

## Listings

1	Problem26. Remove Duplicates from Sorted Array . . . . .	5
2	Problem80. Remove Duplicates from Sorted Array II . . . . .	6
3	Problem33. Search in Rotated Sorted Array . . . . .	8
4	Problem81. Search in Rotated Sorted Array II . . . . .	9
5	Problem4. Median of Two Sorted Arrays . . . . .	10
6	Problem128. Longest Consecutive Sequence . . . . .	11
7	Problem1. TwoSum . . . . .	13
8	Problem15. 3Sum . . . . .	14
9	Problem16. 3Sum Closest . . . . .	15
10	Problem18. 4Sum . . . . .	17
11	Problem27. Remove Element . . . . .	18
12	Problem31. Next Permutation . . . . .	20
13	Problem60. Permutation Sequence . . . . .	21
14	Problem42. Trapping Rain Water . . . . .	22
15	Problem48. Rotate Image . . . . .	24
16	Problem66. Plus One . . . . .	25
17	Problem70. Climbing Stairs . . . . .	25
18	Problem89. Gray Code . . . . .	26
19	Problem73. Set Matrix Zeroes . . . . .	27
20	Problem134. Gas Station . . . . .	28
21	Problem135. Candy . . . . .	29
22	Problem136. Single Number . . . . .	30
23	Problem137. Single Number II . . . . .	32
24	Problem126. Word Ladder II . . . . .	33
25	Problem78. Subsets . . . . .	35
26	Problem90. Subsets II . . . . .	35
27	Problem54. Spiral Matrix . . . . .	36
28	Problem59. Spiral Matrix II . . . . .	37
29	Problem238. Product of Array Except Self . . . . .	38
30	Problem268. Missing Number . . . . .	40
31	Problem209. Minimum Size Subarray Sum . . . . .	41
32	Problem56. Merge Intervals . . . . .	42
33	Problem229. Majority Element II . . . . .	43
34	Problem55. Jump Game . . . . .	44
35	Problem45. Jump Game II . . . . .	46
36	Problem122. Best Time to Buy and Sell Stock II . . . . .	46
37	Problem216. Combination Sum III . . . . .	47
38	Problem11. Container With Most Water . . . . .	48
39	Problem153. Find Minimum in Rotated Sorted Array . . . . .	49
40	Problem154. Find Minimum in Rotated Sorted Array II . . . . .	50
41	Problem57. Insert Interval . . . . .	51
42	Problem287. Find the Duplicate Number . . . . .	52
43	Problem162. Find Peak Element . . . . .	53
44	Problem328. Odd Even Linked List . . . . .	55
45	Problem2. Add Two Numbers . . . . .	56
46	Problem92. Reverse Linked List II . . . . .	58
47	Problem86. Partition List . . . . .	59
48	Problem83. Remove Duplicates from Sorted List . . . . .	60

49	Problem82. Remove Duplicates from Sorted List II . . . . .	61
50	Problem61. Rotate List . . . . .	62
51	Problem19. Remove $N^{th}$ Node From End of List . . . . .	63
52	Problem24. Swap Nodes in Pairs . . . . .	64
53	Problem25. Reverse Nodes in k-Group . . . . .	65
54	Problem138. Copy List with Random Pointer . . . . .	67
55	Problem141. Linked List Cycle . . . . .	68
56	Problem142. Linked List Cycle II . . . . .	69
57	Problem143. Reorder List . . . . .	70
58	Problem65. Valid Number . . . . .	72
59	Problem125. Valid Palindrome . . . . .	74
60	Problem28. Implement strStr() . . . . .	74
61	Problem8. String to Integer (atoi) . . . . .	77
62	Problem67. Add Binary . . . . .	78
63	Problem5. Longest Palindromic Substring . . . . .	80
64	Problem13. Roman to Integer . . . . .	83
65	Problem12. Integer to Roman . . . . .	84
66	Problem44. Wildcard Matching . . . . .	86
67	Problem20. Valid Parentheses . . . . .	88
68	Problem32. Longest Valid Parentheses . . . . .	89
69	Problem84. Largest Rectangle in Histogram . . . . .	91
70	Problem150. Evaluate Reverse Polish Notation . . . . .	92
71	Problem105. Construct Binary Tree from Preorder and Inorder Traversal . . . . .	93
72	Problem106. Construct Binary Tree from Inorder and Postorder Traversal . . . . .	94
73	Problem144. Binary Tree Preorder Traversal . . . . .	94
74	Problem94. Binary Tree Inorder Traversal . . . . .	96
75	Problem145. Binary Tree Postorder Traversal . . . . .	97
76	Problem102. Binary Tree Level Order Traversal . . . . .	98
77	Problem107. Binary Tree Level Order Traversal II . . . . .	99
78	Problem103. Binary Tree Zigzag Level Order Traversal . . . . .	99
79	Problem99. Recover Binary Search Tree . . . . .	101
80	Problem100. Same Tree . . . . .	101
81	Problem101. Symmetric Tree . . . . .	102
82	Problem110. Balanced Binary Tree . . . . .	104
83	Problem114. Flatten Binary Tree to Linked List . . . . .	105
84	Problem116. Populating Next Right Pointers in Each Node . . . . .	106
85	Problem117. Populating Next Right Pointers in Each Node II . . . . .	108
86	Problem96. Unique Binary Search Trees . . . . .	109
87	Problem95. Unique Binary Search Trees II . . . . .	110
88	Problem98. Validate Binary Search Tree . . . . .	112
89	Problem108. Convert Sorted Array to Binary Search Tree . . . . .	113
90	Problem109. Convert Sorted List to Binary Search Tree . . . . .	113
91	Problem111. Minimum Depth of Binary Tree . . . . .	115
92	Problem104. Maximum Depth of Binary Tree . . . . .	115
93	Problem112. Path Sum . . . . .	116
94	Problem113. Path Sum II . . . . .	117
95	Problem124. Binary Tree Maximum Path Sum . . . . .	117
96	Problem129. Sum Root to Leaf Numbers . . . . .	118

97	Problem88. Merge Sorted Array . . . . .	119
98	Problem21. Merge Two Sorted Lists . . . . .	120
99	Problem23. Merge k Sorted Lists . . . . .	121
100	Problem147. Insertion Sort List . . . . .	122
101	Problem148. Sort List . . . . .	123
102	Problem41. First Missing Positive . . . . .	124
103	Problem75. Sort Colors . . . . .	125
104	Problem34. Search for a Range . . . . .	125
105	Problem35. Search Insert Position . . . . .	126
106	Problem74. Search a 2D Matrix . . . . .	127
107	Problem240. Search a 2D Matrix II . . . . .	128
108	Problem62. Unique Paths . . . . .	130
109	Problem63. Unique Paths II . . . . .	131
110	Problem51. N-Queens . . . . .	133
111	Problem52. N-Queens II . . . . .	133
112	Problem93. Restore IP Addresses . . . . .	134
113	Problem39. Combination Sum . . . . .	135
114	Problem40. Combination Sum II . . . . .	136
115	Problem22. Generate Parentheses . . . . .	137
116	Problem37. Sudoku Solver . . . . .	138
117	Problem79. Word Search . . . . .	141
118	Problem120. Triangle . . . . .	142
119	Problem53. Maximum Subarray . . . . .	144
120	Problem131. Palindrome Partitioning . . . . .	144
121	Problem132. Palindrome Partitioning II . . . . .	146
122	Problem85. Maximal Rectangle . . . . .	147
123	Problem97. Interleaving String . . . . .	148
124	Problem87. Scramble String . . . . .	149
125	Problem64. Minimum Path Sum . . . . .	150
126	Problem72. Edit Distance . . . . .	151
127	Problem91. Decode Ways . . . . .	152
128	Problem115. Distinct Subsequences . . . . .	153
129	Problem139. Word Break . . . . .	154
130	Problem140. Word Break II . . . . .	156
131	Problem44. Wildcard Matching . . . . .	158
132	Problem264. Ugly Number II . . . . .	160
133	Problem303. Range Sum Query - Immutable . . . . .	161
134	Problem304. Range Sum Query 2D - Immutable . . . . .	162
135	Problem279. Perfect Squares . . . . .	164
136	Problem221. Maximal Square . . . . .	165
137	Problem152. Maximum Product Subarray . . . . .	165
138	Problem300. Longest Increasing Subsequence . . . . .	167
139	Problem198. House Robber . . . . .	168
140	Problem213. House Robber II . . . . .	170
141	Problem321. Create Maximum Number . . . . .	171
142	Problem322. Coin Change . . . . .	173
143	Problem312. Burst Balloons . . . . .	174
144	Problem121. Best Time to Buy and Sell Stock . . . . .	175
145	Problem309. Best Time to Buy and Sell Stock with Cooldown . . . . .	177
146	Problem123. Best Time to Buy and Sell Stock III . . . . .	179

147 Problem188. Best Time to Buy and Sell Stock IV . . . . . 179

# Solutions to Leetcode Problems in Python

XIN LIU

May 4, 2016

## 1 Array and List

### 1.1 Array

#### 1.1.1 leetcode 26. Remove Duplicates from Sorted Array

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.

解题思路：根据题目要求，不能使用额外的存储空间，所以一定是原地去重，那么一个比较直接的想法就是用不同的值覆盖重复的值。因此，我们用一个指针`j`指向目标数组`A`当前待判断是否是重复值的元素，用另外一个指针`i`指向已经建立好的、没有重复值的序列（数组`A`的一个子序列）的末尾。如果 $A[i] \neq A[j]$ 说明`j`当前指向的元素与新建立的序列中的元素不同，可以添加到新序列中。因此，移动`i`到下一个位置，并将`j`指向的值复制到`i`的位置。如果 $A[i] == A[j]$ ，说明当前`j`指向的元素与新序列中的最后一个元素是重复的，不应该加到新序列中，因此移动`j`到下一个位置，继续后序的筛选。`j`遍历完整个数组后就可以完成去重的任务，并且不需要额外的存储空间，时间复杂度为 $O(n)$ 。示例见下图。这道题另外一个需要注意的地方，也是后续所有题目解题是需要注意的事项是增加边界值的验证。由于leetcode提供的代码验证，已经提供了一些测试用例，但是最好在提交代码之前，自己设计一些测试用例验证自己设计的算法和代码实现，这也是面试题目要考查应试者的一部分内容。

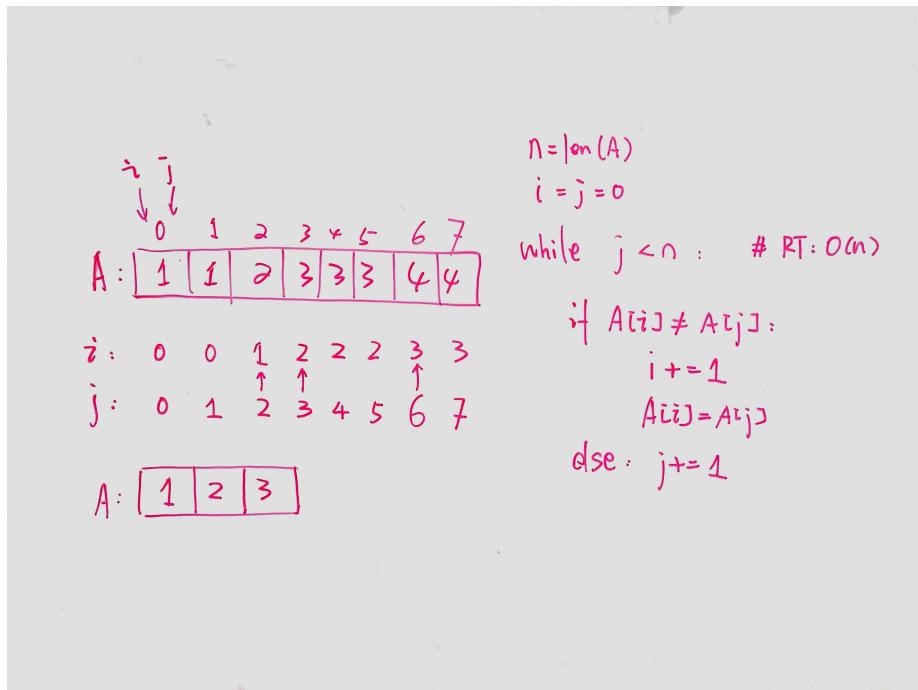
```
1 class Solution(object):
2     def removeDuplicates(self, nums):      # RT: O(n)
3         """
4             :type nums: List[int]
5             :rtype: int
6             """
7             if nums==None or len(nums)==0:
8                 return 0
9             slow=0
10            for fast in range(len(nums)):
```

```

12     if  nums[ fast]==nums[ slow ]:
13         continue
14     else :
15         slow+=1
16         nums[ slow]=nums[ fast ]
17
18     return  slow+1

```

Listing 1: Problem26. Remove Duplicates from Sorted Array



### 1.1.2 leetcode 80. Remove Duplicates from Sorted Array II

Follow up for "Remove Duplicates": What if duplicates are allowed at most twice?

For example, Given sorted array  $\text{nums} = [1, 1, 1, 2, 2, 3]$ , Your function should return length = 5, with the first five elements of  $\text{nums}$  being 1, 1, 2, 2 and 3. It doesn't matter what you leave beyond the new length.

解题思路：这道题可以直接泛化为求解一个允许存在 $k$ 个重复值的排序数组。当 $k = 2$ 时就是本题的情况。算法的设计与leetcode 26. Remove Duplicates from Sorted Array基本一致，差别在于判断重复值的时候，需要增加偏移量 $k$ （代码第15行），因为允许有 $k$ 重复值。见下图示例。

```

1
2 # Generic Alg. for "at most t duplicates" problems
3 class Solution(object):
4     def removeDuplicates2(self, nums):

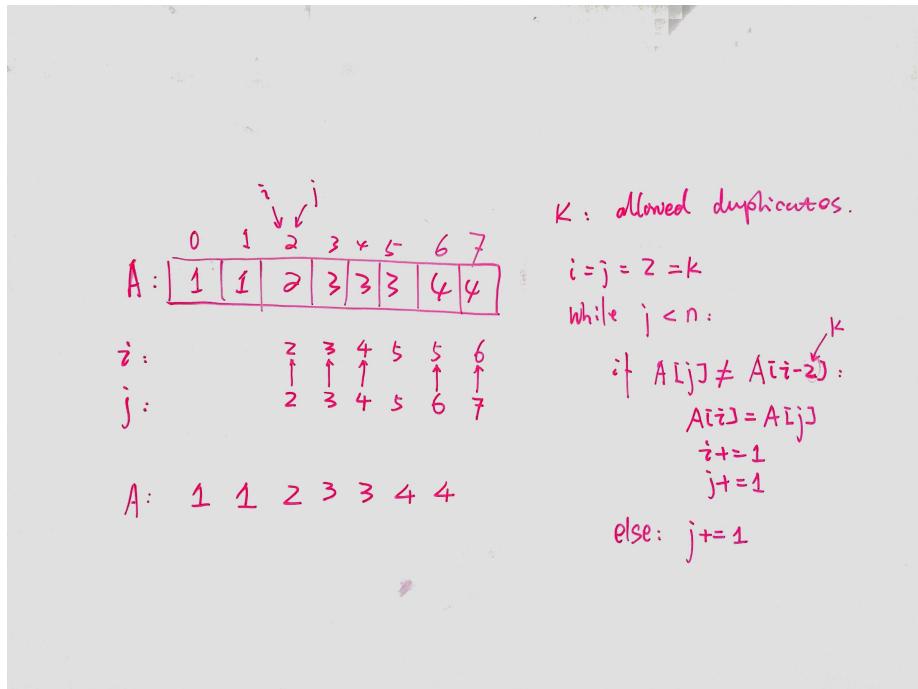
```

```

5     """
6     :type nums: List[int]
7     :rtype: int
8     """
9     if len(nums) <= 2: return len(nums)
10
11    # if k = n, it means duplicates are allowed at most n times
12
13    k = 2
14    index = k
15    for i in range(k, len(nums)):
16        if nums[i] != nums[index-k]:
17            nums[index] = nums[i]
18            index += 1
19
return index

```

Listing 2: Problem80. Remove Duplicates from Sorted Array II



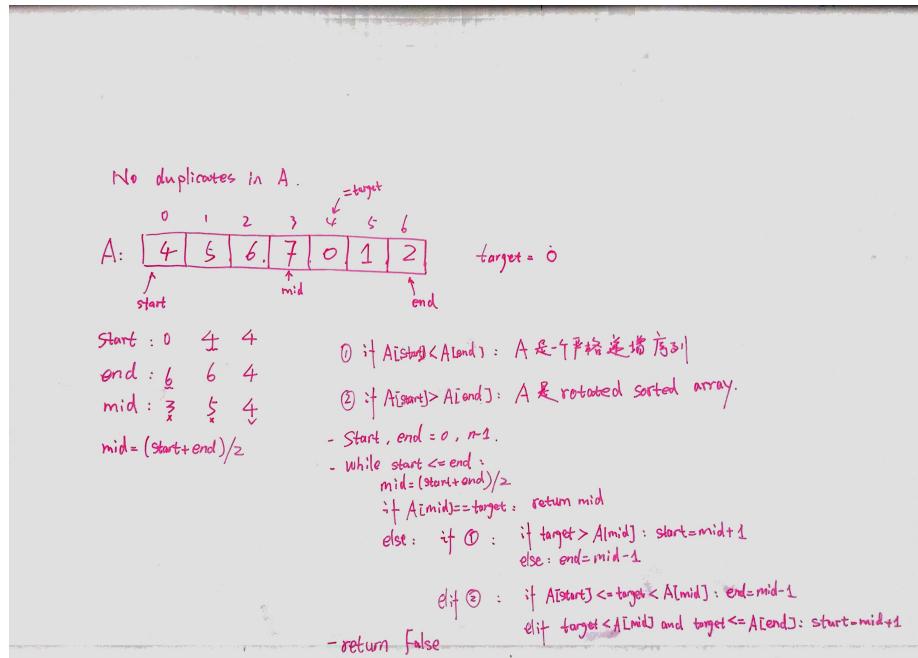
### 1.1.3 leetcode 33. Search in Rotated Sorted Array

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.

You may assume no duplicate exists in the array.

解题思路：对于在有序序列中查找指定值最有效的方法就是二叉搜索算法，时间复杂度为 $O(\log n)$ 。但是，这道题增加了一个难度，即旋转了有序序列，这样就破坏了原有的递增序列特性，从而需要改变二叉搜索排序中的判断条件。这

道题的考点应该在于对二叉搜索算法的理解。



```
1 class Solution(object):
2     def search(self, nums, target): # RT: O(log(n)), space: O(1)
3         """
4             :type nums: List[int]
5             :type target: int
6             :rtype: int
7         """
8         start, end = 0, len(nums)-1
9         while start <= end:
10             mid = (start + end)/2
11             if nums[mid] == target: return mid
12             if nums[start]<=nums[mid]:
13                 if nums[start]<=target<=nums[mid]:
14                     end = mid
15                 else:
16                     start = mid+1
17             else:
18                 if nums[mid]<=target<=nums[end]:
19                     start = mid+1
20                 else:
21                     end = mid
22
23     return -1
```

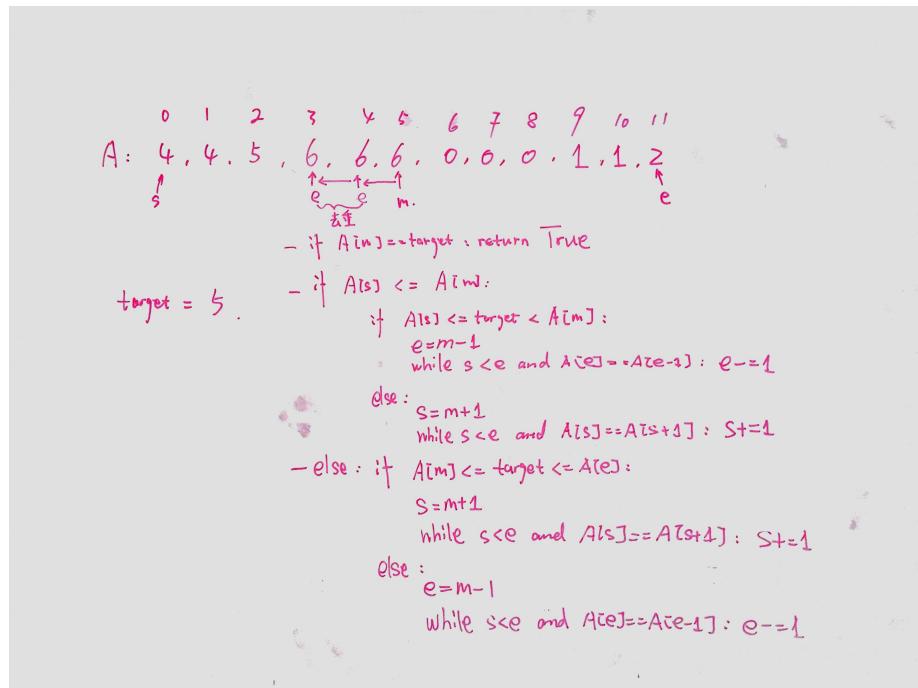
Listing 3: Problem33. Search in Rotated Sorted Array

#### 1.1.4 leetcode 81. Search in Rotated Sorted Array II

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.

解题思路：这道题是在leetcode 33 Search in Rotated Sorted Array基础上改编而来的。不同之处在于允许存在重复元素，这点改变不影响继续使用leetcode 33中设计的二叉搜索算法，但是仍然需要调整定位下一轮搜索起止位置的策略。因为重复元素的出现，我们需要增加一个内层循环去除在边界上的重复元素。在最坏情况下，即数组中的元素都是相同的，那么这个二叉的搜索算法的时间复杂度将增大到 $O(n)$ 。参见下图示例。这里给出的代码进一步简化了程序的写法，但是背后原理是相同的。



```
1 class Solution(object):
2     def search(self, nums, target): # RT: O(n), space: O(1)
3         """
4             :type nums: List[int]
5             :type target: int
6             :rtype: bool
7         """
8         first, last = 0, len(nums)-1
9         while first <= last:
10             mid = (first+last)/2
11             if nums[mid] == target: return True
12             if nums[first] < nums[mid]:
13                 if nums[first]<=target<=nums[mid]:
14                     last = mid
15                 else:
16                     first = mid + 1
17             elif nums[first] > nums[mid]:
18                 if nums[mid]<=target<=nums[last]:
19                     first = mid + 1
20                 else:
```

```

22         last = mid
23     else:
24         first += 1 # deal with duplicates
25
26     return False

```

Listing 4: Problem81. Search in Rotated Sorted Array II

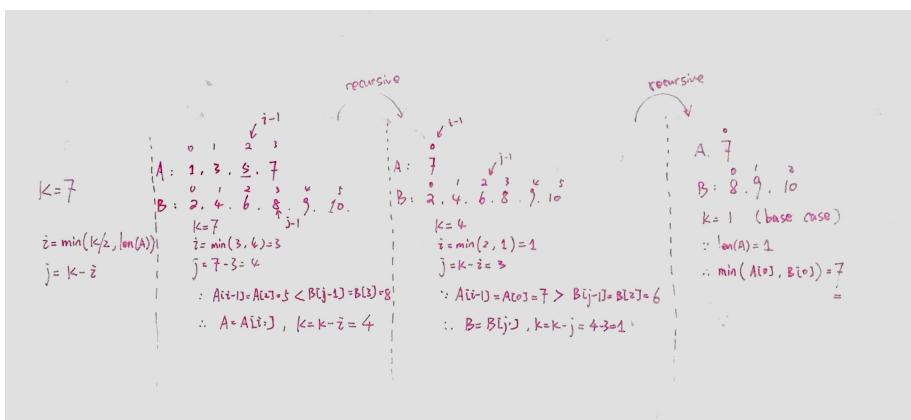
### 1.1.5 leetcode 4. Median of Two Sorted Arrays

There are two sorted arrays  $\text{nums1}$  and  $\text{nums2}$  of size  $m$  and  $n$  respectively. Find the median of the two sorted arrays. The overall run time complexity should be  $O(\log(m + n))$ .

解题思路：这道题要求返回两个已经排好序的数列的中位数。中位数的定义：如果数列有偶数个数值，那么中位数为中间两个数的平均值；如果数列有奇数个数，那么中位数为中间的那个数。比如 $\{1,2,3,4,5\}$ 的中位数为3。 $\{1,2,3,4,5,6\}$ 的中位数为 $(3 + 4)/2 = 3.5$ 。

这题最直接的思路就是将两个数列合并在一起，然后排序，然后找到中位数就行了。可是这样最快也要 $O((m+n)\log(m+n))$ 的时间复杂度，而题目要求数学 $O(\log(m+n))$ 的时间复杂度。那么，这道题其实考察的是二分查找。

参考下图中的示例，我们假设数组A总是比较短的那个数组。我们想找到两个有序数组中第 $k = 7$ 个小的元素呢？利用二分查找的思想，我们可以在数组A中取前 $k/2 = 3$ 个元素 $[1, 3, 5]$ ，在数组B中取前 $k - 3 = 4$ 个元素 $[2, 4, 6, 8]$ ，然后我们比较这两个数列中各自最后一个数值的大小，因为 $5 < 8$ ，那么就暗示了从A中取出的这个数列 $[1, 3, 5]$ 的任何一个元素一定都不是第7个数，那么我们就可以将这3个数从数组A的候选元素中去掉；与此同时，因为去掉了三个数，那么 $k$ 也要减去相应的数值，即我们现在只需要在剩余的两个有序序列中找到第 $k = 7 - 3 = 4$ 个小的数值。以此类推，直到 $k = 1$ 时，再次比较两个数组相应的值，小的一个即为第 $k$ 个小的元素。从上面的描述，我们可以发现这是一个递归的过程，而 $k = 1$ 就是递归定义中的base case。



```

1
2 class Solution(object):
3     # use binary search idea: each time removes k/2 elements at
        most

```

```

4   # RT: O(log(m+n)), m is the length of nums, n is the
5   # length of nums2
6   def findMedianSortedArrays(self, nums1, nums2):
7       """
8           :type nums1: List[int]
9           :type nums2: List[int]
10          :rtype: float
11      """
12      total = len(nums1) + len(nums2)
13      if total % 2 != 0: # total is odd
14          return self.find_kth(nums1, nums2, total//2+1)
15      else:
16          return (self.find_kth(nums1, nums2, total//2) + self.
17                  find_kth(nums1, nums2, total//2+1)) / 2.0
18
19      def find_kth(self, nums1, nums2, k):
20          # always assume the length of nums1 is less than that of
21          # nums2
22          if len(nums1) > len(nums2):
23              return self.find_kth(nums2, nums1, k)
24
25          if len(nums1) == 0:
26              return nums2[k-1]
27          if k == 1:
28              return min(nums1[0], nums2[0])
29
30          n1 = min(k//2, len(nums1))
31          n2 = k - n1
32          if nums1[n1-1] < nums2[n2-1]:
33              return self.find_kth(nums1[n1:], nums2, k-n1)
34          elif nums1[n1-1] > nums2[n2-1]:
35              return self.find_kth(nums1, nums2[n2:], k-n2)
            else:
                return nums1[n1-1]

```

Listing 5: Problem4. Median of Two Sorted Arrays

### 1.1.6 leetcode 128. Longest Consecutive Sequence

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.

```

1  class Solution(object):
2      def longestConsecutive(self, nums):
3          """
4              :type nums: List[int]
5              :rtype: int
6          """
7          if len(nums) <= 1:
8              return len(nums)
9
10         # Because the time complexity is O(n), have to
11         # use an dictionary or hashmap.
12

```

```

13     # If the time complexity is O(nlog(n)),
14     # we can sort nums first.
15     dict = {}
16     for i in nums:
17         dict[i] = False
18
19     longest = 0
20     for i in nums:
21         if dict[i] == True:
22             continue
23         length = 1
24         dict[i] = True
25
26         # check the consecutivity of the right
27         # side of the current key
28         j = i+1
29         while dict.has_key(j):
30             dict[j] = True
31             length += 1
32             j += 1
33
34         # check the consecutivity of the left
35         # side of the current key
36         j = i-1
37         while dict.has_key(j):
38             dict[j] = True
39             length += 1
40             j -= 1
41
42         # check the longest length
43         longest = max(longest, length)
44         if longest == len(nums):
45             break;
46
47     return longest

```

Listing 6: Problem128. Longest Consecutive Sequence

### 1.1.7 leetcode 1. TwoSum

Given an array of integers, find two numbers such that they add up to a specific target number.

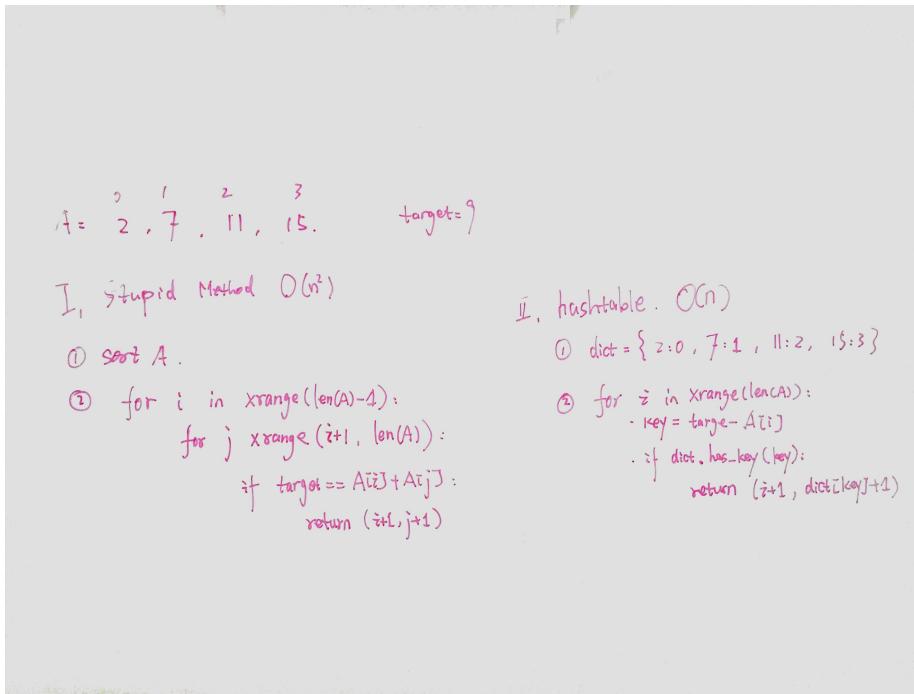
The function twoSum should return indices of the two numbers such that they add up to the target, where index1 must be less than index2. Please note that your returned answers (both index1 and index2) are not zero-based.

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

- Input: numbers={2, 7, 11, 15}, target=9
- Output: index1=1, index2=2

解题思路：这道题最直接的思路就是暴力遍历所有组合方式，时间复杂度就是 $O(n^2)$ 。但是，显然这不是考点。题目的重点应该是数据结构的选择，对算法设计的影响。暴力组合的方法存在的问题是每次选择一个元素后，就需要验证数组的其余元素是否能与这个元素加和后等于目标值，这是造成时间复杂度增高的原因，如果可以在常数时间内完成验证就可以有效降低算法的时间复杂

度。那么可以完成 $O(1)$ 查找的数据结构就是哈希表或者字典。两种方法的示例见下图。另外，需要注意的一点是题目中没有说数组是有序的，所以如果用暴力组合的方法，先进行排序会让代码更简洁些。



```

1 class Solution:
2     # @return a tuple, (index1, index2)
3     def twoSum1(self, num, target):
4         # use an dictionary to implement,
5         # but there are no duplicates in num.
6         # RT is O(n).
7         dict = {}
8         for i in range(len(num)):
9             dict[num[i]] = i
10
11        for j in num:
12            k = target - j
13            if k > 0 and dict.has_key(k) and dict[j] != dict[k]:
14                return (dict[j]+1, dict[k]+1)
15
16        return (-1,-1)
17
18    def twoSum2(self, num, target):
19        # brute force method: Use list to implement.
20        # RT: O(n^2).
21        if len(num) <= 1:
22            return (-1,-1)
23
24        list = []
25        for i in range(len(num)):
26            list.append((i+1, num[i]))
27
28        list.sort(key=lambda x:x[1])
29
      
```

```

30
31     for i in range(len(list)):
32         a = list[i][1]
33         b = target - a
34
35         if(i+1 < len(list)):
36             for j in range(i+1, len(list)):
37                 if b == list[j][1]:
38                     if(list[i][0] <= list[j][0]):
39                         return (list[i][0], list[j][0])
40                     else:
41                         return (list[j][0], list[i][0])
42
43     return (-1,-1)

```

Listing 7: Problem1. TwoSum

### 1.1.8 leetcode 15. 3Sum

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.

Note:

- Elements in a triplet (a,b,c) must be in non-descending order. (ie,  $a \leq b \leq c$ )
- The solution set must not contain duplicate triplets.

For example, given array  $S = \{-1, 0, 1, 2, -1, -4\}$ , A solution set is:  $(-1, 0, 1)$ ,  $(-1, -1, 2)$ .

解题思路：题目要求输出实际的组合值，所以最直接的想法就是DFS遍历决策树。为了达到结果中没有重复组合，需要排序，并且存储过程结果数据结构用集合set。但是，这样设计的算法在OJ的测试过程中超时。第二种算法思想和DFS类似，但是更为通用，可以计算任意一种 $k$ 个数字累加和等于指定值的情况。有什么更好的算法么？

```

1  class Solution(object):
2     def threeSum_dfs(self, nums): # DFS, but TLE
3         """
4             :type nums: List[int]
5             :rtype: List[List[int]]
6             """
7
8         def dfs(depth, target, nums, valuelist):
9             if depth == 0:
10                 if target == 0: res.add(tuple(valuelist))
11                 return
12             if target < nums[0] or target > nums[len(nums)-1]: return
13             for i in xrange(len(nums)-depth+1):
14                 dfs(depth-1, target - nums[i], nums[i+1:], valuelist + [nums[i]])
15             if nums == []: return []
16             nums.sort()
17             res = set()
18             dfs(3, 0, nums, [])
19             return [list(t) for t in res]
20

```

```

21     def threeSum2(self, nums):
22         """
23             :type nums: List[int]
24             :rtype: List[List[int]]
25         """
26         nums.sort() # O(nlog(n))
27
28     # This is a generic k-sum algorithm
29     def ksum(nums, k, target):
30         # if use result = [], it causes TLE and the
31         # final result has duplicates.
32         result = set()
33         i=0
34         if k==2:
35             j = len(nums)-1
36             while i<j:
37                 if nums[i]+nums[j]==target:
38                     result.add((nums[i],nums[j]))
39                     i+=1
40                 elif nums[i]+nums[j]>target:
41                     j-=1
42                 else:
43                     i+=1
44             else: # case: k>2
45                 for i in xrange(len(nums)-k+1):
46                     newtarget = target - nums[i]
47                     subresult = ksum(nums[i+1:], k-1, newtarget)
48                     if subresult:
49                         result = result | set((nums[i],)+nr for nr
50                         in subresult)
51     return result
52     return [list(t) for t in ksum(nums, 3, 0)]

```

Listing 8: Problem15. 3Sum

### 1.1.9 leetcode 16. 3Sum Closest

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.

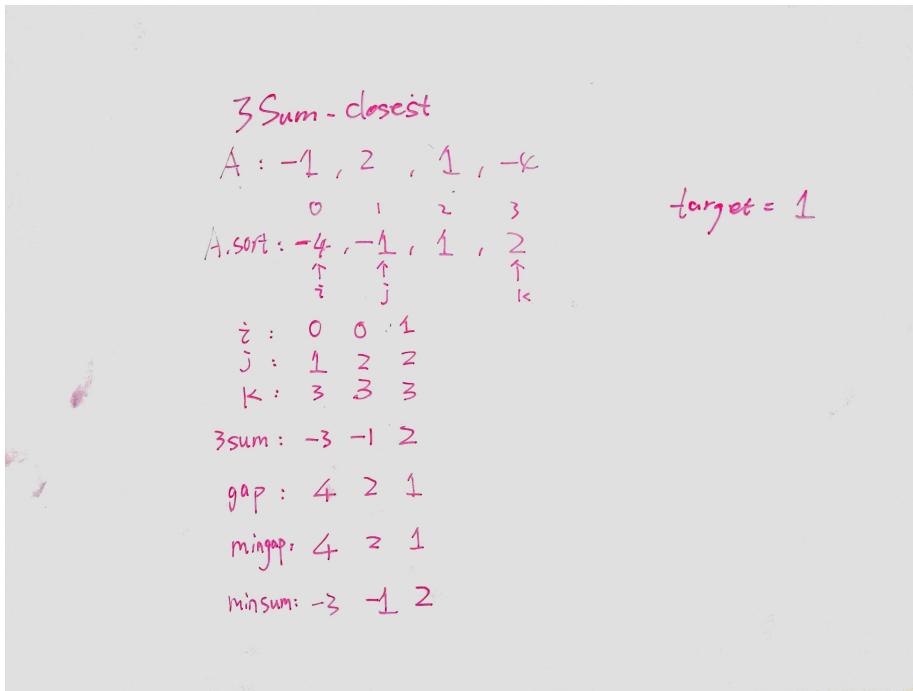
For example, given array  $S = \{-1, 2, 1, -4\}$ , and  $target = 1$ . The sum that is closest to the target is 2. ( $-1 + 2 + 1 = 2$ ).

解题思路：这道题是求所有三个数的组合中，那个组合的累加和在数轴上最接近指定的目标值。构成三个数的组合的通常解法就是先在数组中任选一个数*i*，那么数组中剩下的数就构成了组合中另外两个数*j, k*的候选集合。如果这个数组是有序的，那么*j*和*k*的初始位置就可以分别选在候选集合的两段：如果三者的和小于目标值，那么就可以根据数组是有序递增序列的性质，保持*k*不变，将*j*向高位移动；如果三者之和大于目标值，就将*k*向低位移动；如果三者之和等于目标值，那就直接返回结果，程序结束。参考下图中的示例。

```

1 #
2 # Given an array S of n integers, find three integers in S
3 # such that the sum is closest to a given number, target.
4 # Return the sum of the three integers. You may assume that
5 # each input would have exactly one solution.
6 #

```



```

7 class Solution(object):
8     def threeSumClosest(self, nums, target):      # O(n^2)
9         """
10        :type nums: List[int]
11        :type target: int
12        :rtype: int
13        """
14        nums.sort()      # O(nlog(n))
15        min_gap = sys.maxint
16        result = 0
17
18        for i in range(len(nums)-3+1):    # O(n^2)
19            j=i+1
20            k=len(nums)-1
21            while j<k:
22                sum = nums[i]+nums[j]+nums[k]
23                gap = abs(target-sum)
24                if gap < min_gap:
25                    min_gap = gap
26                    result = sum
27                    if min_gap==0:
28                        return target
29                if sum < target:
30                    j+=1
31                else:
32                    k-=1
33        return result

```

Listing 9: Problem16. 3Sum Closest

### 1.1.10 leetcode 18. 4Sum

Given an array S of n integers, are there elements a, b, c, and d in S such that  $a + b + c + d = \text{target}$ ? 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,  $a \leq b \leq c \leq d$ )
- The solution set must not contain duplicate quadruplets.

For example, given array  $S = \{1, 0, -1, 0, -2, 2\}$ , and target = 0. A solution set is:  $(-1, 0, 0, 1)$   $(-2, -1, 1, 2)$   $(-2, 0, 0, 2)$

解题思路：这道题可以设计三种算法解题：DFS遍历决策树，ksum通用算法，以及利用hashtable的算法。前两种无需开辟额外空间就可以解决问题，但是时间复杂度比较高，OJ上通不过。所以，也可以反应出这道题的考查点应该是第三种算法设计思路，即利用“空间换时间”的思路，选择合适的数据结构，降低算法时间复杂度。使用Python的话，我们可以通过字典来dict来模拟hashtable：字典的key值为数组中每两个元素的和，每个key对应的value为这两个元素的下标组成的元组，元组不一定是唯一的。如对于num=[1,2,3,2]来说，dict={3:[(0,1),(0,3)], 4:[(0,2),(1,3)], 5:[(1,2),(2,3)]}。这样就可以检查(target-key)的差值是不是dict的key值中，如果差值是dict中的一个key并且下标符合要求，那么就找到了这样的一组解。由于需要去重，这里选用set()类型的数据结构，即无序无重复元素集。最后将每个找出来的解(set()类型)转换成list类型输出即可。下面的代码给出了第二、三种算法的实现。

```
1 #
2 # Given an array S of n integers, are there elements a, b, c,
3 # and d in S such that a + b + c + d = target? Find all unique
4 # quadruplets in the array which gives the sum of target.
5 #
6 class Solution(object):
7     def fourSum(self, nums, target):      # time complexity is O(n
8         ^2)
9         nums.sort()
10        n = len(nums)
11        res = set()
12        dict = {}
13        # establish the dict
14        for i in xrange(n-1):
15            for j in xrange(i+1, n):
16                sum = nums[i]+nums[j]
17                # there could be duplicates
18                if dict.has_key(sum):
19                    dict[sum].append((i,j))
20                else:
21                    dict[sum] = [(i,j)]
22        for i in xrange(n-4+1):
23            for j in xrange(i+1, n-4+2):
24                newsum = target - nums[i] - nums[j]
25                if dict.has_key(newsum):
26                    for item in dict[newsum]:
27                        # Elements in a quadruplet (a,b,c,d)
28                        # must be in non-descending order
29                        if j < item[0]:
                                res.add((nums[i],nums[j],nums[item[0]]),
```

```

    nums[item[1]]))
30
31     return [list(t) for t in res]
32
33 # Generic algorithm, which has O(n^3) running time.
34 def fourSum2(self, nums, target): # Time Limit Exceeded
35     """
36     :type nums: List[int]
37     :type target: int
38     :rtype: List[List[int]]
39     """
40     def ksum(nums, k, target):
41         result = set()
42         i = 0
43
44         if k==2:
45             j=len(nums)-1
46             while i<j:
47                 if nums[i]+nums[j] == target:
48                     result.add((nums[i],nums[j]))
49                 elif nums[i]+nums[j] > target:
50                     j-=1
51                 else:
52                     i+=1
53             else: # case: k>2
54                 while i < len(nums)-k+1:
55                     newtarget = target - nums[i]
56                     subresult = ksum(nums[i+1:], k-1, newtarget)
57                     if subresult:
58                         result = result | set((nums[i],)+nr for nr
59                         in subresult)
60                         i+=1
61         return result
62
63     nums.sort()
64     return [list(t) for t in ksum(nums, 4, target)]
```

Listing 10: Problem18. 4Sum

### 1.1.11 leetcode 27. Remove Element

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.

解题思路：这道题比较简单，这里给出了两种思路，参考下面示例图。第二种方法在考虑边界情况的时候比较简单。这里给出第二种思路的代码实现。

```

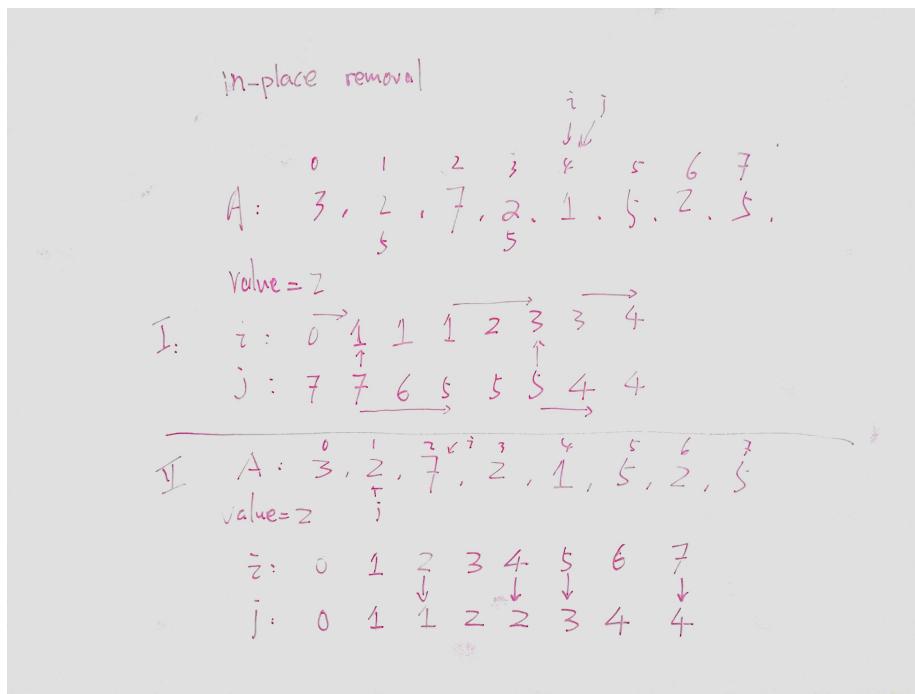
1 class Solution(object):
2     def removeElement(self, nums, val):      # O(n)
3         """
4             :type nums: List[int]
5             :type val: int
6             :rtype: int
7             """
8
9             if len(nums) < 0: return 0
10            index = 0
11            for n in nums:
```

```

12     if n!=val:
13         nums[index] = n
14         index += 1
15     return index

```

Listing 11: Problem27. Remove Element



### 1.1.12 leetcode 31. Next Permutation

Implement next permutation, which rearranges numbers into the lexicographically next greater permutation of numbers.

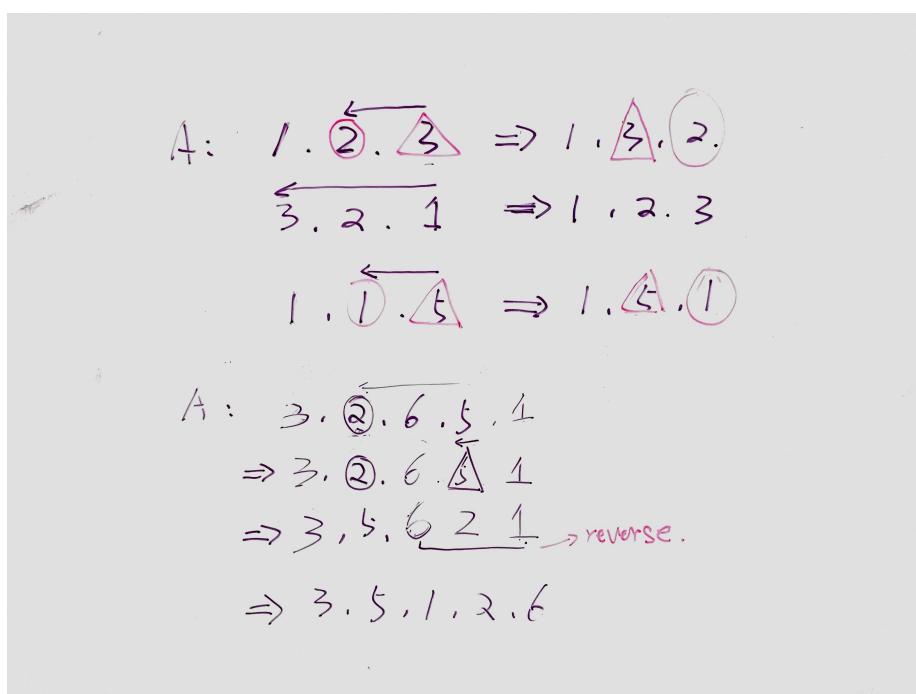
If such arrangement is not possible, it must rearrange it as the lowest possible order (ie, sorted in ascending order). The replacement must be in-place, do not allocate extra memory.

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

解题思路：这道题给出的例子有一定的误导性，所以分析之后容易设计出有缺陷的算法。可以更有代表性的例子进行分析：[3, 2, 6, 5, 1] → [3, 5, 1, 2, 6]。分析过程如下图所示：目标是要找到[3, 2, 6, 5, 1]在递增排列中的下一个排列。我们

知道当目标序列从左到右各数字是按照递减序列给出的，那么这个目标序列就已经是由这些数字能够构成的所有排列中最大的一个，这也暗示我们可以通过检测这种自左向右递减（或者，自右向左递增）的性质来设计算法。因为需要找到目标序列的下一个而且更大的排列，所以检测自右向左递增性质比较可行。具体步骤：首先，自右向左检测目标序列，如果发现某个位置*i*处的数字导致递增的性质结束，那么它就是我们需要替换的数字，具体说就是用一个更大的数字替换它。那么，我们哪一个数字替换它呢？这数字应该从它在目标序列中所在位置的右侧，也就是我们遍历过的数字中寻找。而寻找的策略就是自右向左再次遍历这个序列，如果在位置*j*(*j > i*)处找到第一个比*i*处大的数字，那么就交换*i*和*j*两个位置的值。比如在我们的例子中，*i* = 1，而*j* = 3，交换两值后序列变成[3, 5, 6, 2, 1]；因为32651是所有32xxx形数值中最大的一个，所以我们需要找到交换2和5之后，所有35xxx形数值中最小的一个。再进一步分析，交换2和5之后，*i*之后的序列由651变为621，但是序列的递增性并未改变，这也说明变换后的序列621仍然是由6, 2, 1三个数构成的所有序列中最大的一个，而我们需要的恰恰是最小的一个，那么就直接翻转一下这三个数字的次序就可以得到我们需要的最小值，即126。这样，我们最后得到的结果就是35126，即使题目要求的结果。



```

1 class Solution(object):
2     def nextPermutation(self, nums):      # O(n)
3         """
4             :type nums: List[int]
5             :rtype: void Do not return anything, modify nums in-place
6             instead.
7             """
8         n = len(nums)
9         idxPartitionNumber = -1

```

```

10     # From right to left , find the first digit
11     # (PartitionNumber) which violate the increase trend
12     for x in xrange(n-1, -1, -1):
13         if nums[x-1] < nums[x]:
14             idxPartitionNumber = x-1
15             break
16     if idxPartitionNumber < 0:
17         # From right to left , when all digits are in
18         # incremental order, reverse all digits.
19         nums.reverse()
20     else:
21         # From right to left , find the first digit
22         # (ChangeNumber), which is larger than PartitionNumber;
23         # then , swap ChangeNumber and PartitionNumber
24         for x in xrange(n-1, -1, -1):
25             if nums[x] > nums[idxPartitionNumber]:
26                 nums[x], nums[idxPartitionNumber] = nums[
27                     idxPartitionNumber], nums[x]
28             break
29     # Reverse all digits on the right side of partition
index
        nums[idxPartitionNumber+1:] = nums[idxPartitionNumber
+1:][::-1]

```

Listing 12: Problem31. Next Permutation

### 1.1.13 leetcode 60. Permutation Sequence

The set  $[1,2,3,\dots,n]$  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  $k$ th permutation sequence.

解题思路: 假设  $n = 6$ ,  $k = 400$ :

- 先计算第一位: 第一位为6, 那么它最少也是第 $5! \times 5 + 1$ 个排列, 这是因为第一位为1/2/3/4/5时, 都有 $5!$ 个排列, 因此第一位为6时, 至少是第 $5! \times 5 + 1$ 个排列 (这个排列为612345)。 $5! \times 5 + 1 = 601 > k$ , 所以第一位不可能是6。一个一个地枚举, 直到第一位为4时才行, 这时, 4xxxxx至少为第 $5! \times 3 + 1 = 361$ 个排列。
- 然后计算第二位, 与计算第一位时的区别在于, 46xxxx至少为第 $4! \times 4 + 1 = 97$ 个排列, 这是因为比6小的只有5/3/2/1了。最后可以计算出第二位为2。
- 以此类推, 最终得出第400个排列为425361。

```

1 class Solution(object):
2     # Iterative implementation , RT is O(n)
3     def getPermutation(self, n, k):
4         """
5             :type n: int
6             :type k: int
7             :rtype: str
8             """
9             res = ''
10            # compute the factorial of (n-1)

```

```

12     factorial = 1
13     for i in xrange(1,n): factorial *= i
14     # the permutations is based on 0
15     k = k - 1
16     nums = [x+1 for x in xrange(n)]
17     for x in xrange(n-1, -1, -1):
18         i = k/factorial
19         res += str(nums[i])
20         nums.remove(nums[i])
21         if x != 0:
22             k = k % factorial
23             factorial = factorial / x
24     return res

```

Listing 13: Problem60. Permutation Sequence

### 1.1.14 leetcode 42. Trapping Rain Water

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.



