## 1. Two Sum

hashmap哈希表记录之前的数

## 2. Add Two Numbers

逐位相加

## 3. Longest Substring Without Repeating Characters

hashmap记录位置

## 5. Longest Palindromic Substring

方法1、方法2 见leetcode1

方法3：直接动态规划，base case除了dp[i][i]之外还有di[i][i+1]。时间复杂度n^2，空间n

方法4：Manacher's algorithm马拉车算法。首先把每一字符前后都加上一个特殊字符，首尾再各加一个特殊字符。这样每个回文串都是奇数个字符，而且这个字符的半径就是原来字符的长度。假设有一个数组记录每个点的回文半径，那么原来字符的位置索引就是新位置的索引减去半径再除以2。然后求这个半径数组p，假设i是当前记录的半径点，而它的半径能到最右边的r，那么对于我们需要求的半径点j，它有两种情况，一是就是i，那么我们就按照方法2往两边展开一下就好了；如果j不是i，那么j相对于r有两种情况，一种大于r，那么就还是一次展开，同时更新i和r；一种是小于等于r，那么我们就可以找到j相对于i在i之前的对称点，因为以i为中心，半径为r的之间的串是对称的，那么j的半径不小于对称点长度，同样如果超过r，那就只能把超出部分依次比较，然后更新i和r。由于每次r都是只增加不减少的，所以虽然有两层循环，时间复杂度是n，空间复杂度是n

<https://leetcode.com/problems/longest-palindromic-substring/solution/>

<https://en.wikipedia.org/wiki/Longest_palindromic_substring#Manacher's_algorithm>

<https://leetcode.wang/leetCode-5-Longest-Palindromic-Substring.html>

<https://www.cnblogs.com/grandyang/p/4475985.html>

## 6. ZigZag Conversion

数学分析。每组个数是2\*numRows-2，然后每个元素的位置都有对应

## 7. Reverse Integer

乘以10之前check是不是会overflow

## 8. String to Integer (atoi)

按步骤，要考虑整数溢出

## 9. Palindrome Number

方法1：转换为整数，或者用数组存每一位，需要logn的额外空间

方法2：把这个数反转一半，看看是不是和前一半相等，类似于7题只做一半。需要考虑的是特殊情况，比如这个数末位是0，还有就是数位是偶数还是奇数

<https://leetcode.com/problems/palindrome-number/solution/>

## 10. Regular Expression Matching

方法1：enumerate+DFS。每次遇到星号，都可以选择匹配0到多个，将这些情况都压入栈中遍历，找到一种能都匹配到结尾的办法，时间复杂度最坏是指数级

方法2：动态规划dynamic programming。dp[i][j]表示s[i]到p[j]能不能匹配上。分为两种情况，一种是p[j]不是星号，如果是点或者相等，那么dp[i][j] = dp[i-1][j-1]，否则false；一是p[j]是星号，那么可以看情况是向前匹配一位，还是直接不匹配，其他情况蕴含在前一个子问题中了。本题可以用dp重点在于子问题ij能不能匹配非常有价值，匹配过程就可以看做是一段一段的。实现上，dp一般要在原有长度上加一以便于实现。时间复杂度mn

<https://leetcode.com/problems/regular-expression-matching/solution/>

## 11. Container With Most Water

two pointers两个指针，总是移动小的就行了

## 12. Integer to Roman

逐个转化greedy，或者直接hardcode，每一位对应的字符固定

https://leetcode.com/problems/integer-to-roman/solution/

## 13. Roman to Integer

逐个转换

## 14. Longest Common Prefix

方法1：横向扫描。定义最长的prefix结尾是end，end之前逐一比对

方法2：纵向扫描。先比较每一个单词的第一个字母是不是相等，在比较第二个，等等

<https://leetcode.com/problems/longest-common-prefix/solution/>

## 15. 3Sum

方法1：hashmap，设定两个指针指向前两个i和j，然后找k，一开始用hashmap记录一下位置，就可以找快速找k，为了避免重复，设定ijk递增。

方法2：sorting + two pointers。这种方法不仅适用于相等的情况，还适用于不等的情况。见16题

<https://leetcode.com/problems/3sum/solution/>

## 16. 3Sum Closest

sorting + two pointers。排序。然后首先固定一个起点，第一个数是起点，那么接下来就是在起点之后找两个数，使得三个数相加最接近target，由于后半段是排好序的，那么可以用两个指针指向首尾，如果和大于target，移动末尾的指针，反之移动开始的指针。时间复杂度依然是n^2，空间复杂度取决于排序，为了达到1可以用冒泡排序或者堆排序。

