# General 注意：

1. A=[0]\*10 产生的数组是互不相干的，真实的10个独立变量。

A=[[0]\*10]\*10 产生的2D数组，实际上是第二维度是引用的copy，所以当你修改一个的时候，另外两个也会变

所以二维的数组必须要用A=[[0]\*10 for \_ in range(10)]

2. 修改完程序提交的时候，看看所有的输出类型是不是都convert成要求的类型了。如果有多个return，看看是不是所有的return都按此修改了。

3. 题目多花时间像算法，少花时间写程序

4. 三种遍历 transversal

preorder就是写个function，每次先print root val， 然后tran（left）， tran（right）

inorder就是tran(left),print root, tran(right)

postorder就是tran(left),tran(right),print root

5. 如何同时输出list的中每个元素的index和val？ 用enumerate！

For i,v in enumerate(list)

6. backtracking的时候，对list或者arr，要用.copy(), 否则因为是引用，改变会影响外面。

7. If A 是个9\*9 的list，那么A[3:6][3:6] 是不对的。要想达到matlab的那种效果，需要convert A 成np array

8. List可以用+直接串联

9. 把str变成char array，直接用list(s)

10. 把float 向0方向round，用int(float) 就可以了

11. python不支持函数重载！！！ 不能用同一个函数名定义不同的参数变量！！

12. 检查p是否是None只要用 if p就可以了 e.g. if p and q

13. a==b 只能比较a和b是否指向同一个object，不能比较值

14. python里面char不能直接相减，必须要用ord(‘a’) - ord(‘b’)

15. python里面a/b 不像java那样是整除。会出小数的。必须用a//b 才是整除。尤其是二分查找法里面要小心。

16. sorted(nums)并没有在原来的地方修改所以不对。必须要用nums=sorted(nums)

nums[i+1:][::-1]才是倒序，而不是nums[i+1::-1]

17. 检查数组写坐标的时候，坐标是否大于等于0 并且小于数组的长度

18. 把一个数组的数字都convert成string，用map(str,product）

19. 不用在写a.left==None, 直接写not a.left

20. Python 中计算一个数字的bit里面有多少个1用：bin(x).count("1")

注意，这个写法不用用于count(‘0’)!!! 因为bin产生的是binary literals不是32位的表示！！！

21. 重新读一遍程序。按照自己当初想的思路，走一遍程序，看看程序是不是按照自己的思路在走。（因为很多时候，你写着写着就忘了很多事儿）这种方式是最有效最快速的 Debug 方式。

1. 找到一个非常小非常小的可以让你的程序出错的数据。比如空数组，空串，1-5个数的数组，一个字符的字符串。
2. 在程序的若干位置输出一些中间结果。比如排序之后输出一下，看看是不是真的按照你所想的顺序排序的。这样可以定位到程序出错的部分。
3. 定位了出错的部分之后，查看自己的程序该部分的逻辑是否有错。
4. 在第4步中，如果无法通过肉眼看出错误的部分，就一步步“模拟执行”程序，找出错误。

22. Python 定义inf -inf nan

float('NaN')

float(‘inf’)

-float(‘inf’)

23. Deep copy in python:

Import copy

copy.deepcopy(var)

24. Python 里面，list of list, 只copy是不行的，因为copy的是第二层list的地址。所以要deep copy

25.nums=sorted(nums) 不能inplace sort，所以不对

要想inplace，必须要nums.sort()

26. How to sort list of list:

insertion\_order = sorted(people, key = lambda (h,k): (-h,k))  
 people = sorted(people, key=lambda x: x[1])

按照多维排序：

pass1 = sorted(people, key=lambda x: (x[0], -x[1]), reverse=True)。 先按第一个参数，如果一样就按第二个。

27. python的list，如果list.insert(pos,val)可以在那个index插入，然后把后面的全部往后移动一位

28. Python 2 里面用sorted(nums,cmp=func) 可以排序

Python 3.6里面remove了cmp，所以要写成

class LargerNumKey(str):

def \_\_lt\_\_(x, y):

return x+y > y+x

sorted(map(str, nums), key=LargerNumKey)

29. Sort dictonaries by key/val

import operator  
d = {1: 2, 3: 4, 4: 3, 2: 1, 0: 0}  
print('Original dictionary : ',d)  
sorted\_d = sorted(d.items(), key=operator.itemgetter(0))  
print('Dictionary in ascending order by value : ',sorted\_d)  
sorted\_d = sorted(d.items(), key=operator.itemgetter(0),reverse=True)  
print('Dictionary in descending order by value : ',sorted\_d)

# 重要模板

### Quick select 算法 Quick Select算法的基本思路与Quick Sort类似，重点是partition 基本思想，随机选取一个pivot，小于的放左边，大于等于的放右边，返回pivot的位置 如果pivot == k，则正好找到了第k小的元素 如果pivot > k，则第k小的元素存在于pivot左边 如果pivot < k，则第k小的元素存在于pivot右边 时间复杂度：O(n logn) (in quicksort), O(n) (in quickselect)

def findKthLargest(self, nums, k):

if len(nums)==0:

return None

pivot = nums[0]

left = 0

right = len(nums)-1

p=left+1

while p<=right:

if nums[p]>pivot:

self.swap(nums,p,left)

left+=1

p+=1

elif nums[p]==pivot:

p+=1

else:

self.swap(nums,p,right)

right-=1

if left+1==k:

return pivot

if left<k-1:

return self.findKthLargest(nums[left+1:],k-left-1)

else:

return self.findKthLargest(nums[:left],k)

def swap(self,nums,a,b):

tmp = nums[a]

nums[a]=nums[b]

nums[b]=tmp

### Radix sort

基数排序 (Radix Sort) 是一种非比较型整数排序算法，其原理是将整数按位数切割成不同的数字，然后按每个位数分别比较。排序过程：将所有待比较数值（**正整数**）统一为同样的数位长度，数位较短的数前面补零。然后，从最低位开始，依次进行一次排序。这样从最低位排序一直到最高位排序完成以后, 数列就变成一个有序序列。

先按照个位排序。排完以后有10个list

然后10个list里面分别按照十位排序。 直到最高

最后把排好序的10个list连起来。

或者另一种方法见9-10以后的部分

def radix(nums,d):

buckets = [[] for \_ in range(10)]

for n in nums:

buckets[(n//10\*\*d)%10].append(n)

res=[]

for b in buckets:

res.extend(b)

return res

maxd = len(str(2\*\*32))

for d in range(maxd):

sortedNums = radix(sortedNums,d)

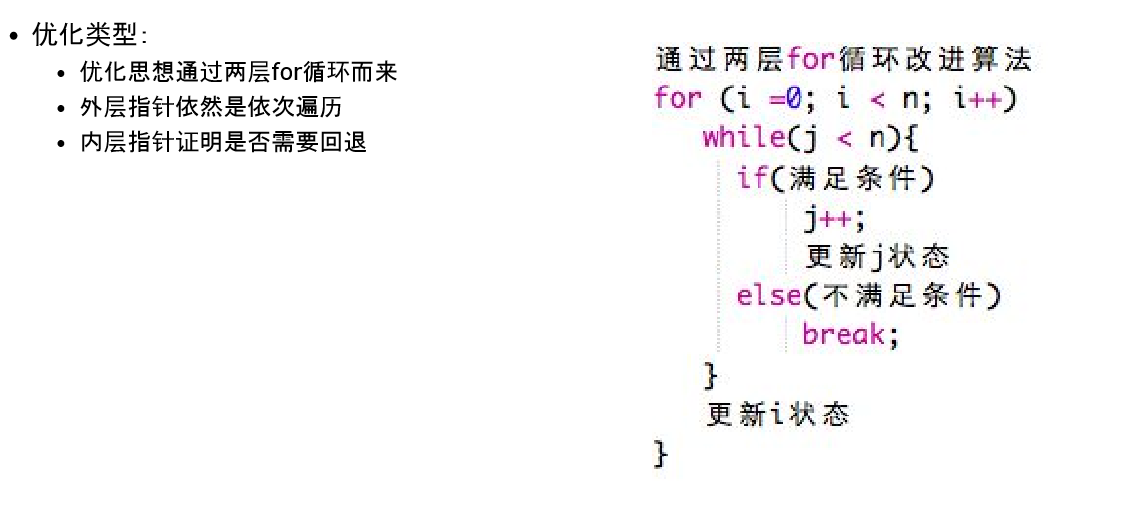
### 3. Union Find 代码

class UnionFind(object):  
 def \_\_init\_\_(self, grid):  
 self.n = len(grid)  
 self.parent = [-1] \* self.n  
 for idx in range(self.n):  
 self.parent[idx] = idx  
  
 def find(self, i):  
 if self.parent[i] != i:  
 self.parent[i] = self.find(self.parent[i])  
 return self.parent[i]  
  
 def union(self, x, y):  
 rootx = self.find(x)  
 rooty = self.find(y)  
 if rootx != rooty:  
 self.parent[rooty] = rootx  
  
 def diff\_groups(self):  
 diff\_groups = set()  
 for i in range(self.n):  
 diff\_groups.add(self.find(i))  
 return len(diff\_groups)  
  
class Solution(object):  
 def findCircleNum(self, M):  
 n = len(M)  
 uf = UnionFind(M)  
  
 for i in xrange(n):  
 for j in xrange(n):  
 if M[i][j] == 1:  
 uf.union(i, j)  
  
 return uf.diff\_groups()

### 4. DFS sample

def numIslands(self, grid):  
 if not grid:  
 return 0  
   
 count = 0  
 for i in range(len(grid)):  
 for j in range(len(grid[0])):  
 if grid[i][j] == '1':  
 self.dfs(grid, i, j)  
 count += 1  
 return count  
  
def dfs(self, grid, i, j):  
 if i<0 or j<0 or i>=len(grid) or j>=len(grid[0]) or grid[i][j] != '1':  
 return  
 grid[i][j] = '#'  
 self.dfs(grid, i+1, j)  
 self.dfs(grid, i-1, j)  
 self.dfs(grid, i, j+1)  
 self.dfs(grid, i, j-1)

### 5. Moving window



209. Minimum Size Subarray Sum

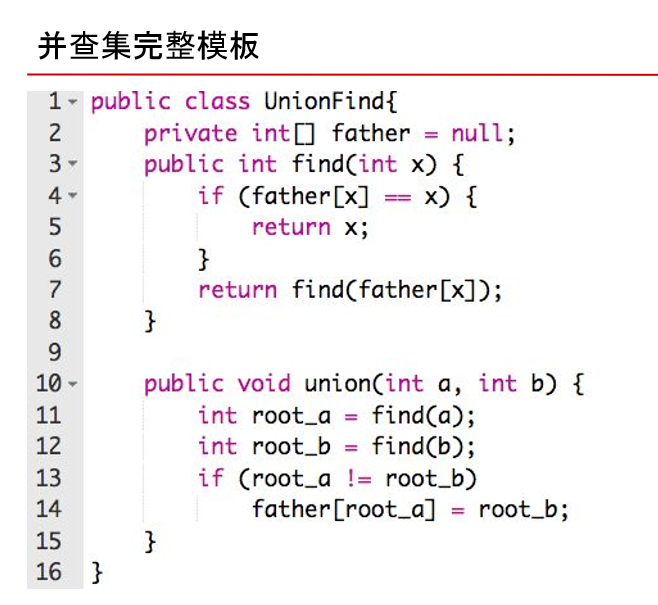
### 6. 见到集合求Min/Max 就要想到堆

时间复杂度N\*Len\*Log(len)+K\*logN (len 是平均每个数组的长度)

第k小的元素，Heap用来维护当前候选集合。

见到数组要想到先排序

### 7. Union-find 的find 和union都是O(1)

1、关于集合合并。

2、判断在不在同一个集合中。

解题你需要做的事：

把其他的操作，转化成这两件事情

200. Number of Islands

DFS. 找到一个1，就用dfs 递归把周边所有的1改成#。 然后count 加一

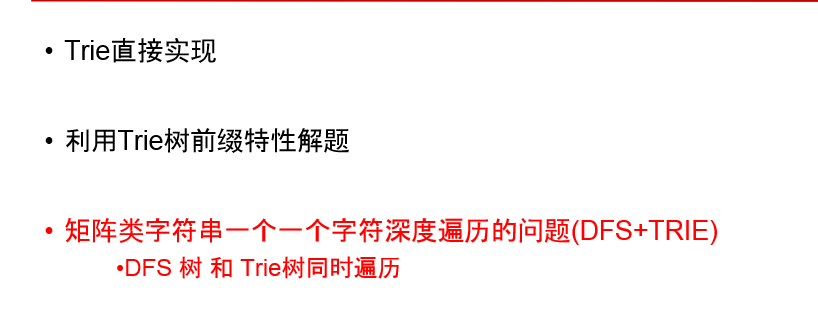
最后跑完全部以后return count

305. Number of Islands II

261. Graph Valid Tree

130. Surrounded Regions

### 8. Trie



208. Implement Trie (Prefix Tree)

211. Add and Search Word - Data structure design

212. Word Search II

425. Word Squares

5. Heap(priority queue)

42. Trapping Rain Water

407. Trapping Rain Water II

295. Find Median from Data Stream

480. Sliding Window Median

6. Stack

栈 Stack 利用栈暂且保存有效信息 + 翻转栈的运用 + 栈优化dfs，变成非递归

翻转stack 见232. Implement Queue using Stacks  
单调栈

找每个元素左边或者右边 第一个比它自身小/大的元素 用单调栈来维护

84. Largest Rectangle in Histogram

85. Maximal Rectangle  
数据结构总结

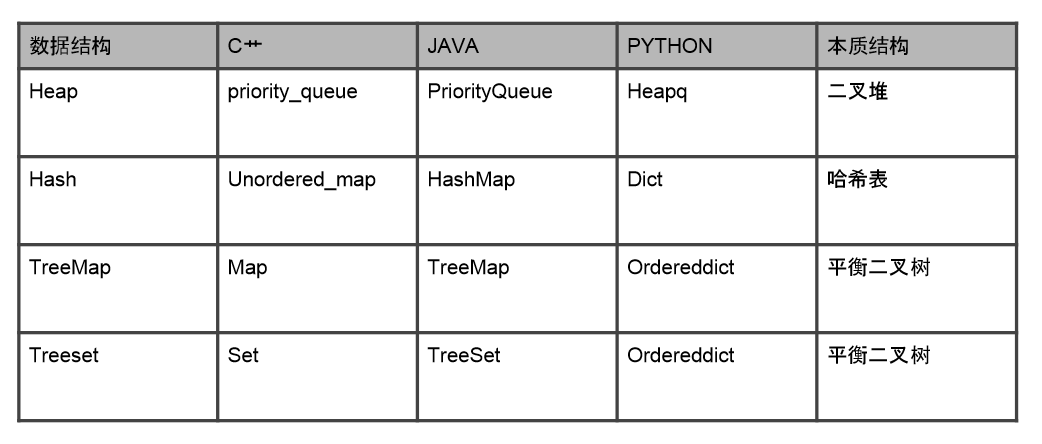
数据结构的题目：

• Heap：求集合的最大值

• Stack： 1. 单调栈的运用找左边和右边第一个比它大的元素 2. 递归转非递归

• Windows problem • a.加一个数 • b.删一个数的方法

### 数据结构查询表



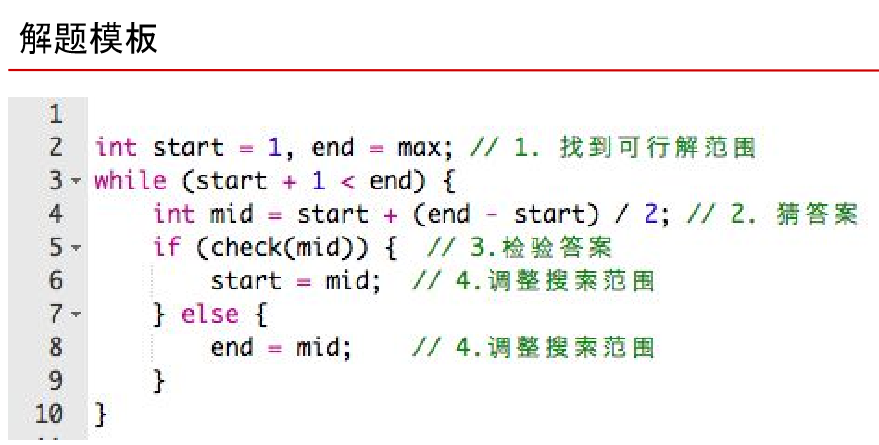
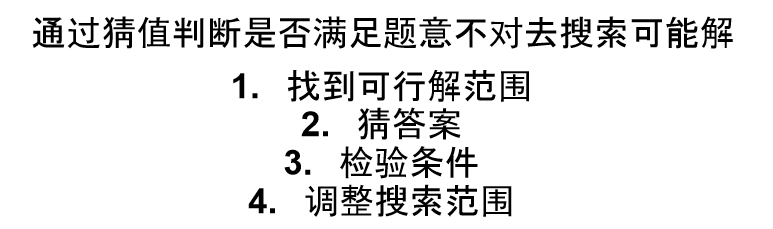
### 二分法