解题思路：开辟两个数组leftmost和rightmost，leftmost[i]表示在height[i]之前（包括 $i$ ）出现过的最高的bar值，而rightmost[i]表示在height[i]之后（包括 $j$ ）出现过的最高的bar值。假设在某个 $i$ 位置处，当leftmost[i]和rightmost[i]都大于height[i]时，那么leftmost[i], height[i], 和rightmost[i]三者构成一个可以储水的“凹”；而这个“凹”的储水量是以height[i]为“底”，leftmost[i]和rightmost[i]两者中的较小值为有效高度计算得到的，即water = min(leftmost[i], rightmost[i]) - height[i]。下面代码的第一个方法实现了上面的设计思路。代码中的第二个方法是对第一个方法的改进，使用了一个同名的变量代替了方法一中的rightmost数组，降低空间复杂度的同时，可以进一步改进效率。下面的示例图给出了方法一的计算过程和及部分图例。

```

1 class Solution(object):
2     def trap1(self, height):
3         n = len(height)
4         leftmost = []
5         leftmax = 0
6         for x in xrange(n):
7             if height[x] > leftmax:
8                 leftmax = height[x]
9                 leftmost.append(leftmax)
10

```

	0	1	2	3	4	5	6	7	8	9	10	11
height	0	1	0	2	1	0	1	3	2	1	2	1
leftmost	0	1	1	2	2	2	2	3	3	3	3	3
rightmost	3	3	3	3	3	3	3	3	2	2	2	1
water	0	0	1	0	1	2	1	0	0	1	0	0

$$\therefore \text{Sum(water)} = 6$$

```

11     rightmost = []
12     rightmax = 0
13     for x in xrange(n-1, -1, -1):
14         if height[x] > rightmax:
15             rightmax = height[x]
16             rightmost.insert(0, rightmax)
17     water = 0
18     for x in xrange(n):
19         if height[x] < min(leftmost[x], rightmost[x]):
20             water += min(leftmost[x], rightmost[x]) - height[x]
21     return water
22
23 def trap2(self, height): # O(n) running time, O(n) addition
24     """
25         :type height: List[int]
26         :rtype: int
27     """
28     n = len(height)
29     # from left to right, compute leftmost[i],
30     # which means the most height on the left
31     # side of the i-th position
32     leftmax = 0
33     leftmost = []
34     for x in xrange(n):
35         if height[x] > leftmax:
36             leftmax = height[x]
37             leftmost.append(leftmax)
38
39     water = 0
40     # rightmost indicates the most height on the
41     # right side of the i-th position from right
42     # to left
43     rightmost = 0
44     for x in xrange(n-1, -1, -1):
45         if height[x] > rightmost:
46             rightmost = height[x]
47         elif min(leftmost[x], rightmost) > height[x]:
48             water += min(leftmost[x], rightmost) - height[x]
49

```

```
50     return water
```

Listing 14: Problem42. Trapping Rain Water

### 1.1.15 leetcode 48. Rotate Image

You are given an  $n \times n$  2D matrix representing an image. Rotate the image by 90 degrees (clockwise).

Follow up: Could you do this in-place?

解题思路：该任务可以通过两个步骤完成。第一步将原矩阵按照行排列分成上下两个部分，依次交换第一行和最后一行，第二行和倒数第二行，以此类推。第二步按照主对角线依次交换两个对称位置上的元素。第二步结束后即可得到最终结果。

进一步扩展这个题目就是求解逆时针旋转90度，那么首先同样完成上面描述的第一步，而第二步是按照次对角线交换对称位置上的一对儿元素。

```
1 class Solution(object):
2     def rotate(self, matrix):      # O(n^2)
3         """
4             :type matrix: List[List[int]]
5             :rtype: void Do not return anything, modify matrix in-place
6             instead.
7             """
8         rows = len(matrix)
9         cols = len(matrix[0])
10
11        if rows<2 or cols<2:
12            return
13
14        # invert the matrix
15        for j in range(cols):
16            for i in range(rows/2):
17                matrix[i][j], matrix[rows-1-i][j] = matrix[rows-1-i]
18                [[j], matrix[i][j]
19
20        # flap the matrix along the main diagonal
21        # k indicates the starting column when the
22        # row changes each time
23        k = 0
24        for i in range(rows):
25            for j in range(k,cols):
26                if i!=j:
27                    matrix[i][j], matrix[j][i] = matrix[j][i],
matrix[i][j]
k+=1
```

Listing 15: Problem48. Rotate Image

### 1.1.16 leetcode 66. Plus One

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.

```
1 class Solution(object):
2     def plusOne(self, digits): # O(n)
3         """
4             :type digits: List[int]
5             :rtype: List[int]
6             """
7         n = len(digits)
8         carry = 1
9         for x in xrange(n-1,-1,-1):
10             val = digits[x] + carry
11             carry = val / 10
12             digits[x] = val % 10
13         if carry == 1:
14             digits = [1] + digits
15         return digits
```

Listing 16: Problem66. Plus One

### 1.1.17 leetcode 70. Climbing Stairs

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?

解题思路：这道题就是比较典型的递归算法，与Fibonacci Number的求解思路一致。这个递归的思路也有明显的用DP算法进行改进的特征，即存在重复计算。下面代码只给出两种DP算法的实现，两者的区别在于空间复杂度。

```
1 class Solution(object):
2     def climbStairs_dp1(self, n): # RT:O(n), Space: O(n)
3         dp = [0 for _ in xrange(n+1)]
4         dp[0] = 1
5         dp[1] = 1
6         for x in xrange(2, n+1):
7             dp[x] = dp[x-1]+dp[x-2]
8         return dp[n]
9
10    def climbStairs_dp2(self, n): # RT: O(n), Space: O(1)
11        """
12            :type n: int
13            :rtype: int
14            """
15        if n==1: return 1
16        if n==2: return 2
17
18        a,b,c = 1,2,0
19        for i in range(2, n):
20            c = a + b
21            a,b = b,c
22        return c
```

Listing 17: Problem70. Climbing Stairs

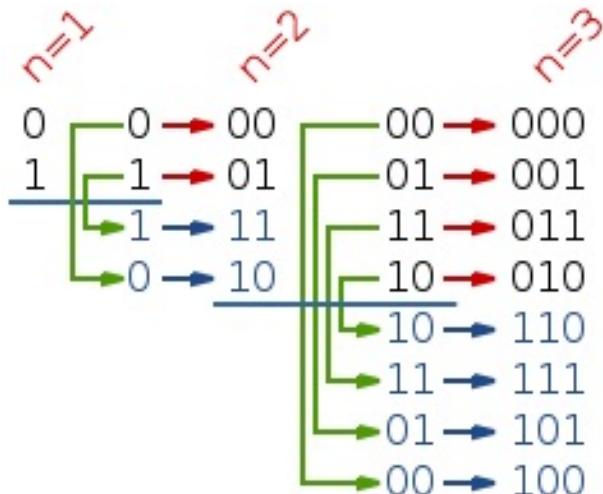
### 1.1.18 leetcode 89. Gray Code

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

解题思路：Gray Code是啥都不知道，这道题只能求助万能的wikipedia了。Wikipedia上给出的reflect-and-prefix方法很有趣，这里提供了来自wikipedia的示例图来说明这个方法的思想。



```
1 class Solution(object):
2     def grayCode(self, n): # RT: O(n)
3         res = []
4         size = 1 << n
5         for i in range(size):
6             res.append((i >> 1) ^ i)
7         return res
```

Listing 18: Problem89. Gray Code

### 1.1.19 leetcode 73. Set Matrix Zeroes

Given a  $m \times n$  matrix, if an element is 0, set its entire row and column to 0. Do it in place.

Follow up: Did you use extra space? A straight forward solution using  $O(mn)$  space is probably a bad idea. A simple improvement uses  $O(m + n)$  space, but still not the best solution. Could you devise a constant space solution?

解题思路：这道题实际就是设计DP算法。根据题目对空间复杂度的要求也可以看出这一点。这里给出一个示例图，图中使用了比题目要求更少的额外空间，即 $O(m)$ ，但是计算复杂度会有所增加。这里展示的这种额外空间的设计思路是和Leetcode51 N-Queens系列题目的做个类比，实际就是相同的想法：用一维数组表示每行的信息，而第*i*个cell中的数值*j*表示目标矩阵中第*i*行、第*j*列的某种对解题有益的某种或某类属性。

A  $m \times n = 5 \times 5$

0	1	2	3	4
0	5	9	1	4
1	3	0	8	1
2	4	4	7	0
3	0	2	0	8
4	6	0	1	2

$\Rightarrow$

0	1	2	3	4
0	0	0	0	2
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0

$T: [ [], [1], [1, 3], [3, 1, 5], [2, 4, 4, 7, 0, 3], [3, 0, 2, 0, 8, 4], [4, 6, 0, 1, 2, 5] ]$

for  $i$  in xrange( $m$ ):  
 if  $T[i] \neq []$ :  
      $A[i][j] = 0 \quad \#$  set the  $i^{th}$  row to zeros  
     for  $y$  in  $A[i]$ :  
         for  $x$  in xrange( $n$ ):  
              $A[x][y] = 0 \quad \left. \begin{array}{l} \# \text{set the } y^{th} \text{ column} \\ \text{to zeros} \end{array} \right\}$

```

1 class Solution(object):
2     def setZeroes(self, matrix):      # Time:O(m*n), Space:O(m+n)
3         """
4             :type matrix: List[List[int]]
5             :rtype: void Do not return anything, modify matrix in-place
6             instead.
7             """
8         m,n = len(matrix), len(matrix[0])
9         zerorows = [False for _ in xrange(m)]
10        zerocols = [False for _ in xrange(n)]
11
12        for x in xrange(m):
13            for y in xrange(n):
14                if matrix[x][y] == 0:
15                    zerorows[x], zerocols[y] = True, True
16
17        for x in xrange(m):
18            for y in xrange(n):
19                if zerorows[x] or zerocols[y]:
20                    matrix[x][y] = 0

```

Listing 19: Problem73. Set Matrix Zeroes

### 1.1.20 leetcode 134. Gas Station

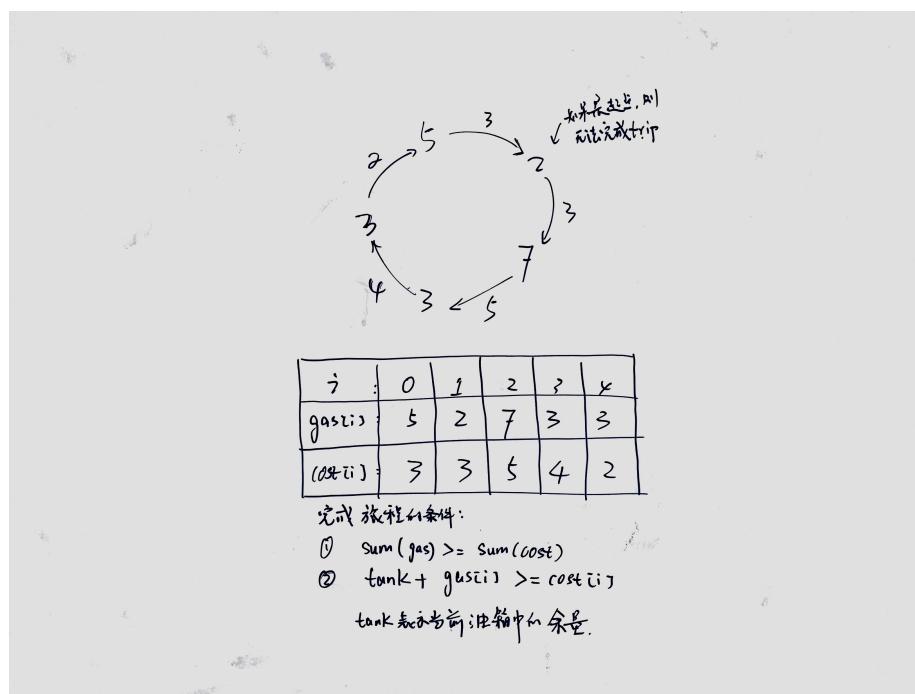
There are  $N$  gas stations along a circular route, where the amount of gas at station  $i$  is  $\text{gas}[i]$ . You have a car with an unlimited gas tank and it costs  $\text{cost}[i]$  of gas to travel from station  $i$  to its next station  $i + 1$ . You begin the journey with an empty tank at one of the gas stations. Return the starting gas station's index if you can travel around the circuit once, otherwise return -1.

Note: The solution is guaranteed to be unique.

解题思路：要想完成一圈必须满足两个条件：

1. 加油站总油量是否能够满足总消耗量；
2. 在每一个加油站，判断当前剩余油量与当前加油站能够提供的油量的总和是否可以完成接下来的消耗；如果不满足，那么说明当前的起点无法完成整个旅程，就要设置起点为下一个加油站。

参考下面的示例图。



```

1 class Solution(object):
2     def canCompleteCircuit(self, gas, cost):      # running time is O
3         (n)
4             """
5                 :type gas: List[int]
6                 :type cost: List[int]
7                 :rtype: int
8             """
9             if sum(gas)<sum(cost): return -1
10            station = 0

```

```

11     diff = 0
12     for i in range(len(gas)):
13         if diff+gas[i]<cost[i]:
14             station = i+1
15             diff = 0
16         else:
17             diff = diff+gas[i]-cost[i]
18
19     return station

```

Listing 20: Problem134. Gas Station

### 1.1.21 leetcode 135. Candy

There are N children standing in a line. Each child is assigned a rating value. You are giving candies to these children subjected to the following requirements:

- Each child must have at least one candy.
- Children with a higher rating get more candies than their neighbors.
- What is the minimum candies you must give?

解题思路：求最少的糖数。首先，给每个孩子一枚糖，保证每个孩子都有糖的要求；然后，保证等级比旁边孩子高的孩子获得等多的糖。那么“旁边”就分为左右两边，因此我们首先从左向右遍历数组，保证所有等级比其左边孩子高的孩子（除去左边第一个孩子），可以获得（比其左边孩子）多一枚糖；同理，为了保证所有等级比其右边孩子高的孩子（除去右边第一个孩子），可以获得（比其右边孩子）多一枚糖，再从右向左遍历数组一次。这样就可以完成题目要求的任务。示例见下图。

```

1  class Solution(object):
2      def candy(self, ratings):    # RT: O(n)
3          """
4              :type ratings: List[int]
5              :rtype: int
6          """
7
8          n = len(ratings)
9          # The child who has the lowest ratings has one candy.
10         candynum = [1 for _ in range(n)]
11
12         # scan the ratings from left to right, if the current
13         # child has higher rating than his predecessor, then
14         # he is given one more candy than his predecessor's.
15         for i in range(1, n):
16             if ratings[i] > ratings[i-1]:
17                 candynum[i] = candynum[i-1] + 1
18
19         # scan the ratings from right to left, if the current
20         # child has higher rating than his next neighbor, then
21         # he is given one more candy than his next neighbor's:
22             if ratings[i] > ratings[i+1] and
23                 candynum[i] <= candynum[i+1]:
24                     candynum[i] = candynum[i+1] + 1
25
26
27     return sum(candynum)

```

Listing 21: Problem135. Candy

i:	0	1	2	3	4	5	6	7
A[i]	3	1	2	5	4	3	2	5
candy[i]	1	1	1	1	1	1	1	1
1st	1	1	2	3	1	1	1	2
2nd	2	1	2	4	3	2	1	2
3rd	2	1	2	4	3	2	1	2

$$\therefore \text{total} = \sum(\text{candy}) = 2+1+2+4+3+2+1+2 = 17$$

$\overrightarrow{2nd}$ : if  $A[i] > A[i-1]$ :  $\text{Candy}[i] = \text{Candy}[i-1] + 1$

$\overleftarrow{3rd}$ : if  $A[i] > A[i+1]$ :  $\text{Candy}[i] = \text{Candy}[i+1] + 1$

### 1.1.22 leetcode 136. Single Number

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

Note: Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory?

解题思路：这道题的额外提示要求不使用额外空间就可以获得结果，所以可以明确这个题目的考点应该不是设计 hashtable 这类的数据结构来解题。这题考查的是位操作，需要明白基本位运算（与、或、非、异或、同或）的意义。针对这道题的描述，存在偶数个重复元素，而只有一个元素出现一次，所以使用异或(xor)操作就可以解决问题。异或操作的定义为： $x \text{ XOR } 0 = x$ ;  $x \text{ XOR } x = 0$ 。如果重复元素是奇数个呢？(leetcode137)

```

1
2 class Solution(object):
3     def singleNumber(self, nums):
4         """
5             :type nums: List[int]
6             :rtype: int
7             """
8         # basic XOR operations: x^x=0, x^0=x
9         # Suppose nums=[2,2,3,4,4], 2^2^3^4^4=(2^2)^3^(4^4)=3
10        # Suppose nums=[2,1,3,2,3], 2^1^3^2^3=3^2^3=1^3=1
11        sn = nums[0]
12        for i in range(1, len(nums)):
13            sn = sn ^ nums[i]
14

```

```
    return sn
```

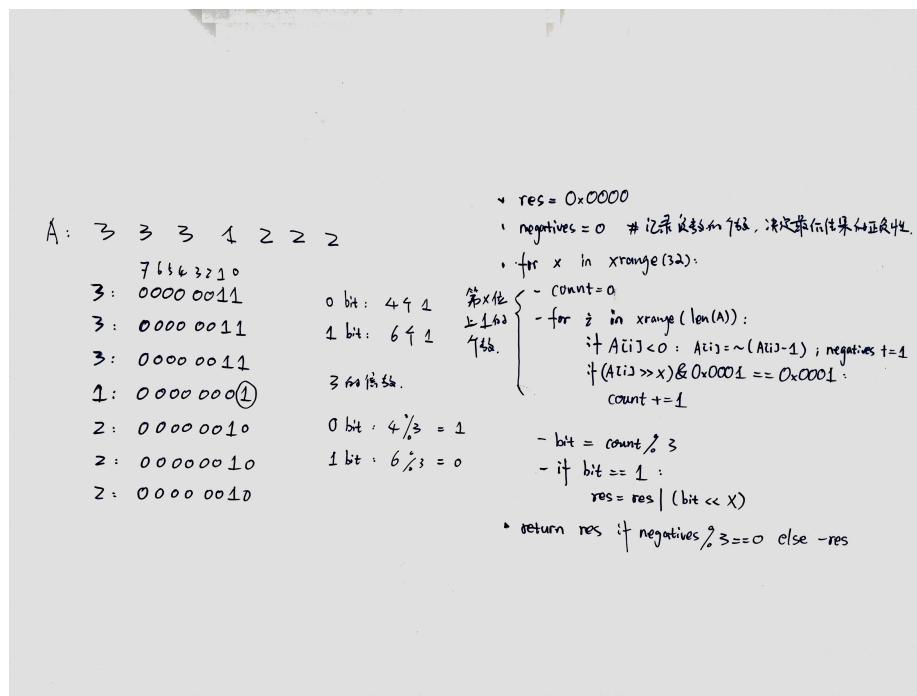
Listing 22: Problem136. Single Number

### 1.1.23 leetcode 137. Single Number II

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

Note: Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory?

解题思路：这道题与leetcode136. Single Number类似，仍然是考查位运算。参考下面示例图，我们将示例数组A中的每一个数值（示例中只是正整数，也可能是负值）用对应的二进制形式表示，可以观察到如下结果，例如在第1位上出现6个1，它们来自与数组A中分别重复出现3次的数值3和2；而在第0位上则出现了4个1，其中三个1来自于A中重复出现3次的数值3，而剩下的一个1来自于我们要寻找的在数组A中只出现一次的数值1。通过对这个简单的示例的观察，我们可以推导出这样一个计算方法：针对数组A中的所有数值，计算他们在每同一位（一共32位）上1出现的个数是否为3的倍数。如果是3的倍数，说明我们寻找的目标值对应的二进制形式在这个位上的数值是0；如果不是3的倍数，说明在这个位上数值是1，那么结果值res对应的位上就同样置为1。题目中数组A中的数值均为32位整数，所以这个计算过程只需要计算32次即可，所以时间复杂度为 $O(32 \times n) = O(n)$ 。另外，需要注意的是数组A中的数值有可能为负值，所以在计算1的个数之前需要将其转换为正值，转换过程即是取反再减1。



```

1
2 class Solution(object):
3     def singleNumber(self, nums):
4         """
5             :type nums: List[int]
6             :rtype: int
7             """
8             # the result is a 32-bit integer
9             res = 0
10            negatives = 0
11            for x in xrange(32):
12                count = 0
13                for i in xrange(len(nums)):
14                    if nums[i] < 0:
15                        nums[i] = ~(nums[i]-1)
16                        negatives += 1
17                    if (nums[i] >> x) & 1 == 1:
18                        count += 1
19                    bit = count % 3
20                    if bit == 1:
21                        res = res | (bit << x)
22            return res if negatives % 3 == 0 else -res

```

Listing 23: Problem137. Single Number II

```

1 // C# implementation
2 public class Solution {
3     public int SingleNumber(int[] nums) {
4         int[] bits = new int[32];
5         int result=0;
6
7         for( int i=0;i<32;i++)
8         {
9             for( int j=0; j<nums.Length; j++)
10             {
11                 bits[i] += nums[j]>>i & 1;
12             }
13             bits[i] %= 3;
14             result |= bits[i]<<i;
15         }
16     }
17     return result;
18 }
19 }
20 }

```

### 1.1.24 leetcode 126. Word Ladder II

Given two words (beginWord and endWord), and a dictionary's word list, find all shortest transformation sequence(s) from beginWord to endWord, such that:

1. Only one letter can be changed at a time
2. Each intermediate word must exist in the word list

For example, Given: beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot","log"], return

`[[“hit”, “hot”, “dot”, “dog”, “cog”], [“hit”, “hot”, “lot”, “log”, “cog”]]`

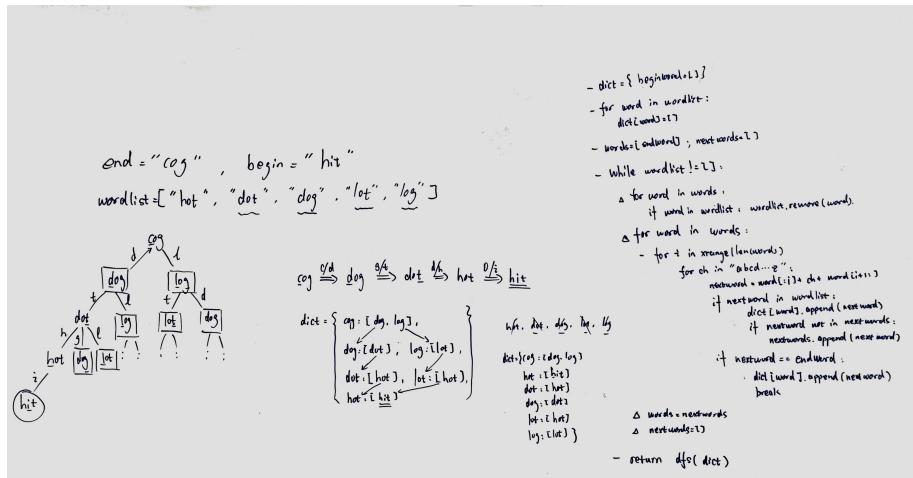
Note:

1. All words have the same length.
2. All words contain only lowercase alphabetic characters.

解题关键点：

1. 这里的dict是python中的set()类型。
2. 使用前驱单词表prevMap，是python中的字典类型。用来记录单词的前驱。比如prevMap={cog:[log,dog]}表示cog的前驱是：log和dog。
3. 在word ladder这道题中我们使用了队列，在word ladder ii中也需要使用队列，只不过在这个程序中使用了两个set()来模拟队列。candidates[previous]中存储的是前面一层的单词。如{dot, lot}为第三层单词。在程序开始执行时，现将前面一层的单词即candidates[previous]中的单词在dict中删除，再将candidates[current]清空，然后根据candidates[previous]中的单词寻找下一层的单词，如{dot, lot}的下一层为{dog, log}，并将{dog, log}存入candidates[current]中，同时将单词存入前驱单词表中。下一次循环开始时，上一次循环的candidates[current]变成了candidates[previous]，而上一次循环的candidates[previous]变成了candidates[current]并清空。如此反复执行，当某一次循环中的candidates[current]中出现了end单词时，说明我们的路径已经找出来了，工作完成了。
4. 路径的重建使用了一个基于DFS算法的子函数buildpath。

下面的示例图给出了分析过程及部分示例代码。图中的示例代码与下面给出的代码有些差别，但是都是基于同样的算法思想构建的。



```

1 class Solution(object):
2     def findLadders(self, beginWord, endWord, wordlist):
3         """
4             :type beginWord: str
5             :type endWord: str
6             :type wordlist: Set[str]
7             :rtype: List[List[int]]
8         """
9         # do dfs to construct paths from beginWord to endWord
10

```

```

11     # based on prevMap
12     def dfsBuildPath(path, word):
13         path.append(word)
14         if len(prevMap[word]) == 0:
15             currPath = path [:]
16             currPath.reverse()
17             res.append(currPath)
18             return
19         for w in prevMap[word]:
20             dfsBuildPath(path, w)
21             path.pop()
22
23     res = []
24     n = len(beginWord)
25
26     prevMap = {}
27     for word in wordlist:
28         prevMap[word] = []
29
30     # use two sets to simulate a queue by
31     # switch them each round
32     candidates = [set(), set()];
33     current, previous = 0, 1
34     candidates[current].add(beginWord)
35
36     while True:
37         current, previous = previous, current
38         for word in candidates[previous]:
39             wordlist.remove(word)
40         candidates[current].clear()
41         for word in candidates[previous]:
42             # each time change one character in 'word'
43             # if the new word exists in wordlist, save
44             # it in prevMap and candidates' current set
45             # for next round.
46             for i in range(n):
47                 part1=word[:i]; part2=word[i+1:]
48                 for j in 'abcdefghijklmnopqrstuvwxyz':
49                     if word[i]!=j:
50                         nextword = part1 + j + part2
51                         if nextword in wordlist:
52                             prevMap[nextword].append(word)
53                             candidates[current].add(nextword)
54
55             # if no previous word exists in wordlist
56             # it means beginWord cannot be transformed
57             # into endWord
58             if len(candidates[current]) == 0: return res
59             if endWord in candidates[current]: break
60
61     dfsBuildPath([], endWord)
62     return res

```

Listing 24: Problem126. Word Ladder II

### 1.1.25 leetcode 78. Subsets

Given a set of distinct integers, nums, return all possible subsets.

Note:

1. Elements in a subset must be in non-descending order.

2. The solution set must not contain duplicate subsets.

For example, If  $\text{nums} = [1,2,3]$ , a solution is:  $[[3],[1],[2],[1,2,3],[1,3],[2,3],[1,2],[]]$

解题思路：由于需要输出具体的结果，所以考虑使用DFS遍历决策树。题目要求子集中的元素是非递减的，所以在DFS之前应该先排序。需要注意的是题目要求不能有重复的子集，去重可以考虑两种方法：（1）使用集合set数据结构；（2）先排序，然后在输出结果之前在判断一下获得的结果在当前结果集中是否存在。下面的代码使用的是第二种方式。

```
1 class Solution(object):
2     def subsets(self, nums):
3         """
4             :type nums: List[int]
5             :rtype: List[List[int]]
6         """
7         def dfs(depth, start, valuelist):
8             if valuelist not in res: res.append(valuelist)
9             if depth==len(nums): return
10            for i in range(start, len(nums)):
11                dfs(depth+1, i+1, valuelist+[nums[i]])
12
13        nums.sort()
14        res = []
15        dfs(0, 0, [])
16
17    return res
```

Listing 25: Problem78. Subsets

### 1.1.26 leetcode 90. Subsets II

Given a collection of integers that might contain duplicates,  $\text{nums}$ , return all possible subsets.

Note:

1. Elements in a subset must be in non-descending order.

2. The solution set must not contain duplicate subsets.

If  $\text{nums} = [1,2,2]$ , a solution is:  $[[2],[1],[1,2,2],[2,2],[1,2],[]]$

解题思路：这道题和leetcode78 Subset的解题思路基本一致，两题的唯一差别在于有重复元素，但是可以直接使用leetcode78的算法。如果想体现两个不同题目在算法设计上的差别，就在每一次循环的时候增加一个判断条件忽略掉重复元素（代码line 13）。但是，要这样做的前提是必须先排序。示例图如下所示。

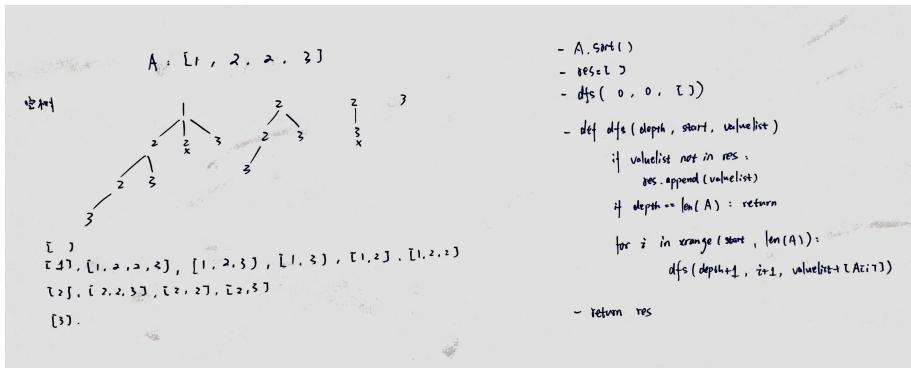
```
1 class Solution(object):
2     def subsetsWithDup(self, nums):
3         """
4
```

```

5     : type  nums: List[int]
6     : rtype: List[List[int]]
7     """
8     def dfs(depth, start, valuelist):
9         if valuelist not in res:
10            res.append(valuelist)
11        if depth==len(nums): return
12        for i in range(start, len(nums)):
13            if i==start or (i>start and nums[i]!=nums[i-1]):
14                dfs(depth+1, i+1, valuelist + [nums[i]])
15    res = []
16    nums.sort()
17    dfs(0,0,[])
18    return res

```

Listing 26: Problem90. Subsets II



### 1.1.27 leetcode 54. Spiral Matrix

Given a matrix of  $m \times n$  elements (m rows, n columns), return all elements of the matrix in spiral order.

For example, Given the following matrix:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

You should return [1,2,3,6,9,8,7,4,5].

解题思路：螺旋遍历数组（从左向右，自上而下），一个比较直接的想法就是在每次遍历之前知道遍历的起点和终点，即每次遍历的矩阵的边界，我们可以用四个变量来记录每次遍历的边界值（上下左右）：top, bottom, left, right。而每次遍历使用那一对儿边界值，可以通过每次遍历的方向（也是上下左右）来确定；同样我们有一个方向变量来记录每次遍历的方向，而且这个方向变量遵循一个规律就是先向右，再向下，然后向左，最后向上，这是一个遍历周期；下一个周期依然遵循同样的模式，直到整个矩阵遍历完。据此，我们可以通过对方向变量模除4的方式确定每次遍历的方向（代码line 29）。

```

1
2 def spiralOrder(self, matrix):
3     res = []

```

```

4     if matrix==[]: return res
5     m, n = len(matrix), len(matrix[0])
6     top, bottom = 0, m-1
7     left, right = 0, n-1
8     # direct indicates the direction of current move
9     # 0:to right, 1:downwards, 2:to left, 3:upwards
10    direct = 0
11    while True:
12        if direct==0:
13            for y in xrange(left, right+1):
14                res.append(matrix[top][y])
15                top += 1
16        if direct==1:
17            for x in xrange(top, bottom+1):
18                res.append(matrix[x][right])
19                right -= 1
20        if direct==2:
21            for y in xrange(right, left-1, -1):
22                res.append(matrix[bottom][y])
23                bottom -= 1
24        if direct==3:
25            for x in xrange(bottom, top-1, -1):
26                res.append(matrix[x][left])
27                left += 1
28        if top>bottom or left>right: return res
29        direct = (direct+1)%4

```

Listing 27: Problem54. Spiral Matrix

### 1.1.28 leetcode 59. Spiral Matrix II

Given an integer  $n$ , generate a square matrix filled with elements from 1 to  $n^2$  in spiral order.

For example, Given  $n = 3$ , You should return the following matrix:

$$\begin{bmatrix} 1 & 2 & 3 \\ 8 & 9 & 4 \\ 7 & 6 & 5 \end{bmatrix}$$

解题思路：这道题是leetcode54. Spiral Matrix的逆过程，但是两题的相同点：本题填充矩阵的顺序即为leetcode54遍历矩阵的顺序。因此，算法在遍历数组的部分是完全相同的，差别在于while循环的结束条件(代码line30)。

```

1 class Solution(object):
2     def generateMatrix(self, n):
3         """
4             :type n: int
5             :rtype: List[List[int]]
6         """
7         if n == 0: return []
8         matrix = [[0 for i in range(n)] for j in range(n)]
9         top = 0; bottom = len(matrix)-1
10        left = 0; right = len(matrix[0])-1
11        direct = 0; count = 0
12        while True:
13            if direct == 0:
14                for i in range(left, right+1):
15                    count += 1; matrix[top][i] = count
16                    top += 1

```

```

18         elif direct == 1:
19             for i in range(top, bottom+1):
20                 count += 1; matrix[i][right] = count
21                 right -= 1
22             elif direct == 2:
23                 for i in range(right, left-1, -1):
24                     count += 1; matrix[bottom][i] = count
25                     bottom -= 1
26             elif direct == 3:
27                 for i in range(bottom, top-1, -1):
28                     count += 1; matrix[i][left] = count
29                     left += 1
30     if count == n*n: return matrix
31     direct = (direct+1) % 4

```

Listing 28: Problem59. Spiral Matrix II

### 1.1.29 leetcode 238. Product of Array Except Self

Given an array of  $n$  integers where  $n > 1$ ,  $\text{nums}$ , return an array output such that  $\text{output}[i]$  is equal to the product of all the elements of  $\text{nums}$  except  $\text{nums}[i]$ . Solve it without division and in  $O(n)$ .

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

Follow up: Could you solve it with constant space complexity? (Note: The output array does not count as extra space for the purpose of space complexity analysis.)

解题思路: 由于 $\text{output}[i] = (x_0 \times x_1 \times \dots \times x_{i-1}) \times (x_{i+1} \times \dots \times x_{n-1})$ ，因此执行两趟循环:

- 第一趟正向遍历数组，计算 $x_0 \rightarrow x_{i-1}$ 的乘积；
- 第二趟反向遍历数组，计算 $x_{i+1} \rightarrow x_{n-1}$ 的乘积。
- 两个结果再相乘就是 $\text{output}[i]$ 的值。

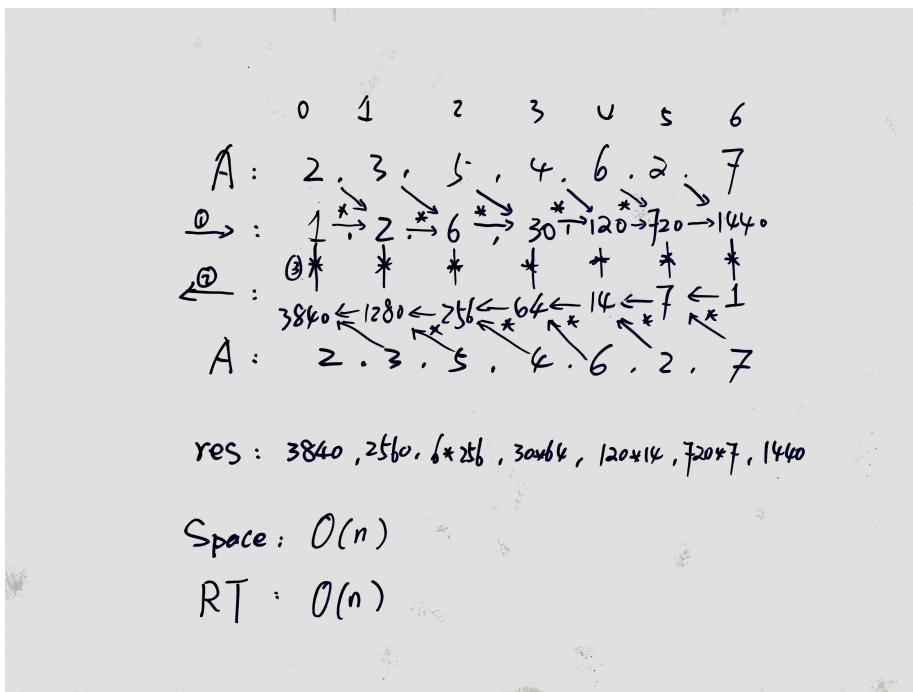
下面的示例图给出了具体的演算过程。

```

1  class Solution(object):
2      def productExceptSelf(self, nums):
3          """
4              :type nums: List[int]
5              :rtype: List[int]
6              """
7
8          n = len(nums)
9          output = [1]*n
10         left = 1
11         for x in range(n-1):
12             left *= nums[x]
13             output[x+1] *= left
14         right = 1
15         for x in range(n-1, 0, -1):
16             right *= nums[x]
17             output[x-1] *= right

```

Listing 29: Problem238. Product of Array Except Self



### 1.1.30 leetcode 268. Missing Number

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.

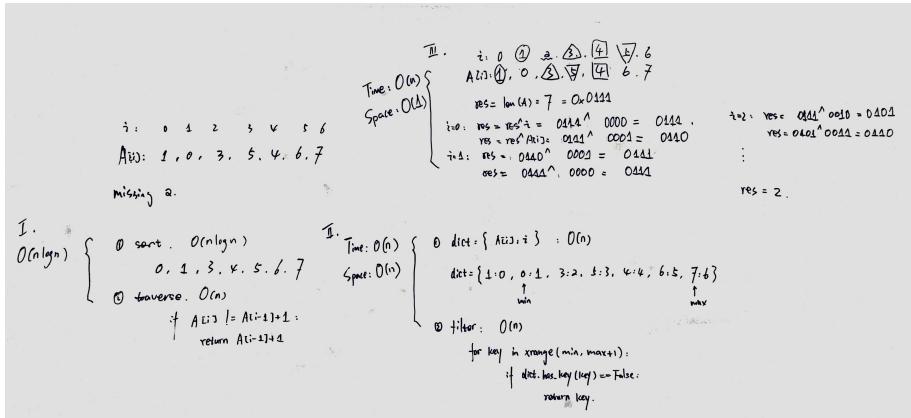
Note: Your algorithm should run in linear runtime complexity. Could you implement it using only constant extra space complexity?

解题思路：这道题可以通过三种算法解决：

- 第一种算法比较简单：先对数组排序，然后依次第比较前后两个数值的差是否为1：如果差值唯一，说明连续；如果差值不唯一，说明找到缺失的那个数值，返回即可。这个算法的时间复杂度受排序影响达到 $O(n\log n)$ ，但不需要额外存储空间。
- 第二种算法是改进第一种算法的时间复杂度，使用hashtable的思想（Python实现用的是dict）。字典的key是数组中的值，value是对应的数组索引在构建字典dict的时候可以记录序列中的最大值和最小值；然后从最小值开始，依次查看当前key是否在dict中，如果在，则用key+1更新key，否则返回当前key即可。这种算法的时间复杂度降到 $O(n)$ ，但空间复杂度上升到 $O(n)$ 。

- 第三种算法进一步改进第二种算法，将空间复杂度降到 $O(1)$ 。算法的思想来自于leetcode136 Single Number。通过观察可以发现，长度为 $n$ 的数组中数值的范围是 $[0..n]$ ，那么全部的索引值和所有值之间存在一种关系：缺失的值只在索引中出现过一次，而未缺失的值则在索引和数值中各出现一次，即出现两次，那么通过异或运算，即可消除掉出现过两次的数值，最后剩下的即为缺失的数值。

示例及演算过程见下面示例图。



```

1
2 class Solution(object):
3     def missingNumber(self, nums): # RT: O(n)
4         """
5             :type nums: List[int]
6             :rtype: int
7         """
8         res = len(nums)
9         for i in xrange(len(nums)):
10            res = res^i
11            res = res^nums[i]
12        return res

```

Listing 30: Problem268. Missing Number

### 1.1.31 leetcode 209. Minimum Size Subarray Sum

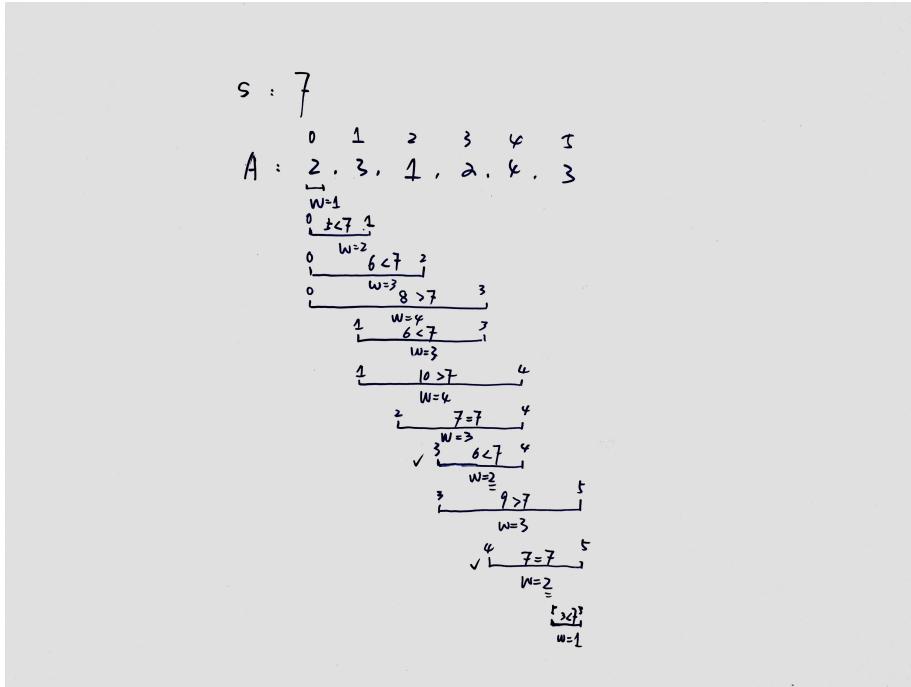
Given an array of  $n$  positive integers and a positive integer  $s$ , find the minimal length of a subarray of which the sum  $s$ . If there isn't one, return 0 instead.

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.

More practice: If you have figured out the  $O(n)$  solution, try coding another solution of which the time complexity is  $O(n \log n)$ .

解题思路：开辟一个窗口，通过滑动窗口的左右边界，取得满足条件的最小窗口大小即为最小子数组的长度。滑动窗口的条件是：如果当前窗口内的所有数值的和小于目标值时，增大窗口，即窗口右边界增1，向当前窗口内增加一个数

值；如果当前窗口内的所有数值和大于或等于目标值时，记录当前窗口值，同时缩小窗口，即窗口左边界增1，将当前窗口最左侧的数值移除窗口。上述过程的结束条件就是窗口右边界达到数组边界。参考下面的示例图。



```

1  class Solution(object):
2      def minSubArrayLen(self, s, nums): # RT: O(n)
3          """
4              :type s: int
5              :type nums: List[int]
6              :rtype: int
7              """
8
9          size = len(nums)
10         start, end, sum = 0, 0, 0
11         window = size + 1
12         while end < size:
13             while end < size and sum < s:
14                 sum += nums[end]
15                 end += 1
16             while start < end and sum >= s:
17                 window = min(window, end - start)
18                 sum -= nums[start]
19                 start += 1
20         return [0, window][window <= size]

```

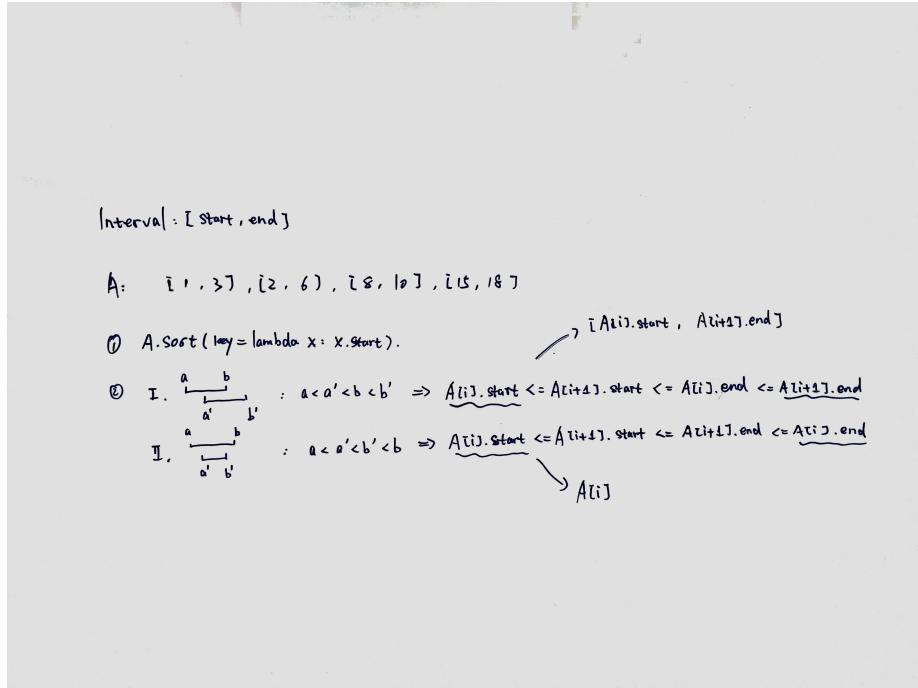
Listing 31: Problem209. Minimum Size Subarray Sum

### 1.1.32 leetcode 56. Merge Intervals

Given a collection of intervals, merge all overlapping intervals.

For example, Given [1,3],[2,6],[8,10],[15,18], return [1,6],[8,10],[15,18].

解题思路：首先按照start的值对区间进行排序（从小到大）；然后比较当前区间的end与下一个区间的start的大小，如果两区间存在重叠，那么就合并这两个区间；否则下一个区间变为当前区间，重复上述过程。参考下面的示例图。注意：在比较开始前，需要排序的动作，因为给定的输入可能不是有序的。



```

1
2 # Definition for an interval.
3 # class Interval(object):
4 #     def __init__(self, s=0, e=0):
5 #         self.start = s
6 #         self.end = e
7
8 class Solution(object):
9     def merge(self, intervals):
10         """
11             :type intervals: List[Interval]
12             :rtype: List[Interval]
13         """
14         intervals.sort(key=lambda x:x.start)
15         m = len(intervals)
16         res = []
17         if m==0: return res
18         tmp = intervals[0]
19         for x in range(1,m):
20             if intervals[x].start <= tmp.end:
21                 tmp.end = max(tmp.end, intervals[x].end)
22             else:
23                 res.append(tmp)
24                 tmp = intervals[x]
25         res.append(tmp)

```

26      **return** res

Listing 32: Problem56. Merge Intervals

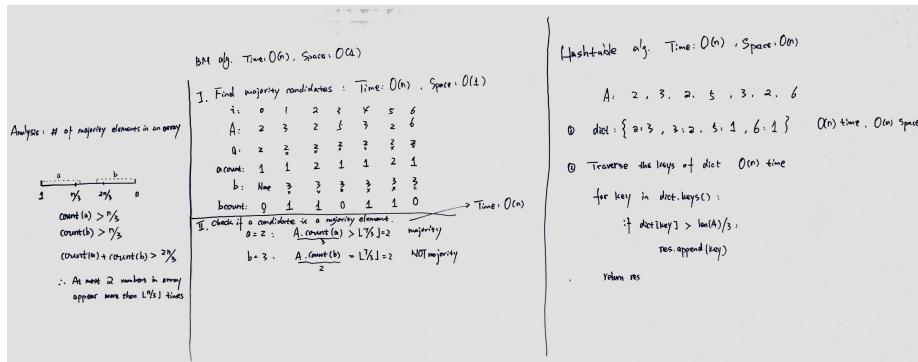
### 1.1.33 leetcode 229. Majority Element II

Given an integer array of size n, find all elements that appear more than  $\lfloor n/3 \rfloor$  times. The algorithm should run in linear time and in  $O(1)$  space.

解题思路：这道题有两种解法：

- 第一种方法是使用Hashtable求解。时间复杂度可以达到 $O(n)$ ，但是空间复杂度也达到 $O(n)$ 。
- 第二种方法是Boyer-Moore majority vote算法的改进算法。算法的思想基本与BM算法一致，分为两个阶段：第一个阶段求出候选的众数；第二阶段是使用题目的条件（出现次数多于 $\lfloor n/3 \rfloor$ ）过滤候选的众数。这个算法可以在 $O(n)$ 时间内完成，同时满足题目对空间复杂度的要求 $O(1)$ 。

参见下面示例图中的分析和演算。



```

1  class Solution(object):
2      def majorityElement_dict(self, nums): # RT: O(n), Space: O(n)
3          dict = {}
4          for num in nums:
5              if dict.has_key(num)==False:
6                  dict[num] = 1
7              else:
8                  dict[num] += 1
9          res = []
10         for key in dict.keys():
11             if dict[key] > len(nums) / 3:
12                 res.append(key)
13         return res
14
15     def majorityElement_BM(self, nums): # RT: O(n), Space: O(1)
16         """
17         :type nums: List[int]
18         :rtype: List[int]
19         """
20         def majorityElement(self, nums):
21             n = len(nums)
22

```

```

23     candidate1, count1 = None, 0
24     candidate2, count2 = None, 0
25     for num in nums:
26         if num == candidate1: count1 += 1
27         elif num == candidate2: count2 += 1
28         elif count1 == 0: candidate1, count1 = num, 1
29         elif count2 == 0: candidate2, count2 = num, 1
30         else: count1 -= 1; count2 -= 1
31     return [x for x in (candidate1, candidate2) if nums.count(x)
) > n/3]

```

Listing 33: Problem229. Majority Element II

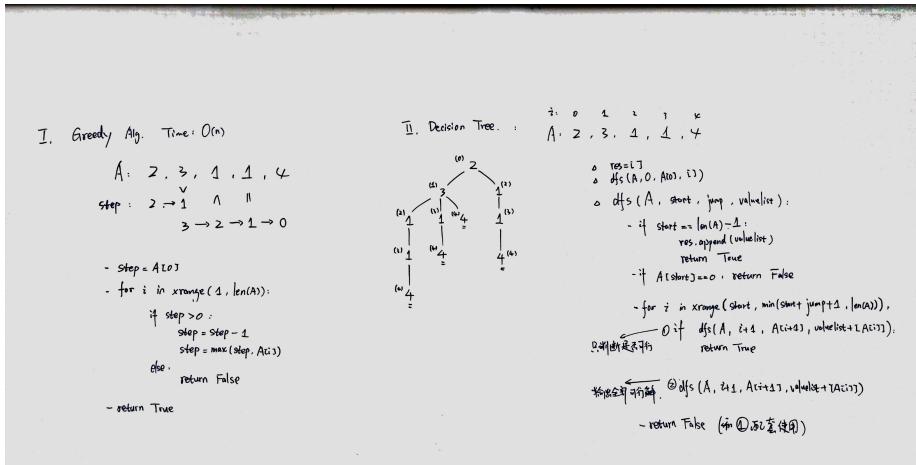
### 1.1.34 leetcode 55. Jump Game

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:

1. A = [2,3,1,1,4], return true.
2. A = [3,2,1,0,4], return false.

解题思路：这道题可以用决策树也可以用贪心算法。但是，决策树会超时。贪心算法是从起点开始，每向前走一步，就比较当前位置可以走的步数和剩余可走步数的大小，每次取两者最大值。如此进行下去，如果在迭代结束之前，可走的最大步数为0，说明没法走到数组的末端。两种算法的示例和演算见下图。



```

1  class Solution(object):
2      def canJump(self, nums):
3          """
4              :type nums: List[int]
5              :rtype: bool
6              """
7          step = nums[0]
8

```

```

9         for i in range(1, len(nums)):
10        if step > 0:
11            step -= 1
12            step = max(step, nums[i])
13        else:
14            return False
15    return True

```

Listing 34: Problem55. Jump Game

### 1.1.35 leetcode 45. Jump Game II

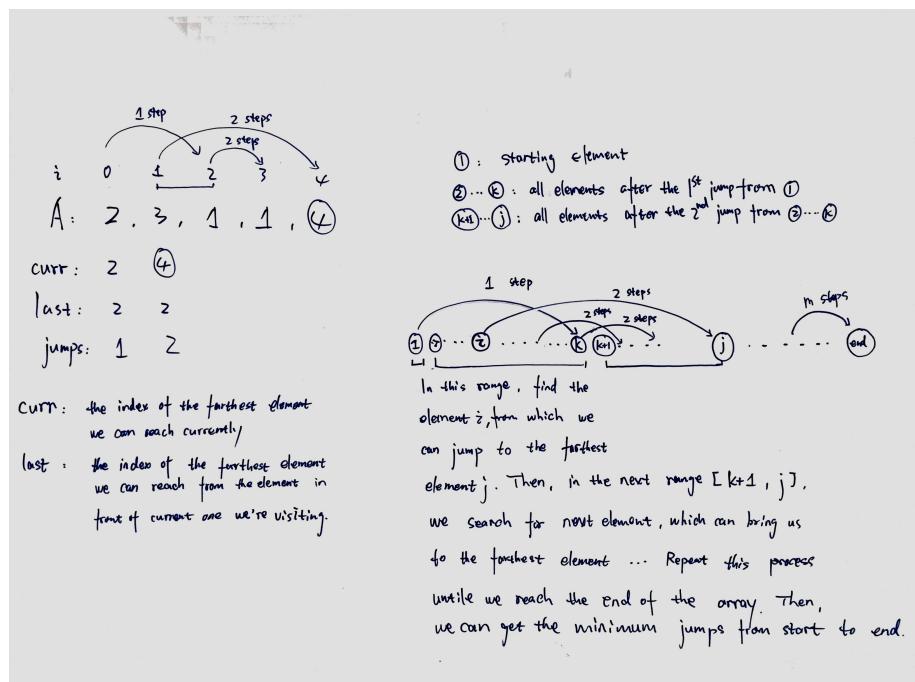
Given an array of non-negative integers, you are initially positioned at the first index of the array. Each element in the array represents your maximum jump length at that position. Your goal is to reach the last index in the minimum number of jumps.