<https://leetcode.com/problems/3sum-closest/solution/>

## 17. Letter Combinations of a Phone Number

回溯，或者DFS、BFS遍历

## 18. 4Sum

方法1：brute force。同1题，15题，可以用双指针或者哈希表，kSum的时间复杂度是n^(k-1)。

<https://leetcode.com/problems/4sum/solution/>

方法2：分治。转换为两个two sum问题。即先算出所有两个数相加的可能，存在哈希表中，然后再遍历所有两个数相加的可能，我们就可以在O(1)的时间找到我们需要的另外两个数相加的和。实现上为了去重还是先数每个数出现多少次，然后去掉重复排序。时间复杂度n^2

## 19. Remove Nth Node From End of List

two pointers。两个指针相隔n，第一个指针到终点，第二个指针指向要删除的位置的前一个

## 20. Valid Parentheses

stack栈

## 21. Merge Two Sorted Lists

依次合并

## 22. Generate Parentheses

DFS+BFS遍历。遍历的时候保留开口数和可闭口数，可开口，那开口，可闭口，那闭口。时间复杂度4^n/n^0.5

## 23. Merge k Sorted Lists

堆heap。分治法见leetcode1

## 24. Swap Nodes in Pairs

三个指针three pointers，分别指向前一个，第一个要换的，第二个要换的，即可

## 25. Reverse Nodes in k-Group

Two Pointers。一个指向这k个元素的前一个，一个指向k个元素的最后一个，在用三个指针就可以把这之间的元素翻转。需要注意的是改变顺序的次序

## 26. Remove Duplicates from Sorted Array

Two Pointers。一个指向新数组的末端，一个指向当前数据的位置，找到当前数据之后直接复制过去即可，没有什么交换

<https://leetcode.com/problems/remove-duplicates-from-sorted-array/solution/>

## 27. Remove Element

Two Pointers。同26题

## 29. Divide Two Integers

方法1：循环相减。每次用被除数减去除数，直到不能再减。时间复杂度n

方法2：分治divide and conquer。每次被除数除以2直到小于除数，然后从后往前算出每个数除以x得到的商和余数，最后一个数显然商是0，余数是本身。根据后一个数，当前数的商是后一个数的商的两倍，余数是两倍再加上除以二得到的余数。如果余数大于除数，则余数减去除数，商加一，可以证明这里余数最多是除数的两倍，所以用循环相减法。最后我们可得整体的商和余数。时间复杂度logn

方法3：Repeated Exponential Searches指数级搜索（同leetcode1）。把除数乘以2^1,2^1,2^2,…,2^n的结果记录，直到大于被除数。那么2^(n-1)就是商的第一部分，用被除数减去可得余数，这时余数依然很大，但是可以再次重复得到商的第二部分。由于除数乘以2的指数的结果每次都一样，而且余数总是小于被除数，所以可以只记录最后一个数，其他的通过除以二（右移）做到。实现中，为了让奇数和偶数都能通过右移实现除以2，我们先加一，再除以2。时间复杂度为logn，空间复杂度1。

## 31. Next Permutation

分析法。分析发现从后向前找到第一个数它大于前一个数，也就是它后面的数都是按照倒序拍好的，那么我们只要在后面找到第一个大于它前一个数的数，然后与前一个数交换，然后把这个数向后的位置反转。如果这个数不存在，则说明整个数组是倒有序的，就反转整个数组

## 33. Search in Rotated Sorted Array

binary search。利用性质就是左边数组的最小值比右边数组的最大值大

## 34. Find First and Last Position of Element in Sorted Array

方法1：binary search。两次搜索分别找上限和下限

<https://leetcode.com/problems/find-first-and-last-position-of-element-in-sorted-array/solution/>

方法2：先找到一个出现的位置，假设这时start是s，end是e，那么第一次出现，就在s-m或者0-s之间，取决于a[s]是不是等于target。同理第二次出现在m-e或者e-$之间。可以减少比对次数

## 35. Search Insert Position

binary search。找存在和位置同时进行，最后返回start即可，start位置上就是第一个大于它的数

<https://leetcode.com/problems/search-insert-position/solution/>

## 36. Valid Sudoku

方法1：逐个比较即可，我的加速方法是每一列和每一个九宫格永久维持一个set，每一行临时维持一个set，判断行的同时也看列和九宫格。

方法2：大神方法，把所有信息整合到一个hashset中，每次添加的信息为 “4 在行 7”这种，这样也能很快判断重复，而且不用维持那么多hashset了

<https://leetcode.com/problems/valid-sudoku/discuss/15472/Short%2BSimple-Java-using-Strings>