保留有答案的那一半

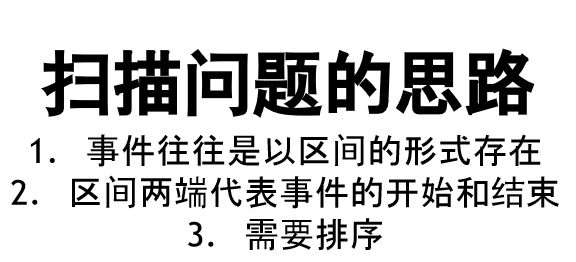
162. Find Peak Element

Binary Search on Result 往往没有给你一个数组让你二分 同样是找到满足某个条件的最大或者最小值

通过猜值判断是否满足题意不对去搜索可能解 1. 找到可行解范围 2. 猜答案 3. 检验条件 4. 调整搜索范围

69. Sqrt(x)  
287. Find the Duplicate Number

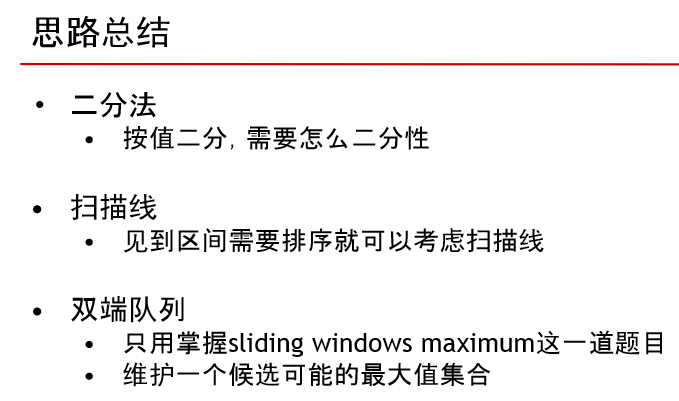
### Sweep-Line 扫描线

218. The Skyline Problem  


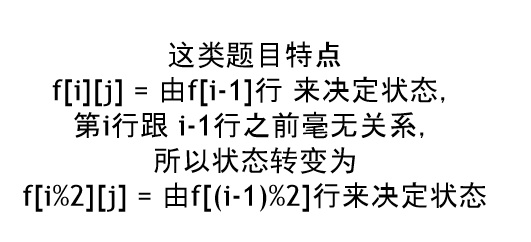
### Deque

Deque：两端都会有push和pop

239. Sliding Window Maximum



### 二维动态规划空间优化



### Memorization

什么时候用记忆化搜索？ 1. 状态转移特别麻烦，不是顺序性。 2. 初始化状态不是很容易找到。

300. Longest Increasing Subsequence

### 区间类Dp

1. 求一段区间的解max/min/count 2. 转移方程通过区间更新 3. 从大到小的更新

312. Burst Balloons

87. Scramble String

### 匹配类动态规划

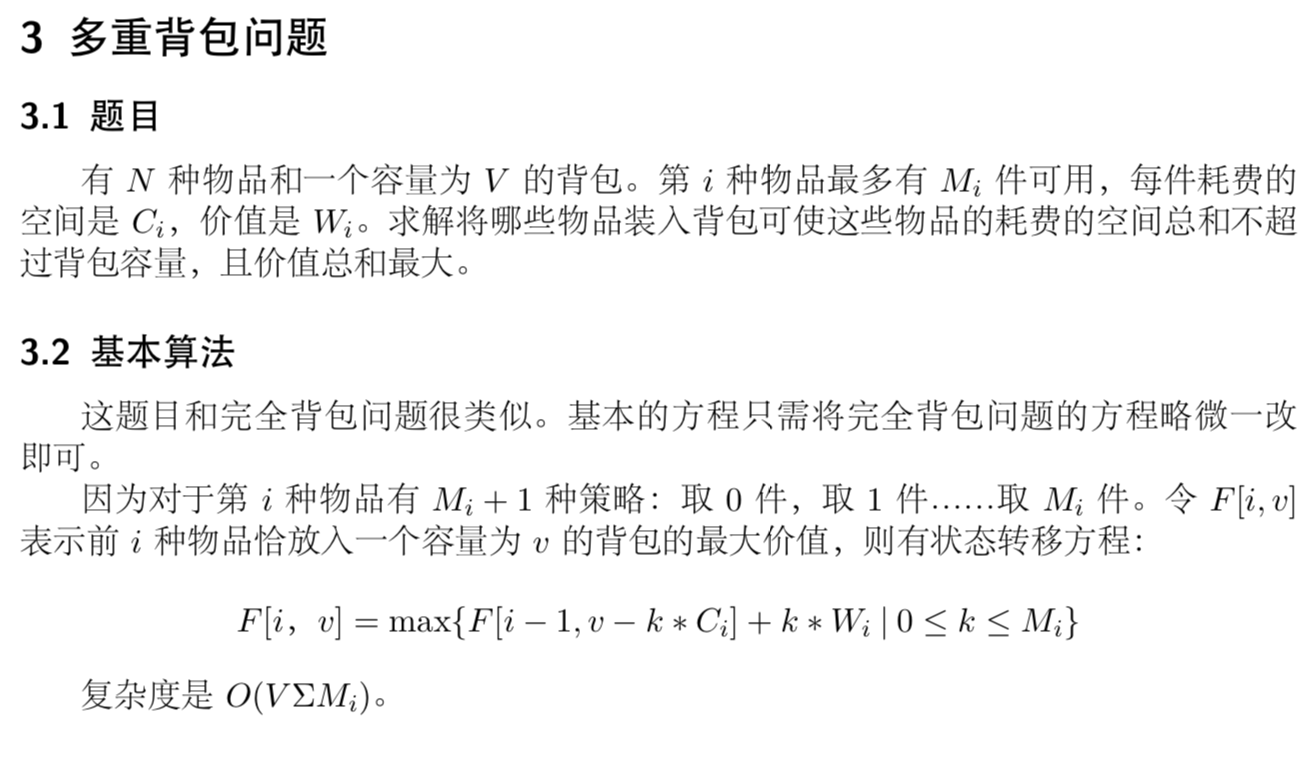
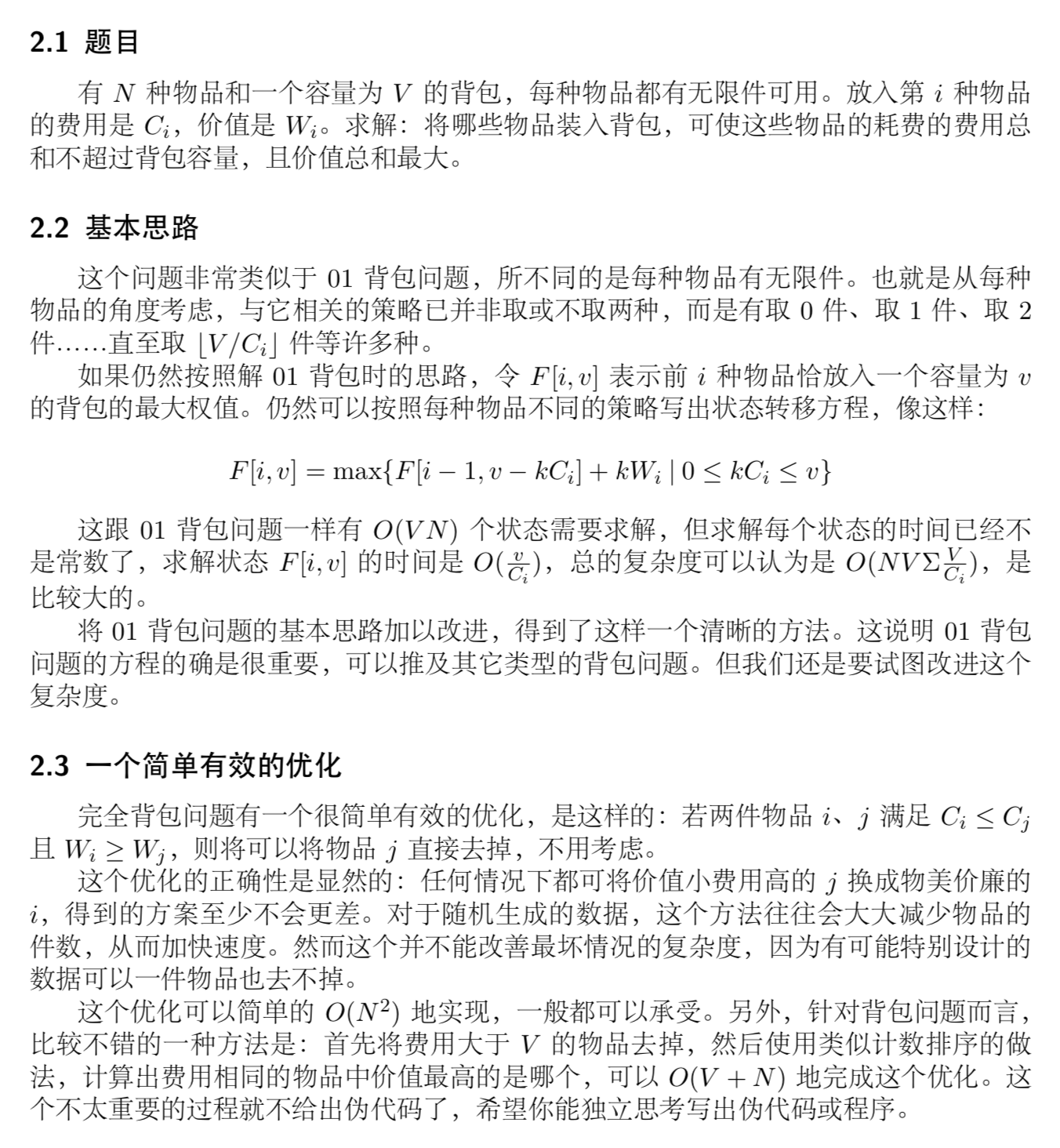
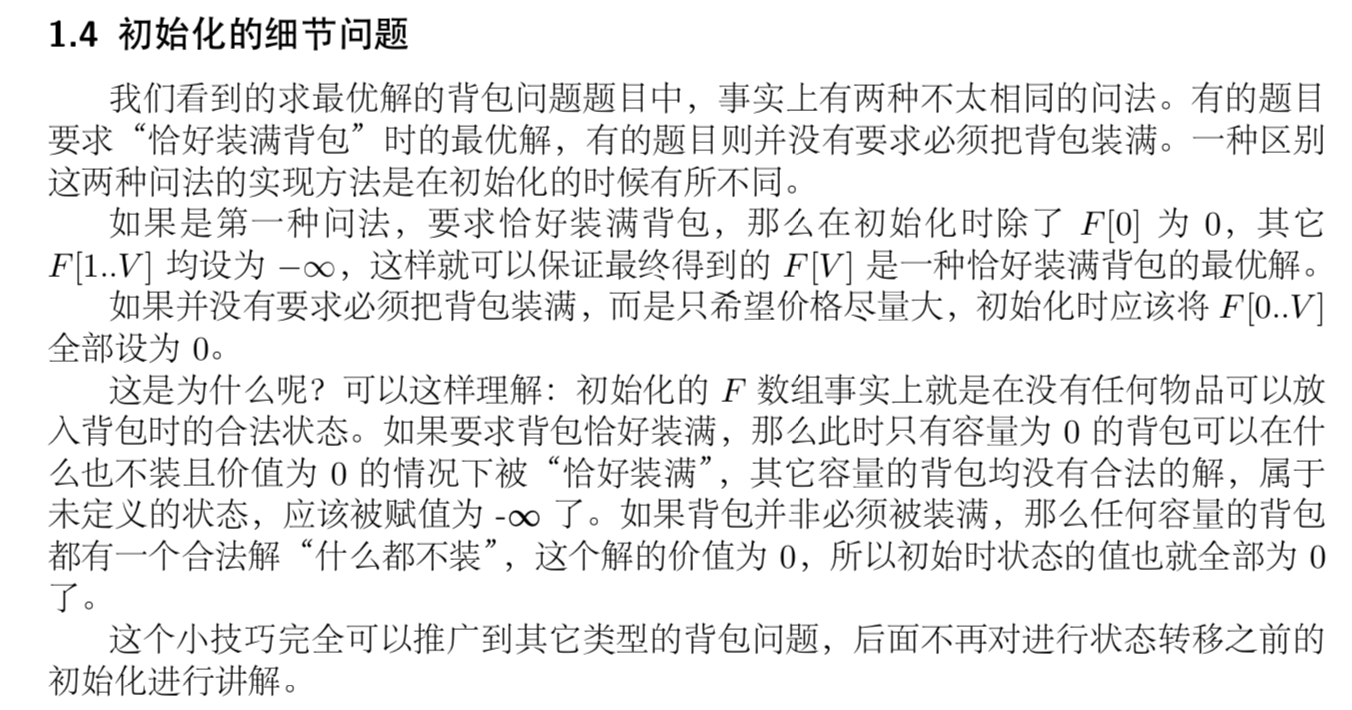
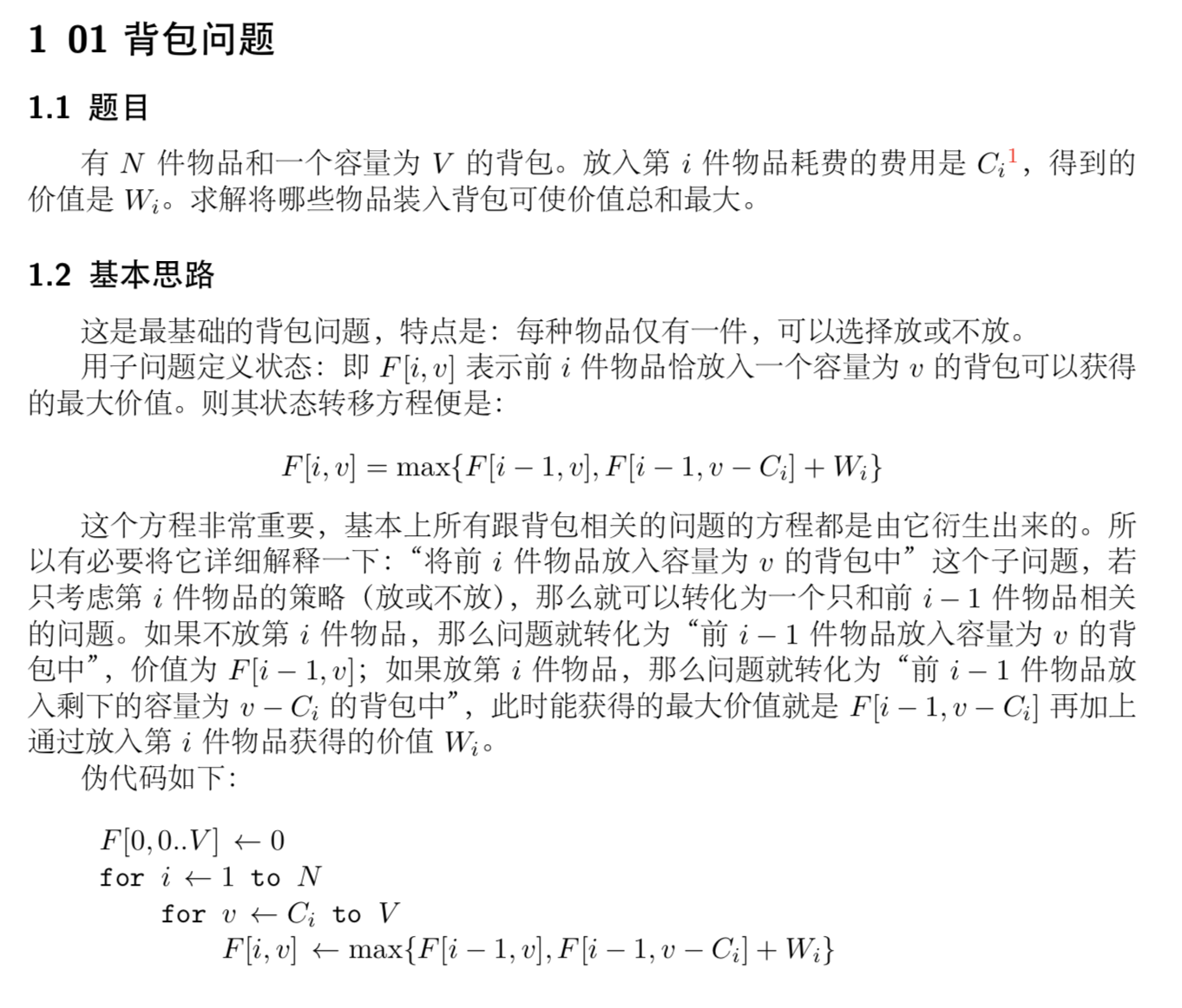
Longest Common Subsequence

Edit Distance

115. Distinct Subsequences  
97. Interleaving String

### 背包类Dp

1. 用值作为DP维度 2. Dp过程就是填写矩阵 3. 可以滚动数组优化





### Topological Sort

1. 在有向图中选一个没有前驱的顶点并且输出
2. 从图中删除该顶点和所有以它为尾的弧（白话就是：删除所有和它有关的边）
3. 重复上述两步，直至所有顶点输出，或者当前图中不存在无前驱的顶点为止，后者代表我们的有向图是有环的，因此，也可以通过拓扑排序来判断一个图是否有环。