For example: Given array A = [2,3,1,1,4]. The minimum number of jumps to reach the last index is 2. (Jump 1 step from index 0 to 1, then 3 steps to the last index.)

Note: You can assume that you can always reach the last index.

这道题的解题思路见示例图右侧部分。示例图左侧为具体示例。

代码说明: 用lastCanReach记录jumpNum次跳跃后, 所能够到达的最大索引值; 用currCanReach记录在lastCanReach的范围内所有索引能够到达的最大索引值。如果当前遍历的索引值超过了lastCanReach的范围, 则用currCanReach更新lastCanReach, 这也意味着需要多跳跃一次, 即jumpNum增1。



```

1
2 class Solution(object):
3     def jump(self, nums):
4         n = len(nums)
5         if n==1: return 0
6         jumpNum = 0
7         currCanReach = 0
8         lastCanReach = 0
9         for x in xrange(n):
10             if x > lastCanReach:
11                 lastCanReach = currCanReach
12                 jumpNum += 1
13                 currCanReach = max(currCanReach, x+nums[x])
14

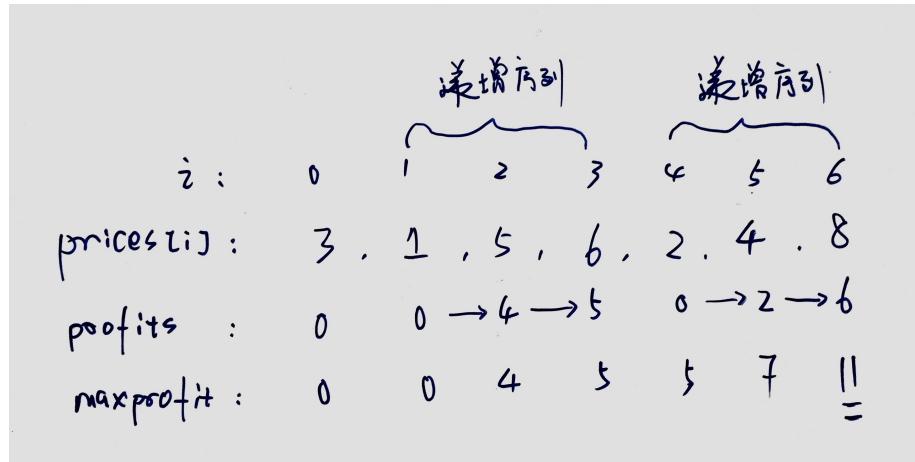
```

Listing 35: Problem45. Jump Game II

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

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).

解题思路：由于可以进行无限次的交易，那么只要是递增序列，就可以进行利润的累加。参考下面的示例图。



```

1
2 class Solution(object):
3     def maxProfit(self, prices):
4         """
5             :type prices: List[int]
6             :rtype: int
7         """
8         maxprofits = 0
9         n = len(prices)
10        if n==0: return maxprofits
11        for i in range(1,n):

```

```

12     if prices[i] > prices[i-1]:
13         maxprofits += prices[i] - prices[i-1]
14     return maxprofits

```

Listing 36: Problem122. Best Time to Buy and Sell Stock II

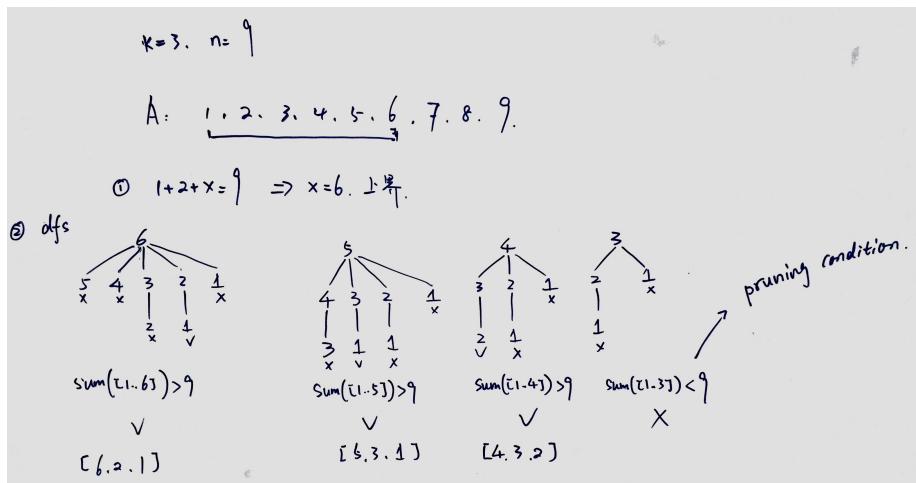
### 1.1.37 leetcode 216. Combination Sum III

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.

Examples:

1. Input:  $k = 3$ ,  $n = 7$ , Output:  $[[1,2,4]]$
2. Input:  $k = 3$ ,  $n = 9$ , Output:  $[[1,2,6], [1,3,5], [2,3,4]]$

解题思路：决策树找出满足条件的所有子集，用DFS。参考下面的示例图。



```

1 class Solution(object):
2     def combinationSum3(self, k, n):
3         """
4             :type k: int
5             :type n: int
6             :rtype: List[List[int]]
7         """
8         def dfs(nums, start, k, n, valuelist):
9             if n==0 and k==0: res.append(valuelist)
10            if k==0: return
11            for x in xrange(start, len(nums)):
12                if n < nums[x]: return
13                dfs(nums, x+1, k-1, n-nums[x], valuelist+[nums[x]])
14
15        res = []
16        nums = [i for i in range(1,10)]
17        dfs(nums, 0, k, n, [])
18

```

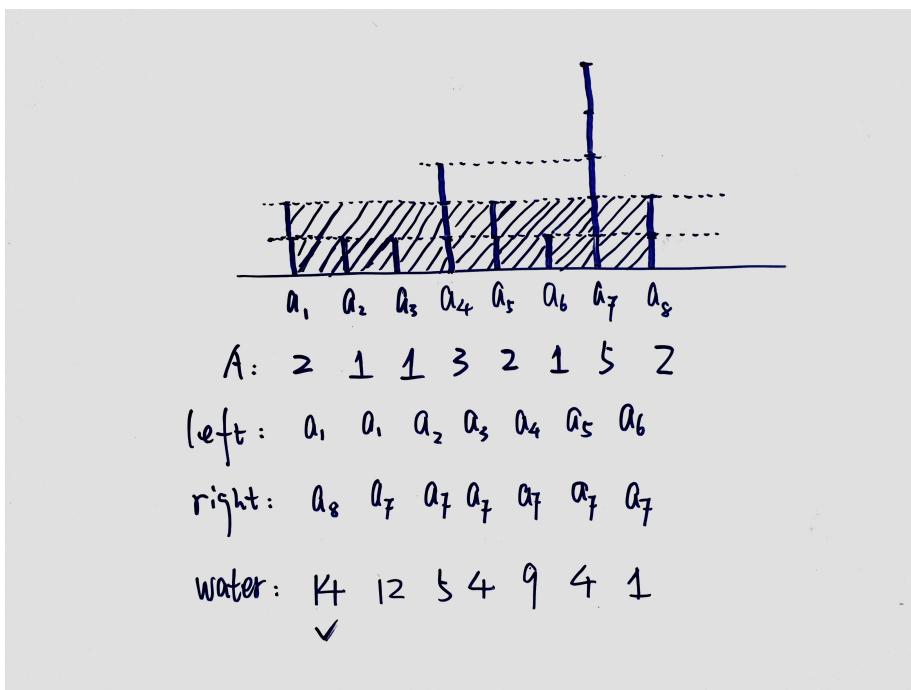
```
19     return res
```

Listing 37: Problem216. Combination Sum III

### 1.1.38 leetcode 11. Container With Most Water

Given  $n$  non-negative integers  $a_1, a_2, \dots, a_n$ , where each represents a point at coordinate  $(i, a_i)$ .  $n$  vertical lines are drawn such that the two endpoints of line  $i$  is at  $(i, a_i)$  and  $(i, 0)$ . Find two lines, which together with x-axis forms a container, such that the container contains the most water. Note: You may not slant the container.

解题思路：这道题与leetcode 42. Trapping Rain Water的解题思路类似。两个隔板的矮的那个的高度乘以两个隔板的间距就是储水量。



```
1 class Solution(object):
2     def maxArea(self, height): # RT: O(n)
3         """
4             :type height: List[int]
5             :rtype: int
6             """
7             n = len(height)
8             left, right = 0, n-1
9             maxarea = 0
10            while left < right:
11                area = min(height[left], height[right]) * abs(right-
12                    left)
13                maxarea = max(maxarea, area)
14                if height[left] < height[right]:
```

```

15         left += 1
16     else:
17         right -= 1
18 return maxarea

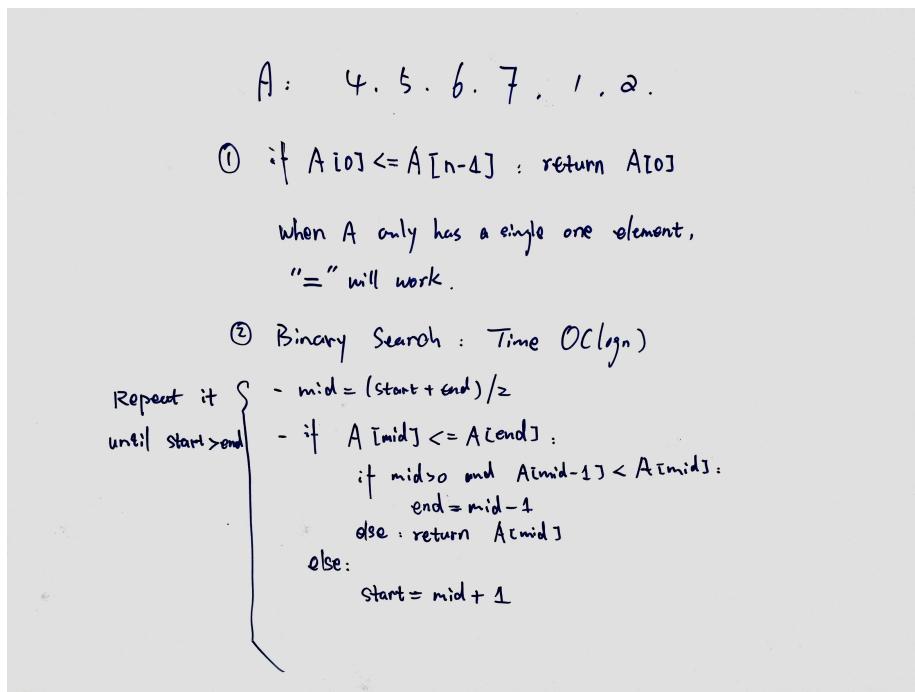
```

Listing 38: Problem11. Container With Most Water

### 1.1.39 leetcode 153. Find Minimum in Rotated Sorted Array

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.

解题思路：这道题与leetcode 33. Search in Rotated Sorted Array的解题思路类似，只是在移动start和end的判断条件上有所区别。两题都是考查binary search算法。参见下面示例图。



```

1 class Solution(object):
2     def findMin(self, nums): # RT: O(logn)
3         n = len(nums)
4         if nums[0] <= nums[-1]:
5             return nums[0]
6         start, end = 0, n-1
7         while start <= end:
8             mid = (start+end)/2
9             if nums[mid] <= nums[end]:
10                 if mid > 0 and nums[mid-1] <= nums[mid]:
11                     end = mid-1
12                 else:
13                     return nums[mid]
14

```

```

15         else :
16             start = mid+1
17     return -1

```

Listing 39: Problem153. Find Minimum in Rotated Sorted Array

#### 1.1.40 leetcode 154. Find Minimum in Rotated Sorted Array II

Follow up for "Find Minimum in Rotated Sorted Array": What if duplicates are allowed? Would this affect the run-time complexity? How and why?

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.

解题思路：这道题与leetcode 81. Search in Rotated Sorted Array II的解题思路类似，只是在移动start和end的判断条件上有所区别。两题都是考查存在重复元素的时候，binary search算法的应用，以及对binary search算法性能的评估。这道题在面试的时候应该是考查对BS搜索算法最差时间复杂度情况的讨论。

```

1  class Solution(object):
2      # binary search + recursion
3      def findMin(self, nums): # RT: O(logn), the worse case in O(n)
4          time
5          n = len(nums)
6          minval = nums[0]
7          if n==1 or nums[0]<nums[-1]:
8              return minval
9          start, end = 0, n-1
10         while start <= end:
11             mid = (start+end)/2
12             minval = min(minval, nums[mid])
13             if nums[mid] < nums[end]:
14                 end = mid-1
15             elif nums[mid] > nums[end]:
16                 start = mid+1
17             else:
18                 if start < mid:
19                     minval = min(minval, self.findMin(nums[start:
20                         mid+1]))
21                     if mid+1 < end:
22                         minval = min(minval, self.findMin(nums[mid+1:
23                         end+1]))
24                     break
25     return minval

```

Listing 40: Problem154. Find Minimum in Rotated Sorted Array II

#### 1.1.41 leetcode 57. Insert Interval

Given a set of non-overlapping intervals, insert a new interval into the intervals (merge if necessary). You may assume that the intervals were initially sorted according to their start times.

Example 1: Given intervals [1,3],[6,9], insert and merge [2,5] in as [1,5],[6,9].

Example 2: Given [1,2],[3,5],[6,7],[8,10],[12,16], insert and merge [4,9] in as [1,2],[3,10],[12,16]. This is because the new interval [4,9] overlaps with [3,5],[6,7],[8,10].

```
1 # Definition for an interval.
2 # class Interval(object):
3 #     def __init__(self, s=0, e=0):
4 #         self.start = s
5 #         self.end = e
6 #
7
8 class Solution(object):
9     def insert(self, intervals, newInterval): # RT: O(nlogn)
10        """
11        :type intervals: List[Interval]
12        :type newInterval: Interval
13        :rtype: List[Interval]
14        """
15        intervals.append(newInterval)
16        intervals.sort(key=lambda x:x.start)
17        n = len(intervals)
18        res = []
19        for i in range(n):
20            if res == []:
21                res.append(intervals[i])
22            else:
23                size = len(res)
24                if res[size-1].start <= intervals[i].start <= res[
25                    size - 1].end:
26                    res[size-1].end = max(res[size-1].end,
27                                         intervals[i].end)
28                else:
29                    res.append(intervals[i])
30
31        return res
```

Listing 41: Problem57. Insert Interval

### 1.1.42 leetcode 287. Find the Duplicate Number

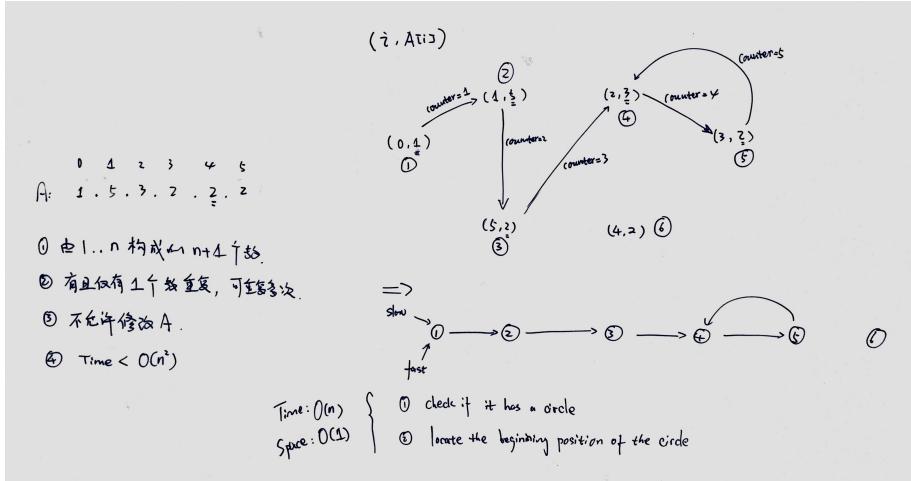
Given an array  $\text{nums}$  containing  $n + 1$  integers where each integer is between 1 and  $n$  (inclusive), prove that at least one duplicate number must exist. Assume that there is only one duplicate number, find the duplicate one.

Note:

1. You must not modify the array (assume the array is read only).
2. You must use only constant,  $O(1)$  extra space.
3. Your runtime complexity should be less than  $O(n^2)$ .
4. There is only one duplicate number in the array, but it could be repeated more than once.

解题思路：这道题比较难，需要对问题重新建模。参见下面示例图。因为题目限定了长为 $n$ 的数组中填充的数值范围在 $1 \dots n$ ，所以数组中的数值和数组的索引值都在这个范围内。假设 $A[i]=j$ ，那么我们可以构建一个有向图，图中结点

的结构可以用一个元组 $(i, A[i])$ 表示；如果图中有两个结点 $(i, A[i])$ 和 $(j, A[j])$ ，当且仅当 $A[i] == j$ 时，存在一条从结点 $(i, A[i])$ 到结点 $(j, A[j])$ 的有向边。如示例图所示，在构建这个有向图的过程中，我们可以观察到，如果数组中存在重复数值，那么在图中一定出现一个环，并且数组中构成环路以后其余数值将构成另外一个有向无环图DAG。为了找到重复数值，只需关注这个有环的图。我们再来看分析一下这个有环图，从起点开始到构成环所经过的路径构成了 $\rho$ 形，这就与leetcode 141/142. Linked List Cycle I/II两题相似：首先用一次遍历确定是否有环；如果环存在，再遍历一遍确定环的起始位置，即重复元素。这部分的算法设计与leetcode142一致。



```

1  class Solution(object):
2      # tortoise and hare principle
3      def findDuplicate1(self, nums): # RT: O(n)
4          # The "tortoise and hare" step. We start at the end of the
5          # array and try
6          # to find an intersection point in the cycle.
7          slow = 0
8          fast = 0
9
10         # Keep advancing 'slow' by one step and 'fast' by two steps
11         until they
12         # meet inside the loop.
13         while True:
14             slow = nums[slow]
15             fast = nums[nums[fast]]
16
17             if slow == fast:
18                 break
19
20         # Start up another pointer from the end of the array and
21         # march it forward
22         # until it hits the pointer inside the array.
23         finder = 0
24         while True:
25             slow = nums[slow]
26             finder = nums[finder]
27
28         # If the two hit, the intersection index is the
29         # duplicate element.

```

```

27     if slow == finder:
28         return slow
29
30     # binary search + pigeonhole principle
31     def findDuplicate2(self, nums): # RT: O(nlogn)
32         """
33             :type nums: List[int]
34             :rtype: int
35         """
36         left, right = 0, len(nums)-1
37         while left <= right:
38             mid = (left+right) >> 1
39             tmp = sum(x <= mid for x in nums)
40             if tmp > mid:
41                 right = mid-1
42             else:
43                 left = mid+1
44         return left
45
46     def findDuplicate3(self, nums): # RT: O(nlogn)
47         n = len(nums)
48         nums.sort()
49         for x in xrange(n-1):
50             if nums[x] == nums[x+1]:
51                 return nums[x]

```

Listing 42: Problem287. Find the Duplicate Number

#### 1.1.43 leetcode 162. Find Peak Element

A peak element is an element that is greater than its neighbors. Given an input array where  $num[i] \neq num[i + 1]$ , 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  $num[-1] = num[n] = -\infty$ .

For example, in array [1, 2, 3, 1], 3 is a peak element and your function should return the index number 2.

解题思路：这道题最直接的解法就是遍历数组，比较相邻的两个数，每次记录较大的数的索引，最后返回记录。这个解法的时间复杂度显然就是 $O(n)$ 。然而搜索某种特殊值时，最高效的算法就是binary search，几乎适用于所有在一堆候选者中（1）找最值（有序的情况下，一次去掉一半），或者（2）满足某种特殊条件的特定值（这种情况下，候选者是有序排列还是无序排列无所谓）。针对本题的两种解法，参考示例图中的演算过程。

```

1  class Solution(object):
2      def findPeakElement(self, num): # RT: O(logn)
3          size = len(num)
4          return self.search(num, 0, size - 1)
5
6      def search(self, num, start, end):
7          if start == end:
8              return start
9          if start + 1 == end:
10              return [start, end][num[start] < num[end]]
11          mid = (start + end) / 2

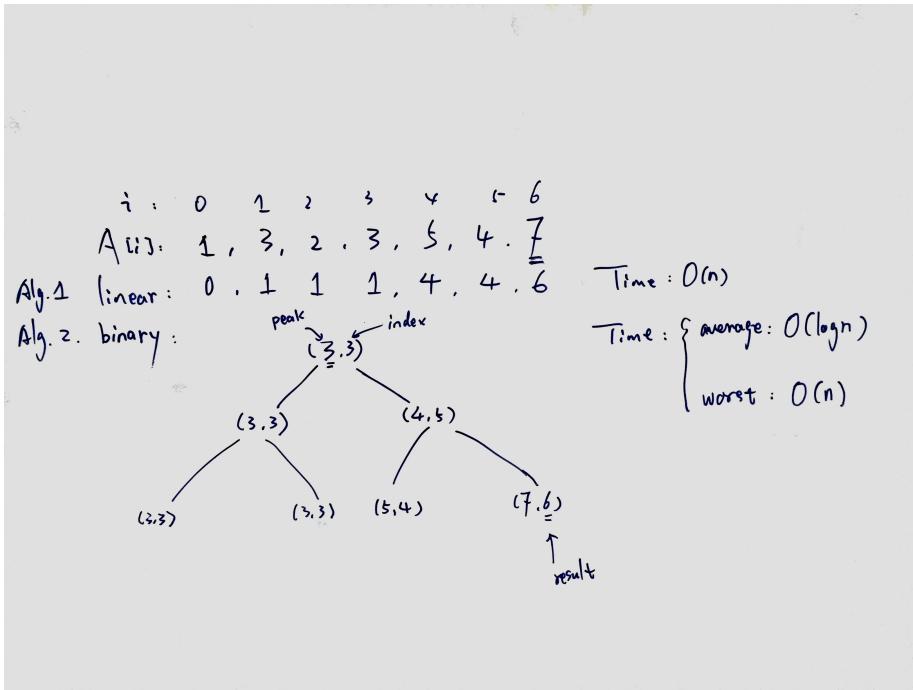
```

```

13     if num[mid] < num[mid - 1]:
14         return self.search(num, start, mid - 1)
15     if num[mid] < num[mid + 1]:
16         return self.search(num, mid + 1, end)
17     return mid
18
19 def findPeakElement2(self, nums): # RT: O(n)
20     """
21     :type nums: List[int]
22     :rtype: int
23     """
24     n = len(nums)
25     for i in range(1, n-1):
26         if nums[i-1] < nums[i] and nums[i] > nums[i+1]:
27             return i
28     return [0, n-1][nums[0] < nums[n-1]]

```

Listing 43: Problem162. Find Peak Element



## 1.2 Singly Linked List

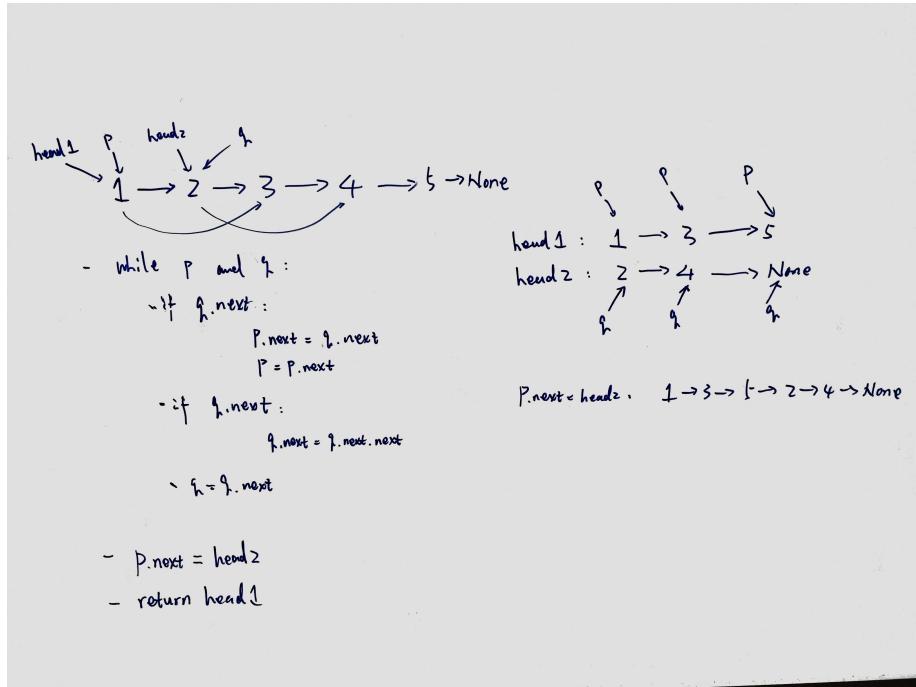
### 1.2.1 leetcode 328. Odd Even Linked List

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  $O(\text{nodes})$  time complexity.

Example: Given 1->2->3->4->5->NULL, return 1->3->5->2->4->NULL.

Note: The relative order inside both the even and odd groups should remain as it was in the input. The first node is considered odd, the second node even and so on ...

解题思路：下面的示例图给出了代码中第二种实现的示例。



```

1  # Definition for singly-linked list.
2  # class ListNode(object):
3  #     def __init__(self, x):
4  #         self.val = x
5  #         self.next = None
6
7  class Solution(object):
8      def oddEvenList1(self, head):
9          """
10             :type head: ListNode
11             :rtype: ListNode
12             """
13
14         if head is None: return head
15         odd = oddHead = head
16         even = evenHead = head.next
17         while even and even.next:
18             odd.next = even.next # 1->3
19             odd = odd.next # 3
20             even.next = odd.next # 2->4
21             even = even.next # 4
22             odd.next = evenHead # end->2
23             return oddHead
24
25     def oddEvenList2(self, head):
26         """

```

```

27     :type head: ListNode
28     :rtype: ListNode
29     """
30     if head is None or head.next is None: return head
31     p = head
32     tmphead = q = head.next
33     while p and q:
34         if q.next:
35             p.next = q.next
36             p = p.next
37         if q.next:
38             q.next = q.next.next
39             q = q.next
40     p.next = tmphead
41     return head

```

Listing 44: Problem328. Odd Even Linked List

### 1.2.2 leetcode 2. Add Two Numbers

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.

For example:

Input: (2 -> 4 -> 3) + (5 -> 6 -> 4)  
Output: 7 -> 0 -> 8

解题思路：这道题与leetcode 66. Plus One类似，差别在于后者是考查数组。有两点需要注意：一是进位的处理：需要一个额外的变量carry记录每次加法以后是否有进位；二是计算到其中一个链表的末尾时，如果有进位情况发生(carry=1)，那么要么累加到更长的那个链表上去，要么就创建一个新结点并添加到结果链表的尾部。下面的示例图中就是创建一个新的链表结点在结果链表的尾部。另外需要注意的一个技巧是在处理数组加法的题目是，可以先判断两个数组的size，这样可以固定size大的作为累加的结果；但是链表需要遍历一遍才能判断size。

```

1  # Definition for singly-linked list .
2  # class ListNode(object):
3  #     def __init__(self, x):
4  #         self.val = x
5  #         self.next = None
6
7  class Solution(object):
8      def addTwoNumbers(self, l1, l2): # RT: O(max{len(l1),len(l2)})
9          """
10          :type l1: ListNode
11          :type l2: ListNode
12          :rtype: ListNode
13          """
14          if l1 == None: return l2
15          if l2 == None: return l1
16
17          h1 = l1; h2 = l2
18          while h1!=None and h2!=None:
19              h1 = h1.next

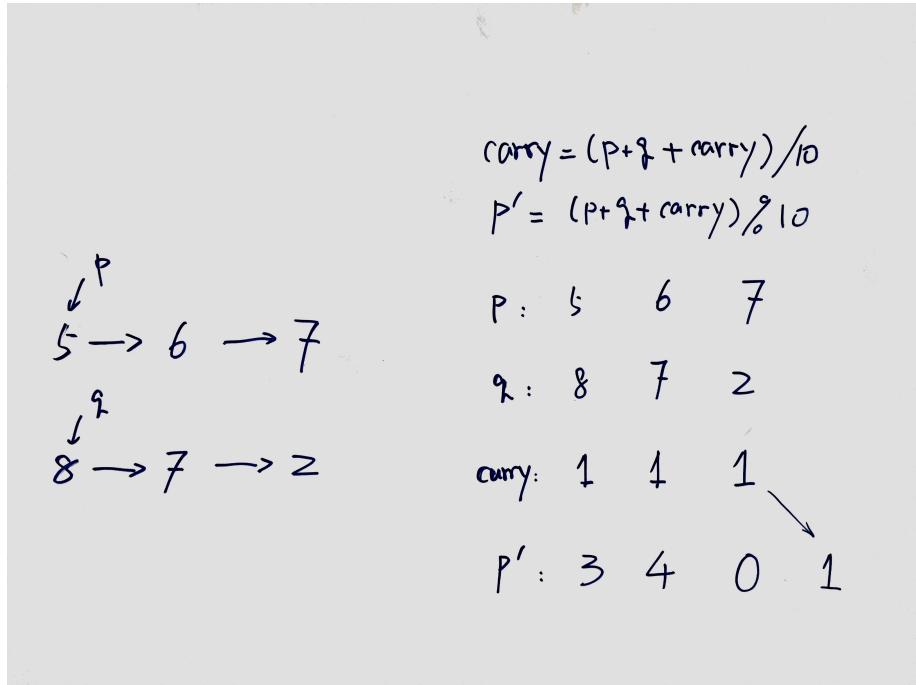
```

```

21         h2 = h2.next
22     if h2!=None: l1 ,l2 = l2 ,l1
23
24     head = l1
25     carry = 0
26     while l2!=None:
27         asum = l1.val + l2.val + carry
28         l1.val = asum%10
29         carry = asum/10
30         # l1 and l2 have same size
31         if l2.next is None and l1.next is None and carry==1:
32             l1.next = ListNode(carry)
33             carry = 0
34             l1 = l1.next
35             l2 = l2.next
36
37     while carry==1 and l1:
38         asum = carry + l1.val
39         l1.val = asum%10
40         carry = asum/10
41         if l1.next is None and carry==1:
42             l1.next = ListNode(carry)
43             break
44         l1 = l1.next
45     return head

```

Listing 45: Problem2. Add Two Numbers



### 1.2.3 leetcode 92. Reverse Linked List II

Reverse a linked list from position m to n. Do it in-place and in one-pass.

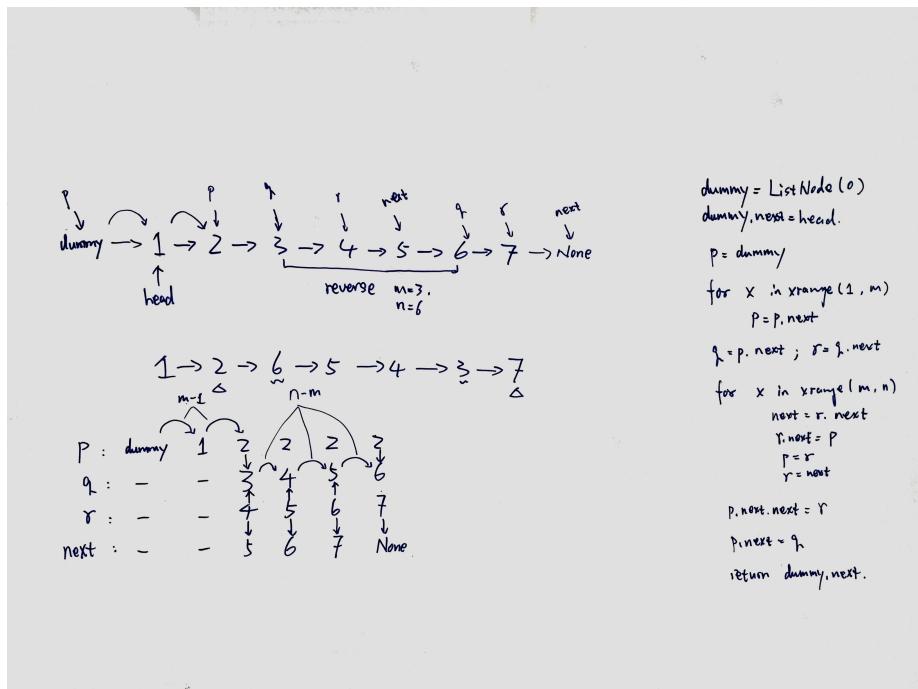
For example:

Given  $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow \text{NULL}$ ,  $m = 2$  and  $n = 4$ ,  
return  $1 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 5 \rightarrow \text{NULL}$ .

Note:

Given  $m, n$  satisfy the following condition:  $1 \leq m \leq n \leq \text{length of list}$ .

解题思路：反转链表考查的是对单链表指针的基本操作。这道题的特点是反转单链表的一段特定区间，需要首先找到这个区间的开始结点，并且记录这个区间开始结点的前驱结点，因为在反转结束后，需要链接两个部分，如示例图中所示： $p$ 记录了需要反转区间的开始结点的前驱节点，反转结束后，只需将 $p$ 指向反转区间新的开始结点就可以完成链接两个链表。另外需要掌握的技巧是，翻转单链表一共需要使用三个指针，如示例图中的 $q, r, next$ 指针， $q$ 和 $r$ 用于反转链的指向， $r$ 和 $next$ 用于记录下一组需要处理的结点。



```

1 def reverseBetween(self, head, m, n): # RT: O(n)
2     """
3         :type head: ListNode
4         :type m: int
5         :type n: int
6         :rtype: ListNode
7     """
8     def reverseBetween(self, head, m, n):
9         if head==None or head.next==None: return head
10        dummy = ListNode(0)
11        dummy.next = head
12        p = dummy
13        for x in xrange(1,m): # move m-1 steps forward
14            p = p.next
15

```

```

16     q = p.next # q is the m-th node
17     r = q.next
18     for x in xrange(m, n):
19         tmp = r.next
20         r.next = q
21         q = r
22         r = tmp
23     p.next.next = r
24     p.next = q
25     return dummy.next

```

Listing 46: Problem92. Reverse Linked List II

#### 1.2.4 leetcode 86. Partition List

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.

For example,

Given 1->4->3->2->5->2 and  $x = 3$ ,  
return 1->2->2->4->3->5.

```

1  # Definition for singly-linked list.
2  # class ListNode(object):
3  #     def __init__(self, x):
4  #         self.val = x
5  #         self.next = None
6
7  class Solution(object):
8      def partition(self, head, x):
9          """
10             :type head: ListNode
11             :type x: int
12             :rtype: ListNode
13             """
14
15         if head==None or head.next==None:
16             return head
17
18         curr = head
19
20         small_head = ListNode(0)
21         small_tail = small_head
22
23         large_head = ListNode(0)
24         large_tail = large_head
25
26         while curr != None:
27             while curr != None and curr.val < x:
28                 small_tail.next = curr
29                 small_tail = small_tail.next
30                 curr = curr.next
31             small_tail.next = None
32
33             while curr != None and curr.val >= x:
34                 large_tail.next = curr

```

```

35         large_tail = large_tail.next
36         curr = curr.next
37         large_tail.next = None
38
39     small_tail.next = large_head.next
40     return small_head.next

```

Listing 47: Problem86. Partition List

### 1.2.5 leetcode 83. Remove Duplicates from Sorted List

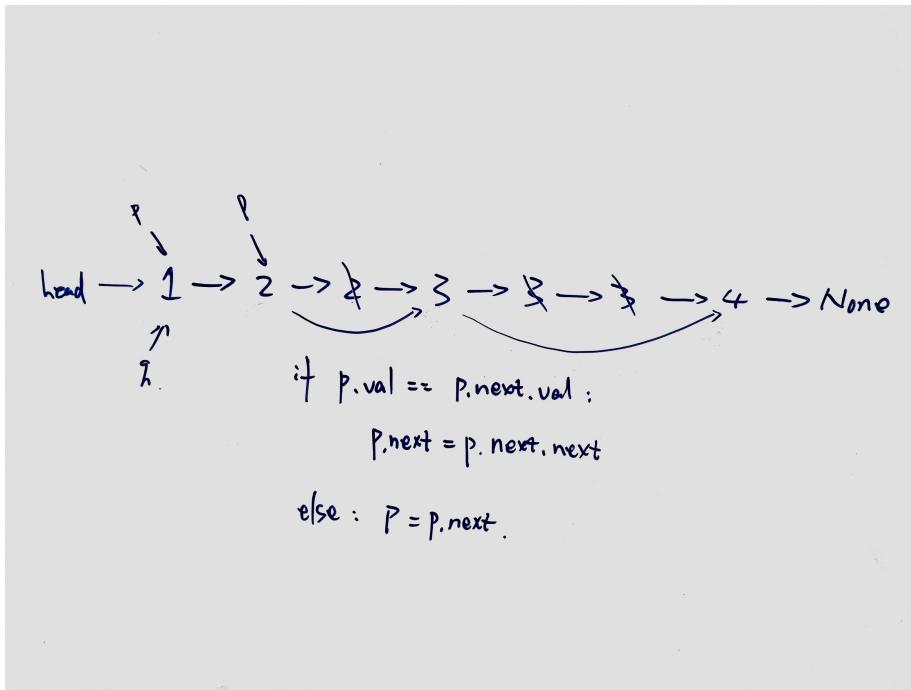
Given a sorted linked list, delete all duplicates such that each element appear only once.

For example,

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

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

解题思路：这道题的关键在于循环以及循环内部判断条件的设置。参考下面示例图。



```

1
2 # Definition for singly-linked list.
3 # class ListNode(object):
4 #     def __init__(self, x):
5 #         self.val = x
6 #         self.next = None
7
8 class Solution(object):
9     def deleteDuplicates(self, head):

```

```

10     """
11     :type head: ListNode
12     :rtype: ListNode
13     """
14     fast = head
15     while fast != None and fast.next != None:
16         if fast.val == fast.next.val:
17             curr = fast
18             while fast!=None and fast.val==curr.val:
19                 fast = fast.next
20                 curr.next = fast
21             else:
22                 fast = fast.next
23     return head

```

Listing 48: Problem83. Remove Duplicates from Sorted List

### 1.2.6 leetcode 82. Remove Duplicates from Sorted List II

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.

```

1
2 # Definition for singly-linked list.
3 # class ListNode(object):
4 #     def __init__(self, x):
5 #         self.val = x
6 #         self.next = None
7
8 class Solution(object):
9     def deleteDuplicates(self, head):
10         if head==None or head.next==None: return head
11         dummy = ListNode(0)
12         dummy.next = head
13         pre, curr, next = dummy, head, head.next
14         while next:
15             if curr.val != next.val:
16                 next = next.next
17                 curr = curr.next
18                 pre = pre.next
19             else:
20                 while next and next.val==curr.val:
21                     next = next.next
22                     curr = next
23                     pre.next = curr
24                     if next: next = next.next
25                     else: break
26     return dummy.next

```

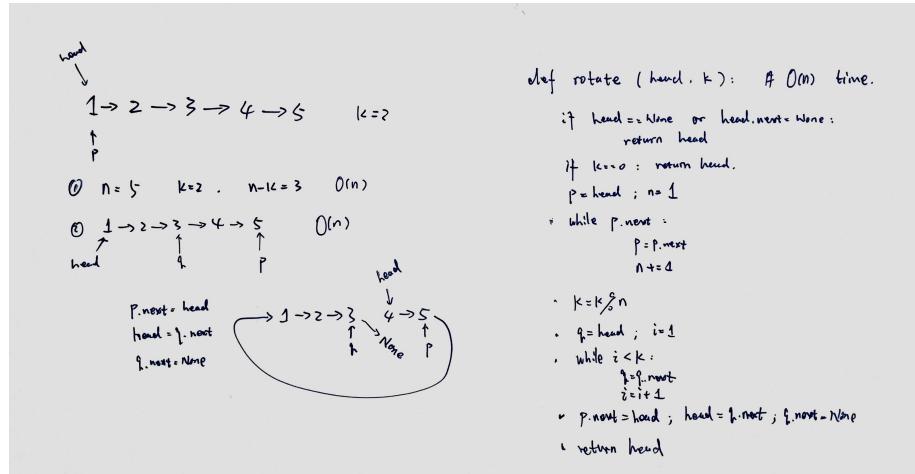
Listing 49: Problem82. Remove Duplicates from Sorted List II

### 1.2.7 leetcode 61. Rotate List

Given a list, rotate the list to the right by k places, where k is non-negative.

For example, given  $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow \text{NULL}$  and  $k = 2$ ,  
 return  $4 \rightarrow 5 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow \text{NULL}$ .

解题思路：因为第 $k$ 个位置是从右侧算的，所以需要定位到这个位置。计算节点总数 $n$ 后即可求得。此外， $k$ 可能大于链中节点的总数 $n$ ，所以在计算节点总数之后要进行处理： $k = k \% n$ 。参考下面示例图。



```

1  # Definition for singly-linked list.
2  # class ListNode(object):
3  #     def __init__(self, x):
4  #         self.val = x
5  #         self.next = None
6
7
8  class Solution(object):
9      def rotateRight(self, head, k): # RT: O(n)
10         """
11             :type head: ListNode
12             :type k: int
13             :rtype: ListNode
14         """
15         if head==None or head.next==None:
16             return head
17
18         tail = head
19         n = 1
20         # compute the length of the linked list
21         while tail.next!=None:
22             tail = tail.next
23             n += 1
24         tail.next = head
25         p = tail
26
27         k = k % n # k could be larger than n
28         for i in range(n-k):
29             p = p.next
30         head = p.next
31         p.next = None
32

```

33      **return** head

Listing 50: Problem61. Rotate List

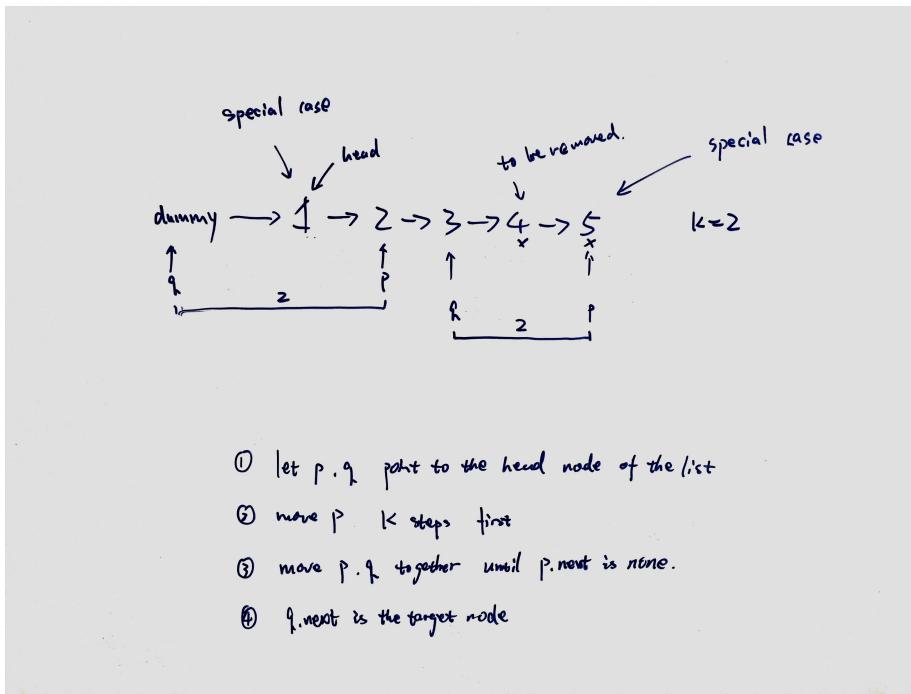
### 1.2.8 leetcode 19. Remove $N^{th}$ Node From End of List

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.

解题思路：这道题有两种思路：第一种思路是先遍历一遍链表得到链表中结点的个数 $m$ ；然后再从头开始遍历链表，但是只向后移动 $m - n$ 步；最后删除目标结点。因此，第一种思路需要遍历两次链表。第二种思路是窗口法，窗口大小为 $n + 1$ ，窗口左侧的边界即为目标结点的前驱结点。这种方法只需要遍历一遍链表，但是需要增加一个额外的头结点，以便处理删除头结点的边界情况。根据OJ的跑分结果，第二种方法更高效。下面的示例图描述了第二种方法。



- ① let  $p, q$  point to the head node of the list
- ② move  $p$   $k$  steps first
- ③ move  $p, q$  together until  $p.next$  is `None`.
- ④  $q.next$  is the target node

```
1
2 # Definition for singly-linked list.
3 # class ListNode(object):
4 #     def __init__(self, x):
5 #         self.val = x
6 #         self.next = None
7
8 class Solution(object):
9     def removeNthFromEnd(self, head, n): # RT: O(n), space: O(1)
10        """
```

```

11     :type head: ListNode
12     :type n: int
13     :rtype: ListNode
14     """
15     if head==None or (head.next==None and n==1): return None
16     dummy = ListNode(0)
17     dummy.next = head
18     p = q = dummy
19     for i in xrange(n): p = p.next
20     while p.next!=None:
21         p = p.next; q = q.next
22     q.next = q.next.next
23     return dummy.next

```

Listing 51: Problem19. Remove  $N^{th}$  Node From End of List

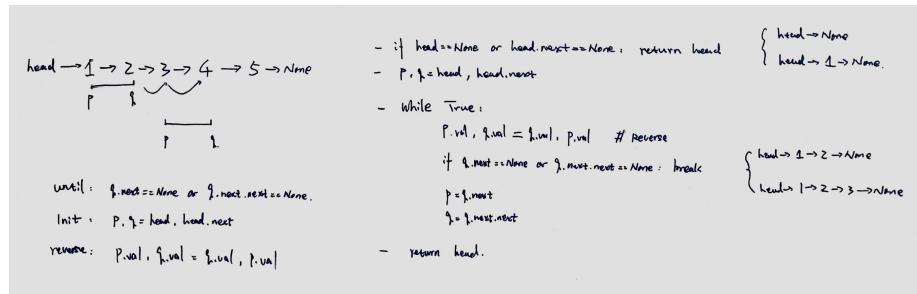
### 1.2.9 leetcode 24. Swap Nodes in Pairs

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.

Your algorithm should use only constant space. You may not modify the values in the list, only nodes itself can be changed.

解题思路：这道题可以使用窗口法，窗口的大小就是2，即窗口包含两个结点。窗口每次移动两个结点的距离。参考下面的示例图。



```

1
2 # Definition for singly-linked list.
3 # class ListNode(object):
4 #     def __init__(self, x):
5 #         self.val = x
6 #         self.next = None
7
8 class Solution(object):
9     def swapPairs(self, head):
10         """
11             :type head: ListNode
12             :rtype: ListNode
13         """
14         if head==None or head.next==None: return head
15         p, q = head, head.next
16         while True:
17             p.val, q.val = q.val, p.val
18             if q.next==None or q.next.next==None: break

```

```

19         p = q.next
20         q = p.next
21     return head

```

Listing 52: Problem24. Swap Nodes in Pairs

### 1.2.10 leetcode 25. Reverse Nodes in k-Group

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

1. For  $k = 2$ , you should return: 2->1->4->3->5
2. For  $k = 3$ , you should return: 3->2->1->4->5

解题思路：这道题的解题过程包括两个主要部分：

- 首先，要确定一个k-group的起止结点khead和ktail，这是反转的区间；同时，还需要记录起点khead的前驱结点p和终点ktail的后继结点c，在将反转后的区间插入到原链表的时候要用到这两个结点。
- 其次是翻转khead和ktail之间的结点。这个部分和之前的一些链表反转的题目的解法一样，通常需要两个指针pre和curr指向准备反转的两个相邻结点，同时用一个临时变量tmp记录curr的下一个结点，以便翻转结束后，将pre和curr移动到下一对儿准备反转的结点。

上述过程可以参见下面的示例图。另外，需要注意的一点是不足 $k$ 长的部分无需翻转！

```

1
2 # Definition for singly-linked list.
3 # class ListNode(object):
4 #     def __init__(self, x):
5 #         self.val = x
6 #         self.next = None
7
8 def reverseKGroup(self, head, k):
9     """
10     :type head: ListNode
11     :type k: int
12     :rtype: ListNode
13     """
14     if head==None or k==1: return head
15
16     dummy = ListNode(0)
17     dummy.next = head
18     p = dummy
19     c = dummy.next
20     while c!=None:
21         khead = c
22         # check if there is a k-length group
23         for i in range(k-1):
24             c = c.next
25             # if no k-length group, do nothing!
26             if c==None: return dummy.next

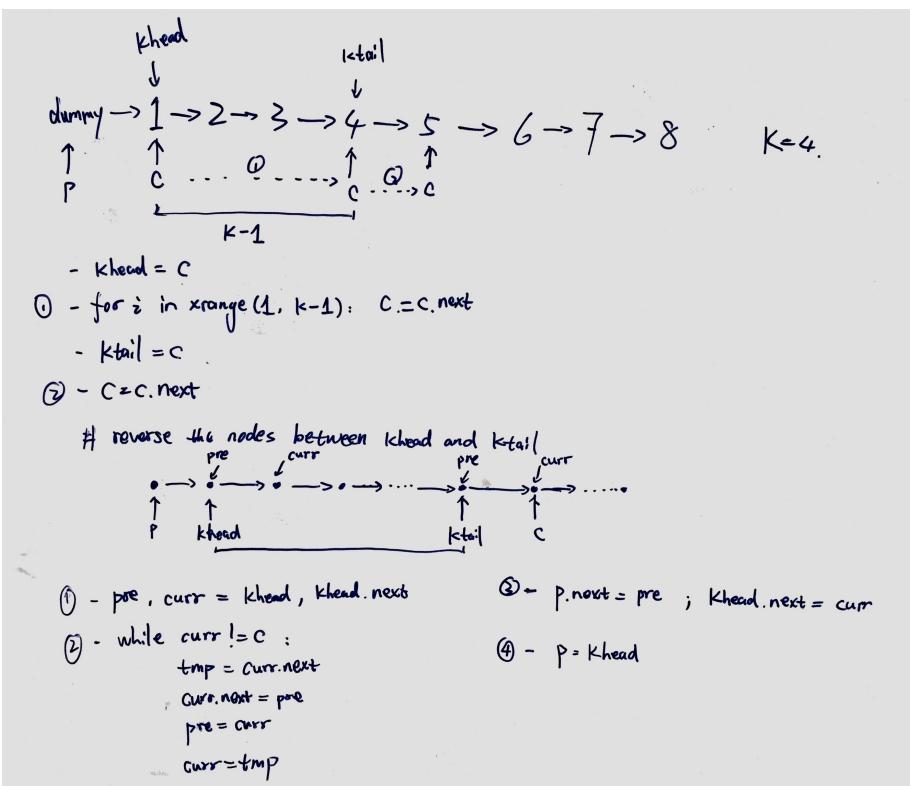
```

```

27     ktail = c
28     c = c.next
29
30     # reverse the part from khead to ktail
31     pre, curr = khead, khead.next
32     while curr!=c:
33         tmp = curr.next
34         curr.next = pre
35         pre = curr
36         curr = tmp
37     khead.next = c
38     p.next = ktail
39     p = khead
40
41 return dummy.next

```

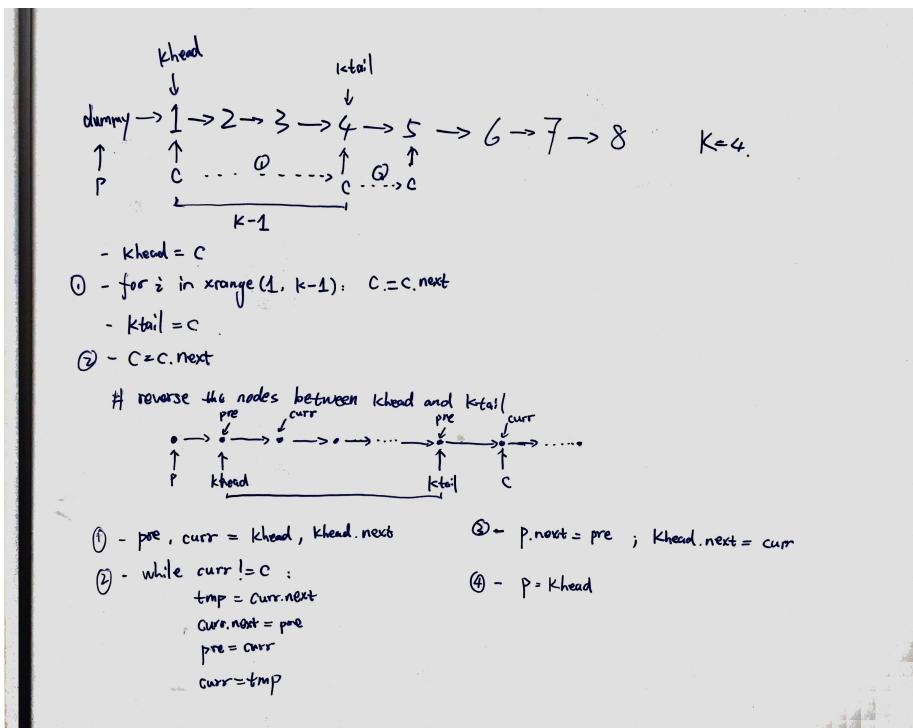
Listing 53: Problem25. Reverse Nodes in k-Group



### 1.2.11 leetcode 138. Copy List with Random Pointer

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.

