSize];

dp[0][0] = grid[0][0];

for(int i = 1;i<gridColSize;i++)

dp[0][i] = dp[0][i-1]+grid[0][i];

for(int i = 1;i<gridRowSize;i++)

dp[i][0] = dp[i-1][0]+grid[i][0];

for(int i = 1;i<gridRowSize;i++)

for(int j = 1;j<gridColSize;j++)

dp[i][j] = min(dp[i-1][j],dp[i][j-1])+grid[i][j];

return dp[gridRowSize-1][gridColSize-1];

}

int min(int a,int b) {

return a<b?a:b;

}

## 成绩：

20ms,beats 31.11%,众数20ms,68.89%

# Middle-题目28：75. Sort Colors

## 题目原文：

Given an array with *n* objects colored red, white or blue, sort them so that objects of the same color are adjacent, with the colors in the order red, white and blue.

Here, we will use the integers 0, 1, and 2 to represent the color red, white, and blue respectively.

## 题目大意：

有三个对象，分别是红色，白色，蓝色，对他们排序使得所有的颜色按照红色-白色-蓝色的顺序，这里使用0,1,2分别代表红色，白色，蓝色。

抽象成算法模型即：

给出一个数组，数组里面只有0,1,2三个数字，对其排序。

## 题目分析：

1. 朴素解法：既然是排序，调用Java的API即可。
2. 计数排序：先扫描一遍数组，统计有多少0，多少1，多少2，然后再扫描一遍数组，按0,1,2的个数分别填充数组。
3. 交换排序：从0开始记录红色和白色的插入位置，扫描数组，如果扫到当前位置是1，则与1应插入的位置交换，如果扫到当前位置是0，则先与1插入的位置交换，再与0插入的位置交换。（因为换到0的位置以后，1的位置也向后移动了）

## 源码：（language：c）

方法1略。

方法2：

void sortColors(int\* nums, int numsSize) {

int red=0,white=0,blue=0;

for(int i = 0;i<numsSize;i++) {

if(nums[i] == 0)

red++;

else if(nums[i] == 1)

white++;

else

blue++;

}

for(int i=0;i < numsSize;i++) {

if(red > 0) {

red--;

nums[i]=0;

}

else if(white > 0) {

white--;

nums[i]=1;

}

else

nums[i]=2;

}

}

方法3：

void sortColors(int\* nums, int numsSize) {

int Insert\_Red=0,Insert\_White=0;

for(int i=0;i<numsSize;i++) {

if(nums[i]==1) {

nums[i]=nums[Insert\_White];

nums[Insert\_White++]=1;

}

else if(nums[i]==0) {

nums[i]=nums[Insert\_White];

nums[Insert\_White++]=nums[Insert\_Red];

nums[Insert\_Red++]=0;

}

}

}

## 成绩：

方法1:1ms，beats 3.63%，众数1ms，75.47%

方法2:0ms，beats 2.63%，众数0ms，97.37%

方法3:2ms，beats 2.63%

## Cmershen的碎碎念：

方法2和方法3的时间复杂度都是O(n)，但方法2扫了两遍数组，方法3扫了一遍，但多次提交都是方法2时间短。

# Middle-题目29：59. Spiral Matrix II

## 题目原文：

Given an integer *n*, generate a square matrix filled with elements from 1 to *n*2 in spiral order.

## 题目大意：

给一个整数n，输出n\*n的方阵，其中1~n2以顺时针的螺旋方式填充。

## 题目分析：

就是简单的模拟，从1开始往里填，注意拐弯的临界条件（走到数组的边缘或者按原方向的下一位已经填好数）。

## 源码：（language：java）

public class Solution {

private final int RIGHT = 1;

private final int DOWN = 2;

private final int LEFT = 3;

private final int UP = 4;

public int[][] generateMatrix(int n) {

int direction = 1;//1,2,3,4 stand for RIGHT,DOWN,LEFT,UP respectively.

int row=0,col=0;//current position

int[][] matrix = new int[n][n];

for(int i=1;i<=n\*n;i++) {

matrix[row][col] = i;

switch (direction) {

case RIGHT: {

if(col+1 == n || matrix[row][col+1] != 0) {

direction=DOWN;

row++;

}

else

col++;

break;

}

case DOWN: {

if(row+1 == n || matrix[row+1][col] != 0) {

direction=LEFT;

col--;

}

else

row++;

break;

}

case LEFT: {

if(col == 0 || matrix[row][col-1] != 0) {

direction=UP;

row--;

}

else

col--;

break;

}

case UP: {

if(matrix[row-1][col] != 0) {

direction=RIGHT;

col++;

}

else

row--;

}

}

}

return matrix;

}

}

## 成绩：

0ms,beats 14.13%,众数0ms，85.87%

# Middle-题目30：48. Rotate Image

## 题目原文：

You are given an *n* x *n* 2D matrix representing an image.

Rotate the image by 90 degrees (clockwise).

## 题目大意：

把一个n\*n的方阵顺时针旋转90度。

## 题目分析：

开一个新的矩阵，原来的第一行变成第n列，第2行变成第n-1列…..然后再复制回去。

## 源码：（language：java）

public class Solution {

public void rotate(int[][] matrix) {

int n=matrix.length;

int[][] newMatrix=new int[n][n];

for(int i=0;i<n;i++)

for(int j=0;j<n;j++)

newMatrix[j][n-1-i]=matrix[i][j];

for(int i=0;i<n;i++)

for(int j=0;j<n;j++)

matrix[i][j]=newMatrix[i][j];

}

}

## 成绩：

0ms，beats 25.09%，众数0ms，74.91%

## Cmershen的碎碎念：

这题的要求本来是不开辟新的数组，那么我想到的办法是一圈一圈的转，但代码还没写好。第一轮刷题时间比较仓促，只考虑AC。

# Middle-题目31：11. Container With Most Water

## 题目原文：

Given *n* non-negative integers *a1*, *a2*, ..., *an*, where each represents a point at coordinate (*i*, *ai*). *n* vertical lines are drawn such that the two endpoints of line *i* is at (*i*, *ai*) and (*i*, 0). Find two lines, which together with x-axis forms a container, such that the container contains the most water.

## 题目大意：

给出一个数组{a1,a2,…an},其中第i个数代表平面坐标系上点(i,ai),那么求两个点，使得这两个点向x轴作出的两条垂线和x轴围成的“杯子”能容纳最多的水。

## 题目分析：

还是一个two pointers的问题。

首先，对两个点（i,ai）和(j,aj)构成的“杯子”能装的水量用代数式表示为:min(ai,aj)\*(j-i).于是问题归结为求此表达式的最大值。那么如何搜索呢？

令两个指针i和j分别指向数组的头和尾，观察两个“杯壁”，若j端较高，则如果存在更优的解，那么更优解的i值一定不是当前值（因为如果j向左找，不管j端有多高，受i端影响，杯子的容量不可能更高），所以j端较高时令i++，同理i端较高时令j--，并不断更新最大水量，当搜索到i=j时整个数组遍历完毕，返回最大值。

## 源码：（language：java）

public class Solution {

public int maxArea(int[] height) {

int i=0;

int j=height.length-1;

int max=0;

while(i<j) {

int area=Math.min(height[i],height[j])\*(j-i);

if(area>max)

max=area;

if(height[i]<height[j])

i++;

else

j--;

}

return max;

}

}

## 成绩：

6ms,beats 57.22%,众数7ms,33.57%

## Cmershen的碎碎念：

此题的朴素解法就是O(n2)的暴力搜索，提交会超时。但似乎此题还有更快的方法。

# Middle-题目32：240. Search a 2D Matrix II

## 题目原文：

Write an efficient algorithm that searches for a value in an m x n matrix. This matrix has the following properties:

Integers in each row are sorted in ascending from left to right.

Integers in each column are sorted in ascending from top to bottom.

For example,

[

[1, 4, 7, 11, 15],

[2, 5, 8, 12, 19],

[3, 6, 9, 16, 22],

[10, 13, 14, 17, 24],

[18, 21, 23, 26, 30]

]

## 题目大意：

在一个满足下列性质的矩阵中寻找一个数：

每一行从左到右，每一列从上到下的数组是递增顺序的。

## 题目分析：

初始化i=0,j=n-1,即从矩阵的右上角开始找。若当前数比target小，则i++（因为只有下面才有更大的数，这样直接排除掉第i行）,若当前数比target大，则j--（因为只有左面才有更小的数，这样直接排除掉第j列）,如果找到边缘还没找到，则返回false。

## 源码：（language：java）

public class Solution {

public boolean searchMatrix(int[][] matrix, int target) {

if(matrix.length==0)

return false;

int rows=matrix.length,cols=matrix[0].length;

int i=0,j=cols-1;

while(j>=0 && i<=rows-1) {

if(matrix[i][j] == target)

return true;

else if(matrix[i][j] < target)

i++;

else

j--;

}

return false;

}

}

## 成绩：

13ms,beats 56.57%,众数13ms,37.38%

## Cmershen的碎碎念：

这道题很经典，传说是MSRA的面试题，因为每次都排除一行（列），而(i,j)点始终位于新矩阵的右上角，故可以看做是一个分治算法，算法很巧妙值得借鉴。

# Middle-题目33：300. Longest Increasing Subsequence

## 题目原文：

Given an unsorted array of integers, find the length of longest increasing subsequence.

For example,

Given [10, 9, 2, 5, 3, 7, 101, 18],

The longest increasing subsequence is [2, 3, 7, 101], therefore the length is 4. Note that there may be more than one LIS combination, it is only necessary for you to return the length.

Your algorithm should run in O(n2) complexity.

Follow up: Could you improve it to O(n log n) time complexity?

## 题目大意：

给出一个数组，求最长递增子序列的长度。

你的算法必须达到O(n2)的复杂度。进一步，你能提高到O(nlogn)吗？

## 题目分析：

1. 朴素解法：暴力分析每一个子列，看看是不是递增的。复杂度是O(2^n),忽略。
2. O（n2）解法（DP）：

记dp[i]为以i为结尾的最长递增子列长度，则有如下转移方程：

dp[i] = max(dp[j])+1,其中j<i,且nums[j]<nums[i]

因为计算dp[i]要遍历所有i前面的数，故复杂度O(n2)

1. O（nlogn）解法：再增加一个辅助数组B，记B[dp[j]]=a[j].即存储长度为dp[j]的子序列的最后一个元素。这样在从前面的dp[0]~dp[n-1]推dp[n]的时候，可以使用二分查找找到满足j<i且B[dp[j]]=a[j]<a[i]的最大j值，并置B[dp[j]+1]=a[i].（算法思路和源码摘自<http://www.cnblogs.com/lonelycatcher/archive/2011/07/28/2119123.html>）

## 源码：（language：java）

O(n2)解法：

public class Solution {

public int lengthOfLIS(int[] nums) {

if(nums.length<=1)

return nums.length;

int[] dp=new int[nums.length];

dp[0]=1;

int maxLIS=1;

for(int i=1;i<nums.length;i++) {

int maxj=0;

for(int j=0;j<i;j++) {

if(dp[j]>maxj && nums[j]<nums[i])

maxj=dp[j];

}

dp[i]=maxj+1;

if(dp[i]>maxLIS)

maxLIS=dp[i];

}

return maxLIS;

}

}

O(nlogn)解法：

public class Solution {

public int lengthOfLIS(int[] nums) {

int n = nums.length;

if(n==0)

return 0;

int[] B = new int[n+1];//数组B；

B[0]=-10000;//把B[0]设为最小，假设任何输入都大于-10000；

B[1]=nums[0];//初始时，最大递增子序列长度为1的最末元素为a1

int Len = 1;//Len为当前最大递增子序列长度，初始化为1；

int p,r,m;//p,r,m分别为二分查找的上界，下界和中点；

for(int i = 1;i<n;i++) {

p=0;r=Len;

while(p<=r)//二分查找最末元素小于ai+1的长度最大的最大递增子序列；

{

m = (p+r)/2;

if(B[m]<nums[i]) p = m+1;

else r = m-1;

}

B[p] = nums[i];//将长度为p的最大递增子序列的当前最末元素置为ai+1;

if(p>Len) Len++;//更新当前最大递增子序列长度；

}

return Len;

}

}

## 成绩：

O(n2)解法：42ms，beats 15.71%，众数1ms，13.76%

O(nlogn)解法：2ms，beats 78.15%

## Cmershen的碎碎念：

最长递增子序列问题是一个十分十分十分经典的DP问题，也是各公司经常用来面试的热门题，关于这个问题的详细讨论可见[joylnwang大神的博客](http://blog.csdn.net/joylnwang/article/details/6766317)，这篇文章中这个问题进行了比较深入的研究。

# Middle-题目34：289. Game of Life

## 题目原文：

According to the [Wikipedia's article](https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life): "The Game of Life, also known simply as Life, is a cellular automaton devised by the British mathematician John Horton Conway in 1970."

Given a board with m by n cells, each cell has an initial state live (1) or dead (0). Each cell interacts with its [eight neighbors](https://en.wikipedia.org/wiki/Moore_neighborhood) (horizontal, vertical, diagonal) using the following four rules (taken from the above Wikipedia article):

1. Any live cell with fewer than two live neighbors dies, as if caused by under-population.
2. Any live cell with two or three live neighbors lives on to the next generation.
3. Any live cell with more than three live neighbors dies, as if by over-population..
4. Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

Write a function to compute the next state (after one update) of the board given its current state.

## 题目大意：

给一个m\*n的网格，每个网格中有一个细胞，它只有生（1）或死（0）两种状态。每个细胞有8个邻居（水平，垂直，对角线），每经过一个时间周期，按如下规则演化：

1. 若一个活着的细胞周围有少于2个活着的邻居，则它会死于孤独。
2. 若一个活着的细胞周围有2,3个活着的邻居，则它会继续活下去。
3. 若一个活着的细胞周围有3个以上活着的邻居，则它会死于人口过多。
4. 若一个死亡的细胞周围恰有3个活着的细胞，则它会复活，就好像繁殖。

给出一个网格当前的状态，求出下一个时间周期的状态。

## 题目分析：

直接模拟即可，在数邻居个数的时候要考虑一些边界情况（因为不一定都有8个邻居），数出来邻居之后按规则变化即可。

## 源码：（language：java）

public class Solution {

private final int DEAD = 0;

private final int ALIVE = 1;

public void gameOfLife(int[][] board) {

int m=board.length;

int n=board[0].length;

int[][] nextState = new int[m][n];

for(int i=0;i<m;i++) {

for(int j=0;j<n;j++) {

//Rule 1:Any live cell with fewer than two live neighbors dies, as if caused by under-population.

if(board[i][j] == ALIVE && getNeighbours(board,i,j) <= 1)

nextState[i][j] = DEAD;

//Rule 3:Any live cell with more than three live neighbors dies, as if by over-population.

else if(board[i][j] == ALIVE && getNeighbours(board,i,j) > 3)

nextState[i][j] = DEAD;

//Rule 4:Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

else if(board[i][j] == DEAD && getNeighbours(board,i,j) == 3)

nextState[i][j] = ALIVE;

else

nextState[i][j]=board[i][j];

//System.out.println("i="+i+",j="+j+",neighbours="+getNeighbours(board,i,j,m,n)+",this state="+board[i][j]+",nextState="+nextState[i][j]);

}

}

for(int i=0;i<m;i++) {

for(int j=0;j<n;j++) {

board[i][j]=nextState[i][j];

}

}

}

private int getNeighbours(int[][] board,int i,int j) {

return getState(board,i-1,j-1) + getState(board,i-1,j) + getState(board,i-1,j+1) + getState(board,i,j-1) + getState(board,i,j+1) + getState(board,i+1,j-1) + getState(board,i+1,j) + getState(board,i+1,j+1);

}

private int getState(int[][] board, int i, int j) {

int m = board.length;

int n = board[0].length;

if(i<0 || j<0 || i==m || j==n)

return 0;

else

return board[i][j];

}

}

## 成绩：

1ms,beats 13.28%,众数1ms,86.47%

## Cmershen的碎碎念：

Discuss中还有一个神算法，有待研读。

# Middle-题目35：77. Combinations

## 题目原文：

Given two integers n and k, return all possible combinations of k numbers out of 1 ... n.

For example,

If n = 4 and k = 2, a solution is:

[

[2,4],

[3,4],

[2,3],

[1,2],

[1,3],

[1,4],

]

## 题目大意：

给出两个正整数n和k，求从1~n中取k个数的所有组合方式。

## 题目分析：

还是一个回溯法，用backtrack(List<List<Integer>> list, List<Integer> sublist, int n, int k, int start,int totaln)函数维护搜索过程，list代表最终返回结果，sublist记录当前搜索路径，n记录当前还有几个数要搜索，k记录当前还要取几个数，start记录当前要搜索的第一个数，totaln一直维持问题最开始的n。

这样递归关系是，每找到一个数字i, n=totaln-i(因为继续搜索的时候只需从i+1开始找)，k=k-1,start=i+1.

一直找到k=0的时候搜索结束。

## 源码：（language：java）

public class Solution {

public static List<List<Integer>> combine(int n, int k) {

List<List<Integer>> list = new ArrayList<List<Integer>>();

backtrack(list,new ArrayList<Integer>(),n,k,1,n);

return list;

}

private static void backtrack(List<List<Integer>> list, List<Integer> sublist, int n, int k, int start,int totaln) {

if(k == 0)

list.add(new ArrayList<Integer>(sublist));

else {

for(int i = start;i < start + n;i++) {

sublist.add(i);

backtrack(list,sublist,totaln-i,k-1,i+1,totaln);

sublist.remove(sublist.size()-1);

}

}

}

}

## 成绩：

3ms,beats 50.28%,众数3ms,37.00%

## cmershen的碎碎念：

本题似乎还可以简化一个变量。

# Middle-题目36：74. Search a 2D Matrix

## 题目原文：

Write an efficient algorithm that searches for a value in an m x n matrix. This matrix has the following properties:

Integers in each row are sorted from left to right.

The first integer of each row is greater than the last integer of the previous row.

For example,

Consider the following matrix:

[

[1, 3, 5, 7],

[10, 11, 16, 20],

[23, 30, 34, 50]

]

## 题目大意：

在一个满足下列性质的矩阵中寻找一个数：

每一行从左到右，每一列从上到下的数组是递增顺序的。且下一行最小的数比上一行最大的数大。

## 题目分析：

方法一：既然做过了[Middle-题目32](#_Middle-题目32：240._Search_a)，就可以用同样的算法。

方法二：因为在第32题的基础上又加一个限定，所以对每行第一个数二分查找，确定该数可能存在的行，再在行内二分查找。

## 源码：（language：java）

方法一略。

方法二：

public class Solution {

public boolean searchMatrix(int[][] matrix, int target) {

int start = 0,end = matrix.length - 1,mid;

if(start == end) // the matrix has only 1 line

return binarySeach(matrix[0],target);

while(start <= end) {

mid = (start + end) / 2;

if(matrix[mid][0] == target)

return true;

else if(matrix[mid][0] < target && (mid == matrix.length-1 || matrix[mid+1][0] > target)) // target may be at line of mid

return binarySeach(matrix[mid],target);

else if(matrix[mid+1][0] <= target) //target lies after line of mid

start = mid + 1;

else

end = mid-1;

}

return false;

}

private boolean binarySeach(int[] nums, int target) {

int start = 0, end = nums.length-1, mid;

while(start <= end) {

mid = (start + end) / 2;

if(nums[mid] == target)

return true;

else if(nums[mid] < target)

start = mid + 1;

else

end = mid - 1;

}

return false;

}

}

## 成绩：

方法一：2ms，beats 3.01%，众数1ms,89.18%

方法二：1ms，beats 6.15%

# Middle-题目37：199. Binary Tree Right Side View

## 题目原文：

Given a binary tree, imagine yourself standing on the right side of it, return the values of the nodes you can see ordered from top to bottom.

For example:

Given the following binary tree,

1 <---

/ \

2 3 <---

\ \

5 4 <---

You should return [1, 3, 4].

## 题目大意：

给出一个二叉树，假设你的视线从右边看过去，返回从上到下你看到的节点（不在最右侧的都被挡上了）。

## 题目分析：

按层次遍历，求每层最后一个节点即可（即队列中改变层号的节点），在[Easy-题目27](#_Easy-题目27：107.Binary_Tree_Level)基础上修改代码即可。

## 源码：（language：java）

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

public class Solution {

public List<Integer> rightSideView(TreeNode root) {

Queue<TreeNode> queue=new LinkedList<TreeNode>();

Queue<Integer> levelqueue=new LinkedList<Integer>();

List<Integer> list=new ArrayList<Integer>();

if(root == null)

return list;

queue.add(root);

levelqueue.add(1);

//list.add(root.val);

while(!queue.isEmpty()) {

TreeNode current = queue.remove();

int curLevel=levelqueue.remove();

if(current.left!=null) {

queue.add(current.left);

levelqueue.add(curLevel+1);

}

if(current.right!=null) {

queue.add(current.right);

levelqueue.add(curLevel+1);

}

if(levelqueue.isEmpty() == true || levelqueue.peek() == curLevel + 1)

list.add(current.val);

}

return list;

}

}

## 成绩：

3ms,beats 10.90%,众数3ms,40.54%

# Middle-题目38：241. Different Ways to Add Parentheses

## 题目原文：

Given a string of numbers and operators, return all possible results from computing all the different possible ways to group numbers and operators. The valid operators are +, - and \*.

Example 1

Input: "2-1-1".

((2-1)-1) = 0

(2-(1-1)) = 2

Output: [0, 2]

Example 2

Input: "2\*3-4\*5"

(2\*(3-(4\*5))) = -34

((2\*3)-(4\*5)) = -14

((2\*(3-4))\*5) = -10

(2\*((3-4)\*5)) = -10

(((2\*3)-4)\*5) = 10

Output: [-34, -14, -10, -10, 10]

## 题目大意：

给出一个算数表达式字符串（有数字和+-\*三种运算符），求出用不同种类的加括号方式能算得的数分别是什么。（允许重复）

## 题目分析：

典型的分治算法。如果字符串只是一个数，则只有1种情况，否则找到其中的运算符，分割成左右两个表达式串，递归求算得的数的序列(分别记为leftList和rightList)，再分别遍历leftList和rightList的每个数，加入在当前运算符下算得的结果即可。

## 源码：（language：java）

public class Solution {

public List<Integer> diffWaysToCompute(String input) {

List<Integer> list=new ArrayList<Integer>();

if(isNumeric(input)) // if input is an integer

list.add(Integer.parseInt(input));

else {

for(int i=0;i<input.length();i++) {

char op=input.charAt(i);

if(op == '+' || op == '-' || op == '\*') {

List<Integer> leftList = diffWaysToCompute(input.substring(0,i));

List<Integer> rightList = diffWaysToCompute(input.substring(i+1));

for(Integer j : leftList)

for(Integer k : rightList)

list.add(operate(j,op,k));

}

}

}

return list;

}

private Integer operate(int a, char op, int b) {

if(op=='+')

return a+b;

else if(op=='-')

return a-b;

else

return a\*b;

}

public boolean isNumeric(String str){

for(int i=str.length();--i>=0;){

int chr=str.charAt(i);

if(chr<48 || chr>57)

return false;

}

return true;

}

}

## 成绩：

9ms,beats 27.25%,众数8ms,21.40%

## cmershen的碎碎念：

根据discuss，本题似乎用HashMap可以提高一点效率。但基本思路不变。

# Middle-题目39：73. Set Matrix Zeroes

## 题目原文：

Given a m x n matrix, if an element is 0, set its entire row and column to 0. Do it in place.

## 题目大意：

给出一个m\*n的矩阵，若一个元素是0，置它所在的整行和整列都为0.

## 题目分析：

时间复杂度是O(mn)无疑，本题考查的是空间复杂度。

方法一：O(m+n)的空间：使用两个数组isRowsZero和isColsZero分别记录每一行和每一列是否需要置0.

方法二：O(1)的空间：使用第一行和第一列记录该行和该列是否应该置零，并使用两个变量rol0和col0记录首行和首列是否需要置0（因为有可能把不是0的信息变成0）

## 源码：（language：java/c）

方法一：

public class Solution {

public void setZeroes(int[][] matrix) {

int rows=matrix.length;

int cols=matrix[0].length;

boolean[] isZeroRow = new boolean[rows];

boolean[] isZeroCol = new boolean[cols];

for(int i = 0;i <rows;i++) {

for(int j = 0;j < cols;j++) {

if(matrix[i][j]==0) {

isZeroRow[i]=true;

isZeroCol[j]=true;

}

}

}

for(int i = 0;i<rows;i++) {

if(isZeroRow[i]) {

for(int j=0;j<cols;j++) {

matrix[i][j]=0;

}

}

}

for(int j=0;j<cols;j++) {

if(isZeroCol[j]) {

for(int i=0;i<rows;i++) {

matrix[i][j]=0;

}

}

}

}

}

方法二：

void setZeroes(int\*\* matrix, int matrixRowSize, int matrixColSize) {

int col0 = 0;

int row0 = 0;

for(int i = 0; i < matrixRowSize; i++) {

if(matrix[i][0] == 0) {

col0 = 1;

break;

}

}

for(int j = 0; j < matrixColSize; j++) {

if(matrix[0][j] == 0) {

row0 = 1;

break;

}

}

for(int i = 1;i < matrixRowSize; i++) {

for(int j = 1;j < matrixColSize; j++) {

if(matrix[i][j] == 0) {

matrix[i][0] = 0;

matrix[0][j] = 0;

}

}

}

for(int i = 1;i < matrixRowSize;i++) {

if(!matrix[i][0]) {

for(int j = 1;j < matrixColSize;j++) {

matrix[i][j]=0;

}

}

}

for(int j = 1;j < matrixColSize;j++) {

if(!matrix[0][j]) {

for(int i = 1;i < matrixRowSize;i++) {

matrix[i][j]=0;

}

}

}

if(col0) {

for(int i = 0; i < matrixRowSize; i++)

matrix[i][0] = 0;

}

if(row0) {

for(int j = 0; j < matrixColSize; j++)

matrix[0][j] = 0;

}

}

## 成绩：

方法一：2ms,beats 19.41%，众数2ms,19.41%

方法二：44ms,beats 21.43%, 众数44ms,69.05%

## cmershen的碎碎念：

空间复杂度是leetcode上很多题的要求，应该也是面试时需要重点注意的一项。

# Middle-题目40：162. Find Peak Element

## 题目原文：

A peak element is an element that is greater than its neighbors.

Given an input array where num[i] ≠ num[i+1], find a peak element and return its index.

The array may contain multiple peaks, in that case return the index to any one of the peaks is fine.

You may imagine that num[-1] = num[n] = -∞.

For example, in array [1, 2, 3, 1], 3 is a peak element and your function should return the index number 2.

## 题目大意：

寻找一个数组的峰值元素（类似于函数中的极大值）

假设num[-1]=num[n]=- ∞。

若数组中有多个峰值元素，返回任意一个都可以。

## 题目分析：

本题依然是一个二分查找的变形。先考虑边界情况，若数组只有1个元素，则他就是峰值；若数组只有两个元素则较大的就是峰值元素。

接下来进行二分查找，设中点为mid，考虑以下情况：

1. mid是端点，那么只看它是否大于第二个点（倒数第二个点）即可；
2. mid点的值大于左边又大于右边，则它就是峰值，返回即可；
3. mid点的“邻域”是递增的，则右半边一定有峰值；
4. mid点的“邻域”是递减的，则左半边一定有峰值；

## 源码：（language：c）

int findPeakElement(int\* nums, int numsSize) {

if(numsSize == 1)

return 0;

else if(numsSize == 2)

return nums[0]<nums[1];

else {

int mid=(numsSize-1)/2;

if(mid == 0 && nums[0]>nums[1])

return 0;

else if(mid == numsSize-1 && nums[numsSize-1] > nums[numsSize-2])

return numsSize-1;

else if(nums[mid] > nums[mid-1] && nums[mid] > nums[mid+1])

return mid;

else if(nums[mid] >= nums[mid-1] && nums[mid] <= nums[mid+1])

return findPeakElement(&nums[mid+1],numsSize-mid-1)+mid+1;

else

return findPeakElement(nums,numsSize-mid-1);

}

}

## 成绩：

0ms,beats 9.38%,众数0ms,90.62%

## cmershen的碎碎念：

二分查找不一定只适用于有序的数组。像本题的数组是不知道顺序的，但适用二分查找。

# Middle-题目41：313. Super Ugly Number

## 题目原文：

Write a program to find the nth super ugly number.

Super ugly numbers are positive numbers whose all prime factors are in the given prime list primes of size k. For example, [1, 2, 4, 7, 8, 13, 14, 16, 19, 26, 28, 32] is the sequence of the first 12 super ugly numbers given primes = [2, 7, 13, 19] of size 4.

Note:

(1) 1 is a super ugly number for any given primes.

(2) The given numbers in primes are in ascending order.

(3) 0 < k ≤ 100, 0 < n ≤ 106, 0 < primes[i] < 1000.