如果我们有如下的一个有向无环图，我们需要对这个图的顶点进行拓扑排序，过程如下：

首先，我们发现V6和v1是没有前驱的，所以我们就随机选去一个输出，我们先输出V6，删除和V6有关的边，得到如下图结果：

然后，我们继续寻找没有前驱的顶点，发现V1没有前驱，所以输出V1，删除和V1有关的边，得到下图的结果：

然后，我们又发现V4和V3都是没有前驱的，那么我们就随机选取一个顶点输出（具体看你实现的算法和图存储结构），我们输出V4，得到如下图结果：

然后，我们输出没有前驱的顶点V3，得到如下结果：

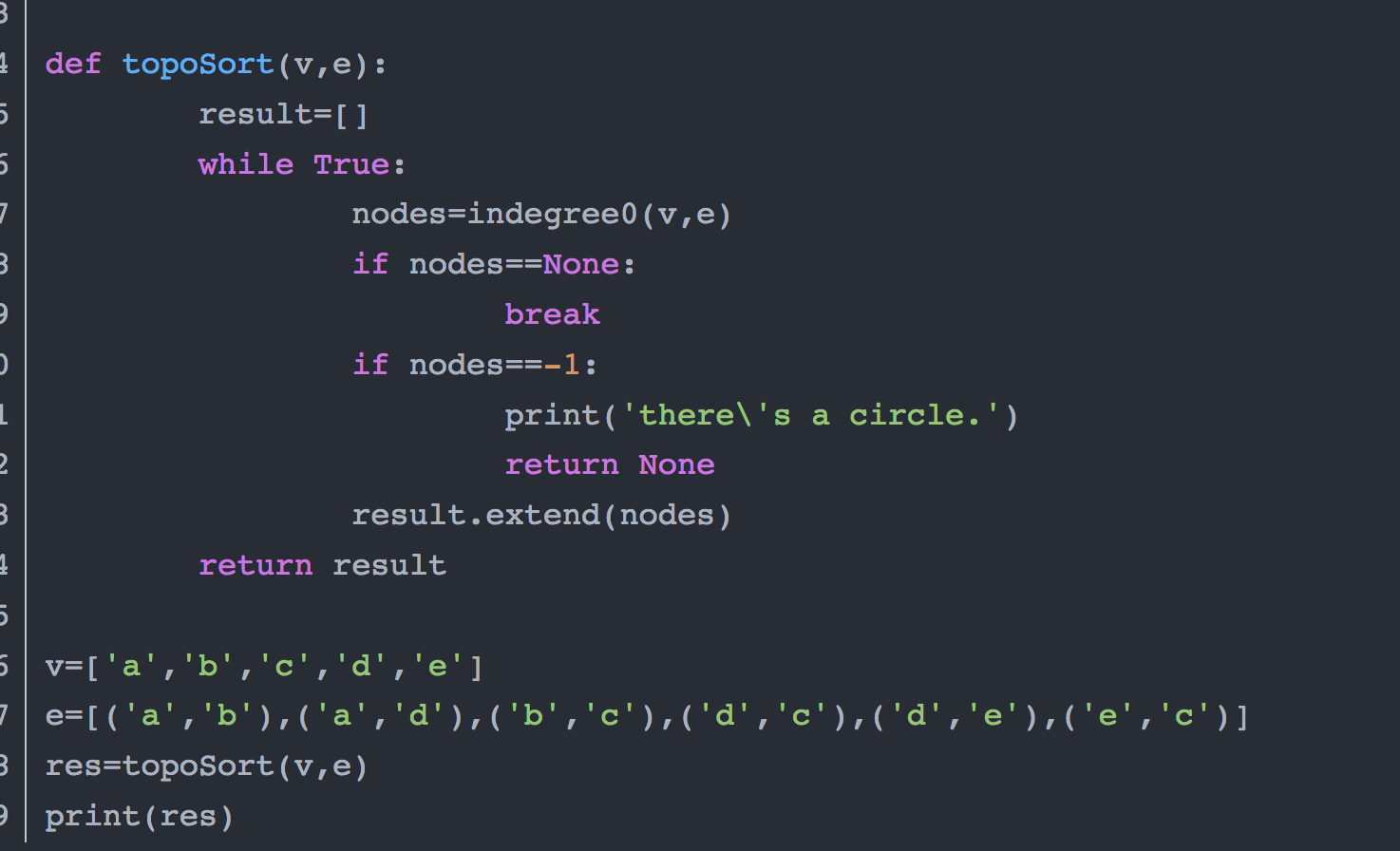
然后，我们分别输出V5和V2，最后全部顶点输出完成，该图的一个拓扑序列为：

v6–>v1—->v4—>v3—>v5—>v2

现在，我们来介绍第二个算法的思路：

其实DFS就是深度优先搜索，它每次都沿着一条路径一直往下搜索，知道某个顶点没有了出度时，就停止递归，往回走，所以我们就用DFS的这个思路，我们可以得到一个有向无环图的拓扑序列，其实DFS很像Kahn算法的逆过程。

v表示顶点：v=['a','b','c','d','e']，e表示有向边：e=[('a','b'),('a','d'),('b','c'),('d','c'),('d','e'),('e','c')]，



4、拓扑排序的代码实现

下面，我们将用两种方法来实现我么的拓扑排序：

1. Kahn算法
2. 基于DFS的拓扑排序算法

首先我们先介绍第一个算法的思路：

Kahn的算法的思路其实就是我们之前那个手动展示的拓扑排序的实现，我们先使用一个栈保存入度为0 的顶点，然后输出栈顶元素并且将和栈顶元素有关的边删除，减少和栈顶元素有关的顶点的入度数量并且把入度减少到0的顶点也入栈

### Dijkstra算法

Dijkstra算法采用的是一种贪心的策略，声明一个数组dis来保存源点到各个顶点的最短距离和一个保存已经找到了最短路径的顶点的集合：T，初始时，原点 s 的路径权重被赋为 0 （dis[s] = 0）。若对于顶点 s 存在能直接到达的边（s,m），则把dis[m]设为w（s, m）,同时把所有其他（s不能直接到达的）顶点的路径长度设为无穷大。初始时，集合T只有顶点s。

然后，从dis数组选择最小值，则该值就是源点s到该值对应的顶点的最短路径，并且把该点加入到T中，OK，此时完成一个顶点，

然后，我们需要看看新加入的顶点是否可以到达其他顶点并且看看通过该顶点到达其他点的路径长度是否比源点直接到达短，如果是，那么就替换这些顶点在dis中的值。

然后，又从dis中找出最小值，重复上述动作，直到T中包含了图的所有顶点。

# 题目解法

### 1 Two Sum.

用O（n）的复杂度，找出已排序数组里面两个数的和等于target。 输出这两个数的坐标。易错点在于，hashmap的key应该说target-nums[i] not nums[i]

### 2. Add two number

给出两个非负的linkedlist表示的数字。都是从个位往大的。输出他们的和，也要结果是linkedlist

设一个零时变量c，然后依次从第一个node开始加。每次都是两个node的和加上c。如果超过10就取mod 10， c取和/10. 如果一个数字的list已经用完，就当成0继续加。最后两个都完了，看c是不是1。 如果是1，就再加一个node。 输出新的list。 新的list先设置一个prehead。然后每次产生一个node。最后输出prehead的next

### 3. 给出一个string。找出其中没有重复的最长的substring

设一个pointer往右走，每次保持initial position和当前最大的substring。 另外还要设一个maxsub 保存全局最大的substring。 每次右移一个字母，检查字母是不是在这个substring里面。 因为substring最大26个字母，所以说O（1）

如果在substring里面，那么更新initial成这个字母的位置+initial+1， 然后把substring从那个字母右边到最后再加上新的字母。 不再substring里面就在substring加上新字母。同时检查如果长度大雨maxsub，更新maxsub。直到指针走到最后为止。

因为最长只能是26个字母，所以是O（n）

### 4. Median of two sorted array (Need record again)

该方法的核心是将原问题转变成一个寻找第k小数的问题（假设两个原序列升序排列），这样中位数实际上是第(m+n)/2小的数。所以只要解决了第k小数的问题，原问题也得以解决。

首先假设数组A和B的元素个数都大于k/2，我们比较A[k/2-1]和B[k/2-1]两个元素，这两个元素分别表示A的第k/2小的元素和B的第k/2小的元素。这两个元素比较共有三种情况：>、<和=。如果A[k/2-1]<B[k/2-1]，这表示A[0]到A[k/2-1]的元素都在A和B合并之后的前k小的元素中。换句话说，A[k/2-1]不可能大于两数组合并之后的第k小值，所以我们可以将其抛弃。

证明也很简单，可以采用反证法。假设A[k/2-1]大于合并之后的第k小值，我们不妨假定其为第（k+1）小值。由于A[k/2-1]小于B[k/2-1]，所以B[k/2-1]至少是第（k+2）小值。但实际上，在A中至多存在k/2-1个元素小于A[k/2-1]，B中也至多存在k/2-1个元素小于A[k/2-1]，所以小于A[k/2-1]的元素个数至多有k/2+ k/2-2，小于k，这与A[k/2-1]是第（k+1）的数矛盾。

当A[k/2-1]>B[k/2-1]时存在类似的结论。

当A[k/2-1]=B[k/2-1]时，我们已经找到了第k小的数，也即这个相等的元素，我们将其记为m。由于在A和B中分别有k/2-1个元素小于m，所以m即是第k小的数。(这里可能有人会有疑问，如果k为奇数，则m不是中位数。这里是进行了理想化考虑，在实际代码中略有不同，是先求k/2，然后利用k-k/2获得另一个数。)

通过上面的分析，我们即可以采用递归的方式实现寻找第k小的数。此外我们还需要考虑几个边界条件：

* 如果A或者B为空，则直接返回B[k-1]或者A[k-1]；
* 如果k为1，我们只需要返回A[0]和B[0]中的较小值；
* 如果A[k/2-1]=B[k/2-1]，返回其中一个；

最终实现的代码为：

* def findKth(self,nums1,nums2,k):  
   if len(nums1)>len(nums2):  
   return self.findKth(nums2,nums1,k)  
   if len(nums1)==0:  
   return nums2[k-1]  
   if k==1:  
   return min(nums1[0],nums2[0])  
   pa = min(len(nums1)-1,k//2-1)  
   pb = k-pa-2 % 此处易错  
   if nums1[pa]==nums2[pb]:  
   return nums2[pb]  
   if nums1[pa]<nums2[pb]:  
   return self.findKth(nums1[pa+1:],nums2,k-pa-1)  
   else:   
   return self.findKth(nums2[pb+1:],nums1,k-pb-1)  
     
     
   def findMedianSortedArrays(self, nums1, nums2):  
   """  
   :type nums1: List[int]  
   :type nums2: List[int]  
   :rtype: float  
   """  
   total = len(nums1)+len(nums2)  
   if total%2==1:  
   return self.findKth(nums1,nums2,total//2+1)  
   else:  
   return (self.findKth(nums1,nums2,total//2)+self.findKth(nums1,nums2,total//2+1))/2

我们可以看出，代码非常简洁，而且效率也很高。在最好情况下，每次都有k一半的元素被删除，所以算法复杂度为log*k*，由于求中位数时k为（m+n）/2，所以算法复杂度为log(m+*n*)。

### 5. **Longest Palindromic Substring**

输入string s，找出最长的palindromic substring

用动态规划，p(i,j) = p(i+1,j-1) & si==sj.

初始化条件是p(i,i) ==1. p(i,i+1)==si==s(i+1)

注意这里，因为i，j需要i+1，j-1的知识。 i是i+1，必须要先知道后面的，所以i必须要从大的推到小的。所以i从n-1推到0

J是需要j-1，所以j要先知道前面的，所以j从i推到n

### 6. **ZigZag Conversion**

把一列字符串zigzag convert以后，按行读出来。输入是字符串和行数

首先要注意只有一行直接输出

然后的话建立行数和数组

再设一个direction的变量，逐航放入就行了

### **7. Reverse Integer**

输入一个32位int，输出reversed的

如果溢出就输出0

注意范围是2\*\*31-1 到-2\*\*31，超过这个范围输出0

然后注意正负号

Python直接转换成字符串然后：：-1就可以

A and 1 or 2  写法

### 8. **String to Integer (atoi)**

Implement atoi which converts a string to an integer.

1. 首先需要丢弃字符串前面的空格；

2. 然后可能有正负号（注意只取一个，如果有多个正负号，那么说这个字符串是无法转换的，返回0。比如测试用例里就有个“+-2”）；

3. 字符串可以包含0~9以外的字符，如果遇到非数字字符，那么只取该字符之前的部分，如“-00123a66”返回为“-123”；

4. 如果超出int的范围，返回边界值（2147483647或-2147483648）。

综上，要求还是有点怪的，不看要求是很难写对的，看了也不一定理解的对。

5. 0为数字中的最高位

### **9. Palindrome Number**

判断一个数字是不是palindrome，但是不能把它转换成字符串

第一要注意的是这个数字可以是负数。 如果是负的直接输出false

然后做一个新的数字。每次把原来数字的末位取出，把之前的\*10 加上新数字。 然后原来数//10

比较新数字和老数字

### 10. **Regular Expression Matching**

输入字符串和pattern，pattern里面可以包括a-z和. \* 判断字符串符不符合这个pattern

This Solution use 2D DP. beat 90% solutions, very simple.

Here are some conditions to figure out, then the logic can be very straightforward.

1, If p.charAt(j) == s.charAt(i) : dp[i][j] = dp[i-1][j-1];  
2, If p.charAt(j) == '.' : dp[i][j] = dp[i-1][j-1];  
3, If p.charAt(j) == '\*':   
 here are two sub conditions:  
 1 if p.charAt(j-1) != s.charAt(i) : dp[i][j] = dp[i][j-2] //in this case, a\* only counts as empty  
 2 if p.charAt(i-1) == s.charAt(i) or p.charAt(i-1) == '.':  
 dp[i][j] = dp[i-1][j] //in this case, a\* counts as multiple a   
 or dp[i][j] = dp[i][j-1] // in this case, a\* counts as single a  
 or dp[i][j] = dp[i][j-2] // in this case, a\* counts as empty

还要注意dp[0][n] and dp[n][0] 的时候，dp[0][0] 肯定要是true.

### **11. Container With Most Water**

一组数字求这些数字高度的挡板之间最多存水多少。注意这里只能是两个之间。不用考虑中间的。

从两边开始往中间搞，也就是所谓的two pointer

因为高度只和矮的有关，所以每次只移动矮的pointer。每次计算新的area，取maxarea，直到相交

### **12. Integer to Roman**

根据整数输入相应的罗马数字

把1，10，100，1000位的各种1-9的pattern写出来。然后根据每一位输入的数字，从左到右连起来就可以了

### **13. Roman to Integer**

首先要做一个roman每个字母对应多少数字的dict

然后从左往右，如果比右边的小，就减去。如果比右边的大，就加上。最后不要忘记加上最后一位字母代表的数字。

### **14. Longest Common Prefix**

给一组字符串，输出最长common prefix

这题要用python的

enumerate(zip(\*strs))

然后一旦有不同的，就输出前面的就可以了。

不过还要考虑strs是空的情况

### **15. 3Sum**

输入一个数组，输出所有和为0的triplets。 输出的必须是triplets必须是unique的

这题很复杂容易错。 一种方法用dict，类似2sum，另一种方法用双针，O（n^2）

首先要sort

然后外围一个指针从左往右，另外两个指针从i+1到len-1, 比较如果和大了就k-=1，和小了就j+=1

最后要注意去重。去重的话，i可以等于j，但是这一次的i和下一次的i不可以相等。所以要for i=1:n, if si=si-1, continue

对j， while sj+1==sj, j+=1 . 对k，while sk-1=sk, k-=1

### **16. 3Sum Closest**

输入一个数组，和一个target，输出和最接近target的triplets

方法和3sum 相同，也是双针。 不过记录所有的最小差值

### **17. Letter Combinations of a Phone Number**

Dfs

首先要build dict，再搞一个全局变量装结果

然后dfs 的程序里面， 如果长度已经够长，就append结果到全局变量并且return

不够长的话，就从dict里面取出那个数字对应的list，然后把list里面每一个连接到目前的string后面，对新的string进行dfs并且index后移一位

最后输出那个装着结果的全局变量

### **18. 4Sum**

按道理和3SUM也是一样的。但是需要再仔细看.