解题思路：这题的难点在于如何处理random指针。方法是为链表中的每一个结点创建一个副本，并将副本插入到每一个原始结点的后面；遍历更新后的链表，让p指向每一个原始结点，使得 $p.next.random = p.random.next$ 。完成所有副本结点的random指针赋值后，再分离出深度复制后的链表。



```

1
2 # Definition for singly-linked list with a random pointer.
3 # class RandomListNode(object):
4 #     def __init__(self, x):
5 #         self.label = x
6 #         self.next = None
7 #         self.random = None
8
9 class Solution(object):
10    def copyRandomList(self, head): # RT: O(n)
11        """
12            :type head: RandomListNode
13            :rtype: RandomListNode
14        """
15        if head==None: return head
16
17        # copy every existed node in the original
18        # list and insert the copy node into the list
19        # just next to the original node.
20        # before: 1->2->3->4
21        # after: 1->1->2->2->3->4->4
22        p = head
23        while p!=None:
24            node = RandomListNode(p.label)
25            tmp = p.next
26            node.next = tmp
27            p.next = node
28            p = node.next
29
30        # deal with the random pointers
31        p = head
32        while p!=None:

```

```

33     if p.random!=None:
34         p.next.random = p.random.next
35     p = p.next.next
36
37     # separate the copy list from the
38     # the hybrid list , and recover the
39     # original list
40     dummy = RandomListNode(0)
41     dummy.next = head.next
42     q = dummy
43     p = head
44     while p!=None:
45         q.next = p.next
46         p.next = p.next.next
47         q = q.next
48         p = p.next
49
50     return dummy.next

```

Listing 54: Problem138. Copy List with Random Pointer

### 1.2.12 leetcode 141. Linked List Cycle

Given a linked list, determine if it has a cycle in it.

Follow up: Can you solve it without using extra space?

```

1  # Definition for singly-linked list .
2  # class ListNode(object):
3  #     def __init__(self, x):
4  #         self.val = x
5  #         self.next = None
6
7
8  class Solution(object):
9      def hasCycle(self, head):
10         """
11             :type head: ListNode
12             :rtype: bool
13         """
14         def hasCycle(self, head):
15             if head==None or head.next==None: return False
16             slow = fast = head
17             while fast.next and fast.next.next:
18                 fast = fast.next.next
19                 slow = slow.next
20                 if fast == slow: return True
21         return False

```

Listing 55: Problem141. Linked List Cycle

### 1.2.13 leetcode 142. Linked List Cycle II

Given a linked list, return the node where the cycle begins. If there is no cycle, return null.

Note: Do not modify the linked list.

Follow up: Can you solve it without using extra space?

```
1 # Definition for singly-linked list.
2 # class ListNode(object):
3 #     def __init__(self, x):
4 #         self.val = x
5 #         self.next = None
6 #
7 class Solution(object):
8     def detectCycle(self, head):
9         """
10         :type head: ListNode
11         :rtype: ListNode
12         """
13
14         if head==None or head.next==None:
15             return None
16
17         # First, check if a cycle exists in
18         # the list.
19         p = head.next.next
20         q = head.next
21         while p!=None and p!=q:
22             if p.next!=None and p.next!=None:
23                 p = p.next.next
24                 q = q.next
25             else:
26                 return None
27
28         if p==None: return None
29         # Second, when p==q, the cycle exists,
30         # and from the node where p meets q, move
31         # pointer r from the beginning of the list
32         # with the same step as q. when r,q meet
33         # at some node, the node is just the result.
34         r = head
35         while r!=q:
36             r = r.next
37             q = q.next
38
39         return r
```

Listing 56: Problem142. Linked List Cycle II

#### 1.2.14 leetcode 143. Reorder List

Given a singly linked list  $L : L_0 \rightarrow L_1 \rightarrow \dots \rightarrow L_{n-1} \rightarrow L_n$ , reorder it to:  $L_0 \rightarrow L_n \rightarrow L_1 \rightarrow L_{n-1} \rightarrow L_2 \rightarrow L_{n-2} \rightarrow \dots$

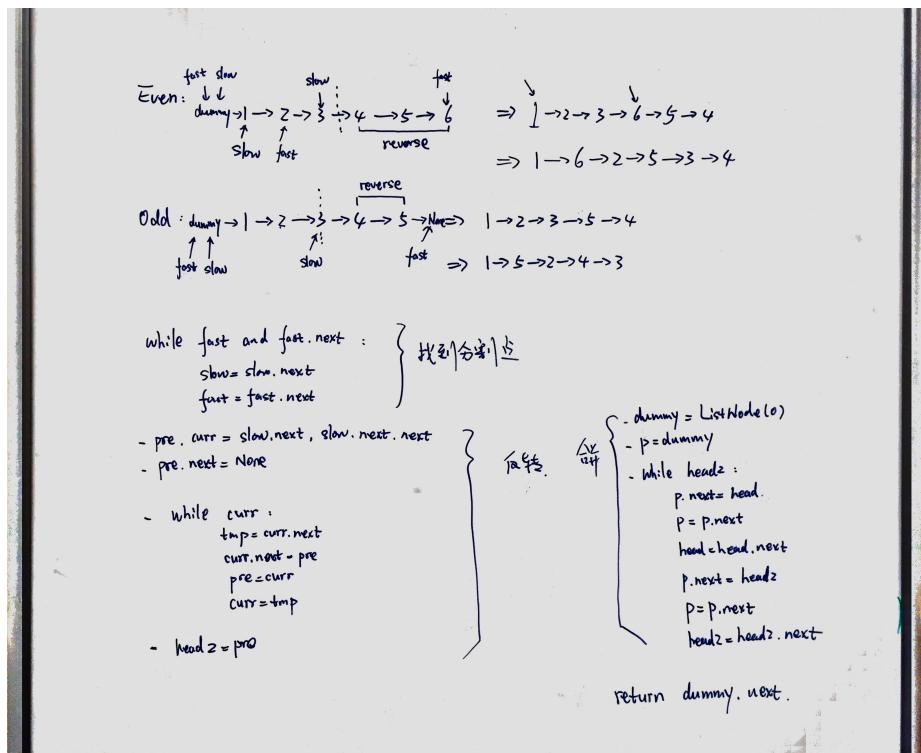
You must do this in-place without altering the nodes' values.

For example, Given 1,2,3,4, reorder it to 1,4,2,3.

解题思路：参见示例图，题目的解法分为三个部分：

- 首先，找到分割点，在分割点之前（包括分割点）为第一部分，分割点之后为第二部分。考虑到原链表中结点的数目可能为奇数也可能为偶数，所以第二部分所含结点的个数不多于第一部分。如示例图所示。

- 其次，反转第二部分的链表结点。
- 最后，合并第一、二两部分的结点，合并的原则是每一轮在第一、二部分各取一个结点加入，直到第二个部分的所有结点都已经插入。最后，如果第一部分还有结点未插入，那么插入所有剩余结点。



```

1 # Definition for singly-linked list.
2 # class ListNode(object):
3 #     def __init__(self, x):
4 #         self.val = x
5 #         self.next = None
6
7
8 class Solution(object):
9     def reorderList(self, head):
10         """
11             :type head: ListNode
12             :rtype: void Do not return anything, modify head in-place
13             instead.
14         """
15         if head==None or head.next==None: return
16         dummy = ListNode(0)
17         dummy.next = head
18         # divide the original list into two parts
19         slow = fast = dummy
20         while fast.next and fast.next.next:
21             fast = fast.next.next
22             slow = slow.next
23         if fast.next==None: # has even nodes

```

```

23     head2 = slow.next
24     tail1 = slow
25     elif fast.next.next==None: # has odd nodes
26         head2 = slow.next.next
27         tail1 = slow.next
28         tail1.next = None
29
30     # reverse the second half part
31     pre, curr = head2, head2.next
32     while curr:
33         tmp = curr.next
34         curr.next = pre
35         pre = curr
36         curr = tmp
37     head2.next = None
38     head2 = pre
39
40     # merge two parts
41     p = head
42     while head2:
43         tmp = head2.next
44         head2.next = p.next
45         p.next = head2
46         p = head2.next
47         head2 = tmp

```

Listing 57: Problem143. Reorder List

## 2 Strings

总结: 与字符串相关的算法题目

1. 字符串匹配类问题: 目前流行的字符串匹配算法有以下七种:

- Brute-force search (暴力法)  $O(m \times n)$
- KMP算法 $O(m + n)$
- Z算法 $O(m + n)$
- Robin-Karp算法: It uses hashing to find any one of a set of pattern strings in a text. Its average and best case running time is  $O(n + m)$  in space  $O(m)$ , but its worst-case time is  $O(m \times n)$ .
- Boyer-Moore算法
- Aho-Corasick算法: worst-time complexity  $O(m + n)$  in space  $O(m)$ . It is a kind of dictionary-matching algorithm that locates elements of a finite set of strings (the "dictionary") within an input text. It matches all strings simultaneously.
- 有限状态机

2. Palindrome类问题

- DP-I算法:  $O(n^2)$  time in  $O(n^2)$  space
- DP-II算法:  $O(n^2)$  time in  $O(1)$  space
- Manacher's algorithm:  $O(n)$  time in  $O(n)$  space

## 2.1 leetcode 65. Valid Number

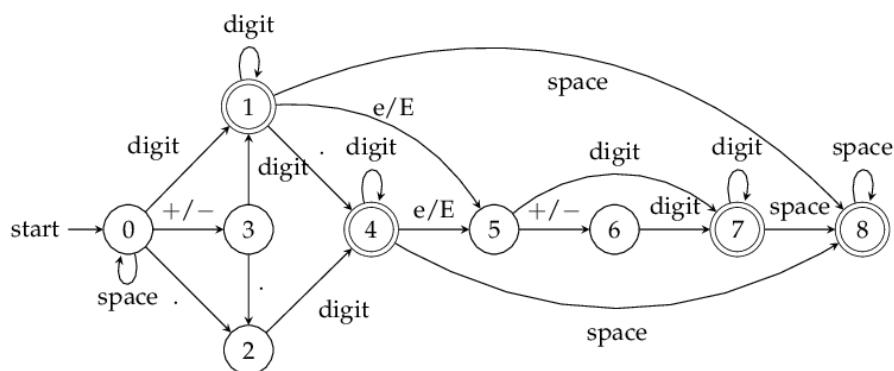
Validate if a given string is numeric. Some examples:

1. "0" => true
2. "0.1" => true
3. "abc" => false
4. "1a" => false
5. "2e10" => true

Note: It is intended for the problem statement to be ambiguous. You should gather all requirements up front before implementing one.

解题思路：建立状态机，如下图所示。这个题有9种状态：

- 0: 初始无输入或者只有space的状态
- 1: 输入了数字之后的状态
- 2: 前面无数字，只输入了dot的状态
- 3: 输入了符号状态
- 4: 前面有数字和有dot的状态
- 5: 'e' or 'E'输入后的状态
- 6: 输入e之后输入Sign的状态
- 7: 输入e后输入数字的状态
- 8: 前面有有效数输入之后，输入space的状态



```
1 class Solution(object):
2     def isNumber(self, s):
3         """
4             :type s: str
5             :rtype: bool
6         """
7         INVALID=0; SPACE=1; SIGN=2; DIGIT=3; DOT=4; EXPONENT=5;
```

```

9      #0invalid ,1space ,2sign ,3digit ,4dot ,5exponent ,6num_inputs
10     transitionTable=[]
11         [-1, 0, 3, 1, 2, -1],      #0 no input or just spaces
12         [-1, 8, -1, 1, 4, 5],      #1 input is digits
13         [-1, -1, -1, 4, -1, -1],   #2 no digits in front just
14             Dot
15         [-1, -1, -1, 1, 2, -1],    #3 sign
16         [-1, 8, -1, 4, -1, 5],    #4 digits and dot in front
17         [-1, -1, 6, 7, -1, -1],   #5 input 'e' or 'E'
18         [-1, -1, -1, 7, -1, -1],  #6 after 'e' input sign
19         [-1, 8, -1, 7, -1, -1],   #7 after 'e' input digits
20         [-1, 8, -1, -1, -1]]      #8 after valid input input
21             space
22         state=0; i=0
23         while i<len(s):
24             inputtype = INVALID
25             if s[i]==',': inputtype=SPACE
26             elif s[i]=='-' or s[i]=='+': inputtype=SIGN
27             elif s[i] in '0123456789': inputtype=DIGIT
28             elif s[i]=='.': inputtype=DOT
29             elif s[i]=='e' or s[i]=='E': inputtype=EXPONENT
30
31             state=transitionTable[state][inputtype]
32             if state== -1: return False
33             else: i+=1
34         return state == 1 or state == 4 or state == 7 or state == 8

```

Listing 58: Problem65. Valid Number

## 2.2 leetcode 125. Valid Palindrome

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

- "A man, a plan, a canal: Panama" is a palindrome.
- "race a car" is not a palindrome.

Note: Have you consider that the string might be empty? This is a good question to ask during an interview. For the purpose of this problem, we define empty string as valid palindrome.

解题思路：这道题的算法设计并不难，而且也是面试中常常会被问及的问题。但是，这道题的考查点是在设计算法之前，是否问清楚了题目中的一些潜在的（或者说没有在题目中明确说明的）需求。比如，目标字符串到底是什么样子的，题目下面有提示说字符串为空时的情形。题目也给出只考虑英文字符和数字，也暗示字符串本身可能包括其他字符，那么就需要在验证palindrome属性之前，对字符串先行处理一下；去掉不符合要求的字符以后才能进行比较。这条信息在题目中已经给出，但是在面试的时候，面试官通常并不会像题目中那样给出这些明确的或者暗示性的信息，因此要特别注意所有题目中给出的各种前提条件。在面试过程中，如果碰到相似的题目时，必须在设计算法前，首先确定题目的前提条件是否一致。如果前提条件是不一样的，那么即使做过类似的题目，设计出来的算法也可能有很大的不同。即使是完全相同的题目，如果在没有询问面试官确定这些前提条件的话，即使设计出了正确的算法，也是失败的过程。

```

1
2 class Solution(object):
3     def isPalindrome1(self, s): # RT: O(n), Space: O(n)
4         """
5             :type s: str
6             :rtype: bool
7         """
8         if s==None: return False
9         if len(s)<=1: return True
10
11        # only extract alphanumeric characters
12        str=[c for c in s.lower() if 97<=ord(c)<=122 or 48<=ord(c)
13        <=57]
14        return str==str[::-1]

```

Listing 59: Problem125. Valid Palindrome

### 2.3 leetcode 28. Implement strStr()

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

解题思路：这道题实际上就是在文本中搜索指定的字符串。至少有四种解决方法方法。为了描述方便，假设heystack就是文本text，长度为 $m$ ，needle就是pattern，即要搜索的字符串，长度为 $n$ 。解决方法如下：

1. 暴力破解法：这种方法就是先在text中找到与pattern的首字符匹配的字符位置；然后在依次验证后面相应的字符是否每一个都匹配。如果匹配，就返回结果；如果出现不匹配的字符，就重新开始整个过程。这种方法的时间复杂度是 $O(m \times n)$ ，空间复杂度是 $O(n)$ 。
2. KMP算法：最著名的字符串匹配算法，算法时间复杂度为 $O(m + n)$ 。该算法分为两个阶段（参考下面的示例图一）：
  - 第一阶段的任务是遍历pattern建立T表。这个T表的作用是在第二阶段的匹配过程中，如果text[i]≠pattern[j]，那么就验证pattern[T[j-1]]与text[i]是否匹配。建立T表的时间上限即为第一阶段的时间复杂度 $O(n)$
  - 第二阶段的任务就是利用第一阶段建立的T表搜索是否有匹配的情况。这个阶段的时间复杂度是 $O(m)$ 。
3. Z-算法：参考下面的示例图二中的描述和演算过程。这个算法的时间复杂度是 $O(m + n)$ ，空间复杂度是 $O(m + n)$ 。
4. Rabin-Karp算法：RK算法的时间复杂度是 $O(m \times n)$ ，空间复杂度是 $O(1)$ 。这个算法最适合在一个text中搜索多个pattern的情形。另外，这个算法的特点是利用哈希函数来确定潜在的匹配项，然后再执行精确匹配的验证。哈希函数和演算的细节参考示例图三。

这道题仅要求判断是否存在匹配。可以进一步扩展为输出所有匹配在text中的起始位置索引。

```

1
2 class Solution(object):
3     def strStr_bruteforce(self, haystack, needle): # RT: O(mn)
4         """

```

I : Populate T

KMP Substring Search

- $O(m+n)$  time
- $O(n)$  space.

P:  $\begin{matrix} a & b & c & a & b & y \end{matrix} \rightarrow n$

T:  $\boxed{0 \ 0 \ 0 \ 1 \ 2 \ 0}$

while  $i < \text{len}(P)$ :

① if  $P[i] == P[i]$  :  
 $T[i] = j+1$   
 $i+1, j+1$

② if  $P[i] != P[i]$  :  
if  $j=0$  :  $T[i]=0$ ;  $i+1$   
else:  $j = T[i-1]$

II. Search.

Text:  $\begin{matrix} a & b & x & a & b & c & a & b & c & a & b & y \end{matrix} \rightarrow m$

pattern:  $\begin{matrix} a & b & c & a & b & y \end{matrix} \rightarrow n$

$\begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \end{matrix}$

$\begin{matrix} ; & & & & & & & & & & & \end{matrix}$

①  $\text{Text}[i] \neq \text{Pattern}[i]$  :  $j = T[i-1] = T[1] = 0$

②  $\text{Text}[i] \neq \text{Pattern}[0]$  :  $\because j=0 \therefore i=i+1=3$

⋮

③  $\text{Text}[8] \neq \text{Pattern}[5]$  :  $j = T[7-1] = T[4] = 2$

④  $\text{Text}[8] = \text{Pattern}[2]$  :  $i = i+1; j = j+1$

} match.

if  $\text{Text}[i] == \text{Pattern}[i]$  :  
 $i = i+1; j = j+1$

else:  
if  $j > 0$  :  $j = T[i-1]$   
else:  $i = i+1$

```

5      :type haystack: str
6      :type needle: str
7      :rtype: int
8      """
9      m, n = len(haystack), len(needle)
10     if n==0: return 0
11     if m<n: return -1
12
13     # the range is the most critical part for the performance
14     for i in range(m-n+1):
15         if haystack[i:i+n]==needle:
16             return i
17

```

Listing 60: Problem28. Implement strStr()

## 2.4 leetcode 8. String to Integer (atoi)

Implement atoi to convert a string to an integer.

text: x a b c a b y a b c  
 pattern: a b c → n=3

## Z-alg. Pattern Matching

## II. How to compute Z.

S:  $\frac{a^2 b \times a a b \times c a a b \times a a b \times a y}{o \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12 \ 13 \ 14 \ 15 \ 16 \ 17 \ 18}$   
 Prefix 1 r  
 Z: 0 1 0 0 4 1 0 0  
 copy  
 K: 从 1 开始;  $\frac{1}{2}$  总是 从 0 开始.

- $\Sigma(k) = l : \Sigma[i=0:l] == S[k:k+l]$ , and  $\Sigma[i=l] != \Sigma[k+l]$
  - $S[5..6..7..]$ 直接 copy  $S[1..2..3]$ , 因为  $S[4..7] = S[0..3]$  且
  - ※  $l$  和  $r$  表示一个窗口, 如果(窗口内的元素的索引值) + (在 prefix 中匹配过的字符数) 不大于窗口右边界对应的索引值, 那么  $\Sigma(j) = \Sigma(i)$ ; 如果超出右边界了, 则将左边界  $l$  设置在  $j$  处, 重

Notes: It is intended for this problem to be specified vaguely (ie, no given input specs). You are responsible to gather all the input requirements up front.

Requirements for atoi:

- The function first discards as many whitespace characters as necessary until the first non-whitespace character is found. Then, starting from this character, takes an optional initial plus or minus sign followed by as many numerical digits as possible, and interprets them as a numerical value.
  - The string can contain additional characters after those that form the integral number, which are ignored and have no effect on the behavior of this function.
  - If the first sequence of non-whitespace characters in str is not a valid integral number, or if no such sequence exists because either str is empty or it contains only whitespace characters, no conversion is performed.
  - If no valid conversion could be performed, a zero value is returned. If the correct value is out of the range of representable values, 2147483647 or -2147483648 is returned.

Rabin-Karp Alg.

I. hash function

- $\text{prime} = 3$  (也可以是其他素数)
- $X = \text{old\_hash} - \text{val}(\text{old\_char})$
- $X = X / \text{prime}$
- $\text{new\_hash} = X + \text{val}(\text{new\_char}) * \text{prime}^{m-1}$

$m$  is the length of pattern.

Ex. pattern: abc ( $m=3$ )  $\text{hash}(abc) = 34$

text: a b e d a b c

①  $\text{hash}(abe) = 1 + 2 \times 3^1 + 5 \times 3^2 = 52 \neq 34$

② rehash(bed) =  $(52 - 1) / 3 + 4 \times 3^1 = 53 \neq 34$   
oldhash ↑ old\_char      ↓ new\_char

③ rehash(edc) =  $(53 - 2) / 3 + 1 \times 3^1 = 26 \neq 34$

④ rehash(dab) =  $(26 - 5) / 3 + 2 \times 3^1 = 25 \neq 34$

⑤ rehash(abc) =  $(25 - 4) / 3 + 3 \times 3^1 = 34 = 34$

if  $\text{rehash}(abc) == \text{hash}(abc)$ :

if pattern == text[i:m]:  
 res.append(i)

$O(mn)$  time  
 $O(1)$  space  
 ↳  $\begin{cases} \text{pattern1} = xy \xrightarrow{\text{hash}} x_1 \\ \text{pattern2} = abc \xrightarrow{\text{hash}} x_2 \\ \text{pattern3} = ade \xrightarrow{\text{hash}} x_3 \end{cases}$   
 用一个hash下， $x_1, x_2, x_3$ 三个值不同，适合于在同一text下搜索多个pattern.

$\text{val}(\text{char}): \text{int}$   
 a → 1  
 b → 2  
 c → 3  
 d → 4  
 e → 5  
 ...  
 z → 26

解题思路：需要特别注意这道题！！！题目的难度不大，但是需要考虑很多边界情况和特殊情况。需要注意以下几点：

- 32位整数的上下界:  $-2147483648 \sim 2147483647$ ;
- 字符串长度为0时应该返回什么值。
- 正负号的处理；
- 字符串前面(leading)和后面(trailing)的空白字符；
- 除了字符串开头的正负号以外外，其他的字符对应的ascii应该在48~57范围内。

```

1 class Solution(object):
2     def myAtoi(self, str):
3         """
4             :type str: str
5             :rtype: int
6             """
7         if len(str)==0: return 0
8         s = str.strip()
9         neg = 0 # 0: positive, 1: negative
10

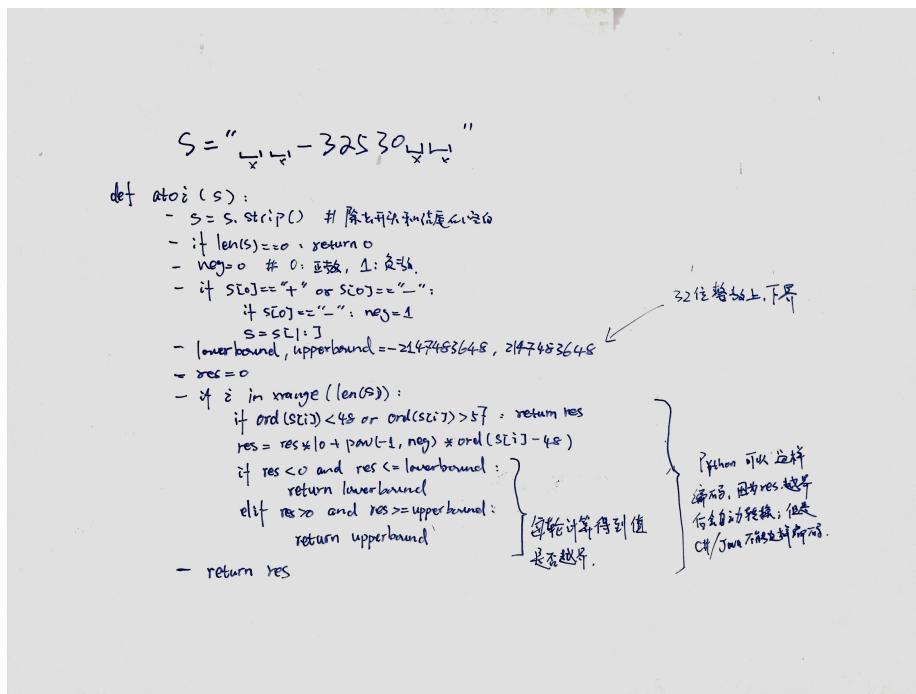
```

```

11     if s[0]=="+" or s[0]=="-":
12         if s[0]=="-": neg=1
13         s=s[1:]
14     lowerbound, upperbound = -2147483648, 2147483647
15     res = 0
16     for i in xrange(len(s)):
17         if ord(s[i])<48 or ord(s[i])>57: return res
18         res = res*10 + pow(-1,neg)*(ord(s[i])-48)
19         if res<0 and res<=lowerbound: return lowerbound
20         elif res>0 and res>=upperbound: return upperbound
21     return res

```

Listing 61: Problem8. String to Integer (atoi)



## 2.5 leetcode 67. Add Binary

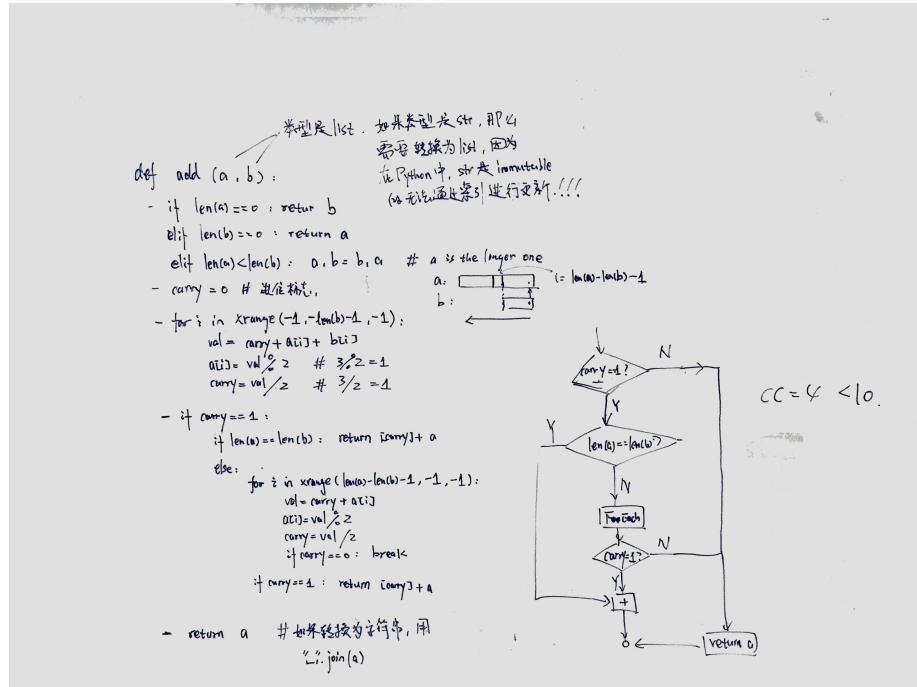
Given two binary strings, return their sum (also a binary string). For example,  $a = "11"$ ,  $b = "1"$ , return  $"100"$ .

解题思路：这道题不难，但是用python实现的时候需要注意string类型的数据是没法通过索引进行更新的，即无法完成 $a[i] = "x"$ 这样的赋值语句。如果想操作数据，就需要先转换成list类型，在通过索引更新。注意，为了方便，示例图中的代码， $a$ 和 $b$ 是list类型，并且list中的是数值而不是字符。

```

1
2 class Solution(object):
3     def addBinary(self, a, b):
4         """
5             :type a: str
6             :type b: str

```



```

7     :rtype: str
8     """
9     if a=="": return b
10    elif b=="": return a
11
12    if len(a)<len(b): a,b=b,a
13
14    # In python, have to use list to
15    # convert integer to str and back.
16    alist=list(a)
17    blist=list(b)
18    carry=0
19    for i in range(-1, -len(blist)-1, -1):
20        if alist[i]=='1' and blist[i]=='1':
21            alist[i]=str(carry)
22            carry=1
23        elif alist[i]=='0' and blist[i]=='0':
24            alist[i]=str(carry)
25            carry=0
26        else: # (a[i]=='1' and b[i]=='0') or (a[i]=='0' and b[i]
27            alist[i]= str((1+carry)%2)
28            carry=(1+carry)/2 # Bug: carry=0
29
30    j=-len(blist)-1
31    while carry==1 and j>=-len(alist):
32        if alist[j]=='1':
33            alist[j]=str(0)
34            carry=1
35            j-=1
36        else:
37            alist[j]=str(1)
38            carry==0

```

```

39         break
40
41     if carry==1 and j<-len(alist):
42         return " ".join(["1"] + alist)
43     else:
44         return " ".join(alist)

```

Listing 62: Problem67. Add Binary

## 2.6 leetcode 5. Longest Palindromic Substring

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.

有两个帖子对这个问题的解法有很好的讨论: <http://articles.leetcode.com/longest-palindromic-substring-part-i> 和<http://articles.leetcode.com/longest-palindromic-substring-part-ii>。

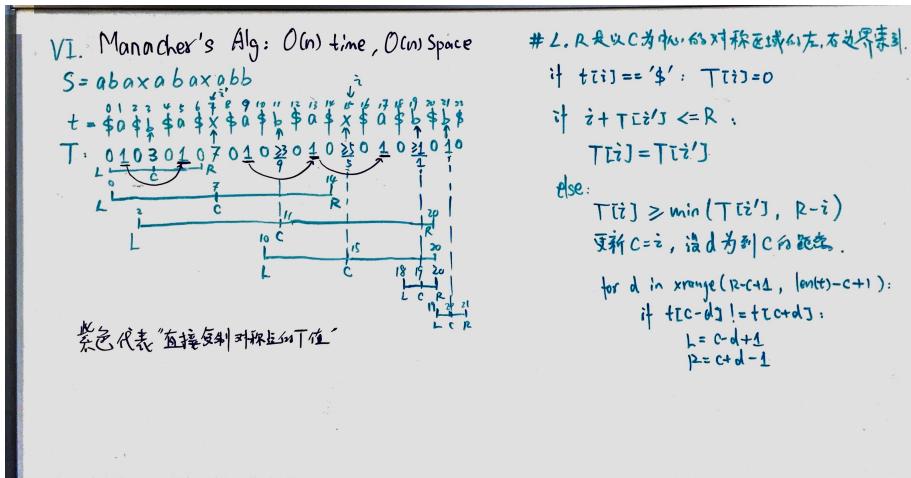
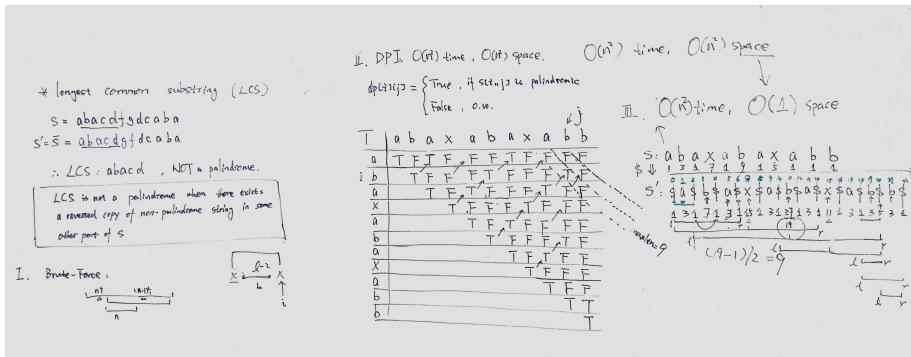
这个题共计有四种解法:

- 第一种解法: brute-force暴力算法, 时间复杂度为 $O(n^3)$ 。每次在n长的字符串中选择两个字符作为子串的起点和终点, 共有 $\frac{n(n-1)}{2}$ 种选择, 而检查每种选择的子串是否为palindrome需要 $O(n)$ 的时间复杂度, 所以整个算法的时间复杂度为 $O(n^3)$ 。
- 第二种解法: dynamic programming, 时间复杂度为 $O(n^2)$ , 空间复杂度为 $O(n^2)$ 。dp[i][j]表示索引i和j之间的子串是否为回文。初始化矩阵时, 需要先判断长度为1和2的子串。然后从长度为3开始, 最大到n, 按照如下算法进行判断当前长度的子串是否为回文: 对于子串 $s_i, s_{i+1}, \dots, s_{j-1}, s_j$ 。如果 $s_{i+1}, \dots, s_{j-1}$ 是回文, 并且 $s_i == s_j$ , 那么字符串 $s_i, \dots, s_j$ 就是回文。
- 第三种解法: 在第二种算法的基础上, 减少空间复杂度至 $O(1)$ 。时间复杂度为 $O(n^2)$ 。算法的思路是每个回文都是从中间元素为中心两侧对称的字符串, 而中心点可能是某个字符也可能是两个字符中间的位置。对于一个长度为n的字符串, 这样的中心点有 $2n - 1$ 个: 有n个字符, 每个字符为一个; 另外, n个字符中, 每两个字符中间的位置为一个中心点, 这样的位置有 $n - 1$ 个。依次以这 $2n - 1$ 个位置为中心点对字符串进行扩展和判定, 获得最大回文子串。每次扩展最多为 $O(n)$ , 所以总的时间复杂度为 $O(n^2)$
- 第四种解法: Manacher's algorithm. 该算法的核心是利用回文字符串的对称性减少计算量。算法需要预处理字符串: 在字符串的首尾和两两字符间插入一个特殊字符“#”。该算法的详解见<http://articles.leetcode.com/longest-palindromic-substring-part-i/>。

```

1 class Solution(object):
2     # method1: DP
3     def longestPalindrome(self, s): # RT: O(n^2) TLE error, Space:
4         O(n^2)
5         n = len(s)

```



```

6      dp = [[False]*n for _ in xrange(n)]
7          for x in xrange(n):
8              dp[x][x] = True
9          for x in xrange(n-1):
10             if s[x]==s[x+1]:
11                 dp[x][x+1] = True
12     maxlenlength = 0
13     idx = 0
14     for length in xrange(3,n+1):
15         for x in xrange(n-length+1):
16             y = x+length-1
17             if s[x]==s[y] and dp[x+1][y-1]:
18                 dp[x][y] = True
19                 if maxlenlength < length:
20                     maxlenlength = length
21                     idx = x
22     return s[idx:idx+maxlength]
23
24 # method2: improved DP
25 def longestPalindrome2(self, s): # RT: O(n^2) TLE error, Space: O(1)
26     def expandAroundCenter(s, c1, c2):
27         l, r = c1, c2
28         n = len(s)
29         while l>=0 and r<=n-1 and s[l]==s[r]:
30             l-=1; r+=1

```

```

31         return s[1+1:r]
32
33     n = len(s)
34     if n==0: return ""
35     maxstring = s[0]
36     for x in xrange(1, n):
37         substring = expandAroundCenter(s, x-1, x)
38         if len(maxstring)<len(substring):
39             maxstring = substring
40
41         substring = expandAroundCenter(s, x, x)
42         if len(maxstring)<len(substring):
43             maxstring = substring
44     return maxstring
45
46 # method3: Manacher's algorithm
47 def longestPalindrome(self, s): # RT: O(n), Space: O(n)
48     def preprocess(s):
49         n = len(s)
50         if n==0: return "^$"
51         res = "^"
52         for x in xrange(n):
53             res += "#" + s[x]
54         res += "#$"
55         return res
56
57     T = preprocess(s)
58     n = len(T)
59     # dp[x] equals to the length of the palindrome centered at
T[x]
60     dp = [0 for _ in xrange(n)]
61     center, rightEdge = 0, 0
62     for x in xrange(1, n-1):
63         # x=center-(x-center)
64         x_mirror = 2*center-x
65
66         dp[x] = min(rightEdge-x, dp[x_mirror]) if rightEdge>x
else 0
67
68         # attempt to expand palindrome centers at x
69         while T[x+dp[x]+1]==T[x-dp[x]-1]: dp[x]+=1
70
71         # If palindrome centered at x expand past rightEdge,
72         # update center and rightEdge based on expanded
palindrome
73         if x+dp[x] > rightEdge:
74             center = x
75             rightEdge = x+dp[x]
76
77         # find the maximum element in dp
78     maxlen = 0
79     centeridx = 0
80     for x in xrange(1,n-1):
81         if maxlen < dp[x]:
82             maxlen = dp[x]
83             centeridx = x
84     start = (centeridx-maxlen-1)/2
85     return s[start:start+maxlen]

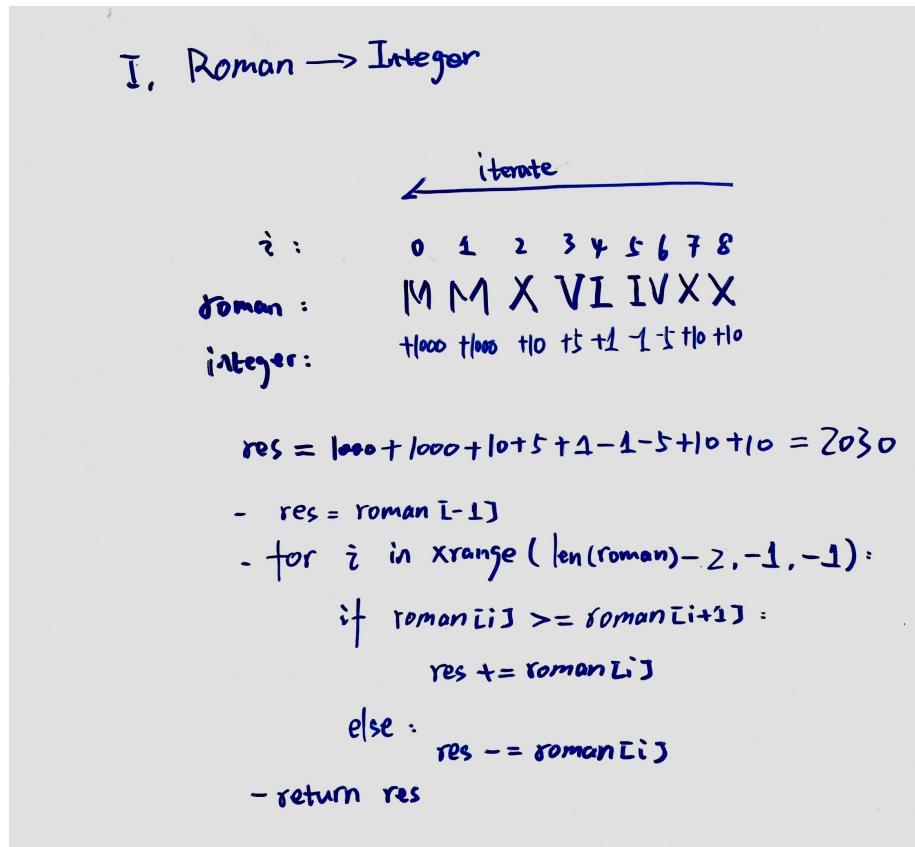
```

Listing 63: Problem5. Longest Palindromic Substring

## 2.7 leetcode 13. Roman to Integer

Given a roman numeral, convert it to an integer. Input is guaranteed to be within the range from 1 to 3999.

解题思路：从后向前遍历字符串。如果当前字符对应的数值不小于其后字符对应的数值，就进行累加；如果小于，则从累加和中减去其对应的数值。参见示例图。



```
1 class Solution(object):
2     def romanToInt(self, s):
3         """
4             :type s: str
5             :rtype: int
6             """
7         n = len(s)
8         dict = {'M':1000, 'D':500, 'C':100, 'L':50, 'X':10, 'V':5, 'I':1}
9         result = dict[s[n-1]]
10        for x in xrange(n-2, -1, -1):
11            if dict[s[x]] >= dict[s[x+1]]:
12                result += dict[s[x]]
13            else:
14                result -= dict[s[x]]
```

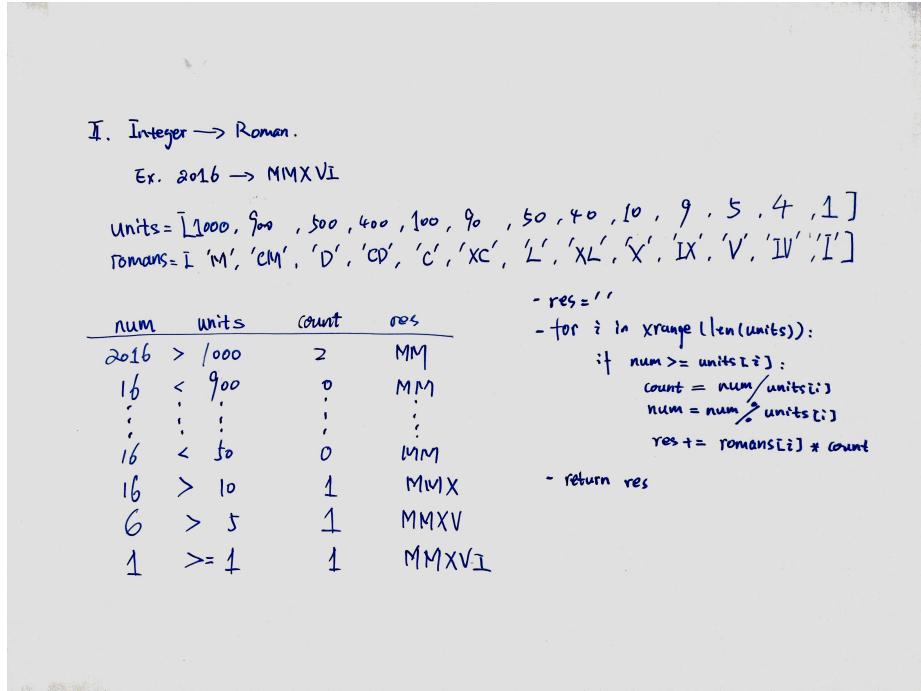
16      return result

Listing 64: Problem13. Roman to Integer

## 2.8 leetcode 12. Integer to Roman

Given an integer, convert it to a roman numeral. Input is guaranteed to be within the range from 1 to 3999.

解题思路：这道题比较简单，但是用Python编写算法时需要注意不能使用dict。因为dict.keys()生成的迭代器中key的次序和实际定义时的次序有可能不同，有可能是乱序，因此无法完成准确实现算法要达到的目的。



```
1 class Solution(object):
2     def intToRoman(self, num):
3         """
4             :type num: int
5             :rtype: str
6             """
7             res = ''
8             if num<=0: return res
9             units = [1000,900,500,400,100,90,50,40,10,9,5,4,1]
10            romans = ['M', 'CM', 'D', 'CD', 'C', 'XC', 'L', 'XL', 'X', 'IX', 'V', 'IV', 'I']
11            for i in xrange(len(units)):
12                if num >= units[i]:
13                    count = num / units[i]
14                    num = num % units[i]
15                    res += romans[i]*count
```

17      return res

Listing 65: Problem12. Integer to Roman

## 2.9 leetcode 44. Wildcard Matching

Implement wildcard pattern matching with support for '?' and '\*'. '?' Matches any single character. '\*' Matches any sequence of characters (including the empty sequence). The matching should cover the entire input string (not partial).

The function prototype should be: bool isMatch(const char \*s, const char \*p)

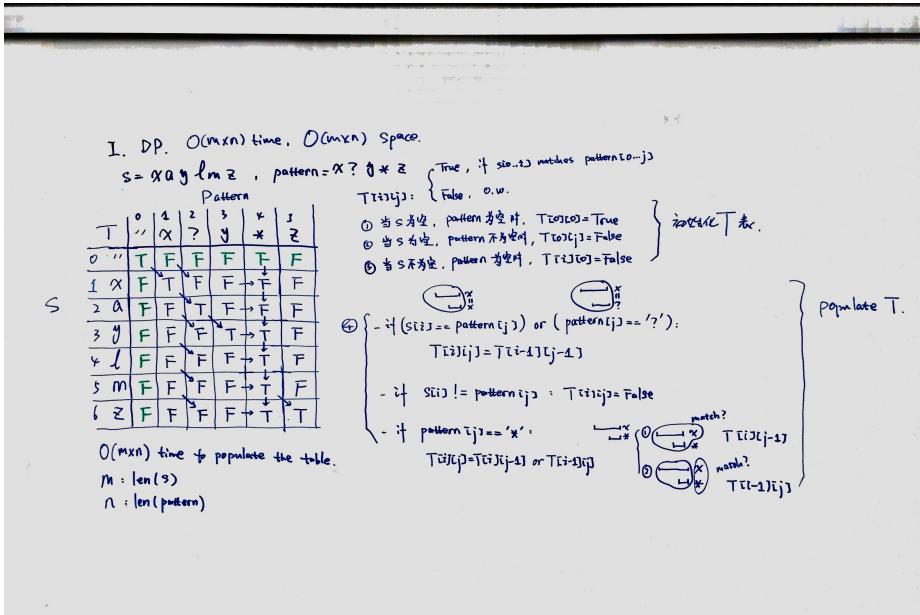
Some examples:

1. isMatch("aa", "a") → false
2. isMatch("aa", "aa") → true
3. isMatch("aaa", "aa") → false
4. isMatch("aa", "\*") → true
5. isMatch("aa", "a\*") → true
6. isMatch("ab", "?\*") → true
7. isMatch("aab", "c\*a\*b") → false

该题目有3种解法:

1. 递归算法: 该种方法的Python实现会超时。
2. 动态规划算法: dp[i][j]表示s串前i个与p串前j个是否匹配。转换方程如下:
  - 当 $p[j] == '?'$  ||  $p[j] == s[i]$ 时,  $dp[i][j] = dp[i-1][j-1]$ ;
  - 当 $p[j] == '*'$ 时, '\*'可以选择匹配 $s[i]$ 或者不匹配 $s[i]$ , 所以 $dp[i][j] = dp[i-1][j] \text{ or } dp[i][j-1]$ 。 $dp[i][j-1]$ 表示'\*'代表0个字符时, 前*i*个字符是否匹配pattern的前*j*-1个字符;  $dp[i-1][j]$ 表示'\*'代表多个字符时, 前*i*-1个字符的匹配情况。
  - 其他情况,  $dp[i][j] = False$
3. 迭代算法: 效率最高, 只需要一次遍历。算法的关键在于, 遇到'\*'时, 记录'\*'的位置 (lastCharMatchStar), 同时记录字符串s的位置 (lastCharMatchStar); 然后继续用'\*'后面一个字符继续和s当前位置的字符比较; 如果后续比较过程中出现不匹配的情况, 则lastCharMatchStar+1, 并从p的lastCharMatchStar+1位置和s的lastCharMatchStar位置重新开始比较。

DP和迭代两种算法的分析和示例参见示例图。

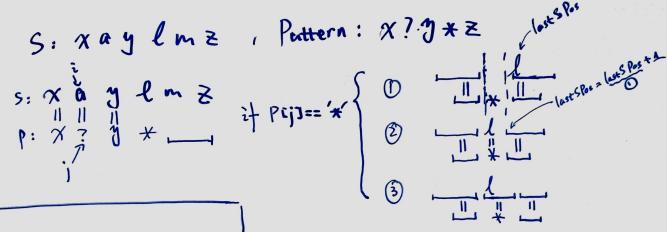


```

1  class Solution(object):
2      def isMatch_recursive(self, s, p): # TLE error
3          """
4              :type s: str
5              :type p: str
6              :rtype: bool
7              """
8
9      if len(p) > len(s):
10         return False
11     elif len(s) == 0 or len(p) == 0:
12         return len(s) == len(p)
13     elif len(s) == 1 and len(p) == 1 and s == p:
14         return True
15     elif s[0] == p[0] or p[0] == '?':
16         return self.isMatch(s[1:], p[1:])
17     elif p[0] == '*':
18         i = 0
19         while i < len(p) and p[i] == '*':
20             i += 1
21         if i == len(p): return True
22         j = 0
23         while j < len(p) and not self.isMatch(s[j:], p[i:]):
24             j += 1
25         return j != len(p)
26     return False
27
28     def isMatch_dp(self, s, p): # RT: O(n^2), Space: O(n^2)
29         """
30             :type s: str
31             :type p: str
32             :rtype: bool
33             """
34     if len(p) - p.count('*') > len(s): return False
35
36     # replace multiple * with one *

```

II. Iteration  $O(n)$  time,  $O(1)$  space.



- While  $i < m$  and  $j < n$ :
  - if ( $s[i] == p[j]$ ) or ( $p[j] == '?'$ ) :
    - $i += 1$ ;  $j += 1$
    - elif  $p[j] == '*' :$
    - else : return False
  - $lastStarPos = j$ ;  $lastSPos = i$
  - $j = j + 1$
  - while  $i < m$  and  $j < n$  and  $s[i] == p[j]$ :
    - $i += 1$ ;  $j += 1$
  - if  $i == m$  and  $j == n$  : return True
  - elif  $i == m$  or  $j == n$  : return False
  - else : #  $s[i] \neq p[j]$ 
    - $lastSPos += 1$
    - $i = lastSPos$
    - $j = lastStarPos + 1$
- return True if  $i == m$  and  $j == n$  else False

```

37     # e.g., a***b**c => a*b*c
38     pattern = []
39     isFirst = True
40     for i in range(len(p)):
41         if p[i]=='*':
42             if isFirst:
43                 pattern.append(p[i])
44                 isFirst = False
45             else:
46                 pattern.append(p[i])
47                 isFirst = True
48     m, n = len(s), len(pattern)
49
50     # initialize dp matrix
51     dp=[[False for _ in range(n+1)] for _ in range(m+1)]
52     dp[0][0]=True
53     if n>0 and pattern[0]=='*': dp[0][1] = True
54
55     # populate the dp matrix
56     for i in range(1, m+1): # for string
57         for j in range(1, n+1): # for pattern
58             if pattern[j-1]=='*':
59                 dp[i][j] = dp[i-1][j] or dp[i][j-1]
60             elif s[i-1]==pattern[j-1] or pattern[j-1]=='?':
61                 dp[i][j] = dp[i-1][j-1]
62             else:
63                 dp[i][j] = False
64     return dp[m][n]
65
66 def isMatch_iterative(self, s, p): # RT: O(n), Space: O(1)
67     """
68     :type s: str

```

```

69     :type p: str
70     :rtype: bool
71     """
72     # the position of the last occurrence of * in p
73     lastStartPos = -1
74     # the position of the last character in s which
75     # matches with *
76     lastCharMatchStar = 0
77     idxPattern = idxStr = 0
78     while idxStr < len(s):
79         # when there is a match, continue to compare
80         # the characters in s and p
81         if idxPattern < len(p) and (s[idxStr] == p[idxPattern] or p
82             [idxPattern] == '?'):
83             idxStr += 1
84             idxPattern += 1
85             continue
86             # when * comes, save the positions of idxPattern
87             # and idxStr
88             elif idxPattern < len(p) and p[idxPattern] == '*':
89                 lastStartPos = idxPattern
90                 lastCharMatchStar = idxStr
91                 idxPattern += 1
92                 continue
93                 # when there is a conflict, and we have a star,
94                 # reset idxPattern to point to the next position
95                 # of previous, and also move lastCharMatchStar
96                 # to its next position, and set idxStr to point
97                 # to lastCharMatchStar's position;
98                 # after these settings, continue to check
99                 elif lastStartPos != -1:
100                     idxPattern = lastStartPos + 1
101                     lastCharMatchStar += 1
102                     idxStr = lastCharMatchStar
103                     continue
104                     # if all the conditions are not satisfied,
105                     # s does not match p.
106                     else: return False
107                     while idxPattern < len(p) and p[idxPattern] == '*':
108                         idxPattern += 1
109                         if idxPattern == len(p): return True
110                         return False

```

Listing 66: Problem44. Wildcard Matching

## 3 Stack and Queue

### 3.1 Stack

#### 3.1.1 leetcode 20. Valid Parentheses

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.

解题思路：这道题考查对栈数据结构的基本知识，问题比较简单。解题的基本步骤参考题图。

Steps:

- (1) if we have a left bracket, then push it into stack.
- (2) if we have a right bracket, then pop the top item out of the stack, and meanwhile to verify if they're a correct pair of brackets.

```
def isValid(s):
    dict = {'>': '(', '}': '{', ')': '['}
    stack = []
    for x in s:
        if x in '({[': stack.append(x))
        else: top = stack.pop()
            if top != dict[x]: return False
    return True if stack == [] else False.
```

```
2 class Solution(object):
3     def isValid(self, s):
4         """
5             :type s: str
6             :rtype: bool
7         """
8         stack = []
9         for x in s:
10            if x in '({[': stack.append(x))
11            else:
12                if stack == []: return False
13                top = stack.pop()
14                if (top == '(' and x == ')') or \
15                    (top == '[' and x == ']') or \
16                    (top == '{' and x == '}'):
17                    continue
18                else: return False
19            if stack != []: return False
20        return True
```

Listing 67: Problem20. Valid Parentheses

### 3.1.2 leetcode 32. Longest Valid Parentheses

Given a string containing just the characters '(' and ')', find the length of the longest valid (well-formed) parentheses substring. For "(()", the longest valid parentheses substring is "()", which has length = 2. Another example is ")()()", where the longest valid parentheses substring is "()()", which has length = 4.