## 38. Count and Say

理解题意，依次count and say即可

## 39. Combination Sum

见leetcode3

## 40. Combination Sum II

见leetcode3

## 41. First Missing Positive

index as hash key。首先，分析得知这个数一定在1-n+1之间，那么除了这之间的数，我们都可以去掉。然后，我们采用对于每一个出现的数，我们把对应的index上的数改成负数，最后只需要看哪个index上的数是正数就知道它没有出现了。可是这样我们发现对于之前想要去掉的数，我们没有办法通过它变成任意一个数来达到想要的效果。所以，我们先取一个数出来，发现1这个数很合适，如果它不在，那我们返回的就是它，而如果它在，我们把想删除的数改成它即可。时间复杂度n，空间复杂度1

<https://leetcode.com/problems/first-missing-positive/solution/>

## 43. Multiply Strings

按照乘法规则逐位相乘相加即可。为了代码的方便，这里首先将字符倒序转为数组，可知第i位和第j位相加的结果在第i+j位，处理进位容易。取出开始的0也容易

## 44. Wildcard Matching

方法1：动态规划。特殊情况是，如果j位上是\*，那么如果任意一个i能和j-1使得dp[i][j-1]=True，那么dp[i][j]就是true。为了实现这一特点，可以在每个是\*的位置记录之前有没有i可以匹配。总体时间复杂度sp，其中sp代表s和p的长度，空间复杂度sp，可以优化到1

方法2：回溯法backtracking。分析发现，每个\*可以匹配任意一个或者多个字符，而如果依次匹配使得s和p不匹配的时候，我们只需要回溯到上一个星号，修改它匹配的字符即可。可以证明修改之前任意一个或者多个星号的行为都可以通过修改最近的星号的匹配字符数来实现。可以证明平均时间复杂度slogp

<https://leetcode.com/problems/wildcard-matching/solution/>

## 45. Jump Game II

### 55. Jump Game

two pointers + greedy。第一个指针总是指向当前位置，第二个指针指向当前能到的最远位置，当第一个指针小于第二个指针时，用当前位置能跳的最远位置更新最远位置，然后当前位置加一，最后只要第二个指针超出结尾就说明结尾一定可达。时间复杂度n，空间1

### 45. Jump Game II

方法1：BFS。第一个点是第一个group，它能到达的所有点是第二个group。依次，下一个group能到达的所有点就是上一个group能到达且比当前位置远的点，直到有一个点包含了终点，复杂度是2n，也就是n。内循环只增不减，所以虽然两层循环，但是是2n

方法2：greedy。观察第一个方法发现下一个group能到的最远的点就是上一个group中所有点能到最远点的最大值。而且每一次步数增加，就是在每一个group结束，进入下一个group就要增加一步。所以我们记录下当前group的终点，然后往当前group遍历，来更新下一个group的终点。如此循环，知道包含终点。时间复杂度n

<https://leetcode.com/problems/jump-game-ii/solution/>

## 46. Permutations

方法1：recursion + DFS。见leetcode-字节跳动

方法2：backtracking。首先选定第一个数，然后把这个数和第一个数调换位置，然后再递归把后面打乱即可，注意回退的时候要把换过顺序的两个数复原。递归终止条件是所有数都被打乱过了，这时候把打乱后的数组复制到结果里面即可，这样没有额外空间

## 47. Permutations II

方法1：recursion + DFS。见leetcode-字节跳动。缺点是每次都要复制频率字典，开销很大

方法2：backtracking。同46题，我们每次需要选定一个数放在当前位置，然后在递归找下一位，之前我们是直接从后面随便提一个数，但是现在有重复，所以我们从counter里面提一个数，这个数的剩余频率要大于零，我们把剩余频率减一然后递归，回来的时候把频率再加上。这样可以减少很多复制。空间复杂度是n

48. Rotate Image

50. Pow(x, n)

## 51. N-Queens

backtracking。按照行遍历，保证每行一个，对于列，设置一个集合，哪一列有queen，把索引放进去。对于对角线，观察发现对于正对角线，row-col是定值，对于反对角线，row+col是定值。所以同样设置两个集合，如果这条对角线上有queen，就把相应的row±col放进去

<https://leetcode.com/problems/n-queens/solution/>

## 52. N-Queens II

同51题，把结果改为计数即可

## 53. Maximum Subarray

动态规划。dp[i]是以i为结尾的最大子数组。那么，对于第i个数，如果加上之前的数反而小于自己，那么就新开一个子串，否则就跟在前面的串里面。时间复杂度n，空间复杂度1