先sort。 然后i从0到n-1， j从i+1 到n-2, k,m 一前一后用two pointer。 关键是不能重复，所以i,j 在for 循环中都是如果和前面一样就continue。 k，m需要在while里面写如果和之前一样就继续while。 此外i可以和j一样，只是i不能和之前的i一样。

### **19. Remove Nth Node From End of List**

快慢指针。注意要设一个dummy node prehead

### **20. Valid Parentheses**

可以是大小中括号。要求valide

这题用stack来做。如果是左括号就push进去，如果是右括号就看最后面的是不是对应的左括号，如果是就pull出来，不是就false

### **21. Merge Two Sorted Lists**

两个指针，比大小。prehead连接。 小的就连上后移。如果一个指针空了，就把结尾连上另一个结束

### **22. Generate Parentheses**

用dfs递归。注意分清楚长度等于的时候add to resolution。 左括号数量少于n可以加左括号，左括号数量超过一半可以加右括号。最后调用dfs函数，从“” 开始即可。

### **23. Merge k Sorted Lists**

有k个sorted linkedlist。 需要merge 他们

这题用priority queue

Python 里面首先要 from Queue import PriorityQueue

Python 里面的priorityqueue如果放入元祖的话，那么默认的就按照tuple的第一个元素大小来比较。所以可以放进(l.val,l)

先把所有k个list的第一个放进pq。 然后从里面取出最小的连到新的list里面。同时把最小的那个的next 放进pq。如果next是null就不放，直到整个pq全部清空。

### **24. Swap Nodes in Pairs**

小心即可。

### **25. Reverse Nodes in k-Group**

这题要一个一个的推。 1,2 reverse, 然后3和2，1 reverse。 然后4和3，2，1 reverse

细心

### **26. Remove Duplicates from Sorted Array**

双针法

不要忘记如果Array是空的情况

### **27. Remove Element**

给一个array和一个val，inplace去掉该val对应的所有的instance，输出新array的长度

这里面注意输出的长度是i。看你的代码，如果每次是修改i 然后i+1，那么显然最后输出的i已经是长度了，不需要i+1

### **28. Implement strStr()**

这题就是搜索substring，输出index

Python里面暴力解法就行了

注意needle为空或者haystack为空的情况

### **29. Divide Two Integers**

反复相加法，算了2的N次方，存在list里面。然后从大向小减，依次去掉那个数，然后记数的count加上2\*\*i i是在list里面的index。易错点在于最后加的时候，sumDict[i]<=dividend， 里面是小于等于。因为要考虑2的整数次方的情况。 最后还要考虑如果是负数的情况。 还要比较溢出的情况。

### **30. Substring with Concatenation of All Words**

给一组words，都是一个长度。再给一个string，查找所有的是那组words里面concatenated的substring 的index

因为都是固定长度，所以每次0，3，6，9； 1，4，7。。。 ； 2，5，8.。。

一定要用hashmap记下每个index对应的word，这样不会重复比较

然后在按照min window的方法来做

这题没有手工写过，不清楚难度

### **31. Next Permutation**

输入一个数字， 输出重新排列的数字里next 大的数字。 如果不available的话，输出最小的

这题要想大，必须大的数字在左边，小的在右边。如果从右到左一直是变大的就没有办法调整。如果一旦有一个数字小于右边的，那么就把这个数字换成右边里面next大的数字，然后对剩下的数字从小到大排列。

sorted(nums)并没有在原来的地方修改所以不对。必须要用nums=sorted(nums)

nums[i+1:][::-1]才是倒序，而不是nums[i+1::-1]

### **32. Longest Valid Parentheses**

输入一个由（）组成的string求最长的valid 括号的string的长度

设立一个stack， 同时记录下idx，leftmost, stack, max

刚开始leftmost = -1

遇到左括号就push 进stack。 遇到反括号就pop出stack。stack里面存的是每一次的index，而不是括号。因为这里只有一种括号，所以不需要考虑括号不匹配的情况。每次pop出来的时候，更新max， 当前的index减去pop出来以后在stack里面最上面的值，然后和之前的max相比取最大值。如果stack变成空，就用idx减去leftmost来比较。 如果stack里面已经是空并且出现一个反括号，那么就更新leftmost

最后return max

### **33. Search in Rotated Sorted Array**

二分查找法，小心mid=(left+right)//2. 因为如果/2，会出小数。python里面整除是//2

先试low<=mid, 此情况下nums[low] <= target <= nums[mid]:

high = mid - 1

再试mid<=high,此情况下if nums[mid] <= target <= nums[high]:

low = mid + 1

else:

high = mid - 1

注意因为mid可以等于low，所以都要用《= 和大于等于

此题非常复杂有四种情况，转折点在左边，右边 \* （大于mid，小于mid） 四种情况要分析清楚。

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

两种方法。一种是用binary search 分别搜索target-.5 and target+0.5 然后就知道起点终点了

另一种是搜索left most and right most. 这种情况就要考虑移动指针的几种情况。 比如找left的时候，如果target==mid, j=mid. 注意这里不能用mid-1 因为mid可能是最左的。 找右边的时候，if target==mid, i=mid

### **35. Search Insert Position**

用binary search

最后比较插入位置时候，

**if nums[l] == target or nums[l] > target:**

**return l**

**Else:**

**return l+1**

### **36. Valid Sudoku**

用引用和pass in ij的方法做。细心

### **37. Sudoku Solver**

Backtracking

每次最后一个heler跑完1-9都不行以后，一定要记得return false

for n in range(1,10):

board[i][j]=str(n)

if self.checkRow(board,i) and self.checkCol(board,j) and self.checkBox(board,i,j):

if self.helper(board):

return True

board[i][j]='.'

return False

### **38. Count and Say**

Easy question. pass

### **39. Combination Sum**

给1 set数字。 然后可以从里面反复取数字。要求取出的数字的和为target。求出所有组合

用backtracking 做。注意给全局变量加入结果的时候，用list.copy(), 否则再backtracking那个list会同时改变result 里面的结果

另外，需要sort list。然后从小到大。 因为可以反复，所以每次送进backtracking的都是candidates[i:], 也就是包括当前的数字。但是不能包括前面的数字，因为那样和前面的结果会duplicate

### **40. Combination Sum II**

此题给出的不是set，而是list。同时每个数字只能用一次，不能反复取。

也是用backtracking 做。为了不重复取，每次用candidates[i+1:]

为了不出现重复的结果，在遍历candidates的时候，如果和之前的一样就要跳过。

if i>0 and c==candidates[i-1]:

continue

### **41. First Missing Positive**

首先要把所有的负数和都变成maxint。第一遍O（n）

然后过一遍，把所有存在的对应的都变成负数。最后看哪一位不是负的。

第二遍的易错点是还要检查那个数字不要超过list的长度。还有就是整数是1开始，坐标是0开始，所以要转换。

### **42. Trapping Rain Water**

给出n个非负int。 然后往里面倒水，总共最多可以保留多少水

这题首先需要考虑如何存水。水的话是左边的最高，右边的最高，然后二者的最低，减去Num[i]

所以先从左往右，算出所有i的左边最高。再从右往左，算出所有i的右边最高。最后从0到n，用min（left,right)- nums[i] 即可

### **43. Multiply Strings**

第一个字符串的第i位乘以第二个字符串的第j位一定是结果的第i+j位，如果i+j已经有值，直接加上去就OK，并用temp保存进位，

最后记得将结果反转，去掉前置0。这样的算法的复杂度是O（n2）

没有自己做过

### **44. Wildcard Matching**

注意这里\*可以表示任何字符串。不需要同一种字符

第二注意f要从0开始，所以长度是len+1， 并且s[i]，p[j] 对应的都是f[i+1][j+1]

### **45. Jump Game II**

问题：给定一个数组，每个元素表示你站在该位置可以跳的最大距离。假设你站在第一个元素，求你可以跳到最后元素的最小跳跃次数。

* 假设 [ start, end ] 表示第 i 次跳跃才能到达的区间，nextEnd 代表在该区间中起跳的下一个最远元素，那么，[ end+1, nextEnd ] 表示第 i+1 次跳才能去到的范围区间。
* 初始化 [start , end] 为 [0,0]，重复执行上面操作，直到 [start, end] 覆盖到终点元素。由于 [start, end] 表示第 i 次跳跃才能到达的区间，所以 i 便是最小的跳跃次数。

在代码实现中， start 变量没有影响到程序的执行，加进去只是为了方便理解。

一个pre记录之前的最大值，一个cur记录走到pre这段可以达到的最大值

不要用for循环，而是每次都让i走到pre停止，记录这段最大的为cur。

最后判断退出条件的时候，一定要cur小于len-1. 而不是小于len。 因为如果cur等于len-1那么就是达到中点了。

### **46. Permutations**

输入一set的数字,给出所有的unique的排列

回溯递归实现

逐一向中间集添加元素，并将当中间集元素个数等于 nums 长度的时候，将中间集添加到结果集中，并终止该层递归

### **47. Permutations II**

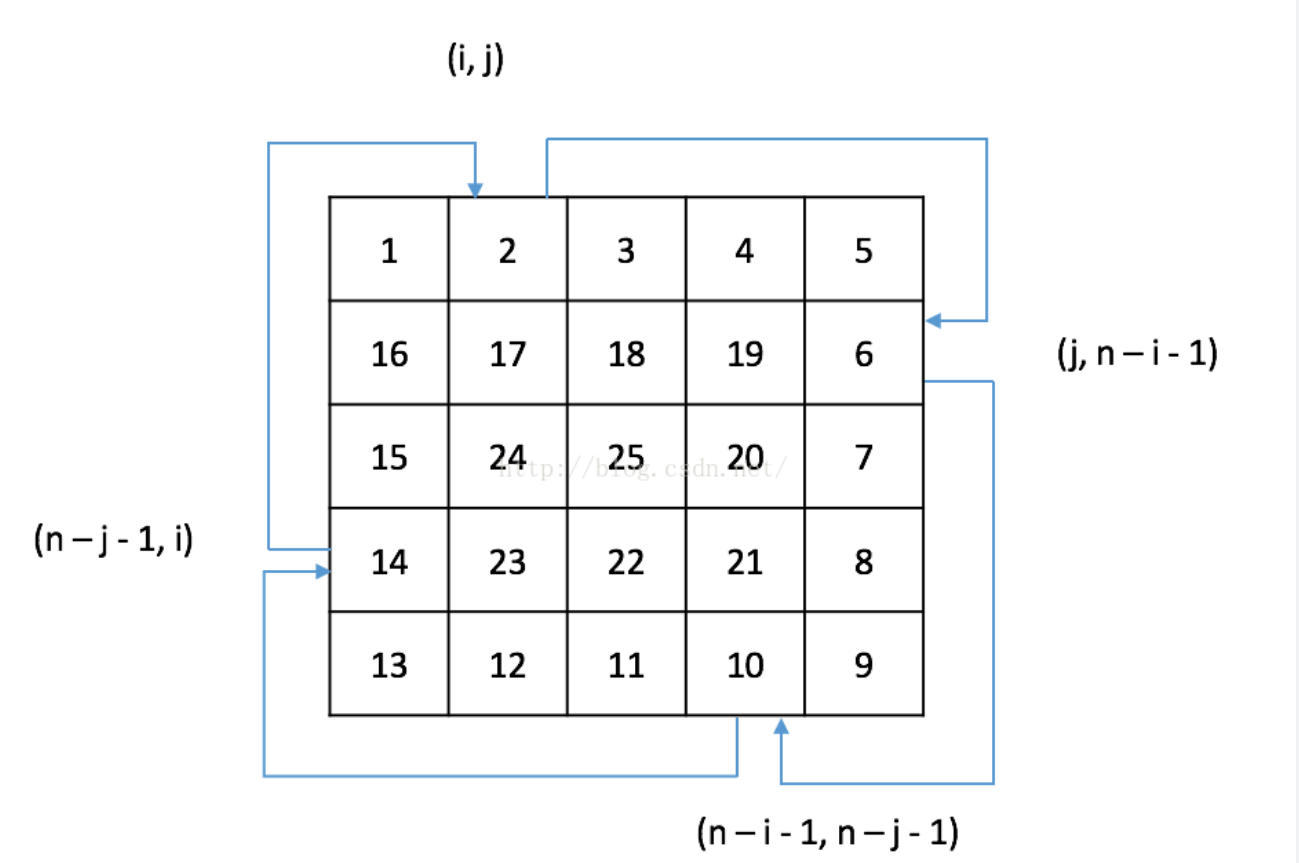
输入一组可以有重复的数字,输出所有unique的排列

跟 Permutations的解法一样，就是要考虑“去重”。先对数组进行排序，这样在DFS的时候，可以先判断前面的一个数是否和自己相等，相等的时候则前面的数必须使用了，自己才能使用，这样就不会产生重复的排列了。To handle duplication, just avoid inserting a number before any of its duplicates.

### **48. Rotate Image**

n\*n image. 必须要inplace旋转

矩阵的顺时针90°旋转，关键是找到坐标关系。本题中 n 的值提前减1了，所以在下面的运算中就没有减1了。



### **49. Group Anagrams**

给一组string，把所有的anagrams都分组

这一题就是要build一个hash，存每个字母的数量。然后每分析下一string，看符合哪个，就塞进去。不符合，就加在后面。 简单的办法就是存每个字母的数量。每个字母的数量之间用其他符号连接。比如#12#3#5#8 。。。

### **50. Pow(x, n)**

计算x的n次方。这题首先要注意溢出的问题，不能超过2\*\*31-1 and -2\*\*31

其次要注意O（n）会超时，要用logn

还要注意n可以是负数

递归实现.因为递归无法记录中间值,所以要用中间变量存n/2 如果n为0 or 1 直接给出答案. 否则如果n为奇数,递归用n/2, n/2, x 如果n为偶数,递归用n/2, n/2

如果n是负数,就转换为1/x

### **51. N-Queens**

1. 看清楚返回的格式是list of string。
2. 返回copy值，而不是只copy address
3. 不LTE的关键是，每次helper里面，i只检查前一个i+1, 不要做循环！因为如果这一行是空的，那么肯定不可能达到N queen！ 这样可以剪很多枝!!
4. 此外，因为每次i都增加且只增加一个，所以就不需要再checkrow了。可以再剪一些枝。

### **52. N-Queens II**

和上题一样，但是为了记录总数，设一个全局变量即可。 注意，python里面全局变量直接在任何函数内self.res=0即可，不需要在init里面定义

### **53. Maximum Subarray**

一组int,求连续的subarray里面最大的和.

O(n)就是一维DP.

假设A(0, i)区间存在k，使得[k, i]区间是以i结尾区间的最大值， 定义为Max[i], 在这里，当求取Max[i+1]时，

Max[i+1] = Max[i] + A[i+1], if (Max[i] + A[i+1] >0)

= 0, if(Max[i]+A[i+1] <0)，如果和小于零，A[i+1]必为负数，没必要保留，舍弃掉

然后从左往右扫描，求取Max数字的最大值即为所求。

### **54. Spiral Matrix**

need to use pointer to pass matrix. np is not available

与Rotate Image那题类似，一层一层处理。但这题有两个注意点：

1. m和n可以不同，所以对于第i层来说，最后一行为lastRow = m-1-i，而最后一列为lastCol = n-1-i。所以层数由min(m,n)决定。

2. 当min(m,n)为奇数时，最后一层为一行或一列，需要特殊处理。

### **55. Jump Game**

给出一个数组。从第一个开始，每个数字代表可以往后跳的位数。 这题用maxpoint来记录最大值，不要用另外一个数字存1，0来记录可否到达会LTE

每次更新maxpoint，知道现有的index大于maxpoint就停。

易错点是最后判断maxp >= len(nums)-1. 一个是要大于等于而不是大于，第二是要len -1 .

### **56. Merge Intervals**

Given a collection of intervals, merge all overlapping intervals.  
  
sorted, intervals is empty array,prev end> next end

### **57. Insert Interval**

将一个interval插入到已经排序好了的intervals中

因为已经排序好了，所以只需要一个个遍历比较看是否需要合并，然后将其添加到结果数组中即可。

遍历时，设当前interval为it，会发生以下三种情况：

1. newInterval在it前面，表现为newInterval.end < it.start，此时先添加newInterval，后添加it，因为newInterval已经添加进去了，所以后面的就无需比较了
2. newInterval在it后面，表现为newInterval.start > it.end，此时只添加it到结果数组中即可，让newInterval再与后面的去比较
3. 两者有重叠部分，需要合并，将it合并到newInterval中

### **58. Length of Last Word**

Too easy. skip

### **59. Spiral Matrix II**

Given a positive integer *n*, generate a square matrix filled with elements from 1 to *n*2 in spiral order.

II比I简单，一样的按层访问法。由于是正方形矩阵，当n为奇数时，最后只会剩下一个数字即matrix[n/2][n/2]，最后不要忘记补填上这个数字。

### **60. Permutation Sequence**

Given *n* and *k*, return the *k*th permutation sequence.

搞一个list

然后每个数字都是（n-1）！

然后把k除以那个数字的结果取出来。再剩下的，继续按照n-2）！来算，一直下去

这题不确定，没有自己做过。

最高位可以取{1, 2, 3, 4}，而每个数重复3! = 6次。所以第k=9个permutation的s[0]为{1, 2, 3, 4}中的第9/6+1 = 2个数字s[0] = 2。

而对于以2开头的6个数字而言，k = 9是其中的第k' = 9%(3!) = 3个。而剩下的数字{1, 3, 4}的重复周期为2! = 2次。所以s[1]为{1, 3, 4}中的第k'/(2!)+1 = 2个，即s[1] = 3。

对于以23开头的2个数字而言，k = 9是其中的第k'' = k'%(2!) = 1个。剩下的数字{1, 4}的重复周期为1! = 1次。所以s[2] = 1.

对于以231开头的一个数字而言，k = 9是其中的第k''' = k''/(1!)+1 = 1个。s[3] = 4

### **61. Rotate List**

把一个linkedlist，向右边rotate k 个nodes。

快慢指针，注意k%len==0的情况。还要先把k%len 以后再rotate。

### 62. Unique Paths

从左上角到右下角，每次可以向下或者向右。求一共多少种走法

这题注意初始化是用两个中括号

剩下就是i,j=i-1，j+i,j-1

来算

### 63. Unique paths 2

和上题目一样，但是有一些obstacle

这题就是像上面那样，两个中扩号初始化。如果有obstacle就定成0

但是要注意初始化的时候，考虑obstacle at 0，0

### **64. Minimum Path Sum**

输入一个二维数组

给出从左上到右下的最小的和。

grid[0,0] 不对，必须写成grid[0][0]. numpy array 才可以那么写

做一个新的二维数组。然后从左往右，每个节点记下最小的sum。 就是上面sum加val，左边sum加val，在取min。

### **65. Valid Number**

* If we see [0-9] we reset the number flags.
* We can only see . if we didn't see e or .
* We can only see e if we didn't see e but we did see a number. We reset number falg After E flag.
* We can only see + and - in the beginning and after an e
* any other character break the validation.