解题思路：这道题是个hard级别的题，解题思路与leetcode 20. Valid Parentheses有所差别，需要注意左括号多和右括号多这两种情况。

```
1 class Solution(object):
2     # use a stack to record the indices of unmatched left
3     # parentheses
4     def longestValidParentheses(self, s): # RT: O(n), Space: O(n)
5         """
6             :type s: str
7             :rtype: int
```

```

8      """
9      maxlen = 0
10     # store the indices of all unmatched left brackets
11     stack = []
12     # index of the last unmatched right bracket
13     lastUnmatchedRight = -1
14     for i in range(len(s)):
15         if s[i]=='(': stack.append(i) # push the INDEX into the
16         stack!!!!
17     else:
18         if stack == []: lastUnmatchedRight = i
19         else:
20             stack.pop()
21             if stack == []:
22                 # current right bracket - the last
23                 unmatched right bracket
24                 maxlen = max(maxlen, i - lastUnmatchedRight
25             )
26             else:
27                 # current right bracket - the last
28                 unmatched left bracket
29                 maxlen = max(maxlen, i - stack[-1])
30
31     return maxlen

```

Listing 68: Problem32. Longest Valid Parentheses

$i: 0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12$   
 $s[i]: ) ( ( ) ) ) ( ( ) ) ( )$   
 $\text{lastUnmatched: } 0 \ 0 \ 0 \ 0 \ 5 \ 6 \ 6 \ 6 \ 6 \ 6 \ 6 \ 6 \ 6$   
 $\downarrow \quad \downarrow \quad \downarrow$   
 $2-0 \quad 4-0 \quad \quad \quad 9-6 \quad 10-6 \quad 12-6$   
 $\text{ maxlen: } 0 \ 0 \rightarrow 2 \ 2 \rightarrow 4 \ 4 \ 4 \ 4 \ 4 \rightarrow 4 \ 4 \rightarrow 6$   
 $\text{stack: } () (1) (2) (3) (4) (5) (7) (8) (7) (11) (1)$

### 3.1.3 leetcode 84. Largest Rectangle in Histogram

Given n non-negative integers representing the histogram's bar height where the width of each bar is 1, find the area of largest rectangle in the histogram. For example, Given heights = [2,1,5,6,2,3], return 10.

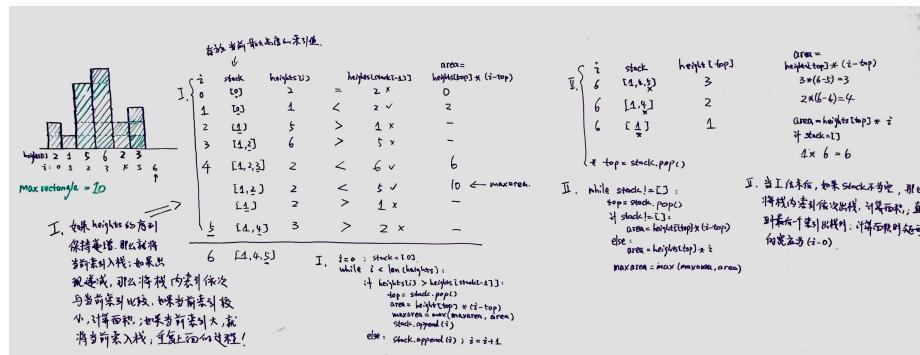
解题思路：参考下面的示例图中的分析和演算过程。强烈建议观看Tushar Roy的讲解视频：<https://www.youtube.com/watch?v=ZmnqCZp9bBs>

```

1  class Solution(object):
2      def largestRectangleArea(self, heights): # RT: O(n), Space: O(n)
3          """
4              :type heights: List[int]
5              :rtype: int
6          """
7          n = len(heights)
8          maxarea, area = 0, 0
9          stack = []
10         x = 0
11         while x < n:
12             if stack == [] or (heights[stack[-1]] <= heights[x]):
13                 stack.append(x); x+=1
14             else:
15                 top = stack.pop()
16                 if stack == []:
17                     area = heights[top] * x
18                 else:
19                     area = heights[top] * (x - stack[-1] - 1)
20             maxarea = max(maxarea, area)
21         while stack != []:
22             top = stack.pop()
23             if stack == []:
24                 area = heights[top] * x
25             else:
26                 area = heights[top] * (x - stack[-1] - 1)
27             maxarea = max(maxarea, area)
28
29         return maxarea

```

Listing 69: Problem84. Largest Rectangle in Histogram



### 3.1.4 leetcode 150. Evaluate Reverse Polish Notation

Evaluate the value of an arithmetic expression in Reverse Polish Notation. Valid operators are  $+$ ,  $-$ ,  $*$ ,  $/$ . Each operand may be an integer or another expression.

Some examples:

- $["2", "1", "+", "3", "*"] \rightarrow ((2 + 1) * 3) \rightarrow 9$

- [”4”, ”13”, ”5”, ”/”, ”+”] → (4 + (13/5)) → 6

这道题对于Python实现有两个陷阱：第一个是栈的先进后出特性，在做减法和除法时，应该是后一个弹出的值作为被减数和被除数；第二个陷阱与Python版本相关，Python2.x的除法是floor division，所以在两个操作数异号时，得到的结果会有错，例如 $-3/2=-2$ ，而不是 $-1$ ；解决的办法是先判断一下两个操作数是否同号，如果是一正一负，那么就先转换成相应的正数，然后做完除法以后将结果置为负值即可（参考代码行21-22）。

```

1  class Solution(object):
2      def evalRPN(self, tokens):
3          """
4              :type tokens: List[str]
5              :rtype: int
6          """
7
8          def evalRPN(self, tokens):
9              stack = []
10             for x in tokens:
11                 if stack==[] or x not in '+-*':
12                     stack.append(int(x))
13                 else:
14                     op1, op2 = stack.pop(), stack.pop()
15                     if x=='+':
16                         stack.append(op1+op2)
17                     elif x=='-':
18                         stack.append(op2-op1)
19                     elif x=='*':
20                         stack.append(op1*op2)
21                     elif x=='/':
22                         # Python2 uses "floor division"
23                         # e.g., -1/2=-1, but not 0
24                         if op1*op2 < 0:
25                             stack.append(-(-op2/op1))
26                         else:
27                             stack.append(op2/op1)
28
29             return stack.pop()

```

Listing 70: Problem150. Evaluate Reverse Polish Notation

## 4 Trees

有关树的考点：

1. 第一类问题是树的先序、中序、后序遍历算法的递归和迭代实现，重点是迭代。很多题目的算法都是基于改进上述三种遍历算法而得到的。需要注意，后序遍历的迭代算法需要一些小技巧。有关二叉树的三种遍历算法见leetcode 144, 94, 145题的解法。
2. 第二类问题源自于第一类问题，主要考查的是对于先序、中序、后序遍历算法得出的结果的理解，比如中序遍历BST二叉搜索数得到的序列是递增序列，题目leetcode 99就是通过这个性质来构造算法。
3. 第三类问题是给出二叉树的先序和中序（或者后序和中序），然后构造二叉树。这里需要注意只有先序和后序是无法构造二叉树的，因为无法确定根的位置。leetcode105、106考查了这个知识点。
4. 第四类问题关于树的结构性的问题。这类问题主要是考查对二叉树的深度和广度两种优先搜索算法的理解。比如leetcode 100、101、102、103、107等题目都是基于上述两种基本算法的改进而得。其中，广度优先搜索算法是重点。

5. 第五类问题是关于树的分支、路径、深度、高度等属性。这类问题考查的实质就是对于树的DFS算法的熟悉和掌握程度。通常用DFS的递归实现即可。

## 4.1 Binary Tree Traversal

### 4.1.1 leetcode 105. Construct Binary Tree from Preorder and In-order Traversal

Given preorder and inorder traversal of a tree, construct the binary tree. Note: You may assume that duplicates do not exist in the tree.

解题思路：二叉树的先序遍历提供了树根的位置，即先序遍历的第一个元素；在确定了树根在中序遍历中的位置以后，中序遍历提供左、右子树的信息。据此，可以递归构造二叉树。这里的关键点是第21行代码，在每次确定树根以后，需要删除先序遍历中的树根值，以缩小先序遍历的长度，否则代码会造成memory limit exceeded错误。参考下面的示例图中的分析。参考下面示例图中的分析和演算。

```

1  # Definition for a binary tree node.
2  # class TreeNode(object):
3  #     def __init__(self, x):
4  #         self.val = x
5  #         self.left = None
6  #         self.right = None
7
8
9  class Solution(object):
10     def buildTree(self, preorder, inorder):
11         """
12             :type preorder: List[int]
13             :type inorder: List[int]
14             :rtype: TreeNode
15         """
16         if len(preorder) == 0 or len(inorder) == 0:
17             return None
18
19         root = TreeNode(preorder[0])
20         i = inorder.index(root.val)
21         preorder.pop(0)
22         root.left = self.buildTree(preorder, inorder[:i])
23         root.right = self.buildTree(preorder, inorder[i+1:])
24
25     return root

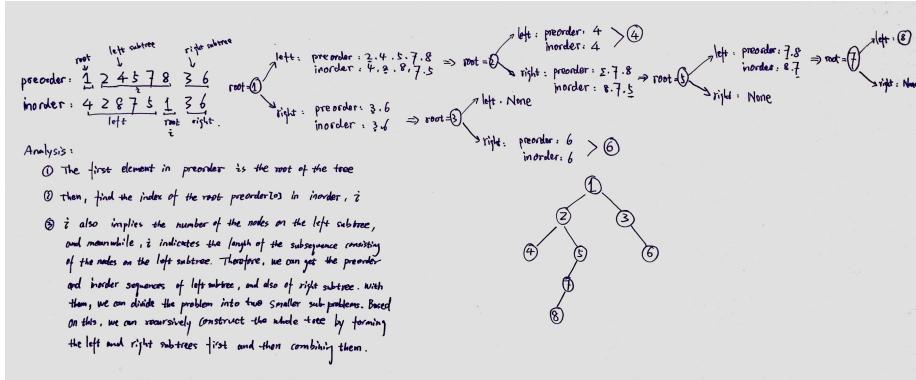
```

Listing 71: Problem105. Construct Binary Tree from Preorder and Inorder Traversal

### 4.1.2 leetcode 106. Construct Binary Tree from Inorder and Postorder Traversal

Given inorder and postorder traversal of a tree, construct the binary tree. Note: You may assume that duplicates do not exist in the tree.

解题思路：与leetcode 105类似的方法。不同之处在于，本题由后序遍历提供了树根的位置信息，即后序遍历的最后一个元素即为树根；在确定了树根在



中序遍历中的位置以后，中序遍历提供左、右子树的信息。据此，可以递归构造二叉树。这里的关键点是第21行代码，在每次确定树根以后，需要删除先序遍历中的树根值，以缩小先序遍历的长度，否则代码会造成memory limit exceeded错误。参考下面示例图中的分析和演算。

```

1
2 # Definition for a binary tree node.
3 # class TreeNode(object):
4 #     def __init__(self, x):
5 #         self.val = x
6 #         self.left = None
7 #         self.right = None
8
9 class Solution(object):
10     def buildTree(self, inorder, postorder): # MLE error
11         """
12             :type inorder: List[int]
13             :type postorder: List[int]
14             :rtype: TreeNode
15         """
16         if len(inorder)==0 or len(postorder)==0:
17             return None
18         root = TreeNode(postorder[-1])
19         i = inorder.index(root.val)
20         postorder.pop()
21         root.right = self.buildTree(inorder[i+1:], postorder)
22         root.left = self.buildTree(inorder[:i], postorder)
23
24     return root

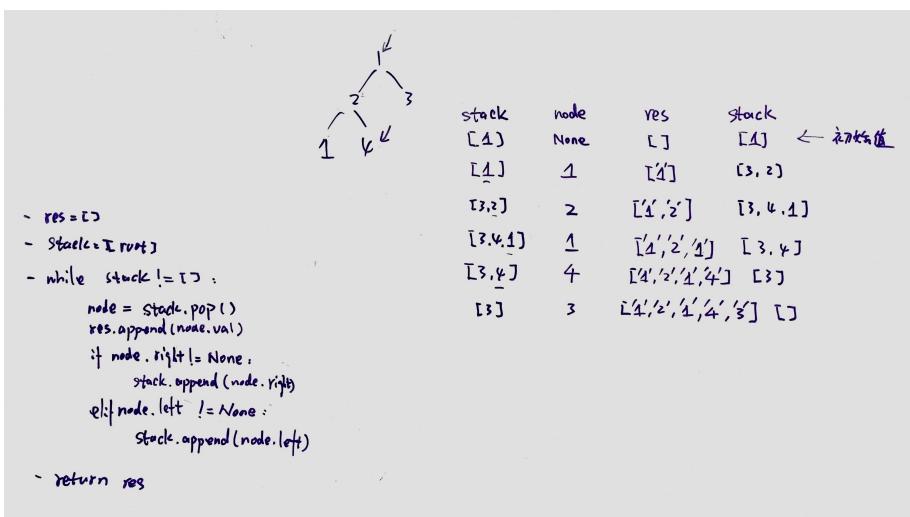
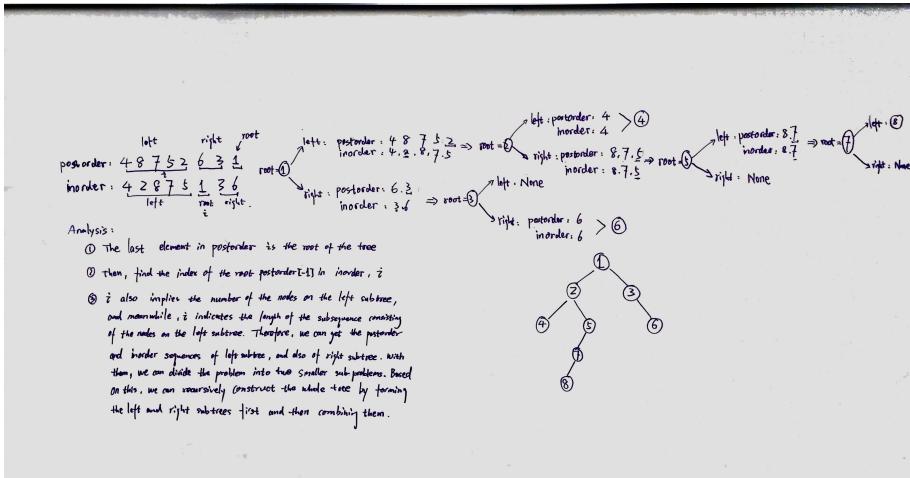
```

Listing 72: Problem106. Construct Binary Tree from Inorder and Postorder Traversal

#### 4.1.3 leetcode 144. Binary Tree Preorder Traversal

Given a binary tree, return the preorder traversal of its nodes' values. Note: Recursive solution is trivial, could you do it iteratively?

解题思路：在使用栈来存放结点的时候，要先放右子树的根结点，再放左子树的根结点。参考下面示例图中的分析和演算。



```

2 # Definition for a binary tree node.
3 # class TreeNode(object):
4 #     def __init__(self, x):
5 #         self.val = x
6 #         self.left = None
7 #         self.right = None
8
9 class Solution(object):
10     # iterative traversal version: RT: O(n), Space: O(n)
11     def preorderTraversal_iterative(self, root):
12         """
13             :type root: TreeNode
14             :rtype: List[int]
15         """
16         result = []
17         if root == None: return result
18         stack = [root]
19         while stack != []:
20             node = stack.pop()
21             result += [node.val]

```

```

22         if node.right is not None: stack.append(node.right)
23         if node.left is not None: stack.append(node.left)
24     return result
25
26 # recursive traversal version: RT: O(n), Space: O(n)
27 def preorderTraversal_recursive(self, root):
28     """
29     :type root: TreeNode
30     :rtype: List[int]
31     """
32     result = []
33     if root == None: return result
34
35     # root
36     result += [root.val]
37     # left subtree
38     if root.left is not None:
39         result += self.preorderTraversal(root.left)
40     # right subtree
41     if root.right is not None:
42         result += self.preorderTraversal(root.right)
43
44     return result

```

Listing 73: Problem144. Binary Tree Preorder Traversal

#### 4.1.4 leetcode 94. Binary Tree Inorder Traversal

Given a binary tree, return the inorder traversal of its nodes' values. Note: Recursive solution is trivial, could you do it iteratively?

```

1  # Definition for a binary tree node.
2  # class TreeNode(object):
3  #     def __init__(self, x):
4  #         self.val = x
5  #         self.left = None
6  #         self.right = None
7
8
9  class Solution(object):
10    # interative traversal
11    def inorderTraversal(self, root):
12        """
13        :type root: TreeNode
14        :rtype: List[int]
15        """
16        result = []
17        if root == None: return result
18
19        stack = []
20        while len(stack)>0 or root!=None:
21            if root!= None:
22                stack.append(root)
23                root = root.left
24            else:
25                root = stack.pop()
26                result.append(root.val)
27                root = root.right
28
29        return result

```

```

30
31     # recursive traversal
32     def inorderTraversal_recursive(self, root):
33         """
34             :type root: TreeNode
35             :rtype: List[int]
36         """
37         result = []
38         if root == None: return result
39
40         if root.left != None:
41             result += self.inorderTraversal(root.left)
42
43         result.append(root.val)
44
45         if root.right != None:
46             result += self.inorderTraversal(root.right)
47
48     return result

```

Listing 74: Problem94. Binary Tree Inorder Traversal

#### 4.1.5 leetcode 145. Binary Tree Postorder Traversal

Given a binary tree, return the postorder traversal of its nodes' values. Note: Recursive solution is trivial, could you do it iteratively?

```

1
2 # Definition for a binary tree node.
3 # class TreeNode(object):
4 #     def __init__(self, x):
5 #         self.val = x
6 #         self.left = None
7 #         self.right = None
8
9 class Solution(object):
10     def postorderTraversal_iterative(self, root):
11         """
12             :type root: TreeNode
13             :rtype: List[int]
14         """
15         result = []
16         if root == None:
17             return result
18
19         pre = None
20         stack = [root]
21         while stack != []:
22             curr = stack[-1]
23             if (curr.left==None and curr.right==None) or (pre and (
24                 pre == curr.left or pre == curr.right)):
25                 result.append(curr.val)
26                 pre = stack.pop() # pre = curr
27             else:
28                 if curr.right:
29                     stack.append(curr.right)
30                 if curr.left:
31                     stack.append(curr.left)
32
33     return result

```

```

33     def postorderTraversal_recursive(self, root):
34         """
35             :type root: TreeNode
36             :rtype: List[int]
37         """
38         result = []
39
40         if root == None:
41             return result
42
43         if root.left != None:
44             result += self.postorderTraversal(root.left)
45         if root.right != None:
46             result += self.postorderTraversal(root.right)
47         result.append(root.val)
48
49     return result

```

Listing 75: Problem145. Binary Tree Postorder Traversal

#### 4.1.6 leetcode 102. Binary Tree Level Order Traversal

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

```

1
2 # Definition for a binary tree node.
3 # class TreeNode(object):
4 #     def __init__(self, x):
5 #         self.val = x
6 #         self.left = None
7 #         self.right = None
8
9 class Solution(object):
10     def levelOrder(self, root):
11         """
12             :type root: TreeNode
13             :rtype: List[List[int]]
14         """
15         res = []
16         if root==None: return res
17         currLevel = [root]
18         while currLevel != []:
19             nextLevel = []
20             values = []
21             for elem in currLevel:
22                 values.append(elem.val)
23                 if elem.left: nextLevel.append(elem.left)
24                 if elem.right: nextLevel.append(elem.right)
25             if values!=[]: res.append(values)
26             currLevel = nextLevel
27
28     return res

```

Listing 76: Problem102. Binary Tree Level Order Traversal

#### 4.1.7 leetcode 107. Binary Tree Level Order Traversal II

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).

For example: Given binary tree {3,9,20,#,#,15,7}, return its bottom-up level order traversal as [[15, 7], [9, 20], [3]].

```

1 # Definition for a binary tree node.
2 # class TreeNode(object):
3 #     def __init__(self, x):
4 #         self.val = x
5 #         self.left = None
6 #         self.right = None
7 #
8
9 class Solution(object):
10     def levelOrderBottom(self, root):
11         """
12             :type root: TreeNode
13             :rtype: List[List[int]]
14         """
15         res = []
16         if root==None: return res
17         currLevel = [root]
18         while currLevel != []:
19             nextLevel = []
20             values = []
21             for item in currLevel:
22                 values.append(item.val)
23                 if item.left: nextLevel.append(item.left)
24                 if item.right: nextLevel.append(item.right)
25             if values!=[]: res.append(values)
26             currLevel = nextLevel
27
28         return res[::-1]

```

Listing 77: Problem107. Binary Tree Level Order Traversal II

#### 4.1.8 leetcode 103. Binary Tree Zigzag Level Order Traversal

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).

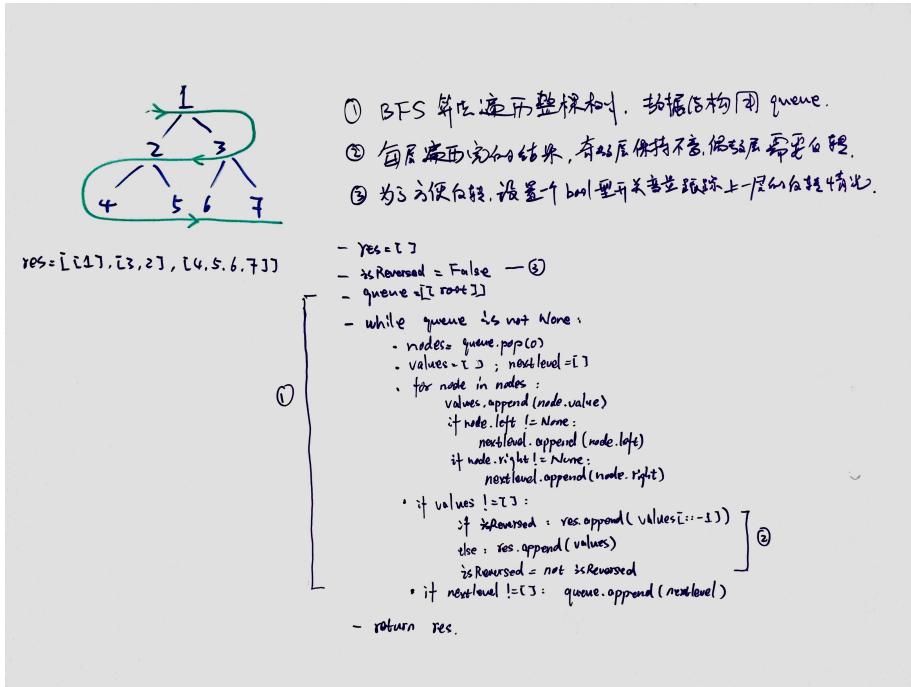
For example: Given binary tree 3,9,20,,15,7, return its bottom-up level order traversal as [[3], [20, 9], [15, 7]].

这道题不要翻转结点列表，而是翻转每次得到的值列表，这样算法比较简单！

```

1 # Definition for a binary tree node.
2 # class TreeNode(object):
3 #     def __init__(self, x):
4 #         self.val = x
5 #         self.left = None
6 #         self.right = None
7 #
8
9 class Solution(object):
10     def zigzagLevelOrder(self, root):
11         """
12             :type root: TreeNode
13             :rtype: List[List[int]]
14         """
15         def zigzagLevelOrder(self, root):
16             """

```



```

17     :type root: TreeNode
18     :rtype: List[List[int]]
19     """
20     res = []
21     if root==None: return res
22     currLevel = [root]
23     reversed = False
24     while currLevel != []:
25         nextLevel = []; values = []
26         for item in currLevel:
27             values.append(item.val)
28             if item.left: nextLevel.append(item.left)
29                 if item.right: nextLevel.append(item.right)
30             if values != []: res.append(values[::-1] if reversed else
values)
31             reversed = not reversed
32             currLevel = nextLevel
33     return res
  
```

Listing 78: Problem103. Binary Tree Zigzag Level Order Traversal

#### 4.1.9 leetcode 99. Recover Binary Search Tree

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?

解题方法：利用中序遍历算法，寻找破坏序列为升序的两个节点，然后交换这两个节点即可。由于题目要求空间复杂度为常数，所以需要使用一个指针pre指

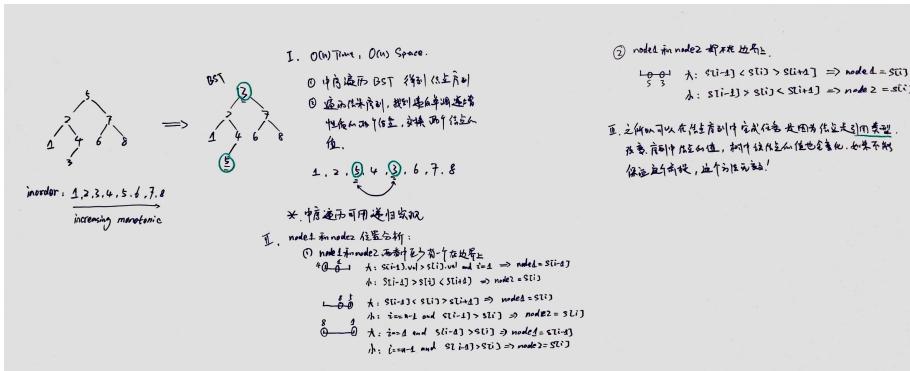
向两个相比较节点的前一个，而root指向当前节点；另外，用 $n_1$ 和 $n_2$ 两个变量存储需要交换的两个结点。参考示例图中的分析。

```

1 # Definition for a binary tree node.
2 # class TreeNode(object):
3 #     def __init__(self, x):
4 #         self.val = x
5 #         self.left = None
6 #         self.right = None
7 #
8
9 class Solution(object):
10     def recoverTree(self, root):
11         """
12             :type root: TreeNode
13             :rtype: void Do not return anything, modify root in-place
14             instead.
15         """
16         def findSwapNodes(root):
17             # inorder traverse the binary tree
18             if root:
19                 findSwapNodes(root.left)
20                 if self.pre and self.pre.val > root.val:
21                     self.n2 = root
22                     if self.n1 == None: self.n1 = self.pre
23                     self.pre = root
24                 findSwapNodes(root.right)
25             self.n1, self.n2, self.pre = None, None, None
26         findSwapNodes(root)
27         self.n1.val, self.n2.val = self.n2.val, self.n1.val

```

Listing 79: Problem99. Recover Binary Search Tree



#### 4.1.10 leetcode 100. Same Tree

Given two binary trees, write a function to check if they are equal or not. Two binary trees are considered equal if they are structurally identical and the nodes have the same value.

解题方法：参考示例图中的分析和代码。

```

1 # Definition for a binary tree node.
2 # class TreeNode(object):
3 #     def __init__(self, x):
4 #         self.val = x
5 #         self.left = None
6 #         self.right = None
7 #
8
9 class Solution(object):
10     def isSameTree(self, p, q):
11         if p == None and q == None:
12             return True
13         if p == None or q == None:
14             return False
15         if p.val != q.val:
16             return False
17         return self.isSameTree(p.left, q.left) and self.isSameTree(p.right, q.right)

```

```

3 # class TreeNode(object):
4 #     def __init__(self, x):
5 #         self.val = x
6 #         self.left = None
7 #         self.right = None
8
9 class Solution(object):
10    def isSameTree_iterative(self, p, q):
11        """
12            :type p: TreeNode
13            :type q: TreeNode
14            :rtype: bool
15        """
16        if p==None: return q==None
17        if q==None: return p==None
18        stack = [(p,q)]
19        while stack!=[]:
20            root1, root2 = stack.pop()
21            if root1==None and root2==None: continue
22            elif root1 and root2 and root1.val==root2.val:
23                stack.append((root1.left, root2.left))
24                stack.append((root1.right, root2.right))
25            else: return False
26        return True
27
28    def isSameTree_recursive(self, p, q):
29        """
30            :type p: TreeNode
31            :type q: TreeNode
32            :rtype: bool
33        """
34        if p == None: return q == None
35        if q == None: return False
36        if p.val == q.val:
37            return self.isSameTree(p.left, q.left) and self.
38            isSameTree(p.right, q.right)
39        else:
40            return False

```

Listing 80: Problem100. Same Tree

#### 4.1.11 leetcode 101. Symmetric Tree

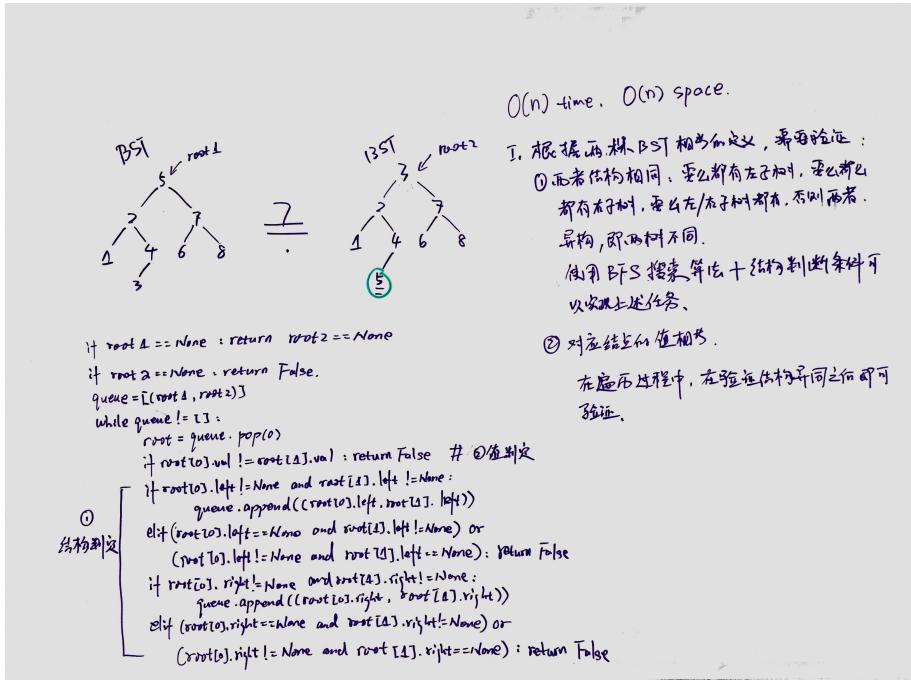
Given a binary tree, check whether it is a mirror of itself (ie, symmetric around its center). Note: Bonus points if you could solve it both recursively and iteratively.

解题方法：验证树的结构性问题，考查的重点是对二叉树遍历算法的理解。参考示例图中的分析和代码。

```

1
2 # Definition for a binary tree node.
3 # class TreeNode(object):
4 #     def __init__(self, x):
5 #         self.val = x
6 #         self.left = None
7 #         self.right = None
8
9 class Solution(object):
10    def isSymmetric_iterative(self, root):
11        if root==None: return True

```

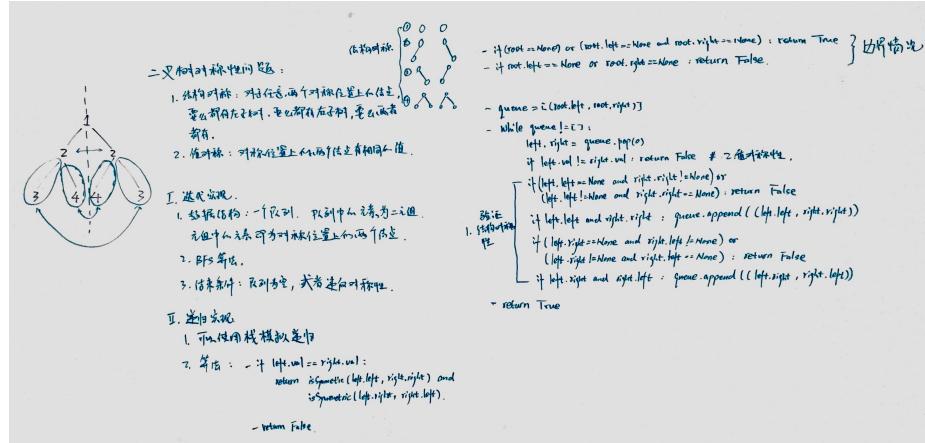


```

12     lqueue = [root.left]
13     rqueue = [root.right]
14     while lqueue != [] and rqueue != []:
15         root1, root2 = lqueue.pop(0), rqueue.pop(0)
16         if root1 == None and root2 == None: continue
17         elif root1 and root2 and root1.val == root2.val:
18             lqueue.append(root1.left); lqueue.append(root1.
right)
19             rqueue.append(root2.right); rqueue.append(root2.
left)
20         else: return False
21     return True
22
23 def isSymmetric_recursive(self, root):
24     """
25     :type root: TreeNode
26     :rtype: bool
27     """
28     def checkSymmetry(root1, root2):
29         if root1 == None: return root2 == None
30         if root2 == None: return False
31         if root1.val == root2.val:
32             return checkSymmetry(root1.left, root2.right) and
checkSymmetry(root1.right, root2.left)
33         else: return False
34
35     if root == None: return True
36     return checkSymmetry(root.left, root.right)

```

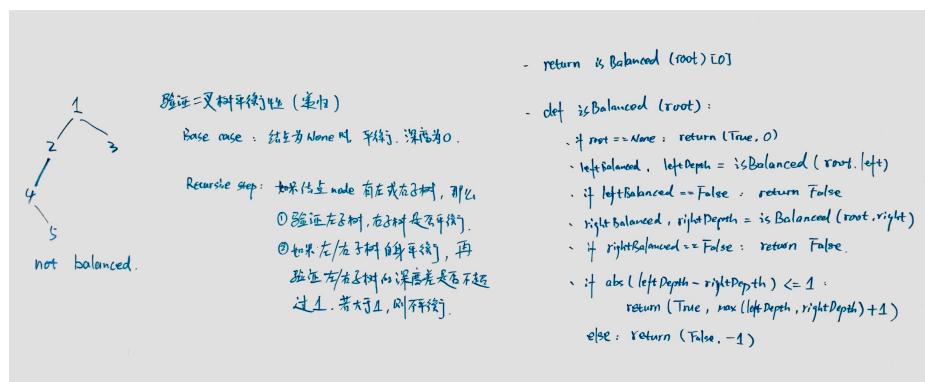
Listing 81: Problem101. Symmetric Tree



#### 4.1.12 leetcode 110. Balanced Binary Tree

Given a binary tree, determine if it is height-balanced. For this problem, a height-balanced binary tree is defined as a binary tree in which the depth of the two subtrees of every node never differ by more than 1.

解题方法：考查平衡二叉树的基本性质。参考示例图中的分析和代码。



```

1
2 # Definition for a binary tree node.
3 # class TreeNode(object):
4 #     def __init__(self, x):
5 #         self.val = x
6 #         self.left = None
7 #         self.right = None
8
9 class Solution(object):
10     def isBalanced(self, root):
11         def helper(root):
12             if root==None: return (True, 0)
13             # check left subtree
14             left = helper(root.left)
15             if left[0]==False: return (False, -1)
16             # check right subtree
17             right = helper(root.right)
  
```

```

18     if right[0]==False: return (False, -1)
19     # check if the depths of the two subtrees
20     # differ by more than 1
21     if abs(left[1]-right[1])<=1:
22         return (True, max(left[1], right[1])+1)
23     else: return (False, -1)
24     return helper(root)[0]

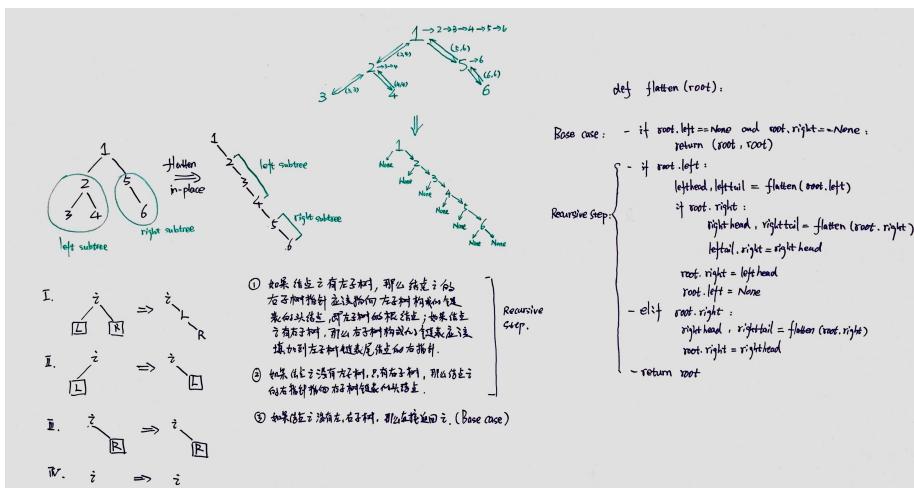
```

Listing 82: Problem110. Balanced Binary Tree

#### 4.1.13 leetcode 114. Flatten Binary Tree to Linked List

Given a binary tree, flatten it to a linked list in-place.

解题方法：这道题考查的基本点还是对于树的遍历算法的理解和掌握。观察题目给出的例子（参考下面示例图中的例子）可以发现：结果中的结点序列实际上就是二叉树的先序遍历的结果，所以考查点实际就是对二叉树先序遍历的掌握。如果不借助额外的存储，算法可以考虑使用递归实现。参考示例图中的分析和代码。



```

1
2 # Definition for a binary tree node.
3 # class TreeNode(object):
4 #     def __init__(self, x):
5 #         self.val = x
6 #         self.left = None
7 #         self.right = None
8
9 class Solution(object):
10     def flatten(self, root): # RT: O(n)
11         """
12             :type root: TreeNode
13             :rtype: void Do not return anything, modify root in-place
14             instead.
15         """
16         p = root
17         stack = []
18         while p:

```

```

19         if stack != []:
20             p.right = stack.pop()
21             p = p.right
22         else: break
23     elif p.left and p.right:
24         stack.append(p.right)
25         p.right = p.left
26         p.left = None
27         p = p.right
28     elif p.left:
29         p.right = p.left
30         p.left = None
31         p = p.right
32     elif p.right:
33         p = p.right

```

Listing 83: Problem114. Flatten Binary Tree to Linked List

#### 4.1.14 leetcode 116. Populating Next Right Pointers in Each Node

Given a binary tree

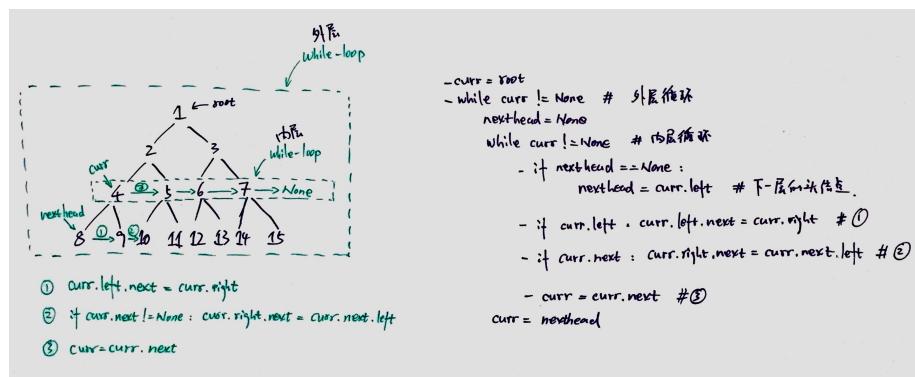
```

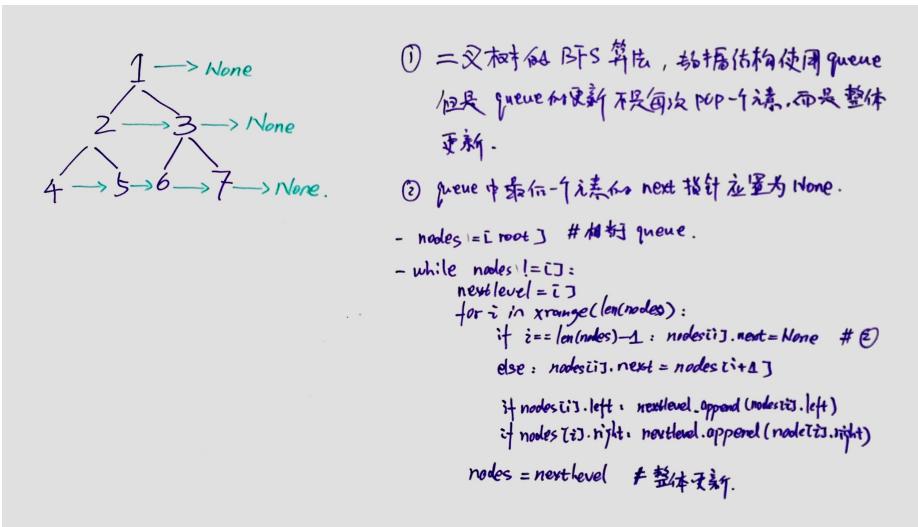
struct TreeLinkNode {
    TreeLinkNode *left;
    TreeLinkNode *right;
    TreeLinkNode *next;
}

```

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. Initially, all next pointers are set to NULL. Note: You may only use constant extra space. You may assume that it is a perfect binary tree (ie, all leaves are at the same level, and every parent has two children).

解题方法：这道题考查的基本点是二叉树的BFS算法。这道题与leetcode 102. Binary Tree Level Order Traversal的核心思想和考点完全相同。空间复杂度为 $O(1)$ 的算法可以参考示例图一。示例图二中的算法使用了额外的存储空间，但是图中的算法也可以解决leetcode117的问题。





```

2 # Definition for binary tree with next pointer.
3 # class TreeLinkNode(object):
4 #     def __init__(self, x):
5 #         self.val = x
6 #         self.left = None
7 #         self.right = None
8 #         self.next = None
9
10 class Solution(object):
11     def connect_recursive(self, root): # RT: O(n), Space: O(1)
12         """
13             :type root: TreeLinkNode
14             :rtype: nothing
15         """
16         if root and root.left:
17             root.left.next = root.right
18             if root.next:
19                 root.right.next = root.next.left
20             self.connect(root.left)
21             self.connect(root.right)
22
23     def connect_iterative(self, root): # RT: O(n), Space: O(1)
24         """
25             :type root: TreeLinkNode
26             :rtype: nothing
27         """
28         if root:
29             curr = root
30             while curr:
31                 nextLevel = None
32                 # traverse all nodes of current level
33                 while curr:
34                     if curr.left:
35                         # locate the start node of next level
36                         if nextLevel == None:
37                             nextLevel = curr.left
38                         curr.left.next = curr.right
39
39                     if curr.next:
40

```

```

41             curr.right.next = curr.next.left
42             curr = curr.next
43         # set current node as the first node of next level
44         curr = nextLevel

```

Listing 84: Problem116. Populating Next Right Pointers in Each Node

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

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.

解题方法：这道题与leetcode116的差别在于，本题中二叉树不是满树，即树中结点的左、右子结点不一定存在。leetcode116示例中的算法也可以解决本题。

```

1  # Definition for binary tree with next pointer.
2  # class TreeLinkNode(object):
3  #     def __init__(self, x):
4  #         self.val = x
5  #         self.left = None
6  #         self.right = None
7  #         self.next = None
8
9
10 class Solution(object):
11     def connect(self, root): # RT: O(n), Space: O(1)
12         if root:
13             # base step
14             nodeCurrLevel = root
15             nodeNextLevel, firstNodeNextLevel = None, None
16             while nodeCurrLevel:
17                 if nodeCurrLevel.left:
18                     if nodeNextLevel:
19                         nodeNextLevel.next = nodeCurrLevel.left
20                         nodeNextLevel = nodeCurrLevel.left
21                     if firstNodeNextLevel==None:
22                         firstNodeNextLevel = nodeNextLevel
23                 if nodeCurrLevel.right:
24                     if nodeNextLevel:
25                         nodeNextLevel.next = nodeCurrLevel.right
26                         nodeNextLevel = nodeCurrLevel.right
27                     if firstNodeNextLevel==None:
28                         firstNodeNextLevel = nodeNextLevel
29                 nodeCurrLevel = nodeCurrLevel.next
30             # recursive step
31             self.connect(firstNodeNextLevel)

```

Listing 85: Problem117. Populating Next Right Pointers in Each Node II

## 4.2 Binary Search Trees

### 4.2.1 leetcode 96. Unique Binary Search Trees

Given n, how many structurally unique BST's (binary search trees) that store values 1...n?

解题方法:

- 这题是求二叉树的棵数。这里有个解题技巧：一般来说求数量，要首先想到使用动态规划（dp），而如果要求的不只是数量，还要把所有的树都枚举出来，就要使用dfs（深度优先搜索）来遍历决策树了。参见示例图中的分析和示例。

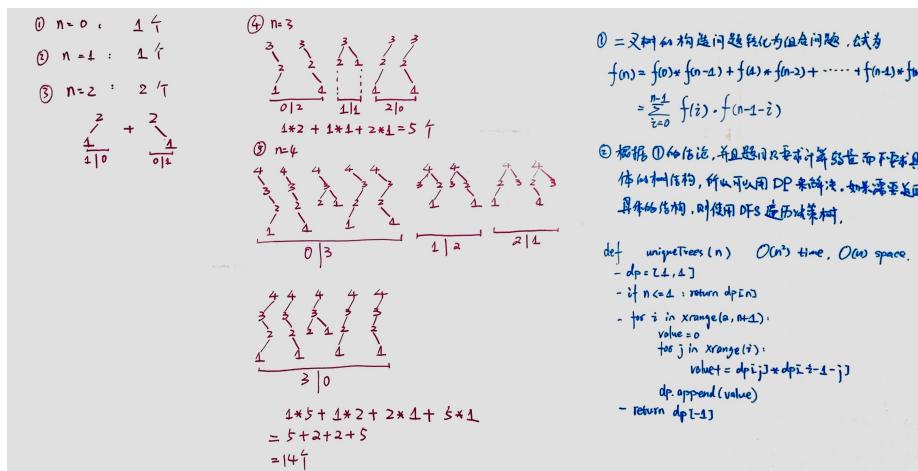
- 这道题是使用动态规划来解决的，状态转移方程：

- $n=0$ 时，为空树，那么 $dp[0]=1$ ；
- $n=1$ 时，显然也是1， $dp[1]=1$ ；
- 对于 $n \geq 2$ 时， $dp[n]=dp[0]*dp[n-1]+dp[1]*dp[n-2]+\dots+dp[n-1]*dp[0]$ ；

- 这道题背后的知识点是catalan numbers(卡特兰数)，属于组合数学的知识点。其应用范围比较广，建议阅读维基百科中的介绍，有利于扩充知识点！

- Catalan numbers satisfy the recurrence relation:

$$C_0 = 1, C_{n+1} = \sum_{i=0}^n C_i \cdot C_{n-i} \quad \text{for } n \geq 0$$



```

1  class Solution(object):
2      def numTrees_recursive(self, n): # LTE error
3          if n==0 or n==1: return 1
4          res = 0
5          for i in xrange(n):
6              res += self.numTrees(i) * self.numTrees(n-1-i)
7          return res
8
9      def numTrees_dp(self, n): # RT: O(n^2), Space: O(n) for DP
10     """
11         :type n: int
12         :rtype: int
13         """
14     dp = [1, 1]
15     if n<=1: return dp[n]
16     for i in range(2, n+1):
17

```

```

18     value = 0
19     for j in range(i):
20         value += dp[j]*dp[i-1-j]
21     dp.append(value)
22 return dp[-1]

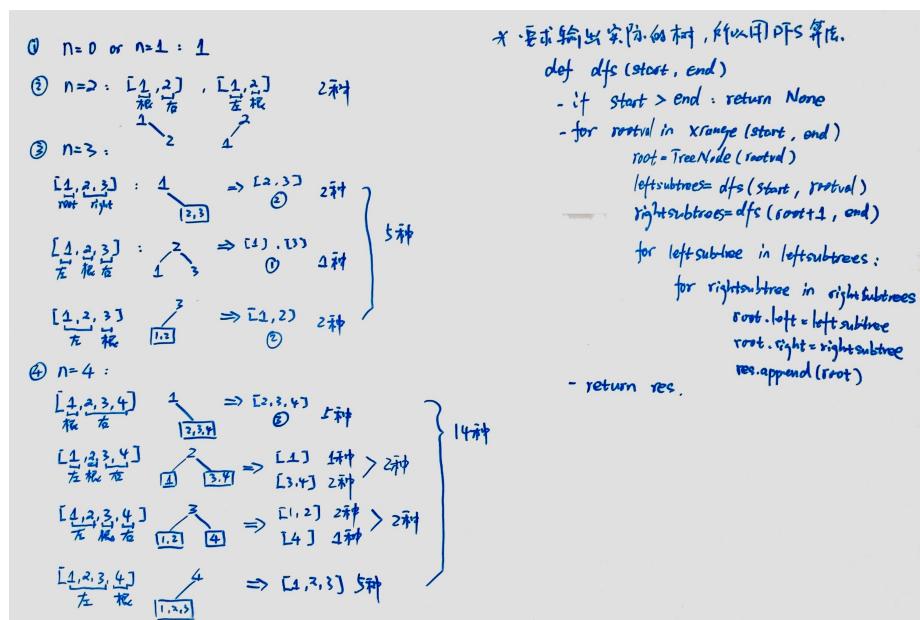
```

Listing 86: Problem96. Unique Binary Search Trees

#### 4.2.2 leetcode 95. Unique Binary Search Trees II

Given n, generate all structurally unique BST's (binary search trees) that store values 1...n.

解题方法: 典型的DFS遍历决策树输出结果的题目。参见示例图中的分析。



```

1 # Definition for a binary tree node.
2 # class TreeNode(object):
3 #     def __init__(self, x):
4 #         self.val = x
5 #         self.left = None
6 #         self.right = None
7 #
8
9 class Solution(object):
10     def generateTrees(self, n):
11         """
12             :type n: int
13             :rtype: List[TreeNode]
14         """
15         def dfs(start, end):
16             # left/right subtree is None
17             if start > end: return [None]
18             res = []
19             for rootval in xrange(start, end+1):

```

```

20     # get all possible left subtrees
21     lefttrees = dfs(start, rootval-1)
22     # get all possible right subtrees
23     righttrees = dfs(rootval+1, end)
24     # build up all possible trees
25     for lefttree in lefttrees:
26         for righttree in righttrees:
27             root = TreeNode(rootval)
28             root.left = lefttree
29             root.right = righttree
30             res.append(root)
31     return res
32
33 if n==0: return []
34 return dfs(1,n)

```

Listing 87: Problem95. Unique Binary Search Trees II

#### 4.2.3 leetcode 98. Validate Binary Search Tree

Given a binary tree, determine if it is a valid binary search tree (BST).

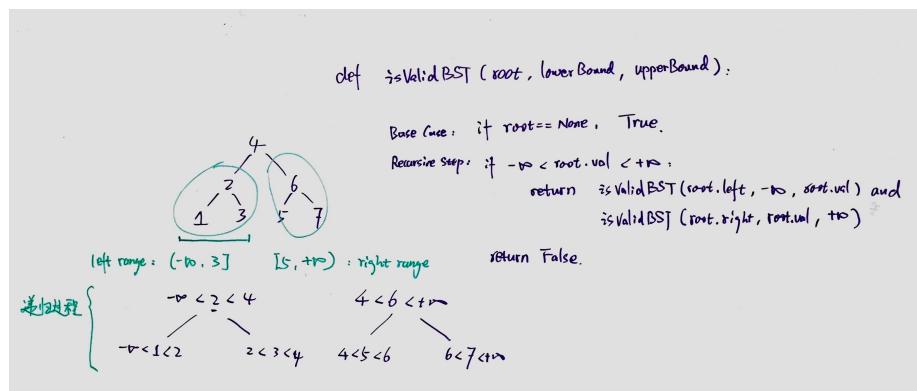
Assume a BST is defined as follows:

1. The left subtree of a node contains only nodes with keys less than the node's key.
2. The right subtree of a node contains only nodes with keys greater than the node's key.
3. Both the left and right subtrees must also be binary search trees.