## 题目大意：

写一个程序计算第n个超级丑数。

超级丑数如下定义：超级丑数是一种正数，它的所有质因子都在给出的序列中。

例如：给定质数序列2,7,13,19，则1, 2, 4, 7, 8, 13, 14, 16, 19, 26, 28, 32都是在这个质数序列下的丑数。

注意：

（1）1是任意质数序列下的丑数；

（2）质数序列是递增顺序的；

给出质数序列，求该序列下的第k个超级丑数。

## 题目分析：

观察例子中的丑数序列：

1

2

4=2\*2

7

8=4\*2

13

14=7\*2

16=8\*2

19

26=13\*2

28=14\*2

32=16\*2

可以发现每个丑数都可以看做是素数序列中的一个数与较小的丑数的乘积。

那么可以使用两个辅助数组index和vals，其中index记录当前每个素数应该与已有解中的哪个数相乘，vals记录候选解，每次从vals中筛选出最小解，并令对应下标的index加1.最后用数组q存储每轮迭代的最优解。这样直接描述可能有些难以理解，以下给出迭代过程。

算法迭代过程如下：（设质数序列primes = {2,7,13,19}）

初始化：

q={1} // 因为1永远是超级丑数

vals={0,0,0,0}

index={0,0,0,0} // 所有素数都和q中第0个元素相乘

第一轮筛选：

vals = {2,7,13,19}

选择2为最优解，加入数组q中，并更新index

index = {1,0,0,0} //素数2与q中第1个元素相乘，其他都与第0个元素相乘

q={1,2}

第二轮筛选：

vals = {4,7,13,19}

选择4为最优解，加入数组q中，并更新index：

index = {2,0,0,0} //素数2与q中第2个元素相乘，其他都与第0个元素相乘

q={1,2,4}

第三轮筛选：

vals = {8,7,13,19}

选择7为最优解，加入数组q中，并更新index：

index = {2,1,0,0} //素数2与q中第2个元素相乘，7与第1个元素相乘，其他都与第0个元素相乘

q={1,2,4,7}

第四轮筛选：

vals = {8,14,13,19}

选择8为最优解，加入数组q中，并更新index：

index = {3,1,0,0} //素数2与q中第2个元素相乘，7与第1个元素相乘，其他都与第0个元素相乘

q={1,2,4,7}

……

一直到第k轮，返回q中最后一个元素即可。

## 源码：（language：java）

public class Solution {

public int nthSuperUglyNumber(int n, int[] primes) {

int size = primes.length;

int q[] = new int[n];

int idxes[] = new int[size];

int vals[] = new int[size];

q[0] = 1;

for (int i = 1; i < n; i++) {

for (int j = 0; j < size; j++) {

vals[j] = q[idxes[j]] \* primes[j];

}

q[i] = findMin(vals);

for (int j = 0; j < size; j++) {

if (vals[j] == q[i]) {

idxes[j] += 1;

}

}

}

return q[n - 1];

}

public int findMin(int[] nums) {

int min = nums[0];

for (int i = 1; i < nums.length; i++) {

min = Math.min(min, nums[i]);

}

return min;

}

}

## 成绩：

39ms,beats 38.05%,众数32ms,6.33%

# Middle-题目42/43：274. H-Index && 275. H-Index II

## 题目原文：

274. H-Index

Given an array of citations (each citation is a non-negative integer) of a researcher, write a function to compute the researcher's h-index.

According to the definition of h-index on Wikipedia: "A scientist has index h if h of his/her N papers have at least h citations each, and the other N − h papers have no more than h citations each."

For example, given citations = [3, 0, 6, 1, 5], which means the researcher has 5 papers in total and each of them had received 3, 0, 6, 1, 5 citations respectively. Since the researcher has 3 papers with at least 3 citations each and the remaining two with no more than 3 citations each, his h-index is 3.

275. H-Index II

Follow up for H-Index: What if the citations array is sorted in ascending order? Could you optimize your algorithm?

## 题目大意：

274. H-Index

给出一个研究者的引用量数组，写一个算法计算这个研究者的h指数。

H指数的定义：一个人的h指数是指在一定期间内他发表的论文至少有h篇的被引频次不低于h次。

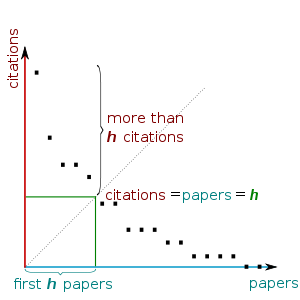
例如，给出citations数组=[3,0,6,1,5],代表这个研究者写了5篇论文，引用次数分别是3,0,6,1,5.那么他的h值是3,因为有3篇论文的引用次数>=3次。

275. H-Index II

若citations数组是升序排列的，能否优化算法？

## 题目分析：

根据wiki中给出的算法：



把引用数降序排列画在坐标系中，位于直线y=x上面的点个数就是h值。由于本题中引用数是降序排列的，所以改为统计位于直线x+y=n(n为论文数目)上方点的数目即可。因为数组是有序的，所以可以使用二分查找。

## 源码：（language：java）

public class Solution {

public int hIndex(int[] citations) {

int count=0,length = citations.length;

int start=0,end=length-1,mid=0;

while(start<=end) {

mid = (start+end) / 2;

if(citations[mid] >= length-mid) {

if(mid==0 || citations[mid-1]<length-mid+1)

return length-mid;

else if(citations[mid-1]>=length-mid)

end=mid-1;

}

else

start=mid+1;

}

return 0;

}

}

## 成绩：

H-Index:4ms,beats 9.28%,众数1ms,38.82%

H-Index II:13ms,beats 31.68%,众数12ms,24.72%

## Cmershen的碎碎念：

在第一题中，使用二分查找比无脑查找慢，但第二题中无脑插的耗时是213ms，远差于二分查找。

# Middle-题目44：334. Increasing Triplet Subsequence

## 题目原文：

Given an unsorted array return whether an increasing subsequence of length 3 exists or not in the array.

Formally the function should:

Return true if there exists i, j, k

such that arr[i] < arr[j] < arr[k] given 0 ≤ i < j < k ≤ n-1 else return false.

Your algorithm should run in O(n) time complexity and O(1) space complexity.

## 题目大意：

给出一个数组，判断是否存在一个递增的长度为3的子序列。

## 题目分析：

1. 朴素解法：O(n3)暴力搜索；
2. 最长递增子列：引用[Middle-题目33](#_Middle-题目33：300._Longest_Increasing)的方法判断最长递增子列的长度是否>=3，最好的时间复杂度是O(nlogn)。
3. O(n)算法：扫描一遍数组，令a1是当前最小值，a2是a1以后次小值，则如果当前的数比a2还大，就存在。

## 源码：（language：java）

public class Solution {

public boolean increasingTriplet(int[] nums) {

int a1 = Integer.MAX\_VALUE,a2 = Integer.MAX\_VALUE;

for(int num : nums) {

if(num<=a1)

a1=num;

else if(num<=a2)

a2=num;

else

return true;

}

return false;

}

}

## 成绩：

1ms,beats 34.32%,众数1ms,65.68%

# Middle-题目45：215. Kth Largest Element in an Array

## 题目原文：

Find the **k**th largest element in an unsorted array. Note that it is the kth largest element in the sorted order, not the kth distinct element.

## 题目大意：

寻找一个数组中第k大的数。

## 题目分析：

1. 朴素解法O(nlogn):排序，然后返回倒数第k个元素。
2. Quick Select算法O（n）:

S1.选择一个中轴（可以使用快排中的三者取中法），比中轴大的数放到左边，比中轴小的数放到右边；

S2.然后求出左边的长度l,若l==k,则中轴即为所求；若l>k,则从左边数组里面找第k大的数，若l<k，则从右边数组里面找第k-l个数。

最坏的情况每个数遍历一遍，复杂度O(n).

## 源码：（language：java）

方法一：

public class Solution {

public int findKthLargest(int[] nums, int k) {

Arrays.sort(nums);

return nums[nums.length - k];

}

}

方法二：（代码来自discuss）

public class Solution {

public int findKthLargest(int[] nums, int k) {

return select(nums, k-1);

}

private int select(int[] nums, int k) {

int left = 0, right = nums.length-1;

while(true) {

if(left == right)

return nums[left];

int pivotIndex = medianOf3(nums, left, right);

pivotIndex = partition(nums, left, right, pivotIndex);

if(pivotIndex == k)

return nums[k];

else if(pivotIndex > k)

right = pivotIndex-1;

else

left = pivotIndex+1;

}

}

//Use median-of-three strategy to choose pivot

private int medianOf3(int[] nums, int left, int right) {

int mid = left + (right - left) / 2;

if(nums[right] > nums[left])

swap(nums, left, right);

if(nums[right] > nums[mid])

swap(nums, right, mid);

if(nums[mid] > nums[left])

swap(nums,left, mid);

return mid;

}

private int partition(int[] nums, int left, int right, int pivotIndex) {

int pivotValue = nums[pivotIndex];

swap(nums, pivotIndex, right);

int index = left;

for(int i = left; i < right; ++i) {

if(nums[i] > pivotValue) {

swap(nums, index, i);

++index;

}

}

swap(nums, right, index);

return index;

}

private void swap(int[] nums, int a, int b) {

int temp = nums[a];

nums[a] = nums[b];

nums[b] = temp;

}

}

## 成绩：

方法一：7ms,beats 74.67%,众数4ms,13.67%

方法二：2ms,beats 97.12%

## Cmershen的碎碎念：

这道题好像是一道很经典的题，似乎在《算法导论》中有对这道题大篇幅的详细描述。

# Middle-题目46：129. Sum Root to Leaf Numbers

## 题目原文：

Given a binary tree containing digits from 0-9 only, each root-to-leaf path could represent a number.

An example is the root-to-leaf path 1->2->3 which represents the number 123.

Find the total sum of all root-to-leaf numbers.

For example,

1

/ \

2 3

The root-to-leaf path 1->2 represents the number 12.

The root-to-leaf path 1->3 represents the number 13.

Return the sum = 12 + 13 = 25.

## 题目大意：

有一个二叉树，节点是0-9构成的数字，这样从根节点到叶子节点的每一条路径构成一个数，计算所有路径上的数的和。

## 题目分析：

本题就是一个记忆化的前序遍历，一直记录着前面的数，每次向孩子搜索的时候把当前数×10加上当前节点的数，搜到叶子节点的时候加到当前的和里面。

## 源码：（language：java）

public class Solution {

private int currentTotal = 0;

public int sumNumbers(TreeNode root) {

dfs(root,0);

return currentTotal;

}

private void dfs(TreeNode node, int currentSum) {

if(node == null)

return;

else if(isLeaf(node)) {

currentSum+=node.val;

currentTotal+=currentSum;

}

else {

currentSum+=node.val;

dfs(node.left,currentSum\*10);

dfs(node.right,currentSum\*10);

}

}

private boolean isLeaf(TreeNode node) {

return node.left==null && node.right==null;

}

}

## 成绩：

1ms,beats 28.93%,众数1ms,60.87%

# Middle-题目47：80. Remove Duplicates from Sorted Array II

## 题目原文：

Follow up for "Remove Duplicates":

What if duplicates are allowed at most twice?

## 题目大意：

接[Easy-题目29](#_Easy-题目29：26._Remove_Duplicates)，这次允许一个元素最多出现两次。

## 题目分析：

和上题差不多，只是这次比较的两个指针是i和i-2，如果不相等则计数加一。因为这种情况下i-2所指元素最多只能加入两次.(若数组为….1,1,1,2,….则第一个1会跳过去，第二个和第三个1会加入)

## 源码：（language：c）

int removeDuplicates(int\* nums, int numsSize) {

int start = 0;

for(int i = 0;i < numsSize;i++) {

if(i == 0 || i == 1)

nums[start++] = nums[i];

else {

if(nums[i] != nums[start-2])

nums[start++] = nums[i];

}

}

return start;

}

## 成绩：

8ms，beats 11.11%,众数8ms,87.30%

## Cmershen的碎碎念：

本题还可以推广，若允许一个元素最多出现n次？是不是直接改为i和i-n两个指针即可？有待探索。

# Middle-题目48：81. Search in Rotated Sorted Array II

## 题目原文：

Follow up for "Search in Rotated Sorted Array":

What if duplicates are allowed?

Would this affect the run-time complexity? How and why?

Write a function to determine if a given target is in the array.

## 题目大意：

对一个移位过后的升序数组(数组元素可能重复)，判断一个数是否存在。

## 题目分析：

无脑找。O(n).

## 源码：（language：java）

public class Solution {

public boolean search(int[] nums, int target) {

for(int num:nums)

if(num==target)

return true;

return false;

}

}

## 成绩：

1ms,beats 21.80%,众数1ms,78.20%

## Cmershen的碎碎念：

本题本应该考察的是二分搜索，可测试用例没有体现比无脑找的优势，而更是有21.8%的提交代码（可能是试图使用二分搜索）反而慢了。可能此题的test case是有问题的。

# Middle-题目49：279. Perfect Squares

## 题目原文：

Given a positive integer n, find the least number of perfect square numbers (for example, 1, 4, 9, 16, ...) which sum to n.

## 题目大意：

给出正整数n，求出把n拆成最少的完全平方数之和的个数。

例如,12=4+4+4，拆出3个完全平方数，返回3.

13=4+9，最少拆出2个完全平方数，返回2.

## 题目分析：

方法一：（动态规划）记dp[i]为组成i的平方数的最小个数，则可以从dp[i]推到dp[i+j^2]:

dp[i+j^2]=min(dp[i+j^2],dp[i]+1),即i+j^2这个数可能有更优的拆分方式是拆出j^2,否则维持原来的拆分方式。

初始化：dp[1]=1(1只能拆出1),dp[i]=i(最坏的拆分方法是把每个数i都拆成i个1，然后进一步优化

迭代过程：

i=1时，更新2,5,10,17….dp(n^2+1)=2;此时dp[2]=2

i=2时，更新3,6,11,18….dp(n^2+2)=3;

……

外层循环从1到n，内层循环从i到根号n，故总复杂度O(nlogn).

方法二：（拉格朗日四平方和定理）根据Lagrange四平方定理，任何一个正整数都可以表示成不超过四个整数的平方之和。这样只需考虑什么时候返回1,2,3,4即可。

关于lagrange四平方定理的证明见[这篇论文](http://www.changhai.org/articles/science/mathematics/four_square_theorem.php)。

1. 若一个数n是4的倍数，则一直除以4化简，道理很简单，因为若k可以拆成k1+k2+k3+k4(其中ki是完全平方数或0)，则4k=4k1+4k2+4k3+4k4，其中4ki显然也是完全平方数。
2. 如果一个数除以8余7，则一定是由4个完全平方数组成。证明见碎碎念部分。
3. 接下来就剩下1,2,3三种情况了，暴力搜索一下1和2的情况，所以搜索{i∈Z|i∈[0,sqrt(n)]}，如果找到能开出j-i的数，则返回1或2（如果i=0或根号n，则是1，否则是2.
4. 若找不到，则一定是3.

因为这里的暴力搜索是从0到根号n的，所以算法复杂度是O(logn)!!!!!!

## 源码：（language：java）

方法一：

public class Solution {

public int numSquares(int n) {

int[] dp = new int[n+1];

for(int i = 0;i<=n;i++)

dp[i]=i;

for(int i = 1;i\*i <= n;i++)

dp[i\*i]=1;

for(int i=1;i<=n;i++) {

for(int j=1;i+j\*j<=n;j++) {

dp[i+j\*j]=Math.min(dp[i+j\*j], dp[i]+1);

}

}

return dp[n];

}

}

方法二：

public class Solution {

public int numSquares(int n) {

while(n%4 == 0)

n/=4;

if(n%8 == 7)

return 4;

for (int a=0; a\*a<=n; ++a) {

int b = (int)Math.sqrt(n - a\*a);

if (a\*a + b\*b == n)

return func(a) + func(b);

}

return 3;

}

private int func(int a) {

return a==0?0:1;

}

}

## 成绩：

方法一：66ms,beats 72.43%,众数68ms,6.46%

方法二：4ms,beats 97.67%

## Cmershen的碎碎念：

由此可见数学是科学之母，然而我不是学数学的，那些定理的证明对我来说如同天书……还有人说，@jianchao.li.fighter这个作者出的题都跟数学有关系。

2016.3.11补充步骤2的证明部分：

首先根据四平方数定理，任何正整数都可以拆成不超过4个完全平方数的和。

显然，任何完全平方数n2要么是4的倍数，要么被4除余1.（这个证明略）

假设存在正整数a1,a2使得,那么a1和a2必然一个是偶数一个是奇数，则可表示为4s+4t+1(s,t),是模4余1的，与模8余7矛盾。

假设存在正整数a1,a2,a3使得,若有一个数是奇数，另外两个数是偶数，同上易得平方和是模4余1的， 若三个数都是奇数

# Middle-题目50：142. Linked List Cycle II

## 题目原文：

Given a linked list, return the node where the cycle begins. If there is no cycle, return null.

## 题目大意：

判断一个单链表有没有环。如果有环，返回环的起点，没有环返回null

## 题目分析：

还是用HashSet存，第一个重复的节点就是环的起点。

## 源码：（language：java）

/\*\*

\* Definition for singly-linked list.

\* class ListNode {

\* int val;

\* ListNode next;

\* ListNode(int x) {

\* val = x;

\* next = null;

\* }

\* }

\*/

public class Solution {

public ListNode detectCycle(ListNode head) {

HashSet<ListNode> set = new HashSet<ListNode>();

for(ListNode node=head;node!=null;node=node.next) {

if(!set.add(node))

return node;

}

return null;

}

}

## 成绩：

12ms,beats 5.82%，众数1ms,81.12%

# Middle-题目51：331. Verify Preorder Serialization of a Binary Tree

## 题目原文：

One way to serialize a binary tree is to use pre-order traversal. When we encounter a non-null node, we record the node's value. If it is a null node, we record using a sentinel value such as #.

\_9\_

/ \

3 2

/ \ / \

4 1 # 6

/ \ / \ / \

# # # # # #

For example, the above binary tree can be serialized to the string "9,3,4,#,#,1,#,#,2,#,6,#,#", where # represents a null node.

Given a string of comma separated values, verify whether it is a correct preorder traversal serialization of a binary tree. Find an algorithm without reconstructing the tree.

Each comma separated value in the string must be either an integer or a character '#' representing null pointer.

## 题目大意：

给出一个用逗号分隔的字符串，其中数字代表节点的值，#号代表空节点，判断是否是一个合法的二叉树的前序遍历序列。

## 题目分析：

方法一（朴素解法）：用堆栈，每次遇到一个数字或者#号都入栈，如果栈顶的前三个元素依次是#,#,数字，则把它们三个都弹出来，再把#号压入栈（代表去掉了一个叶子节点），如此往复若最后栈里只有一个#（所有节点遍历完毕），则是合法的。

方法二：从discuss中看到的一个神算法，好像机理是一样的，只是没有用到堆栈，大致意思是，初始化一个变量count从后向前搜索，遇到#号就+1，遇到数字就-1，看最后是不是1，如果中途不够减则返回false。此算法具体思路及验证正确性，还有待进一步研究。

## 源码：（language：java）

方法一：

public class Solution {

public boolean isValidSerialization(String preorder) {

String[] nodes = preorder.split(",");

Stack<String> stack = new Stack<String>();

for(String str : nodes) {

stack.push(str);

int size = stack.size();

while(size > 2 && stack.get(size-1).equals("#") && stack.get(size-2).equals("#") && !stack.get(size-3).equals("#")) {

stack.pop();

stack.pop();

stack.pop();

stack.push("#");

size = stack.size();

}

}

return stack.size() == 1 && stack.peek().equals("#");

}

}

方法二：

public class Solution {

public boolean isValidSerialization(String preorder) {

int len = preorder.length();

int count = 0;

for(int i=len-1; i>=0; i--){

char tmp = preorder.charAt(i);

if(tmp == ','){

continue;

}else if(tmp == '#'){

count++;

}else if(tmp != ',' && tmp != '#' && i!=0 && preorder.charAt(i-1)!=','){

continue;

}else{

if(count<2){

return false;

}else{

count--;

}

}

}

return count==1;

}

}

## 成绩：

方法一：25ms,beats 10.82%,众数11ms,22.68%

方法二：4ms,beats 98.75%

## Cmershen的碎碎念：

其实方法一浪费时间主要是在基于正则表达式的split函数和STL中堆栈函数，而方法二直接对数组操作，但似乎本质是一样的（因为方法二中的count也是统计的#号个数）。

# Middle-题目52：78. Subsets

## 题目原文：

Given a set of distinct integers, nums, return all possible subsets.

Note:

Elements in a subset must be in non-descending order.

The solution set must not contain duplicate subsets.

## 题目大意：

给出一个无重复元素的数组，求出他的所有子集。

要求：

子集中的元素不能是递减的，且结果中不能有重复的子集。

## 题目分析：

既然是不重复的那就好办了，求数组的长度n，并遍历一个n位长的比特串，其中哪一位为1，原数组的哪个元素就加入子集。

## 源码：（language：java）

public class Solution {

public List<List<Integer>> subsets(int[] nums) {

Arrays.sort(nums);

List<List<Integer>> subsets = new ArrayList<List<Integer>>();

int count = 1<<nums.length;

for(int j=0;j<count;j++) {

List<Integer> subset = new ArrayList<Integer>();

int index=0;

int i=j;

while(i!=0) {

if(i%2==1)

subset.add(nums[index]);

index++;

i=i>>1;

}

subsets.add(subset);

}

return subsets;

}

}

## 成绩：

2ms，beats 59.76%,众数3ms,44.63%

## Cmershen的碎碎念：

计算机是基于二进制运算的，所以能用bit manipulate方法解决的问题往往都比较方便。但此题复杂度达到O(2^n),使用位运算比单纯的回溯也仅仅是易于理解。

# Middle-题目53：114. Flatten Binary Tree to Linked List

## 题目原文：

Given a binary tree, flatten it to a linked list in-place.

For example,

Given

1

/ \

2 5

/ \ \

3 4 6

The flattened tree should look like:

1

\

2

\

3

\

4

\

5

\

6

## 题目大意：

把一个二叉树平铺到右子树上。

## 题目分析：

先递归平铺左右子树，再把左子树接到根节点右孩子上即可。

## 源码：（language：java）

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

public class Solution {

public void flatten(TreeNode root) {

if(root==null){

return;

}

flatten(root.left);

flatten(root.right);

TreeNode ptr = root.left;

if(ptr!=null){

while(ptr.right!=null){

ptr = ptr.right;

}

ptr.right = root.right;

root.right = root.left;

root.left = null;

}

}

}

## 成绩：

1ms,beats 34.42%,众数1ms,65.58%

# Middle-题目54：39. Combination Sum

## 题目原文：

Given a set of candidate numbers (C) and a target number (T), find all unique combinations in C where the candidate numbers sums to T.

The same repeated number may be chosen from C unlimited number of times.

Note:

All numbers (including target) will be positive integers.

Elements in a combination (a1, a2, … , ak) must be in non-descending order. (ie, a1 ≤ a2 ≤ … ≤ ak).

The solution set must not contain duplicate combinations.

For example, given candidate set 2,3,6,7 and target 7,

A solution set is:

[7]

[2, 2, 3]

## 题目大意：

给出一系列候选数，及一个目标数，计算目标数拆分成候选数之和（允许重复）的所有拆分方式。

## 题目分析：

贪心+回溯。使用backtrack(List<List<Integer>> list, List<Integer> sublist, int[] candidates, int target, int startPos)维护搜索过程，参数意义不必过多介绍，每次加入一个候选数，就从当前候选数开始往下搜。

## 源码：（language：java）

public class Solution {

public List<List<Integer>> combinationSum(int[] candidates, int target) {

Arrays.sort(candidates);

List<List<Integer>> list = new ArrayList<List<Integer>>();

backtrack(list, new ArrayList<Integer>(), candidates, target, 0);

return list;

}

private void backtrack(List<List<Integer>> list, List<Integer> sublist, int[] candidates, int target, int startPos) {

if(target == 0) {

list.add(new ArrayList<Integer>(sublist));

}

else {

for(int i = startPos; i < candidates.length; i++) {

int temp = candidates[i];

if(temp <= target) {

sublist.add(temp);

backtrack(list,sublist,candidates,target-temp,i);

sublist.remove(sublist.size()-1);

}

}

}

}

}

## 成绩：

6ms,beats 64.44%,众数5ms,24.50%

# Middle-题目55：329. Longest Increasing Path in a Matrix

## 题目原文：

Given an integer matrix, find the length of the longest increasing path.

From each cell, you can either move to four directions: left, right, up or down. You may NOT move diagonally or move outside of the boundary (i.e. wrap-around is not allowed).

Example 1:

nums = [

[9,9,4],

[6,6,8],

[2,1,1]

]

Return 4

The longest increasing path is [1, 2, 6, 9].

## 题目大意：

给出一个矩阵，寻找最长的升序路径。只能沿上下左右四个方向运动。

## 题目分析：

从每个节点开始深搜矩阵，保持着搜索路径即可。每个dfs函数都返回从当前点出发的最大长度。

## 源码：（language：java）

public class Solution {

int max = 1;

public int longestIncreasingPath(int[][] matrix) {

if (matrix==null || matrix.length==0 || matrix[0].length==0)

return 0;

int[][] lenBoard = new int[matrix.length][matrix[0].length];

int[][] dir = new int[][] { {0,0,1,-1}, {-1,1,0,0} };

for (int i=0; i<matrix.length; ++i) {

for (int j=0; j<matrix[0].length; ++j) {

if (lenBoard[i][j] == 0) {

dfs(matrix, lenBoard, dir, i, j, Integer.MIN\_VALUE);

}

}

}

return max;

}

private int dfs(int[][] mat, int[][] lenBoard, int[][] dir, int row, int col, int prev) {

if (row<0 || row>=mat.length || col<0 || col>=mat[0].length || prev>=mat[row][col]) {

return 0; // out of board or path stops incresing

}

if (lenBoard[row][col] > 0) { return lenBoard[row][col]; } // current position already computed

int neighborMax = 0;

for (int i=0; i<4; ++i) { // retrieve the length of lip from its 4 neighbors

neighborMax = Math.max(neighborMax, dfs(mat, lenBoard, dir, row+dir[0][i], col+dir[1][i], mat[row][col]));

}

int localMax = 1 + neighborMax;

lenBoard[row][col] = localMax;

max = Math.max(max, localMax);

return localMax;

}

}

## 成绩：

16ms,beats 64.61%，众数16ms，19.26%

## Cmershen的碎碎念：

本题于2016.3.11难度调整为Hard。

# Middle-题目56：109. Convert Sorted List to Binary Search Tree

## 题目原文：

iven a singly linked list where elements are sorted in ascending order, convert it to a height balanced BST.

## 题目大意：

把单链表转换成二叉平衡树。

## 题目分析：

把单链表转换成数组，再调用[Middle-题目15](#_Middle-题目15：108._Convert_Sorted)的函数就可以了。如果要直接做的话，使用快慢指针法找到中点，再递归即可。

## 源码：（language：java）

public class Solution {

public TreeNode sortedListToBST(ListNode head) {

List<Integer> list = new ArrayList<Integer>();

for(ListNode temp = head;temp!=null;temp=temp.next)

list.add(temp.val);

Integer[] nums = new Integer[list.size()];

return sortedArrayToBST(list.toArray(nums));

}

public TreeNode sortedArrayToBST(Integer[] nums) {

if(nums.length==0)

return null;

TreeNode bst=createBST(nums,0,nums.length-1);

return bst;

}

public TreeNode createBST(Integer[] nums,int start,int end) {

int mid = (start + end) / 2;

TreeNode root = new TreeNode(nums[mid]);

if(start < mid )

root.left = createBST(nums,start,mid-1);

if(mid < end)

root.right = createBST(nums,mid+1,end);

return root;

}

}

## 成绩：

4ms,beats 1.06%,众数1ms, 59.87%

# Middle-题目57：90. Subsets II

## 题目原文：

Given a collection of integers that might contain duplicates, nums, return all possible subsets.

Note:

Elements in a subset must be in non-descending order.

The solution set must not contain duplicate subsets.

For example,

If nums = [1,2,2], a solution is:

[

[2],

[1],

[1,2,2],

[2,2],

[1,2],

[]

]

## 题目大意：

接[Middle-题目52](#_Middle-题目52：78._Subsets)，但这里面的数组会有重复元素，但子集中仍不允许有重复元素。

## 题目分析：

这次适合回溯法，但要注意，如果遇到当前数字和前一个数字相同，则跳过去。

使用backtrack(int[] nums, int index, List<List<Integer>> result, List<Integer> sublist)函数记录搜索过程，其中index记录当前搜索到的数字的索引(相同时索引取最后一个)，那么如果循环时nums[i]==nums[i-1]时则跳过。

## 源码：（language：java）

public class Solution {

public List<List<Integer>> subsetsWithDup(int[] nums) {

List<List<Integer>> result = new ArrayList<> ();

if(nums == null) {

return result;

}

Arrays.sort(nums);

backtrack(nums, 0, result, new ArrayList<Integer> ());

return result;

}

private void backtrack(int[] nums, int index, List<List<Integer>> result, List<Integer> sublist) {

if(index <= nums.length) {

result.add(new ArrayList<> (sublist));

}

for(int i = index; i < nums.length; i++) {

if(i != index && nums[i] == nums[i - 1]) {

continue;

}

sublist.add(nums[i]);

backtrack(nums, i + 1, result, sublist);

sublist.remove(sublist.size() - 1);

}

}

}

## 成绩：

3ms,beats 41.85%,众数3ms,41.53%

## Cmershen的碎碎念：

一个较笨的办法就是在上一题的基础上判断是否存在这个集合序列，但时间长达37ms,因为这样会搜出很多无效解。

# Middle-题目58：213. House Robber II

## 题目原文：

Note: This is an extension of House Robber.

After robbing those houses on that street, the thief has found himself a new place for his thievery so that he will not get too much attention. This time, all houses at this place are arranged in a circle. That means the first house is the neighbor of the last one. Meanwhile, the security system for these houses remain the same as for those in the previous street.

## 题目大意：

接[Easy-题目26](#_Easy-题目26：198._House_Robber)，但是现在房子围成了环形，仍然求最大利益。

## 题目分析：

分第一家是否偷讨论，如果偷第一家，那么最后一家不能偷，则退化成第一家到倒数第二家的线性问题，如果不偷第一家，那么相当于从第二家到最后一家的线性问题。

## 源码：（language：c）

int rob(int\* nums, int numsSize) {

if(numsSize==0)

return 0;

else if(numsSize==1)

return nums[0];

else if(numsSize==2)

return nums[0]>nums[1]?nums[0]:nums[1];

else

return max(rob1(nums,numsSize-1),rob1(nums+1,numsSize-1));

}

int max(int a,int b) {

return a>b?a:b;

}

int rob1(int\* nums, int numsSize) {

if(numsSize==0)

return 0;

if(numsSize==1)

return nums[0];

else if(numsSize==2)

return nums[0]>nums[1]?nums[0]:nums[1];

else

{

int i;

int dp1,dp2,dp;

dp1=nums[0];

dp2=nums[0]>nums[1]?nums[0]:nums[1];

for(i=2;i<numsSize;i++)

{

dp=dp2>dp1+nums[i]?dp2:dp1+nums[i];

dp1=dp2;

dp2=dp;

}

return dp;

}

}

## 成绩：

1ms，众数100%

## Cmershen的碎碎念：

一开始想到对环上每个节点遍历一次，后来发现只讨论第一个节点即可。

# Middle-题目59：201. [Bitwise AND of Numbers Range](http://blog.csdn.net/xudli/article/details/45912649)

## 题目原文：

Given a range [m, n] where 0 <= m <= n <= 2147483647, return the bitwise AND of all numbers in this range, inclusive.

For example, given the range [5, 7], you should return 4.