### **66. Plus One**

输入一个数组。左边是高位，右边是低位。 给这个数字加一。求新的数字的数组

**此题要注意： range(10,0,-1) 不会走到0，只到1. 要走0 需要写range(10,-1,-1)**

**然后就是【1】+【】就可以连接两个数组。**

### **67. Add Binary**

**easy。输入两个string，都是由1和0组成。输出两个按照2进制相加的和。**

**从右边到左边，再搞一个carry 变量。**

### 68. Text Justification

改写了一下，把逻辑理顺了一点点。下面是改写的思路：

1. 一开始仍然是初始化， 初始化结果集合res， 一个用来处理当前行的StringBuider sb，以及一个lineWordCount = 0
2. 接下来遍历数组，先计算出当前单词word以及它的长度， 接下来我们主要分三种情况来考虑
   1. wordLen + sb.length() == maxWidth，这时我们找到一个结果，在sb中append当前单词，然后把sb.toString()加入到结果集合中，重置sb和lineWordCount
   2. wordLen + sb.length() < maxWidth，这时候说明我们仍然可以在当前行里塞单词，我们先append(word)，再append一个空格" "， 更新lineWordCount++
   3. 否则说明当前wordLen + sb.length() > maxWidth，这时候我们必须对当前行sb进行处理，然后才可以继续后面的操作。对这种情况我们又可以分为两种子情况:
      1. 当lineWordCount == 1，这时候这一行只有一个单词，我们只需要在当前行sb的后面补足空格，直到补充到maxWidth为止
      2. 否则lineWordCount > 1，这时这一行有多个单词，我们执行一个分拆出来的函数distributeSpaces来把单词中的空格平均分配
      3. 经过上两步求出了当前行sb之后，我们可以把sb加入到结果集中，重置sb和lineWordCount，因为我们并没有处理当前单词，所以要减少index i，用i--来重新处理当前单词
3. 主循环结束之后我们处理最后一行，根据题目意思，在sb中补空格直到maxWidth，然后将其加入到结果集中。
4. 最后返回结果

关于辅助函数distributeSpaces，主要逻辑分为以下几个步骤：

1. 当一行有多个单词的时候，我们需要把多余的空格均匀分配到每个已有的空格slot里，假如不够分，则尽量放到左边的slot里，这就有了我们的函数
2. 我们的函数包括了上面主逻辑第3步的1和2小步，分为lineWordCount == 1时和lineWordCount > 1两种情况考虑
   1. 当lineWordCount == 1，这时候这一行只有一个单词，我们只需要在当前行sb的后面补足空格，直到补充到maxWidth为止
   2. 否则lineWordCount > 1，这时这一行有多个单词，我们执行一个分拆出来的函数distributeSpaces来把单词中的空格平均分配
      1. 这里我们首先计算出一共要分配多少个空格，这里newSpaceTotal = maxWidth - sb.length() + 1， 因为之前我们在每个单词后面都加入了一个空格，所以计算时要把这个考虑进去
      2. 我们有多少个slot可以插入， 因为每两个单词中间就算一个slot，所以slots = lineWordCount - 1
      3. 接下来我们先把sb转换为String s，再重置sb.setLength(0)， 对于s， 从0到 s.legnth() - 2进行遍历 （不遍历最后一个空格）。
      4. 假如当前的字符为c，我们在sb中append(c)， 假如c为空格的话，我们需要进行额外的判断:
         1. 当newSpaceTotal <= 0时， 用continue跳过
         2. 设置一个int spaceToAdd = 0，代表当前要加入的空格数
            1. 当newSpaceTotal > slots时， 我们计算这个slot可以添加多少个空格

spaceToAdd = (int) Math.ceil((double) newSpaceTotal / slots)， 这里要向上取整

newSpaceTotal -= spaceToAdd

slot--

* + - * 1. 否则slots数目大于等于newSpaceTotal， 我们最多每个slot可以分配一个space，所以

spaceToAdd = 1

newSpaceTotal--

* + - * 1. 接下来根据spaceToAdd的数目，我们在sb里面append(" ")。

### 69. Sqrt(x)

**输入x保证是非负的。返回平方根的int**

**这题要用二分查找法**

**取中点，如果中点的平方大于x，那么right= mid-1. Else left=mid+1**

### 70. Climbing Stairs

**Pass**

### 71. Simplify Path

**输入一个unix style的文件的绝对路径，简化它**

**要考虑/.. And /. 都要简化成/**

**然后用stack做**

### 72. Edit Distance

**输入两个word，可以插入，替换，delete一个字母为一个操作。 计算需要多少步才能把一个词变成另外一个**

**1.如果str1的第i个，也就是str1[i-1]和str2的第j个也就是str2[j-1]相等的话，那么**

**dis[i][j] = dis[i-1][j-1]**

**2.如果str[i-1] != str2[j-1]**

**2.1 通过替换操作把str[i-1]替换成str2[j-1]，那么**

**dis[i][j] = dis[i-1][j-1] + 1;**

**2.2 通过插入操作在str1后面插入str2[j-1], 那么就相当于计算**

**dis[i][j] = dis[i][j-1] + 1;**

**2.3 通过插入操作在str2后面插入str1[i-1],那么就是**

**dis[i][j] = dis[i-1][j] + 1;**

**在上述三个中选一个最小的。迭代更新。**

**如上我们就看到了每一个子问题的求解方式，那么对于答案：**

**1、首先构造边界的值，如i=0，j=0的时候，编辑距离就是和已匹配的长度一样，因为全都要做插入**

**2、按照顺序遍历**

**3、输出最终位置的答案。**

### 73. Set Matrix Zeroes

**给出一个matrix，如果里面一个元素是0，那么把它的row和col都算成0. 只能inplace计算**

**把结果都存在row 1 and col1. 但是为了不改变结果，改用负号来表示。所以如果这一行有0， 就把row1 and col1变成负的。**

### 74. Search a 2D Matrix

**在一个2d matrix里面搜索一个数字，这个2d matrix的规律是每一行从小到大，并且每一行第一个都比前一行最后一个大**

**这一题用二分法做。先对每一行的首位二分查找，锁定那一行。 然后对那行里面进行二分查找，找到那个数字。**

### 75. Sort Colors

**一个数组里面有，1，2三种数字。 用1pass algorithm，把他们sort**

**用三个指针，一个0->n, 一个0-》 n， 一个n->0. 其中两个如果替换成0或者2了就移动，还有一个从左往右移动寻找0或者2.如果是0或者2 就和相应的左边或者右边的交换。 直到中间的和右边的相碰同时要cosider k==j的case**

### 76. Minimum Window Substring

输入sring S 和一个短string。 求长string里面包括短string所有字符的最短substring

时间复杂度要求O（n）

用i往后走，一直到符合情况。然后j在左边开始走，一直到不符合情况。每次i符合情况以后，不要存最小string，要先走j直到不符合情况以后，再存前一个j的，才是最小情况。

### 77. Combinations

**给出n,k 输出所有的combination of k 个数从1-n中抽出来**

**Backtracking。 终止条件是数量达到k**

### 78. Subsets

**给出一set 数字，输出所有的subset**

**Backtracking. 为了避免重复，需要定义function helper(self,nums,tmp,res,pos)。 其中pos为下一个要开始的插入的字符在原来的nums的位置。这样才不会插入原来就有的数字。**

### 79. Word Search

**给一个2D的board，查某个word是否在里面。要求上下左右为相邻的字母**

**Backtracking 的方法来解决这道题目。我们需要先从board中找到第一个字符出现的位置，然后从这个位置开始往上，下，左，右开始寻找；这里我们要借助一个dfs(x,y, index, board, word)的函数来寻找； 当index==word.length的时候就找到了对应的字符，返回true; 注意边界条件的check，x,y要保证在board内，或者当board[x][y]!=word.charAt(index)的时候都返回false。**

**复杂度： Time复杂度: 遍历整个m \* n 的board的时间复杂度为m \* n,对于每个点都会往上下左右来遍历寻找， k为word长度，大概要遍历4^k次，所以总的时间复杂度大概在m*n*4^k.**

**Space： 由于word长度为k，recursive space大概在O(k).**

### 80. Remove Duplicates from Sorted Array II

**一个sorted array，每个元素最多出现两次，否则去掉多余的。只能用O(1) extra space**

**注意是sorted，用2 pointer 返回时候返回 i 而不是i +1**

### 81. Search in Rotated Sorted Array II

**一个排序的数组，但是在某个点上rotate了。这里 不同的是，里面会有重复的数字。**

**本题采用二分法实现，但是比较挠头的是边界问题，而且元素有重复，相比纯粹递增的数组难度要大得多，要解决这个问题，首先要对所有可能情况进行分类，然后对每种可能的类别进行相应的处理，**

**暂且不考虑nums[mid] = nums[left]的情况，本题大致可以简化为上图两种情况，可能的情况划分出来，那么解决本题就比较容易了：**

**当 nums[mid] = nums[left] 时，这时由于很难判断 target 会落在哪，那么只能采取 left++**

**当 nums[mid] > nums[left] 时，这时可以分为两种情况，判断左半部比较简单**

**当 nums[mid] < nums[left] 时，这时可以分为两种情况，判断右半部比较简单**

### 82. Remove Duplicates from Sorted List II

**输入一个已排序linkedlist，去掉所有重复的。只留下distinct的。就是说如果这个数字出现了两次就要删掉，出现一次的才留下。这题要很小心。 用三个指针。**

### 83. Remove Duplicates from Sorted List

**输入一个以排序的linkedlist，去掉所有重复的，每个数字只留下一个。 这个用两个指针和pre就好了。**

### 84. Largest Rectangle in Histogram

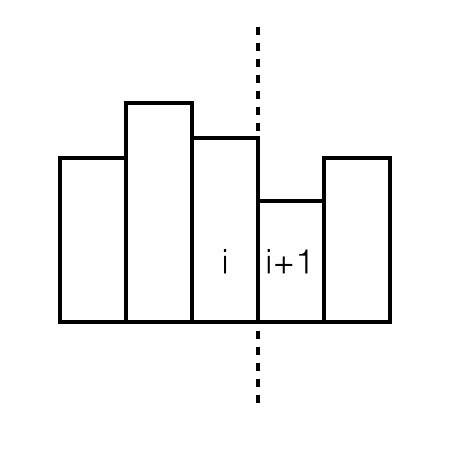
方法1： 每一个rectangle都有个以他为最高的最大的rectangle。所以从1-n，依次往左到最大，往右到最大。 最后求出最大的面积。 O(n^2)

方法2： 方法II： 辅助栈法（网上很多人采用的方法）

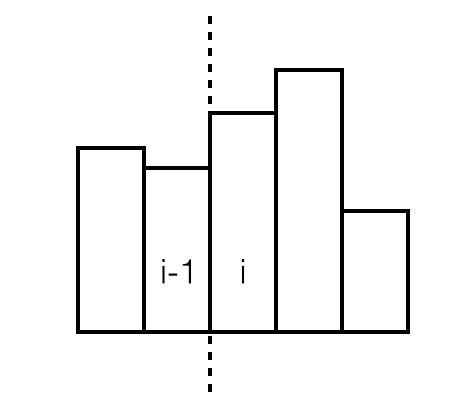
根本思想是：依次遍历所有矩形条，尝试计算以该矩形条为高度的矩形面积。但是在遍历的时候我们不知道后面还有什么样的矩形条怎么办？没关系，对于没法确定面积的矩形，压栈，留着以后处理，而对于那些已经可以确定计算出面积的矩形条，留着也没用，弹栈。

如果我们能知道一个矩形条向左向右最远能延伸多远，我们就能计算出以该矩形条为高的矩形面积了！我们怎么知道向左向右能延伸多远？观察下面几种情况：

情况1，第i个矩形比右边相邻的第i+1个矩形高，如下图所示。意味着，以height[i]为高的矩形的右边界就是第i个矩形，因为右边界不能更右了（废话），也不会在左边（向左只会让矩形面积减小）。所以在这种情况下，我们可以立即确定以第i个矩形的高度height[i]为高度的最大矩形面积的右边界。

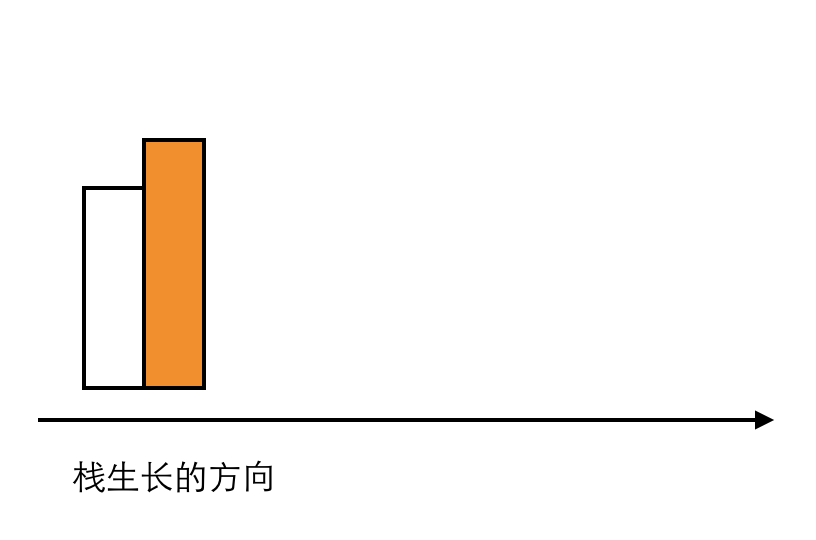


情况2，第i个矩形比左边相邻的第i-1个矩形高，如下图所示。意味着，以height[i]为高的矩形的左边界就是第i个矩形，因为左边界不能更左了（废话），也不会出现在右边（因为向右只会另矩形面积减小）。所以在这种情况下，我们立即就可以确定以第i个矩形的高度height[i]为高度的最大举行面积的左边界。

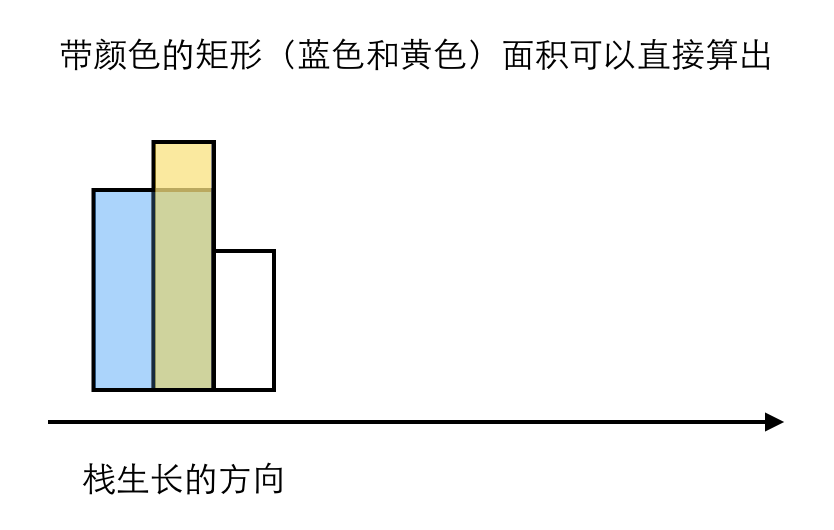


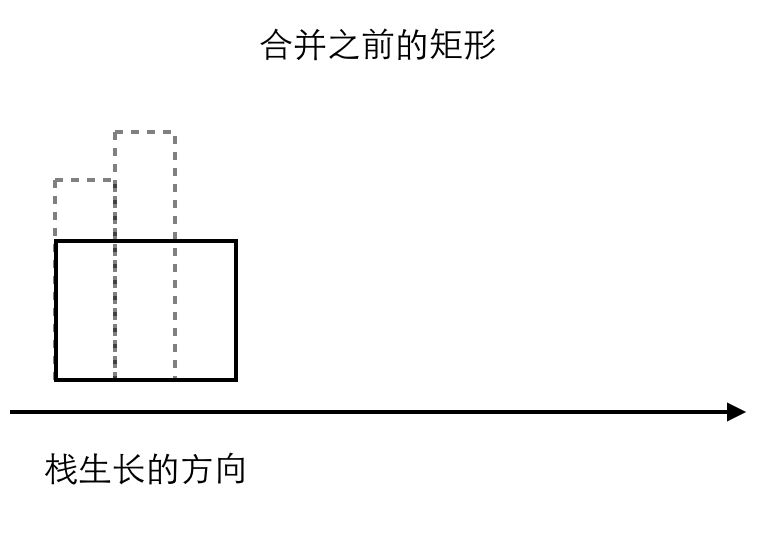
现在，我们依次遍历各个矩形条，遍历过的矩形条压入栈中保存，则不难发现下面的现象：

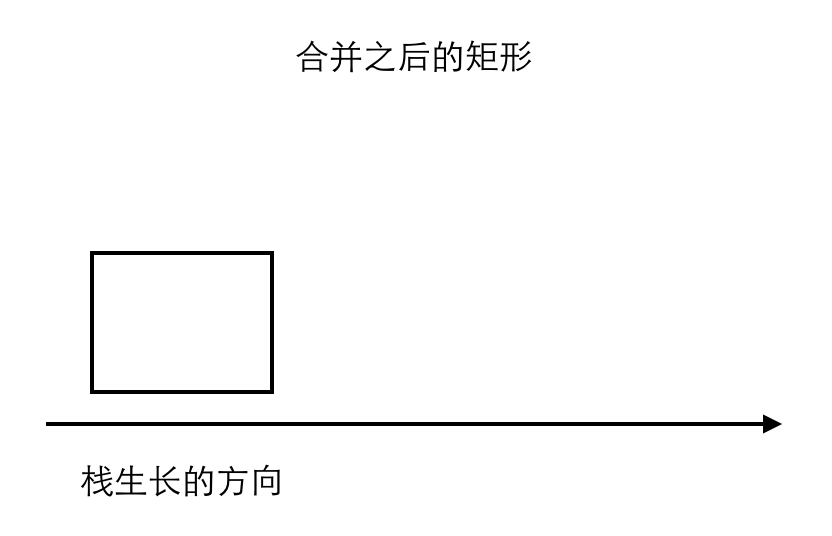
如果当前矩形条的高度高于栈顶的矩形条的高度，对应上面的情况2，我们可以立即得出，当前矩形一定是以它为高的矩形的左边界。那么现在我们还不能确定其右边界，所以除了入栈什么都不用做，如下图所示：