或者理解为，用t表示当前的最大值，g表示全局最大值，每次t都加上后面的数，然后更新r，下一轮的时候，如果t小于零，那么必定是开始新的，所以把t重置为0

<https://leetcode.com/problems/maximum-subarray/solution/>

## 54. Spiral Matrix

一层一层的遍历即可，注意到每一层都是遍历四周，然后下一轮开始的起点是(i, i)。边界情况就是最后一圈可能没有四周，只有两周或者一周

<https://leetcode.com/problems/spiral-matrix/solution/>

## 55. Jump Game

见leetcode4-45

## 56. Merge Intervals

排序，然后遍历intervals，不修改input，新建list存结果，遍历的时候每次只要看看能不能与前一个合并即可。时间复杂度nlogn，空间取决于排序

<https://leetcode.com/problems/merge-intervals/solution/>

## 57. Insert Interval

方法1：binary search。建立在可以修改input的基础上，找到起终点，合并，然后删掉不必要的。但是边界情况及其复杂，而且如果需要删除，复杂度仍然是n

方法2：greedy。遍历intervals，每次看看当前的和之前以及给定的interval能不能合并，能就合并，不能就复制下来。不修改输入，复杂度始终是n

<https://leetcode.com/problems/insert-interval/solution/>

## 58. Length of Last Word

rfind或者split或者直接循环遍历

<https://leetcode.com/problems/length-of-last-word/solution/>

## 59. Spiral Matrix II

同55题，取值变赋值即可

<https://leetcode.com/problems/spiral-matrix-ii/solution/>

## 60. Permutation Sequence

数学方法。考虑第一个数，确定第一个数后面有n-1的阶乘种可能，所以k是(n-1)!的几个倍数，第一个数就应该选几，然后k再减去(n-1)!的整数倍。注意剩下的倍数和值之间的对应关系发生了变化，要删掉这个选过的数，所以我们用一个list依次记录没有选过的数，这样删除选过的值剩下的键值对刚好满足关系。剩余位的选值依次类推。时间复杂度n^2

答案说的太复杂没有看，不过看代码跟我的答案一样，开心

<https://leetcode.com/problems/permutation-sequence/solution/>

## 61. Rotate List

方法1：答案中的方法。首先找到list的长度n，顺便把list首尾相接。然后新的头在n-k%n，新的尾结点在n-k%n-1，所以找到第n-k%n-1个节点，他的下一个节点就是返回值，而我们再把它的next设置为none就可以解开环。时间复杂度2n，空间1

<https://leetcode.com/problems/rotate-list/solution/>

方法2：我的方法。已知我们就是要把倒数k%n的节点设为新的头，我们用三个指针，一个指向头，一个指向这个元素，一个指向结尾就总是可以实现这个变换，所以我们只要使得两个指针相隔k走到结尾即可，当前k大于长度的情况可以在相隔k的时候一起判断。时间复杂度2n，空间1

## 62. Unique Paths

排列组合，，计算方式一是用math库里面的阶乘算，一是用scipy.special中的comb

<https://leetcode.com/problems/unique-paths/solution/>

## 63. Unique Paths II

动态规划dynamic programming。每个点的路径等于右边的路径加下面的路径，前提的是自己不是石头。时间复杂度mn，空间如果用原数组，就是n，否则是min(m,n)

<https://leetcode.com/problems/unique-paths-ii/solution/>

## 64. Minimum Path Sum

<https://leetcode.com/problems/minimum-path-sum/>

动态规划dynamic programming。一维空间，把列增加一，使得不需要判断边界。这一列初始的时候是0，后来是无穷大

65. Valid Number

<https://leetcode.com/problems/valid-number/solution/>

## 66. Plus One

逐位相加即可

## 67. Add Binary

方法1：用内置函数，int(a,b)可以将a转为b进制的整数。format(a,b)可以按照b代表的要求来format参数a。缺点是除了Python之外的语言都会面临字符串太长时候的问题

方法2：逐位相加，缺点是慢，复杂度高

方法3：位运算。同leetcode3-371题。时间复杂度是m+n

<https://leetcode.com/problems/add-binary/solution/>

## 68. Text Justification

按要求做即可，假设s表示总空格数，k表示剩下的空档，那么当前的空格数就是s/k上取整

## 69. Sqrt(x)

方法1：binary search二分搜索，x大于等于2之后，x的开跟一定在x和1/2x之间，二分搜索即可，时间复杂度logn

方法2：Newton's Method牛顿法。转为求f(a)=a\*\*2-x=0，f’(a)=2a，，初始a是x。然后不断缩小。类似leetcode3-367题，时间复杂度logn且比其他logn方法收缩的更快