## 题目大意：

求[m.n]内所有整数按位与起来的值。

## 题目分析：

方法一：（分治法）若存在k∈[0,31],使得m≤2k<n<2k+1,则返回0，因为[m,n]的二进制数的第k位必定是有0也有1，且第k+1位都是0.而如果2k≤m<n<2k+1,则[m,n]的第k位全是1，那么结果就是2k+[m-2k,n-2k]区间相与值，这样不断降低问题规模，最终会只剩1-2个数，（因为终究会跟2比一次），直接返回m&n即可。

方法二：（基于位运算，来自<http://blog.csdn.net/xudli/article/details/45912649>博客）归根结底本题要求的是m和n二进制数的公共前缀，所以先按位右移，记录下移动位数，再按位左移回去即为所求。

## 源码：（language：java）

方法一：

public class Solution {

private static int[] pows = new int[]{1,2,4,8,16,32,64,128,256,512,1024,2048,4096,8192,16384,32768,65536,131072,262144,524288,1048576,2097152,4194304,8388608,16777216,33554432,67108864,134217728,268435456,536870912,1073741824};

public static int rangeBitwiseAnd(int m, int n) {

if(m>pows[30])

return pows[30] + rangeBitwiseAnd(m-pows[30],n-pows[30]);

for(int i = 1;i < 31; i++) {

if(m<pows[i] && n>pows[i])

return 0;

else if(m>=pows[i] && n<pows[i+1])

return pows[i] + rangeBitwiseAnd(m-pows[i],n-pows[i]);

}

return m&n;

}

}

方法二：

public class Solution {

public int rangeBitwiseAnd(int m, int n) {

int bit = 0;

while(m!=n) {

m>>=1;

n>>=1;

bit++;

}

return m<<bit;

}

}

## 成绩：

都是9ms，beats 19.89%，众数9ms,42.27%

## Cmershen的碎碎念：

第二种想法十分巧妙也容易理解，但第一种想法更为朴素。

# Middle-题目60：120. Triangle

## 题目原文：

Given a triangle, find the minimum path sum from top to bottom. Each step you may move to adjacent numbers on the row below.

For example, given the following triangle

[

[2],

[3,4],

[6,5,7],

[4,1,8,3]

]

The minimum path sum from top to bottom is 11 (i.e., 2 + 3 + 5 + 1 = 11).

## 题目大意:

给出一个三角形，求出从顶点到底边路径的最小值，每次只能向下移动到相邻的元素上。

## 分析：

令dp[i][j]代表从顶点走到[I,j]点的最小和，那么要么是从正上方来的，要么是从左上方来的，故有：

dp[i][j] = min(dp[i-1][j], dp[i-1][j-1])+triangle[i][j]，对两边的特殊讨论一下。

最后再对最后一行的dp值取最小值。

## 源码：（language：java）

public class Solution {

public int minimumTotal(List<List<Integer>> triangle) {

int size = triangle.size();

if(size == 1)

return triangle.get(0).get(0);

int[][] dp = new int[size][size];

dp[0][0] = triangle.get(0).get(0);

int min = Integer.MAX\_VALUE;

for(int i = 1; i < size; i++) {

for(int j = 0 ; j <= i ; j++) {

if(j == 0)

dp[i][j] = triangle.get(i).get(j) + dp[i-1][j];

else if(j == i)

dp[i][j] = triangle.get(i).get(j) + dp[i-1][j-1];

else

dp[i][j] = triangle.get(i).get(j) + Math.min(dp[i-1][j], dp[i-1][j-1]);

if(i == size-1)

min = Math.min(min, dp[i][j]);

}

}

return min;

}

}

## 成绩：

5ms,beats 57.96%,众数4ms,23.76%

## Cmershen的碎碎念：

这道题在本科的《算法设计与分析》课上学过原题，可见是一个很经典也很有代表性的DP问题。

# Middle-题目61：63. Unique Paths II

## 题目原文：

Follow up for "Unique Paths":

Now consider if some obstacles are added to the grids. How many unique paths would there be?

An obstacle and empty space is marked as 1 and 0 respectively in the grid.

For example,

There is one obstacle in the middle of a 3x3 grid as illustrated below.

[

[0,0,0],

[0,1,0],

[0,0,0]

]

The total number of unique paths is 2.

## 题目大意：

在[Middle-题目21](#_Middle-题目21：62._Unique_Paths)中，再增加一个障碍矩阵，有障碍的点是不能走的，仍然求不同的路径个数。

## 题目分析：

改动不大，有路障的点路径个数置0即可。需要注意一些特殊情况：如果起点有路障，则整个矩阵都是0；如果第一行（列）有路障，则路障点的右边（下边）都是0.。

## 源码：（language：java）

public class Solution {

public int uniquePathsWithObstacles(int[][] obstacleGrid) {

int m = obstacleGrid.length;

int n = obstacleGrid[0].length;

int[][] dp = new int[m][n];

dp[0][0] = obstacleGrid[0][0]==0?1:0;

for(int i=1;i<m;i++)

if(obstacleGrid[i][0] == 0)

dp[i][0]=dp[i-1][0];

for(int i=1;i<n;i++)

if(obstacleGrid[0][i] == 0)

dp[0][i]=dp[0][i-1];

for(int i=1;i<m;i++)

for(int j=1;j<n;j++)

if(obstacleGrid[i][j] == 0)

dp[i][j] = dp[i-1][j] + dp[i][j-1];

return dp[m-1][n-1];

}

}

## 成绩：

1ms,beats 17.74%,众数1ms,82.26%

# Middle-题目62：86. Partition List

## 题目原文：

Given a linked list and a value x, partition it such that all nodes less than x come before nodes greater than or equal to x.

You should preserve the original relative order of the nodes in each of the two partitions.

## 题目大意:

给出一个链表和一个值x,把＜x的节点都搬到x前面，大于x的节点都搬到x后面，并保持原有的顺序。

## 题目分析：

类似快排，注意一些边界情况和特殊情况即可。多看看错误的test case，多在纸上画画就好了。

## 源码：（language：java）

public class Solution {

public static ListNode partition(ListNode head, int x) {

if(head == null || head.next == null) return head;

ListNode beforex = null,afterx = null;

for(ListNode node = head; node!=null; node=node.next) {

if(node.val<x) {

if(beforex == null)

beforex = new ListNode(node.val);

else {

ListNode temp = beforex;

while(temp.next!=null)

temp=temp.next;

temp.next=new ListNode(node.val);

}

}

else {

if(afterx == null)

afterx = new ListNode(node.val);

else {

ListNode temp = afterx;

while(temp.next!=null)

temp=temp.next;

temp.next=new ListNode(node.val);

}

}

}

ListNode temp = beforex;

if (temp!=null) {

while(temp.next!=null)

temp=temp.next;

temp.next=afterx;

return beforex;

}

else

return afterx;

}

}

## 成绩：

1ms,beats 4.36%,众数1ms,95.64%

## cmershen的碎碎念：

本题的代码写的很复杂，以后应考虑有没有可以合并的边界情况。

# Middle-题目63：147. Insertion Sort List

## 题目原文：

Sort a linked list using insertion sort.

## 题目大意：

使用插入排序对单链表排序。

## 题目分析：

使用直接插入排序，即从第二个元素开始，每次寻找它应该在前面的有序区间内所在的位置。

## 源码：（language：c）

struct ListNode\* insertionSortList(struct ListNode\* head) {

if(!head || !head->next)

return head;

else {

struct ListNode\* sortedHead = (struct ListNode\*)malloc(sizeof(struct ListNode));

sortedHead->val = head->val;

sortedHead->next = NULL;

for (struct ListNode\* node = head->next; node; node = node->next) {

if(node->val <= sortedHead->val) { // this node is smaller than the smallest node of sortedHead

struct ListNode\* newNode = (struct ListNode\*)malloc(sizeof(struct ListNode));

newNode->val = node->val;

newNode->next = sortedHead;

sortedHead = newNode;

}

else {

struct ListNode\* temp;

for(temp = sortedHead; temp; temp = temp->next) {

if(temp->val <= node->val && (!temp->next || temp->next->val >node->val)) { //insert this node after temp

struct ListNode\* newNode = (struct ListNode\*)malloc(sizeof(struct ListNode));

newNode->val = node->val;

newNode->next = temp->next;

temp->next = newNode;

break;

}

}

}

}

return sortedHead;

}

}

## 成绩：

76ms,beats 25%,众数72ms,25%

## cmershen的碎碎念：

本题代码还是过于复杂，而且可以在有序区使用二分查找优化之。

# Middle-题目64：95. Unique Binary Search Trees II

## 题目原文：

Given n, generate all structurally unique BST's (binary search trees) that store values 1...n.

For example,

Given n = 3, your program should return all 5 unique BST's shown below.

1 3 3 2 1

\ / / / \ \

3 2 1 1 3 2

/ / \ \

2 1 2 3

## 题目大意：

给出n，输出节点值为1到n的所有二叉排序树。

## 题目分析：

使用分治法，分别令根节点为k，再递归生成1~k,k+1~n对应的二叉排序树列表，逐一加入到以k为根节点的列表中。

## 源码：（language：java）

public class Solution {

public List<TreeNode> generateTrees(int n) {

if(n==0) return new ArrayList<TreeNode>();

return generateTrees(1, n);

}

private List<TreeNode> generateTrees(int start, int end){

if(start>end) {

List<TreeNode> list = new ArrayList<TreeNode>();

list.add(null);

return list;

}

if(start==end) {

List<TreeNode> list = new ArrayList<TreeNode>();

list.add(new TreeNode(start)); return list;

}

List<TreeNode> roots = new ArrayList<TreeNode>();

for(int i=start;i<=end;i++){

List<TreeNode> leftTrees = generateTrees(start, i-1);

List<TreeNode> rightTrees = generateTrees(i+1, end);

for(int j=0;j<leftTrees.size();j++){

for(int k=0;k<rightTrees.size();k++){

TreeNode root = new TreeNode(i);

root.left = leftTrees.get(j);

root.right = rightTrees.get(k);

roots.add(root);

}

}

}

return roots;

}

}

## 成绩：

2ms,beats 88.62%,众数3ms,53.26%

## cmershen的碎碎念：

大概从这段时间的题目开始，独立完成已经有难度，代码和分析有很多会来源于百度到的各路大神的博客和discuss中，但都加以我自己的理解。

# Middle-题目65：34. Search for a Range

## 题目原文：

Given a sorted array of integers, find the starting and ending position of a given target value.

## 题目大意：

给出排序好的数组，寻找目标值所在的下标区间。如果目标值不存在，返回[-1,-1].

## 题目分析：

朴素解法是逐一比较，找到开始节点后再向后找，优化解法是使用两次二分查找。**而二分查找可以使用C++的lower\_bound和upper\_bound两个内置的接口实现。（这个学习了。）**

## 源码：（language：cpp）

class Solution {

public:

vector<int> searchRange(vector<int>& nums, int target) {

auto p1 = lower\_bound(nums.begin(), nums.end(), target);

auto p2 = upper\_bound(nums.begin(), nums.end(), target);

vector<int> res; res.clear();

if(\*p1 != target) {

res.push\_back(-1);

res.push\_back(-1);

return res;

}

res.push\_back(p1 - nums.begin());

res.push\_back(p2 - nums.begin() - 1);

return res;

}

};

## 成绩：

12ms,beats 7.83%,众数12ms, 90.36%

# Middle-题目66：236. Lowest Common Ancestor of a Binary Tree

## 题目原文：

Given a binary tree, find the lowest common ancestor (LCA) of two given nodes in the tree.

## 题目大意：

寻找**普通二叉树**中两个节点的公共祖先。

## 题目分析：

看到了discuss中的一个极为巧妙的算法，先在左子树中找p或q，记为left，再到右子树中找，记为right，如果找到了（p或q都行），就把root往上传，没找到记为null，再判断返回的left和right哪个不是null，公共祖先就在哪里。

如果没有理解的话，看以下两种情况：

1. p是q的祖先或者q是p的祖先，那么一定会搜索到root==p或root==q的情况，这样返回root，然后root的父节点上递归的两个函数一个为空一个为root，根据第5行则返回root，以此类推把root传到整个树的根节点。
2. p和q分别位于某节点的左子树和右子树中，那么该节点上递归返回的left和right都不为空，第四行会返回这个root（即公共祖先），再依次往上传。

## 源码：（language：java）

public class Solution {

public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {

if (root == p || root == q || root == null) return root;

TreeNode left = lowestCommonAncestor(root.left, p, q);

TreeNode right = lowestCommonAncestor(root.right, p, q);

if (left != null && right != null) return root;

return left != null ? left : right;

}

}

## 成绩：

13ms,beats 28.45%,众数13ms，45.40%

## cmershen的碎碎念：

朴素解法是记录下p和q的路径，再求公共部分的最后一个节点，此解法的设计是基于p和q的第一次出现，既简洁又易于理解，堪称妙绝！

# Middle-题目67：16. 3Sum Closest

## 题目原文：

Given an array S of n integers, find three integers in S such that the sum is closest to a given number, target. Return the sum of the three integers. You may assume that each input would have exactly one solution.

For example, given array S = {-1 2 1 -4}, and target = 1.

The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).

## 题目大意：

给出一个数组和一个和值S，求三个数使得它们的和与和值S最为接近。

## 题目分析：

不管是3sum，4sum还是更多的ksum，可以证明算法复杂度的下界是O(nk-1),(只是在某博客里看到的，具体证明我也不知道)

Ksum通用算法如下：

（1）排序，复杂度O(nlogn);

（2）外层循环遍历k-2个指针，暴力O(nk-2)，内层循环中使用two pointers的思想，一个指针i指向头，一个指针j指向尾，计算k个指针对应元素的和，如果大于sum则j--,如果小于sum则i++，这样i和j两个指针合起来扫一遍，故总复杂度O(nk-1)。

本题因为是找最为接近的，所以维护一个最小误差即可

## 源码：（language：java）

public class Solution {

public static int threeSumClosest(int[] nums, int target) {

Arrays.sort(nums);

int min = 9999999;

int length = nums.length-1;

for(int middle = 0;middle < length; middle++) {

int i = 0,j = length, error = 9999999;

while(i<middle && j>middle) {

error = nums[i] + nums[middle] + nums[j] - target;

if(error == 0)

return target;

else if (error > 0)

j--;

else

i++;

min=Math.abs(error)<Math.abs(min)?error:min;

}

}

return target+min;

}

}

## 成绩：

15ms ,beats 35.55%,众数14ms,12.16%

## Cmershen的碎碎念：

似乎用HashMap可以把2sum优化到线性，4sum优化到O(n2),具体有待继续研读。

# Middle-题目68：106. Construct Binary Tree from Inorder and Postorder Traversal

## 题目原文：

Given inorder and postorder traversal of a tree, construct the binary tree.

## 题目大意：

给出一棵二叉树中序和后序遍历的序列，构建这个二叉树。

## 题目分析：

中序遍历的顺序是左-中-右，而后序遍历是左-右-中，所以取后序遍历的最后一个元素到中序遍历串中匹配，匹配到之后递归根据子树的中序和后序序列构建左右子树。

## 源码：（language：c）

struct TreeNode\* buildTree(int\* inorder, int inorderSize, int\* postorder, int postorderSize) {

if(inorderSize == 0 && postorderSize == 0)

return NULL;

else {

int middle = postorder[postorderSize-1];

int i;

for (i = 0;inorder[i] != middle; i++);

struct TreeNode\* root = (struct TreeNode\*)malloc(sizeof(struct TreeNode)); //generate root

root->val = middle;

root->left = buildTree(inorder, i, postorder, i);

root->right = buildTree(&inorder[i+1], inorderSize-i-1, &postorder[i], inorderSize-i-1);

return root;

}

}

## 成绩：

20ms,35.14%,众数20ms,43.24%

## Cmershen的碎碎念：

本题的Java入口参数是(int[],int[]),而C中使用的是指针，这样可以有效的截取子数组。所以指针虽然很容易引发各种缺陷，是危险的，但有时也提供了很大的方便。

# Middle-题目69：330. Patching Array

## 题目原文：

Given a sorted positive integer array nums and an integer n, add/patch elements to the array such that any number in range [1, n] inclusive can be formed by the sum of some elements in the array. Return the minimum number of patches required.

Example 1:

nums = [1, 3], n = 6

Return 1.

Combinations of nums are [1], [3], [1,3], which form possible sums of: 1, 3, 4.

Now if we add/patch 2 to nums, the combinations are: [1], [2], [3], [1,3], [2,3], [1,2,3].

Possible sums are 1, 2, 3, 4, 5, 6, which now covers the range [1, 6].

So we only need 1 patch.

Example 2:

nums = [1, 5, 10], n = 20

Return 2.

The two patches can be [2, 4].

Example 3:

nums = [1, 2, 2], n = 5

Return 0.

## 题目大意：

给出一个列表nums和一个目标值n，计算至少还需要补充几个数，才能使得列表nums中的数可以组合成[1,n]的所有整数（不允许重复）。

## 题目分析：

算法思路来自<http://www.bubuko.com/infodetail-1345277.html>这篇文章，大致是这样的：

算法先定义miss代表目前[1,n]之间不能表示的最小数（或者表述为[1,miss-1]都能表示出来），因为数组还没遍历，所以目前连1都表示不了。接下来遍历数组，如果当前的数num<=miss，则更新miss为miss+num(道理很简单，1~miss-1都可表示，那必然包含了1~num，那么必然1~miss-1+num都可表示了)，如果num>miss,则我们就需要补充一个数，为了补充的数最少，我们补充miss这个数，这样就能表示到miss\*2这么多了，故更新miss=miss\*2(因为我们补充了miss，所以现在不能表示的最小数就是miss\*2)以此类推直到miss>n。

## 源码：（language：java）

public class Solution {

public int minPatches(int[] nums, int n) {

long miss = 1;

int add = 0;

int i = 0;

while (miss <= n) {

if (i < nums.length && nums[i] <= miss){

miss += nums[i++];

} else {

miss += miss;

add += 1;

}

}

return add;

}

}

## 成绩：

1ms,beats 15%,众数1ms，85%

## Cmershen的碎碎念：

这是一个很巧妙的贪心算法，不参考博客我是想不到。有一个需要注意的就是miss声明为long，这是因为n很大时，计算miss\*2可能会溢出。

# Middle-题目70：103. Binary Tree Zigzag Level Order Traversal

## 题目原文：

Given a binary tree, return the zigzag level order traversal of its nodes' values. (ie, from left to right, then right to left for the next level and alternate between).

For example:

Given binary tree {3,9,20,#,#,15,7},

3

/ \

9 20

/ \

15 7

return its zigzag level order traversal as:

[

[3],

[20,9],

[15,7]

]

## 题目大意：

给出一个二叉树，求锯齿形遍历结果。

## 题目分析：

其实还是基于层次遍历的，在[Easy-题目27](#_Easy-题目27：107.Binary_Tree_Level)基础上加入一个flag，记录该行是应该正向还是应该反向就好了。

## 源码：（language：java）

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

public class Solution {

public List<List<Integer>> zigzagLevelOrder(TreeNode root) {

Queue<TreeNode> queue=new LinkedList<TreeNode>();

Queue<Integer> levelqueue=new LinkedList<Integer>();

queue.add(root);

levelqueue.add(1);

boolean isReverse = true;

List<List<Integer>> result=new ArrayList();

List<Integer> temp=new ArrayList<Integer>();

// temp.add(root.val);

// result.add(temp);

if(root==null)

return result;

while(!queue.isEmpty())

{

TreeNode current=queue.remove();

int curLevel=levelqueue.remove();

//System.out.println(current.val+" level="+curLevel);

if(curLevel==result.size())

{

temp.add(current.val);

}

else

{

if(isReverse)

Collections.reverse(temp);

result.add(temp);

isReverse = !isReverse;

temp=new ArrayList<Integer>();

temp.add(current.val);

}

if(current.left!=null)

{

queue.add(current.left);

levelqueue.add(curLevel+1);

}

if(current.right!=null)

{

queue.add(current.right);

levelqueue.add(curLevel+1);

}

}

if(isReverse)

Collections.reverse(temp);

result.add(temp);

result.remove(0);

// Collections.reverse(result);

return result;

}

}

## 成绩：

4ms,beats 4.14%,众数3ms,45.49%

# Middle-题目71：105. Construct Binary Tree from Preorder and Inorder Traversal

## 题目原文：

Given preorder and inorder traversal of a tree, construct the binary tree.

## 题目大意：

给出一棵二叉树中序和前序遍历的序列，构建这个二叉树。

## 题目分析：

中序遍历的顺序是左-中-右，而后序遍历是中-左-右，所以取前序遍历的第一个元素到中序遍历串中匹配，匹配到之后递归根据子树的中序和前序序列构建左右子树。

## 源码：（language：c）

struct TreeNode\* buildTree(int\* preorder, int preorderSize, int\* inorder, int inorderSize) {

if(preorderSize == 0 && inorderSize == 0)

return NULL;

else {

int middle = preorder[0];

int i;

for (i = 0;inorder[i] != middle; i++);

struct TreeNode\* root = (struct TreeNode\*)malloc(sizeof(struct TreeNode)); //generate root

root->val = middle;

root->left = buildTree(preorder+1, i, inorder, i);

root->right = buildTree(preorder+i+1, preorderSize-i-1, inorder+i+1, inorderSize-i-1);

return root;

}

}

## 成绩：

20ms,beats 23.68%,众数20ms,36.84%

## Cmershen的碎碎念：

要注意，前序和后序遍历是不能唯一确定二叉树的。但是前序（中序，后序）+层次遍历能否确定二叉树，如何还原二叉树，还有待考虑。

# Middle-题目72：17. Letter Combinations of a Phone Number

## 题目原文：

Given a digit string, return all possible letter combinations that the number could represent.

A mapping of digit to letters (just like on the telephone buttons) is given below.



Input:Digit string "23"

Output: ["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"].

## 题目大意：

给出一个九键键盘，和数字序列，求出所有可能的对应字母组合。

## 题目分析：

还是一个暴力的回溯法，使用backtrack(List<String> list, String digits, String currentLetter, int index)维护搜索过程，其中list是字母组合的列表，digits是数字序列，currentLetter是当前的搜索结果，index记录digits串中该搜索的位置。

## 源码：（language：java）

public class Solution {

private String[][] letters = new String[][] {

{" "},

{},

{"a","b","c"},

{"d","e","f"},

{"g","h","i"},

{"j","k","l"},

{"m","n","o"},

{"p","q","r","s"},

{"t","u","v"},

{"w","x","y","z"}

};

public List<String> letterCombinations(String digits) {

List<String> list = new ArrayList<String>();

if(digits.length() > 0)

backtrack(list, digits, new String(), 0);

return list;

}

private void backtrack(List<String> list, String digits, String currentLetter, int index) {

if(index == digits.length()) {

list.add(currentLetter);

return;

}

else {

int num = digits.charAt(index) -'0';

for(int i = 0; i < letters[num].length; i++) {

currentLetter+=letters[num][i];

backtrack(list, digits, currentLetter, index+1);

currentLetter = currentLetter.substring(0,currentLetter.length()-1);

}

}

}

}

## 成绩：

1ms，beats 46.02%，众数1ms,53.98%

# Middle-题目73：55. Jump Game

## 题目原文：

Given an array of non-negative integers, you are initially positioned at the first index of the array.

Each element in the array represents your maximum jump length at that position.

Determine if you are able to reach the last index.

For example:

A = [2,3,1,1,4], return true.

A = [3,2,1,0,4], return false.

## 题目大意：

给出一个非负数的数组，你一开始站在数组的开头，每个元素代表你在这个位置能跳几步，问能否跳到最后一步。

## 题目分析：

从后向前思考，数组的最后一位是几都无所谓，所以从数组的倒数第二位开始考虑。要想跳到最后一位，则倒数第二位至少为1，如不满足，倒数第三位至少为2，倒数第四位至少为3，……这样找到了要想跳到最后一位必须经过的一个位置，再从这个位置开始考虑，前一位至少为1，前两位至少为2，……这样再找到一个必经的位置，如果最后能找到起点，则返回true。

## 源码：（language：c）

bool canJump(int\* nums, int numsSize) {

int i,j,pos=numsSize-1;

for(j=numsSize-2;j>-1;j--){

if(nums[j]>=pos-j){

pos = j;

}

}

return !pos;

}

## 成绩：

4ms,beats 33.33%,众数4ms, 66.67%

## Cmershen的碎碎念：

好多题解的方法都是从前往后DFS回溯，但数组较大时会搜索到大量无用解，而本算法只遍历一遍数组。

# Middle-题目74：50. Pow(x, n)

实现Math.pow(double,int)函数。直接水过。略。

# Middle-题目75：113. Path Sum II

## 题目原文：

Given a binary tree and a sum, find all root-to-leaf paths where each path's sum equals the given sum.

For example:

Given the below binary tree and sum = 22,

5

/ \

4 8

/ / \

11 13 4

/ \ / \

7 2 5 1

return

[

[5,4,11,2],

[5,8,4,5]

]

## 题目大意：

给出一个二叉树和目标和值，输出从根节点到叶子节点的路径为目标和值的所有路径。

## 题目分析：

还是一个回溯法，backtrack(List<List<Integer>> list, List<Integer> sublist, TreeNode root, int sum)维护搜索过程，其中sum记录还剩下的和值，如果sum恰好为当前节点值且当前节点是叶子节点（想想为什么不能是root==null的时候停止？这样每个路径会添加两遍），则停止搜索，否则递归向子树搜索。

## 源码：（language：java）

public class Solution {

public List<List<Integer>> pathSum(TreeNode root, int sum) {

List<List<Integer>> list = new ArrayList<List<Integer>>();

if(root!=null)

backtrack(list, new ArrayList<Integer>(), root, sum);

return list;

}

private void backtrack(List<List<Integer>> list, List<Integer> sublist, TreeNode root, int sum) {

sublist.add(root.val);

if(sum-root.val == 0 && root.left == null && root.right == null)

list.add(new ArrayList<Integer>(sublist));

if(root.left!=null)

backtrack(list, sublist, root.left, sum-root.val);

if(root.right!=null)

backtrack(list, sublist, root.right, sum-root.val);

sublist.remove(sublist.size()-1);

}

}

## 成绩：

3ms,beats 14.57%,众数3ms,51.99%

# Middle-题目76：47. Permutations II

## 题目原文：

Given a collection of numbers that might contain duplicates, return all possible unique permutations.

## 题目大意：

接[Middle-题目24](#_Middle-题目24：46._Permutations)，这回数组元素允许重复，还是求所有可能的排列情况。

## 题目分析：

方法一：（很慢的朴素解法）首先构造一个hashMap，统计每个元素出现的次数，然后开始回溯，先统计这个元素在hashMap中还剩多少个，记为times,然后分别加入1个，2个……再向下搜索，直到hashmap为空则找到一个解。

方法二：discuss中的一个一行解决的办法，暂时不是很能理解。

## 源码：（language：java/python）

方法一：

public class Solution {

public static List<List<Integer>> permuteUnique(int[] nums) {

HashMap<Integer,Integer> set = new HashMap<Integer,Integer>();

for(int num:nums) {

if(set.containsKey(num))

set.put(num, set.get(num)+1);

else

set.put(num, 1);

}

List<List<Integer>> list = new ArrayList<List<Integer>>();

backtrack(list, new ArrayList<Integer>(), set, 0);

return list;

}

private static void backtrack(List<List<Integer>> list, List<Integer> sublist, HashMap<Integer,Integer> set,int last) {

boolean allZero = true;

for(int value:set.values()) {

if(value!=0) {

allZero = false;

break;

}

}

if(allZero)

list.add(new ArrayList<Integer>(sublist));

for(int key:set.keySet()) {

if(key != last || sublist.size() == 0) {

int times = set.get(key);

for(int j = 1; j <= times; j++) { // count how many keys there are in set,each times add j of them after sublist

set.put(key, times-j);

for(int k = 0; k<j; k++)

sublist.add(key);

backtrack(list, sublist, set, key);

for(int k = 0; k<j; k++)

sublist.remove(sublist.size()-1);

set.put(key, times);

}

}

}

}

}

方法二：

class Solution(object):

def permuteUnique(self, nums):

"""

:type nums: List[int]

:rtype: List[List[int]]

"""

return [[n] + p for n in set(nums) for p in self.permuteUnique(nums[:nums.index(n)] + nums[nums.index(n) + 1:])] or [[]]

## 成绩：

方法一：19ms,beats 22.52%,众数5ms,19.29%

方法二：136ms,beats 27.77%,众数128ms,15.55%

## Cmershen的碎碎念：

解法一原则上回溯过程也是不重不漏的，也没有浪费时间在无效解上，但成绩很差，感觉问题出在了hashmap上，可以考虑其他数据结构（例如数组）。解法二有待深入了解python的语法后慢慢理解。但其实不是一个很快的算法。（而且原则上不推荐这组简短但晦涩的代码）

# Middle-题目77：92. Reverse Linked List II

## 题目原文：

Reverse a linked list from position m to n. Do it in-place and in one-pass.

For example:

Given 1->2->3->4->5->NULL, m = 2 and n = 4,

return 1->4->3->2->5->NULL.

## 题目大意：

给出一个单链表和两个数m,n，翻转链表的m~n位。

## 题目分析：

还是用two pointers的思想，记住m的前驱，令其后继指向n，而n的后继指向m的next，以此类推。（在纸上画画，并多考虑一些特殊情况，如m是开头，n是结尾等）

## 源码：（language：c）

struct ListNode\* reverseBetween(struct ListNode\* head, int m, int n) {

if (m == n || !head || !head->next)

return head;

int i = 1;

struct ListNode\* mPtr = head, \*nPtr = head;

for (struct ListNode\* node = head; node; node = node->next, i++) {

if (i == m)

mPtr = node;

if (i == n)

nPtr = node;

}

if (mPtr == head) {

while (mPtr != nPtr) {

struct ListNode\* temp = mPtr;

mPtr = mPtr->next;

temp->next = nPtr->next;

nPtr->next = temp;

}

return nPtr;

}

else {

struct ListNode\* mPtrPrev = head;

while (mPtrPrev->next != mPtr)

mPtrPrev = mPtrPrev->next;

mPtrPrev->next = nPtr;

while (mPtr != nPtr) {

struct ListNode\* temp = mPtr;

mPtr = mPtr->next;

temp->next = nPtr->next;

nPtr->next = temp;

}

return head;

}

}

## 成绩：

0ms，100%

## Cmershen的碎碎念：

这道题在纯链表节点处理的题里面算是较难的了，须结合源码并在纸上演示一下才能理解指针间的关系。

# Middle-题目78：131. Palindrome Partitioning

## 题目原文：

Given a string s, partition s such that every substring of the partition is a palindrome.