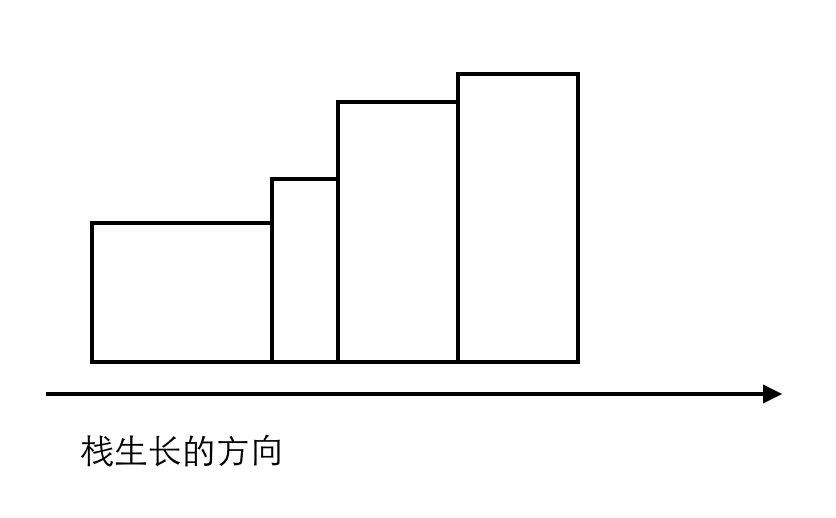
如果当前矩形条的高度小于或等于栈顶矩形条的高度，对应上面的情况1，我们可以立即得出：栈顶的矩形一定是以它为高的矩形的右边界所以，我们可以立刻得到以栈顶的矩形条高度为高度的最大矩形的面积（下图中带颜色的两个矩形）！既然我们算出了前一个条最大矩形的面积，那么也就没必要再留着它了。所以，可以放心把它删掉，或者说合并。



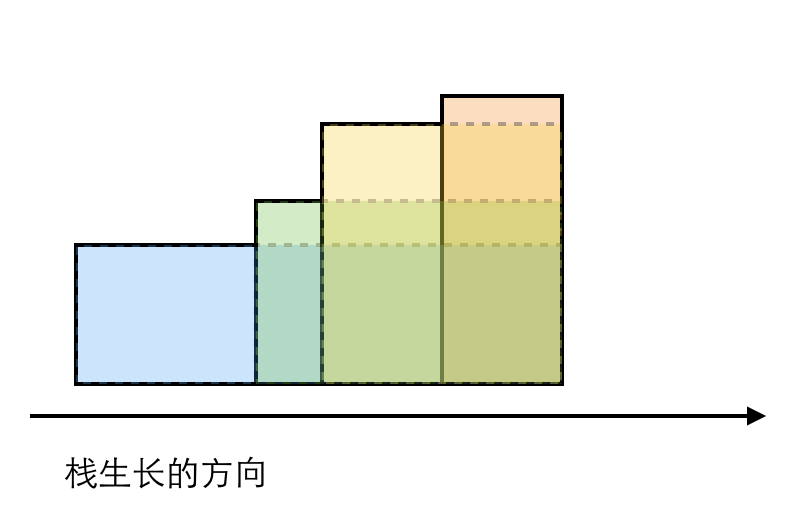




按照上述操作遍历完所有矩形条之后，栈中的矩形一定是下面这个样子。（栈里面所有的条肯定是按照高度依次递增的）



此时，对于任何一个矩形条，我们可以确定，它的右边界一定是整个栈里所保留的条形图的最右边界，而它的左边界一定是它这个条自己的左边界，所以，剩余的矩形条也可以立即算出其对应的最大矩形的面积。



其实，只需要在所有的矩形条最后添加一个高度为0的虚拟矩形条，可以省略上面的两小步。这个虚拟矩形条起收割作用。（见代码第2行）

显然时间复杂度是O(n)。

具体代码实现有两个技巧：

1. 我们只需要在辅助栈保存矩形的右边界坐标即可，不需要保存高度，因为可以通过右边界坐标得到（height[i] ），也不需要保存左边界坐标，因为上一个矩形的右边界坐标+1就是当前矩形的左边界。

2. 在直方图最后添加一个高度为0的虚拟矩形条，这样保证一次遍历之后栈里面的矩形都被正确处理过了，否则需要再重复一遍。

### 85. Maximal Rectangle

给定一个只含0和1的数组，求含1的最大矩形面积。

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

这样的题一般看来都是有O(n\*m)的解法的。

借助上一题 [Largest Rectangle in Histogram](http://www.cnblogs.com/higerzhang/p/4107268.html) 的解法。

我们现在把矩阵的每一行当做是上一题的问题，然后遍历所有的行数，取最大数就是解。

这里的每一行的高度怎么确定呢。

1.如果当前点为‘0’，那么高度就是0

2.如果当前点为‘1’，那么高度就是往上连续1的个数（包括自己）。

求高度是一个动态规划过程。用dp[i][j]来存某个位置的高度那么

对于第i行第j列的点：

if (matrix[i][j] == '1')

　　dp[i][j] = dp[i-1][j] + 1;

else

　　dp[i][j] = 0;

### 86. Partition List

**给出一个list和x，让所有小于x的node排在所有大于x的node后面。但是其他的原有顺序要保持**

**这题用两个链表。 遇到比x小的接到list 1，比x大的接到list 2. 最后把两个list连起来。**

**最后一定要记住把pr.next=None, 否则溢出**

### 87. Scramble String

输入两个string，判断一个string是不是另一个的scramble string

Scramble 意思是把一个string切成两半，然后左右对调。对切成的两半也可以继续切然后对调。

这题hard很难。自己思考一下左一座吧

还是recursive比较好理解一些，三维dp以后再说了。下面分析一下recursive的几个点：

1. 首先判断边界
2. 其次，当s1等于s2的时候，我们判断可以返回如true
3. 否则我们对排序后的 s1和s2进行一个比较，假如不等，则我们舍去
4. 否则我们进入遍历的循环体，注意starting index是从1开始
   1. 我们设置s11, s12, s21和s22,然后递归判断(s11, s21)以及(s12和s22)这两个pair，假如同时满足scramble，则我们可以返回true
   2. 否则，我们尝试交换过一次的结果，即重设s21和s22从尾部开始split。然后比较新的(s11, s22)以及(s12和s21)这两个pair，假如同时满足条件则返回true
5. 否则返回false
6. 复杂度的来说 ，不考虑substring的复杂度话， recursive depth大约是n，branching factor是4，所以时间复杂度是大约是O(n4)， 空间复杂度也是O(n4)

### 88. Merge Sorted Array

**给两个sorted数组num1 num2把num2排到num1 里面。不能用第三个数组。**

**这一题需要从num1 的后面开始放。因为只能在num 1 inplace 搞，所以设定i=m-1,j=n-1,从i+j=1开始**

### 89. Gray Code

Gray code是一串数字，这些数字的二进制表示每两个之间都只差1个bit

输入n是数字的二进制的位数，输出该位数的一连串的gray code。必须要从0开始

格雷码是很经典的问题，规则其实很简单，在二进制形式下，任何响铃的两个值的二进制表示形式只有一位是不同的，我们可以找找规律。

一位就是简单的：0,1

两位是：00,01,11,10

三位是：000,001,011,010,110,111,101,100

发现什么规律没有？我们把一位的两个数，前面加上0，就是二位的头两个数，前面加上1再反序，就是二位的后两个数。把二位的前面加上0，就是三位的头四个数，把二位的前面加上1再反过来，就是三位的后四个数。

也就是说，对于每多一位的格雷码，前面一半的第一位都是0，后面一半的第一位都是1，其余的位，前后两半正好是中间对称的，前面一半就是少一位的格雷码序列，后面一半时把其反序。

知道这个规律就好做了，我们可以递归来做，每次取n-1位的格雷码来做上述操作，对于一位的格雷码，直接赋值是0,1就可以了。

不过题目要求返回的是十进制数，而不是字符串，所以我们最好直接操作十进制数，这里前面加0其实就不用做什么，前面加1的话可以将1左移n-1位然后与各个数字相加即可。

注意题目说的n是非负数，所以要考虑n=0的情况，测试用例的n=0时返回的是0。

### 90. Subsets II

**非高频medium**

**输入一组数字，但可以有重复，输出所有subsets**

这道子集合之二是之前那道 [Subsets 子集合](http://www.cnblogs.com/grandyang/p/4309345.html) 的延伸，这次输入数组允许有重复项，其他条件都不变，只需要在之前那道题解法的基础上稍加改动便可以做出来，我们先来看非递归解法，拿题目中的例子[1 2 2]来分析，根据之前 [Subsets 子集合](http://www.cnblogs.com/grandyang/p/4309345.html) 里的分析可知，当处理到第一个2时，此时的子集合为[], [1], [2], [1, 2]，而这时再处理第二个2时，如果在[]和[1]后直接加2会产生重复，所以只能在上一个循环生成的后两个子集合后面加2，发现了这一点，题目就可以做了，我们用last来记录上一个处理的数字，然后判定当前的数字和上面的是否相同，若不同，则循环还是从0到当前子集的个数，若相同，则新子集个数减去之前循环时子集的个数当做起点来循环，这样就不会产生重复了

### 91. Decode Ways

这道题要求解码方法，跟之前那道 [Climbing Stairs 爬梯子问题](http://www.cnblogs.com/grandyang/p/4079165.html) 非常的相似，但是还有一些其他的限制条件，比如说一位数时不能为0，两位数不能大于26，其十位上的数也不能为0，出去这些限制条件，根爬梯子基本没啥区别，也勉强算特殊的斐波那契数列，当然需要用动态规划Dynamci Programming来解。建立一位dp数组，长度比输入数组长多多2，全部初始化为1，因为斐波那契数列的前两项也为1，然后从第三个数开始更新，对应数组的第一个数。对每个数组首先判断其是否为0，若是将改为dp赋0，若不是，赋上一个dp值，此时相当如加上了dp[i - 1], 然后看数组前一位是否存在，如果存在且满足前一位不是0，且和当前为一起组成的两位数不大于26，则当前dp值加上dp[i - 2], 至此可以看出来跟斐波那契数组的递推式一样，DP array size mistakes, consider 0

### 92. Reverse Linked List II

One pass把一个linkedlist reverse

反转整个链表的变种，指定了起点和终点。由于m=1时会变动头节点，所以加入一个dummy头节点  
  
1. 找到原链表中第m-1个节点start：反转后的部分将接回改节点后。  
从dummy开始移动m-1步  
  
D->1->2->3->4->5->NULL  
 |  
 st  
  
2. 将从p = start->next开始，长度为L = n-m+1的部分链表反转。  
 \_\_\_\_\_\_\_\_\_\_  
 | |  
 | V  
D->1->2<-3<-4 5->NULL   
 | | |   
 st p h0   
  
3. 最后接回  
  
 \_\_\_\_\_\_\_\_\_\_  
 | |  
 | V  
D->1 2<-3<-4 5->NULL   
 |\_\_\_\_\_\_\_\_|

### 93. Restore IP Addresses

题目明确指出输入字符串只含有数字，所以当某段是三位时，我们要判断其是否越界（>255)，还有一点很重要的是，当只有一位时，0可以成某一段，如果有两位或三位时，像 00， 01， 001， 011， 000等都是不合法的，所以我们还是需要有一个判定函数来判断某个字符串是否合法。这道题其实也可以看做是字符串的分段问题，在输入字符串中加入三个点，将字符串分为四段，每一段必须合法，求所有可能的情况。根据目前刷了这么多题，得出了两个经验，一是只要遇到字符串的子序列或配准问题首先考虑动态规划DP，二是只要遇到需要求出所有可能情况首先考虑用递归。这道题并非是求字符串的子序列或配准问题，更符合第二种情况，所以我们要用递归来解。我们用k来表示当前还需要分的段数，如果k = 0，则表示三个点已经加入完成，四段已经形成，若这时字符串刚好为空，则将当前分好的结果保存。若k != 0, 则对于每一段，我们分别用一位，两位，三位来尝试，分别判断其合不合法，如果合法，则调用递归继续分剩下的字符串，最终和求出所有合法组合，

### 94. Binary Tree Inorder Traversal

Iterative 方法

1.python check list is empty: if a: don't use len(a)>0 or isempty!!

2. check if root is None first

3. 用stack, 每次遇见一个node,就不停的push left 进stack,直到left 为None. 然后从stack里面拿出一个, print val, 然后在对right进行left pushin stack的方法循环.直到最好stack为空

### 96 Unique Binary Search Trees I

给出一个n,输出1-n的不同的bst有多少种

dynamic programming

don't forget n=1,n=2 return first

for i in range(3,n+1):

for j in range(0,n):

p[i]+=p[j]\*p[i-j-1]

### 95 Unique Binary Search Trees II

非高频medium

95题没有写要,写一遍

就是列出所有的结构.用recursion and memorization

### 97. Interleaving String

用二维数组动态规划来做。 注意的是，要先check s3的长度和s1+s2是否相等。不等直接返回false，否则容易超出边界。

第二，二维数组要用len+1. 因为你要保留一个0,0是代表两边都不取任何数字的。递推关系容易错，细心写。

### 98. Validate Binary Search Tree

输入一个BST,判断是否valid

勿忘记root 为none

非A 用 not a , 而不是 !A

写个helper, 同时输出valid,min,max 即可

返回用(True,min,max)

接受需要unpack,用valid,min,max = ...

### 99. Recover Binary Search Tree

非高频hard

### 100. Same Tree

比较两个tree是不是一样。 写个递归，先简单的，最后反复调用自身。

### 101. Symmetric Tree

注意要写个isMirror 的function。先比较两个root是不是空，同为空，然后val是否一样。然后递归比较 left.left vs right.right, left.right vs right.left

### 102. Binary Tree Level Order Traversal

搞一个list。然后每一层都依次从左到右把node放进list里面。然后最后把list 加紧res 里面

然后再跳进下一层创建新list。 继续下去。 直到那个list为空结束。

### 103. Binary Tree Zigzag Level Order Traversal

类似上题，但是要求zigzag。 就把res里面，每隔一层反序一次就可以了。

### 104. Maximum Depth of Binary Tree

求一棵树的最大深度。

递归,helper每次返回maxdepth+1

### 105. Construct Binary Tree from Preorder and Inorder Traversal

Binary tree一共3种访问顺序：preorder, inorder, postorder。这道题目的推广就是给定一个binary tree的两种访问顺序，重构该binary tree。这类题目给定的两个排序中，如果包括了inorder，且没有重复元素，就非常好解了。解这类题有两个关键点：

1. 在inorder中寻找root的位置，从而从序列中分割出左右子树。

Inorder: left subtree | root | right subtree

Preorder: root | left subtree | right subtree

Postorder: left subtree | right subtree | root

可见root是preorder序列的第一个节点，也是postorder的最后一个节点。所以给定这两个序列的任意一个我们即知道了root->val。通过搜索inorder序列，可以定位root所在的位置，从而也得到了left subtree和right subtree的节点数。

2. 递归构建

当root，left/right subtree都确定后

root->left = construct(inorder(left subtree), preorder(left subtree))

root->right = construct(inorder(right subtree), preorder(right subtree))

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

postorder的最后一个是root。所以从inorder里面找到，然后左边的是左子树的inorder，同样长度是postorder的left。递归调用。

易错点：递归右边子树的时候，postorder[i:], inorder[i+1:]

### 107. Binary Tree Level Order Traversal II

Easy pass.

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

每次找中点做root.

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

产生root的时候,注意不要写成listnode,而是treenode

### 110. Balanced Binary Tree

1. 這題會用到之前寫過的[LeetCode 104. Maximum Depth of Binary Tree](https://skyyen999.gitbooks.io/-leetcode-with-javascript/content/questions/questions/104md.md)
2. 尋找node左樹的深度與右樹的深樹後相減，如果差超過1，表示非高度平衡樹
3. 如果沒差超過1，傳入左節點與右節點繼續判斷是否為高度平衡樹

### **111. Minimum Depth of Binary Tree**

Easy。 递归

### **112. Path Sum**

输入一个tree和一个target，求是不是又一个path的sum等于target。必须要是root to leaf path

Easy. 递归。

### **113. Path Sum II**

这题要找出所有的sum。而不是判断有没有。

这题用dfs和递归。 list 可以通过+号直接串联

与上一题不同的是，要存储所有的root - leaf路径，所以在DFS的基础上我们还要增加backtracking。回溯的点有两个，一个是当前root满足条件，返回时，此时回溯保证当前root的sibling结果正确； 另外一个是遍历完当前root的左子树和右子树时， 这时回溯也是保证sibling结果正确。比如 root = [0, 1, 1], sum = 1，则计算完左边的1时可以正确计算到右边的1。或者 root = [0, 1, 1, 0, 0]， sum = 1， 遍历完左边的两个0后可以正确计算到右边1的结果。

### **114. Flatten Binary Tree to Linked List**

给一个binary tree，变成全部只有right child的tree

这题不要忘记改变以后把左child 设置成none

### **115. Distinct Subsequences**

看到有关字符串的子序列或者配准类的问题，首先应该考虑的就是用动态规划Dynamic Programming来求解，这个应成为条件反射。而所有DP问题的核心就是找出递推公式，想这道题就是递推一个二维的dp数组，下面我们从题目中给的例子来分析，这个二维dp数组应为：

首先，若原字符串和子序列都为空时，返回1，因为空串也是空串的一个子序列。若原字符串不为空，而子序列为空，也返回1，因为空串也是任意字符串的一个子序列。而当原字符串为空，子序列不为空时，返回0，因为非空字符串不能当空字符串的子序列。理清这些，二维数组dp的边缘便可以初始化了，下面只要找出递推式，就可以更新整个dp数组了。我们通过观察上面的二维数组可以发现，当更新到dp[i][j]时，dp[i][j] >= dp[i][j - 1] 总是成立，再进一步观察发现，当 T[i - 1] == S[j - 1] 时，dp[i][j] = dp[i][j - 1] + dp[i - 1][j - 1]，若不等， dp[i][j] = dp[i][j - 1]，所以，综合以上，递推式为：

dp[i][j] = dp[i][j - 1] + (T[i - 1] == S[j - 1] ? dp[i - 1][j - 1] : 0)

### **116. Populating Next Right Pointers in Each Node**

输入perfect binary tree

递推：在第i层的所有next pointer都连接好的情况下，如何连接第i+1层的next pointer?