<https://leetcode.com/problems/sqrtx/solution/>

<https://en.wikipedia.org/wiki/Newton%27s_method>

方法3：指数相加。从1开始，每次加2,4,8...直到超过x，然后又开始新一轮的2的指数的相加

## 70. Climbing Stairs

方法1：动态规划之后发现是1,2开头的Fibonacci Number的第i位，时间复杂度n，空间1

方法2：直接用Fibonacci Number的公式，公式是1，1开头的

## 71. Simplify Path

<https://leetcode.com/problems/simplify-path/>

用split函数，然后遍历即可

## 72. Edit Distance

<https://leetcode.com/problems/edit-distance/>

dynamic programming。可以证明所有的insert都可以通过delete来实现，所以如果两个字符相等，那么最小值就是去掉这两个字符的值，如果不等，那么有三种可能

## 73. Set Matrix Zeroes

O(1)空间复杂度的方法是用最左边一列和最上边一行来做标记，也就是如果这一列需要归零，把对应位置标上，为了和这一列原有的0区分，先遍历这一行和一列看看有没有0

## 74. Search a 2D Matrix

<https://leetcode.com/problems/search-a-2d-matrix/>

binary search。假设flatten开来，那么索引i回到原数组的索引就是i//n, i%n，从而直接二分搜索

## 75. Sort Colors

two pointers。两个指针分别指向第一个1和最后一个1的位置，还有一个是当前遍历的位置，0往前调，2往后调

<https://leetcode.com/problems/sort-colors/solution/>

76. Minimum Window Substring

## 77. Combinations

Backtracking。可以通过算出当前位置的最小值和最大值，然后在这之间取值保证是合理的来加速。答案另一种方法根本看不懂lol

## 78. Subsets

方法1：BFS/DFS。每次从栈里取出来的时候，都把当前值加入结果，然后从后面遍历找值，放回栈，时间复杂度2^n

.方法2：Cascading。对于每个元素，都可以放或者不放，所以每个元素都要把结果数组拿出来复制一份然后把这个元素放进去，原来的那份就是不放

方法3：Backtracking。没啥说的，每次都把当前值放进去就行

方法4：Lexicographic (Binary Sorted)。还是用一个二进制数来表示当前的结果，如果数位上是1，那么取这个值，否则不取。生成这个mask的办法是从2^n到2^(n+1)这些数除去第一位刚好满足要求

<https://leetcode.com/problems/subsets/solution/>

## 79. Word Search

backtracking+DFS。找到起点，然后dfs下去，用backtracking

## 80. Remove Duplicates from Sorted Array II

two pointers+Overwriting unwanted duplicates.一个指向当前写到的位置，一个指向当前遍历到的位置，符合条件就写到当前位置同时后移，否则只后移遍历

<https://leetcode.com/problems/remove-duplicates-from-sorted-array-ii/solution/>

## 81. Search in Rotated Sorted Array II

binary search。主要是如果中间的和尾部的相等，那么没有办法判断往哪里走，只能一个个比对，我的方法是把中间向尾部移动，如果一直相等，那么答案在前面，如果不是，那么答案在新位置到尾部。总归是不浪费所有的比较

<https://leetcode.com/problems/search-in-rotated-sorted-array-ii/solution/>

## 82. Remove Duplicates from Sorted List II

首先搞一个Sentinel Head。然后三个指针，当前添加位置，当前遍历位置，当前遍历位置的前一个，当前遍历位置跟前后都不相同就加入。答案有别的方法，比较复杂

<https://leetcode.com/problems/remove-duplicates-from-sorted-list-ii/solution/>

## 83. Remove Duplicates from Sorted List

如果下一个和自己相同，就跳过

## 84. Largest Rectangle in Histogram

ascending stack。见leetcode1-84.保持栈递增，当遇到一个数比栈头小，那么对于栈头元素来说，它的右边界就是当前比它小的这个数（不包含这个数），而它的左边界就是栈中它前面的元素（不包含这个元素），那么包含这个元素的最大面积就可计算了。实现上，我们往高度数组尾端推入-1，往栈里面也推入-1，这样就好像给两边加上了-1的边界。如果是不支持负数索引的函数，就往栈里面推入-1即可。时间复杂度n，空间复杂度n

<https://leetcode.com/problems/largest-rectangle-in-histogram/solution/>

## 85. Maximal Rectangle

首先，我们遍历每一行，把二维数组转为一维，每个数表示这一点能达到的最大高度

方法1：转为84题，见leetcode4-84题

方法2：dynamic programming。用left和right两个数组分别表示第j个数最大高度情况下左右两边能到达的边界。这个边界可以根据上一次的边界和这一次的0,1情况常数时间得到