解题方法: 一个比较直观的解法就是中序遍历一个BST。判断中序遍历的结果, 如果下面两种情况出现, 那么这个BST就不是合法的。

- 中序遍历的结果序列不是严格的递增序列, 或者
- 序列中存在重复元素

这道题如果使用非递归的解法, 那么就是考核中序遍历算法。关于递归算法的分析参见示例图。



```

1  # Definition for a binary tree node.
2  # class TreeNode(object):
3  #     def __init__(self, x):
4  #         self.val = x
5  #         self.left = None
6  #         self.right = None
7
8
9  class Solution(object):
10     # iterative implementation
11     def isValidBST(self, root):
12         """
13             :type root: TreeNode
14             :rtype: bool
15         """
16         if root==None:
17             return True
18         vals = self.inorderTraversal(root)
19         # case1: the result from inorder traversal is not a
20         # increasing sequence
21         # case2: the result from inorder traversal contains
22         # duplicates
23         # for the two cases above, return false
24         if (len(vals)>1 and (vals != sorted(vals))) or (len(vals)>
25             len(list(set(vals)))):
26             return False
27         return True
28
29     def inorderTraversal(self, root):
30         vals = []
31         if root:
32             stack = []
33             while stack!=[] or root:
34                 if root:
35                     stack.append(root)
36                     root = root.left
37                 else:
38                     root = stack.pop()
39                     vals.append(root.val)
40                     root = root.right
41         return vals
42
43     # recursive implementation
44     def isValidBST_recursive(self, root):
45         """
46             :type root: TreeNode
47             :rtype: bool
48         """
49         import sys
50         return self.validBST(root, -sys.maxint-1, sys.maxint)
51
52     def validBST(self, root, lower, upper):
53         if root==None: return True
54         return root.val>lower and root.val<upper and self.validBST(
55             root.left, lower, root.val) and self.validBST(root.right, root.
56             val, upper)

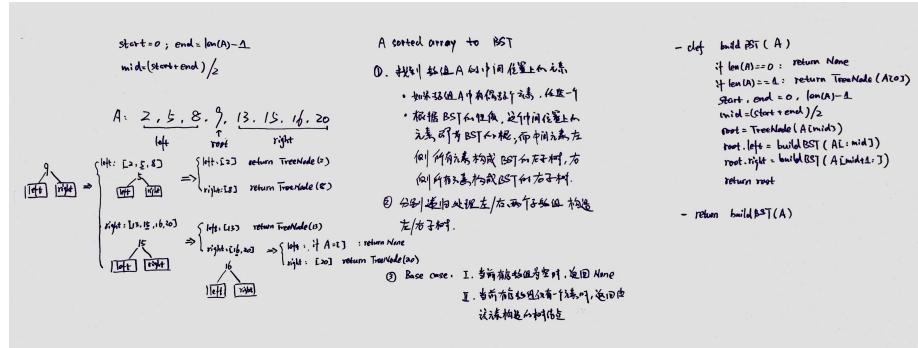
```

Listing 88: Problem98. Validate Binary Search Tree

#### 4.2.4 leetcode 108. Convert Sorted Array to Binary Search Tree

Given an array where elements are sorted in ascending order, convert it to a height balanced BST.

解题方法: 分析过程和示例见下面的示例图。



```

1 # Definition for a binary tree node.
2 # class TreeNode(object):
3 #     def __init__(self, x):
4 #         self.val = x
5 #         self.left = None
6 #         self.right = None
7 #
8
9 class Solution(object):
10     def sortedArrayToBST(self, nums): # RT: O(n)
11         """
12             :type nums: List[int]
13             :rtype: TreeNode
14         """
15         start, end = 0, len(nums)-1
16         if start > end: return None
17         if start == end: return TreeNode(nums[start])
18         mid = (start+end)/2
19         root = TreeNode(nums[mid])
20         root.left = self.sortedArrayToBST(nums[start:mid])
21         root.right = self.sortedArrayToBST(nums[mid+1:])
22         return root
  
```

Listing 89: Problem108. Convert Sorted Array to Binary Search Tree

#### 4.2.5 leetcode 109. Convert Sorted List to Binary Search Tree

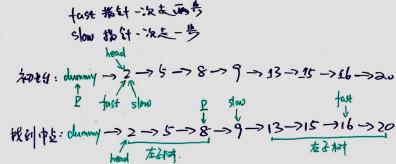
Given a singly linked list where elements are sorted in ascending order, convert it to a height balanced BST.

解题方法: 这道题与leetcode108的区别在于转换有序的链表, 找到链表的中间位置需要用快慢指针的方法在一趟遍历中确定, 所以时间复杂度增大到 $O(n)$ 。分析及示例见下面的示例图。

```

1 # Definition for singly-linked list.
2 # class ListNode(object):
3 #     def __init__(self, x):
4 #         self.val = x
5 #         self.next = None
  
```

### A SORTED LIST TO BST



- ① fast 和 slow 指针同时走，fast -> 快进两步，slow -> 一步，当 fast.next 或者 fast.next.next 为 None 时停止，此时 slow 指向中间结点。（奇偶均可行）
- ② 在移动 fast 和 slow 之前，增加一个 dummy 节点指向 head 位置，并将 p 指向 dummy。在移动 fast 和 slow 时，同步移动 p，每次向前一步，移出慢结点，p 是 slow 的前驱结点。

- ③ 从快到慢移动后，slow 指向从后至前为 BST 的根结点。
- ④ 分别单独处理由 head 和 slow.next 为头结点的左子树和右子树，构造左 / 右子树。
- ⑤ Base case： if head == None : return None  
if head.next == None : return TrueNode(head.val)

```
- def buildBST (head):
    base case
    dummy = TrueNode(0)
    dummy.next = head
    p, fast, slow = dummy, head, head
    while fast.next and fast.next.next:
        fast = fast.next.next
        slow = slow.next
        p = p.next
    p.next = None
    root = TrueNode(slow.val)
    root.left = buildBST(head)
    root.right = buildBST(slow.next)
    return root
- return buildBST(head)
```

```

4 #     def __init__(self , x):
5 #         self.val = x
6 #         self.next = None
7
8 # Definition for a binary tree node.
9 # class TreeNode(object):
10 #     def __init__(self , x):
11 #         self.val = x
12 #         self.left = None
13 #         self.right = None
14
15 class Solution(object):
16     def sortedListToBST(self , head): # RT: O(n) , Space: O(n)
17         """
18             :type head: ListNode
19             :rtype: TreeNode
20             """
21         if head==None:
22             return None
23
24         nums = []
25         while head!=None:
26             nums.append(head.val)
27             head = head.next
28
29         return self.sortedArrayToBST(sorted(nums))
30
31     def sortedArrayToBST(self , nums):
32         if len(nums) == 0:
33             return None
34         if len(nums) == 1:
35             return TreeNode(nums[0])
36         if len(nums)/2==0:
37             index = len(nums)/2-1
38         else:
39             index = len(nums)/2
40         root = TreeNode(nums[index])
41         root.left = self.sortedArrayToBST(nums[:index])
```

```

42     root.right = self.sortedArrayToBST(nums[index+1:])
43
44     return root

```

Listing 90: Problem109. Convert Sorted List to Binary Search Tree

## 4.3 Binary Search Trees Recursion

### 4.3.1 leetcode 111. Minimum Depth of Binary Tree

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.

解题思路：这道题实质上就是考查DFS遍历算法的基本知识，可以用递归实现。用两个变量分别记录当前树的最小深度和当前分支的深度；每遍历完一个分支，如果当前分支的深度小于树当前的最小深度，那么用当前分支深度更新数的最小深度。

```

1
2 # Definition for a binary tree node.
3 # class TreeNode(object):
4 #     def __init__(self, x):
5 #         self.val = x
6 #         self.left = None
7 #         self.right = None
8
9 class Solution(object):
10     def minDepth(self, root):
11         """
12             :type root: TreeNode
13             :rtype: int
14         """
15         if root==None:
16             return 0
17         if root.left==None and root.right==None:
18             return 1
19         elif root.left==None:
20             return self.minDepth(root.right)+1
21         elif root.right==None:
22             return self.minDepth(root.left)+1
23         else:
24             leftDepth = self.minDepth(root.left)
25             rightDepth = self.minDepth(root.right)
26             return min(leftDepth, rightDepth)+1

```

Listing 91: Problem111. Minimum Depth of Binary Tree

### 4.3.2 leetcode 104. Maximum Depth of Binary Tree

Given a binary tree, find its maximum depth. The maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

解题思路：这道题与leetcode111的问题类似，解决的方法类似。

```

1
2 # Definition for a binary tree node.

```

```

3 # class TreeNode(object):
4 #     def __init__(self, x):
5 #         self.val = x
6 #         self.left = None
7 #         self.right = None
8
9 class Solution(object):
10    def maxDepth(self, root):
11        """
12            :type root: TreeNode
13            :rtype: int
14        """
15        if root==None:
16            return 0
17        if root.left==None and root.right==None:
18            return 1
19        elif root.left==None:
20            return self.maxDepth(root.right)+1
21        elif root.right==None:
22            return self.maxDepth(root.left)+1
23        else:
24            return max(self.maxDepth(root.left), self.maxDepth(root
    .right))+1

```

Listing 92: Problem104. Maximum Depth of Binary Tree

#### 4.3.3 leetcode 112. Path Sum

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.

解题方法: 这道题考查的是DFS算法。根据题目的描述, 递归实现的base case部分应该判断在满足两个条件时: 当前结点是否是叶子结点, 如果是叶子结点, 那么root-to-leaf这条路径上所有结点的和是否等于特定值。如果满足上述条件的路径出现, 那么就可以结束程序, 不需要输出结果。leetcode113要求输出实际结果, 所以需要DFS整棵树。

```

1
2 # Definition for a binary tree node.
3 # class TreeNode(object):
4 #     def __init__(self, x):
5 #         self.val = x
6 #         self.left = None
7 #         self.right = None
8
9 class Solution(object):
10    def hasPathSum(self, root, sum):
11        """
12            :type root: TreeNode
13            :type sum: int
14            :rtype: bool
15        """
16        def dfs(root, sum):
17            if root==None: return False
18            if root.val==sum and root.left==None and root.right==
    None: return True
19            return dfs(root.left, sum-root.val) or dfs(root.right,
    sum-root.val)
20

```

```
21     return dfs(root, sum)
```

Listing 93: Problem112. Path Sum

#### 4.3.4 leetcode 113. Path Sum II

Given a binary tree and a sum, find all root-to-leaf paths where each path's sum equals the given sum.

解题方法: 这道题考查的是DFS算法。注意base case部分的定义。

```
1 # Definition for a binary tree node.
2 # class TreeNode(object):
3 #     def __init__(self, x):
4 #         self.val = x
5 #         self.left = None
6 #         self.right = None
7 #
8
9 class Solution(object):
10     def pathSum(self, root, sum):
11         """
12             :type root: TreeNode
13             :type sum: int
14             :rtype: List[List[int]]
15         """
16         def dfs(root, sum, valuelist):
17             if root.val==sum and root.left==None and root.right==
18                 None:
19                 res.append(valuelist)
20                 if root.left:
21                     dfs(root.left, sum-root.val, valuelist+[root.left.
22                         val])
23                 if root.right:
24                     dfs(root.right, sum-root.val, valuelist+[root.right
25                         .val])
26             res = []
27             if root==None: return res
28             dfs(root, sum, [root.val])
29         return res
```

Listing 94: Problem113. Path Sum II

#### 4.3.5 leetcode 124. Binary Tree Maximum Path Sum

Given a binary tree, find the maximum path sum. For this problem, a path is defined as any sequence of nodes from some starting node to any node in the tree along the parent-child connections. The path does not need to go through the root.

解题方法: 这道题考查的是DFS算法。关于路径最大和(maximum path sum)的解释见示例图。

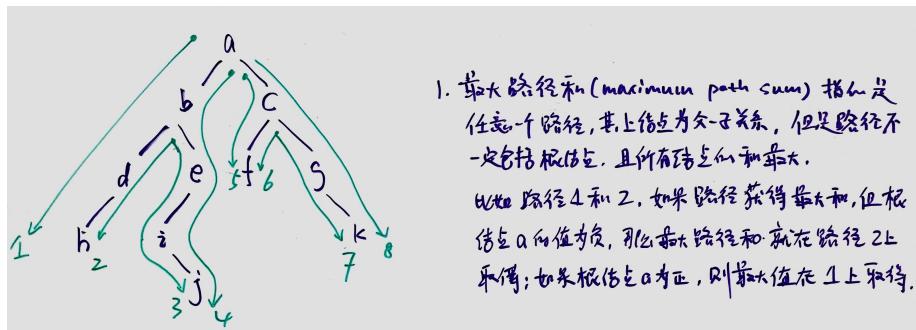
```
1 # Definition for a binary tree node.
2 # class TreeNode(object):
3 #     def __init__(self, x):
```

```

4     def __init__(self, x):
5         self.val = x
6         self.left = None
7         self.right = None
8
9     class Solution(object):
10        def maxPathSum(self, root):
11            """
12                :type root: TreeNode
13                :rtype: int
14            """
15            def dfs(root):
16                if root==None: return 0
17                sum = root.val
18                lmax = rmax = 0
19                if root.left:
20                    lmax = dfs(root.left)
21                    if lmax>0: sum += lmax
22                if root.right:
23                    rmax = dfs(root.right)
24                    if rmax>0: sum += rmax
25                self.max = max(self.max, sum)
26                return max(root.val, root.val+lmax, root.val+rmax)
27
28            if root==None: return 0
29            self.max = -10000000
30            dfs(root)
31            return self.max

```

Listing 95: Problem124. Binary Tree Maximum Path Sum



#### 4.3.6 leetcode 129. Sum Root to Leaf Numbers

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.

解题方法: 这道题考查的是DFS算法。

```

1
2 # Definition for a binary tree node.
3 # class TreeNode(object):
4 #     def __init__(self, x):
5 #         self.val = x

```

```

6     #         self.left = None
7     #         self.right = None
8
9 class Solution(object):
10    def sumNumbers(self, root):
11        """
12            :type root: TreeNode
13            :rtype: int
14        """
15        def dfs(root, sum):
16            if root.left==None and root.right==None:
17                self.totalsum += sum
18            else:
19                if root.left: dfs(root.left, sum*10+root.left.val)
20                if root.right: dfs(root.right, sum*10+root.right.
21                    val)
22
23        if root==None: return 0
24        self.totalsum = 0
25        dfs(root, root.val)
26        return self.totalsum

```

Listing 96: Problem129. Sum Root to Leaf Numbers

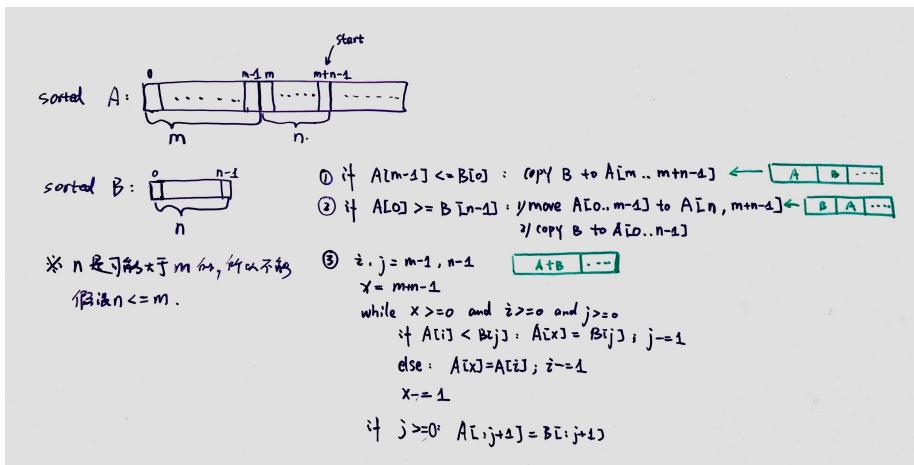
## 4.4 sort

### 4.4.1 leetcode 88. Merge Sorted Array

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

Note: You may assume that nums1 has enough space (size that is greater or equal to  $m + n$ ) to hold additional elements from nums2. The number of elements initialized in nums1 and nums2 are m and n respectively.

解题方法: 参考示例图中的分析。



```

1 class Solution(object):
2

```

```

3     def merge(self, nums1, m, nums2, n):
4         """
5             :type nums1: List[int]
6             :type m: int
7             :type nums2: List[int]
8             :type n: int
9             :rtype: void Do not return anything, modify nums1 in-place
10            instead.
11            """
12            x, y, z = m-1, n-1, m+n-1
13            while x >= 0 and y >= 0:
14                if nums1[x] < nums2[y]:
15                    nums1[z] = nums2[y]
16                    y -= 1
17                else:
18                    nums1[z] = nums1[x]
19                    x -= 1
20            z -= 1
21            if y >= 0: nums1[:y+1] = nums2[:y+1]

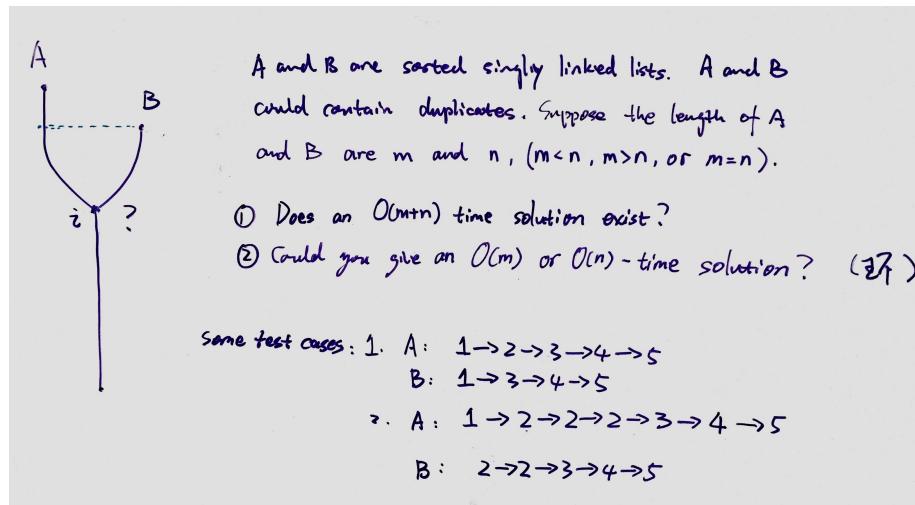
```

Listing 97: Problem88. Merge Sorted Array

#### 4.4.2 leetcode 21. Merge Two Sorted Lists

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.

解题方法: 这道题考查的就是对于链表的基本操作。这道题可以衍生出下面示例图中的一道比较难的题目。



```

1 # Definition for singly-linked list.
2 # class ListNode(object):
3 #     def __init__(self, x):
4 #         self.val = x
5 #         self.next = None
6 #
7 # class Solution(object):
8

```

```

9     def mergeTwoLists(self, l1, l2):
10    """
11        :type l1: ListNode
12        :type l2: ListNode
13        :rtype: ListNode
14    """
15        if l1==None: return l2
16        elif l2==None: return l1
17
18        dummy = ListNode(0)
19        ptr = dummy
20        while l1 and l2:
21            if l1.val<=l2.val:
22                ptr.next = l1
23                ptr = ptr.next
24                l1 = l1.next
25            else:
26                ptr.next = l2
27                ptr = ptr.next
28                l2 = l2.next
29            if l1: ptr.next = l1
30            elif l2: ptr.next = l2
31        return dummy.next

```

Listing 98: Problem21. Merge Two Sorted Lists

#### 4.4.3 leetcode 23. Merge k Sorted Lists

Merge k sorted linked lists and return it as one sorted list. Analyze and describe its complexity.

解题方法: 这道题的一种比较直观的解法是两两合并, 需要调用k-1次Merge算法, 那么时间复杂度是 $O(k \log n)$ 。效率更高的解法是模拟最小堆来实现, 时间复杂度可以减少到 $O(kn)$ 。

```

1
2 # Definition for singly-linked list.
3 # class ListNode(object):
4 #     def __init__(self, x):
5 #         self.val = x
6 #         self.next = None
7
8 class Solution(object):
9     def mergeKLists(self, lists): # use min heap
10    """
11        :type lists: List[ListNode]
12        :rtype: ListNode
13    """
14    heap = []
15    for node in lists:
16        if node:
17            heap.append((node.val, node))
18    heapq.heapify(heap)
19    dummy = ListNode(0)
20    curr = dummy
21    while heap:
22        val, node = heapq.heappop(heap)
23        curr.next = ListNode(val)
24        curr = curr.next
25        if node.next:
26            heapq.heappush(heap, (node.next.val, node.next))

```

```
27     return dummy.next
```

Listing 99: Problem23. Merge k Sorted Lists

#### 4.4.4 leetcode 147. Insertion Sort List

Sort a linked list using insertion sort.

1. traverse A  $O(n)$   
2. If A is sorted, then return.  
3. If A is not sorted, do insertion sort  
  
Insertion:  
① compare head and current node  $O(1)$ : head pointer  
② compare tail and current node  $O(1)$ : tail pointer  
③ If ① and ② not satisfied, traverse  $O(n)$   
 current sorted list

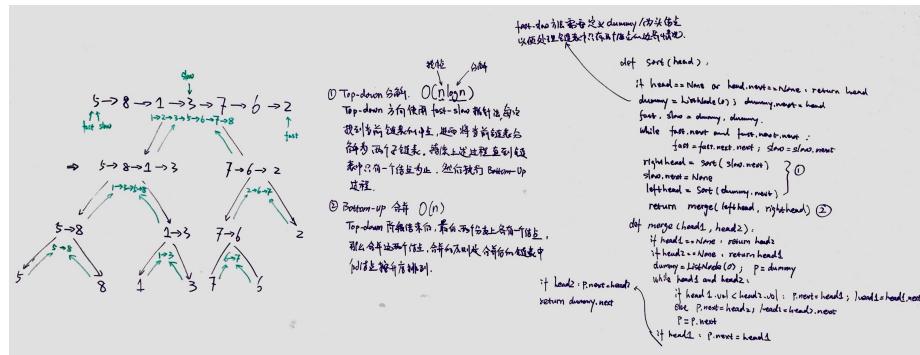
```
1 # Definition for singly-linked list.
2 # class ListNode(object):
3 #     def __init__(self, x):
4 #         self.val = x
5 #         self.next = None
6
7
8 class Solution(object):
9     def insertionSortList(self, head): # RT: O(n^2)
10        """
11            :type head: ListNode
12            :rtype: ListNode
13            """
14        if not head:
15            return head
16        dummy = ListNode(0)
17        dummy.next = head
18        curr = head
19        while curr.next:
20            if curr.next.val < curr.val:
21                pre = dummy
22                while pre.next.val < curr.next.val:
23                    pre = pre.next
24                tmp = curr.next
25                curr.next = tmp.next
26                tmp.next = pre.next
27                pre.next = tmp
28            else:
29                curr = curr.next
30        return dummy.next
```

Listing 100: Problem147. Insertion Sort List

#### 4.4.5 leetcode 148. Sort List

Sort a linked list in  $O(n \log n)$  time using constant space complexity.

解题方法：这道题考查的是合并排序在单链表上的应用。分析及示例见下面的示例图。



```

1 # Definition for singly-linked list.
2 # class ListNode(object):
3 #     def __init__(self, x):
4 #         self.val = x
5 #         self.next = None
6
7
8 class Solution(object):
9     def sortList(self, head):
10         """
11             :type head: ListNode
12             :rtype: ListNode
13         """
14         if head==None or head.next==None:
15             return head
16
17         fast = slow = head
18         while fast.next and fast.next.next:
19             fast = fast.next.next
20             slow = slow.next
21
22         head1 = head
23         head2 = slow.next
24         slow.next = None
25         head1 = self.sortList(head1)
26         head2 = self.sortList(head2)
27         head = self.merge(head1, head2)
28         return head
29
30     def merge(self, head1, head2): # RT: O(n)
31         if head1 == None: return head2
32         if head2 == None: return head1
33         dummy = ListNode(0)
34         p = dummy
35         while head1 and head2:
36             if head1.val <= head2.val:
37                 p.next = head1
38                 head1 = head1.next
39                 p = p.next
            else:

```

```

40         p.next = head2
41         head2 = head2.next
42         p = p.next
43     if head1:
44         p.next = head1
45     if head2:
46         p.next = head2
47     return dummy.next

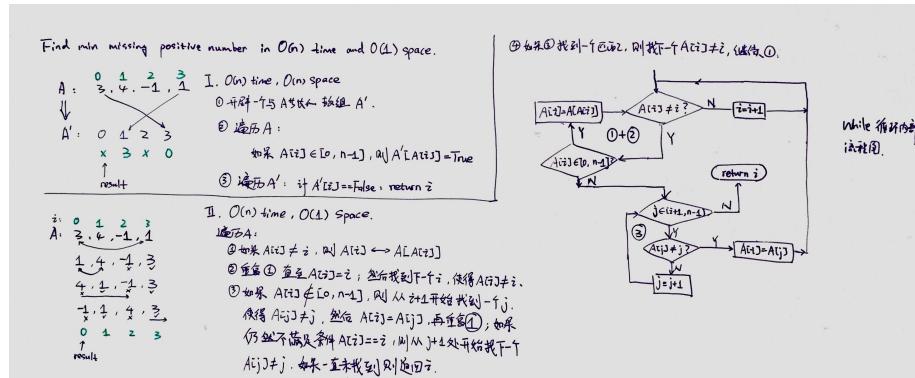
```

Listing 101: Problem148. Sort List

#### 4.4.6 leetcode 41. First Missing Positive

Given an unsorted integer array, find the first missing positive integer. For example, Given [1,2,0] return 3, and [3,4,-1,1] return 2. Your algorithm should run in  $O(n)$  time and uses constant space.

解题方法：这道题一个比较直观的解法就是先排序，然后遍历有序数组，找到缺失的最小的正整数，但是这个做法需要 $O(n \log n)$ 的时间复杂度，无法满足题目要求的 $O(n)$ 时间复杂度。分析及示例见下面的示例图。示例图中，为了便于理解第二种算法的描述，给出了一个简易程序流程图。这个流程图仅用于理解描述，因此不是非常严谨，在用流程图分析时间复杂度时容易产生误导。



```

1 class Solution(object):
2     def firstMissingPositive(self, nums):
3         """
4             :type nums: List[int]
5             :rtype: int
6         """
7
8         if len(nums)==0: return 1
9         n = len(nums)
10        table = [False for _ in range(n+1)]
11        for i in range(n):
12            if 1<=nums[i]<=n:
13                table[nums[i]] = True
14        for i in range(1, n+1):
15            if table[i]==False:
16                return i
17        return len(table)

```

Listing 102: Problem41. First Missing Positive

#### 4.4.7 leetcode 75. Sort Colors

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. Note: You are not suppose to use the library's sort function for this problem.

解题方法: 首先遍历数组, 对red, white, blue分别计数; 然后再遍历一遍数组, 根据red, white, blue的数量, 重新初始化数组, 使得数组第一部分都是red, 中间部分都是white, 最后部分都是blue。这个方法的时间复杂度是 $O(n)$ 。

```
1 class Solution(object):
2     def sortColors(self, nums):
3         """
4             :type nums: List[int]
5             :rtype: void Do not return anything, modify nums in-place
6             instead.
7             """
8         if len(nums)>0:
9             red = white = blue = 0
10            for num in nums:
11                if num==0: red+=1
12                elif num==1: white+=1
13                elif num==2: blue+=1
14            for i in range(len(nums)):
15                if i<=red-1: nums[i]=0
16                elif i<=red+white-1: nums[i]=1
17                elif i<=red+white+blue-1: nums[i]=2
```

Listing 103: Problem75. Sort Colors

## 5 Search

### 5.1 leetcode 34. Search for a Range

Given a sorted array of integers, find the starting and ending position of a given target value. Your algorithm's runtime complexity must be in the order of  $O(\log n)$ . If the target is not found in the array, return [-1, -1].

For example, Given [5, 7, 7, 8, 8, 10] and target value 8, return [3, 4].

解题方法: 题目要求的时间复杂度是 $O(\log n)$ , 所以思考的方向就是二叉搜索算法。先用二叉搜索找到target在数组中的起始点, 然后从起始点开始再用二叉搜索找到target在数组中出现的最后位置。也就是进行两次二叉搜索。除了运用两次二叉搜索算法, 也可以用DivideConquer的方法解决。

```
1 class Solution(object):
2     def searchRange_BS(self, nums, target):
3         """
4             :type nums: List[int]
5             :type target: int
6             """
```

```

7     :rtype: List[int]
8
9     begin = end = -1
10    # find the first occurrence of target in nums
11    front, back = 0, len(nums)-1
12    while front <= back:
13        mid = (front+back)/2
14        if nums[mid] < target: front = mid+1
15        elif nums[mid] > target: back=mid-1
16        else: begin = mid; back = mid-1
17    # if begin is -1, it means there is no target in nums
18    if begin===-1: return [-1,-1]
19    # find the last occurrence of target in nums
20    front, back = begin, len(nums)-1
21    while front <= back:
22        mid = (front+back)/2
23        if nums[mid] < target: front = mid+1
24        elif nums[mid] > target: back = mid-1
25        else: end = mid; front = mid+1
26    return [begin, end]
27
28 def searchRange_DC(self, nums, target):
29     def search(start, end):
30         if nums[start]==target==nums[end]:
31             return [start, end]
32         if nums[start]<=target<=nums[end]:
33             mid = (start+end)/2
34             left, right = search(start, mid), search(mid+1, end)
35             return max(left, right) if -1 in left+right else [
36                 left[0], right[1]]
37             return [-1,-1]
38         return search(0, len(nums)-1)

```

Listing 104: Problem34. Search for a Range

## 5.2 leetcode 35. Search Insert Position

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

二叉搜索算法的左侧边界是值得关注的地方。

```

1
2 class Solution(object):
3     def searchInsert(self, A, target):
4         left = 0; right = len(A) - 1
5         while left <= right:

```

```

6     mid = ( left + right ) / 2
7     if A[mid] < target:
8         left = mid + 1
9     elif A[mid] > target:
10        right = mid - 1
11    else:
12        return mid
13    return left

```

Listing 105: Problem35. Search Insert Position

### 5.3 leetcode 74. Search a 2D Matrix

Write an efficient algorithm that searches for a value in an  $m \times n$  matrix. This matrix has the following properties:

1. Integers in each row are sorted from left to right.
2. The first integer of each row is greater than the last integer of the previous row.

For example, Consider the following matrix:  $\begin{bmatrix} 1 & 3 & 5 & 7 \\ 10 & 11 & 16 & 20 \\ 23 & 30 & 34 & 50 \end{bmatrix}$

Given target = 3, return true.

解题思路：参考示例图中的分析。

$$A = \left( \begin{array}{cccc} 1 & 3 & 5 & 7 \\ 10 & 11 & 16 & 20 \\ 23 & 30 & 34 & 50 \end{array} \right) \quad t = 16$$

- ① 首先在  $A[i][0]$  上做 Binary Search，满足条件： $A[mid-1][0] < target \leq A[mid][0]$  时停止搜索
- ② 然后在  $A[mid][j]$  上做 Binary Search，如果找到返回 True，否则 False

```

1
2 class Solution(object):

```

```

3     def searchMatrix(self, matrix, target): # RT: O(log(m+n))
4         """
5             :type matrix: List[List[int]]
6             :type target: int
7             :rtype: bool
8         """
9         m, n = len(matrix), len(matrix[0])
10        start, end = 0, m*n-1
11        while start<=end:
12            mid = (start+end)/2
13            # position = row*n+col
14            row, col = mid/n, mid%n
15            if matrix[row][col]<target:
16                start = mid+1
17            elif matrix[row][col]>target:
18                end = mid-1
19            else: return True
20        return False

```

Listing 106: Problem74. Search a 2D Matrix

#### 5.4 leetcode 240. Search a 2D Matrix II

Write an efficient algorithm that searches for a value in an  $m \times n$  matrix. This matrix has the following properties:

1. Integers in each row are sorted in ascending from left to right.
2. Integers in each column are sorted in ascending from top to bottom.

$$\begin{bmatrix} 1 & 4 & 7 & 11 & 15 \\ 2 & 5 & 8 & 12 & 19 \\ 3 & 6 & 9 & 16 & 22 \\ 10 & 13 & 14 & 17 & 24 \\ 18 & 21 & 23 & 26 & 30 \end{bmatrix}$$

For example, Consider the following matrix:

Given target = 5, return true. Given target = 20, return false.

解题思路:

- 第一种解法: 从矩阵的右上角开始, 执行两重循环: 外循环递增枚举每行, 内循环递减枚举列。
- 第二种解法: 将矩阵按中心点划分为四个区域, 分治法, 以矩形中点为基准, 将矩阵拆分成左上, 左下, 右上, 右下四个区域: 若中点值<目标值, 则舍弃左上区域, 从其余三个区域再行查找; 若中点值>目标值, 则舍弃右下区域, 从其余三个区域再行查找。时间复杂度递推式:  $T(n) = 3T(n/2) + c$ 。

参见下面的示例图。

```

1 class Solution(object):
2     def searchMatrix1(self, matrix, target): # RT: O(m+n)
3         """
4             :type matrix: List[List[int]]
5             :type target: int
6             :rtype: bool
7         """
8

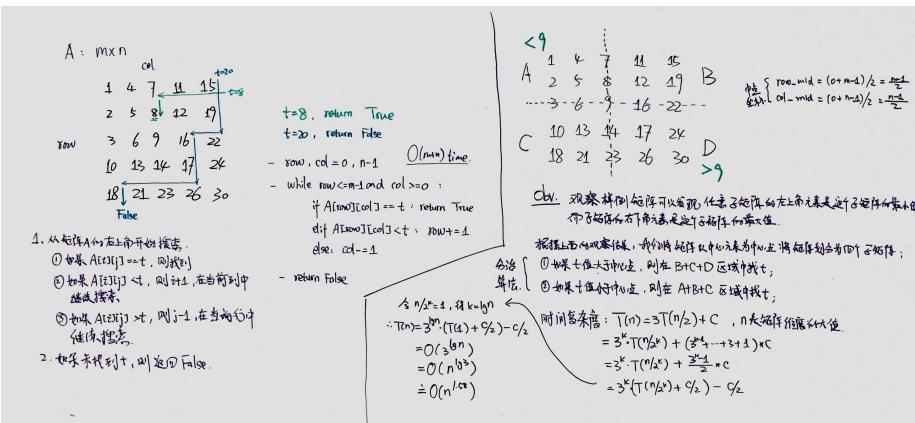
```

```

9     m, n = len(matrix), len(matrix[0])
10    y = n-1
11    for x in range(m):
12        while y >= 0 and matrix[x][y] > target:
13            y -= 1
14        if matrix[x][y] == target:
15            return True
16    return False
17
18 def searchMatrix(self, matrix, target): # RT: O(n^1.58)
19     """
20         :type matrix: List[List[int]]
21         :type target: int
22         :rtype: bool
23     """
24     def helper(matrix, rowStart, rowEnd, colStart, colEnd, target):
25         if rowStart > rowEnd or colStart > colEnd: return False
26         rowMid = (rowStart+rowEnd)/2
27         colMid = (colStart+colEnd)/2
28         if matrix[rowMid][colMid] > target:
29             return helper(matrix, rowStart, rowMid-1, colStart,
30                           colMid-1, target) or
31                           helper(matrix, rowMid, rowEnd, colStart, colMid-1,
32                           target) or
33                           helper(matrix, rowStart, rowMid-1, colMid, colEnd,
34                           target)
35         elif matrix[rowMid][colMid] < target:
36             return helper(matrix, rowMid+1, rowEnd, colMid+1,
37                           colEnd, target) or
38             helper(matrix, rowMid+1, rowEnd, colStart, colMid,
39                           target) or
40             helper(matrix, rowStart, rowMid, colMid+1, colEnd,
41                           target)
42         else: return True
43     m, n = len(matrix), len(matrix[0])
44     return helper(matrix, 0, m-1, 0, n-1, target)

```

Listing 107: Problem240. Search a 2D Matrix II



## 6 DFS

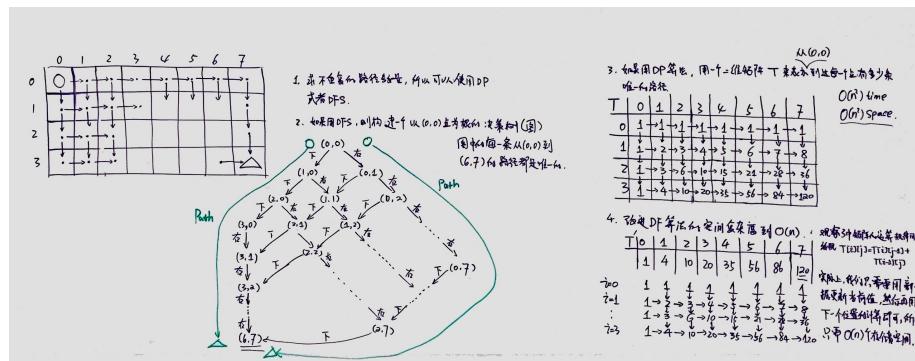
### 6.1 leetcode 62. Unique Paths

A robot is located at the top-left corner of a  $m \times n$  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? Note: m and n will be at most 100.

解题方法：这里给出三种算法，分别在时间复杂度和空间复杂度上进行改进。

1. 第一种方法是决策树的DFS算法。这类问题与N-Queens问题统称为递归回溯问题。算法的起点位置是从终点开始进行递归计算：到达(m,n)坐标路径有两种： $(m,n-1) \rightarrow (m,n)$ ，或者 $(m-1,n) \rightarrow (m,n)$ ；到达 $(m,n-1)$ 和 $(m-1,n)$ 的路径又分别有两种，即 $(m,n-2) \rightarrow (m,n-1)$ 和 $(m-1,n-2)$ ， $(m-1,n-1) \rightarrow (m-1,n)$ 和 $(m-2,n)$ ，以此类推。结束条件就是到达起始位置(1,1)，而到达(1,1)点的路径只有一条。DFS算法的缺点是存在重复计算，造成超时问题。
2. 第二种方法是DP算法，针对重复计算的部分加以改进，降低时间复杂度，为此增加额外的空间存储计算过的结果。空间复杂度为 $O(n^2)$ 。 $dp[i][j]$ 表示从 $(0,0)$ 到达 $(i,j)$ 的路径数目。转换方程： $dp[x][y] = dp[x-1][y] + dp[x][y-1]$ ，初始值 $dp[0][0] = 1$ 。
3. 第三种方法从空间复杂度方面改进第二种算法，将空间复杂度降低到 $O(n)$ 。

三种算法的示例和分析见示例图。



```
1 class Solution(object):
2     def uniquePaths_dfs(self, m, n): # TLE error
3         """
4             :type m: int
5             :type n: int
6             :rtype: int
7             """
8
9         if m<1 or n<1: return 0
10        if m==1 or n==1: return 1
11        return self.uniquePaths(m-1,n) + self.uniquePaths(m,n-1)
12
```

```

13     def uniquePaths_dp1(self, m, n): # RT: O(n^2), Space: O(n^2)
14         dp = [[0]*n for _ in range(m)]
15         dp[0][0] = 1
16         for x in range(m):
17             for y in range(n):
18                 if x+1<m:
19                     dp[x+1][y] += dp[x][y]
20                 if y+1<n:
21                     dp[x][y+1] += dp[x][y]
22
23         return dp[m-1][n-1]
24
25     def uniquePaths_dp2(self, m, n): # RT: O(n^2), Space: O(n)
26         if m<n: m,n = n,m
27         dp = [0 for _ in range(n)]
28         dp[0] = 1
29         for x in range(m):
30             for y in range(n-1):
31                 dp[y+1] += dp[y]
32         return dp[n-1]

```

Listing 108: Problem62. Unique Paths

## 6.2 leetcode 63. Unique Paths II

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. Note: m and n will be at most 100.

解题方法：这道题对leetcode62 Unique Paths增加了一些条件来增大难度。但是，实际上还是可以使用leetcode62的解法，只是在DFS的分支决策前，或DP算法累加路径数量前，增加判断条件。使用DP算法的辅助矩阵dp[i][j]表示从(0,0)到(i,j)的路径数目，转换方程:  $dp[x][y] = dp[x-1][y] + dp[x][y-1]$ , 初始值 $dp[0][0]=1$ 。分析可以参考leetcode62解答中的示例图。

```

1  class Solution(object):
2      # DP solution
3      def uniquePathsWithObstacles_dp(self, obstacleGrid):
4          """
5              :type obstacleGrid: List[List[int]]
6              :rtype: int
7              """
8
9          m,n = len(obstacleGrid), len(obstacleGrid[0])
10         dp = [[0]*n for _ in range(m)]
11         dp[0][0] = 0 if obstacleGrid[0][0] == 1 else 1
12         for x in range(m):
13             for y in range(n):
14                 if x+1<m and obstacleGrid[x+1][y] != 1:
15                     dp[x+1][y] += dp[x][y]
16                 if y+1<n and obstacleGrid[x][y+1] != 1:
17                     dp[x][y+1] += dp[x][y]
18
19         return dp[m-1][n-1]
20
21     # DFS solution
22     def uniquePathsWithObstacles(self, obstacleGrid): # TLE error
23         m,n = len(obstacleGrid), len(obstacleGrid[0])
24         Solution.cache = [[0]*(n+1) for _ in range(m+1)]

```

```

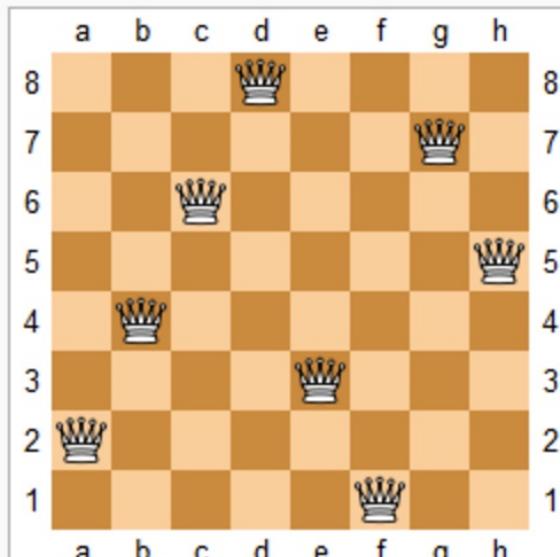
24     if obstacleGrid[0][0]: return 0
25     if m==1 and n==1: return 1
26     return self.dfs(m,n,obstacleGrid)
27
28     def dfs(self , m, n, obstacleGrid):
29         if m<1 or n<1: return 0
30         if obstacleGrid[m-1][n-1]==1: return 0
31         if m==1 and n==1: return 1
32         if Solution.cache[m][n]>0: return Solution.cache[m][n]
33         else:
34             Solution.cache[m][n] = self.dfs(m-1,n,obstacleGrid) +
35             self.dfs(m,n-1,obstacleGrid)
            return Solution.cache[m][n]

```

Listing 109: Problem63. Unique Paths II

### 6.3 leetcode 51. N-Queens

The n-queens puzzle is the problem of placing  $n$  queens on an  $n \times n$  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.



One solution to the eight queens puzzle

解题思路：这类问题统称为递归回溯问题，也可以叫做对决策树的深度优先搜索（dfs）。N皇后问题有个技巧的关键在于棋盘的表示方法，这里使用一个数组board就可以表达了， $board[i]=j$ 表示棋盘的第*i*行的皇后放在了第*j*列上：比如 $board=[1, 3, 0, 2]$ ，这是4皇后问题的一个解，意思是：在第0行，皇后放在第1列；在第1行，皇后放在第3列；在第2行，皇后放在第0列；在第3行，皇后放在第2列。这道题提供一个递归解法，下道题使用非递归。`check`函数用来检查在第*k*行，皇后是否可以放置在第*j*列。

```

1  class Solution(object):
2      def solveNQueens(self, n):
3          """
4              :type n: int
5              :rtype: List[List[str]]
6          """
7          def check(k,y): # check if the k-th Queen can be put in
8              column j
9                  # check the Queenss positions in first k rows
10                 for x in range(k):
11                     # the first condition checks if the j-th column is
12                     available
13                     # the second condition checks if the diagonal
14                     places are available
15                         if board[x]==y or abs(k-x)==abs(board[x]-y):
16                             return False
17                         return True
18
19             def dfs(row, valuelist):
20                 if row==n: res.append(valuelist)
21                 else:
22                     # iterate n columns
23                     for col in range(n):
24                         if check(row, col):
25                             board[row] = col
26                             s = '.' * n
27                             dfs(row+1, valuelist+[s[:col]+ 'Q'+s[col
28                             +1:]])
29
30             board = [-1 for _ in range(n)]
31             res = []
32             dfs(0, [])
33             return res

```

Listing 110: Problem51. N-Queens

## 6.4 leetcode 52. N-Queens II

Follow up for N-Queens problem. Now, instead outputting board configurations, return the total number of distinct solutions.

解题思路：这道题的算法和N-Queens I的算法一样使用DFS深度遍历决策树，但是本题的算法要简单些，因为不需要输出每一种棋盘的布局。

```

1  class Solution(object):
2      def totalNQueens(self, n): # recursion+backtracking
3          """
4              :type n: int
5              :rtype: int
6          """
7          def check(k,j):
8              for i in range(k):
9                  if board[i]==j or abs(i-k)==abs(board[i]-j):
10                      return False
11              return True
12
13          def dfs(depth):
14

```

```

15         if depth == n: self.count += 1
16     else:
17         for i in range(n):
18             if check(depth, i):
19                 board[depth] = i
20                 dfs(depth+1)
21
22     self.count = 0
23     board = [0 for _ in range(n)]
24     dfs(0)
25     return self.count

```

Listing 111: Problem52. N-Queens II

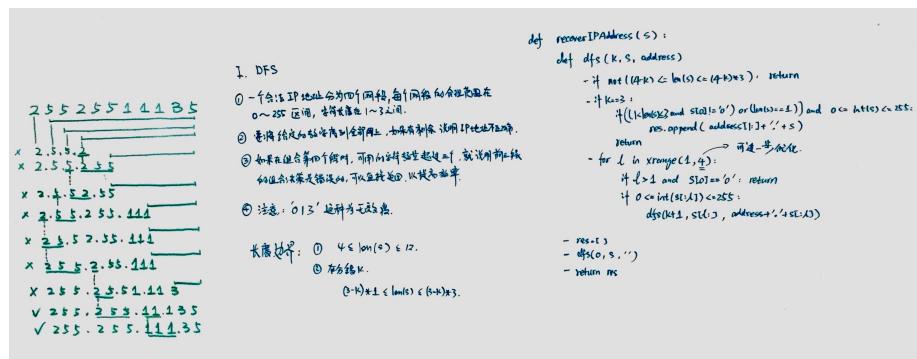
## 6.5 leetcode 93. Restore IP Addresses

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)

解题思路：这道题考查的是回溯算法。IP地址可以划分为四段，在每个段上，字符串的长度在[1,3]这个区间，即段号在[0,255]这个区间。使用DFS算法遍历决策树时，DFS内部判断条件的定义会影响决策深度：

1. 第一个条件(line 9)是判断当前字符串的总长度是否在有效区间，即保证剩余各段上至少可以分配到一个字符，而至多可以分配到三个字符；少于或者多于这个区间规定的字符数，都无法取得合法IP地址，因此也就没必要再进行后续的尝试。
2. 第二个条件(line 10)是判断递归是否到达了最后一个段位。如果已经处于最后一个段位，那么要么找到合法的IP地址，要么就不合法，然后返回到上一层，无需在进入下一层递归。
3. 第三个条件(line 14)是判断在每一个段位需要搜索几次。当前代码是固定搜索三次，实际可以进一步优化。



```

1 class Solution(object):
2

```

```

3     def restoreIpAddresses(self, s):
4         """
5             :type s: str
6             :rtype: List[str]
7         """
8         def dfs(k, s, address):
9             if not ((4-k) <= len(s) <= (4-k)*3): return
10            if k == 3:
11                if ((1 < len(s) <= 3 and s[0] != '0') or len(s) == 1) and
12                    0 <= int(s) <= 255:
13                    res.append(address[1:] + '.' + s)
14                return
15                for l in xrange(1, 4):
16                    if l > 1 and s[0] == '0': return
17                    if 0 <= int(s[:l]) <= 255:
18                        dfs(k + 1, s[l:], address + '.' + s[:l])
19            res = []
20            dfs(0, s, '')
21        return res

```

Listing 112: Problem93. Restore IP Addresses

## 6.6 leetcode 39. Combination Sum

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  $(a_1, a_2, \dots, a_k)$  must be in non-descending order. (ie,  $a_1 \leq a_2 \leq \dots \leq a_k$ ). 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].

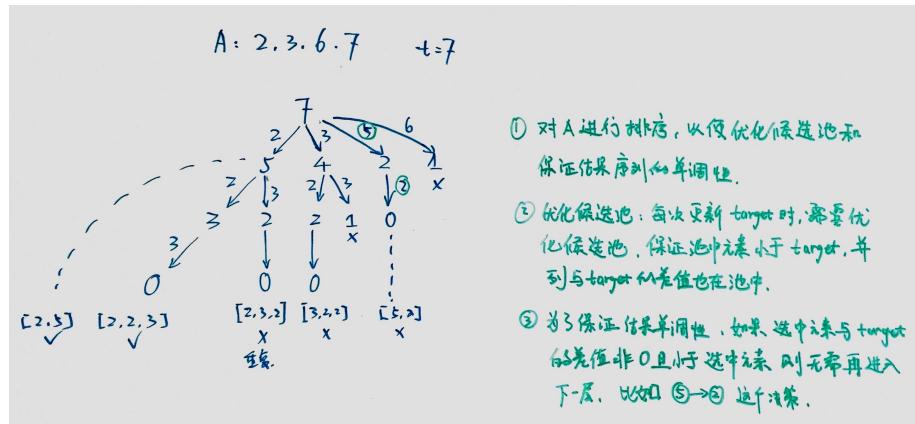
解题思路：本题的题目告知同一个数字可以使用多次，也就是在构建决策树时，候选池中候选数字的数量保持不变。另外题目要求输出所有符合条件的组合，所以可以基本肯定的是使用DFS算法遍历决策树。针对这道题，DFS递归算法的基本条件有两种定义方式：一种是累加和等于目标值target时，输出组合；另一种是target递减到0时，输出组合。

```

1 class Solution(object):
2     def combinationSum(self, candidates, target):
3         """
4             :type candidates: List[int]
5             :type target: int
6             :rtype: List[List[int]]
7         """
8         def dfs(t, start, valuelist):
9             if t == 0: res.append(valuelist)
10            for i in range(start, len(candidates)):
11                if candidates[i] <= t:
12                    dfs(t - candidates[i], i, valuelist + [candidates[i]])
13            candidates.sort()
14            res = []
15            dfs(target, 0, [])
16        return res

```

Listing 113: Problem39. Combination Sum



## 6.7 leetcode 40. Combination Sum II

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 ( $a_1, a_2, \dots, a_k$ ) must be in non-descending order. (ie,  $a_1 \leq a_2 \leq \dots \leq a_k$ ). 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].

解题思路：这道题考查的是回溯算法，利用DFS遍历决策树。与leetcode 39 Combination Sum I不同，本题的候选数字中是存在重复元素的，因此为了满足结果唯一性这个条件，需要对base step增加去重的判断条件(line 11)。

```

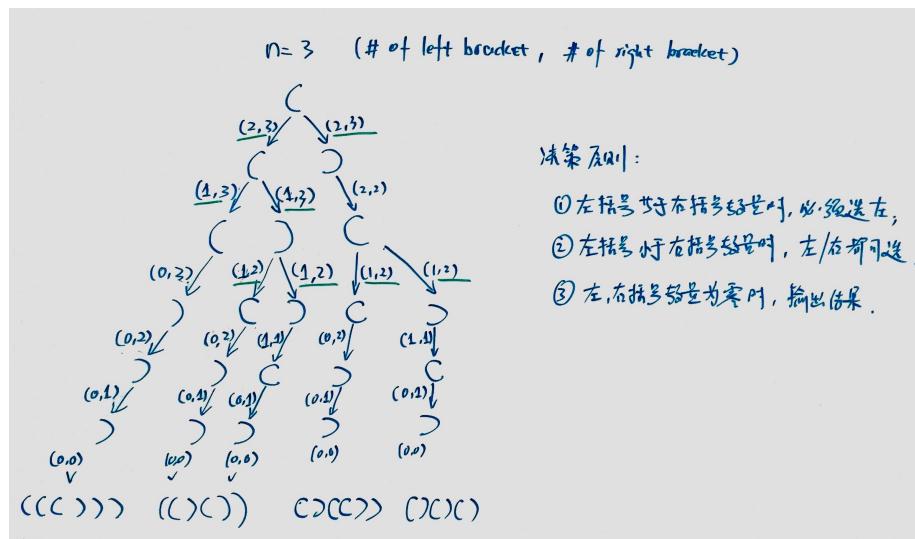
1  class Solution(object):
2      def combinationSum2(self, candidates, target):
3          """
4              :type candidates: List[int]
5              :type target: int
6              :rtype: List[List[int]]
7          """
8
9          def dfs(nums, t, start, valuelist):
10             if t==0:
11                 if valuelist not in res:
12                     return res.append(valuelist)
13             else:
14                 for i in range(start, len(nums)):
15                     if t<nums[i]: return
16                     dfs(nums, t-nums[i], i+1, valuelist+[nums[i]])
17
18         candidates.sort()
19         res = []
20         dfs(candidates, target, 0, [])
21         return res
  
```

Listing 114: Problem40. Combination Sum II

## 6.8 leetcode 22. Generate Parentheses

Given  $n$  pairs of parentheses, write a function to generate all combinations of well-formed parentheses. For example, given  $n = 3$ , a solution set is: "((()))", "(()())", "((())()", "()(())", "()()()

解题思路：这道题考查的是回溯算法，利用DFS遍历决策树。每一次的决策就是选左括号还是选右括号，决策的原则及示例图。



```

1  class Solution(object):
2      def generateParenthesis(self, n):
3          """
4              :type n: int
5              :rtype: List[str]
6          """
7          def dfs(left, right, valuelist):
8              if left<0 or right<0: return
9              if left==right==0: res.append(valuelist)
10             if left==right:
11                 dfs(left-1, right, valuelist+'(')
12             if left<right:
13                 dfs(left-1, right, valuelist+')')
14                 dfs(left, right-1, valuelist+')')
15             res = []
16             dfs(n,n,'')
17         return res
18

```

Listing 115: Problem22. Generate Parentheses

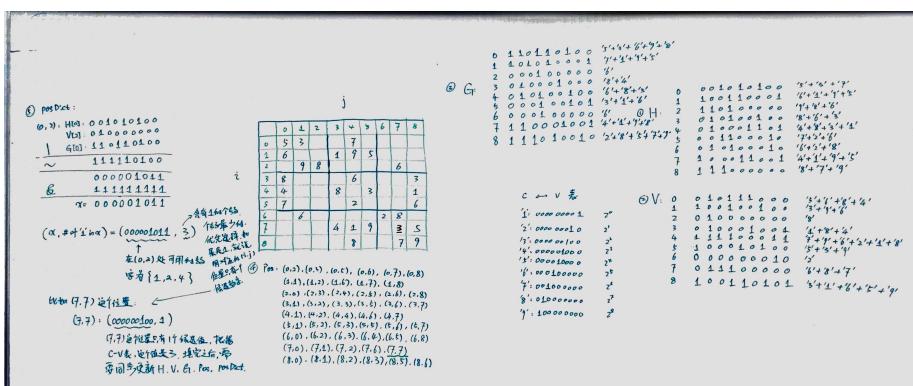
## 6.9 leetcode 37. Sudoku Solver

Write a program to solve a Sudoku puzzle by filling the empty cells. Empty cells are indicated by the character '.'. You may assume that there will be only one unique solution.