显然从第i层的最左节点开始依次通过next pointer遍历这一层，同时将他们的children，即第i+1层的节点依次通过next pointer连接起来。连接的时候要分情况处理。

初始情况：对于顶层，只有一个节点root，所以该层连接已经完成。

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

输入的是any binary tree

### **118. Pascal's Triangle and** 119. Pascal's Triangle II

用nextrow来计算下一层，然后跟新row为nextrow。

易错点： 循环内还是用同样的变量做了指针，并且搞错了行数计算

### **120. Triangle**

输入一个三角，算出从上到底部最小的路径sum。

简单的dp。注意坐标不要搞错。

### **121. Best Time to Buy and Sell Stock**

输入一个数组，数字代表每天的股价，只能交易一次，求最大可能的利润

从左往右，存下左边的最低价。然后每天可以更新当天卖出的利润，也就是p-minp，保存最大利润。

输出最大的利润

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

你可以交易无限多次。求最多可能的利润

这样就把所有的涨的天数的利润都加起来就可以了。

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

你最多只能交易两次。求最多都利润  
动态规划

非高频hard 暂不收入

### **124. Binary Tree Maximum Path Sum**

1. 与常规path sum不同，这题的path sum可以不起始于root，也可以不终止于leaf。

2. 这样的path可以归纳为两种情况：

(1) root->leaf path中的一段：即题目例子中的1-2或1-3。

(2) 两个节点之间经过它们lowest common ancestor (LCA)的path：即题目中的2-1-3。

3. 显然top-down的递归并不适用于这题，因为对于类型(2)的path，它的path sum同时取决于LCA左右sub path的最大值。而这样的path sum是无法通过top-down传递来获取的。

4. 所以这里可以采用bottom-up的递归方法:

对于节点x:

定义PS1(x)为从x出发向leaf方向的第一类path中最大的path sum。

定义PS2(x)为以x为LCA的第二类path中的最大path sum。

显然如果我们求得所有节点x的PS1 & PS2，其中的最大值必然就是要求的max path sum。

5. 如何求PS1(x)、PS2(x)?

(1) PS1(x) 、PS2(x)至少应该不小于x->val，因为x自己就可以作为一个单节点path。

(2) PS1(x) 、 PS2(x)可以从PS1(x->left)和PS1(x->right)来求得：

PS1(x) = max [ max(PS1(x->left), 0), max(PS1(x->right), 0) ] + x->val

PS2(x) = max(PS1(x->left), 0) + max(PS1(x->right), 0) + x->val

注意这里并不需要PS2(x->left) 以及 PS2(x->right) 因为子节点的2型path无法和父节点构成新的path。

6. 需要返回PS1(x)以供上层的节点计算其PS1 & PS2.

7. 对于leaf节点：PS1(x) = PS2(x) = x->val.

### **125. Valid Palindrome**

判断一个字符串是不是palindrome。要求只考虑0-9a-z， 并且case insensative.

 可以用isalnum（）

### **127 Word Ladder I**

思路: 这个就是图论算法中的单源最短路, 求单源最短路比较通用的算法是BFS和Dijkstra, 其区别是BFS不能用于带权重的图中, 而后者可以, 可以看出在本题中两个字符串之间是无权重的, 也就是如果连通就是1, 不联通就是无穷. BFS和Dijkstra的区别是前者的时间复杂度是O(n), 后者最多优化到O(m log n), 所以如果条件成立一般选择BFS要更好.

其思路就是先把起点加到队列中, 然后每次将字典中与队首距离为1的字符串加进队列, 直到最后出队列的是终点字符串, 为确保终点字符串存在, 我们可以先在字典中加进去终点字符串.

而在本题中在寻找与一个字符串相距为1的的字典中另一个字符串时如果一个个遍历字典消耗时间比较多, 每次时间复杂度是O(n). 在单个字符串不是很长的情况下, 一个个查看改变一个字符然后在字典中查看是否存在效率要更高些, 其时间复杂度是O(k log n), 其中k为单个字符串长度, n为字典长度.

### **126. Word Ladder II**

个人感觉这道题是相当有难度的一道题，它比之前那道[Word Ladder 词语阶梯](http://www.cnblogs.com/grandyang/p/4539768.html)要复杂很多，全场第四低的通过率12.9%正说明了这道题的难度，我也是研究了网上别人的解法很久才看懂，然后照葫芦画瓢的写了出来，下面这种解法的核心思想是BFS，大概思路如下：我们的目的是找出所有的路径，我们建立一个路径集paths，用以保存所有路径，然后是起始路径p，在p中先把起始单词放进去。然后定义两个整型变量level，和minLevel，其中level是记录循环中当前路径的长度，minLevel是记录最短路径的长度，这样的好处是，如果某条路径的长度超过了已有的最短路径的长度，那么舍弃，这样会提高运行速度，相当于一种剪枝。还要定义一个set变量words，用来记录已经循环过的路径中的词，然后就是BFS的核心了，循环路径集paths里的内容，取出队首路径，如果该路径长度大于level，说明字典中的有些词已经存入路径了，如果在路径中重复出现，则肯定不是最短路径，所以我们需要在字典中将这些词删去，然后将words清空，对循环对剪枝处理。然后我们取出当前路径的最后一个词，对每个字母进行替换并在字典中查找是否存在替换后的新词，这个过程在之前那道[Word Ladder 词语阶梯](http://www.cnblogs.com/grandyang/p/4539768.html)里面也有。如果替换后的新词在字典中存在，将其加入words中，并在原有路径的基础上加上这个新词生成一条新路径，如果这个新词就是结束词，则此新路径为一条完整的路径，加入结果中，并更新minLevel，若不是结束词，解将新路径加入路径集中继续循环。

### **128. Longest Consecutive Sequence**

这道题要求求最长连续序列，并给定了O(n)复杂度限制，我们的思路是，使用一个集合set存入所有的数字，然后遍历数组中的每个数字，如果其在集合中存在，那么将其移除，然后分别用两个变量pre和next算出其前一个数跟后一个数，然后在集合中循环查找，如果pre在集合中，那么将pre移除集合，然后pre再自减1，直至pre不在集合之中，对next采用同样的方法，那么next-pre-1就是当前数字的最长连续序列，更新res即可

### **129. Sum Root to Leaf Numbers**

非高频medium

先求出每个叶子节点的值，然后将这些值相加。求叶子节点的值可以通过深度优先遍历，一遍往下，一遍增加各个节点的值，如果是叶子节点就将数组值的和加入到一个只有叶子节点的数组中，最后求这个只有叶子节点的数组的值得和。

### **130. Surrounded Regions**

非高频medium

一种比较巧妙的思路是从边缘的'O'出发，通过DFS，所有能够遍历的'O'都可以暂时被标记为'#'，那么剩下未能被标记的'O'说明被surrounded，需要在遍历结束之后全部转为'X'。（用DFS或者BFS要警惕栈溢出）

另一种方法是用Union Find，将与边缘相连通的'O'全部union到一个dummy node（也可以用hasEdge[]来存储，不过内存占用更多）, 最终将没有和这个dummy node是一个component的'O'点全部标记为'X'。

+

### **131. Palindrome Partitioning**

非高频medium

一看到return all xxxx，就猜到可能要用回溯。这道题就是比较典型的递归+回溯。递归前要判断当前的子字符串是否palindrome，答案是false的话要continue。

仔细想一想代码可以简化不少。主要分为三部分。1是题目给定的方法，2是辅助方法，用来递归和回溯，3是判断string是否是palindrome。注意考虑清楚需要多少变量，以及时间空间复杂度。

Time Complexity: O(n!)

Space Complexity: O(n ^ 2)

遇到要求所有组合、可能、排列等解集的题目，一般都是用DFS + backtracking来做。要分割回文的前提是能够判断回文。在做DFS的时候，如果每次从start出发查找s[start:end]是否是回文，会使算法复杂度大大增加。而在Longest Palindromic Substring这题中已经知道如何用DP来计算任意s[i:j]是否是回文。因此可以先计算该判断矩阵，在DFS的时候用来剪枝，大大提高效率。

### **132. Palindrome Partitioning II**

非高频hard

### **133. Clone Graph**

非高频medium

思路：克隆图的一个难点就是一个结点的邻居可能在已经出现过，这样你只要把他的指针加到邻居集合中即可，也有可能这个结点还没出现过，因此你需要新建一个这个结点，因此我们需要一个hash表来对结点做一一映射．

本题有两种方法来做，广度搜索BFS和深度搜索DFS.

做映射的时候，key是原图的node，val是新图的node。这样看见原图的node是已经出现的就可以直接找到新图的node。

### **134. Gas Station**

问题：在一个环形的路径上，有 n 个加油站。每个加油站 i 有汽油 gas[i] ，从 i 行驶到下一个加油站需要耗油 cost[i]，汽车的油箱没有限制。找出一个加油站，从该加油站能走完整个环形路径。其中，本题目只有唯一一个解。

思路：

环形路径并且需要走完一整圈，则有：

1)，若全部加油站的汽油 小于 路径总消耗油量，则不可能走完全程。

2)，反之，若全部加油站的汽油 大于等于 路径总消耗油量，那么必然有至少存在一个站可以出发走完全程。

在网上看到一个讲解，才明白，而解决方案的关键就是题目的限制条件“本题目只有唯一一个解”。

从i到j，如果i到不了j，那么i到j的中间的都到不了

所以只要从0到len-1， 看哪一个点可以走到最后tank不为负，就是那个点。因为如果为负了，那么一定不是中间的，可以继续从后面的找。

### **135. Candy**

非高频hard暂时不收入。

### 136. Single Number

给一串整数，每个数字出现2次，只有一个出现一次。求那个数字。

设置s为0. 然后让s^ 所有的数字。因为每个数字异或两次都是0. 0异或一个数字一次就变成那个数字。所以结果就是那个数字。

### **137. Single Number II**

Round 1: 简单bug free方法是先set,求和乘以3减去对list的求和.但是要extra memory

高级方法是因为每个bit的数字会出现3次和1次,所以一次对每个bit上的1进行count,如果不是3的倍数就留着,然后结果convert成32bit.但是对python,32位没有负数,所以要convert,convert方法是 if ans>=2\*\*31: ans-=2\*\*32

### **138. Copy List with Random Pointer**

1. 易错点,create 新node的时候不要写成ListNode,要写randomListNode

用dict,key是原来的list的node,value是新的list 的node

### **139. Word Break**

给一个字典,一个string,决定是否能字典里面的词组成string. Brutal force 递归的时间复杂是O(k^K), k为字典里面词的数目.

这道题仍然是动态规划的题目，我们总结一下动态规划题目的基本思路。首先我们要决定要存储什么历史信息以及用什么数据结构来存储信息。然后是最重要的递推式，就是如从存储的历史信息中得到当前步的结果。最后我们需要考虑的就是起始条件的值。

接下来我们套用上面的思路来解这道题。首先我们要存储的历史信息res[i]是表示到字符串s的第i个元素为止能不能用字典中的词来表示，我们需要一个长度为n的布尔数组来存储信息。然后假设我们现在拥有res[0,...,i-1]的结果，我们来获得res[i]的表达式。思路是对于每个以i为结尾的子串，看看他是不是在字典里面以及他之前的元素对应的res[j]是不是true，如果都成立，那么res[i]为true，写成式子是



假设总共有n个字符串，并且字典是用HashSet来维护，那么总共需要n次迭代，每次迭代需要一个取子串的O(i)操作，然后检测i个子串，而检测是constant操作。所以总的时间复杂度是O(n^2)（i的累加仍然是n^2量级），而空间复杂度则是字符串的数量，即O(n)。

### **140. Word Break II**

输入一个字典,一个string.可以在string里面加入空格,使得每个词都是字典里面的.求出所有加空格的组合.

这道题是之前那道[Word Break 拆分词句](http://www.cnblogs.com/grandyang/p/4257740.html)的拓展，那道题只让我们判断给定的字符串能否被拆分成字典中的词，而这道题加大了难度，让我们求出所有可以拆分成的情况，就像题目中给的例子所示。之前的版本中字典wordDict的数据类型是HashSet，现在的不知为何改成了数组vector，而且博主看到第二个例子就笑了，PPAP么，哈哈。

根据博主行走江湖多年的经验，像这种返回结果要列举所有情况的题，十有八九都是要用递归来做的。当我们一时半会没有啥思路的时候，先不要考虑代码如何实现，如果就给你一个s和wordDict，不看Output的内容，你会怎么找出结果。比如对于例子1，博主可能会先扫一遍wordDict数组，看有没有单词可以当s的开头，那么我们可以发现cat和cats都可以，比如我们先选了cat，那么此时s就变成了 "sanddog"，我们再在数组里找单词，发现了sand可以，最后剩一个dog，也在数组中，于是一个结果就出来了。然后回到开头选cats的话，那么此时s就变成了 "anddog"，我们再在数组里找单词，发现了and可以，最后剩一个dog，也在数组中，于是另一个结果也就出来了。那么这个查询的方法很适合用递归来实现，因为s改变后，查询的机制并不变，很适合调用递归函数。再者，我们要明确的是，如果不用记忆数组做减少重复计算的优化，那么递归方法跟brute force没什么区别，大概率无法通过OJ。所以我们要避免重复计算，如何避免呢，还是看上面的分析，如果当s变成 "sanddog"的时候，那么此时我们知道其可以拆分成sand和dog，当某个时候如果我们又遇到了这个 "sanddog"的时候，我们难道还需要再调用递归算一遍吗，当然不希望啦，所以我们要将这个中间结果保存起来，由于我们必须要同时保存s和其所有的拆分的字符串，那么可以使用一个HashMap，来建立二者之间的映射，那么在递归函数中，我们首先检测当前s是否已经有映射，有的话直接返回即可，如果s为空了，我们如何处理呢，题目中说了给定的s不会为空，但是我们递归函数处理时s是会变空的，这时候我们是直接返回空集吗，这里有个小trick，我们其实放一个空字符串返回，为啥要这么做呢？我们观察题目中的Output，发现单词之间是有空格，而最后一个单词后面没有空格，所以这个空字符串就起到了标记当前单词是最后一个，那么我们就不要再加空格了。接着往下看，我们遍历wordDict数组，如果某个单词是s字符串中的开头单词的话，我们对后面部分调用递归函数，将结果保存到rem中，然后遍历里面的所有字符串，和当前的单词拼接起来，这里就用到了我们前面说的trick。for循环结束后，记得返回结果res之前建立其和s之间的映射，方便下次使用

### **141. Linked List Cycle**

快慢指针。bug free. 快的走两步，慢的走一步。如果有出现指向同一个指针，就有cycle。如果快的走到none了，就没有cycle。 bug free

### **142. Linked List Cycle II**

slow, fast, fast 两步，slow 一步。 第一次相遇，快的整整比慢的多走一圈。 所有2n = n+k, 所以多走的n步就是cycle的长度。 此时slow的走了n步，进入圈子m步，走完整圈还需要k-m 步。（k是圈子的长度，也等于n）。 而如果此时从投开始，再走k-m步，正好是到圈子的起点，两者会在七点相遇。 还要注意slow fast都要从dummy head走起

**143. Reorder List**1. 一开始看错了题目.是前后两半,后面reverse再交织.而不是奇偶交织

2. reverse的方法写错了N次.牢记住: def reverse(self,head):

prev=None

p=head

while p!=None:

tmp=p.next

p.next=prev

prev=p

p=tmp

return prev

3. start from pre. so slow can stop at mid.

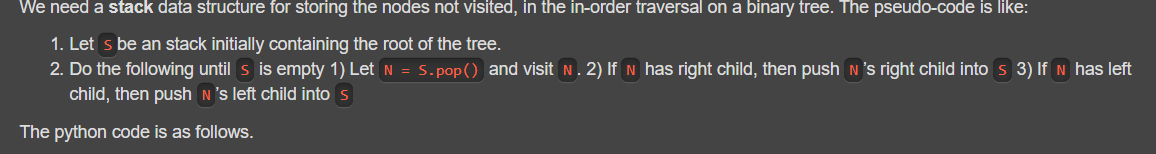
要重做

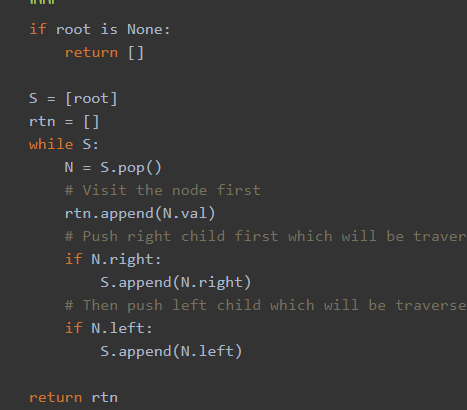
### 

### **144. Binary Tree Preorder Traversal**

非高频medium

要求用iterative methods





### **145. Binary Tree Postorder Traversal**

### **146. LRU Cache**

要求get 和put 操作都是O(1). 如果put的时候超过最大长度，就按照最近用的次数排序，删掉最迟用过的那个。

Queue肯定不行。 Hashmap没有顺序也不行。 linkedhashmap可以，但是这题就是让你自己写一个linkedhashmap

这题要用一个hashmap，还要用个doubly linkedlist。因为需要对头和尾都有控制，drop的时候可以迅速drop掉

首先不能用数组必须要用linkedlist，否则无法用O1时间取出某个node

我们需要个HashMap，来记录存的值, key为所插入的key，value为所存的node。其中，我们定义链表的头部为旧的元素， 尾部为最新的元素。

对于以下操作，

当set一个元素，hashmap的size< capacity的时候，我们之间将这个元素插入链表尾部。

当set一个元素，这个元素已经存在于hashMap中的时候，我们需要将这个元素的value更新，并且更新hashmap的索引，最后把这个元素拿出来，插到尾部。

当get一个元素的时候，我们需要返回这个元素值得同时，把这个元素拿出来，放到链表尾部。

### **147. Insertion Sort List**

！！！怎么做都不对要反复做

### **148. Sort List**

输入一个linkedlist，要求把它sort，时间复杂度O（nlogn）

看见O（nlogn）用分治法

还要写一个merge的函数

易错点是快慢指针找到中点以后，把slow链接的next设为None。否则无限循环。就是把两个list分隔开

### **149. Max Points on a Line**

解这个平面几何题有3个要点：

1. 如何判断共线?