<https://leetcode.com/problems/maximal-rectangle/solution/>

## 86. Partition List

Sentinel Head + two pointers.分成两个列表，依次往里面加东西就行

## 88. Merge Sorted Array

<https://leetcode.com/problems/merge-sorted-array/>

方法1：先把第一个数组向后复制，使得最后一个元素到达尾端，然后再合并，时间复杂度m+n，但其实是2m+n，多一次复制。空间复杂度1

方法2：直接从后向前合并，这样也是从后向前写入，没有复制了，时间复杂度m+n

## 89. Gray Code

找规律

方法1：动态规划dynamic programming。我们发现n+1的结果就先是n的结果，然后再把n的结果镜像且最左边补1。我们可以直接用结果数组来迭代，所以空间复杂度1

方法2：数学分析。可知对于格雷码。证明有点小复杂

<https://leetcode.com/problems/gray-code/solution/>

<https://blog.csdn.net/u012501459/article/details/46790683>

## 90. Subsets II

同leetcode1-78题，但是每次加入的是之后的不重复的值，所以要先排序，distinct number就在边缘处

<https://leetcode.com/problems/subsets-ii/solution/>

## 91. Decode Ways

dynamic programming。如果当前位置单独有效，那么就等于前面字符的分类数，如果当前位置和前面一位可以组成两位数且有效，那么还要加上去掉这两位数的方法。简而言之，就是一个字符有两种解释方法，要不单独，要不和前面一起。时间n，空间可以优化到1

<https://leetcode.com/problems/decode-ways/solution/>

## 92. Reverse Linked List II

### 206. Reverse Linked List

方法1：迭代。只需要两个指针，一个是current，一个previous，将current指向previous即可，初始时previous是none。时间n，空间1

方法2：迭代。用三个指针，分别是每次将列表的下一个元素调到最前面来。三个指针分别是头，当前需要调的元素和当前元素的前一个

方法3：递归。从后往前，假设后面n-1个都反序了而且返回了新的头，注意这时当前元素刚好指向新链表的末尾，所以把当前元素添加到末尾并将自己的next设置为none即可。时间复杂度n，空间复杂度n

<https://leetcode.com/problems/reverse-linked-list/solution/>

### 92. Reverse Linked List II

将第一个指针移到起始的位置，然后从这个位置开始翻转right-left个元素，同206题有三种方法。

<https://leetcode.com/problems/reverse-linked-list-ii/solution/>

## 93. Restore IP Addresses

Backtracking (DFS)。一个个去试即可，注意每个字符最长是3，所以最多只有27种可能

## 94. Binary Tree Inorder Traversal

方法1：递归

方法2：用栈，左节点先入栈，退栈时访问自己，然后推右节点。为了避免无限循环，加一个指针指向当前栈遍历到的位置。有两种方法，一种是入栈的时候直接入到最左节点，这样退栈的时候左子树一定是访问过的。一种是退栈的时候退出所有没有右节点的节点，这样保证左子树不会被重复访问。时间和空间都是n

方法3：Morris Traversal。见leetcode1-94.简而言之就是如果我们访问的节点是有左节点的，那么当前节点的访问顺序就是在访问玩左子树的最后一个节点再访问，这最后一个节点就是左子树的最右节点，所以只要当前节点有左子树，我们就把当前节点链到左子树的最右节点的有节点，把根重置为左子树的根。如果没有左子树，那么就是访问自己，然后右节点，也就是把自己推入结果，指针移到右节点即可。这样会修改树，最后树退化为链表就是答案。时间复杂度n，空间复杂度1

<https://leetcode.com/problems/binary-tree-inorder-traversal/solution/>

## 95. Unique Binary Search Trees II

### 96. Unique Binary Search Trees

方法1：动态规划。见leetcode1-95

方法2：Mathematical Deduction数学分析Catalan number

<https://leetcode.com/problems/unique-binary-search-trees/solution/>

<https://en.wikipedia.org/wiki/Catalan_number>

### 95. Unique Binary Search Trees II

动态规划。同96题，但是这题需要具体实现，用递归，把左子树的列表和右子树的列表交叉遍历即可得结果

<https://leetcode.com/problems/unique-binary-search-trees-ii/solution/>

## 96. Unique Binary Search Trees

见leetcode4-96

## 97. Interleaving String

动态规划。dp[i][j]表示s3前i+j个字符能不能被s1的i个字符和s2的j个字符组成。注意这个题目只要是interleave有两个条件是自然满足的。时间复杂度mn，空间可优化到n

98. Validate Binary Search Tree

<https://leetcode.com/problems/validate-binary-search-tree/solution/>

## 100. Same Tree

递归，或者比对BFS遍历顺序

<https://leetcode.com/problems/same-tree/solution/>

## 101. Symmetric Tree

方法1：BFS迭代。BFS每一层都应该是对称的

方法2：递归recursion。一棵树对称的条件是左右两颗子树是镜像，左右两颗子树是镜像的条件是根相等，然后左边的左子树是右边右子树的镜像，且左边的右子树是右边左子树的镜像。时间复杂度n，空间复杂度n