Return all possible palindrome partitioning of s.

For example, given s = "aab",

Return

[

["aa","b"],

["a","a","b"]

]

## 题目大意：

返回字符串s的所有回文子串的分割。

## 题目分析：

使用backtrack(List<List<String>> list, List<String> sublist, String s)维护搜索过程，每次从s的开头开始截取字符串，每次判断一下是否为回文串，如果是就放入sublist，然后去掉加入sublist的部分继续向下搜索，直到s为空。

## 源码：（language：java）

public class Solution {

public List<List<String>> partition(String s) {

List<List<String>> list = new ArrayList<List<String>>();

backtrack(list,new ArrayList<>(),s);

return list;

}

private void backtrack(List<List<String>> list, List<String> sublist, String s) {

if(s.length() == 0) {

list.add(new ArrayList<String>(sublist));

}

else {

for(int i = 1; i<=s.length();i++) {

if(isPalindrome(s.substring(0, i))) {

sublist.add(s.substring(0, i));

backtrack(list, sublist, s.substring(i));

sublist.remove(sublist.size()-1);

}

}

}

}

public boolean isPalindrome(String s) {

if(s==null||s.length()==0)

return true;

for(int i = 0,j = s.length()-1;i<j;i++,j--) {

if(s.charAt(i)!=s.charAt(j))

return false;

}

return true;

}

}

## 成绩：

8ms,beats 31.90%,众数8ms,29.29%

# Middle-题目79：40. Combination Sum II

## 题目原文：

Given a collection of candidate numbers (C) and a target number (T), find all unique combinations in C where the candidate numbers sums to T.

Each number in C may only be used once in the combination.

Note:

All numbers (including target) will be positive integers.

Elements in a combination (a1, a2, … , ak) must be in non-descending order. (ie, a1 ≤ a2 ≤ … ≤ ak).

The solution set must not contain duplicate combinations.

For example, given candidate set 10,1,2,7,6,1,5 and target 8,

A solution set is:

[1, 7]

[1, 2, 5]

[2, 6]

[1, 1, 6]

## 题目大意：

在[Middle-题目54](#_Middle-题目54：39._Combination_Sum)的基础上，现在允许重复数字出现，但每个重复数字只能用一次，还是求所有的排列情况。

## 题目分析：

在[Middle-题目54](#_Middle-题目54：39._Combination_Sum)的基础上修改，还是先排序，相同的元素分别统计，然后去重的思路与[Middle-题目76](#_Middle-题目76：47._Permutations_II)相同。

## 源码：（language：java）

public class Solution {

public List<List<Integer>> combinationSum2(int[] candidates, int target) {

Arrays.sort(candidates);

List<List<Integer>> list = new ArrayList<List<Integer>>();

backtrack(list, new ArrayList<Integer>(), candidates, target, 0);

return list;

}

private void backtrack(List<List<Integer>> list, List<Integer> sublist, int[] candidates, int target, int startPos) {

if(target == 0) {

list.add(new ArrayList<Integer>(sublist));

return;

}

else {

for(int i = startPos; i < candidates.length; i++) {

int temp = candidates[i];

if(temp <= target) {

int iCount = 1;

while(i+1!=candidates.length&&candidates[i]==candidates[i+1]) {

iCount++;

i++;

}

for(int j = 1;j<=iCount;j++) {

for(int k = 0;k<j;k++)

sublist.add(temp);

backtrack(list,sublist,candidates,target-temp\*j,i+1);

for(int k = 0;k<j;k++)

sublist.remove(sublist.size()-1);

}

}

}

}

}

}

## 成绩：

9ms,beats 62.50%,众数5ms,18.18%

# Middle-题目80：200. Number of Islands

## 题目原文：

Given a 2d grid map of '1's (land) and '0's (water), count the number of islands. An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Example 1:

11110

11010

11000

00000

Answer: 1

Example 2:

11000

11000

00100

00011

Answer: 3

## 题目大意：

给出一个矩阵，其中1代表岛，0代表海水，求出有几个岛。

## 题目分析：

本题是一个标准的不能再标准的dfs问题，从一个未访问的“岛”点开始从四个方向搜索，遇到“海水”或者矩阵边界就退出。

## 源码：（language：java）

public class Solution {

public int numIslands(char[][] grid) {

if (grid.length==0 || grid[0].length==0)

return 0;

int count = 0;

boolean[][] visited = new boolean[grid.length][grid[0].length];

for (int i=0; i<grid.length; i++) {

for (int j=0; j<grid[0].length; j++) {

if (grid[i][j]=='1' && !visited[i][j]) {

count++;

dfs(grid, visited, i, j);

}

}

}

return count;

}

private void dfs(char[][] grid, boolean[][] visited, int row, int col) {

if (row<0 || row>=grid.length || col<0 || col>=grid[0].length || visited[row][col] || grid[row][col]=='0') // grid[i][j] is out of range

return;

visited[row][col] = true;

dfs(grid, visited, row-1, col);

dfs(grid, visited, row+1, col);

dfs(grid, visited, row, col-1);

dfs(grid, visited, row, col+1);

}

}

## 成绩：

5ms,beats 25.10%，众数3ms,40.17%

## Cmershen的碎碎念：

其实本题的算法也适用于计算机图形学中的“颜色填充算法”，其原理一模一样。

# Middle-题目81：264. Ugly Number II

## 题目原文：

Write a program to find the n-th ugly number.

Ugly numbers are positive numbers whose prime factors only include 2, 3, 5. For example, 1, 2, 3, 4, 5, 6, 8, 9, 10, 12 is the sequence of the first 10 ugly numbers.

Note that 1 is typically treated as an ugly number.

## 题目大意：

求第n个丑陋数。

丑陋数指的是质因数只有2，3，5的正整数。注意1是丑数。

## 题目分析：

其实本题是前面的[Middle-题目41](#_Middle-题目41：313._Super_Ugly)的退化情况，相当于求列表[2,3,5]下的超级丑数。因此算法本质与那道题完全一样，只是这次只维护2，3，5三个素数的索引值即可。

## 源码：（language：java）

public class Solution {

public int nthUglyNumber(int n) {

int l1=1,l2=1,l3=1;

int a,b,c;

int[] uglyNumber = new int[n+1];

uglyNumber[1] = 1;

for(int i = 2;i<=n;i++) {

a=uglyNumber[l1]\*2;

b=uglyNumber[l2]\*3;

c=uglyNumber[l3]\*5;

int min = min(a,b,c);

if(min==a)

l1++;

if(min==b)

l2++;

if(min==c)

l3++;

uglyNumber[i]=min;

}

return uglyNumber[n];

}

private int min(int a, int b, int c) {

if(a<=b && a<=c)

return a;

else if(b<=a && b<=c)

return b;

else

return c;

}

}

## 成绩：

7ms,beats 96.36%,众数9ms,12.39%

# Middle-题目82：134. Gas Station

## 题目原文：

There are N gas stations along a circular route, where the amount of gas at station i is gas[i].

You have a car with an unlimited gas tank and it costs cost[i] of gas to travel from station i to its next station (i+1). You begin the journey with an empty tank at one of the gas stations.

Return the starting gas station's index if you can travel around the circuit once, otherwise return -1.

## 题目大意：

有n个加油站围成一个圈，其中第i个加油站的油量是gas[i].

你有一个汽车，每从第i个加油站开到第i+1个，要消耗cost[i]的汽油。初始油箱为空，问这个汽车能否开完一圈，如果能则返回一个起点位置，否则返回-1.

## 题目分析：

首先朴素算法就是从每个节点开始开一圈，看看能不能走到头。这样提交会超时。

那么从discuss看到的一个贪心算法，从节点0开始向前开，如果开完一个站点还有油就向前开，如果油量不足，就把出发站点往回移动一站，并从出发点重新开始开，直到遍历完一圈，如果最后油箱里的油≥0，则返回最后的出发点，否则返回-1.

## 源码：（language：java）

public class Solution {

public int canCompleteCircuit(int[] gas, int[] cost) {

if (gas == null) {

return -1;

}

// Note: The Solution object is instantiated only once and is reused by each test case.

int count = gas.length;

int n = 0;

int gasInCar = 0;

int begin = 0;

int end = 0;

int i = 0;

while (n < count - 1) {

gasInCar += gas[i] - cost[i];

if (gasInCar >=0) {//forward

end++;

i=end;

} else {

begin--;

if (begin < 0) {

begin = count - 1;

}

i = begin;

}

n++;

}

gasInCar += gas[i] - cost[i];

if (gasInCar >= 0) {

return begin;

} else {

return -1;

}

}

}

## 成绩：

1ms,beats 5.79%,众数1ms,82.52%

## Cmershen的碎碎念：

这个算法的理解和实现都不算难，但暂时没想到如何严格证明其正确性。

# Middle-题目83：49. Group Anagrams

## 题目原文：

Given an array of strings, group anagrams together.

For example, given: ["eat", "tea", "tan", "ate", "nat", "bat"],

Return:

[

["ate", "eat","tea"],

["nat","tan"],

["bat"]

]

## 题目大意：

给出一个字符串数组，将所有的同构词分组。

## 题目分析：

使用一个HashMap，其中key是每个同构词的最小字典序序列，例如ate,eat,tea的key都是aet，value是一个list，记录该key的同构词（排序后与key相同的都加入value对应的list中），然后根据key对字符串分组。

## 源码：（language：java）

public class Solution {

public List<List<String>> groupAnagrams(String[] strs) {

Map<String, List<String>> map = new HashMap<String, List<String>>();

for(String str : strs) {

char[] carr = str.toCharArray(); // 将单词按字母排序

Arrays.sort(carr);

String key = new String(carr);

List<String> list = map.get(key);

if(list == null) {

list = new ArrayList<String>();

}

list.add(str);

map.put(key, list);

}

List<List<String>> res = new ArrayList<List<String>>();

for(String key : map.keySet()) {

List<String> curr = map.get(key);

Collections.sort(curr); // 将列表按单词排序

res.add(curr);

}

return res;

}

}

## 成绩：

26ms,beats 73.2%,众数33ms,8.78%

## Cmershen的碎碎念：

原来我的思路是每加入一个字符串的时候，就比较和哪个字符串是同构关系，但因为每次都比较，在数据量大的时候会tle，参考leetcode中构造HashMap的方法解决超时问题。因为每次只需要和一个key比较就可以了。

# Middle-题目84：82. Remove Duplicates from Sorted List II

## 题目原文：

Given a sorted linked list, delete all nodes that have duplicate numbers, leaving only distinct numbers from the original list.

For example,

Given 1->2->3->3->4->4->5, return 1->2->5.

Given 1->1->1->2->3, return 2->3.

## 题目大意：

给出一个排序好的链表，删除所有重复元素，只保留一个，且保持原来的顺序。

## 题目分析：

维护两个指针p1和p2，其中p2指向p1下一个节点，每次判断p1和p2的值是否相等，如果相等则删掉p2，且p2向后推，直到p2不等于p1，再令p1等于p2继续判断，总共扫描一遍链表即可。

## 源码：（language：c）

struct ListNode\* deleteDuplicates(struct ListNode\* head) {

if (!head || !head->next)

return head;

else {

struct ListNode\* ptr1 = head, \*ptr2 = head;

while (ptr2) {

ptr1 = ptr2;

while (ptr2 && ptr2->val == ptr1->val)

ptr2 = ptr2->next;

if (ptr1->next == ptr2) // no duplicate

ptr1 = ptr2;

else if (ptr1 == head)

head = ptr2;

else {

struct ListNode\* prev = head;

while (prev->next != ptr1)

prev = prev->next;

prev->next = ptr2;

ptr1 = ptr2;

}

}

return head;

}

}

## 成绩：

4ms,beats 8.00%,众数4ms,92%

# Middle-题目85：310. Minimum Height Trees

## 题目原文：

For a undirected graph with tree characteristics, we can choose any node as the root. The result graph is then a rooted tree. Among all possible rooted trees, those with minimum height are called minimum height trees (MHTs). Given such a graph, write a function to find all the MHTs and return a list of their root labels.

Format

The graph contains n nodes which are labeled from 0 to n - 1. You will be given the number n and a list of undirected edges (each edge is a pair of labels).

You can assume that no duplicate edges will appear in edges. Since all edges are undirected, [0, 1] is the same as [1, 0] and thus will not appear together in edges.

Example 1:

Given n = 4, edges = [[1, 0], [1, 2], [1, 3]]

0

|

1

/ \

2 3

return [1]

Example 2:

Given n = 6, edges = [[0, 3], [1, 3], [2, 3], [4, 3], [5, 4]]

0 1 2

\ | /

3

|

4

|

5

return [3, 4]

## 题目大意：

给出一个树，它的任何一个顶点都可以看做树的根节点，写一个函数，输出使得树高度最小的顶点的列表。

输入参数是顶点个数和边的列表（用二维数组表示）。

例如，输入n = 4, 边列表= [[1, 0], [1, 2], [1, 3]]

则以1为根节点的树的高度最小。

## 题目分析：

不管树有多复杂，使得高度最小的顶点至多有2个。（我也想不出证明过程……）

暴力解法是对每个节点求高度。这里有一个比较好的算法，即每轮去掉度为1的节点，直到顶点剩下1个或2个（显然不可能是3个或更多，因为没有环），那么这1个或2个节点即为所求。

## 源码：（language：java）

public class Solution {

public List<Integer> findMinHeightTrees(int n, int[][] edges) {

Map<Integer, Set<Integer>> adjList = new HashMap<Integer,Set<Integer>>();

boolean[] isLeaf = new boolean[n];

int[] degrees = new int[n];

int nextVertical = 0, currentVerticals = n;

for(int i = 0; i<n;i++)

adjList.put(i, new HashSet<Integer>());

for(int[] edge : edges) {

adjList.get(edge[0]).add(edge[1]);

degrees[edge[0]]++;

adjList.get(edge[1]).add(edge[0]);

degrees[edge[1]]++;

}

while(currentVerticals > 2) {

for(int i = 0;i<n;i++) {

if(degrees[i]==1) {

isLeaf[i] = true;

}

}

for(int i = 0;i<n;i++) {

if(isLeaf[i]) {

nextVertical = adjList.get(i).iterator().next();

degrees[i]--;

degrees[nextVertical]--;

adjList.get(i).remove(nextVertical);

if(degrees[i]==0)

currentVerticals--;

adjList.get(nextVertical).remove(i);

if(degrees[nextVertical]==0)

currentVerticals--;

isLeaf[i] = false;

}

}

}

List<Integer> list = new ArrayList<Integer>();

for(Integer key : adjList.keySet()) {

if(adjList.get(key).size()!=0)

list.add(key);

}

if(n==1)

list.add(0);

if(currentVerticals==0)

list.add(nextVertical);

return list;

}

}

这里给出discuss中一位大神给出的算法，思路是一样的，但省略了很多循环：

public class Solution {

public List<Integer> findMinHeightTrees(int n, int[][] edges) {

if (n == 1)

return Collections.singletonList(0);

List<Set<Integer>> adj = new ArrayList<>(n);

for (int i = 0; i < n; i++)

adj.add(new HashSet<>());

for (int[] edge : edges) {

adj.get(edge[0]).add(edge[1]);

adj.get(edge[1]).add(edge[0]);

}

List<Integer> leaves = new ArrayList<>();

for (int i = 0; i < n; i++)

if (adj.get(i).size() == 1) leaves.add(i);

while (n > 2) {

n -= leaves.size();

List<Integer> newLeaves = new ArrayList<>();

for (int i : leaves) {

int j = adj.get(i).iterator().next();

adj.get(j).remove(i);

if (adj.get(j).size() == 1) newLeaves.add(j);

}

leaves = newLeaves;

}

return leaves;

}

}

## 成绩：

算法一（我写的）：170ms,beats 14.07%,众数56ms,5.12%

算法二（from discuss）：58ms,beats 68.02%

## Cmershen的碎碎念：

这是leetcode里面第一个涉及图的题，图应该是数据结构里面的难点，因此这题的成绩很分散。同时说明，即使算法思路相同，内部实现细节不同也会导致成绩有很大差异。比如记录邻接表的时候人家用的是list的下标而不是使用Map的key值，而一直用leaves数组维护当前叶子节点而不是一直在数度，也是比较巧妙的。

# Middle-题目86：207. Course Schedule

## 题目原文：

There are a total of n courses you have to take, labeled from 0 to n - 1.

Some courses may have prerequisites, for example to take course 0 you have to first take course 1, which is expressed as a pair: [0,1]

Given the total number of courses and a list of prerequisite pairs, is it possible for you to finish all courses?

For example:

2, [[1,0]]

There are a total of 2 courses to take. To take course 1 you should have finished course 0. So it is possible.

2, [[1,0],[0,1]]

There are a total of 2 courses to take. To take course 1 you should have finished course 0, and to take course 0 you should also have finished course 1. So it is impossible.

## 题目大意：

你要上n门课，课的编号分别是0,…n-1.

有一些课程有先修课，例如要学习课程0需先学习课程1，这样用一个有序数对[0,1]表示。

给出课程的数目和先修课程的有序数对列表，判断是否能修完所有课程。

## 题目分析：

这也是一道关于图的问题。用有向图来表示课程的先修关系，例如数对[0.1]对应有向图中0->1的边。判断是否存在拓扑序列即可。

## 源码：（language：java）

public class Solution {

public boolean canFinish(int numCourses, int[][] prerequisites) {

Map<Integer, Set<Integer>> adjList = new HashMap<Integer,Set<Integer>>();

int[] indegree = new int[numCourses];

for(int i = 0; i<numCourses;i++)

adjList.put(i, new HashSet<Integer>());

for(int[] edge : prerequisites) {

if(!adjList.get(edge[0]).contains(edge[1])) {

adjList.get(edge[0]).add(edge[1]);

indegree[edge[1]]++;

}

}

int topoVertical = findZeroIndegree(indegree);

while(topoVertical!=-1) {

indegree[topoVertical] = -1; // -1 stands for visited

for(Integer vertical : adjList.get(topoVertical)) {

indegree[vertical]--;

}

adjList.remove(topoVertical);

topoVertical = findZeroIndegree(indegree);

}

return isAllVisited(indegree);

}

private int findZeroIndegree(int[] indegree) {

for(int i = 0;i<indegree.length;i++)

if(indegree[i]==0)

return i;

return -1;

}

private boolean isAllVisited(int[] indegree) {

for(int i : indegree)

if(i!=-1)

return false;

return true;

}

}

## 成绩：

24ms,beats 47.84%，众数12ms,6.33%

## Cmershen的碎碎念：

首先复习拓扑排序的定义和算法：对一个有向无环图(Directed Acyclic Graph简称DAG)G进行拓扑排序，是将G中所有顶点排成一个线性序列，使得图中任意一对顶点u和v，若边(u,v)∈E(G)，则u在线性序列中出现在v之前。通常，这样的线性序列称为满足拓扑次序(Topological Order)的序列，简称拓扑序列。

算法：

(1) 选择一个入度为0的顶点并输出之；

(2) 从网中删除此顶点及所有出边。

循环结束后，若输出的顶点数小于网中的顶点数，则输出“有回路”信息，否则输出的顶点序列就是一种拓扑序列。

而课程先修（或一个工程中各个子工程先后完成关系）是拓扑排序中最基本的问题。

# Middle-题目87：209. Minimum Size Subarray Sum

## 题目原文：

Given an array of n positive integers and a positive integer s, find the minimal length of a subarray of which the sum ≥ s. If there isn't one, return 0 instead.

For example, given the array [2,3,1,2,4,3] and s = 7,

the subarray [4,3] has the minimal length under the problem constraint.

## 题目大意：

给出一个正整数的数组，和一个整数s，求出使得和值≥s的最小长度。

## 题目分析：

使用两个指针start和end，end向后移到sum>s，然后start向右移，直到sum＜s，并更新minLength值。End指针移动到结尾时循环，返回minLength.（源自西施豆腐渣博客<http://blog.csdn.net/xudli/article/details/45715257>）

## 源码：（language：java）

public class Solution {

public int minSubArrayLen(int s, int[] nums) {

int start = 0;

int end = 0;

int sum = 0;

int min = Integer.MAX\_VALUE;

while(start<nums.length && end<nums.length) {

while(sum<s && end<nums.length) {

sum += nums[end++];

}

while(sum>=s && start<=end) {

min = Math.min(min, end-start);

sum -= nums[start++];

}

}

return min==Integer.MAX\_VALUE ? 0 : min;

}

}

## 成绩：

1ms,beats 18.63%,众数1ms,81.30%

## Cmershen的碎碎念：

这道题一开始想到的是排序后再从后向前查找（因为数字大才有可能序列短嘛。），这个解法显然是错误的，因为会打乱原顺序，但提交时竟然只有2个test case没通过，且暴力枚举之，竟超越了100%……

# Middle-题目88：31. Next Permutation

## 题目原文：

Implement next permutation, which rearranges numbers into the lexicographically next greater permutation of numbers.

If such arrangement is not possible, it must rearrange it as the lowest possible order (ie, sorted in ascending order).

The replacement must be in-place, do not allocate extra memory.

Here are some examples. Inputs are in the left-hand column and its corresponding outputs are in the right-hand column.

1,2,3 → 1,3,2

3,2,1 → 1,2,3

1,1,5 → 1,5,1

## 题目大意：

求出一个数组的下一个字典序的排序结果。

## 题目分析：

使用C++中的库函数next\_permutation实现（在头文件algorithm中）。

函数接口：bool next\_permutation( iterator start, iterator end);

即求start-end这个序列的下一个排序。

## 源码：（language：cpp）

class Solution {

public:

void nextPermutation(vector<int>& nums) {

next\_permutation(nums.begin(), nums.end());

}

};

## 成绩：

12ms,beats 10.04%,众数12ms,88.95%

## Cmershen的碎碎念：

C++11中，Next\_permutation的源码如下：

bool \_Next\_permutation(\_BidIt \_First, \_BidIt \_Last, \_Pr \_Pred)

{ // permute and test for pure ascending, using \_Pred

\_BidIt \_Next = \_Last;

if (\_First == \_Last || \_First == --\_Next)

return (false);

for (; ; )

{ // find rightmost element smaller than successor

\_BidIt \_Next1 = \_Next;

if (\_DEBUG\_LT\_PRED(\_Pred, \*--\_Next, \*\_Next1))

{ // swap with rightmost element that's smaller, flip suffix

\_BidIt \_Mid = \_Last;

for (; !\_DEBUG\_LT\_PRED(\_Pred, \*\_Next, \*--\_Mid); )

;

\_STD iter\_swap(\_Next, \_Mid);

\_STD reverse(\_Next1, \_Last);

return (true);

}

if (\_Next == \_First)

{ // pure descending, flip all

\_STD reverse(\_First, \_Last);

return (false);

}

}

}

大致思路：

在当前序列中，从尾端向前寻找两个相邻元素，前一个记为\*i，后一个记为\*t，并且满足\*i < \*t。然后再从尾端寻找另一个元素\*j，如果满足\*i < \*j，即将第i个元素与第j个元素对调，并将第t个元素之后（包括t）的所有元素颠倒排序，即求出下一个序列了。

# Middle-题目89：229. Majority Element II

## 题目原文：

Given an integer array of size n, find all elements that appear more than ⌊ n/3 ⌋ times. The algorithm should run in linear time and in O(1) space.

## 题目大意：

给出一个数组，寻找所有出现次数超过[n/3]的元素。要求O(n)时间及O(1)空间复杂度。

## 题目分析：

方法一（朴素解法，最坏O(n)空间复杂度）用HashMap统计每个元素出现次数，返回超过n/3的。

方法二：类似于前面的[寻找唯一主元素（Easy-题目11）](#_Easy-题目11：169._Majority_Element)，这次因为出现次数超过n/3，所以至多有2个。那么维护两个重复次数c1,c2就可以得到出现次数最多的前两个元素，再扫描一遍看看是否超过n/3即可。

## 源码：（language：java）

方法一：

public class Solution {

public List<Integer> majorityElement(int[] nums) {

List<Integer> list = new ArrayList<Integer>();

HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();

for(int num:nums) {

if(map.containsKey(num)) {

int count = map.get(num);

count++;

map.put(num,count);

if(count > nums.length/3 && !list.contains(num))

list.add(num);

}

else {

map.put(num, 1);

if(nums.length < 3)

list.add(num);

}

}

return list;

}

}

方法二：

public class Solution {

public List<Integer> majorityElement(int[] nums) {

// 1, 2

List<Integer> res = new ArrayList<>();

if(nums==null || nums.length==0) return res;

if(nums.length==1) {

res.add(nums[0]);

return res;

}

int m1 = nums[0];

int m2 = 0;

int c1 = 1;

int c2 = 0;

for(int i=1; i<nums.length; i++) {

int x = nums[i];

if(x==m1) ++c1;

else if(x==m2) ++c2;

else if(c1==0) {

m1 = x;

c1 = 1;

} else if(c2==0) {

m2 = x;

c2 = 1;

} else {

--c1; --c2;

}

}

c1 = 0; c2 = 0;

for(int i=0; i<nums.length; i++) {

if(m1 == nums[i]) ++c1;

else if(m2 == nums[i]) ++c2;

}

if(c1>nums.length/3) res.add(m1);

if(c2>nums.length/3) res.add(m2);

return res;

}

}

## 成绩：

方法一：35ms,beats 5.75%,众数4ms,31.82%

方法二：5ms,beats 29.28%

## Cmershen的碎碎念：

是否能进一步优化，使得只扫描一遍数组？因为我观察到还有约25%的提交代码做到3ms。

# Middle-题目90：306. Additive Number

## 题目原文：

Additive number is a string whose digits can form additive sequence.

A valid additive sequence should contain at least three numbers. Except for the first two numbers, each subsequent number in the sequence must be the sum of the preceding two.

For example:

"112358" is an additive number because the digits can form an additive sequence: 1, 1, 2, 3, 5, 8.

1 + 1 = 2, 1 + 2 = 3, 2 + 3 = 5, 3 + 5 = 8

"199100199" is also an additive number, the additive sequence is: 1, 99, 100, 199.

1 + 99 = 100, 99 + 100 = 199

Note: Numbers in the additive sequence cannot have leading zeros, so sequence 1, 2, 03 or 1, 02, 3 is invalid.

Given a string containing only digits '0'-'9', write a function to determine if it's an additive number.

## 题目大意：

判断一个由’0’-‘9’组成的字符串是不是” Additive Number”.

Additive Number 的定义是，可以拆成至少3个子串，其中任意前两个子串对应数字的和都等于后一个子串。

例如“112358”、“199100199”都是Additive Number。

需要注意的是，Additive Number中拆出的子串不能是以0开头的，例如”1023”不是Additive Number**，**因为拆成”1”,”02”,”3”不是合法的。

## 题目分析：

用的是暴力的解法，因为考虑Additive Number拆出的子串有可能非常长，故使用BigInteger存储（但是看别人的ac代码好像用的是long，这严格意义上说是不严谨的），从第一个数字开始暴力枚举，拆出前两个数并求和，再转换成字符串，看看拆掉前两个数后的子串是不是以这个“和字符串”开头，如果是，则减掉第一个数继续判断。以此类推直到找不到这样三个子串，或者拆掉前两个数的子串恰好等于“和字符串”。

## 源码：（language：java）

import java.math.BigInteger;

public class Solution {

public boolean isAdditiveNumber(String num) {

int length = num.length();

boolean result = false;

for(int i = 0;i<length-1;i++){

//if(num.charAt(i+1)=='0')

// continue;

String temp1 = num.substring(0, i+1);

if(temp1.length() > 1 && temp1.startsWith("0"))

continue;

BigInteger num1 = new BigInteger(num.substring(0, i+1)); // start: 0 ,end: i

for(int j= i+1;j<length-1;j++) {

String temp2 = num.substring(i+1,j+1);

if(temp2.length()>1 && temp2.startsWith("0"))

break;

BigInteger num2 = new BigInteger(num.substring(i+1,j+1));

BigInteger sum = num1.add(num2);

if(num.length()-j-1<sum.toString().length())

break;

else if(num.substring(j+1, j+1+sum.toString().length()).equals(sum.toString())) {

if(1+j+sum.toString().length() == num.length()) {

result= true;

break;

}

else

result = result || isAdditiveNumber(num.substring(i+1));

}

}

}

return result;

}

}

## 成绩：

23ms,beats 1.60%,众数3ms,39.48%

## Cmershen的碎碎念：

本题成绩较差，感觉是由于BigInteger拖慢了时间，看了其他的ac代码，有的是用long（这是不严谨的），有的是直接底层实现了高精度加法。

# Middle-题目91：69. Sqrt(x)

题目要求实现Sqrt(x),用库函数水过故略。

# Middle-题目92：139. Word Break

## 题目原文：

Given a string s and a dictionary of words dict, determine if s can be segmented into a space-separated sequence of one or more dictionary words.

For example, given

s = "leetcode",

dict = ["leet", "code"].

Return true because "leetcode" can be segmented as "leet code".

## 题目大意：

给出一个字符串s和一个字典dict，判断s是否可以拆分成一个或多个字典内单词的连接。

例如s=”leetcode”,dict=[“leet”,”code”]则可以。

## 题目分析：

方法一：朴素解法，超时。从字典中寻找能否匹配s的前缀，如果能，则向下搜索，如果不能，则退回来再找下一个能匹配s前缀的单词。因为字符串过长的时候，可能浪费很多时间在错误解上，故超时。

方法二：（动态规划，解法来自“快乐de胖虎”大神的博客）记dp[i]表示字符串s的前i个字符能否在字典中匹配，则初始化dp[0]=true,其余为false（规定任意字典都能表示空串），则转移方程如下：

即前i个字符能否匹配取决于前j个字符能否匹配并且后i-j个字符在字典中，因此求以上表达式的析取。

## 源码：（language：java）

public class Solution {

public boolean wordBreak(String s, Set<String> dict){

int len = s.length();

boolean[] arrays = new boolean[len+1];

arrays[0] = true;

for (int i = 1; i <= len; ++i){

for (int j = 0; j < i; ++j){

if (arrays[j] && dict.contains(s.substring(j, i))){

// f(n) = f(0,i) + f(i,j) + f(j,n)

arrays[i] = true;

break;

}

}

}

return arrays[len];

}

}

## 成绩：

12ms,beats 31.03%,众数12ms,24.67%

## Cmershen的碎碎念：

初次看到此题想到的是贪心算法，即找到就向下搜，但若s=”abc”,dict=[“ab”,”a”,”bc”]时则会先搜到”ab”这个错误解，而朴素的回溯法在面对s=”aaaaaaaaaaaaaaaaaab”,dict=[“a”,”aa”,”aaa”,”aaaa”,”aaaaa”]时会陷入很慢的搜索，因为无论如何向下搜索都是错误解，且aaaaaaaaaaaaaaaaaa拆成若干a的组合有很多，算法都会遍历一遍，却不知字符b在字典里根本不存在。

不过此处提一个思考题，”aaaaaaaaaaaaaaaaaa”拆成1~5个a的连接的不同方法有多少种？

# Middle-题目93：60. Permutation Sequence

## 题目原文：

The set [1,2,3,…,n] contains a total of n! unique permutations.