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

解题思路：这里给出两种不同的算法：

- 第一种算法是DFS遍历决策树。
    - 算法首先将当前棋盘的布局分别按照行、列、9空格保存到三个数组中(line 28 36)。这样的做法方便在其后的计算中，可以O(1)的时间复杂度验证候选数字是否置于(x,y)位置(line 14)。
    - 然后递归遍历决策树(line 20)。在决策失败的情况下，恢复之前的棋盘变更(line23 24)，再选择其他的候选数字(line10 12)。
  - 第二种算法是利用位运算（太神奇了）。这个算法的分析见后面的示例图。



```
1
2 class Solution(object):
3     def solveSudoku(self, board):
4         """
5             :type board: List[List[str]]
6             :rtype: void Do not return anything, modify board in-place
7             instead.
8         """
9         # use hash tables to save the existing numbers in current
10        board
11        def dfs(board):
```

```

10     for x in xrange(9):
11         for y in xrange(9):
12             if board[x][y]=='.':
13                 for c in xrange(9):
14                     if usedRows[x][c] or usedCols[y][c] or
15                         usedBlocks[3*(x/3)+(y/3)][c]:
16                         continue
17                         board[x][y] = str(c+1)
18                         usedRows[x][c] = usedCols[y][c] =
19                         usedBlocks[3*(x/3)+(y/3)][c] = True
20
21                         # recursive step
22                         if dfs(board): return True
23
24                         # rollback the current change
25                         board[x][y] = '.'
26                         usedRows[x][c] = usedCols[y][c] =
27                         usedBlocks[3*(x/3)+(y/3)][c] = False
28                         return False
29
30                         return True
31
32                         usedRows = [[0]*9 for _ in xrange(9)]
33                         usedCols = [[0]*9 for _ in xrange(9)]
34                         usedBlocks = [[0]*9 for _ in xrange(9)]
35                         for x in xrange(9):
36                             for y in xrange(9):
37                                 if board[x][y]=='.': continue
38                                 c = int(board[x][y])-1
39                                 usedRows[x][c] = usedCols[y][c] = usedBlocks[3*(x
40 /3)+(y/3)][c] = True
41                                 dfs(board)
42
43
44
45
46     # bit-wise operation implementation: just 52ms!!!
47     def solveSudoku(self, board):
48         """
49             :type board: List[List[str]]
50             :rtype: void Do not return anything, modify board in-place
51             instead.
52             """
53             pos,H,V,G=[],[0]*9,[0]*9,[0]*9 #Empty cells'position,
54             horizontal,vertical,grid
55             ctoV={str(i):1<<(i-1) for i in range(1,10)} #eg:'4'=>1000
56             self.vtoC={1<<(i-1):str(i) for i in range(1,10)} #eg
57             :100=>'3'
58             for i, row in enumerate(board):
59                 for j, c in enumerate(row):
60                     if c=='.':
61                         v = ctoV[c]
62                         H[i],V[j],G[i/3*3+j/3] = H[i]|v,V[j]|v,G[i/3*3+
63                         j/3]|v
64                     else:
65                         pos+=(i,j),
66                         #dict {((i,j):(possible vals(bit-identify),count)}
67                         posDict={(i,j):(x, self.countOnes(x)) for i,j in pos \
68                         for x in [0x1ff & ~ (H[i]|V[j]|G[i/3*3+j/3])]}
69                         self.solve(board, posDict)
70
71     def countOnes(self, n):
72         count=0
73         while n:

```

```

64             count ,n = count+1 , n & ~( n& (~n + 1))
65         return count
66
67     def solve(self ,board ,posDict):
68         if len(posDict)==0:
69             return True
70         p = min(posDict .keys() , key= lambda x: posDict [x][1]) #
71         candidate=posDict [p][0]
72         while candidate:
73             v=candidate & (~candidate + 1) #get last '1'
74             candidate &= ~v
75             tmp=self .updata(board ,posDict ,p,v) #updata board and
76             posDict
77             if self .solve(board ,posDict): #solve next position
78                 return True
79             self .recovery(board ,posDict ,p,v,tmp) #backtrack-->
80             recovery
81             return False
82
83     def updata(self ,board ,posDict ,p,v):
84         i ,j=p[0] ,p[1]
85         board [i ][j]=self .vtoC[v]
86         tmp=[posDict [p]]
87         del posDict [p]
88         for key in posDict .keys():
89             if i == key[0] or j ==key[1] or (i /3,j /3)==(key [0]/3 ,
90             key [1]/3) : #relevant points
91                 if posDict [key ][0]&v: #need modify
92                     posDict [key ][0]&= ~v
93                     posDict [key ][1]-= 1
94                     tmp+=key , #Record these points .
95         return tmp
96
97     def recovery (self ,board ,posDict ,p,v,tmp):
98         board [p[0]][p[1]]='.'
99         posDict [p]=tmp[0]
100        for key in tmp[1:]:
101            posDict [key ][0]]=v
102            posDict [key ][1]+=1

```

Listing 116: Problem37. Sudoku Solver

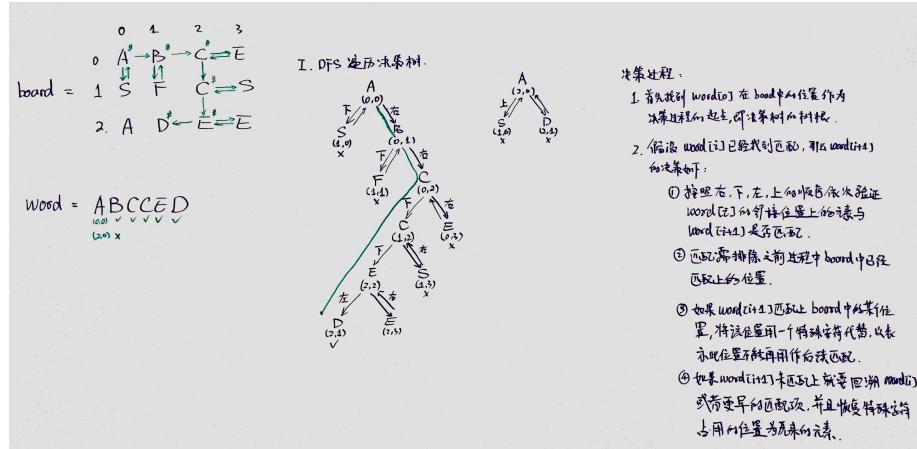
## 6.10 leetcode 79. Word Search

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 =  $\begin{bmatrix} 'A' & 'B' & 'C' & 'E' \\ 'S' & 'F' & 'C' & 'S' \\ 'A' & 'D' & 'E' & 'E' \end{bmatrix}$

- word = "ABCED", -> returns true,
- word = "SEE", -> returns true,
- word = "ABCB", -> returns false.

解题思路：本题使用DFS遍历决策树。在word中某个字符与某个格子中的字符匹配时，下一个决策步骤有四种选择，即当前格子相邻接的上、下、左、右四个格子；在做出决策进入下一步前，需要判断所选择的格子是否已经被用过（下面的算法使用特殊字符“#”作为占位符，表示格子已被使用）：如果已经被用过，那么放弃当前决策，选择其他邻接格子；如果未被使用，则继续DFS决策过程，直到找到在格子中找到所有完全匹配的字符。



```

1  class Solution(object):
2      def exist(self, board, word):
3          """
4              :type board: List[List[str]]
5              :type word: str
6              :rtype: bool
7          """
8
9          def dfs(x, y, word):
10             if len(word)==0: return True
11             else:
12                 # up
13                 if x>0 and board[x-1][y]==word[0]:
14                     tmp = board[x][y]
15                     board[x][y] = '#'
16                     if dfs(x-1,y,word[1:]):
17                         return True
18                     board[x][y]=tmp
19
20                 # down
21                 if x<len(board)-1 and board[x+1][y]==word[0]:
22                     tmp = board[x][y]
23                     board[x][y] = '#'
24                     if dfs(x+1,y,word[1:]):
25                         return True
26                     board[x][y]=tmp
27
28                 # left
29                 if y>0 and board[x][y-1]==word[0]:
30                     tmp = board[x][y]
31                     board[x][y] = '#'
32                     if dfs(x,y-1,word[1:]):
33                         return True
34                     board[x][y]=tmp
35

```

```

36     # right
37     if y<len(board[0])-1 and board[x][y+1]==word[0]:
38         tmp = board[x][y]
39         board[x][y] = '#'
40         if dfs(x,y+1,word[1:]):
41             return True
42         board[x][y] = tmp
43
44     return False
45
46 m, n = len(board), len(board[0])
47 for x in range(m):
48     for y in range(n):
49         if board[x][y]==word[0]:
50             if dfs(x,y,word[1:]):
51                 return True
52
53 return False

```

Listing 117: Problem79. Word Search

## 7 Dynamic Programming

### 7.0.1 leetcode 120. Triangle

Given a triangle, find the minimum path sum from top to bottom. Each step you may move to adjacent numbers on the row below. Note: Bonus point if you are able to do this using only  $O(n)$  extra space, where  $n$  is the total number of rows in the triangle.

解题思路：这道题可以用DFS和DP两种算法。由于决策树中存在重复分支，因此用DP算法更为高效，关键是存储空间的优化。DP算法有两种设计方法：自顶向下和自底向上。分析及演算参考下面的示例图。

- 第一种方法是为决策树中的每一个节点设立一个存储空间。那么，再次访问一个已经计算过的分支节点的时候，计算量仅为为 $O(1)$ 。这个方法的空间复杂度为 $O(n^2)$ ，实际使用的空间为 $O(n!)$ 。其转换方程 $dp[x][y] = :$  (1)  $\text{triangle}[m - 1]$ , if  $x = m - 1$ ; (2)  $\min(dp[x + 1][y], dp[x + 1][y + 1]) + \text{triangle}[x][y]$ , otherwise.
- 第二种方法是将存储空间减少到 $O(n)$ ：因为上层节点的数量小于其下一层结点的数量，因此可以按照最底层的节点数量开辟存储空间，重复的节点只算一次，即为最底层数组的大小。其转换方程 $dp[y] = :$  (1)  $\text{triangle}[m - 1]$ , if  $x = m - 1$ ; (2)  $\min(dp[y], dp[y + 1]) + \text{triangle}[x][y]$ , otherwise.
- 第三种方法也是DP算法，但是利用当前数组的空间存储中间计算的结果，整个计算过程结束后原数组会被破坏。如果不考虑保留原数组，那么这种方法不需要额外的存储空间。

```

1 class Solution(object):
2     def minimumTotal_dp1(self, triangle): # Space: O(n^2)
3         """
4             :type triangle: List[List[int]]
5             :rtype: int
6             """
7         m = len(triangle)
8

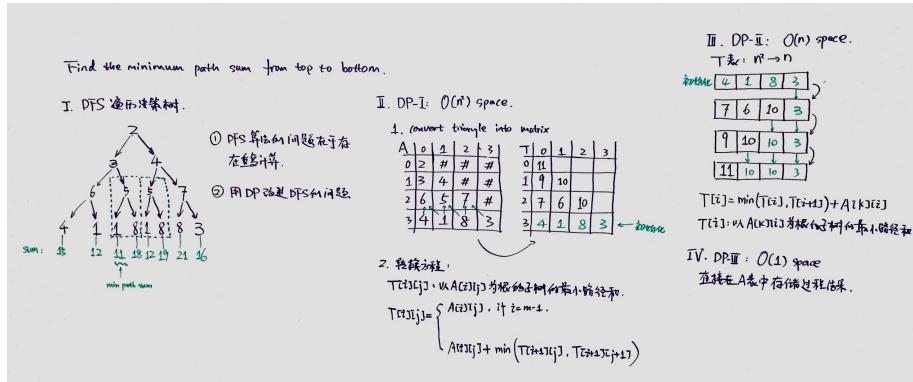
```

```

9     dp = [[0]*m for _ in xrange(m)]
10    dp[m-1] = triangle[m-1]
11    for x in xrange(m-2, -1, -1):
12        for y in xrange(x+1):
13            dp[x][y] = min(dp[x+1][y], dp[x+1][y+1])+triangle[x][y]
14    return dp[0][0]
15
16 def minimumTotal_dp2(self, triangle): # Space: O(n)
17     """
18     :type triangle: List[List[int]]
19     :rtype: int
20     """
21     m = len(triangle)
22     dp = triangle[m-1]
23     for x in xrange(m-2, -1, -1):
24         for y in xrange(x+1):
25             dp[y] = min(dp[y], dp[y+1]) + triangle[x][y]
26     return dp[0]
27
28 def minimumTotal_dp3(self, triangle): # Space: O(1) in-place
29     """
30     :type triangle: List[List[int]]
31     :rtype: int
32     """
33     m = len(triangle)
34     for x in xrange(m-2, -1, -1):
35         for y in xrange(x+1):
36             triangle[x][y] += min(triangle[x+1][y], triangle[x+1][y+1])
37     return triangle[0][0]

```

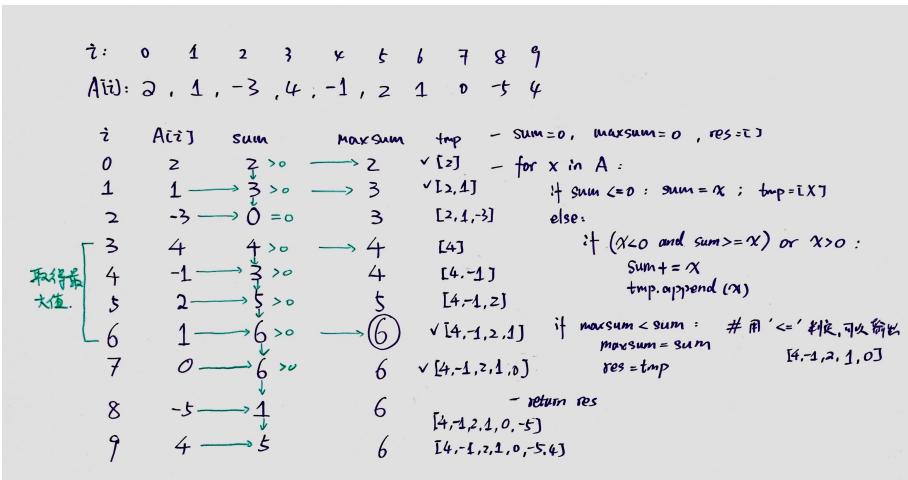
Listing 118: Problem120. Triangle



### 7.0.2 leetcode 53. Maximum Subarray

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.

解题思路：这道题要求子矩阵中的元素是连续的，即保持元素在数组中的相对位置，所以不能考虑排序。



```

1 class Solution(object):
2     def maxSubArray(self, nums):
3         """
4             :type nums: List[int]
5             :rtype: int
6         """
7
8         thisSum = 0
9         maxSum = -10000
10        for i in range(0, len(nums)):
11            if thisSum < 0: thisSum = 0
12            thisSum = thisSum + nums[i]
13            maxSum = max(thisSum, maxSum)
14
15        return maxSum

```

Listing 119: Problem53. Maximum Subarray

### 7.0.3 leetcode 131. Palindrome Partitioning

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  $[[ "aa", "b"], [ "a", "a", "b"]]$

解题思路：这道题用了DFS+DP的设计：DP表存储了任意长的子字符串是否为回文的信息，DFS为整体的决策框架。参见下面的示例图。

```

1 class Solution(object):
2     # dfs + dp
3     def partition(self, s):
4         """
5             :type s: str
6             :rtype: List[List[str]]
7         """
8
9         def dfs(s, offset, valuelist):
10            if len(s) == 0:
11                res.append(valuelist)
12                return

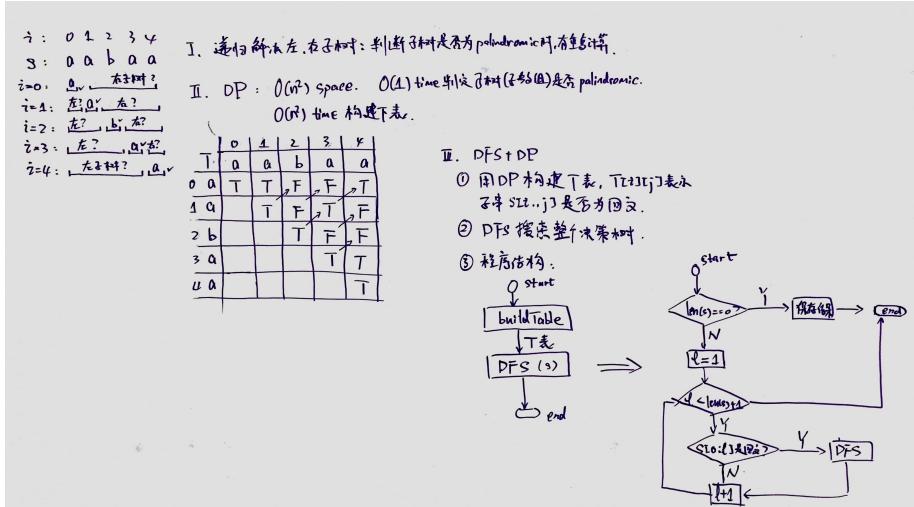
```

```

13     # try each possible sub-string
14     for l in range(1, len(s) + 1):
15         if table[0+ offset ][l-1+ offset ]:
16             dfs(s[l:], offset+1, valuelist + [s[:l]])
17
18     def buildTable(s):
19         n = len(s)
20         table = [[False] * n for _ in xrange(n)]
21         for length in xrange(1, n + 1):
22             for i in xrange(n - length + 1):
23                 j = (i + length - 1) % n
24                 if i == j:
25                     table[i][j] = True
26                 elif i == j - 1:
27                     table[i][j] = s[i] == s[j]
28                 else:
29                     table[i][j] = (s[i] == s[j]) and table[i +
30                         1][j - 1]
31
32     res = []
33     table = buildTable(s)
34     dfs(s, 0, [])
35     return res

```

Listing 120: Problem131. Palindrome Partitioning



#### 7.0.4 leetcode 132. Palindrome Partitioning II

Given a string  $s$ , partition  $s$  such that every substring of the partition is a palindrome. Return the minimum cuts needed for a palindrome partitioning of  $s$ . For example, given  $s = "aab"$ , Return 1 since the palindrome partitioning  $["aa", "b"]$  could be produced using 1 cut.

解题思路：由于不需要穷举出所有符合条件的回文分割，而是需要找到一个字符串  $s$  回文分割的最少分割次数，分割出来的字符串都是回文字符串。求次数的问题，不需要  $dfs$ ，会超时，求次数要考慮动态规划（dp）。对于程序的说

明:  $p[i][j]$  表示从字符  $i$  到  $j$  是否为一个回文字符串。 $dp[i]$  表示从第  $i$  个字符到最后一个字符, 最少的分割次数。

```

1  class Solution(object):
2      def minCut(self, s):
3          """
4              :type s: str
5              :rtype: int
6              """
7
8          n = len(s)
9          dp = [[-1 for _ in range(n+1)] for _ in range(n+1)]
10         p = [[False for _ in range(n+1)] for _ in range(n+1)]
11         p[0][0] = True
12
13         # l: the length of substring to check each time
14         for l in range(1, n+1):
15             for i in range(1, n-l+1):
16                 j = i+l-1
17                 if i == j: # when s[i:j+1] is a single character
18                     dp[i][j] = 0
19                     p[i][j] = True
20                 elif s[i]==s[j] and (i+1==j or p[i+1][j-1]==True):
21                     dp[i][j] = 0
22                     p[i][j] = True # s[i:j+1] is palindrome
23                 else:
24                     tmp = []
25                     for k in range(i, j+1):
26                         tmp.append(dp[i][k] + dp[k+1][j])
27                     dp[i][j] = min(tmp) + 1
28
29         return dp[0][n]
30
31     def minCut(self, s):
32         """
33             :type s: str
34             :rtype: int
35             """
36
37         # the min cuts of the string s[i:]
38         dp = [0 for i in range(len(s)+1)]
39         # p[i][j] is True, if s[i,j] is palindrome; otherwise,
40         False
41         p = [[False for i in range(len(s))] for j in range(len(s))]
42         # set dp[i] the upper bound
43         for i in range(len(s)+1):
44             dp[i] = len(s) - i
45         for i in range(len(s)-1, -1, -1):
46             for j in range(i, len(s)):
47                 # case1: i==j
48                 # case2: i==j+1, and s[i]==s[j]
49                 # case3: s[i]==s[j], and s[i+1...j-1] is palindrome
50                 if i==j or (s[i]==s[j] and i+1==j) or (s[i] == s[j] and p[i+1][j-1]):
51                     p[i][j] = True
52                     dp[i] = min(1+dp[j+1], dp[i])
53
54         return dp[0]-1

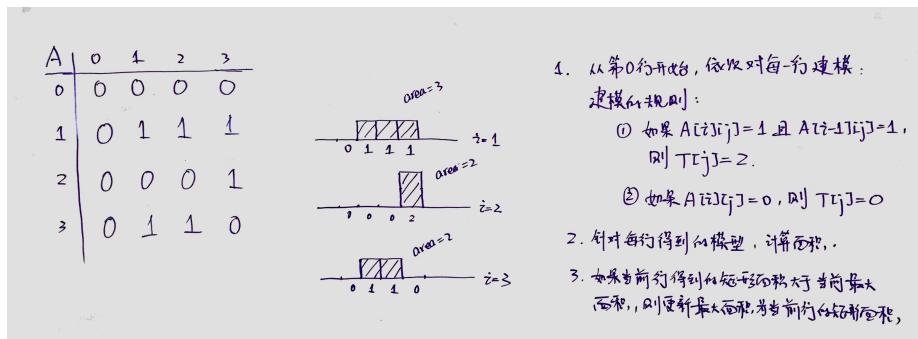
```

Listing 121: Problem132. Palindrome Partitioning II

### 7.0.5 leetcode 85. Maximal Rectangle

Given a 2D binary matrix filled with 0's and 1's, find the largest rectangle containing all ones and return its area.

解题思路：找出矩阵中最大的矩形，矩阵中只包含1。这道题需要Largest Rectangle in Histogram题目的结论。对于这个矩阵，对于每一行，我们按照Largest Rectangle in Histogram题目的算法求解一遍，最后得出的就是最大的矩阵。



```

1  class Solution(object):
2      def maximalRectangle(self, matrix):
3          """
4              :type matrix: List[List[str]]
5              :rtype: int
6          """
7
8          if len(matrix)==0: return 0
9          m, n = len(matrix), len(matrix[0])
10         dp = [0 for _ in xrange(n)]
11         maxarea = 0
12         for i in xrange(m):
13             for j in xrange(n):
14                 dp[j] = dp[j]+1 if matrix[i][j]=='1' else 0
15                 maxarea = max(maxarea, self.largestRectangleArea(dp))
16         return maxarea
17
18     # largest rectangle in histogram
19     def largestRectangleArea(self, height):
20         stack = []; area = 0
21         i = 0
22         while i<len(height):
23             if stack==[] or height[i]>height[stack[-1]]:
24                 stack.append(i)
25                 i += 1
26             else:
27                 top = stack.pop()
28                 width = i if stack==[] else i-stack[-1]-1
29                 area = max(area, width * height[top])
30         while stack!=[]:
31             top = stack.pop()
32             width = i if stack==[] else len(height)-stack[-1]-1
33             area = max(area, width * height[top])
34
35     return area

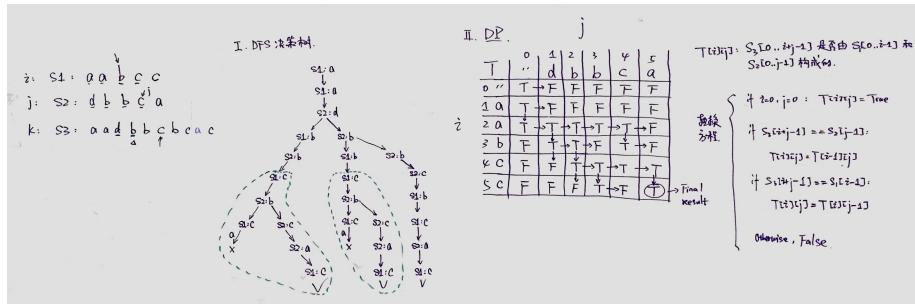
```

Listing 122: Problem85. Maximal Rectangle

### 7.0.6 leetcode 97. Interleaving String

Given s1, s2, s3, find whether s3 is formed by the interleaving of s1 and s2. For example, Given:s1 = "aabcc", s2 = "dbbca", When s3 = "aadbcbcac", return true. When s3 = "aadbbaaccc", return false.

解题思路：这道题可以用DFS和DP两种方法解决。DFS的问题就是重复计算，DP可以避免这个问题。具体的分析和示例见下面的示例图。



```

1  class Solution(object):
2      def isInterleave(self, s1, s2, s3):
3          """
4              :type s1: str
5              :type s2: str
6              :type s3: str
7              :rtype: bool
8          """
9
10         if len(s1)+len(s2)!=len(s3):
11             return False
12         dp = [[False for _ in range(len(s2)+1)] for _ in range(len(s1)+1)]
13         dp[0][0] = True
14
15         for i in range(1, len(s1)+1):
16             dp[i][0] = dp[i-1][0] and s3[i-1]==s1[i-1]
17         for j in range(1, len(s2)+1):
18             dp[0][j] = dp[0][j-1] and s3[j-1]==s2[j-1]
19
20         for i in range(1, len(s1)+1):
21             for j in range(1, len(s2)+1):
22                 dp[i][j] = (dp[i-1][j] and s1[i-1]==s3[i+j-1]) or (
23                     dp[i][j-1] and s2[j-1]==s3[i+j-1])
24
25         return dp[len(s1)][len(s2)]

```

Listing 123: Problem97. Interleaving String

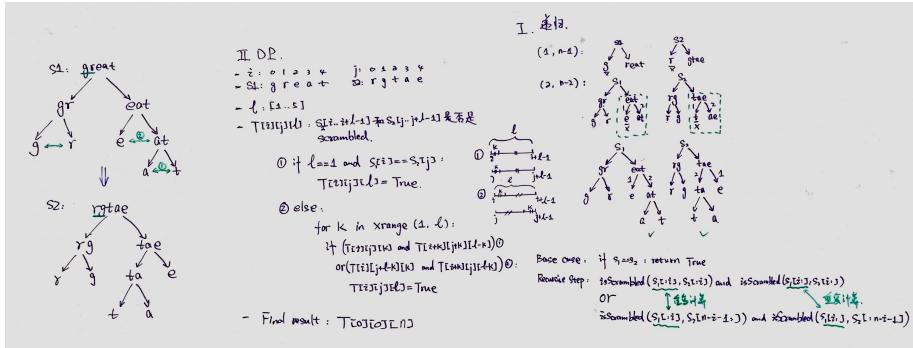
### 7.0.7 leetcode 87. Scramble String

Given a string s1, we may represent it as a binary tree by partitioning it to two non-empty substrings recursively. Given two strings s1 and s2 of the same length, determine if s2 is a scrambled string of s1.

解题思路：这道题有两种解法：

- 第一种解法是使用DFS遍历决策树。计算过程存在重复部分。
- 第二种解法DP，改进DFS算法，使用三维数组存储中间结果，改进计算时间。 $dp[i][j][l]$ 表示 $S_1[i..i+l-1]$ 和 $S_2[j..j+l-1]$ 是否是scrambled。

两种解法的分析和示例见下面的示例图。



```

1  class Solution(object):
2      def isScramble_dp(self, s1, s2):
3          """
4              :type s1: str
5              :type s2: str
6              :rtype: bool
7          """
8
9          if len(s1) != len(s2): return False
10         n = len(s1)
11         dp = [[[False for k in xrange(n+1)] for j in xrange(n)] for i in xrange(n)]
12         for k in xrange(1, n+1):
13             for i in xrange(n+1-k):
14                 for j in xrange(n+1-k):
15                     if k==1: dp[i][j][k] = s1[i]==s2[j]
16                     else:
17                         for l in xrange(1,k):
18                             if dp[i][j][k]: break
19                             else:
20                                 dp[i][j][k] = dp[i][j][1] and dp[i+1][j+1][k-1] or dp[i][j+k-1][1] and dp[i+1][j][k-1]
21         return dp[0][0][n]
22
23     def isScramble_recursive(self, s1, s2):
24         """
25             :type s1: str
26             :type s2: str
27             :rtype: bool
28         """
29         if len(s1) != len(s2): return False
30         if s1==s2: return True
31         l1 = list(s1)
32         l2 = list(s2)
33         l1.sort()
34         l2.sort()
35         if l1!=l2: return False
36         n = len(s1)
37         for i in range(1,n):

```

```

38     if self.isScramble(s1[:i], s2[:i]) and self.isScramble(
39         s1[i:], s2[i:]):
40             return True
41     if self.isScramble(s1[:i], s2[n-i:]) and self.
42         isScramble(s1[i:], s2[:n-i]):
43             return True
44     return False

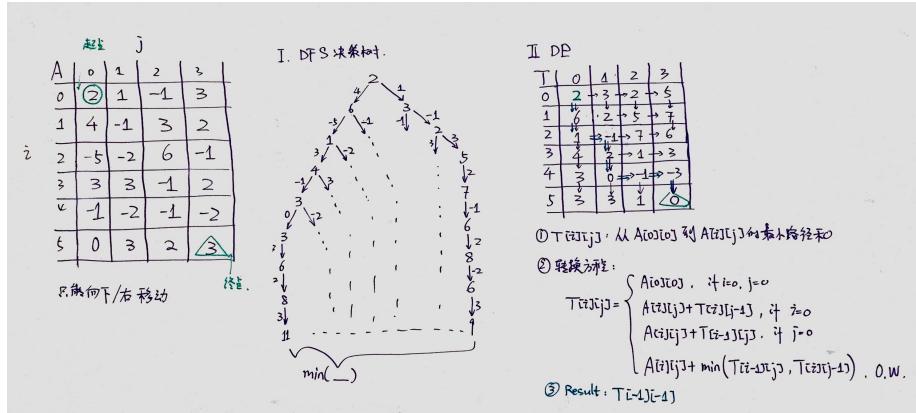
```

Listing 124: Problem87. Scramble String

### 7.0.8 leetcode 64. Minimum Path Sum

Given a  $m \times n$  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.

解题思路：这道题有两种解法，DFS和DP。两种解法的分析和示例见下面的示例图。



```

1
2 class Solution(object):
3     def minPathSum(self, grid):
4         """
5             :type grid: List[List[int]]
6             :rtype: int
7         """
8         m = len(grid); n = len(grid[0])
9         dp = [[0 for _ in range(n)] for _ in range(m)]
10        dp[0][0] = grid[0][0]
11        for i in range(1, n):
12            dp[0][i] = dp[0][i-1] + grid[0][i]
13        for i in range(1, m):
14            dp[i][0] = dp[i-1][0] + grid[i][0]
15        for i in range(1, m):
16            for j in range(1, n):
17                dp[i][j] = min(dp[i-1][j], dp[i][j-1]) + grid[i][j]
18        return dp[m-1][n-1]

```

Listing 125: Problem64. Minimum Path Sum

### 7.0.9 leetcode 72. Edit Distance

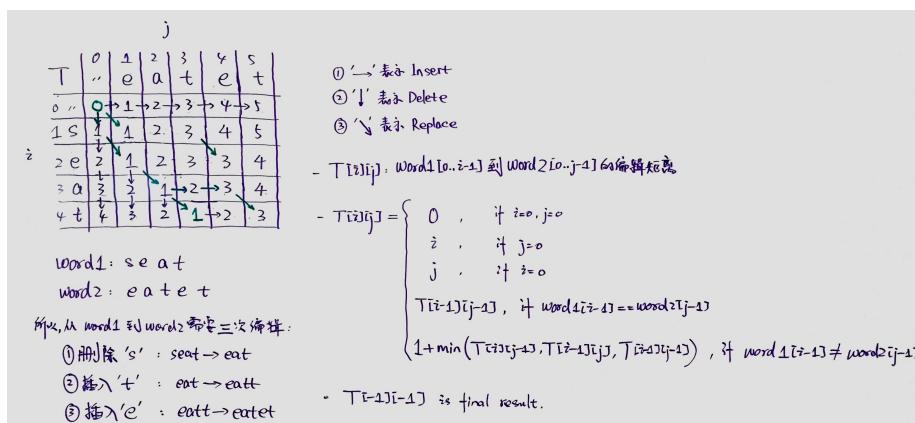
Given two words word1 and word2, find the minimum number of steps required to convert word1 to word2. (each operation is counted as 1 step.) You have the following 3 operations permitted on a word:

- Insert a character
- Delete a character
- Replace a character

解题思路：这道题是很有名的编辑距离问题。比较高效的解法就是用动态规划。状态转移方程是这样的： $dp[i][j]$ 表示word1[0...i-1]到word2[0...j-1]的编辑距离。状态转化方程是：

1. 第一种情况： $dp[i][0]$ 等于*i*, 因为只需要做*i*次删除操作就可以了。
2. 第二种情况： $dp[0][i]$ 等于*i*, 因为只需做*i*次插入操作就可以了。
3. 第三种情况： $dp[i][j] =$

- 如果 $word1[i-1]==word[j-1]$ ,  $dp[i][j]=dp[i-1][j-1]$ ;意思是，当前位置上的两个字符相同，无需插入、删除或者替代操作，所以到达当前位置时累计的操作数量等同于前一个位置累计的操作数量。
- 如果 $word1[i-1]!=word[j-1]$ ,  $dp[i][j]=\min(dp[i-1][j], dp[i][j-1], dp[i-1][j-1])+1$ ;意思是，如果当前位置处两个对应的字符不相同，那么累计的操作数取dp矩阵中当前位置的上面、左面和对角线左上三个邻接位置中累计操作数最小的一个再加1个操作。如果上面的最小，意味着增加一次删除(Delete)操作；如果左面的最小，意味着增加一次插入(Insert)操作；如果对角位置最小，意味着增加一次替代(Replace)操作。



```

1
2 class Solution(object):
3     def minDistance(self, word1, word2):
4         """
5             :type word1: str

```

```

6      :type word2: str
7      :rtype: int
8      """
9      m = len(word1)+1; n = len(word2)+1
10     dp = [[0 for _ in range(n)] for _ in range(m)]
11     # dp(i,j)→dp(i,j+1) means an insertion operation
12     for j in range(n): dp[0][j] = j
13     # dp(i,j)→dp(i+1,j) means an deletion operation
14     for i in range(m): dp[i][0] = i
15     # dp(i-1,j-1)→dp(i,j) means two things:
16     # first, NO operation if word1[i]==word[j];
17     # second, a replace operation if word[i]!=word[j].
18     for i in range(1, m):
19         for j in range(1, n):
20             if word1[i-1]==word2[j-1]:
21                 dp[i][j] = dp[i-1][j-1]
22             else:
23                 dp[i][j] = 1 + min(dp[i-1][j], dp[i][j-1], dp[i-1][j-1])
24     return dp[m-1][n-1]

```

Listing 126: Problem72. Edit Distance

#### 7.0.10 leetcode 91. Decode Ways

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.

解题思路：求次数，所以考虑用DP算法。dp[i]表示s[0...i-1]这*i*个数字的解码方法有多少种，初始条件dp[0]=1, dp[1]=1。根据题目中对于编码方式的描述，解码有下面四种情况：

- 如果s[i-2]和s[i-1]这两个字符构成10~26但不包括10和20之间的两位数时，比如21，那么可以有两种编码方式(BA, U)，所以dp[i]=dp[i-1]+dp[i-2]。
- 如果s[i-2]和s[i-1]构成的是10或者20，那么dp[i]=dp[i-2]。比如2102，'2'只有一种解码，'21'有两种解码，因为'0'没有解码方式，所以要想解码成功只能将'10'看成整体，那么'10'的解码方式的数量和'2'一样，只能有一种解码方式，即dp[3]=dp[1]=1，而dp[2]=2；因为s[1]和s[2]构成'10'，所以dp[4]=dp[2]=2。
- 如果s[i-2]和s[i-1]构成的数字不是前面两种情况，并且s[i-1]!='0'，那么dp[i]=dp[i-1]。
- 如果以上三种情况都不满足，则不符合解码要求，解码方式为0。比如，'09'没有解码方式

```

1 class Solution(object):
2     def numDecodings(self, s):
3         """
4             :type s: str
5             :rtype: int
6             """
7

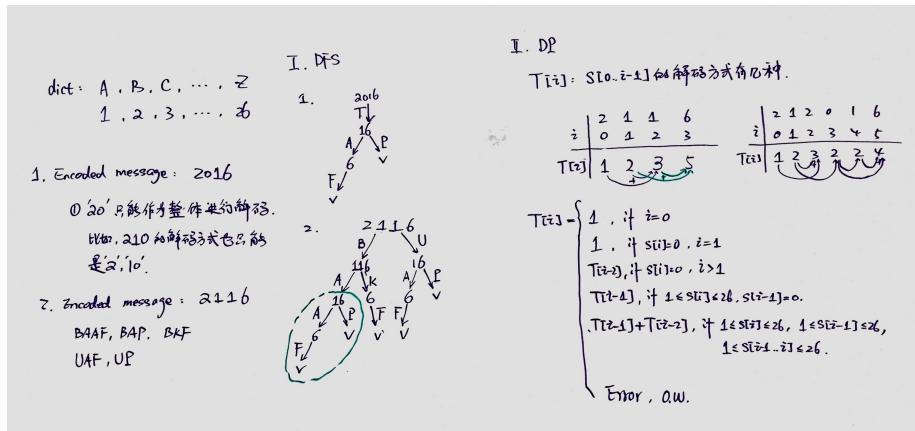
```

```

8         if s=="" or s[0]=='0': return 0
9         dp = [1,1]
10        for i in range(2, len(s)+1):
11            if 10 <= int(s[i-2:i]) <=26 and s[i-1]!='0':
12                dp.append(dp[i-2]+dp[i-1])
13            elif int(s[i-2:i]) ==10 or int(s[i-2:i]) ==20:
14                dp.append(dp[i-2])
15            elif s[i-1]!='0':
16                dp.append(dp[i-1])
17            else:
18                return 0
19
20        return dp[len(s)]

```

Listing 127: Problem91. Decode Ways



### 7.0.11 leetcode 115. Distinct Subsequences

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. Here is an example: S = "rabb-bit", T = "rabbit", Return 3.

解题分析：这道题使用动态规划来解决。题的意思是：S的所有子串中，有多少子串是T。下面来看看状态转移方程。分析和示例可以参考下面示例图。

```

1 class Solution(object):
2     def numDistinct(self, s, t):
3         """
4             :type s: str
5             :type t: str
6             :rtype: int
7             """
8         m, n = len(s)+1, len(t)+1
9         dp = [[0 for j in xrange(n)] for i in xrange(m)]
10        for i in xrange(m):
11            for j in xrange(n):
12

```

	0	1	2	3	4	5	6
0	1	0	0	0	0	0	0
1	1	1	0	0	0	0	0
2	0	1	1	0	0	0	0
3	b	1	1	1	0	0	0
4	b	1	1	1	1	0	0
5	b	1	1	1	1	1	0
6	b	1	1	1	1	1	1
7	t	1	1	1	1	1	1

$s = rabbbit$   
 $t = rabbbit$

$T[i][j] = S[0..i-1]$  包含  $t[0..j-1]$  的子集。

$T[i][j] = \begin{cases} T[i-1][j-1] + T[i-1][j], & \text{if } S[i] == t[j] \\ T[i-1][j], & \text{if } S[i] \neq t[j] \\ 1, & \text{if } j=0 \\ 0, & \text{if } i=0, j>0 \end{cases}$

```

13     dp[i][0] = 1
14
15     for i in range(1, m):
16         for j in range(1, min(i+1, n)):
17             if s[i-1] == t[j-1]:
18                 dp[i][j] = dp[i-1][j] + dp[i-1][j-1]
19             else:
20                 dp[i][j] = dp[i-1][j]
21
22     return dp[len(s)][len(t)]

```

Listing 128: Problem115. Distinct Subsequences

### 7.0.12 leetcode 139. Word Break

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".

解题思路：因为这道题不需要给出如何分割的答案，只需要判断“能不能”分割为字典中的单词，所以考虑使用DP，而不是DFS。这里给出两种DP算法设计，区别在于两种算法的空间复杂度。

- 第一种算法 $wordBreak\_dp1$ 中， $dp[i][j]$ 表示 $s[i-1..j-1]$ 是否可以分割开。设计的思想是分别验证字符串 $s$ 的 $l$ 长子串是否在字典中， $1 \leq l \leq n$ 。
- 第二种算法 $wordBreak\_dp2$ 改进了方法一的空间复杂度，使用一维数组， $dp[i]$ 表示 $s[0..i-1]$ 是可以分割的。示例及分析可以参考下面示例图。

```

1 class Solution(object):
2     def wordBreak_dp2(self, s, wordDict):
3         """
4             :type s: str
5             :type wordDict: Set[str]
6             :rtype: bool
7             """
8             # dp[i] indicates s[0..i-1] can be segmented
9

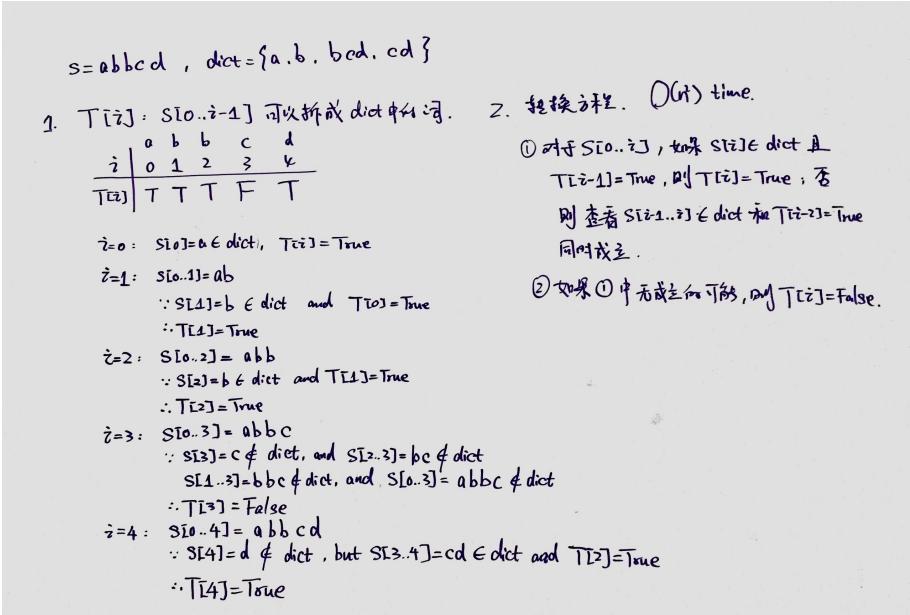
```

```

10     n = len(s)
11     dp = [False for _ in xrange(n+1)]
12     dp[0] = True
13     for l in xrange(1, n+1):
14         for i in range(l):
15             if dp[i] and s[i:l] in wordDict:
16                 dp[l] = True
17                 break
18     return dp[-1]
19
20 def wordBreak_dp1(self, s, wordDict):
21     n = len(s)
22     dp = [[False for j in xrange(n)] for i in xrange(n)]
23     for l in xrange(1, n+1):
24         for i in xrange(n-l+1):
25             j = i+l-1
26             if s[i:j+1] in wordDict:
27                 dp[i][j] = True
28             else:
29                 for k in xrange(i+1, j+1):
30                     if dp[i][k-1] and dp[k][j]:
31                         dp[i][j] = True
32                         break
33     return dp[0][n-1]

```

Listing 129: Problem139. Word Break

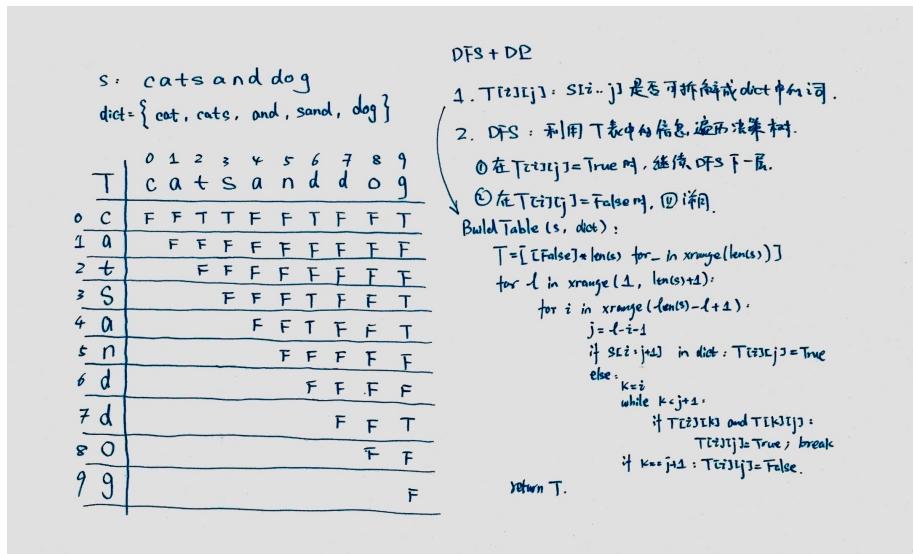


### 7.0.13 leetcode 140. Word Break II

Given a string s and a dictionary of words dict, add spaces in s to construct a sentence where each word is a valid dictionary word. Return all such possible sentences.

For example, given  $s = \text{"catsanddog"}$ ,  $\text{dict} = [\text{"cat"}, \text{"cats"}, \text{"and"}, \text{"sand"}, \text{"dog"}]$ . A solution is  $[\text{"cats and dog"}, \text{"cat sand dog"}]$ .

解题思路：与leetcode139 word break不同，这道题要求输出所有的分割方式，那么就需要考虑使用DFS。不过，因为决策树太大，会计算超时，所以不能直接使用DFS。因此，需要对决策树进行剪枝：利用leetcode139 word break算法生成的矩阵，在DFS之前，先判定字符串是否可以被分割，如果不能被分割，直接跳过这一枝。这道题考核的是DP+DFS。参考下面的示例图。



```

1 class Solution(object):
2     def wordBreak(self, s, wordDict):
3         """
4             :type s: str
5             :type wordDict: Set[str]
6             :rtype: List[str]
7         """
8
9         # to find partition ways, need to use dfs based
10        # on dp result
11        Solution.result = []
12        self.dfs(s, wordDict, '')
13        return Solution.result
14
15    def dfs(self, s, dict, stringlist):
16        if self.check(s, dict):
17            if len(s)==0: Solution.result.append(stringlist[1:])
18            for i in range(1, len(s)+1):
19                if s[:i] in dict:
20                    self.dfs(s[i:], dict, stringlist + ' ' + s[:i])
21
22        # check if s can be partitioned based on dict
23    def check(self, s, dict):
24        dp = [False for _ in range(len(s)+1)]
25        dp[0] = True # empty string is substring of any string
26        for i in range(1, len(s)+1):
27            for k in range(i):
28                if dp[k] and s[k:i] in dict:

```

```

29         dp[i] = True
30         break
31     return dp[len(s)]

```

Listing 130: Problem140. Word Break II

## 7.1 leetcode 44. Wildcard Matching

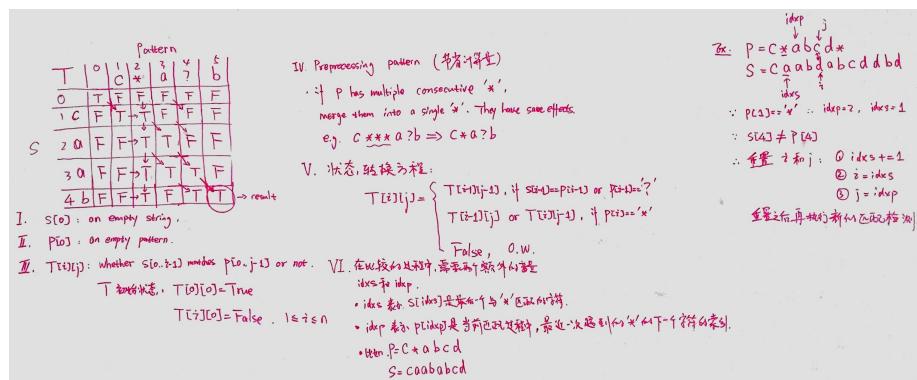
Implement wildcard pattern matching with support for '?' and '\*'. '\*' Matches any single character. '\*' Matches any sequence of characters (including the empty sequence). The matching should cover the entire input string (not partial).

The function prototype should be: bool isMatch(const char \*s, const char \*p)

Some examples:

1. isMatch("aa", "a") → false
2. isMatch("aa", "aa") → true
3. isMatch("aaa", "aa") → false
4. isMatch("aa", "\*") → true
5. isMatch("aa", "a\*") → true
6. isMatch("ab", "?\*") → true
7. isMatch("aab", "c\*a\*b") → false

解题思路：这道题可以用递归解决（代码实现line 3-26），但是因为重复计算量太大，在字符串s和pattern比较长的情况下，就会造成计算超时的问题。因此，设计DP算法是考核的重点。最直接的DP算法是使用一个二维矩阵dp[i][j]存储计算过程的中间结果：矩阵的行表示字符串s，矩阵的列表示pattern，dp[i][j]表示s[0..i-1]与p[0..j-1]是否匹配。转换方程定义及示例参见下图（图中的VI部分是改进DP算法的一些说明）。代码实现line 28-64。这个算法的时间复杂度是 $O(m \times n)$ ，用于填充数组；相应的存储空间就是矩阵的大小 $O(m \times n)$ 。此外，我们还可以从减少空间复杂度的角度进一步改进上面的算法。改进后的算法实现可以参见代码line 66-103。



```

1
2     class Solution(object):
3         def isMatch_recursive(self, s, p): # TLE error
4             """
5                 :type s: str
6                 :type p: str
7                 :rtype: bool
8             """
9             if len(p)>len(s):
10                 return False
11             elif len(s)==0 or len(p)==0:
12                 return len(s) == len(p)
13             elif len(s)==1 and len(p)==1 and s==p:
14                 return True
15             elif s[0]==p[0] or p[0]=='?':
16                 return self.isMatch(s[1:],p[1:])
17             elif p[0]=='*':
18                 i = 0
19                 while i<len(p) and p[i]=='*':
20                     i += 1
21                     if i==len(p): return True
22                     j = 0
23                     while j<len(p) and not self.isMatch(s[j:], p[i:]):
24                         j += 1
25                     return j!=len(p)
26             return False
27
28     def isMatch_dp1(self, s, p): # RT: O(n^2), Space: O(n^2)
29         """
30             :type s: str
31             :type p: str
32             :rtype: bool
33         """
34         if len(p) - p.count('*') > len(s): return False
35
36         # replace multiple * with one *
37         # e.g., a***b**c => a*b*c
38         pattern = []
39         isFirst = True
40         for i in range(len(p)):
41             if p[i]=='*':
42                 if isFirst:
43                     pattern.append(p[i])
44                     isFirst = False
45                 else:
46                     pattern.append(p[i])
47                     isFirst = True
48         m, n = len(s), len(pattern)
49
50         # initialize dp matrix
51         dp=[[False for _ in range(n+1)] for _ in range(m+1)]
52         dp[0][0]=True
53         if n>0 and pattern[0]=='*': dp[0][1] = True
54
55         # populate the dp matrix
56         for i in range(1, m+1): # for string
57             for j in range(1, n+1): # for pattern
58                 if pattern[j-1]=='*':
59                     dp[i][j] = dp[i-1][j] or dp[i][j-1]
60                 elif s[i-1]==pattern[j-1] or pattern[j-1]=='?':
61                     dp[i][j] = dp[i-1][j-1]
62                 else:

```

```

63             dp[i][j] = False
64     return dp[m][n]
65
66     def isMatch_dp2(self, s, p): # RT: O(n), Space: O(1)
67         """
68             :type s: str
69             :type p: str
70             :rtype: bool
71         """
72         # the position of the last occurrence of * in p
73         lastStartPos = -1
74         # the position of the last character in s which matches
75         with *
76             lastCharMatchStar = 0
77             idxs = 0
78             while idxs < len(s):
79                 # when there is a match, continue to compare the
80                 # characters in s and p
81                 if idxp < len(p) and (s[idxs] == p[idxp] or p[idxp] == '?'):
82                     idxs += 1
83                     idxp += 1
84                     continue
85                 # when * comes, save the positions of idxp and idxs
86                 elif idxp < len(p) and p[idxp] == '*':
87                     lastStartPos = idxp
88                     lastCharMatchStar = idxs
89                     idxp += 1
90                     continue
91                 # when there is a conflict, and we have a star, reset
92                 # idxp
93                 # to point to the next position of previous, and also
94                 # move ss
95                 # to its next position, and set idxs to point to ss's
96                 # position;
97                 # after these settings, continue to check
98                 elif lastStartPos != -1:
99                     idxp = lastStartPos + 1
100                    lastCharMatchStar += 1
101                    idxs = lastCharMatchStar
102                    continue
103                # if all the conditions are not satisfied, s does not
104                # match p.
105                else: return False
106            while idxp < len(p) and p[idxp] == '*':
107                idxp += 1
108            if idxp == len(p): return True
109            return False

```

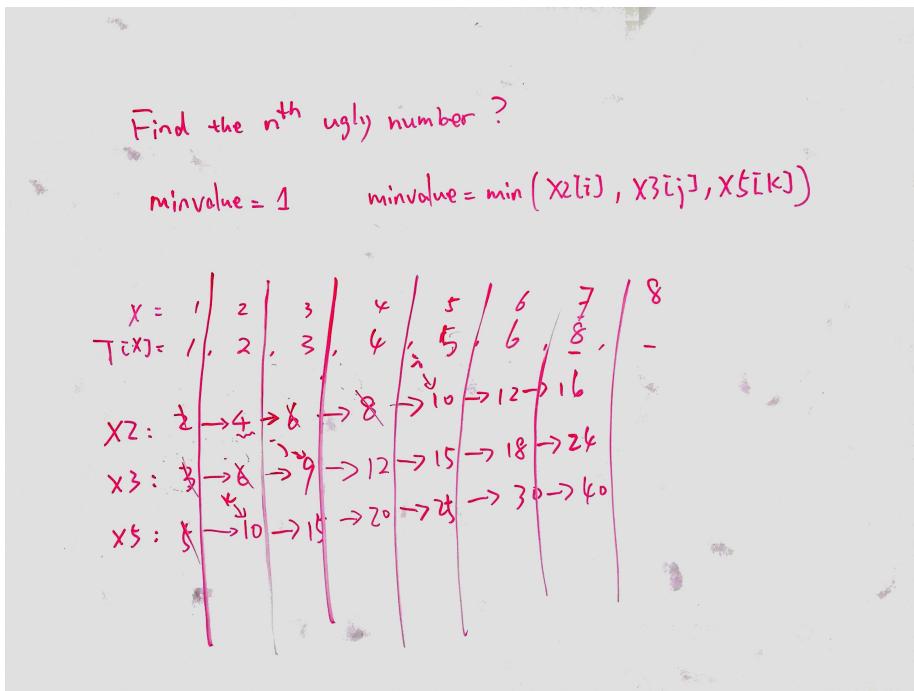
Listing 131: Problem44. Wildcard Matching

## 7.2 leetcode 264. Ugly Number II

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.