<https://leetcode.com/problems/symmetric-tree/solution/>

## 102. Binary Tree Level Order Traversal

BFS

## 103. Binary Tree Zigzag Level Order Traversal

方法1：BFS。遍历完reverse，或者每次都从后往前，用一个标志位记录是不是先添加右子树即可。时间n，空间n

方法2：DFS。用一个变量记录层数即可，双层反序，时间n，空间logn。但是反序添加有点n2，用deque，或者加完反序

<https://leetcode.com/problems/binary-tree-zigzag-level-order-traversal/solution/>

## 104. Maximum Depth of Binary Tree

DFS。迭代的话带深度就行

## 107. Binary Tree Level Order Traversal II

方法1：同leetcode4-102题，BFS然后翻转

方法2：DFS带level，带level可以知道最后数组放在什么位置

<https://leetcode.com/problems/binary-tree-level-order-traversal-ii/solution/>

## 108. Convert Sorted Array to Binary Search Tree

递归recursion。根在中间，用左边构造左子树，右边构造右子树

答案里面很多关于BST的知识很有用

## 109. Convert Sorted List to Binary Search Tree

方法1：递归。跟数组一样，找中点，然后递归。虽然每次找中点的复杂度是n，但是总共找logn次，所以复杂度是nlogn，空间是logn

方法2：转为数组，时间n，空间n

方法3：Inorder Simulation模拟中序遍历。见leetcode1-109.BST的中序遍历是递增数组，所以对数组进行中序遍历模拟，也就是先找左子树，然后把链表后移一位，模拟访问自己，然后找右子树。为了知道左子树的边界，先要遍历链表，找到长度为n，那么中间就在n/2。我们不需要真正找到这个中点，只需要明确左子树遍历的边界即可。时间复杂度n，空间复杂度logn

<https://leetcode.com/problems/convert-sorted-list-to-binary-search-tree/solution/>

## 110. Balanced Binary Tree

递归recursion。递归找左右子树高度，如果左右子树不平衡或者高度差大于1，不行。时间复杂度n，空间复杂度n

## 111. Minimum Depth of Binary Tree

递归，dfs，bfs

## 112. Path Sum

递归，dfs，bfs

## 113. Path Sum II

dfs+backtracking。可以迭代用栈实现，保持p数组，栈中额外压入层数和pathsum即可

<https://leetcode.com/problems/path-sum-ii/solution/>

## 114. Flatten Binary Tree to Linked List

同leetcode4-94题，Morris Traversal。中序遍历是先访问自己，右子树是在访问完左子树再访问的。所以把左子树放到右子树上，把原来的右子树放到原来的左子树的最右边节点即可。

答案罗里吧嗦一堆很烦

<https://leetcode.com/problems/flatten-binary-tree-to-linked-list/solution/>

## 115. Distinct Subsequences

动态规划。dp[i][j]表示s中i个字符和p中j个字符有几种可能。相等时，可删除可保留，不等时，只能删除。注意会用到j-1的值，所以j反向遍历，免得破坏之前的值。时间mn，空间可优化到n

<https://leetcode.com/problems/distinct-subsequences/solution/>

## 116. Populating Next Right Pointers in Each Node

动态规划。假设这一层的next已经弄好了，那么对于这一层每个节点，我们把左右子树相连，然后把右子树和下一个点的左子树相连即可。也就是沿着最左边向右一层层遍历即可

## 117. Populating Next Right Pointers in Each Node II

动态规划+three pointers。同leetcode4-116，但是多了很多特殊情况。实现上我们采用三个指针，一个指向最左边的节点，一个指向当前节点，一个指向下一层节点的前一个节点（初始值是none）。这样所有的查询和设置都可以在遍历当前节点的时候实现。时间复杂度n，空间复杂度1

<https://leetcode.com/problems/populating-next-right-pointers-in-each-node-ii/solution/>

## 118. Pascal's Triangle

Dynamic Programming。按题意操作即可

## 119. Pascal's Triangle II

<https://leetcode.com/problems/pascals-triangle-ii/>

方法1：动态规划。同118题，可以优化空间

方法2：数学。可以证明，第n行第k列（索引从0开始）就是组合数，因为组合数k具有性质。同时我们用组合数除以前一个数，所以可以直接依次求出结果

<https://leetcode.com/problems/pascals-triangle-ii/solution/>

## 120. Triangle

方法1：动态规划自上而下。每一行每一列的最小值都是上面相邻的两个最小的那个加上自己。空间复杂度n

方法2：动态规划自下而上。如果我们覆盖input，那么空间复杂度1

注意这个题不能用dfs，会超时，因为dfs其实访问每个点很多遍，每一条可能的路径都遍历了，但是其实中途我们可以不断砍枝的。这个题用动态规划之后很像118和119题