By listing and labeling all of the permutations in order,

We get the following sequence (ie, for n = 3):

"123"

"132"

"213"

"231"

"312"

"321"

Given n and k, return the kth permutation sequence.

## 题目大意：

给出n和k，求出n的全排列中字典序第k位的排列。

## 题目分析：

观察3个数的全排列，可见最高位分别是1,1,2,2,3,3，那么可以分为3组，第1组最高位为1，第二组最高位为2，第三组最高位为3，每一组内又分别是剩下两个数的按字典序的全排列，因此可以递归的缩小问题规模。

构造一个动态数组list，初始化为[1,2,…,n],并把n!个数分为n组，算出k在第几组及组内序号,按组号对应从list中取出这一位，然后更新k为组内的序号，直到list为空则找到答案。文字描述可能还是有些抽象，接下来举例说明。

例：n=4,k=15.

Step1.首先初始化动态数组list=[1,2,3,4],4个数的全排列有4!=24种，分为4组，则15在第2组种，组内序号为3（组号和组内序号都从0开始）则取出list的2号元素’3’加入答案中，并更新n=3,k=3.

Step2.list=[1,2,4],3个数的全排列有6种，分为3组，则3在第1组，组内序号为1.取出list的1号元素’2’，并更新n=2,k=1

Step3.list=[1,4]，2个数的全排列有2种，分为2组，则1在第1组，组内序号为0，取出list的1号元素’4’，并更新n=1,k=0

Step4.list=[1],1个数的排列只有1种，加入第0号元素’1’即可。

最后得到答案”3241”.

## 源码：（language：java）

public class Solution {

private String current = "";

public String getPermutation(int n, int k) {

List<Integer> list = new ArrayList<Integer>();

for(int i = 1;i<=n;i++)

list.add(i);

backtrack(list, n, k);

return current;

}

private void backtrack(List<Integer> list, int n, int k) {

if(list.isEmpty())

return;

else {

int groups = factorial(n-1);

int groupid = (k-1)/groups;

current+=String.valueOf(list.get(groupid));

list.remove(groupid);

backtrack(list, n-1, k-groupid\*groups);

}

}

public int factorial(int num){

int sum=1;

for(int i=1;i<=num;i++){

sum \*= i;

}

return sum;

}

}

## 成绩：

4ms,beats 5.39%,众数3ms,41.65%

## Cmershen的碎碎念：

似乎本题可以用之前求排列数里面的next\_permutation实现。还有就是本题的n!是现场算的，如果离线算出，应该还可以快1ms.

# Middle-题目94：322. Coin Change

## 题目原文：

You are given coins of different denominations and a total amount of money amount. Write a function to compute the fewest number of coins that you need to make up that amount. If that amount of money cannot be made up by any combination of the coins, return -1.

Example 1:

coins = [1, 2, 5], amount = 11

return 3 (11 = 5 + 5 + 1)

Example 2:

coins = [2], amount = 3

return -1.

## 题目大意：

假设你有若干面额的钱币，面额由数组coins给出，再给出一个总的钱币数amount。计算至少需要几枚钱币才能凑成amount？如果能，返回最小钱数，如果不能，返回-1.

例如：coins = [1, 2, 5], amount = 11

return 3 (11 = 5 + 5 + 1)

## 题目分析：

使用一个O(n2)的DP解决。设dp[i]代表钱数为i所需要的硬币数，先初始化dp[0]=0,dp[i]=MAX\_VALUE-1(为什么这里有个-1稍后说明)，然后转移关系是：

即已知钱数i所需要的硬币数，可以推出再加一枚硬币可以表示的钱数.最终dp[amount]即为所求。（这时候看出来为什么写了个-1吗？）

## 源码：（language：java）

public class Solution {

public int coinChange(int[] coins, int amount) {

int dp[] = new int[amount + 1];

for (int i = 1; i <= amount; i++)

dp[i] = Integer.MAX\_VALUE-1;

for (int i = 0; i <= amount; i++) {

for (int j = 0; j < coins.length; j++) {

if (i + coins[j] <= amount)

dp[i + coins[j]] = Math.min(dp[i + coins[j]], dp[i] + 1);

}

}

return dp[amount] == Integer.MAX\_VALUE-1 ? -1 : dp[amount];

}

}

## 成绩：

23ms,beats 83.94%,众数25ms,9.55%

## Cmershen的碎碎念：

读完代码之后应该能解释这个-1了。如果还没理解那么揭晓答案：计算机中用补码储存数字，所以如果不写-1，那么在第9行判断dp[i]+1的时候会造成溢出，导致变成MIN\_VALUE，那么dp[i+coins[j]]不管是多少都会被冲掉。

此外这题好像可以用母函数在O(n)复杂度内做，我组合数学学得不好，也没查到具体的ac代码。（这题又是那个fighter.io出的，貌似他出的题总离不开数学）

# Middle-题目95：222. Count Complete Tree Nodes

## 题目原文：

Given a complete binary tree, count the number of nodes.

## 题目大意：

给出一个完全二叉树，求节点数。

## 题目分析：

如果直接递归数节点，会超时。所以要考虑完全二叉树的性质：左右子树都是完全二叉树。所以分别求左右子树高度，如果相等则为满二叉树，如果不等则递归统计左右子树即可。

## 源码：（language：java）

public class Solution {

public int countNodes(TreeNode root) {

if(root == null)

return 0;

else {

int leftHeight = 0,rightHeight = 0;

TreeNode left = root,right = root;

while(left!=null) {

left=left.left;

leftHeight++;

}

while (right!=null) {

right = right.right;

rightHeight++;

}

if(leftHeight == rightHeight) // full binary tree

//return (int) (Math.pow(2, leftHeight)-1);

return (2<<(leftHeight-1)) - 1;

else

return 1+countNodes(root.left)+countNodes(root.right);

}

}

}

## 成绩：

112ms,beats 59.26%,众数116ms，3.89%

## cmershen的碎碎念：

其实完全二叉树还有一个性质，就是左右子树至少有一个是满二叉树。这样的话只需要一个递归即可。

# Middle-题目96：187. Repeated DNA Sequences

## 题目原文：

All DNA is composed of a series of nucleotides abbreviated as A, C, G, and T, for example: "ACGAATTCCG". When studying DNA, it is sometimes useful to identify repeated sequences within the DNA.

Write a function to find all the 10-letter-long sequences (substrings) that occur more than once in a DNA molecule.

For example,

Given s = "AAAAACCCCCAAAAACCCCCCAAAAAGGGTTT",

Return:

["AAAAACCCCC", "CCCCCAAAAA"].

## 题目大意：

给出一个DNA序列（由’A’,’C’,’G’,’T’组成的字符串），求其中所有的在该DNA序列中出现至少两次的10位子序列。

## 题目分析：

方法一：（朴素解法，基于哈希表）构建一个HashMap，其中Key是原序列的所有的10位子序列（空间复杂度O(n)），value是这个序列的出现次数。如果发现某个key出现次数为2则加入列表（超过2则跳过，否则列表会出现重复串）

方法二：（基于Bit manipulation，来自discuss中大神的算法）分别把’A’,’C’,’G’,’T’编码为00,01,10,11.先计算前9个字符的编码，然后从第10个字符开始，去掉最高位的编码（与0xfffff求与），再加上最后一位的编码，放入集合中，如果集合重复了，则加入最终返回的列表。

## 源码：（language：java）

方法一：

public class Solution {

public List<String> findRepeatedDnaSequences(String s) {

List<String> list = new ArrayList<String>();

HashMap<String, Integer> dict = new HashMap<String, Integer>();

for(int i = 0;i<=s.length()-10;i++) {

String substr = s.substring(i,i+10);

if(!dict.containsKey(substr)) {

dict.put(substr, 1);

}

else {

int value = dict.get(substr) + 1;

if(value>2)

continue;

dict.put(substr, value);

if(value == 2)

list.add(substr);

}

}

return list;

}

}

方法二：

public class Solution {

public List<String> findRepeatedDnaSequences(String DNA) {

ArrayList<String> res = new ArrayList<String>();

if(DNA.length()<10) return res;

HashSet<Integer> once = new HashSet<Integer>();

HashSet<Integer> twice = new HashSet<Integer>();

int[] map = new int[26];

map['A'-'A'] = 0;

map['C'-'A'] = 1;

map['G'-'A'] = 2;

map['T'-'A'] = 3;

int enc = 0;

for(int i=0; i<9; ++i){

enc <<=2;

enc |= map[DNA.charAt(i)-'A'];

}

for(int j=9; j<DNA.length(); ++j){

enc <<=2;

enc &= 0xfffff;

enc |= map[DNA.charAt(j)-'A'];

if(!once.add(enc) && twice.add(enc))

res.add(DNA.substring(j-9,j+1));

}

return res;

}

}

## 成绩：

方法一：40ms,beats 67.09%，众数38ms,5.66%

方法二：32ms,beats 95.90%

## cmershen的碎碎念：

第二种方法有两个巧妙的处理方法是值得学习的：一是enc &= 0xfffff;表示去掉原字符串最高位编码，二是if(!once.add(enc) && twice.add(enc))代表这个编码第二次出现（虽然这个我还不太理解）

# Middle-题目97：148. Sort List

## 题目原文：

Sort a linked list in O(n log n) time using constant space complexity.

## 题目大意：

对一个单链表排序，要求时间复杂度O(nlogn),空间复杂度O(1)

## 题目分析：

参考discuss中的算法，使用快排，取pivot为第一个节点值。

## 源码：（language：java）

public class Solution {

public ListNode sortList(ListNode h){

if(h == null || h.next == null)

return h;

/\*split into three list\*/

ListNode fakesmall = new ListNode(0), small = fakesmall;

ListNode fakelarge = new ListNode(0), large = fakelarge;

ListNode fakeequal = new ListNode(0), equal = fakeequal;

ListNode cur = h; // pivot is h.

while(cur != null){

if(cur.val < h.val){

small.next = cur;

small = small.next;

}

else if(cur.val == h.val){

equal.next = cur;

equal = equal.next;

}

else{

large.next = cur;

large = large.next;

}

cur = cur.next;

}

// put an end.

small.next = equal.next = large.next = null;

// merge them and return . merge reusing below one. merge for quicksort should be simplified.

return merge(merge(sortList(fakesmall.next), sortList(fakelarge.next)),fakeequal.next) ;

}

private ListNode merge(ListNode h, ListNode m){

ListNode fake = new ListNode(0), cur = fake;

while(h != null && m != null){

if(h.val < m.val){

cur.next = h;

h = h.next;

}

else{

cur.next = m;

m = m.next;

}

cur = cur.next;

}

cur.next = (h == null ? m : h);

return fake.next;

}

}

## 成绩：

8ms,beats 42.09%,众数8ms,31.36%

## cmershen的碎碎念：

数组的快排中pivot是使用三者取中法，而链表不是随机存取的，使用三者取中法要遍历一次链表，反而不能加速。

# Middle-题目98：316. Remove Duplicate Letters

## 题目原文：

Given a string which contains only lowercase letters, remove duplicate letters so that every letter appear once and only once. You must make sure your result is the smallest in lexicographical order among all possible results.

Example:

Given "bcabc"

Return "abc"

Given "cbacdcbc"

Return "acdb"

## 题目大意：

给出一个字符串，去掉其中重复的字母，并使得剩下的字符串是字典序最小的。

例如aba,可以去掉两个a中之一，得到的子串是”ab”或”ba”,要取字典序最小的”ab”.

## 题目分析：

参考discuss中的一个神解法，大致思路是使用两个指针模拟栈，具体细节有待研究。

## 源码：（language：java）

public class Solution {

public String removeDuplicateLetters(String s) {

/\*\*

\* First loop: use an array cnt[] to count the number of times

\* appeared for each letter in s.

\*

\* Second loop (Greedy): use a stack, pop() while (!stack.isEmpty()

\* && (sc = stack.peek()) >= c && cnt[sc] > 0)

\*/

int i, n = s.length();

int[] cnt = new int[128];

boolean[] inRes = new boolean[128]; // whether a char is in res[]

char[] res = s.toCharArray(); // simulate a stack

for (i = 0; i < n; i++)

cnt[res[i]]++;

char c, sc;

int end = -1;

// now cnt[c] means the remaining count of the char c

for (i = 0; i < n; i++) {

c = res[i];

if (inRes[c]) {

cnt[c]--;

continue;

}

while (end >= 0 && (sc = res[end]) >= c && cnt[sc] > 0) {

end--;

inRes[sc] = false;

}

res[++end] = c;

cnt[c]--;

inRes[c] = true;

}

return String.valueOf(res).substring(0, end + 1);

}

}

## 成绩：

3ms,beats 95.32%，众数7ms,9.99%

## cmershen的碎碎念：

度娘上各大博客中比较流行的是这个代码：

public class Solution {

public String removeDuplicateLetters(String s) {

int[] cnt = new int[26];

int pos = 0; // the position for the smallest s[i]

for (int i = 0; i < s.length(); i++) cnt[s.charAt(i) - 'a']++;

for (int i = 0; i < s.length(); i++) {

if (s.charAt(i) < s.charAt(pos)) pos = i;

if (--cnt[s.charAt(i) - 'a'] == 0) break;

}

return s.length() == 0 ? "" : s.charAt(pos) + removeDuplicateLetters(s.substring(pos + 1).replaceAll("" + s.charAt(pos), ""));

}

}

大致思路是每次去掉一个字符，取字典序较小的子串继续进行比较，但这个算法虽然是O(n)复杂度，但并不快(40ms+).因此还是想有时间认真研读一下那个3ms的解法。

# Middle-题目99：227. Basic Calculator II

## 题目原文：

Implement a basic calculator to evaluate a simple expression string.

The expression string contains only non-negative integers, +, -, \*, / operators and empty spaces . The integer division should truncate toward zero.

You may assume that the given expression is always valid.

Some examples:

"3+2\*2" = 7

" 3/2 " = 1

" 3+5 / 2 " = 5

## 题目大意：

计算一个表达式字符串的值，其中含有+,-,\*,/，空格和数字

## 题目分析：

既然用JavaScript的eval水过去会错，那就认真算一下。

首先没有括号，不用考虑括号的问题。那么从左到右先乘除后加减即可。遍历字符串两遍。

## 源码：（language：java）

public class Solution {

public int calculate(String s) {

s=s.replace(" ","");

String[] nums = s.split("[\\+\\-\\\*\\/]");

List<Integer> numberList = new ArrayList<>();

for(String num : nums)

numberList.add(Integer.parseInt(num));

int index = 0;

for (int i = 0; i<s.length(); i++) {

char ch = s.charAt(i);

if (ch == '+' || ch == '-')

index++;

else if (ch == '\*' || ch == '/') {

int op1 = numberList.get(index);

int op2 = numberList.get(index+1);

numberList.remove(index);

numberList.remove(index);

numberList.add(index, operate(op1,ch,op2));

}

}

int result = numberList.get(0); index = 1;

for(int i = 0;i<s.length();i++) {

char ch = s.charAt(i);

if(ch == '+' || ch == '-')

result = operate(result, ch, numberList.get(index++));

}

return result;

}

private Integer operate(int a, char op, int b) {

if(op=='+')

return a+b;

else if(op=='-')

return a-b;

else if(op=='\*')

return a\*b;

else

return a/b;

}

}

## 成绩：

90ms,beats 13.83%,众数26ms,3.63%

## Cmershen的碎碎念：

主要浪费时间还是在split上。

# Middle-题目100：18. 4Sum

## 题目原文：

Given an array S of n integers, are there elements a, b, c, and d in S such that a + b + c + d = target? Find all unique quadruplets in the array which gives the sum of target.

Note:

Elements in a quadruplet (a,b,c,d) must be in non-descending order. (ie, a ≤ b ≤ c ≤ d)

The solution set must not contain duplicate quadruplets.

## 题目大意：

给出一个数组和目标值target，找出4个数使得和值为target，求出所有解。

## 题目分析：

既然不要求保持顺序，那么用ksum的通法，先排序再固定k-2个指针即可

## 源码：（language：java）

public class Solution {

public List<List<Integer>> fourSum(int[] nums, int target) {

List<List<Integer>> list = new ArrayList<List<Integer>>();

Arrays.sort(nums);

for(int i = 0;i<nums.length-3;i++) {

for(int j = nums.length-1;j-i>=3;j--) {

int sum1 = nums[i]+nums[j];

int m1 = i+1,m2 = j-1;

while(m1<m2) {

int sum = sum1+nums[m1]+nums[m2];

if(sum == target) {

List<Integer> sublist = new ArrayList<Integer>();

sublist.add(nums[i]);

sublist.add(nums[m1]);

sublist.add(nums[m2]);

sublist.add(nums[j]);

if(!list.contains(sublist))

list.add(sublist);

m1++;

}

else if(sum > target)

m2--;

else // sum < target

m1++;

}

}

}

return list;

}

}

## 成绩：

56ms,beats 59.49%,众数57ms,4.51%

## Cmershen的碎碎念：

本题的去重处理没有做，而是直接遍历list来排除重复解的，否则还能更快一些。

# Middle-题目101：332. Reconstruct Itinerary

## 题目原文：

Given a list of airline tickets represented by pairs of departure and arrival airports [from, to], reconstruct the itinerary in order. All of the tickets belong to a man who departs from JFK. Thus, the itinerary must begin with JFK.

Note:

If there are multiple valid itineraries, you should return the itinerary that has the smallest lexical order when read as a single string. For example, the itinerary ["JFK", "LGA"] has a smaller lexical order than ["JFK", "LGB"].

All airports are represented by three capital letters (IATA code).

You may assume all tickets form at least one valid itinerary.

Example 1:

tickets = [["MUC", "LHR"], ["JFK", "MUC"], ["SFO", "SJC"], ["LHR", "SFO"]]

Return ["JFK", "MUC", "LHR", "SFO", "SJC"].

Example 2:

tickets = [["JFK","SFO"],["JFK","ATL"],["SFO","ATL"],["ATL","JFK"],["ATL","SFO"]]

Return ["JFK","ATL","JFK","SFO","ATL","SFO"].

Another possible reconstruction is ["JFK","SFO","ATL","JFK","ATL","SFO"]. But it is larger in lexical order.

## 题目大意：

给出一系列机票的列表，用一个二元组[from,to]表示。重新构造出一个从”JFK”出发的旅行线路。如果有多个解，求字典序最小的。

## 题目分析：

先将机票列表构造成邻接表，其中邻接列表部分使用优先队列维护字典序，然后从”JFK”点开始搜索，每次取出优先队列的队首节点（字典序最小）并再次搜索，直到某个节点的优先队列为空时将当前节点加入线路列表的头部（因为最后一个节点也是最后一个旅行到的节点，应是在表尾的）

## 源码：（language：java）

public class Solution {

LinkedList<String> res;

Map<String, PriorityQueue<String>> mp;

public List<String> findItinerary(String[][] tickets) {

res = new LinkedList<String>();

mp = new HashMap<String, PriorityQueue<String>>();

for (String[] ticket : tickets) {

if (!mp.containsKey(ticket[0])) {

mp.put(ticket[0], new PriorityQueue<String>());

}

mp.get(ticket[0]).offer(ticket[1]);

}

dfs("JFK");

return res;

}

public void dfs(String cur) {

while (mp.containsKey(cur) && !mp.get(cur).isEmpty()) {

dfs(mp.get(cur).poll());

}

res.addFirst(cur);

}

}

## 成绩：

12ms,beats 81.78%,众数14ms,12.37%

## Cmershen的碎碎念：

一开始使用的朴素的dfs解法，每次搜到解则比较字典序，并更新列表，但会超时。而这个算法的本质是贪心法，因为每次拿出字典序最小的点时，如果有解则一定是最优解。（假设邻接表中某个节点对应优先队列是(…p1,…p2…),p1<p2,那么取出p1后找到一组解，如果p2也存在解，因为此时加入最终列表的是p1，这个解一定比p2对应的解小。

那么还剩下最后一个疑问，算法在优先队列为空时跳出循环，把当前节点加入解序列中。为什么一个点对应的出边能全部遍历一遍，则这个点一定有解？因为这道题是新题，网上对此题的讨论还不太多。

此外有一个巧妙之处在于结果使用了LinkedList，因为是从尾向头加入结果的，所以需要LinkedList的addFirst函数。

# Middle-题目102：150. Evaluate Reverse Polish Notation

## 题目原文：

Evaluate the value of an arithmetic expression in Reverse Polish Notation.

Valid operators are +, -, \*, /. Each operand may be an integer or another expression.

## 题目大意：

计算逆波兰式的值。

## 题目分析：

在《数据结构》课的堆栈部分，我们学习过如果求逆波兰式的值，在此复习一下：

初始化一个堆栈，然后遍历逆波兰式数组，如果是操作数则入栈，如果是操作符则把两个操作数弹出来进行运算，运算结果压入堆栈，最终栈中只有一个元素，即表达式的值。

## 源码：（language：java）

public class Solution {

public int evalRPN(String[] tokens) {

Stack<Integer> stack = new Stack<Integer>();

for(String token : tokens) {

if(token.equals("+") || token.equals("-") || token.equals("\*") || token.equals("/")) {

int num1 = stack.pop();

int num2 = stack.pop();

stack.push(operate(num2,token,num1));

}

else

stack.push(Integer.parseInt(token));

}

return stack.peek();

}

private int operate(int a, String op, int b) {

if(op.equals("+"))

return a+b;

else if(op.equals("-"))

return a-b;

else if(op.equals("\*"))

return a\*b;

else {

return a/b;

}

}

}

## 成绩：

17ms,beats 39.27%,众数16ms,17.94%

## cmershen的碎碎念：

逆波兰式是一种没有括号的，但运算结果唯一的表达式，对计算机来说要比中缀表达式容易理解很多。关于中缀表达式转逆波兰式的算法见百度百科，之前我们的课程设计也做过这个。

# Middle-题目103：221. Maximal Square

## 题目原文：

Given a 2D binary matrix filled with 0's and 1's, find the largest square containing all 1's and return its area.

For example, given the following matrix:

1 0 1 0 0

1 0 1 1 1

1 1 1 1 1

1 0 0 1 0

Return 4.

## 题目大意：

给出一个0-1矩阵，求其中最大的由1组成的正方形，返回其面积。

## 题目分析：

还是Dp问题，设dp[i][j]为右下角为(i,j)的正方形的**边长**，则有如下转移方程：

dp[i][j] = min(dp[i-1][j],dp[i-1][j-1],dp[i][j-1])+1

初始化第一行（列）的dp[0][i]和dp[j][0]为矩阵的原值。

这个转移方程用文字比较难以描述，在纸上画一下三种情况就很好理解了。

## 源码：（language：java）

public class Solution {

public int maximalSquare(char[][] matrix) {

int row = matrix.length;

if(row == 0)

return 0;

int col = matrix[0].length;

int maxSquare = 0;

int[][] dp = new int[row][col];

for(int i = 0 ;i<col;i++) {

dp[0][i]=matrix[0][i]-'0';

if(matrix[0][i] == '1')

maxSquare = 1;

}

for(int i = 0;i<row;i++) {

dp[i][0]=matrix[i][0]-'0';

if(matrix[i][0] == '1')

maxSquare = 1;

}

for(int i = 1;i<row;i++) {

for(int j=1;j<col;j++) {

if(matrix[i][j] == '1')

dp[i][j] = Math.min(dp[i-1][j], Math.min(dp[i-1][j-1], dp[i][j-1]))+1;

if(dp[i][j] > maxSquare)

maxSquare = dp[i][j];

}

}

return maxSquare\*maxSquare;

}

}

## 成绩：

14ms,beats 58.15%,众数17ms,11.81%

# Middle-题目104：43. Multiply Strings

## 题目原文：

Given two numbers represented as strings, return multiplication of the numbers as a string.

Note: The numbers can be arbitrarily large and are non-negative.

## 题目大意：

给出两个用字符串表示的数字，求他们的乘积（用字符串表示）。

数字可以任意大，且是非负数。

## 题目分析：

python的int是无限精度的，所以直接水过去就行了。

## 源码：（language：python）

class Solution(object):

def multiply(self, num1, num2):

"""

:type num1: str

:type num2: str

:rtype: str

"""

return str(int(num1)\*int(num2))

## 成绩：

56ms,beats 77.45%,众数52ms,9.18%

## cmershen的碎碎念：

java中也可以用biginteger类实现，但好像不是很快。

**需记住，python是强类型语言，实现本题的时候必须类型转换！（我以前一直以为脚本语言都是弱类型的）**

# Middle-题目105：93. Restore IP Addresses

## 题目原文：

Given a string containing only digits, restore it by returning all possible valid IP address combinations.

For example:

Given "25525511135",

return ["255.255.11.135", "255.255.111.35"]. (Order does not matter)

## 题目大意：

给出一个数字组成的字符串，求出能组成的合法IP地址的所有组合。

## 题目分析：

就是深搜，使用回溯法实现。使用参数dots记录加入了几个点，如果加入了3个点且剩下的字符串是合理的IP数字（无前导0且小于256）就加入最终的列表。这里面有一个剪枝处理是，剩下的数字的长度不大于剩下的点数\*3（即形如.255.1000的IP一定不合法）。

比较麻烦的是对0的处理，如果两个点之间的字符串以0开始，则必须是0，”02”,“00”……都不合法。

## 源码：（language：java）

public class Solution {

public List<String> restoreIpAddresses(String s) {

List<String> ipList = new ArrayList<String>();

backtrack(ipList,"", 0, s);

return ipList;

}

private void backtrack(List<String> ipList, String currentIP, int dots, String rest) {

if(dots==3 && isAvailableIP(rest)) {

currentIP+=rest;

ipList.add(currentIP);

return;

}

else if(rest.length() <= (4-dots)\*3) {

for(int i =1;i<=Math.min(rest.length()-1, 3);i++) {

String temp = rest.substring(0,i);

if(isAvailableIP(temp)) {

backtrack(ipList,currentIP+temp+".", dots+1, rest.substring(i));

}

}

}

}

private boolean isAvailableIP(String num) {

if(num.startsWith("0")) {

return num.equals("0"); // 02,00... aren't available.

}

else {

return Integer.parseInt(num)<256;

}

}

}

## 成绩：

3ms,beats 89.21%,众数4ms，33.87%

## cmershen的碎碎念：

此题真的是我原创的，而且是提交到discuss的第一道题。还有一个vote呢~~我想是因为那个对字符串长度的剪枝处理加速了吧。

# Middle-题目106：5. Longest Palindromic Substring

## 题目原文：

Given a string S, find the longest palindromic substring in S. You may assume that the maximum length of S is 1000, and there exists one unique longest palindromic substring.

## 题目大意：

求出字符串S的最长回文子串。

## 题目分析：

使用了discuss的神算法，我根本看不懂……求大神解答……（好像还是个O(n2)的算法）大概是每向右增加一个字符，就判断能不能向左延伸之类的。

本题有一个线性算法，叫manacher算法，参见<https://en.wikipedia.org/wiki/Longest_palindromic_substring#CITEREFManacher1975>

## 源码：（language：java）

方法一：

public class Solution {

public String longestPalindrome(String s) {

char[] ca = s.toCharArray();

int rs = 0, re = 0;

int max = 0;

for(int i = 0; i < ca.length; i++) {

if(isPalindrome(ca, i - max - 1, i)) {

rs = i - max - 1; re = i;

max += 2;

} else if(isPalindrome(ca, i - max, i)) {

rs = i - max; re = i;

max += 1;

}

}

return s.substring(rs, re + 1);

}

private boolean isPalindrome(char[] ca, int s, int e) {

if(s < 0) return false;

while(s < e) {

if(ca[s++] != ca[e--]) return false;

}

return true;

}

}

方法二：（manacher算法，改编自wikipaedia上的代码）

public class Solution {

public String longestPalindrome(String s) {

if (s==null || s.length()==0)

return "";

char[] s2 = addBoundaries(s.toCharArray());

int[] p = new int[s2.length];

int c = 0, r = 0; // Here the first element in s2 has been processed.

int m = 0, n = 0; // The walking indices to compare if two elements are the same

for (int i = 1; i<s2.length; i++) {

if (i>r) {

p[i] = 0; m = i-1; n = i+1;

} else {

int i2 = c\*2-i;

if (p[i2]<(r-i)) {

p[i] = p[i2];

m = -1; // This signals bypassing the while loop below.

} else {

p[i] = r-i;

n = r+1; m = i\*2-n;

}

}

while (m>=0 && n<s2.length && s2[m]==s2[n]) {

p[i]++; m--; n++;

}

if ((i+p[i])>r) {

c = i; r = i+p[i];

}

}

int len = 0; c = 0;

for (int i = 1; i<s2.length; i++) {

if (len<p[i]) {

len = p[i]; c = i;

}

}

char[] ss = Arrays.copyOfRange(s2, c-len, c+len+1);

return String.valueOf(removeBoundaries(ss));

}

private char[] addBoundaries(char[] cs) {

if (cs==null || cs.length==0)

return "||".toCharArray();

char[] cs2 = new char[cs.length\*2+1];

for (int i = 0; i<(cs2.length-1); i = i+2) {

cs2[i] = '|';

cs2[i+1] = cs[i/2];

}

cs2[cs2.length-1] = '|';

return cs2;

}

private char[] removeBoundaries(char[] cs) {

if (cs==null || cs.length<3)

return "".toCharArray();

char[] cs2 = new char[(cs.length-1)/2];

for (int i = 0; i<cs2.length; i++) {

cs2[i] = cs[i\*2+1];

}

return cs2;

}

}

## 成绩：

方法一：9ms,beats 96.88%,众数13ms,4.50%

方法二：11ms,beats 92.14%

## cmershen的碎碎念：

Manacher, Glenn (1975), "A new linear-time "on-line" algorithm for finding the smallest initial palindrome of a string", Journal of the ACM 22 (3): 346–351, doi:10.1145/321892.321896.