解题思路：这道题使用DP算法，因为结果是一个值，并且是线性决定的，所以使用一个变量记录当前结果，使用三个一维数组分别记录当前结果与

三个不同因数的乘积。参考下图中的示例，我们用minvalue来存储每一轮计算的结果，minvalue的初始值是ugly number的第一个值1。在每一轮循环中，minvalue分别乘以因数2、3、5，并将结果分别存放到x2、x3、x5这三个列表中，并取这三个列表各自的第一个元素中最小的一个更新minvalue，同时移除三个首元素中与其值相同的元素，以保持三个列表的首元素始终是与minvalue不同值。



```

1  class Solution(object):
2      def nthUglyNumber(self, n):
3          """
4              :type n: int
5              :rtype: int
6              """
7
8          if n==0: return 0
9          if n==1: return 1
10         target = 1
11         x2, x3, x5 = [], [], []
12         for i in range(n-1):
13             x2.append(target*2)
14             x3.append(target*3)
15             x5.append(target*5)
16             target = min(x2[0], x3[0], x5[0])
17
18             # remove the duplicates
19             if target==x2[0]: x2.pop(0)
20             if target==x3[0]: x3.pop(0)
21             if target==x5[0]: x5.pop(0)
22         return target
23
24     def nthUglyNumber_dp(self, n): # TLE error
25         """

```

```

26     :type n: int
27     :rtype: int
28     """
29     dp = [False]
30     count = 0; i = 0
31     while count < n:
32         i += 1
33         if i in [1, 2, 3, 5]:
34             dp.append(True)
35         elif i%2 == 0:
36             dp.append(dp[i/2])
37         elif i%3 == 0: dp.append(dp[i/3])
38         elif i%5 == 0: dp.append(dp[i/5])
39         else: dp.append(False)
40
41         if dp[i]==True:
42             count += 1
43
44     return i

```

Listing 132: Problem264. Ugly Number II

### 7.3 leetcode 303. Range Sum Query - Immutable

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

Example: Given nums = [-2, 0, 3, -5, 2, -1]

1. sumRange(0, 2) -> 1
2. sumRange(2, 5) -> -1
3. sumRange(0, 5) -> -3

Note: You may assume that the array does not change. There are many calls to sumRange function.

解题思路：这道题是比较明显的DP算法求解题目，只需要一个一维数组储存nums[0..i]的和dp[i]，那么sumRange(i,j)的值就是dp[j]-dp[i-1]了。参见下面示例图中的示例和定义。

```

1  class NumArray(object):
2      def __init__(self, nums):
3          """
4              initialize your data structure here.
5              :type nums: List[int]
6              """
7
8          self.accumulate = [0]
9          for i in nums:
10              self.accumulate.append(self.accumulate[-1] + i)
11
12      def sumRange(self, i, j):
13          """
14              sum of elements nums[i .. j], inclusive.
15              :type i: int

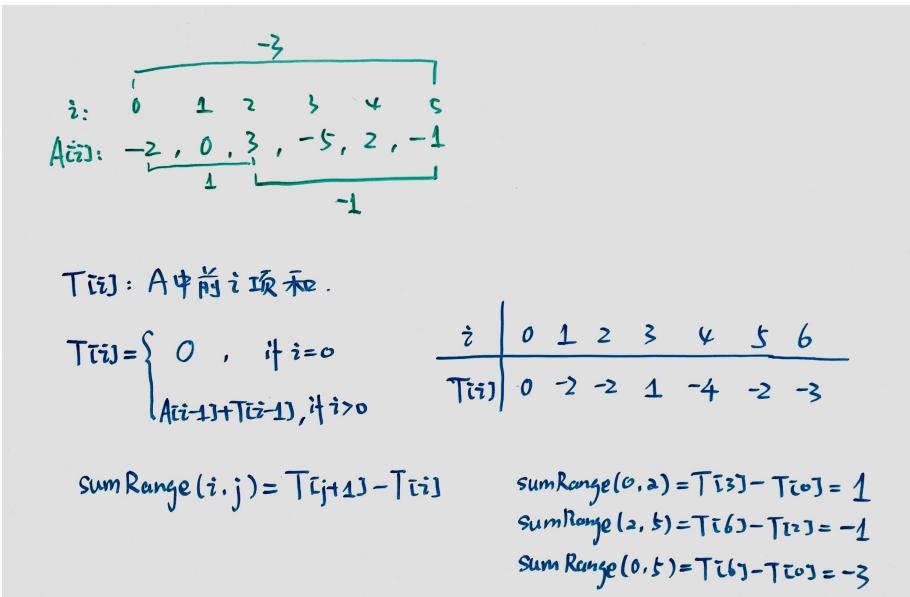
```

```

16     :type j: int
17     :rtype: int
18     """
19     if i<0 or j>=len(self.accumulate) or i>j: return 0
20     return self.accumulate[j+1] - self.accumulate[i]

```

Listing 133: Problem303. Range Sum Query - Immutable



## 7.4 leetcode 304. Range Sum Query 2D - Immutable

Given a 2D matrix matrix, find the sum of the elements inside the rectangle defined by its upper left corner (row1, col1) and lower right corner (row2, col2). Note:

1. You may assume that the matrix does not change.
2. There are many calls to sumRegion function.
3. You may assume that ( $\text{row1} \leq \text{row2}$ ) and ( $\text{col1} \leq \text{col2}$ ).

解题思路：这道题在leetcode303 Range Sum Query的基础上通过增加数组维度来增加难度。但是解题方法的核心思想一致，用相应的二维数组来存储计算过程中间的临时结果。具体的示例演算和分析参考下面的示例图。

```

1
2 class NumMatrix(object):
3     def __init__(self, matrix):
4         """
5             initialize your data structure here.
6             :type matrix: List[List[int]]
7             """

```

```

8     if matrix==[]:
9         return
10    self.m = len(matrix); self.n = len(matrix[0])
11    self.accumulate = [[0 for _ in range(self.n)] for _ in
12    range(self.m)]
13    for i in range(self.m):
14        for j in range(self.n):
15            self.accumulate[i][j] = self.accumulate[i][j-1] +
matrix[i][j]
16
17    def sumRegion(self, row1, col1, row2, col2):
18        """
19            sum of elements matrix[(row1,col1)..(row2,col2)], inclusive
20
21            :type row1: int
22            :type col1: int
23            :type row2: int
24            :type col2: int
25            :rtype: int
26        """
27
28        if row1>row2 or col1>col2 or row1<0 or row2>=self.m or col1
29        <0 or col2>=self.n:
30            return 0
31        result = 0
32        for i in range(row1, row2+1):
33            if col1 == 0:
34                result += self.accumulate[i][col2]
35            else:
36                result += self.accumulate[i][col2] - self.
37                accumulate[i][col1-1]
38        return result

```

Listing 134: Problem304. Range Sum Query 2D - Immutable

	j				
i	0	1	2	3	4
A	0	3	0	1	4
	1	5	6	3	2
	2	1	2	0	1
	3	4	1	0	1
	4	1	0	3	0
	5				5

	j				
i	0	1	2	3	4
T	0	3	3	4	8
	1	8	14	18	24
	2	9	17	21	28
	3	13	22	26	34
	4	14	23	30	38
	5				56

④ if $i_0=0$ and $j_0=0$ :	$T[i_0][j_0] = A[i_0][j_0]$
$T[i_0][j_0] = T[i_0-1][j_0]$	
⑤ if $i_0=0$ and $j_0 \neq 0$ :	$T[i_0][j_0] = T[i_0][j_0-1]$
$T[i_0][j_0] = T[i_0][j_0-1]$	
⑥ if $i_0 \neq 0$ and $j_0=0$ :	$T[i_0][j_0] = T[i_0-1][j_0]$
$T[i_0][j_0] = T[i_0-1][j_0]$	
⑦ if $i_0 \neq 0$ and $j_0 \neq 0$ :	$T[i_0][j_0] = T[i_0-1][j_0-1]$
$T[i_0][j_0] = T[i_0-1][j_0-1]$	

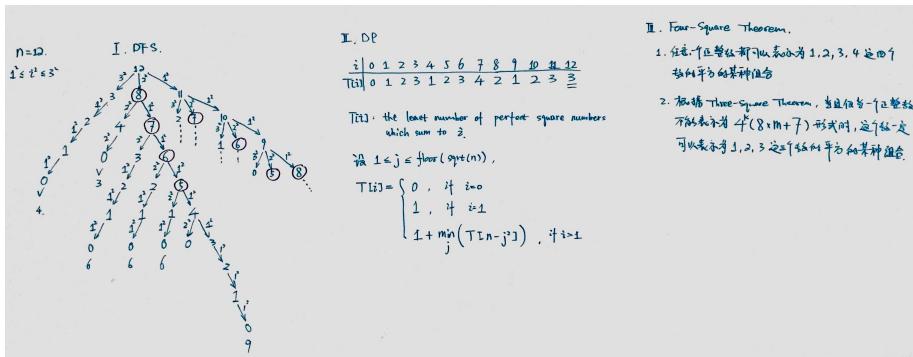
  

⑧ if $i_0=0$ and $j_0 \neq 0$ :	$T[i_0][j_0] = A[i_0][j_0] + T[i_0][j_0-1]$
$T[i_0][j_0] = A[i_0][j_0] + T[i_0-1][j_0]$	
⑨ if $i_0 \neq 0$ and $j_0=0$ :	$T[i_0][j_0] = A[i_0][j_0] + T[i_0][j_0-1]$
$T[i_0][j_0] = A[i_0][j_0] + T[i_0-1][j_0]$	
⑩ if $i_0 \neq 0$ and $j_0 \neq 0$ :	$T[i_0][j_0] = A[i_0][j_0] + T[i_0-1][j_0-1]$
$T[i_0][j_0] = A[i_0][j_0] + T[i_0-1][j_0-1]$	

## 7.5 leetcode 279. Perfect Squares

Given a positive integer n, find the least number of perfect square numbers (for example, 1, 4, 9, 16, ...) which sum to n. For example, given n = 12, return 3 because  $12 = 4 + 4 + 4$ ; given n = 13, return 2 because  $13 = 4 + 9$ .

解题思路：这道题有三种解法：DFS, DP和数论中的四平方和定理。这里特别说明一下四平方和定理(Lagrange's four-square theorem)的解法，DFS和DP的解法可以参考示例图。根据四平方和定理，任意一个自然数均可表示为4个整数的平方和，其实是可以表示为4个以内的平方数之和，那么就是说返回结果只有1,2,3或4其中的一个。算法首先要化简 $n$ 。根据Adrien-Marie Legendre's three-square theorem，当且仅当一个正整数不能表示为 $4^k(8m+7)$ 这种形式时，这个正整数可以表示为三个整数的平方和。因此，算法首先除去正整数 $n$ 中包含的因子数4 (代码行line 4)；然后，再判断处理后的正整数 $n$ 除以8的余数是否为7，如果成立，那么根据上面的三平方定理，正整数 $n$ 只能表示为四个整数的平方和(line 5)；若果不成立， $n$ 只能表示为一个、两个、或者三个整数平方和的形式。算法需要进一步判断 $n$ 是否可以表示为一个或者两个整数平方和的形式(line 7-12)；如果也不行，那么根据四平方和定理， $n$ 一定可以表示为三个整数平方和的形式。



```

1  class Solution(object):
2      def numSquares(self, n):
3          while n%4==0: n/=4
4          if n%8==7: return 4
5          m = int(n**0.5)+1
6          for x in xrange(m):
7              y = int((n-x*x)**0.5)
8              if x*x + y*y == n:
9                  if x>0 and y>0:
10                     return 2
11                 else: return 1
12
13     return 3

```

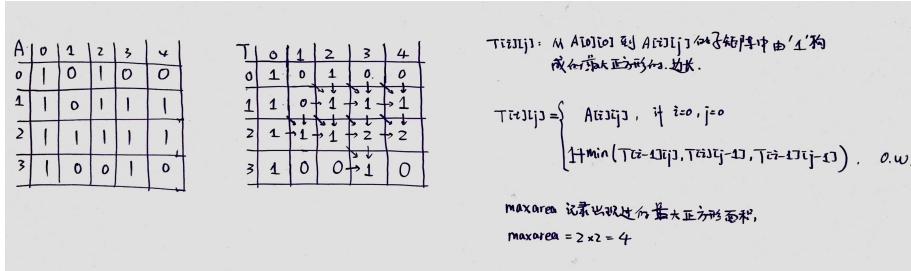
Listing 135: Problem279. Perfect Squares

## 7.6 leetcode 221. Maximal Square

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, and return 4.

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

解题思路：这道题和leetcode85 Maximal Rectangle是相似的题目，但是解题思路完全不同。 $dp[i][j]$ 表示以(i,j)为右下角的正方形的最大边长。转换方程及示例见下图。



```

1  class Solution(object):
2      def maximalSquare(self, matrix):
3          """
4              :type matrix: List[List[str]]
5              :rtype: int
6              """
7
8          if matrix==[]: return 0
9          m, n = len(matrix), len(matrix[0])
10         dp = [[0]*n for _ in xrange(m)]
11         dp[0][0] = matrix[0][0]
12         width = 0
13         for i in xrange(0, m):
14             for j in xrange(0, n):
15                 if matrix[i][j]=='0':
16                     dp[i][j] = 0
17                 elif i==0 or j==0:
18                     dp[i][j] = int(matrix[i][j])
19                 else:
20                     dp[i][j] = 1 + min(dp[i][j-1], dp[i-1][j], dp[i-1][j-1])
21             width = max(width, dp[i][j])
22
23         return width*width

```

Listing 136: Problem221. Maximal Square

## 7.7 leetcode 152. Maximum Product Subarray

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.

解题思路：这道题可以用两种不同的DP算法解决。参考下面示例图中的示例和说明。

```

1  class Solution(object):
2      def maxProduct(self, nums): # RT: O(n)
3          """
4              :type nums: List[int]
5              :rtype: int
6              """
7

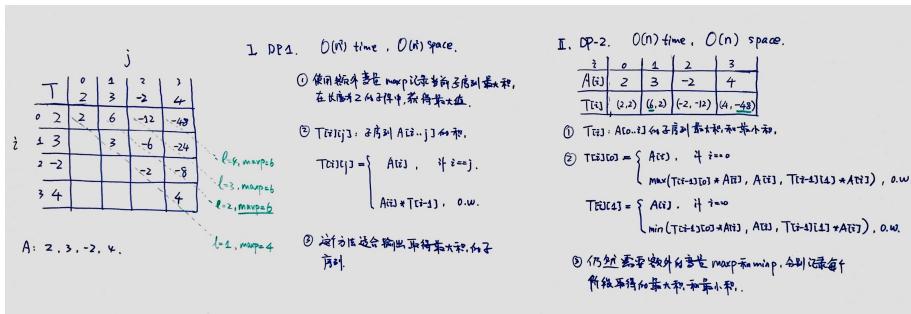
```

```

8     n = len(nums)
9     if n==0: return n
10    # current max product
11    maxp = nums[0]
12    # current min product
13    minp = nums[0]
14    # max product
15    maxproduct = nums[0]
16    for i in range(1, n):
17        tmp = maxp
18        maxp = max(maxp*nums[i], nums[i], minp*nums[i])
19        minp = min(tmp*nums[i], nums[i], minp*nums[i])
20        maxproduct = max(maxval, maxp)
21    return maxproduct

```

Listing 137: Problem152. Maximum Product Subarray



## 7.8 leetcode 300. Longest Increasing Subsequence

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  $O(n^2)$  complexity.

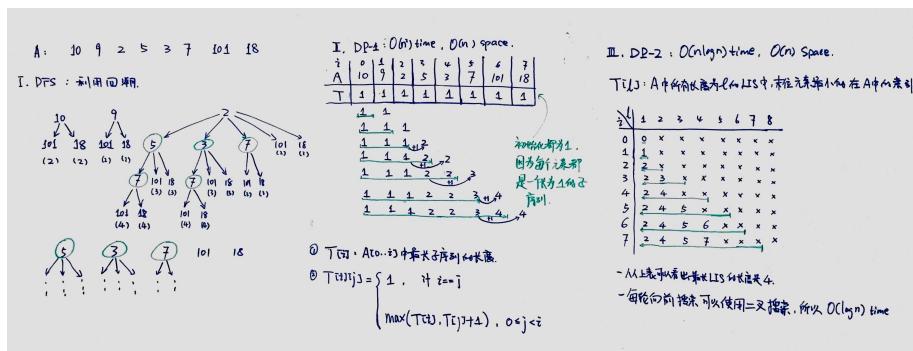
Follow up: Could you improve it to  $O(n \log n)$  time complexity?

解题思路：这道题可以使用DFS和DP两种算法设计思路。但是题目要求用 $O(n^2)$ 和 $O(n \log n)$ 两种时间复杂度算法实现，显然不能用DFS。

- $O(n^2)$  动态规划算法：开辟一个与给定数组 $nums$ 等长的数组 $T$ 用于存储中间过程结果。 $T[i]$ 表示 $nums[0..i]$ 的最大递增子序列(LIS)的长度。对于 $nums$ 中的每一个位置 $i$ 来说，对应的LIS的长度至少是1，也就是数组 $nums$ 是一个严格递减的序列，所以 $T[i]$ 的初始值可以设置为1。在计算每一个 $T[i]$ 时，需要比较 $nums[i]$ 和 $nums[0..i-1]$ 中的每一个元素，如果 $nums[i] > nums[j]$ ,  $0 \leq j \leq i - 1$ , 说明 $nums[j]$ 和 $nums[i]$ 构成递增子序列，因此 $nums$ 的前 $i$ 个元素构成的最大子序列的长度就等于前 $j$ 个元素构成的最大子序列的长度加1，即 $T[j]+1$ ；此外，考虑到在当前位置 $j$ 之前， $T[i]$ 可能取得了更大的值，因此应该取 $T[i]$ 当前值和 $T[j]+1$ 之间较大的一个值更新当前 $T[i]$ 。参见下面示例图中的示例。

- $O(n \log n)$  动态规划算法：算法的设计需要长度为  $n$  的数组  $T$ ，其中  $T[i]$  存储的是  $A$  中所有长度为  $i$  的 LIS 中，最小末尾元素在数组  $A$  中的索引。算法由内外两层循环构成，外层循环用于遍历数组  $A$ ，时间复杂度为  $O(n)$ ；内层循环用于在  $[1..i-1]$  范围内搜索当前元素  $A[i]$  可以增加哪个 LIS 的长度，并更新  $T$  表中相应长度的 LIS 的对应的值。内层循环用于搜索，所以可以使用 binary search 算法，时间复杂度为  $O(\log n)$ 。所以，整个算法的时间复杂度为  $O(n \log n)$ 。参见下面示例图中的示例。

强烈推荐 youtube 上 Tushar Roy 录制的关于 Longest Increasing Subsequence 问题的视频讲解。



```

1  class Solution(object):
2      def lengthOfLIS_dp1(self, nums): # RT: O(n^2)
3          """
4              :type nums: List[int]
5              :rtype: int
6              """
7
8          n = len(nums)
9          if n==0: return n
10         dp = [1 for _ in range(n)]
11         longest = 1
12         for i in range(n):
13             for j in range(i):
14                 if nums[j]<nums[i]:
15                     dp[i] = max(dp[i], dp[j]+1)
16         return max(dp)
17
18     def lengthOfLIS_dp2(self, nums): # RT: O(n log n)
19         """
20             :type nums: List[int]
21             :rtype: int
22             """
23
24         n = len(nums)
25         if n==0: return n
26         dp = [0 for _ in range(n)]
27         length = 0
28         for i in range(1, n):
29             if nums[dp[0]] > nums[i]:
30                 dp[0] = i
31             elif nums[dp[length]] < nums[i]:
32                 length += 1
33                 dp[length] = i
34             else:
35                 # do a binary search to find the ceiling

```

```

35             # of nums[i] and put it there
36             index = self.getCeilIndex(nums, dp, length, nums[i
37             ])
38             dp[index] = i
39         return length+1
40
41     def getCeilIndex(self, nums, dp, length, val): # binary search:
42         O(logn)
43         start = 0; end = length
44         while start <= end:
45             mid = (start+end)/2
46             if mid<length and nums[dp[mid]]<val<=nums[dp[mid+1]]:
47                 return mid+1
48             elif val < nums[dp[mid]]:
49                 end = mid - 1
50             else:
51                 start = mid + 1
52         return -1

```

Listing 138: Problem300. Longest Increasing Subsequence

## 7.9 leetcode 198. House Robber

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.

解题思路：标准的DP算法。这里给出两种实现：第一种实现使用一个 $n$ 长的数组 $dp[i]$ 记录到第 $i$ 个房子的时候，根据抢钱的约束条件，可以得到的最大钱数取决于（第 $i-1$ 个房子可以得到的钱数）与（第 $i-2$ 个房子可以得到的钱数+第 $i$ 个房子里面的钱数）两者最大值；第二种实现的算法设计与第一种相同，只是用两个变量记录过程计算的临时结果，进一步降低空间复杂度。参考下面示例图中的示例。

```

1  class Solution(object):
2      def rob_dp1(self, nums): # RT: O(n), Space: O(n)
3          """
4              :type nums: List[int]
5              :rtype: int
6              """
7
8          n = len(nums)
9          if n==0: return 0
10         dp = [0 for _ in xrange(n)]
11         dp[0] = nums[0]
12         for i in xrange(1, n):
13             if i==1:
14                 dp[i] = nums[i] if dp[i-1]<nums[i] else dp[i-1]
15             continue
16             dp[i] = max(dp[i-1], dp[i-2]+nums[i])

```

```

17         return dp[-1]
18
19     def rob_dp2( self ,  nums): # RT: O(n) ,  Space: O(1)
20         """
21             :type  nums:  List[int]
22             :rtype:  int
23         """
24         n = len(nums)
25         if n==0: return 0
26         if n==1: return nums[0]
27         pre , curr = nums[0] , max(nums[0:2])
28         for i in xrange(2, n):
29             pre = max(curr , pre+nums[i])
30             pre , curr = curr , pre
31
31     return curr

```

Listing 139: Problem198. House Robber

$i : 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7$   
 $A[i]: 8, 7, 5, 3, 9, 4, 6, 8.$

$T[i]: 8, 8, 13, 13, 22, 22, 28, 30.$

$T[i]$ : 前*i*个房子可以抢到的最大钱数。

$$T[i] = \begin{cases} A[0] & \text{if } i=0 \\ \max(A[0], A[1]), & \text{if } i=1 \\ \max(T[i-2]+A[i], T[i-1]), 0. \text{W.} & \end{cases}$$

## 7.10 leetcode 213. House Robber II

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. 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.

解题思路：这道题与House Robber的区别在于房子构成一个环形，而抢劫的约束条件不变。差别在于，如果抢劫第一栋房子，那么除了第二栋房子，最后一栋房子也不能抢。所以，解题最直接的手段就是打破这个环形布局为线性布

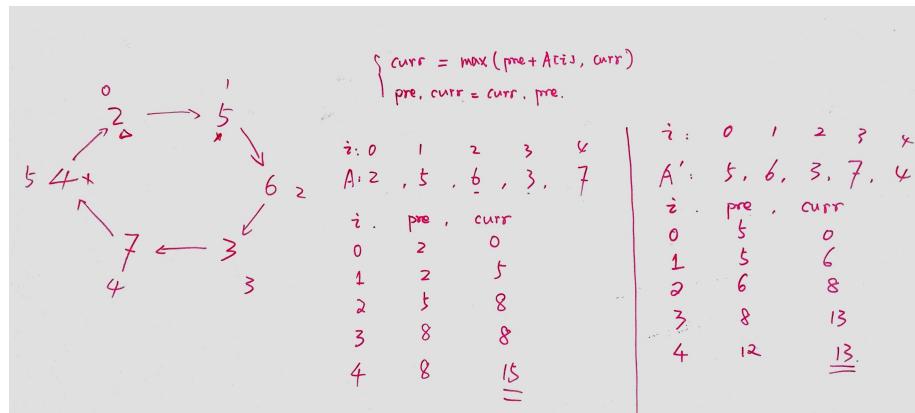
局，这样就可以使用House Robber的算法解题。根据题目中的描述，环形布局可以拆成两个线性布局，即 $\text{nums}[0..n-2]$ 和 $\text{nums}[1..n-1]$ 。拆分完成后，就可以调用两次House Robber中的算法，分别求出两种线性布局各自的最大值，最后返回两者大的一个即得到最后结果。参考下面示例图中的演算过程。

```

1  class Solution(object):
2      def rob(self, nums):
3          """
4              :type nums: List[int]
5              :rtype: int
6          """
7
8          if nums == None or len(nums) == 0: return 0
9          elif len(nums) == 1: return nums[0]
10         # As the houses form a circle, if rob the 1st house,
11         # you cannot rob the last one, so need 2 dp scan.
12         return max(self.roblinear(nums[:len(nums)-1]), self.
13                 roblinear(nums[1:]))
14
15     def roblinear(self, nums):
16         n = len(nums)
17         if n==1: return nums[0]
18         dp = [nums[0]]
19         dp.append(max(nums[1], dp[0]))
20         for i in range(2, n):
21             dp.append(max(dp[i-1], dp[i-2]+nums[i]))
22
23     return dp[n-1]

```

Listing 140: Problem213. House Robber II



## 7.11 leetcode 321. Create Maximum Number

Given two arrays of length m and n with digits 0-9 representing two numbers. Create the maximum number of length k  $\leq m + n$  from digits of the two. The relative order of the digits from the same array must be preserved. Return an array of the k digits. You should try to optimize your time and space complexity.

- Example 1:

- $\text{nums1} = [3, 4, 6, 5]$
- $\text{nums2} = [9, 1, 2, 5, 8, 3]$
- $k = 5$
- return  $[9, 8, 6, 5, 3]$

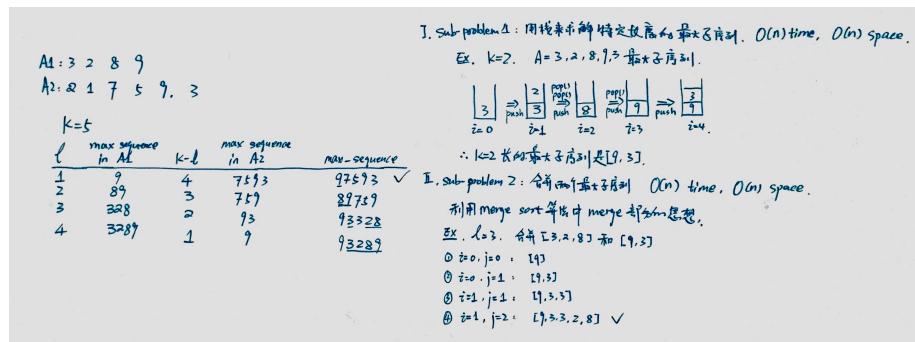
- Example 2:

- $\text{nums1} = [6, 7]$
- $\text{nums2} = [6, 0, 4]$
- $k = 5$
- return  $[6, 7, 6, 0, 4]$

- Example 3:

- $\text{nums1} = [3, 9]$
- $\text{nums2} = [8, 9]$
- $k = 3$
- return  $[9, 8, 9]$

解题思路：这道题可以分解为两个子问题：求解一个数组中特定长度的最大值子序列；保持数组中各个元素相对位置不变的情况下，合并两个数组获得最大序列。对于子问题1，可以利用栈保存最大值子序列，时间复杂度为 $O(n)$ ， $n$ 为数组长度。对于子问题2，两数组的合并可以类比归并排序中的merge操作，只不过在选择两数组中较大的元素时，需要对数组剩余部分的元素进行比较。



```

1 class Solution(object):
2     def maxNumber(self, nums1, nums2, k):
3         """
4             :type nums1: List[int]
5             :type nums2: List[int]
6             :type k: int
7             :rtype: List[int]
8

```

```

9
10     """
11     m, n = len(nums1), len(nums2)
12     res = []
13     for i in range(k+1):
14         j = k-i
15         if i>m or j>n: continue
16         tmp1 = self.getMax(nums1, i)
17         tmp2 = self.getMax(nums2, j)
18         # replace tmp = self.merge(tmp1, tmp2) to next line
19         tmp = [max(tmp1,tmp2).pop(0) for _ in tmp1 + tmp2]
20         res = max(res, tmp)
21     return res
22
23     # get the max k-length largest sequence in nums
24     def getMax(self, nums, k):
25         stack = []
26         size = len(nums)
27         for x in range(size):
28             while stack and size-x > k-len(stack) and stack[-1]<
29                 nums[x]:
30                 stack.pop()
31             if len(stack) < k:
32                 stack.append(nums[x])
33     return stack
34
35     # merge two subsequence into the largest one
36     # and keep the relative order of the digits in
37     # each array
38     def merge(self, nums1, nums2):
39         res = []
40         while nums1 or nums2:
41             if nums1 > nums2:
42                 res.append(nums1.pop(0))
43             else:
44                 res.append(nums2.pop(0))
45     return res

```

Listing 141: Problem321. Create Maximum Number

## 7.12 leetcode 322. Coin Change

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.

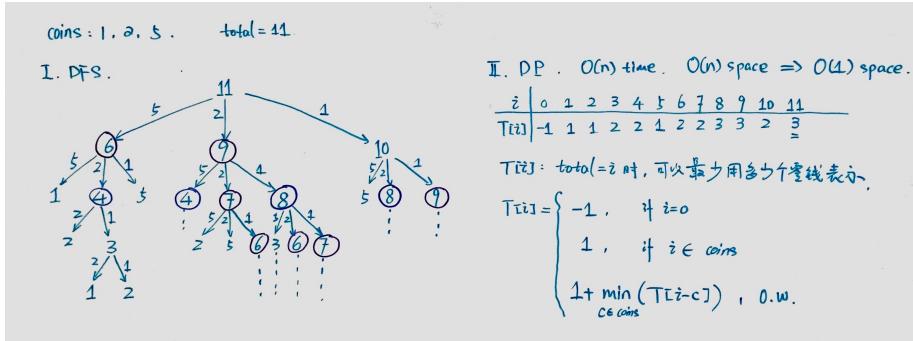
Example 1: coins = [1, 2, 5], amount = 11, return 3 ( $11 = 5 + 5 + 1$ )

Example 2: coins = [2], amount = 3, return -1.

Note: You may assume that you have an infinite number of each kind of coin.

阶梯思路：如下图所示，图中左侧是根据Example1构建决策树，图中圆圈表示的结点即是出现重复计算的结点。因为只需要返回构成指定数目的最小钱币数，所以可以使用DP算法解决。如图中右侧表格所示，需要一个长度等于amount的数组T用于存储中间过程结果，以避免重复计算。T[i]表示当amount为i时，所需的最少硬币数；初始值T[0]=0，转换方程： $T[i] = 1 + \min(T[i-1], T[i-2], \dots, T[i-coins[j]] + 1)$

$\min(T[i - c_1], T[i - c_2], \dots, T[i - c_{\text{len}(\text{coins})-1}])$ 。代码实现见line3-22。另一种实现更为简洁，参见代码line24-38。



```

1  class Solution(object):
2      def coinChange_dp1(self, coins, amount):
3          coins.sort()
4          if amount==0: return 0
5          if amount < coins[0]: return -1
6          dp = [-1 for _ in xrange(amount+1)]
7          candidates = []
8          for c in coins:
9              if c <= amount:
10                 dp[c] = 1
11                 candidates.append(c)
12
13             for i in xrange(candidates[0]+1, amount+1):
14                 if i not in coins:
15                     minvalue = -1
16                     for coin in candidates:
17                         if i>=coin and dp[i-coin]!=-1:
18                             if minvalue == -1: minvalue=dp[i-coin]
19                             else: minvalue = min(minvalue, dp[i-coin])
20
21             dp[i] = 1 + minvalue if minvalue > 0 else minvalue
22
23     return dp[-1]
24
25
26     :type coins: List[int]
27     :type amount: int
28     :rtype: int
29
30     dp = [0] + [-1] * amount
31     for x in range(amount):
32         if dp[x] < 0: continue
33         for c in coins:
34             if x+c > amount:
35                 continue
36             if dp[x+c]<0 or dp[x+c]>dp[x]+1:
37                 dp[x+c] = dp[x]+1
38
39     return dp[amount]

```

Listing 142: Problem322. Coin Change

## 7.13 leetcode 312. Burst Balloons

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 you burst balloon i you will get  $\text{nums}[\text{left}] * \text{nums}[\text{i}] * \text{nums}[\text{right}]$  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.

Example: Given [3, 1, 5, 8], Return 167.

$\text{nums} = [3, 1, 5, 8] : 3 \times 1 \times 5 \rightarrow [3, 5, 8] : 3 \times 5 \times 8 \rightarrow [3, 8] : 3 \times 8 \rightarrow [8] : 8 \rightarrow [] : 167$

Note:

1. You may imagine  $\text{nums}[-1] = \text{nums}[n] = 1$ . They are not real therefore you can not burst them.
2.  $0 \leq n \leq 500, 0 \leq \text{nums}[i] \leq 100$

解题思路：在分析这道题的时候，我们可以先假设我们只有一个气球，那么刺破这个气球所得到的最大币值就是这个气球表示的币值；如果假设我们有两个气球，那么就由两种顺序刺破这两个气球，在两个气球所表示的币值不同时，这两种不同的刺破顺序，也会得到不同的总币值：比如在[1,...,3,5,...,1]中，左右两个1是额外加上去辅助计算的边界，而不是实际的气球。如果3是最后一个被刺破的气球对应币值，那么刺破这个气球可以获得的总币值是 $(1 \times 3 \times 1) + ([1,...,3] \text{区间的最大币值}) + ([3,...,1] \text{区间的最大币值})$ ；类似地，如果5是最后一个被刺破的气球，那么可以获得的总币值是 $(1 \times 5 \times 1) + ([1,...,5] \text{区间的最大币值}) + ([5,...,1] \text{区间的最大币值})$ 。而我们应该选择刺破哪一个才可以保证我们可以获得最大的总币值呢？这就依赖四个子问题：[1,...,3]区间的最大币值、[3,...,1]区间的最大币值、[1,...,5]区间的最大币值、以及[5,...,1]区间的最大币值。分析到这里，我们可以看出问题的解决方法：将每一个气球*i*作为最后一个被刺破的，然后将整个问题分解为两个更小的问题，即对[0...i-1]和[i+1...n-1]两个气球组分别求解可以获得的最大币值。而整个问题的最终解就是这三个部分总和最大的一个。由于计算过程中有重复计算的情况出现，所以使用一个二维数组dp记录中间过程值。dp[i][j]表示在索引*i*和*j*范围内，可以获得的最大币值，其中*i*和*j*是边界索引，因此不刺破这两处的气球。转换方程： $dp[i][j] = \max(dp[i][j], dp[i][k] + \text{nums}[i] * \text{nums}[k] * \text{nums}[j] + dp[k][j])$ 。计算从左右两个边界中间只包含一个元素的情况开始考虑。示例及计算过程见下图。

```
1 class Solution(object):
2     def maxCoins(self, nums):
3         """
4             :type nums: List[int]
5             :rtype: int
6             """
7             newnums = [1] + nums + [1]
8             n = len(newnums)
9             dp = [[0]*n for _ in range(n)]
10            # the length of each span
11            for length in range(2,n):
```

```

13     # the left bound of the range to check
14     for start in range(n-length):
15         # the right bound of the range to check
16         end = start+length
17         for lastBalloon in range(start+1,end):
18             dp[start][end] = max(dp[start][end], dp[start][
19                 lastBalloon]+newnums[start]*newnums[lastBalloon]*newnums[end]+
dp[lastBalloon][end])
20     return dp[0][n-1]

```

Listing 143: Problem312. Burst Balloons

$\text{len}=3$	$\text{len}=5$
$\begin{array}{l} \text{I. } 1, 3, 1, 5 : \\ \quad \text{① } x=1, 0+13+15=30 \vee \\ \quad \text{② } x=1, 3+5+0=15 \end{array}$	$A = 1, 3, 1, 5, 8, 1$
$\begin{array}{l} \text{I. } 3, 1, 5, 8 : \\ \quad \text{① } x=1, 0+24+40=64 \\ \quad \text{② } x=5, 15+120+0=135 \vee \end{array}$	$\begin{array}{ c c c c c c } \hline 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & 0 & 0 & 30 & 159 & 167 \\ \hline 1 & 0 & 0 & 0 & 15 & 159 & 167 \\ \hline 2 & 0 & 0 & 0 & 0 & 40 & 48 \\ \hline 3 & 0 & 0 & 0 & 0 & 0 & 40 \\ \hline \end{array}$ $\text{len}=5$
$\begin{array}{l} \text{II. } 1, 5, 8, 1 : \\ \quad \text{① } x=5, 0+5+60=65 \\ \quad \text{② } x=8, 60+8+0=68 \vee \end{array}$	$\begin{array}{ c c c c c c } \hline 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 3 & 1 & 5 & 8 & 1 \\ \hline 0 & 0 & 0 & 15 & 159 & 167 \\ \hline 1 & 0 & 0 & 0 & 0 & 0 \\ \hline 2 & 0 & 0 & 0 & 0 & 0 \\ \hline 3 & 0 & 0 & 0 & 0 & 0 \\ \hline \end{array}$ $\text{len}=4$
$\begin{array}{l} \text{len}=4 \\ \text{I. } 0, 1, 2, 4 : \\ \quad \text{① } x=2, 0+24+125=159 \vee \\ \quad \text{② } x=4, 3+8+40=51 \\ \quad \text{③ } x=5, 30+10+0=70 \end{array}$	$\begin{array}{ c c c c c c } \hline 0 & 1 & 2 & 4 & 0 & 0 \\ \hline 1 & 3 & 1 & 5 & 8 & 1 \\ \hline 2 & 0 & 0 & 0 & 0 & 0 \\ \hline 3 & 0 & 0 & 0 & 0 & 0 \\ \hline 4 & 0 & 0 & 0 & 0 & 0 \\ \hline 5 & 0 & 0 & 0 & 0 & 0 \\ \hline \end{array}$ $\text{len}=2$
$\begin{array}{l} \text{II. } 1, 2, 3, 5 : \\ \quad \text{① } x=1, 0+3+48=51 \\ \quad \text{② } x=5, 15+15+40=70 \\ \quad \text{③ } x=8, 135+240+0=159 \vee \end{array}$	$\begin{array}{ c c c c c c } \hline 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 3 & 1 & 5 & 8 & 1 \\ \hline 2 & 0 & 0 & 15 & 159 & 167 \\ \hline 3 & 0 & 0 & 0 & 0 & 0 \\ \hline 4 & 0 & 0 & 0 & 0 & 0 \\ \hline 5 & 0 & 0 & 0 & 0 & 0 \\ \hline \end{array}$ $\text{len}=3$

## 7.14 leetcode 121. Best Time to Buy and Sell Stock

Say you have an array for which the  $i$ th 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.

解题思路：分析可知，当天可以获得的最大利润取决于(1)当天的股票价格与目前为止这只股票的最低买入价格的差值，以及(2)前一天可以获得的最大利润，两者中值较大的即为当天可以获得的最大利润。因此，DP算法即可解决。 $dp[x]$ 表示第 $x$ 天的最大利润。 $\text{minprice}$ 表示到第 $x$ 天为止股票的最低价格。转换方程： $dp[x] = \max(\text{prices}[x]-\text{minprice}, dp[x-1])$ 。示例及计算过程见下图。

```

1
2 class Solution(object):
3     def maxProfit(self, prices):
4         """
5             :type prices: List[int]
6             :rtype: int
7         """
8         n = len(prices)
9         if n==0: return 0
10        dp = [0 for _ in range(n)]
11        minprice = prices[0]
12        for x in range(1, n):
13            if prices[x] > minprice:
14                dp[x] = max(prices[x]-minprice, dp[x-1])
15            else:
16                minprice = prices[x]

```

```

17     dp[x] = dp[x-1]
18
19     return dp[n-1]

```

Listing 144: Problem121. Best Time to Buy and Sell Stock

i	0	1	2	3	4	5	6
prices[i]	4	5	3	10	1	9	4
minprice	4	4	3	3	1	1	1
profits[i]	0	1	1	7	7	8	8

minprice : 当前出现过的最低股票价格.

profits[i] : 前*i*天可以获得的最大利润.

$$\text{profits}[i] = \begin{cases} 0 & \text{if } i=0 \\ \max(\text{profits}[i-1], \text{prices}[i] - \text{minprice}), \text{ o.w.} \end{cases}$$

### 7.15 leetcode 309. Best Time to Buy and Sell Stock with Cooldown

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) with the following restrictions:

1. You may not engage in multiple transactions at the same time (i.e., you must sell the stock before you buy again).
2. After you sell your stock, you cannot buy stock on next day. (i.e., cooldown 1 day)

Example:

- prices = [1, 2, 3, 0, 2]
- maxProfit = 3
- transactions = [buy, sell, cooldown, buy, sell]

解题分析：这道题独特的地方在于增加了cooldown的规则，即卖了股票之后的第二天不能买股票。此外，由于对交易次数没有限制，在一天之内最多可以完成一次买卖，所以需要实时记录每一天买卖的情况，这就需要用两个数组holdDP和notHoldDP记录每一天买卖股票的收益：holdDP[i]表示第*i*持有股

票的收益； $\text{notHoldDP}[i]$ 表示第*i*天不持有股票的收益。如果在第*i*天持有股票，那么有两种情况：(1)第*i*-1天已经买入或者持有股票了，又或者(2)第*i*天新买入股票。针对这两种情况，利润的计算方法是：(1)对于第一种情况，因为没有花钱，所以利润是 $\text{holdDP}[i-1]$ ；(2)对于第二种情况，花钱购入新股票，就要在现有最大利润中扣除买新股票的钱，所以利润是 $\text{notHoldDP}[i-2]-\text{prices}[i]$ 。这里需要说明不是 $\text{notHoldDP}[i-1]-\text{prices}[i]$ 的原因，因为cooldown规则要求前一天卖出股票，第二天不能买入，要停一天。再来看，在第*i*天不持有股票的情况，也分为两种：(1)第*i*-1天已经把股票卖了，那么当前最大利润就因为没有卖出而没有变化；(2)第*i*天把持有的股票卖出，那么利润就是卖出时当天的价格与前一天在购入股票上的花销的差值。综上，得到转换方程：

- $\text{notHoldDP}[i] = \max(\text{notHoldDP}[i-1], \text{holdDP}[i-1]+\text{prices}[i])$
- $\text{holdDP}[i] = \max(\text{holdDP}[i-1], \text{notHoldDP}[i-2]-\text{prices}[i])$

这里需要额外说明：因为买入或者持有股票是花钱或者钱没在自己口袋里，所以被视为负利润，因此初始化的时候 $\text{holdDP}[0]$ 是负值，这样每次计算 $\text{notHoldDP}[i]$ 时，如果是当天卖出股票往兜里装钱，那么收益就是前一天持有股票的收益加上当天股票价格，而不是减去。

$i$	0	1	2	3	4	5	6
$\text{prices}[i]$	3	5	6	2	6	4	3
$\text{hold}$	-3	-3	-3	0	0	0	3
$\text{not\_hold}$	0	2	3	3	6	6	6

→ 大值为结果

1.  $\text{hold}[i]$ : 第*i*天持有股票时，拥有的最大利润。  
 $\text{not\_hold}[i]$ : 第*i*天不持有股票时，拥有的最大利润。

2.  $\text{hold}[i] = \begin{cases} -\text{prices}[0], & \text{if } i=0 \\ \max(\text{hold}[i-1], \text{not\_hold}[i-2]-\text{prices}[i]), & \text{o.w.} \end{cases}$

$\text{not\_hold}[i] = \begin{cases} 0, & \text{if } i=0 \\ \max(\text{not\_hold}[i-1], \text{hold}[i]+\text{prices}[i]), & \text{o.w.} \end{cases}$

```

1 class Solution(object):
2     def maxProfit(self, prices):
3         """
4             :type prices: List[int]
5             :rtype: int
6             """
7             size = len(prices)
8             if size < 2:
9                 return 0
10            holdDP = [None] * size
11            notHoldDP = [None] * size
12            notHoldDP[0], notHoldDP[1] = 0, max(0, prices[1] - prices
13            [0])

```

```

14     holdDP[0], holdDP[1] = -prices[0], max(-prices[0], -prices
15         [1])
16         for x in range(2, size):
17             notHoldDP[x] = max(notHoldDP[x-1], holdDP[x-1] +
18                 prices[x])
19             holdDP[x] = max(holdDP[x-1], notHoldDP[x-2] - prices[
20                 x])
21
22     return notHoldDP[-1]

```

Listing 145: Problem309. Best Time to Buy and Sell Stock with Cooldown

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

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 at most two transactions. Note: You may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

解题思路：因为交易次数最多为两次，并且两次交易必须是第一次交易完成之后才能进行第二次交易，所以可以把整个过程分为两个阶段，而这两个阶段如同两个彼此消长的交易窗口，分别求出在不同窗口尺寸的情况下，两个窗口各自获得的最大利润，并保存到 $\text{profits1}[i]$ 和 $\text{profits2}[i]$ 中；最后再求出 $\text{profits1}[i]+\text{profits2}[i]$ 的最大值，即为两次交易后可以获得的最大利润。参考示例图中的示例和演算过程。

$i$	0	1	2	3	4	5	6
$\text{prices}[i]$	3	5	6	2	6	4	3
$\text{profits1}$	0	2	3	4	4	4	4
$\text{profits2}$	4	4	4	4	0	0	0

- ①  $\text{profits1}[i]$ : 第一次交易发生在第*i*天时，可以获得的最大利润。  
 $\text{profits2}[i]$ : 第二次交易发生在第*i*天以后可以获得的最大利润。

②  $\text{profits1}[i] = \begin{cases} 0, & \text{if } i=0 \\ \max(\text{profits}[i-1], \text{prices}[i] - \min\text{price}), & \text{o.w.} \end{cases}$

$\text{profits2}[i] = \begin{cases} 0, & \text{if } i=n-1 \\ \max(\text{profits}[i+1], \max\text{price} - \text{prices}[i]), & \text{o.w.} \end{cases}$

③  $\text{maxprofit} = \max_{0 \leq i \leq n-1} (\text{profits1}[i] + \text{profits2}[i])$

```

1
2 class Solution(object):
3     def maxProfit(self, prices):
4         """
5             :type prices: List[int]
6             :rtype: int
7         """
8         n = len(prices)
9         if n==0: return 0
10        # the profits gained in the first i days
11        profits1 = [0 for _ in range(n)]
12        # the profits gained after i days
13        profits2 = [0 for _ in range(n)]
14
15        # compute the profits gained in the first transaction
16        minprice = prices[0]
17        profits1[0] = 0
18        for i in range(1,n):
19            profits1[i] = max(profits1[i-1], prices[i]-minprice)
20            minprice = min(minprice, prices[i])
21
22        # compute the profits gained in the second transaction
23        maxprice = prices[n-1]
24        profits2[n-1] = 0
25        for i in range(n-2,-1,-1):
26            profits2[i] = max(profits2[i+1], maxprice-prices[i])
27            maxprice = max(maxprice, prices[i])
28
29        # merge the profits gained in these two transactions
30        maxprofit = 0
31        for i in range(n):
32            maxprofit = max(maxprofit, profits1[i]+profits2[i])
33        return maxprofit

```

Listing 146: Problem123. Best Time to Buy and Sell Stock III

## 7.16 leetcode 188. Best Time to Buy and Sell Stock IV

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 at most  $k$  transactions. Note: You may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

解题思路：问题的实质是从长度为n的prices数组中挑选出至多 $(2 \times k)$ 个元素，组成一个交易（买卖）序列。交易序列中的首次交易为买入，其后卖出和买入操作交替进行。总收益为交易序列中的(偶数项之和-奇数项之和)。dp[j]表示完成j次交易时的最大收益，转移方程如下： $dp[j] = \max(dp[j], dp[j - 1] + prices[i] * [1, -1][j \% 2])$ 。当j为奇数时，交易类型为买入；当j为偶数时，交易类型为卖出。为避免超时，令最大交易次数为k，数组长度为size，则当 $k > size / 2$ 时，问题可以转化为leetcode122 Best Time to Buy and Sell Stock II。参考示例图中的示例及演算过程。

```

1
2 class Solution(object):
3     def maxProfit(self, k, prices):
4         """
5             :type k: int

```

```

6     : type prices: List[int]
7     : rtype: int
8     """
9     size = len(prices)
10    if k > size / 2:
11        return self.quickSolve(size, prices)
12    dp = [None] * (2 * k + 1)
13    dp[0] = 0
14    for i in range(size):
15        for j in range(1, min(2 * k, i + 1) + 1):
16            dp[j] = max(dp[j], dp[j - 1] + prices[i] * [1, -1][
j % 2])
17    return dp[2 * k]
18
19 def quickSolve(self, size, prices):
20     sum = 0
21     for x in range(size - 1):
22         if prices[x + 1] > prices[x]:
23             sum += prices[x + 1] - prices[x]
24     return sum

```

Listing 147: Problem188. Best Time to Buy and Sell Stock IV

i	0	1	2	3	4	5	6
prices[i]	3	5	6	2	6	4	3
j	1	2	3	4	5	6	
T[i][j]	-3	x	x	x	x	x	i=0
	-3	2	x	x	x	x	i=1
	-3	3	-3	x	x	x	i=2
	-2	3	1	3	x	x	i=3
	-2	4	1	7	1	x	i=4
	-2	4	1	7	3	7	i=5
	-2	4	1	7	4	7	i=6

↑

at most  $k$  transactions.  
 $k \leq 3$ .

奇数代表买入  
 偶数代表卖出  
 一次买入一次卖出代表一次交易。

T[i][j]: 在第*i*天时，进行第*j*次买入或卖出交易时可以获得的[最大利润](#)。

$T[i][j] = \begin{cases} 0, & \text{if } j=0 \\ \max_{1 \leq j \leq i} (\bar{T}[i][j], \bar{T}[i][j-1] + \begin{cases} \text{①买入: } -\text{prices}[i] \\ \text{②卖出: } +\text{prices}[i] \end{cases}), & \text{otherwise} \end{cases}$