<https://leetcode.com/problems/triangle/solution/>

## 121. Best Time to Buy and Sell Stock

见leetcode3-309-121

## 122. Best Time to Buy and Sell Stock II

见leetcode3-309-121

## 123. Best Time to Buy and Sell Stock III

见leetcode3-309-123

## 124. Binary Tree Maximum Path Sum

递归。对于每个点，我们都返回带有这个点的一条最大单向路径，然后设置整体的最大路径为全局变量，对于每个点来说，得到左右两颗子树返回的单向最大值，那么以这个点为根的最大值一共有以下几种情况，自己加左子树最大值，自己加右子树，左子树加自己加右子树，仅自己。以此修改全局最大值，而返回值则是去掉第三种情况。时间复杂度n，空间复杂度h

## 125. Valid Palindrome

two pointers。忽略case：都转为小写；判断alphanumeric：用库

## 128. Longest Consecutive Sequence

排序。时间复杂度nlogn，空间1

或者用set，每个数双向展开，看看有多长，每个数遍历一次，时间复杂度n。或者也可以单向展开，找起点，起点的标志是比它小1的数不在set中

<https://leetcode.com/problems/longest-consecutive-sequence/solution/>

## 129. Sum Root to Leaf Numbers

方法1：DFS + preorder traversal.递归有点像backtracking，每次修改当前的pathsum即可，迭代的话要把当前的pathsum一起压入栈

方法2：Morris Preorder Traversal。把左子树的最右节点指向根节点，这样如果我们发现一个点的左子树的最右节点指向根节点，那么就说明它访问过了，我们就把这个link去掉，然后访问右子树，注意这里去掉这个link可能会出现新的叶子节点，否则设置上这个link并访问左子树。实际上每个叶子节点第一次不会被访问，它们的右子树被设置为上一层的根，所以他们被用来回到上一层的根，而回到上一层的根之后，在遍历回来就会发现左子树访问过了，这时候才会break掉这个link，然后访问这个叶子节点

<https://leetcode.com/problems/sum-root-to-leaf-numbers/solution/>

## 130. Surrounded Regions

DFS or BFS.如果想不被包围那么必定和边界上的O连接，所以我们可以从边界上的O进行搜索，把相连的O找出来，这个O保留，其余的翻转

## 134. Gas Station

见leetcode1-134

实现上，遍历remain数组，如果区间和小于零，就重置区间和，然后起点设置为i+1。如果总total大于0，则最后的起点就是答案；否则无解

## 135. Candy

见leetcode1-135

实现上，用oldslope和newslope来表示上一次的坡度走向和这一次的坡度走向，每一个一上一下或者是相等，都要计算值。注意这里每一个山峰计算的时候不包含当前的数，把当前的数放到下一次计算。细节过多，建议看答案

<https://leetcode.com/problems/candy/solution/>

## 136. Single Number

见leetcode1-136

## 137. Single Number II

见leetcode1-137

<https://blog.csdn.net/yutianzuijin/article/details/50597413>

最简单的办法是用一个32位数组表示每个数出现了几次，甚至这个数组也可以省掉，直接一位位的计算即可。接下来运用电路知识，实行状态转换，可以是通用解。最后再去掉临时变量，类似于swap函数，流弊

标准答案方法1是用哈希表然后求和乘以三再减去原来的和，就是出现一次的数的两倍。位操作的答案很难理解，见评论

<https://leetcode.com/problems/single-number-ii/solution/>

<https://leetcode.com/problems/single-number-ii/solution/563404>

## 138. Copy List with Random Pointer

见leetcode1-138

交叉之后每一个random指向的点的copy就在下一个点，所以可以重置每个点的random。最后再把链表恢复过来

## 139. Word Break

动态规划。对于字符s第i位之前能break的条件是前面有一个j，使得j能被拆，而j-i又能被拆。把词典转成哈希表能更快的查询。注意时间复杂度是n^3，因为构造子字符串需要n的时间

<https://leetcode.com/problems/word-break/solution/>

## 140. Word Break II

可以看做动态规划记录解的位置即可。官方解答有点复杂

https://leetcode.com/problems/word-break-ii/solution/

## 141. Linked List Cycle

见leetcode1-141

## 142. Linked List Cycle II

见leetcode1-142

## 143. Reorder List

见leetcode1-143。其中三个指针翻转列表的方法是，三个指针分别指向前一个节点、当前节点和后一个节点，每次将当前节点指向前一个节点，然后将三个指针后移即可。Python实现只需要两个指针，因为Python可以同时对两个变量赋值

## 144. Binary Tree Preorder Traversal