这篇论文详细论证了Manacher算法，如果无聊可以研究一下。

方法二不如方法一快是因为数据量不够大（S.length<10^3）,且此题个人认为应归为hard。

# Middle-题目107：61. Rotate List

## 题目原文：

Given a list, rotate the list to the right by k places, where k is non-negative.

For example:

Given 1->2->3->4->5->NULL and k = 2,

return 4->5->1->2->3->NULL.

## 题目大意：

把一个链表循环右移K位。

## 题目分析：

先把链表连成环（尾节点连到head上），并统计节点个数count，然后从head开始把第count-k%count个节点拆下来就行了。

## 源码：（language：c）

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* struct ListNode \*next;

\* };

\*/

struct ListNode\* rotateRight(struct ListNode\* head, int k) {

if(!head || !head->next)

return head;

struct ListNode\* end;

int count=1;

for(end = head; end->next; end = end->next)

count++;

end->next = head;

struct ListNode \*p1=head,\*p2=head->next;

for(int i=1;i<count-k%count;i++) {

p1=p1->next;

p2=p2->next;

}

p1->next = NULL;

return p2;

}

## 成绩：

4ms,beats 6.02%，众数4ms,93.98%

## cmershen的碎碎念：

本题有一个陷阱在于，如果直接从环上找第k个点是错误的，因为这里的k可能远大于链表长度，而对环来说移动一圈相当于没动，所以取余即可。

# Middle-题目108：79. Word Search

## 题目原文：

Given a 2D board and a word, find if the word exists in the grid.

The word can be constructed from letters of sequentially adjacent cell, where "adjacent" cells are those horizontally or vertically neighboring. The same letter cell may not be used more than once.

For example,

Given board =

[

['A','B','C','E'],

['S','F','C','S'],

['A','D','E','E']

]

word = "ABCCED", -> returns true,

word = "SEE", -> returns true,

word = "ABCB", -> returns false.

## 题目大意：

给出一个矩阵，和一个单词，判断单词是否能由矩阵中一条路径连接出来。

## 题目分析：

一个简单的dfs，教科书式的dfs写法即可，搜到字母错误或者搜到数组边缘则退出，否则每次向四个方向搜索，注意这里的dfs是有返回值的，返回该条件下是否有解。

## 源码：（language：java）

public class Solution {

public boolean exist(char[][] board, String word) {

int row = board.length;

if(row==0)

return false;

int col = board[0].length;

boolean[][] visited = new boolean[row][col];

for(int i = 0;i<row;i++) {

for(int j=0;j<col;j++) {

if(word.charAt(0) == board[i][j]) {

if(dfs(board, i, j, visited, word, 0))

return true;

}

}

}

return false;

}

private boolean dfs(char[][] board, int i, int j, boolean[][] visited, String word, int index) {

if(index == word.length())

return true;

else if(i<0 || i>board.length-1 || j<0 || j>board[0].length-1 || word.charAt(index)!=board[i][j] || visited[i][j])

return false;

else {

visited[i][j]=true;

boolean result = dfs(board, i-1, j, visited, word, index+1) || dfs(board, i+1, j, visited, word, index+1) || dfs(board, i, j-1, visited, word, index+1) || dfs(board, i, j+1, visited, word, index+1);

visited[i][j] = false;

return result;

}

}

}

## 成绩：

13ms,beats 58.16%,众数14ms,14.58%

## cmershen的碎碎念：

本题基本上是一个教科书式的dfs，区别是dfs带有返回值判断是否有解。

# Middle-题目109：143. Reorder List

## 题目原文：

Given a singly linked list L: L0→L1→…→Ln-1→Ln,

reorder it to: L0→Ln→L1→Ln-1→L2→Ln-2→…

## 题目大意：

给出一个链表，要求按“头尾相接”的方式重排。

## 题目分析：

没有按照题目要求去做，先用一个数组保存下所有节点的值，再依次填上。

## 源码：（language：java）

public class Solution {

public void reorderList(ListNode head) {

List<Integer> list = new ArrayList<Integer>();

for(ListNode node = head;node!=null;node=node.next)

list.add(node.val);

int i = 0,j = list.size()-1;

ListNode node = head;

boolean addi = true;

while (i <= j) {

node.val = addi?list.get(i++):list.get(j--);

addi=!addi;

node=node.next;

}

}

}

## 成绩：

8ms,beats 3.74%,众数3ms,51.93%

## cmershen的碎碎念：

如果此题改用LinkedList，会超时。

# Middle-题目110：2. Add Two Numbers

## 题目原文：

You are given two linked lists representing two non-negative numbers. The digits are stored in reverse order and each of their nodes contain a single digit. Add the two numbers and return it as a linked list.

Input: (2 -> 4 -> 3) + (5 -> 6 -> 4)

Output: 7 -> 0 -> 8

## 题目大意：

给出两个数，用链表逆序的存储每个数位，求他们的和，并也用链表逆序存储数位。

## 题目分析：

方法一：用BigInteger储存两个数字，再加起来，再生成对应链表。

方法二：模拟整数竖式算法（好在是逆序的， 正序就麻烦了）

## 源码：（language：java/c）

方法一：

import java.math.BigInteger;

public class Solution {

public ListNode addTwoNumbers(ListNode l1, ListNode l2) {

BigInteger num1 = BigInteger.ZERO;

BigInteger num2 = BigInteger.ZERO;

BigInteger temp = BigInteger.ONE;

while(l1!=null) {

//num1+=temp\*l1.val;

num1=num1.add(temp.multiply(BigInteger.valueOf(l1.val)));

temp=temp.multiply(BigInteger.TEN);

l1=l1.next;

}

temp=BigInteger.ONE;

while(l2!=null) {

num2=num2.add(temp.multiply(BigInteger.valueOf(l2.val)));

temp=temp.multiply(BigInteger.TEN);

l2=l2.next;

}

BigInteger sum = num1.add(num2);

if(sum.compareTo(BigInteger.ZERO)==0)

return new ListNode(0);

else {

ListNode l = new ListNode(sum.mod(BigInteger.TEN).intValue());

ListNode p=l;

while(sum.compareTo(BigInteger.ZERO)!=0) {

sum=sum.divide(BigInteger.TEN);

if(sum.compareTo(BigInteger.ZERO)!=0) {

p.next = new ListNode(sum.mod(BigInteger.TEN).intValue());

p=p.next;

}

}

return l;

}

}

}

方法二：(from discuss)

struct ListNode\* addTwoNumbers(struct ListNode\* l1, struct ListNode\* l2) {

struct ListNode\* result = l1 ? l1 : l2;

struct ListNode\* carry = l1 ? l2 : l1;

struct ListNode\* node = result;

struct ListNode\* tmp;

int c = 0;

int s;

while (l1 || l2) {

s = c;

if (l1) { s += l1->val; l1=l1->next; }

if (l2) { s += l2->val; l2=l2->next; }

c = s > 9 ? 1 : 0;

node->val = c ? s - 10 : s;

if (l1) {

node = node->next = l1;

} else if (l2) {

node = node->next = l2;

} else {

node->next = NULL;

}

}

if (c) {

carry->val = c;

node->next = carry;

node = node->next;

}

node->next = NULL;

return result;

}

## 成绩：

方法一：28ms,beats 1.57%,众数4ms,63.50%

方法二：20ms,beats 18.86%,众数20ms,68.68%

# Middle-题目111：54. Spiral Matrix

## 题目原文：

Given a matrix of m x n elements (m rows, n columns), return all elements of the matrix in spiral order.

For example,

Given the following matrix:

[

[ 1, 2, 3 ],

[ 4, 5, 6 ],

[ 7, 8, 9 ]

]

You should return [1,2,3,6,9,8,7,4,5].

## 题目大意：

按顺时针螺旋顺序遍历二维数组。

## 题目分析：

跟[Middle-题目29](#_Middle-题目29：59._Spiral_Matrix)很像，在前面题基础上简单改一下代码即可。一个是读数组一个是写数组，所以读数组要使用一个visited[][]数组维护是否读过，而写数组只要判断是否为0即可。

## 源码：（language：java）

public class Solution {

private final int RIGHT = 1;

private final int DOWN = 2;

private final int LEFT = 3;

private final int UP = 4;

public List<Integer> spiralOrder(int[][] matrix) {

List<Integer> list = new ArrayList<Integer>();

int row = matrix.length;

if(row==0)

return list;

int col = matrix[0].length;

boolean[][] visited = new boolean[row][col];

int count = 0,direction = RIGHT,i=0,j=0;

while (count < row \* col) {

list.add(matrix[i][j]);

visited[i][j] = true;

switch (direction) {

case RIGHT:

if (j==col-1 || visited[i][j+1]) {

direction=DOWN;

i++;

}

else

j++;

break;

case DOWN:

if (i==row-1 || visited[i+1][j]) {

direction = LEFT;

j--;

}

else

i++;

break;

case LEFT:

if (j==0 || visited[i][j-1]) {

direction = UP;

i--;

}

else

j--;

break;

case UP:

if (visited[i-1][j]) {

direction = RIGHT;

j++;

}

else

i--;

break;

default:

break;

}

count++;

}

return list;

}

}

## 成绩：

1ms,beats 3.18%,众数1ms,71.07%

# Middle-题目112：152. Maximum Product Subarray

## 题目原文：

Find the contiguous subarray within an array (containing at least one number) which has the largest product.

For example, given the array [2,3,-2,4],

the contiguous subarray [2,3] has the largest product = 6.

## 题目大意：

寻找最大乘积的连续子数组。

## 题目分析：

使用两个dp数组dpmax和dpmin，其中dpmax[i]和dpmin[i]分别表示以i为结尾的子数组的最大、最小乘积。那么初始化dpmax[0]和dpmin[0]均为nums[0]，转移方程如下：

dpmax[i]=max(nums[i],dpmax[i-1]\*nums[i],dpmin[i-1]\*nums[i])

dpmin[i]=min(nums[i],dpmax[i-1]\*nums[i],dpmin[i-1]\*nums[i])

这是因为nums[i]可能是正数也可能是负数，如果是负数且前一项的最小积是负数，则会变成最大乘积，而如果最大乘积和最小乘积都是负数或0，且nums[i]是正数，则它本身是最大乘积。所以最大值和最小值可能有三种情况。

## 源码：（language：java）

public class Solution {

public int maxProduct(int[] nums) {

int length = nums.length;

if(length==0)

return 0;

int[] dpmax = new int[length];

int[] dpmin = new int[length];

dpmax[0] = nums[0];

dpmin[0] = nums[0];

int max = dpmax[0];

for(int i = 1;i<length;i++) {

dpmax[i] = max(nums[i],dpmax[i-1]\*nums[i],dpmin[i-1]\*nums[i]);

dpmin[i] = min(nums[i],dpmax[i-1]\*nums[i],dpmin[i-1]\*nums[i]);

if(dpmax[i]>max)

max=dpmax[i];

}

return max;

}

private int max(int a,int b,int c) {

return a>b?(a>c?a:c):(b>c?b:c);

}

private int min(int a,int b,int c) {

return a<b?(a<c?a:c):(b<c?b:c);

}

}

## 成绩：

5ms,beats 10.47%,众数5ms,36.73%

## cmershen 的碎碎念：

1. 本题如果用一个dp数组，即dp[i][j]维护从i~j的最大乘积，也能实现但会超时。
2. 对nums很长的情况下，可能会涉及到BigInteger高精度计算，题中没有超过int范围，故test case不够严密。
3. 这是第一次遇到同时维护多个dp数组的题，也是拓宽了思路。

# Middle-题目113：324. Wiggle Sort II

## 题目原文：

Given an unsorted array nums, reorder it such that nums[0] < nums[1] > nums[2] < nums[3]....

Example:

(1) Given nums = [1, 5, 1, 1, 6, 4], one possible answer is [1, 4, 1, 5, 1, 6].

(2) Given nums = [1, 3, 2, 2, 3, 1], one possible answer is [2, 3, 1, 3, 1, 2].

## 题目大意：

对一个数组按摇摆顺序排序，即nums[0] < nums[1] > nums[2] < nums[3]…

## 题目分析：

方法一：O(nlogn)时间复杂度，O(n)空间复杂度：先对原数组排序，然后取两个指针s和t分别指向中点和尾节点，开一个与原数组等长的新数组，从头开始填，如果下标为偶数从s开始向左填，如果为奇数从t开始向左填，再覆盖回去。

方法二：O(n)时间复杂度，O(1)空间复杂度：来自discuss中的神算法，有待以后研究。

## 源码：（language：java/cpp）

方法一：

public class Solution {

public void wiggleSort(int[] nums) {

Arrays.sort(nums);

int[] temp = new int[nums.length];

int s = (nums.length + 1)/2, t = nums.length;

for (int i = 0; i < nums.length; i++) {

temp[i] = (i%2== 0) ? nums[--s] : nums[--t] ;

}

for (int i = 0; i < nums.length; i++)

nums[i] = temp[i];

}

}

方法二：

class Solution {

public:

void wiggleSort(vector<int>& nums) {

int n = nums.size();

// Find a median.

auto midptr = nums.begin() + n / 2;

nth\_element(nums.begin(), midptr, nums.end());

int mid = \*midptr;

// Index-rewiring.

#define A(i) nums[(1+2\*(i)) % (n|1)]

// 3-way-partition-to-wiggly in O(n) time with O(1) space.

int i = 0, j = 0, k = n - 1;

while (j <= k) {

if (A(j) > mid)

swap(A(i++), A(j++));

else if (A(j) < mid)

swap(A(j), A(k--));

else

j++;

}

}

};

## 成绩：

方法一：7ms,beats 52.74%,众数7ms,35.82%

方法二：56ms,beats 71.59%,众数56ms,27.79%

## cmershen的碎碎念：

一开始想到的是O(n2)的方法，即来回的冒泡排序，但发现是不对的。看到提示中说使用中位数，就想到了方法一，这样可以保证左半边永远小于右半边，故能确保小于大于交替出现。而这道题中似乎还涉及了一个O(n)求数组中位数的算法，这个好像跟之前做过的求数组第K大的数是一样的。

# Middle-题目114：3. Longest Substring Without Repeating Characters

## 题目原文：

Given a string, find the length of the longest substring without repeating characters. For example, the longest substring without repeating letters for "abcabcbb" is "abc", which the length is 3. For "bbbbb" the longest substring is "b", with the length of 1.

## 题目大意：

求一个字符串的没有重复字符的最长子串的长度。（不一定连续）

## 题目分析：

（算法思路来自度娘）定义一个布尔型数组sign[256]，记录每个字符是否出现过。然后两个指针i和j都从头开始，先让j向右移动，直到第一次出现重复元素时，移动i至排除掉这个重复元素，再移动j，这样一直维护着以j为结尾的最长子串，所以能得到最优解。

## 源码：（language：java）

public class Solution {

public int lengthOfLongestSubstring(String s) {

int i,j,max=0;

boolean[] sign = new boolean[256];

for(i=0,j=0;j<s.length();j++) {

while(sign[s.charAt(j)])

sign[s.charAt(i++)] = false;

sign[s.charAt(j)] = true;

max=Math.max(max,j-i+1);

}

return max;

}

}

## 成绩：

5ms,beats 91.60%，众数18ms，8.16%

## Cmershen的碎碎念：

该算法只需线性复杂度，既优雅又易于理解，值得学习！

# Middle-题目115：71. Simplify Path

## 题目原文：

Given an absolute path for a file (Unix-style), simplify it.

For example,

path = "/home/", => "/home"

path = "/a/./b/../../c/", => "/c"

## 题目大意：

化简一个Unix格式的路径。

## 题目分析：

显然用堆栈。根据hint里面提到的边界情况做以下处理：

1. 对多个/和.不作处理
2. 对..则弹出栈顶元素，如果栈空（已经是根路径）则不处理
3. 将两个/之间的子串压入堆栈

最后遍历栈，组成最后的目录。如果栈空则输出’/’。

## 源码：（language：java）

public class Solution {

public String simplifyPath(String path) {

String[] paths = path.split("/");

List<String> list = new LinkedList<String>();

Stack<String> stack = new Stack<String>();

for (String str : paths) {

if (str.equals(""))

continue;

else if (str.equals("..")) {

if(!stack.isEmpty())

stack.pop();

continue;

}

else if (str.equals("."))

continue;

else

stack.add(str);

}

String result = "";

for(int i = 0;i<stack.size();i++) {

result=result+"/"+stack.get(i);

}

if(result=="")

result+="/";

return result;

}

}

## 成绩：

12ms,beats 44.53%,众数12ms,18.84%

# Middle-题目116：224. Basic Calculator

## 题目原文：

Implement a basic calculator to evaluate a simple expression string.

The expression string may contain open ( and closing parentheses ), the plus + or minus sign -, non-negative integers and empty spaces .

You may assume that the given expression is always valid.

## 题目大意：

计算一个表达式的值，包括+,-,和括号还有空格。

## 题目分析：

这回用eval水过去。

## 源码：（language：JavaScript）

var calculate = function(s) {

return eval(s)

};

## 成绩：

140ms,beats 78.13%,众数176ms,184ms,9.38%

# Middle-题目117：98. Validate Binary Search Tree

## 题目原文：

Given a binary tree, determine if it is a valid binary search tree (BST).

## 题目大意：

给出一个二叉树，判断是不是合法的二叉搜索树。

## 题目分析：

看中序遍历是否递增。

## 源码：（language：java）

public class Solution {

public boolean isValidBST(TreeNode root) {

List<Integer> list = new ArrayList<Integer>();

inOrderVisit(list,root);

return isSortedList(list);

}

private void inOrderVisit(List<Integer> list, TreeNode node) {

if(node!=null) {

inOrderVisit(list, node.left);

list.add(node.val);

inOrderVisit(list, node.right);

}

}

private boolean isSortedList(List<Integer> list) {

if(list.size()<2)

return true;

else {

for(int i = 0;i<list.size()-1;i++) {

if(list.get(i)>=list.get(i+1))

return false;

}

return true;

}

}

}

## 成绩：

4ms,beats 12.79%,众数1ms,72.10%

# Middle-题目118：210. Course Schedule II

## 题目原文：

There are a total of n courses you have to take, labeled from 0 to n - 1.

Some courses may have prerequisites, for example to take course 0 you have to first take course 1, which is expressed as a pair: [0,1]

Given the total number of courses and a list of prerequisite pairs, return the ordering of courses you should take to finish all courses.

There may be multiple correct orders, you just need to return one of them. If it is impossible to finish all courses, return an empty array.

## 题目大意：

在[Middle-题目86](#_Middle-题目86：207._Course_Schedule)的基础上，求修课的序列。如果不可能修完所有课程，返回一个空序列。

## 题目分析：

跟上一题差不多，这次是求拓扑序列，但返回值有点奇怪，不知道empty array为什么有时候指的是new int[1]有时候又是new int[0]

## 源码：（language：java）

public class Solution {

public int[] findOrder(int numCourses, int[][] prerequisites) {

if(numCourses==1)

return new int[1];

Map<Integer, Set<Integer>> adjList = new HashMap<Integer,Set<Integer>>();

int[] indegree = new int[numCourses];

for(int i = 0; i<numCourses;i++)

adjList.put(i, new HashSet<Integer>());

for(int[] edge : prerequisites) {

if(!adjList.get(edge[1]).contains(edge[0])) {

adjList.get(edge[1]).add(edge[0]);

indegree[edge[0]]++;

}

}

int[] order = new int[numCourses];

int i=0;

int topoVertical = findZeroIndegree(indegree);

while(topoVertical!=-1) {

indegree[topoVertical] = -1;

for(Integer vertical : adjList.get(topoVertical)) {

if(indegree[vertical]!=-1)

indegree[vertical]--;

}

adjList.remove(topoVertical);

order[i++]=topoVertical;

topoVertical = findZeroIndegree(indegree);

}

if(i==numCourses)

return order;

else

return new int[0];

}

private int findZeroIndegree(int[] indegree) {

for(int i = 0;i<indegree.length;i++)

if(indegree[i]==0) {

return i;

}

return -1;

}

}

## 成绩：

37ms,beats 21.31%,众数10ms,8.81%

## Cmershen的碎碎念：

这道题我觉得oj的判题怪怪的，因为一直提示Special judge: No expected output available. 不知道为什么要返回new int[1]而不是new int[0]或null。所以此题的ac率超低，但难度应该跟前一题是一样的。

# Middle-题目119：127. Word Ladder

## 题目原文：

Given two words (beginWord and endWord), and a dictionary's word list, find the length of shortest transformation sequence from beginWord to endWord, such that:

Only one letter can be changed at a time

Each intermediate word must exist in the word list

For example,

Given:

beginWord = "hit"

endWord = "cog"

wordList = ["hot","dot","dog","lot","log"]

As one shortest transformation is "hit" -> "hot" -> "dot" -> "dog" -> "cog",

return its length 5.

## 题目大意：

给出两个单词beginWord和endWord，和一个字典，求出从beginWord到endWord的最短转换序列，要求中间单词必须是字典中的，且每次只能转换一个字母。

## 题目分析：

表面上看是一个单源最短路径问题，博客中说这道题的test case和题目要求都有问题，所以我也就没做，以下只贴出ac代码。

## 源码：（language：java）

public class Solution {

public int ladderLength(String beginWord, String endWord, Set<String> wordList) {

Set<String> beginSet = new HashSet<String>(), endSet = new HashSet<String>();

int len = 1;

int strLen = beginWord.length();

HashSet<String> visited = new HashSet<String>();

beginSet.add(beginWord);

endSet.add(endWord);

while (!beginSet.isEmpty() && !endSet.isEmpty()) {

if (beginSet.size() > endSet.size()) {

Set<String> set = beginSet;

beginSet = endSet;

endSet = set;

}

Set<String> temp = new HashSet<String>();

for (String word : beginSet) {

char[] chs = word.toCharArray();

for (int i = 0; i < chs.length; i++) {

for (char c = 'a'; c <= 'z'; c++) {

char old = chs[i];

chs[i] = c;

String target = String.valueOf(chs);

if (endSet.contains(target)) {

return len + 1;

}

if (!visited.contains(target) && wordList.contains(target)) {

temp.add(target);

visited.add(target);

}

chs[i] = old;

}

}

}

beginSet = temp;

len++;

}

return 0;

}

}

## 成绩：

47ms,beats 89.34%,众数97ms,2.42%

## Cmershen的碎碎念：

据说后面的word ladder 2这道题也是有问题的。

# Middle-题目120：179. Largest Number

## 题目原文：

Given a list of non negative integers, arrange them such that they form the largest number.

For example, given [3, 30, 34, 5, 9], the largest formed number is 9534330.

Note: The result may be very large, so you need to return a string instead of an integer.

## 题目大意：

给出一串由非负数组成的数组，求出连缀而成的最大整数。

注意：这个结果可能非常大，所以返回一个字符串。

## 题目分析：

观察例子,最大的数的排列顺序分别为9,5,34,3,30，观察到任意两个元素a和b，若a在b的前面则ab连缀的数一定比ba连缀的数大。据此自定义一个排序算法对数组排序并按序连缀即可。

## 源码：（language：java）

public class Solution {

private class MyComparator implements Comparator<Integer> {

public int compare(Integer o1, Integer o2) {

// TODO Auto-generated method stub

String s1=o1.toString()+o2.toString();

String s2=o2.toString()+o1.toString();

return s2.compareTo(s1);

}

}

public String largestNumber(int[] nums) {

List<Integer> list = new ArrayList<Integer>();

for(int num:nums)

list.add(num);

Collections.sort(list, new MyComparator());

String result = "";

for(Integer i:list)

result+=String.valueOf(i);

if(result.startsWith("00"))

result = "0";

return result;

}

}

## 成绩：

123ms,beats 64.21%,众数123ms,6.42%

## Cmershen的碎碎念：

本题的关键在于找到合适的排序规则，一开始想到的是按字典序排列，但不能处理34>3>30的问题，后又想到补全字符串，但太慢。最后发现只要比较连缀结果的大小即可。by the way，lambda表达式咋写？我看到的ac代码大多用的是匿名类。

# Middle-题目121：15. 3Sum

## 题目原文：

Given an array S of n integers, are there elements a, b, c in S such that a + b + c = 0? Find all unique triplets in the array which gives the sum of zero.

## 题目大意：

给出一个数组s，求三元组(a,b,c)，其中a,b,c∈s且a+b+c=0。

## 题目分析：

典型的ksum问题，不求原数组下标，所以排序后固定一个指针即可。

## 源码：（language：java）

public class Solution {

public List<List<Integer>> threeSum(int[] nums) {

Arrays.sort(nums);

List<List<Integer>> list = new LinkedList<List<Integer>>();

for(int i = 0;i<nums.length-2;i++) {

if(i>0 && nums[i]==nums[i-1])

continue;

int j=i+1,k=nums.length-1;

while(j<k) {

if(j>i+1 && nums[j]==nums[j-1]) {

j++;

continue;

}

if(k<nums.length-1&&nums[k]==nums[k+1]) {

k--;

continue;

}

int sum = nums[i]+nums[j]+nums[k];

if(sum==0) {

List<Integer> sublist = new LinkedList<Integer>();

sublist.add(nums[i]);

sublist.add(nums[j]);

sublist.add(nums[k]);

list.add(sublist);

j++;

}

else if(sum>0) {

k--;

}

else {

j++;

}

}

}

return list;

}

}

## 成绩：

9ms,beats 34.07%,众数8ms,31.75%

## Cmershen的碎碎念：

本题解决了重复数字的问题，确保解集中没有重复元素。但同样的去重逻辑放到4sum题里不知道为什么就不行。。。。。。

# Middle-题目122：220. Contains Duplicate III

## 题目原文：

Given an array of integers, find out whether there are two distinct indices i and j in the array such that the difference between nums[i] and nums[j] is at most t and the difference between i and j is at most k.

## 题目大意：

给出一个数组nums，判断是否存在这样两个下标i和j使得|nums[i]-nums[j]≤t，且|j-i|≤k.

## 题目分析：

巧用TreeSet这个类，它是一个集合，元素不可重复，不可维护加入集合的顺序，但集合是有序的，有floor(n)和ceiling(n)两个方法取得元素n相邻的两个元素。

那么构造一个TreeSet，并维持TreeSet的长度为K（如果i超过k就把nums[i-k]弹出去）。然后设n=nums[i].每次判断n-floor(n)是否≤t或者ceiling(n)是否≤t（因为TreeSet的长度不超过k，所以如果存在floor(n)或者ceiling(n)，其在数组的下标与n的差值必然不超过k），若存在则找到解，返回true，若不存在则加入n，弹出nums[i-k]，总的时间复杂度是O(nlogk) (因为TreeSet的add和remove都是logk复杂度的)

## 源码：（language：java）

public class Solution {

public boolean containsNearbyAlmostDuplicate(int[] nums, int k, int t) {

if(k < 1 || t < 0)

return false;

TreeSet<Integer> set = new TreeSet<>();

for(int i = 0; i < nums.length; i++){

int n = nums[i];

if(set.floor(n) != null && n <= t + set.floor(n) ||

set.ceiling(n) != null && set.ceiling(n) <= t + n)

return true;

set.add(n);

if (i >= k)

set.remove(nums[i - k]);

}

return false;

}

}

## 成绩：

68ms,beats 10.72%,众数23ms,5.88%

## Cmershen的碎碎念：

本题类似于滑动窗口，是第一道需要用到TreeSet的题，但是效率不高，可能有O(n)的线性算法。TreeSet基于红黑树实现，它遗失插入顺序但保证集合内有序，而红黑树是一种应用广泛的用于查找的二叉树，关于红黑树的详细了解可自行百度。类似的数据结构还有TreeMap（保证key有序）。

# Middle-题目123：335. Self Crossing

## 题目原文：

You are given an array x of n positive numbers. You start at point (0,0) and moves x[0] metres to the north, then x[1] metres to the west, x[2] metres to the south, x[3] metres to the east and so on. In other words, after each move your direction changes counter-clockwise.

Write a one-pass algorithm with O(1) extra space to determine, if your path crosses itself, or not.

Example 1:

Given x = [2, 1, 1, 2],

Return true (self crossing)

Example 2:

Given x = [1, 2, 3, 4],

Return false (not self crossing)

Example 3:

Given x = [1, 1, 1, 1],

Return true (self crossing)

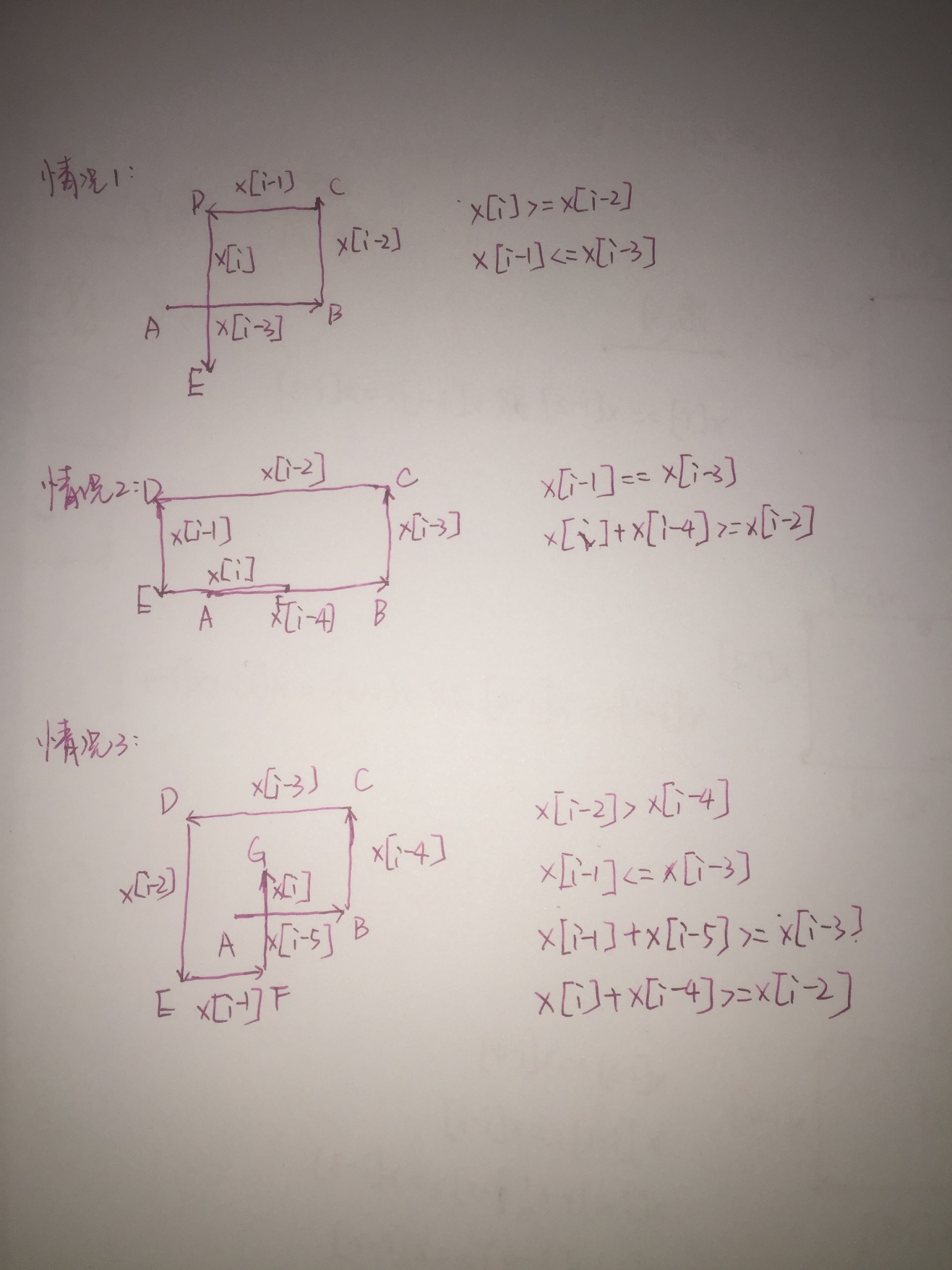
## 题目大意：

给出一个正数组成的数组x，你从坐标系原点开始走，向上走x[0]米，再向左走x[1]米,再向下走x[2]米，再向下走x[3]米，以此类推的**逆时针**走，判断运动轨迹是否相交。

## 题目分析：

参考了discuss中一位大神的算法，解释如下：

首先，只走3步是不可能产生交点的，所以考虑走4步，走5步，走6步产生交点的情况，而走7步与走5步是等价的，所以只需讨论三种情况即可。（画图太渣，还是传手写照片吧）

需注意情况3中第一个条件是没有等号的，如果取等则变为情况2！

## 源码：（language：java）

public class Solution {

public boolean isSelfCrossing(int[] x) {

for(int i=3;i<x.length;i++){

if(i>=3&&x[i]>=x[i-2]&&x[i-1]<=x[i-3])

return true;

if(i>=4&&x[i-1]==x[i-3]&&x[i-2]<=x[i]+x[i-4])

return true;

if(i>=5&&x[i-2]>x[i-4]&&x[i-1]<=x[i-3]&&x[i-1]+x[i-5]>=x[i-3]&&x[i]+x[i-4]>=x[i-2])

return true;

}

return false;

}

}

## 成绩：

0ms，beats 43.49%，众数0ms,56.51%

## Cmershen的碎碎念：

本算法简洁优美，堪称绝妙！还有，高中的平面向量还记得么~~

# Middle-题目124：91. Decode Ways

## 题目原文：

A message containing letters from A-Z is being encoded to numbers using the following mapping:

'A' -> 1

'B' -> 2

...

'Z' -> 26

Given an encoded message containing digits, determine the total number of ways to decode it.

For example,

Given encoded message "12", it could be decoded as "AB" (1 2) or "L" (12).

The number of ways decoding "12" is 2.

## 题目大意：

给出一个字符串，判断有多少种按字母解码的方式。例如”12”可以解码为"AB" (1 2) "L" (12).

## 题目分析：

看题很明显想到是DP，但对’0’的处理是个很麻烦的问题。

基本的dp方程很简单：设dp[i]表示前i个字符的解码方式数。则有：

dp[i] = dp[i-1] (s[i]<’9’ && s[i]>’0’)

dp[i-1]+dp[i-2] (“00”<s[i-1]+s[i]<”27”)

但这都是不考虑0的情况。所以公式有误，还要进行修正

首先考虑s[i]=’0’，若s[i-1]=’1’或’2’，则dp[i]=dp[i-2],否则dp[i]=0 （也可直接return 0，因为”30”,”40”….这样的子串不论如何都不可按大写字母解码）

接下来考虑s[i-1]=’0’,因为已经排除掉s[i]=’0’的情况，故最后一位必可解码（因为一定是’1’~’9’了），最后两位必不可解码（因为”01”,”02”…必不可解码），直接取dp[i]=dp[i-1]即可。

## 源码：（language：java）

public class Solution {

public int numDecodings(String s) {

if(s.length() == 0) return 0;

if(s.charAt(0) == '0') return 0;

if(s.length() == 1) return s.charAt(0) > '0' ? 1:0;

int dp[] = new int[s.length()+1];

dp[0] = dp[1] = 1;

for(int i=2; i<=s.length(); i++){

dp[i] = dp[i-1];

if(s.charAt(i-1) == '0')

if (s.charAt(i-2) == '1' || s.charAt(i-2) == '2')

dp[i] = dp[i-2];

else return 0;

else if(s.charAt(i-2) == '0'){

dp[i] = dp[i-1];

}

else if(s.charAt(i-2) == '1' || (s.charAt(i-2) == '2' && s.charAt(i-1) < '7') )

dp[i] += dp[i-2];

}

return dp[s.length()];

}

}

## 成绩：

2ms,beats 70.90%，众数5ms,26.83%

## Cmershen的碎碎念：

一开始的解法是设dp[i] = dp[i-1]\*c1+dp[i-2]\*c2并讨论c1和c2的情况，发现很难做到不重不漏，尤其是在对0 的处理上.但这个思路也可以解决。

# Middle-题目125：130. Surrounded Regions

## 题目原文：

Given a 2D board containing 'X' and 'O', capture all regions surrounded by 'X'.

A region is captured by flipping all 'O's into 'X's in that surrounded region.

For example,

X X X X

X O O X

X X O X

X O X X

After running your function, the board should be:

X X X X

X X X X

X X X X

X O X X

## 题目大意：

给出一个由’X’和’O’组成的数组，把所有被’X’围住的’O’都改成’X’。（在边缘的不用改）

## 题目分析：

DFS和BFS都行，只要对边缘的’O’搜索即可，搜索到的’O’都改成’B’，最后再遍历整个矩阵，把’O’改成’X’,把’B’改成’O’即可。（ac代码用的是BFS）

## 源码：（language：java）

public class Solution {

private class Point {

int x;

int y;

Point(int x, int y) {

this.x = x;

this.y = y;

}

}

public void solve(char[][] board) {

if (board == null || board.length == 0)

return;

int rows = board.length, columns = board[0].length;

int[][] direction = { { -1, 0 }, { 1, 0 }, { 0, 1 }, { 0, -1 } };

for (int i = 0; i < rows; i++) {

for (int j = 0; j < columns; j++) {

if ((i == 0 || i == rows - 1 || j == 0 || j == columns - 1) && board[i][j] == 'O') {

Queue<Point> queue = new LinkedList<>();

board[i][j] = 'B';

queue.offer(new Point(i, j));

while (!queue.isEmpty()) {

Point point = queue.poll();

for (int k = 0; k < 4; k++) {

int x = direction[k][0] + point.x;

int y = direction[k][1] + point.y;

if (x >= 0 && x < rows && y >= 0 && y < columns && board[x][y] == 'O') {

board[x][y] = 'B';

queue.offer(new Point(x, y));

}

}

}

}

}

}

for (int i = 0; i < rows; i++) {

for (int j = 0; j < columns; j++) {

if (board[i][j] == 'B')

board[i][j] = 'O';

else if (board[i][j] == 'O')

board[i][j] = 'X';

}

}

}

}

## 成绩：

12ms,beats 38.36%,众数7ms,9.03%

## Cmershen的碎碎念：

好像也可以用并查集解决，但我搜到一份ac代码提交上去发现并不快，而且我也不是很熟悉并查集。就用了这份代码顺便复习下BFS的写法。

# Middle-题目126：151. Reverse Words in a String

## 题目原文：

Given an input string, reverse the string word by word.

For example,

Given s = "the sky is blue",

return "blue is sky the".

## 题目大意：

输入一个字符串，把空格分隔的字符串翻转。

## 题目分析：

先修剪(trim)掉两边的空格，然后按空格分隔，逆序输出即可。

## 源码：（language：java）

public class Solution {

public String reverseWords(String s) {

s=s.trim();

String[] words = s.split(" ");

String result = "";

for(int i = words.length-1;i>0;i--) {

if(!words[i].equals(""))

result=result+words[i]+" ";

}

if(words.length!=0)

result+=words[0];

return result;

}

}

## 成绩：

58ms,beats 8.93%,众数3ms,13.42%

## Cmershen的碎碎念：

本题成绩较差，因为用到了两个String的库函数。其实这道题非常非常的水，问题出现在很多奇怪的边界情况，如首尾的空格，还有只有一个单词的情况……

# Middle-题目127：29. Divide Two Integers

## 题目原文：

Divide two integers without using multiplication, division and mod operator.

If it is overflow, return MAX\_INT.

## 题目大意：

不用乘法、除法、取模实现两个数的除法，如果溢出则返回MAX\_INT（232-1）

## 题目分析：

直接用a/b水过去的，正确的解法好像是模拟二进制的竖式除法，用位运算和加减解决。

## 源码：（language：java）

public class Solution {

public int divide(int dividend, int divisor) {

return (dividend==-2147483648 && divisor == -1)?Integer.MAX\_VALUE:dividend/divisor;

}

}

## 成绩：

2ms,beats 75.64%,众数3ms,66.70%

## Cmershen的碎碎念：

原则上除法是CPU的底层上用硬件实现的，但可以用高级语言代码去模拟。据论坛中所说，实现除法、乘法、乘方等数值运算的问题在面试中还是比较常见的，有时间整理一下。

# Middle-题目128：166. Fraction to Recurring Decimal

## 题目原文：

Given two integers representing the numerator and denominator of a fraction, return the fraction in string format.

If the fractional part is repeating, enclose the repeating part in parentheses.

For example,

Given numerator = 1, denominator = 2, return "0.5".

Given numerator = 2, denominator = 1, return "2".

Given numerator = 2, denominator = 3, return "0.(6)".

## 题目大意：

把一个分数转换成小数，如果是循环小数，则用括号把循环节扩起来，输出对应的字符串。

## 题目分析：

就是模拟竖式除法，每一步迭代都记录下商和余数，用string记录下每一步的商，用一个HashMap记录下<余数,对应商的下标>。如果出现余数为0，则是有限小数，直接输出string即可，如果找到一个在HashMap中存在的余数，则说明发现了循环节，到HashMap中get一下就知道了循环节的开始点，这样也就结束了循环。

## 源码：（language：java）

public class Solution {

public String fractionToDecimal(int numerator, int denominator) {

if (numerator == 0) return "0";

if (denominator == 0) return "";

String ans = "";

//如果结果为负数

if ((numerator < 0) ^ (denominator < 0)) {

ans += "-";

}

//下面要把两个数都转为正数，为避免溢出，int转为long

long num = numerator, den = denominator;

num = Math.abs(num);

den = Math.abs(den);

//结果的整数部分

long res = num / den;

ans += String.valueOf(res);

//如果能够整除，返回结果

long rem = (num % den) \* 10;

if (rem == 0) return ans;

//结果的小数部分

HashMap<Long, Integer> map = new HashMap<Long, Integer>();

ans += ".";

while (rem != 0) {

//如果前面已经出现过该余数，那么将会开始循环

if (map.containsKey(rem)) {

int beg = map.get(rem); //循环体开始的位置

String part1 = ans.substring(0, beg);

String part2 = ans.substring(beg, ans.length());

ans = part1 + "(" + part2 + ")";

return ans;

}

//继续往下除

map.put(rem, ans.length());

res = rem / den;

ans += String.valueOf(res);

rem = (rem % den) \* 10;

}

return ans;

}

}

## 成绩：

5ms,beats 19.25%,众数4ms,59.22%

## Cmershen的碎碎念：

本题乃Middle难度中ac率最低一题，但原理也不是很复杂，而是难在各种奇奇怪怪的边界情况，Leetcode的test case很多都侧重于一些特殊情况（这适合面试题，锻炼严谨的思维）而传统的oj题及赛题则更多侧重于大数据量下会不会tle。

# Middle-题目129：337. House Robber III（增补1）

## 题目原文：

The thief has found himself a new place for his thievery again. There is only one entrance to this area, called the "root." Besides the root, each house has one and only one parent house. After a tour, the smart thief realized that "all houses in this place forms a binary tree". It will automatically contact the police if two directly-linked houses were broken into on the same night.

Determine the maximum amount of money the thief can rob tonight without alerting the police.

Example 1:

3

/ \

2 3

\ \

3 1

Maximum amount of money the thief can rob = 3 + 3 + 1 = 7.

Example 2:

3

/ \

4 5

/ \ \

1 3 1

Maximum amount of money the thief can rob = 4 + 5 = 9.

## 题目大意：

此题是[Easy-题目26](#_Easy-题目26：198._House_Robber)和Middle-题目58的变形，这次这个贼偷的是二叉树，每偷一个节点，它的父节点和子节点都不能偷了，求最大收益。

## 题目分析：

我们很熟悉的前两题都是使用动态规划，而这道题用的是深搜，深搜的返回结果是一个两个元素的数组profit，其中profit[0]代表偷当前节点的时候得到的最大收益，profit[1]代表不偷当前节点时获得的最大收益。那么每次搜索左右子树的最大收益记为leftprofit和rightprofit，并取：

profit[0] = leftProfit[1] + rightProfit[1] + root.val;

profit[1] = Math.max(leftProfit[0],leftProfit[1]) + Math.max(rightProfit[0], rightProfit[1]);

直到遇到空节点返回new int[2].

## 源码：（language：java）

public class Solution {

public int rob(TreeNode root) {

int[] profit = dfs(root);

return Math.max(profit[0], profit[1]);

}

private int[] dfs(TreeNode root) {

int[] profit = new int[2];

// profit[0] stands for the maximum profit with robbing root, while profit[1] stands for the maximum profit without robbing root.

if(root!=null) {

int[] leftProfit = dfs(root.left);

int[] rightProfit = dfs(root.right);

profit[0] = leftProfit[1] + rightProfit[1] + root.val;

profit[1] = Math.max(leftProfit[0],leftProfit[1]) + Math.max(rightProfit[0], rightProfit[1]);

return profit;

}

return new int[2];

}

}

## 成绩：

1ms（统计信息还未显示）

## Cmershen的碎碎念：

本题的递归之前的题较少接触，递归的关系不太容易看出来，其实递归的设计应该是考虑函数的入口参数和返回值以及函数的功能，而不是强调递推关系式（类似斐波那契数列的递归方法或者欧几里得算法求gcd），那样构思很多时候是想不出来的（比如本题）。

# Middle-题目130：338. Counting Bits（增补2）

## 题目原文：

Given a non negative integer number num. For every numbers i in the range 0 ≤ i ≤ num calculate the number of 1's in their binary representation and return them as an array.

Example:

For num = 5 you should return [0,1,1,2,1,2].

## 题目大意：

输入一个正整数num，返回一个num+1长的数组，代表从0~num每个数二进制码中1的个数。

## 题目分析：

用Java中的Integer.bitCount这个函数水过去。

## 源码：（language：java）

public class Solution {

public int[] countBits(int num) {

int[] bits = new int[num+1];

for(int i = 0;i<=num;i++)

bits[i] = Integer.bitCount(i);

return bits;

}

}

## 成绩：

5ms（统计信息还未显示）

## Cmershen的碎碎念：

附JDK中bitCount的算法（大概是一个很巧妙的”分治算法”的位运算，复杂度O(1)）

public static int bitCount(int i) {

// HD, Figure 5-2

i = i - ((i >>> 1) & 0x55555555);

i = (i & 0x33333333) + ((i >>> 2) & 0x33333333);

i = (i + (i >>> 4)) & 0x0f0f0f0f;

i = i + (i >>> 8);

i = i + (i >>> 16);

return i & 0x3f;

}

有时间把那些二进制位写开认真分析一下。

# Hard-题目1：52. N-Queens II

## 题目原文：

Follow up for N-Queens problem.

Now, instead outputting board configurations, return the total number of distinct solutions.

## 题目大意：

N皇后问题都很熟悉了，输入棋盘规模n，判断有多少不同的解。

## 题目分析：

离线算法，百度去查一下n皇后的解就可以了。

## 源码：（language：java）

public class Solution {

public int totalNQueens(int n) {

int[] ans = new int[]{0, 1, 0, 0, 2, 10, 4, 40, 92, 352};

return ans[n];

}

}

## 成绩：

0ms，beats 98.57%，众数3ms，23.66%

## Cmershen的碎碎念：

如果真的是面试中遇到，请参考正常的八皇后问题解法（用4个数组记录横向、竖向、主对角线、副对角线是否可放皇后），然后一行一行放皇后求解。

# Hard-题目2：287. Find the Duplicate Number

## 题目原文：

Given an array nums containing n + 1 integers where each integer is between 1 and n (inclusive), prove that at least one duplicate number must exist. Assume that there is only one duplicate number, find the duplicate one.

## 题目大意：

给出一个长度为n+1的数组，其中数组元素都在[1,n]范围，已知输入的数组只有一个数字是重复的，找出那个重复的数字。

## 题目分析：

使用HashSet可以水过去。

## 源码：（language：java）

public class Solution {

public int findDuplicate(int[] nums) {

HashSet<Integer> set = new HashSet<Integer>();

for(int num : nums)

if(!set.add(num))

return num;

return -1;

}

}

## 成绩：

6ms,beats 23.40%,众数1ms，28.98%

## Cmershen的碎碎念：

其实本题的follow up 要求O(n^2)以下时间复杂度，O(1)空间复杂度，且不能移动数组。因此本题的hashset法其实是不合要求的（O(n)空间复杂度），也不能排序。

<http://blog.sina.com.cn/s/blog_a131087e0102vtb1.html>中提到个很巧妙的two pointers解法，有待下次修正时认真研读。（大致是有重复元素时会迭代成环？）

# Hard-题目3：312. Burst Balloons

## 题目原文：

Given n balloons, indexed from 0 to n-1. Each balloon is painted with a number on it represented by array nums. You are asked to burst all the balloons. If the you burst balloon i you will get nums[left] \* nums[i] \* nums[right] coins. Here left and right are adjacent indices of i. After the burst, the left and right then becomes adjacent.

Find the maximum coins you can collect by bursting the balloons wisely.

Note:

(1) You may imagine nums[-1] = nums[n] = 1. They are not real therefore you can not burst them.

(2) 0 ≤ n ≤ 500, 0 ≤ nums[i] ≤ 100

Example:

Given [3, 1, 5, 8]

Return 167

nums = [3,1,5,8] --> [3,5,8] --> [3,8] --> [8] --> []

coins = 3\*1\*5 + 3\*5\*8 + 1\*3\*8 + 1\*8\*1 = 167

## 题目大意：

给出n个气球，每个气球上有一个数字，进行以下的游戏：玩家每次刺破一个气球，获得m元钱，其中m为刺破的气球及其相邻两个气球上面的数字的乘积（规定第-1个和第n个气球上的数字为1，相当于边缘的气球只考虑一边即可），计算玩家可能获得的最大收益。

## 题目分析：

设dp[i][j]表示从第i个气球到第j个气球的最大收益，则有如下转移方程：

很好理解，从第i个气球到第j的气球的最大收益要暴力枚举从其中的第k位刺破所带来的收益。最终问题即求dp[1][n].

## 源码：（language：cpp）

class Solution {

public:

int maxCoins(vector<int>& nums) {

for(int i=0;i<nums.size();++i){

if(nums[i]==0){

nums.erase(nums.begin()+i);

--i;

}

}

int n=nums.size();

if(n==0) return 0;

nums.insert(nums.begin(),1);

nums.insert(nums.end(),1);

int m=nums.size();

vector<vector<int>> dp(m,vector<int>(m,0));

for(int count=1;count<=n;++count){

for(int start=1;start+count-1<=n;++start){

int bestcoins=0;

for(int b=0;b<count;++b){

bestcoins=max(bestcoins,dp[start][start+b-1]+nums[start-1]\*nums[start+b]\*nums[start+count]+dp[start+b+1][start+count-1]);

}

dp[start][start+count-1]=bestcoins;

}

}

return dp[1][n];

}

};

## 成绩：

36ms,beats 61.75%,众数44ms,22.51%

## Cmershen的碎碎念：

本题的转移方程及其实现都不难，难在想到这个递推关系。而通常会想到回溯法，显然要暴力枚举每种扎气球的方法，其复杂度为O(n!)，在n=500的数据量下是不能接受的。考虑到每扎破一个气球，都退化成求两边的最佳扎破方法，故可以使用dp，本算法复杂度O(n3).

# Hard-题目4：145. Binary Tree Postorder Traversal

## 题目原文：

Given a binary tree, return the postorder traversal of its nodes' values.

For example:

Given binary tree {1,#,2,3},

1

\

2

/

3

return [3,2,1].

## 题目大意：

求二叉树的后序遍历。

## 题目分析：

只考虑ac，故水过去。

## 源码：（language：java）

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

public class Solution {

public List<Integer> postorderTraversal(TreeNode root) {

List<Integer> result = new ArrayList<Integer>();

traverse(root,result);

return result;

}

public void traverse(TreeNode node,List<Integer> list) {

if(node!=null) {

traverse(node.left,list);

traverse(node.right,list);

list.add(node.val);

}

return;

}

}

## 成绩：

1ms，beats 70.01%，众数2ms,62.51%

## Cmershen的碎碎念：

本题的递归算法很简洁易懂，哪怕是数据结构的初学者也能轻松写出。但是非递归解法比较复杂，好像要用两个栈（严蔚敏版数据结构应该讲了），且时间复杂度并未提高，所以我觉得这是一个挺无聊的问题。Btw，任何递归算法都存在等效的非递归算法吗？

# Hard-题目5：154. Find Minimum in Rotated Sorted Array II

## 题目原文：

Suppose a sorted array is rotated at some pivot unknown to you beforehand.

(i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2).

Find the minimum element.

The array may contain duplicates.

## 题目大意：

对一个移位后的排序数组（可能存在重复数字）求最小元素。

## 题目分析：

从头扫一遍即可。也可以通过找跳转点（从最大值突变到最小值，期望是暴力枚举法的1/2）

## 源码：（language：java）

public class Solution {

public int findMin(int[] nums) {

int min = Integer.MAX\_VALUE;

for(int num : nums)

min=Math.min(num,min);

return min;

}

}

## 成绩：

1ms,beats 5.33%,众数1ms,94.67%

# Hard-题目6：117. Populating Next Right Pointers in Each Node II

## 题目原文：

Follow up for problem "Populating Next Right Pointers in Each Node".

What if the given tree could be any binary tree? Would your previous solution still work?

Note:

You may only use constant extra space.

For example,

Given the following binary tree,

1

/ \

2 3

/ \ \

4 5 7

After calling your function, the tree should look like:

1 -> NULL

/ \

2 -> 3 -> NULL

/ \ \

4-> 5 -> 7 -> NULL

## 题目大意：

给一个普通二叉树添加next节点。

## 题目分析：

与[Middle-题目17](#_Middle-题目17：116._Populating_Next)类似，区别是这里面的二叉树是普通二叉树，上一题则是完全二叉树。所以左孩子的next不一定是右孩子，右孩子的next也不一定是父节点next的左孩子，而是从父节点开始，一直向右找到一个有孩子的同层节点，然后指向其孩子（如果有左孩子就是左孩子，没有就是右孩子）。

所以每排完一层之后向下深搜的时候，要先搜右子树，再搜左子树，故本题要用**逆前序遍历！！！，**如果是常规的前序遍历，向左子树深搜的时候由于右子树没处理，会有一些next没连上。

## 源码：（language：java）

public class Solution {

public void connect(TreeLinkNode root) {

if(root!=null)

dfs(root,root.left,root.right);

}

private void dfs(TreeLinkNode parent, TreeLinkNode leftchild, TreeLinkNode rightchild) { // asserted that parent isn't null

TreeLinkNode p = parent;

while (p.next!=null && (p==parent || (p.left==null&&p.right==null)))

p=p.next;

if(leftchild!=null) {

if(rightchild!=null)

leftchild.next = rightchild;

else {

if(p!=parent)

leftchild.next = p.left!=null?p.left:p.right;

}

}

if(rightchild!=null) {

if(p!=parent)

rightchild.next = p.left!=null?p.left:p.right;

}

if(rightchild!=null)

dfs(rightchild, rightchild.left, rightchild.right);

if(leftchild!=null)

dfs(leftchild, leftchild.left, leftchild.right);

}

}

## 成绩：

2ms,beats 32.02%,众数2ms,39.09%

## Cmershen的碎碎念：

Follow up有说到，要常数的空间复杂度，不知道递归是不是不合要求的？如果不合要求，那么可以通过改造前序遍历来实现，貌似会很麻烦……

# Hard-题目7：42. Trapping Rain Water

## 题目原文：

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it is able to trap after raining.



For example,

Given [0,1,0,2,1,0,1,3,2,1,2,1], return 6.

## 题目大意：

给出n个非负整数数组，其中height[i]代表坐标轴上横坐标为[i,i+1]的矩形的高度，计算这些矩形和x轴围成的“容器”能装多少水。

## 题目分析：

本题是[Middle-题目31](#_Middle-题目31：11._Container_With)的变形，也是用两个指针i和j分别指向数组头和结尾，记两边矩形的高度分别为height[i]和height[j],分两种情况讨论：

1. height[i]<height[j],则i向右移动，如果后面矩形的高度小于height[i]，则其高度差值一定可以装水(因为j那边比i高，不管中间是什么情况，j那侧一定不是“短板”，如果后面矩形的高度大于Height[i]，则停止滑动i，因为已经找到一个“杯子”并计算出它的水量了。
2. height[i]>height[j]，同理j向左移动，找到一个高度大于等于height[j]的矩形，并记录差值（可以装水）。

等于的情况放到任何一种里面都可以，因为等于的时候装水量为0，最终到I,j相遇时循环停止。整个数组只扫描了一遍。

## 源码：（language：java）

public class Solution {

public int trap(int[] height) {

int res=0;

int l=0;

int r=height.length-1;

while(l<r){

int temp=Math.min(height[l],height[r]);

if(temp==height[l]){

l++;

while(l<r&&height[l]<=temp){

res+=temp-height[l];

l++;

}

}

else{

r--;

while(l<r&&height[r]<=temp){

res+=temp-height[r];

r--;

}

}

}

return res;

}

}

## 成绩：

1ms,beats 87.85%,众数2ms,71.18%

## Cmershen的碎碎念：

也没什么要注意的，认真理解了上一题之后本题变化也不是很大。关键是注意“寻找杯子”的过程。

# Hard-题目8：128. Longest Consecutive Sequence

## 题目原文：

Given an unsorted array of integers, find the length of the longest consecutive elements sequence.

For example,

Given [100, 4, 200, 1, 3, 2],

The longest consecutive elements sequence is [1, 2, 3, 4]. Return its length: 4.

Your algorithm should run in O(n) complexity.

## 题目大意：

给出一个无序的数组，寻找最长的连续元素，返回其长度。

例如数组[100,4,200,1,3,2]的最长连续元素是[1,2,3,4]所以返回4.

要求在线性时间复杂度内解决。

## 题目分析：

要求没做到，用了朴素的办法水过去的。复杂度O(nlogn).,先排序，再扫描一遍数组寻找最大的连续元素。如果遇到重复元素则跳过。

## 源码：（language：java）

public class Solution {

public int longestConsecutive(int[] nums) {

Arrays.sort(nums);

int longest = 1;

int max = 1;

for(int i = 1;i<nums.length;i++) {

if(nums[i]==nums[i-1])

continue;

else if(nums[i]-nums[i-1]==1) {

longest++;

if(longest>max)

max = longest;

}

else

longest=1;

}

return max;

}

}

## 成绩：

5ms,beats 95.00%,众数16ms,11.95%

## Cmershen的碎碎念：

本题的tags给的是并查集，我想到的大致思路是，把所有相邻的都加入一个集合中，并维护每个集合的最大值和最小值，遍历数组的每个元素n，如果n-1或n+1都在集合中，则把这两个集合并起来，如果只有一个在，则加入相应集合，如果都不在则自成一个集合。但还没实现，而且也不确定是否是O(n)时间的。而O(nlogn)的复杂度下的完成时间确是很快的（似乎小于线性算法），可能是用例的数量不够大。

# Hard-题目9：301. Remove Invalid Parentheses

## 题目原文：

Remove the minimum number of invalid parentheses in order to make the input string valid. Return all possible results.

Note: The input string may contain letters other than the parentheses ( and ).

Examples:

"()())()" -> ["()()()", "(())()"]

"(a)())()" -> ["(a)()()", "(a())()"]

")(" -> [""]

## 题目大意：

给出一个括号和字母的字符串，其中有一些括号是失配的，求出所有去掉失配括号的解。

例如：

"()())()" -> ["()()()", "(())()"]

（这个例子中多了一个右括号，但有两种去掉右括号的方式，故最后答案有2个解）

## 题目分析：

参考别人博客的解法，使用DFS求解。

首先定义“失配度”，代表字符串或其子串中失配括号的个数，博客中给出的解法大多使用了堆栈，其实不用，只用一个整形（相当于堆栈长度）维护即可，遇到左括号+1，遇到右括号如果长度不为0则减1，否则失配度+1，扫完字符串后加上这个长度（失配左括号数）。

然后从源字符串开始搜索，遍历字符串，如果当前字符串不是括号则跳过，否则计算去掉这个括号的失配度，如果比当前失配度小1，则继续向下搜索这个子串（说明继续寻找可以找到解），如果比当前失配度大1，则说明去掉这个括号不能得到解（因为继续搜索下去即使找到解也去掉了原来已经匹配的括号！），如果失配度=0则找到解。

这里需要用一个HashSet记录所有搜索过的字符串，如果一个字符串已经搜索过则不再搜索。这是个很重要的剪枝，否则会超时。因为对”((((((((((((((((((((((((((())”这个串，如果不做这个处理，第一层就会对每个去掉左括号的子串搜索一遍，而它们是一样的，可想而知有多少重复操作！

## 源码：（language：java）

public class Solution {

HashSet<String> visited = new HashSet<String>();

private int getMisses(String s) { // calculate the missing num of parentheses

int parentheses = 0;

int miss = 0;

for(int i = 0;i<s.length();i++) {

if(s.charAt(i) == '(')

parentheses++;

else if(s.charAt(i) == ')'){

if(parentheses==0)

miss++;

else

parentheses--;

}

}

miss+=parentheses;

return miss;

}

private void dfs(List<String> list, String s, int miss) {

if(miss == 0)

list.add(s);

else {

for(int i = 0;i<s.length();i++) {

String nextStr = removeCharAt(s, i);

if(!visited.contains(nextStr)) {

visited.add(nextStr);

int nextMiss = getMisses(nextStr);

if(nextMiss<miss)

dfs(list, nextStr, nextMiss);

}

}

}

}

private String removeCharAt(String s, int pos) {

return s.substring(0, pos) + s.substring(pos + 1);

}

public List<String> removeInvalidParentheses(String s) {

List<String> list = new ArrayList<String>();

dfs(list,s,getMisses(s));

return list;

}

}

## 成绩：

23ms,beats 57.67%,众数3ms,13.37%

## Cmershen的碎碎念：

3ms的办法也用到了递归，但不知道有没有回溯，在discuss <https://leetcode.com/discuss/81478/easy-short-concise-and-fast-java-dfs-3-ms-solution>中有介绍，但满篇E文让我懒得看，而博客中的中文分析大多与上述解法类似（我也是基于中文博客做的），还有一种基于bfs的，更慢且不易理解。

# Hard-题目10：33. Search in Rotated Sorted Array

## 题目原文：

Suppose a sorted array is rotated at some pivot unknown to you beforehand.

(i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2).

You are given a target value to search. If found in the array return its index, otherwise return -1.

## 题目大意：

对一个移位过的，无重复数字的数组，寻找目标数字并返回其下标。如果没有则返回-1.

## 题目分析：

一开始想到的思路是找到跳变点（即移动点），然后两边都是有序的，再二分查找，但是想不到logn复杂度的，就决定直接无脑找了。

## 源码：（language：java）

public class Solution {

public int search(int[] nums, int target) {

for(int i = 0;i<nums.length;i++)

if(nums[i]==target)

return i;

return -1;

}

}

## 源码：

1ms，beats 5.34%，众数1ms，94.64%

# Hard-题目11：315. Count of Smaller Numbers After Self

## 题目原文：

You are given an integer array nums and you have to return a new counts array. The counts array has the property where counts[i] is the number of smaller elements to the right of nums[i].

Example:

Given nums = [5, 2, 6, 1]

To the right of 5 there are 2 smaller elements (2 and 1).

To the right of 2 there is only 1 smaller element (1).

To the right of 6 there is 1 smaller element (1).

To the right of 1 there is 0 smaller element.

Return the array [2, 1, 1, 0].

## 题目大意：

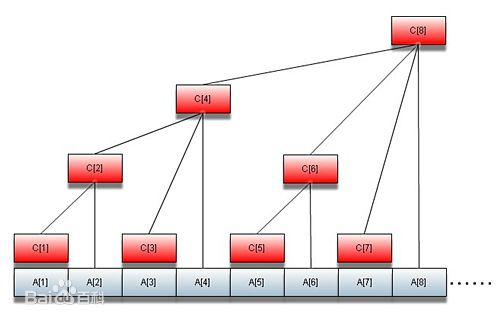
给出一个数组，计算每个元素右边有几个比它小的元素，组成一个新的数组返回。

## 题目分析：

这里需要用到一种高级数据结构叫做“树状数组”，又叫Binary Indexed Tree或Fenwick tree,其特点是查询和修改节点都是O(logn)的，用于快速求数组的前n项和。

构建方法：

设原数组是a[1…n],按如图方式构造数组c：（from 百度百科）



C[1]=a[1],c[2]=a[1]+a[2],….c[8]=a[1]+…+a[8],

树状数组有两个api,这两个api的时间复杂度都是O(logn)的，n为数组长度，具体实现略，有兴趣可以自行百度。

void add(int\* tree,int k,int num) // 在**a[k]**上增加num，更新树状数组tree

int get(int\* tree,int k) // 求a[1]+…+a[k]的和

再回到这道题，先求最小值和最大值，记为min和max，然后平移区间使得min=1,（若min不为1，则整个数组都减去min和1的差值）然后基于数组num(其中最小值修正为1，对应最大值为max-min+1) 构建树状数组tree[1…max-min+1]，其中对应的a数组的a[i]代表num数组中i出现的次数。

接下来逆向扫描num数组，每扫到一个元素num[i]，调用get函数求树状数组中a[1]到a[num[i]-1]的和，这个和值就是num数组中i右边比num[i]小的个数！！（这里最难理解，稍后说明），再调用add函数把num[i]加到a中。

时间复杂度：扫描一遍数组O(n),每个元素调用了两个O(logk)复杂度的api，其中k为树状数组长度，总复杂度为O(nlog(max-min)).

上面说了这么多有点抽象，下面举个栗子~~（看明白的童鞋或者嫌我墨迹的童鞋略过吧）

输入数组：[5,6,-1,1] (答案为:[2,2,0,0])

首先修正数组使得最小值为1(否则会出现数组越界):

nums[]=[6,8,1,3],min=1,max=8

然后构建tree[1..8]（因为tree的值很难写，就写图中下方的a数组）

逆向扫描数组：此时a数组初始化为全0

nums[i]=3,调用get函数求a[1]+a[2]为0（即1,2这两个数还没出现过），调用add函数使得a[3]++,

此时a数组：[0,0,1,0,0,0,0,0]

Nums[i]=1,调用get函数返回0（因为1左边没有数据了）,调用add函数使得a[1]++.

此时a数组：[1,0,1,0,0,0,0,0]

Nums[i]=8,调用get函数求a[1]+…+a[7]=2，调用add函数使得a[8]++

此时a数组：[1,0,1,0,0,0,0,1]

Nums[i]=6,调用get函数求a[1]+…+a[5]=2,调用add函数使得a[6]++.

此时a数组：[1,0,1,0,0,1,0,1]

从下往上读取get的返回值，得到数组[2,2,0,0]即为答案。

## 源码：（language：java）

public class Solution {

public List<Integer> countSmaller(int[] nums) {

LinkedList<Integer> res = new LinkedList<Integer>();

if (nums == null || nums.length == 0) {

return res;

}

// find min value and minus min by each elements, plus 1 to avoid 0 element

int min = Integer.MAX\_VALUE;

int max = Integer.MIN\_VALUE;

for (int i = 0; i < nums.length; i++) {

min = (nums[i] < min) ? nums[i]:min;

}

int[] nums2 = new int[nums.length];

for (int i = 0; i < nums.length; i++) {

nums2[i] = nums[i] - min + 1;

max = Math.max(nums2[i],max);

}

int[] tree = new int[max+1];

for (int i = nums2.length-1; i >= 0; i--) {

res.addFirst(get(nums2[i]-1,tree));

update(nums2[i],tree);

}

return res;

}

private int get(int i, int[] tree) {

int num = 0;

while (i > 0) {

num +=tree[i];

i -= i&(-i);

}

return num;

}

private void update(int i, int[] tree) {

while (i < tree.length) {

tree[i] ++;

i += i & (-i);

}

}

}

## 成绩：

8ms，beats 97.33%，众数11ms，8.79%

## Cmershen的碎碎念：

返回类型是List<Integer>,因为要从下往上读get函数的返回值，故使用链表的实现类LinkedList，因为提供了addFirst()方法.

# ****Hard-题目12：115. Distinct Subsequences****

## 题目原文：

Given a string S and a string T, count the number of distinct subsequences of T in S.

A subsequence of a string is a new string which is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (ie, "ACE" is a subsequence of "ABCDE" while "AEC" is not).

Here is an example:

S = "rabbbit", T = "rabbit"

Return 3.

## 题目大意：

给出一个字符串s和t，其中t是s的子串，求出删除s中字符得到t的不同删除方法。（或者理解为从s中取出t的不同取法。）

例如S=”rabbbit”,T=”rabbit”

其中S中的3个b取出两个得到T，有C(3,2)=3种取法，返回3.

## 题目分析：

因为可以看做是S中部分串取出得到T中部分串的子问题，所以想到用dp。

构建dp数组，其中dp[i][j]表示s[0..i]取出t[0..j]字符的方法数。那么首先考虑初始化：

dp[0][0]显然等价于s[0]==t[0],而先初始化j=0的情况，因为这时只有一个字符，扫描s，只要s[i]==t[0]，则取出的方法数+1，否则不变（很好理解，因为多出的字符与t[0]不一样，这个多出的字符没有任何意义）。

接下来从j=1（2个字符）开始考虑转移方程：

如果s[i]==t[j],那么这个多出来的t[j]的来源有两种情况，一是取自s[i],那么只要从s[0…i-1]取出t[0…j-1]即可，二是不取s[i],那么相当于从s[0…i-1]取出t[0…j]，故这种情况下的转移方程为dp[i][j]=dp[i-1][j-1]+dp[i-1][j];如果s[i]!=t[j]，那么这个多出来的s[i]对取出t[0…j]没有任何贡献，还是要从s[0…i-1]中去寻找，故dp[i][j]=dp[i-1][j].剩下的就是对一些特殊的边界情况讨论。例如s.length<=t.length。

## 源码：（language：java）

public class Solution {

public int numDistinct(String s, String t) {

int slen = s.length(),tlen = t.length();

if(slen<tlen)

return 0;

else if(slen == tlen)

return s.equals(t)?1:0;

else {

int[][] dp = new int[slen][tlen];

dp[0][0] = s.charAt(0)==t.charAt(0)?1:0;

for(int i = 1;i<slen;i++) {

if(s.charAt(i)==t.charAt(0))

dp[i][0] = dp[i-1][0]+1;

else

dp[i][0] = dp[i-1][0];

}

for(int j = 1;j<tlen;j++) {

for(int i = j;i<slen;i++) {

if(s.charAt(i) == t.charAt(j))

dp[i][j] = dp[i-1][j]+dp[i-1][j-1];

else

dp[i][j] = dp[i-1][j];

}

}

return dp[slen-1][tlen-1];

}

}

}

## 成绩：

16ms,beats 62.01%,众数20ms,14.82%

## Cmershen的碎碎念：

这是一个非常典型的DP问题，而我一开始考虑复杂了，因为我是基于组合数公式C(n,k)=C(n-1,k-1)+C(n-1,k)做的，还增加了对s[i]和t[j]是否存在的考虑，但其实没有必要。因为如果t[j]出现了s中没出现过的字符，自然前一项也为0了。

# Hard-题目13：72. Edit Distance

## 题目原文：

Given two words word1 and word2, find the minimum number of steps required to convert word1 to word2. (each operation is counted as 1 step.)

You have the following 3 operations permitted on a word:

a) Insert a character

b) Delete a character

c) Replace a character

## 题目大意：

要把word1变成word2，总共分几步？

————赵本山小品台词迷之乱入。。。

每一步可以增加一个字符、减少一个字符、修改一个字符。求把字符串word1变成word2的最小步数。

## 题目分析：

设dp[i][j]表示把word1的前i个字符变成word2的前j个字符的最小步数，首先初始化dp[0][j]=j(显然如此，因为把空串变成一个长度为j的字符串只能增加)，同理dp[i][0]=i.

接下来转移方程亦不难理解：考虑dp[i][j]，如果word1[i]==word2[j]，那么这一位就没变，只需要把前面的字符以最小步数变过来就可以，因此dp[i][j]=dp[i-1][j-1]；如果不等，则只有三种情况：

1. 把word1的前i-1个字符变成word2的前j-1个字符，再修改word1[i]为word2[j]；
2. 把word1的前i个字符变成word2的前j-1个字符，再加上word2[j]；
3. 把word1的前i-1的字符变成word2的前j个字符，再删除word[i]。

因此dp[i][j]=min(dp[i-1][j-1],dp[i][j-1],dp[i-1][j])+1.

## 源码：（language：java）

public class Solution {

public int minDistance(String word1, String word2) {

int len1 = word1.length(),len2 = word2.length();

int[][] dp = new int[len1+1][len2+1];

for (int i = 1; i<=len1; i++)

dp[i][0] = i;

for (int j = 1; j<=len2; j++)

dp[0][j] = j;

for (int i = 1;i<=len1;i++) {

for(int j = 1;j<=len2;j++) {

if(word1.charAt(i-1)!=word2.charAt(j-1)) {

int min = Math.min(dp[i-1][j], Math.min(dp[i-1][j-1], dp[i][j-1]));

dp[i][j] = min+1;

}

else

dp[i][j] = dp[i-1][j-1];

}

}

return dp[len1][len2];

}

}

## 成绩：

13ms,beats 63.16%，众数13ms,19.74%

## Cmershen的碎碎念：

相传这个算法在斯坦福大学的NLP公开课上讲过，我也是在上学期的《计算机语言学》课上了解到这个算法的。既然先入为主了，我并不觉得这道题有Hard难度。（比之前面几题各种高级数据结构……匪夷所思的算法……）

# Hard-题目14：25. Reverse Nodes in k-Group

## 题目原文：

Given a linked list, reverse the nodes of a linked list k at a time and return its modified list.

If the number of nodes is not a multiple of k then left-out nodes in the end should remain as it is.

You may not alter the values in the nodes, only nodes itself may be changed.

Only constant memory is allowed.

For example,

Given this linked list: 1->2->3->4->5

For k = 2, you should return: 2->1->4->3->5

For k = 3, you should return: 3->2->1->4->5

## 题目大意：

给出一个链表，要求k个一组地进行翻转，如果数组长度不是k的倍数，则最后一组不用翻转。**要求使用常数的空间复杂度。**

## 题目分析：

使用双指针，分别指向头结点（记为p1）和头结点后第k个节点（记为p2，如果长度不足k，则直接返回头结点），翻转p1-p2的部分（又是一个双指针），然后递归翻转p2以后的部分。

## 源码：（language：cpp）

class Solution {

public:

ListNode \*reverseKGroup(ListNode \*head, int k) {

if (!head || !(head->next) || k < 2)

return head;

// count k nodes

ListNode \*nextgp = head;

for (int i = 0; i < k; i++)

if (nextgp)

nextgp = nextgp->next;

else

return head;

// reverse

ListNode \*prev = NULL, \*cur = head, \*next = NULL;

while (cur != nextgp) {

next = cur->next;

if (prev)

cur->next = prev;

else

cur->next = reverseKGroup(nextgp, k);

prev = cur;

cur = next;

}

return prev;

}

};

## 成绩：

24ms,beats 41.82%，众数24ms,58.05%

## Cmershen的碎碎念：

本题非常考验链表指针的细节操作，稍有不慎链表后面的信息就丢失了。

# Hard-题目15：239. Sliding Window Maximum

## 题目原文：

Given an array nums, there is a sliding window of size k which is moving from the very left of the array to the very right. You can only see the k numbers in the window. Each time the sliding window moves right by one position.

For example,

Given nums = [1,3,-1,-3,5,3,6,7], and k = 3.

Window position Max

--------------- -----

[1 3 -1] -3 5 3 6 7 3

1 [3 -1 -3] 5 3 6 7 3

1 3 [-1 -3 5] 3 6 7 5

1 3 -1 [-3 5 3] 6 7 5

1 3 -1 -3 [5 3 6] 7 6

1 3 -1 -3 5 [3 6 7] 7

Therefore, return the max sliding window as [3,3,5,5,6,7].

## 题目大意：

给出一个数组和一个k长度的滑动窗口，把滑动窗口从数组开头滑到结尾，求每个滑动窗口内子数组的最大值，组成一个新数组并返回。

## 题目分析：

朴素解法是暴力枚举每个滑动窗口的最大值，时间复杂度是O(nk),虽然也能ac，但一个hard难度的题不可能只要求到这步。

根据hint，考虑用双端队列，维持队列长度小于等于k，且队头永远为滑动窗口最大值，队列后面跟的是可能成为最大值的候选解。每次加入一个元素，先看队头元素的下标是否已经在滑动窗口之外，如果在则出队，接下来判断这个元素是否大于队尾的值，如果大于则从队尾一直弹出（因为有了这个更大的数，前面的数在往右滑的过程中不可能是解），每次记录下队头的值，即为当前滑动窗口的最大值。

那么还有一个问题，如何知道队头是否滑到了窗口外面呢？答案是，我们的deque只要存的是下标就可以了。因为队列的操作全都是常数时间的，这样的时间复杂度就降到了O(n).

## 源码：（language：java）

public class Solution {

public int[] maxSlidingWindow(int[] nums, int k) {

if(nums.length==0)

return new int[0];

int[] window = new int[nums.length-k+1];

int j=0;

Deque<Integer> deque = new LinkedList<>();

for(int i = 0;i<nums.length;i++) {

while (!deque.isEmpty() && nums[deque.getLast()] < nums[i]) {

deque.removeLast();

}

deque.addLast(i);

if(deque.getFirst() == i-k)

deque.removeFirst();

if(i>=k-1)

window[j++] = nums[deque.getFirst()];

}

return window;

}

}

## 成绩：

29ms,beats 62.07%,众数31ms,9.54%

## Cmershen的碎碎念：

这道题在Leetcode里面算是难题了，且对面试来说算是比较困难的一题，Leetcode的国内山寨版Lintcode的难度评分中将其评为Very Hard难度（lintcode里面very hard难度只有两题，另一题是涉及到线段树的skyline problem），足见此题对普通面试者的难度。

此外discuss中还有一个仅4ms的解法，有机会研读一下。

# Hard-题目16：164. Maximum Gap

## 题目原文：

Given an unsorted array, find the maximum difference between the successive elements in its sorted form.

Try to solve it in linear time/space.

Return 0 if the array contains less than 2 elements.

You may assume all elements in the array are non-negative integers and fit in the 32-bit signed integer range.

## 题目大意：

给出一个无序的数组，求出排序后相邻两个元素差的最大值。如果数组长度小于2，则返回0.

## 题目分析：

懒得做了，直接按题面要求做。

## 源码：（language：java）

public class Solution {

public int maximumGap(int[] nums) {

if(nums.length<2)

return 0;

Arrays.sort(nums);

int maxgap = Integer.MIN\_VALUE;

for(int i = 1;i<nums.length;i++) {

if(nums[i]-nums[i-1]>maxgap)

maxgap=nums[i]-nums[i-1];

}

return maxgap;

}

}

## 成绩：

4ms,beats 92.15%,众数6ms,27.29%

## Cmershen的碎碎念：

不知道leetcode后台的test case怎么设计的，有很多题用朴素解法比绞尽脑汁想出的低复杂度算法还要快。

# Hard-题目17：87. Scramble String

## 题目原文：

Given a string s1, we may represent it as a binary tree by partitioning it to two non-empty substrings recursively.

Below is one possible representation of s1 = "great":

great

/ \

gr eat

/ \ / \

g r e at

/ \

a t

To scramble the string, we may choose any non-leaf node and swap its two children.

For example, if we choose the node "gr" and swap its two children, it produces a scrambled string "rgeat".

rgeat

/ \

rg eat

/ \ / \

r g e at

/ \

a t

We say that "rgeat" is a scrambled string of "great".

Similarly, if we continue to swap the children of nodes "eat" and "at", it produces a scrambled string "rgtae".

rgtae

/ \

rg tae

/ \ / \

r g ta e

/ \

t a

We say that "rgtae" is a scrambled string of "great".

## 题目大意：

给出Scramble string的定义是，把s1从其中某处拆开，两边组成一棵二叉树的左子树和右子树，再递归进行下去，直到只有一个字符，这样组成了一棵二叉树，其中叶子节点的“轮廓”依次是字符串中的每个字符。如果存在一种二叉树，把其中的一个节点的左右孩子调换，得到的“轮廓”是s2，则称s1和s2是scramble。

给出两个字符串s1和s2，判断s1和s2是不是一对scramble string.

## 题目分析：

粗看起来很难，其实可以s1和s2是scramble string只有以下几种情况：

1. s1==s2.
2. 存在i，使得s1的前i个字符和s2的前i个字符是scramble，后半部分也是scramble。（相当于左子树和右子树不用调换）
3. 存在i，使得s1的前i个字符和s2的后i个字符是scramble，s1的后半部分和s2的前半部分是scramble。（相当于从第i个点处把s1拆开，再交换左右子树。）

接下来就是递归处理了。需要做一个剪枝处理，就是当s1和s2中的字符不能一一对应时，则一定不是scramble的。

## 源码：（language：java）

public class Solution {

public boolean isScramble(String s1, String s2) {

int len1 = s1.length(),len2 = s2.length();

if(len1 != len2)

return false;

else if(len1==1)

return s1.charAt(0)==s2.charAt(0);

else if(s1.equals(s2))

return true;

else {

char[] ca1 = s1.toCharArray(),ca2 = s2.toCharArray();

Arrays.sort(ca1);

Arrays.sort(ca2);

for(int i = 0;i<len1;i++)

if(ca1[i]!=ca2[i])

return false;

boolean result = false;

for(int i = 1;i<len1;i++) {

String s11 = s1.substring(0, i);

String s12 = s1.substring(i);

String s21 = s2.substring(0, i);

String s22 = s2.substring(i);

String s23 = s2.substring(0,len2-i);

String s24 = s2.substring(len2-i);

if(isScramble(s11,s21)&&isScramble(s12,s22) || (isScramble(s11,s24)&&isScramble(s12,s23))) {

result = true;

break;

}

}

return result;

}

}

}

## 成绩：

7ms,beats 54.91%，众数4ms,21.84%

## Cmershen的碎碎念：

这道题还可以用dp来消除递归，我一开始尝试的是记忆化搜索，但没想到Java中有什么合适的数据结构来记录搜索过的字符串。还有这个字符串的切分很适合用python写（比如用[i:],[:i]等很简洁的表示法），但我python写的太少了……语法不太熟练……

# Hard-题目18：51. N-Queens

## 题目原文：

The n-queens puzzle is the problem of placing n queens on an n×n chessboard such that no two queens attack each other.



Given an integer n, return all distinct solutions to the n-queens puzzle.

Each solution contains a distinct board configuration of the n-queens' placement, where 'Q' and '.' both indicate a queen and an empty space respectively.

## 题目大意：

给出n，输出N皇后问题的所有解。（别告诉我没做过八皇后，计算机专业的都懂得……）

## 题目分析：

使用回溯，每放一个皇后判断一下这个位置是否可以被之前的皇后吃掉，如果可以就继续放下去，然后搜索下一行，直到最后一行放上了皇后。

## 源码：（language：java）

public class Solution {

public List<List<String>> solveNQueens(int n) {

List<List<String>> list = new ArrayList<List<String>>();

backtrack(n,0,list,new boolean[n][n]);

return list;

}

private void backtrack(int n,int queens,List<List<String>> list,boolean[][] chessboard) {

if(n==queens) {

list.add(convert(chessboard,n));

}

else {

for(int j = 0;j<n;j++) { // press a queen at chessboard[queens][j]

boolean canPress = true;

for(int s = 0;s<queens;s++) {

if(!canPress)

break;

for(int t = 0;t<n;t++) {

if(chessboard[s][t]) {

if(t==j || Math.abs(s-queens)==Math.abs(t-j)) {

canPress = false;

break;

}

}

}

}

if(canPress) {

chessboard[queens][j] = true;

backtrack(n, queens+1, list, chessboard);

chessboard[queens][j] = false;

}

}

}

}

private List<String> convert(boolean[][] chessboard,int n) {

List<String> list = new ArrayList<String>();

for(int i = 0;i<n;i++) {

String line = "";

for(int j = 0;j<n;j++) {

if(chessboard[i][j])

line+="Q";

else

line+=".";

}

list.add(line);

}

return list;

}

}

## 成绩：

31ms,beats 2.57%,众数6ms，19.35%

## Cmershen的碎碎念：

本题成绩较差，一是没有进行任何剪枝，二是构造了辅助数组chessBoard存储棋盘，每找到一个解就遍历棋盘转换成题目要求的格式。这两点都是开销很大的。

另外在此感叹一下，八皇后是一道非常经典的回溯问题，可以说是当年学习C语言时的第一道坎。四年前当我们的计导课第一次讲到八皇后的问题时，那是一道很多人为之倾倒、乃至为之刷夜的难题，但如今却可以写出优雅的回溯。说明我们编程能力真的已经得到很大的提升了。

# Hard-题目19：99. Recover Binary Search Tree

## 题目原文：

Two elements of a binary search tree (BST) are swapped by mistake.

Recover the tree without changing its structure.

Note:

A solution using O(n) space is pretty straight forward. Could you devise a constant space solution?

## 题目大意：

二叉排序树上的两个节点互换位置了。你能用O(1)的空间复杂度把这两个节点恢复过来吗？

## 题目分析：

抓住二叉排序树的重要性质——中序遍历递增来解决此题。中序遍历这棵树，如果发现了gap（当前节点比前驱节点值小）则记录下来，要分两种情况，如果gap为1个，就是相邻的两个节点互换了，如果为两个，就是第一个gap的左节点和第二个gap的右节点互换了，分别记录一下互换的节点就好了，每次用一个类成员变量记录上次访问的节点，故空间复杂度为O(1).

## 源码：（language：java）

public class Solution {

//private LinkedList<TreeNode> trace = new LinkedList<TreeNode>();

private int gap = 0;

private TreeNode lastvisit,node1,node2;

public void recoverTree(TreeNode root) {

if(root!=null) {

inOrderVisit(root);

}

int temp = node1.val;

node1.val = node2.val;

node2.val = temp;

}

private void inOrderVisit(TreeNode root) {

if(root!=null) {

inOrderVisit(root.left);

if(lastvisit!=null && root.val<lastvisit.val) {

if(gap == 0)

node1 = lastvisit;

node2 = root;

gap++;

}

lastvisit = root;

inOrderVisit(root.right);

}

}

}

## 成绩：

4ms,beats 42.88%,众数4ms,54.80%

## Cmershen的碎碎念：

本学妹不是很清楚，二叉树递归算O(logn)空间复杂度吗？

# Hard-题目20：138. Copy List with Random Pointer

## 题目原文：

A linked list is given such that each node contains an additional random pointer which could point to any node in the list or null.

Return a deep copy of the list.

## 题目大意：

深度拷贝一个链表，这个链表是普通单链表的基础上增加了一个random指针，可能指向链表的任何一个节点，故称为随机链表。

## 题目分析：

要遍历两遍原链表，第一遍不带random节点复制原链表，并用一个哈希表建立原链表和新链表的映射关系，其中原链表的第i个节点是key，新链表的第i个节点是value（想想为什么不能反过来？）。扫第二遍的时候复制random节点，原链表和新链表同时从头开始扫，令新链表当前节点的random值等于原链表random在哈希表中的映射。总复杂度O(n).

## 源码：（language：cpp）

class Solution {

public:

RandomListNode \*copyRandomList(RandomListNode \*head) {

map<RandomListNode \*,RandomListNode \*> m;

m[NULL]=NULL;

RandomListNode \*t=new RandomListNode(0);

RandomListNode \*p1=head,\*p2=t;

while(p1)

{

p2->next=new RandomListNode(p1->label);

m[p1]=p2->next;

p1=p1->next;

p2=p2->next;

}

p1=head;

p2=t->next;

while(p1)

{

p2->random=m[p1->random];

p1=p1->next;

p2=p2->next;

}

p2=t->next;

return p2;

}

};

## 成绩：

128ms,beats 19.31%,众数108ms,19.45%

## Cmershen的碎碎念：

一开始我的思路是用一个哈希表把链表的每个节点映射到[1,length]中，然后也是第一遍复制不带random的，第二遍并行扫两个链表，得到原来的random指向的“下标”，再在新链表里面从头开始找，但这样的时间复杂度就为O(n2)了，不可取。

# Hard-题目21：123. Best Time to Buy and Sell Stock III

## 题目原文：

Say you have an array for which the ith element is the price of a given stock on day i.

Design an algorithm to find the maximum profit. You may complete at most two transactions.

## 题目大意：

给出一个数组，代表一个股票每一天的股价。现在你最多可以买卖两次，求出最大利益。

## 题目分析：

使用两个dp数组，dp1从正向扫描，其中dp1[i]表示0~i天买卖**一次的最大收益，**dp2从反向扫描，dp2[i]表示第i~n-1天买卖**一次的最大收益**。然后再并行扫一遍dp1和dp2，计算dp1[i]+dp2[i+1]的最大值。

dp1因为是正向扫的，所以维护当前子序列的最小值min，可以由dp[i-1]推出dp[i]:

dp[i]=max(dp[i-1],prices[i]-min)，即如果在第i天卖了可以获得更大的收益，则在第i天卖掉，否则与在第i-1天之前卖是一样的。

同理，维护最大值max可以由dp2[i+1]推出dp2[i]:

dp2[i]=max(dp2[i+1],max-prices[i])，道理同上。

最后有的同学还会提出这样的问题：如果只买卖一次是最优的呢？

没关系，dp1[n-1]就是一次买卖的最大收益。

## 源码：（language：java）

public class Solution {

public int maxProfit(int[] prices) {

if(prices.length<2)

return 0;

int[] dp1 = new int[prices.length];

int[] dp2 = new int[prices.length];

int min = prices[0];

for (int i = 1; i < prices.length; i++) {

if (prices[i] < min)

min = prices[i];

dp1[i] = Math.max(dp1[i - 1], prices[i] - min);

}

int max = prices[prices.length - 1];

for (int i = prices.length - 2; i >= 0; i--) {

if (prices[i] > max)

max = prices[i];

dp2[i] = Math.max(dp2[i + 1], max - prices[i]);

}

int profit = 0;

for (int i = 0; i < prices.length - 1; i++) {

if (dp1[i] + dp2[i + 1] > profit)

profit = dp1[i] + dp2[i + 1];

}

return max;

}

}

## 成绩：

3ms,beats 42.07%,众数2ms,38.19%

## Cmershen的碎碎念：

一开始我想到的是套用股票第1题的函数求dp1和dp2，这样需要n2的复杂度，提交上去超时了。然后发现dp1可以用左边项来推，dp2也可以用右边项来推。这样就降到O(n)时间复杂度了。