两点成一直线，所以两点没有共线不共线之说。对于点p1(x1, y1)，p2(x2, y2)，p3(x3, y3)来说，共线的条件是p1-p2连线的斜率与p1-p3连线的斜率相同，即

(y2-y1)/(x2-x1) = (y3-y1)/(x3-x1)

所以对共线的n点，其中任意两点连线的斜率相同。

2. 如何判断最多的共线点?

对于每个点p出发，计算该点到所有其他点qi的斜率，对每个斜率统计有多少个点符合。其中最多的个数加1（出发点本身）即为最多的共线点。

3. 特殊情况

当x1 = x2，y1!=y2时，为垂直连线。计算斜率时分母为0会出错。

当x1 = x2，y1 = y2时，两点重合。则(x2, y2)和所有(x1, y1)的连线共线。

### **150. Evaluate Reverse Polish Notation**

python round float to 0 : int(float). 此外python 2 和python 3还不一样。python 3: int(-0.5) = 0. python2: int(-0.5)=-1

### **151. Reverse Words in a String**

把一句话里面的词倒序输出，用一个空格间隔。注意如果有多个空格连着也要变成一个。

split()对有多个空格的，会分出好几个空的元素。所以要去掉

### 152. Maximum Product Subarray

本题其实是Maximum Subarray Sum问题的延伸，不同的是这里需要考虑元素的符号。

DP的四要素

* 状态：
  + max\_product[i]: 以nums[i]结尾的max subarray product
  + min\_product[i]: 以nums[i]结尾的min subarray product
* 方程：
  + max\_product[i] = getMax(max\_product[i-1] \* nums[i], min\_product[i-1] \* nums[i], nums[i])
  + min\_product[i] = getMin(max\_product[i-1] \* nums[i], min\_product[i-1] \* nums[i], nums[i])
* 初始化：
  + max\_product[0] = min\_product[0] = nums[0]
* 结果：
  + 每次循环中 max\_product[i] 的最大值

### 153. Find Minimum in Rotated Sorted Array

和Search in Rotated Sorted Array I这题换汤不换药。同样可以根据A[mid]和A[end]来判断右半数组是否sorted：

原数组：0 1 2 4 5 6 7

情况1： **6** 7 0 **1** 2 4 **5**

情况2： **2** 4 5 **6** 7 0 **1**

**(1) A[mid] < A[end]**：A[mid : end] sorted => min不在A[mid+1 : end]中

**搜索A[start : mid]**

**(2) A[mid] > A[end]**：A[start : mid] sorted且又因为该情况下A[end]<A[start] => min不在A[start : mid]中

**搜索A[mid+1 : end]**

(3) base case：

a. start = end，必然A[start]为min，为搜寻结束条件。

b. start + 1 = end，此时A[mid] = A[start]，而min = min(A[mid], A[end])。而这个条件可以合并到(1)和(2)中。

### **155. Min Stack**

设计一个stack，可以在O（1）时间内完成下面操作

Push，pop，top，getmin

这题设计一个list作为stack，在设计一个list作为每一个stack元素对应的当时的min。 然后需要getmin的适合就输出min list的top就可以了

易错点是list有pop，没有peek。

### **157. Read N Characters Given Read4**

首先输入数组的长度我们是不知道的。

read4这个API中的参数是读了的char，也就是每次read4之后返回的char[4]是从给定文件中读的。

然后为啥是两参数？char[] buf和上面的一个道理，是输出的数组，而不是源文件；给int n是因为这个文件的长度可能比我们给的大，这样我们只需要读入N个字符到 char[] buf中；也有可能短，那么我们只能读完所有的文件并且返回文件长度。

### **158. Read N Characters Given Read4 II - Call multiple times**

比如先call了n=3，然后call n=5，那么第一次就读入了4个char，第二次call应该把上一次的最后一个char拿来。也就是说要有个cache取缓存已读取的字符，然后从这个cache里面取。

每次读4个字符，放入 cache，然后

in = Math.min(n - total, buff.size()) 这个是精髓，这样就知道还需要多少个字符了。如果是in=0了，那么说明已经够了或者没字符了，这样就不用再读取了。否则无论哪个大，接下来都需要往buf中放入in个字符（可能是剩的少，也有可能要求的少）。

### **160. Intersection of Two Linked Lists**

检测两个linkedlist从哪个节点开始重合的。

先检测两个的长度

然后快的先跑那么长度差。

然后同时同步往后跑，看从哪个开始一样

### 162. Find Peak Element

这道题是求数组的一个峰值，这个峰值可以是局部的最大值，这里用遍历整个数组找最大值肯定会出现Time Limit Exceeded，我们要考虑使用类似于二分查找法来缩短时间，由于只是需要找到任意一个峰值，那么我们在确定二分查找折半后中间那个元素后，和紧跟的那个元素比较下大小，如果大于，则说明峰值在前面，如果小于则在后面。这样就可以找到一个峰值了，

### 164. Maximum Gap

在一个没排序的数组里，找出排序后的相邻数字的最大差值。

要求用线性时间和空间。

如果用nlgn的话，直接排序然后判断就可以了。so easy

但我们要的是线性时间。

其实这个思想在算法课上有讲过。用桶的思想。把数组分成几个桶，然后判断相邻桶的最大与最小之间的差值。关键是要知道每个桶的长度，已经桶的个数。

### 165. Compare Version Numbers

按照. Split, 然后依次比较.

don't forget 1.0 and 1 are same version

### 167. Two Sum II - Input array is sorted

用hashtable或者two pointer因为sorted，所以two pointer从两边开始，和小就左指针右移，和大就右指针左移动

### 168. Excel Sheet Column Title

1. need to do again. hard. be careful. build list first. every time need to use num-1 for both % and //

### 169. Majority Element

输入一组数，要求出现次数大于一半的数字

可以做个dictionary。首先通过长度算出threshold = len/2

然后一次往dict里面放，如果频次大于len/2,就找到return了

O（n）

### **171. Excel Sheet Column Number**

每次取最后一位，乘以26的对应的倍数，然后把字符串去掉最后一位，直到字符串为0

### 172. Factorial Trailing Zeroes

forget n==5 case. 先把n//5的结果加上去。然后divider乘以5.再把n//25d结果加上去。以此类推。

### 173. Binary Search Tree Iterator

写一个二叉查找树，每次返回树中的下一个最小节点.hasnext and next 平均 O(1), memory O(h), h 为tree height

思路：维护一个栈，先将根结点的左子树全部压栈，每次弹出栈顶元素，若某次弹出的栈顶元素有右子树，比如3，此时需要将以该节点的右子树为根的子树的左子节点全部压栈

### 179. Largest Number

定一个数组，求将数组元素拼凑成一个值最大的整数。

将整型数组的转化为字符串数组，用 std::sort 对字符串数组排序，使得排序后元素直接连接成为最大值整数。

使用默认的比较函数排序，会得到不一定是最大值，例如 {360, 36} 排序后得到的是 36036 ，明显不是小于 36360。解决这个问题，就是重写比较函数，使得排序后的两个元素连接成为较大的整数即可，很简洁。

解答這邊提供一個很簡單的思維:

A # B > B # A, then A > B

有這種關係就可以排序。

Python 2 里面用sorted(nums,cmp=func) 可以排序

Python 3.6里面remove了cmp，所以要写成

class LargerNumKey(str):

def \_\_lt\_\_(x, y):

return x+y > y+x

sorted(map(str, nums), key=LargerNumKey)

### 187. Repeated DNA Sequences

这题已经说了只要10 letter long的substing

最直接的想法就是从头遍历，把substring加入到HashMap中，然后进行比较

这题要注意的是, d is a dict. For k in d就可以遍历keys,不需要for k in d.keys()

此外,for k in d 比for k in d.keys() 要快得多.

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

Say you have an array for which the ith element is the price of a given stock on day i.

If you were only permitted to complete at most k transaction (ie, buy one and sell one share of the stock), design an algorithm to find the maximum profit.

分析如下：

感觉是一道非常困难的题目，不会做，全面看网上的讨论。

题目的关键是要写出下面的动态转移方程:

local[i][j]=max(global[i-1][j-1]+max(diff,0),local[i-1][j]+diff)，

global[i][j]=max(local[i][j],global[i-1][j])，

而这个方程的具体理解是这样的。

首先global比较简单，不过是不断地和已经计算出的local进行比较，把大的保存在global中。

然后看local,关键是要理解local的定义，local[i][j]表示，前i天进行了j次交易，并且第i天进行了第j次交易的最大利润，所以local[i][j]中必然有一次交易，也就是当近一次交易，发生在第i天。 local由两个部分的比较完成。

第一部分是，global[i-1][j-1]+max(diff,0), 表示的就是，前面把之前的j - 1次交易，放在之前的i - 1天，然后让第i天来进行第j次交易，那么加入此时diff(price[i] - price[i - 1])大于零，那么正好可以可借助这次交易的机会增长里利润(利润= diff)，否则的话，如果diff小于零，那就在第i天当天进行一次买卖，凑一次交易的次数，但是产生利润为0.

第二部分是, local[i-1][j]+diff， 这里的 local[i-1][j]表示的是，前面j次交易在第i -1天就已经完成了，可是因为说了local[a][b]一定要表达在第a天完成了b次交易的最大利润，所以就需要强制使得交易在第i天发生，为了实现这一点，只需要在local[i - 1][j]的基础上，加上diff ( = price[i] - price[i - 1])就可以了。如果diff < 0 那也没有办法，因为必须满足local的定义。接下来算global的时候，总会保证取得一个更大的值。

关于正道题目的全文的分析，[Code Ganker](http://blog.csdn.net/linhuanmars/article/details/23236995)的还挺不错。 下面全文摘抄作者的分析:

"这道题是Best Time to Buy and Sell Stock的扩展，现在我们最多可以进行两次交易。我们仍然使用动态规划来完成，事实上可以解决非常通用的情况，也就是最多进行k次交易的情况。

这里我们先解释最多可以进行k次交易的算法，然后最多进行两次我们只需要把k取成2即可。我们还是使用“局部最优和全局最优解法”。我们维护两种量，一个是当前到达第i天可以最多进行j次交易，最好的利润是多少（global[i][j]），另一个是当前到达第i天，最多可进行j次交易，并且最后一次交易在当天卖出的最好的利润是多少（local[i][j]）。下面我们来看递推式。

全局的比较简单，

global[i][j]=max(local[i][j],global[i-1][j])，

也就是去当前局部最好的，和过往全局最好的中大的那个（因为最后一次交易如果包含当前天一定在局部最好的里面，否则一定在过往全局最优的里面）。

对于局部变量的维护，递推式是

local[i][j]=max(global[i-1][j-1]+max(diff,0),local[i-1][j]+diff)，

也就是看两个量，第一个是全局到i-1天进行j-1次交易，然后加上今天的交易，如果今天是赚钱的话（也就是前面只要j-1次交易，最后一次交易取当前天），第二个量则是取local第i-1天j次交易，然后加上今天的差值（这里因为local[i-1][j]比如包含第i-1天卖出的交易，所以现在变成第i天卖出，并不会增加交易次数，而且这里无论diff是不是大于0都一定要加上，因为否则就不满足local[i][j]必须在最后一天卖出的条件了）。

上面的算法中对于天数需要一次扫描，而每次要对交易次数进行递推式求解，所以时间复杂度是O(n\*k)，如果是最多进行两次交易，那么复杂度还是O(n)。空间上只需要维护当天数据皆可以，所以是O(k)，当k=2，则是O(1)。"

### 

### 189. Rotate Array

easy。三种方法

### 190. Reverse Bits

res=res<<1+n&1 not working

must be res<<=1 res+=n&1

此外，32位所以要用i in range(32), 才行。否则res往左移动的位数不够。

### **191. Number of 1 Bits**

输出一个数字里面有多少个1 bits

n>>1 输出右移的数字，并不改变原数字。所以要n>>=1

用n&1 来一位一位读，然后右移。

或者用 bin(n).count(‘1’). 注意，这个写法不用用于count(‘0’)!!! 因为bin产生的是binary literals不是32位的表示！！！

### **198. House Robber**

简单的dynamic programming

Bug free

### 200. Number of Islands

非常基础的一道DFS深度优先搜索的题目，当我们遍历数组发现一个位置为'1'时，就是用dfs将其周围的1全部变为0，然后递归继续，这样所有连通成一片的'1'就是一个island，sum++，最后返回sum即可。

### 202. Happy Number

题目说的是对任意一个正整数，不断各个数位上数字的平方和，若最终收敛为1，则该数字为happy number，否则程序可能从某个数开始陷入循环。

这里我们使用一个哈希map表存储已经出现过的数字，如果下次还出现，则返回false。

### 203. Remove Linked List Elements

注意check last element is none

### 204. Count Primes

Count the number of prime numbers less than a non-negative number,n.  
Solution  
这道题首先想到的方法是从1到n-1依次验证是否为prime。但是这样的时间复杂度为O(n^2)。  
如果使用埃拉托斯特尼筛法（Sieve of Eratosthenes），则可以将时间复杂度降低到O(nloglogn)。具体方法如下：

我们从2开始遍历到根号n，先找到第一个质数2，然后将其所有的倍数全部标记出来，然后到下一个质数3，标记其所有倍数，一次类推，直到根号n，此时数组中未被标记的数字就是质数。之所以不用考虑比自己小的倍数（比如3从3倍开始而不用考虑2倍）是因为自己已经被作为小的数的倍数考虑过（即2的3倍），因此每一个数都从自身倍数开始直到n

下面这个代码非常牛叉。

def countPrimes(self, n):

**if** n < 3:

**return** 0

primes = [True] \* n

primes[0] = primes[1] = False

**for** i **in** range(2, int(n \*\* 0.5) + 1):

**if** primes[i]:

primes[i \* i: n: i] = [False] \* len(primes[i \* i: n: i])

**return** sum(primes)

### 205. Isomorphic Strings

做2个map。依次比对。要做两个dict，互相转换。如果只做一个容易不对。

### **206. Reverse Linked List**

要用两种方法做。iteratively 和recursively

细心些按照一个case走一遍

### 207. Course Schedule

求Course Schedule，等同问题是有向图检测环，vertex是course， edge是prerequisite。我觉得一般会使用Topological Sorting拓扑排序来检测。一个有向图假如有环则不存在Topological Order。一个DAG的Topological Order可以有大于1种。 常用的Topological Sorting算法有两种

1. Kahn's Algorithms (wiki)： BFS based， start from with vertices with 0 incoming edge，insert them into list S，at the same time we remove all their outgoing edges，after that find new vertices with 0 incoming edges and go on. 详细过程见Reference里Brown大学的课件。
2. Tarjan's Algorithms (wiki)： DFS based， loop through each node of the graph in an arbitrary order，initiating a depth-first search that terminates when it hits any node that has already been visited since the beginning of the topological sort or the node has no outgoing edges (i.e. a leaf node). 详细过程见Reference里 NYU的课件。

知道了理论，下面就是coding了，早就发现自己的coding能力确实比较弱，即使明白算法也很难迅速写出正确的程序。多练习，一点点进步吧。互换edge[0]和edge[1]也能检测环，不过输出的就是逆序的Tolopolical order了。

### 208. Implement Trie (Prefix Tree)

于TrieNode的设计:

1. 一般的R-way Trie就是每一个节点TrieNode有R个子节点，我们可以用一个数组来表示子节点。
2. 数组的大小要根据题意来定，比如这道题说明了alphabet = 'a' 到 'z', 那么我们的R就等于26，做一个26-way Trie就可以了。
3. 要有一个boolean变量isWord来表明这个节点是否是单词的结尾。
4. 在Trie里面的首先要初始化一个root节点。 TrieNode root = new TrieNode(); 这个节点我们不保存任何字母。

对于insert

1. 首先处理一下边界条件
2. 设置一个变量d = 0代表深度depth
3. 设置一个TrieNode node = root，这里算是获取一个root的reference
4. 当d < word.length()时，我们根据word.charAt(d) - 'a'来算得 word的第一个字母应该保存的位置index
5. 查看word.next[index]，假如其为空，那么我们要创建一个新的TrieNode。不为空的时候不用管，直接运行6。
6. 更新node = node.next[index]， d++， 继续处理word中的下一个字母
7. 全部遍历完毕以后，设置node.isWord = true，表明root到这个node的路径是一个单词。

对于search和startWith

1. 1 ~ 4步跟insert都一样
2. 查看woird.next[index]，假如其为空直接返回false
3. 更新node = node.next[index], d++，继续查找word中的下一个字母
4. 最后一步，对于Search来说，我们要根据node.isWord来决定是否查找成功。 对于startWith来说我们直接返回true。

### 215. Kth Largest Element in an Array

用heap 或者quick select

build heap 是O（n）

heapify 是O（logn）

所以build heap，然后依次取k给即可。

用quick select 也是O(n)

### 218. The Skyline Problem

扫描线

思路  
如果按照一个矩形一个矩形来处理将会非常麻烦，我们可以把这些矩形拆成两个点，一个左上顶点，一个右上顶点。将所有顶点按照横坐标排序后，我们开始遍历这些点。遍历时，通过一个堆来得知当前图形的最高位置。堆顶是所有顶点中最高的点，只要这个点没被移出堆，说明这个最高的矩形还没结束。对于左顶点，我们将其加入堆中。对于右顶点，我们找出堆中其相应的左顶点，然后移出这个左顶点，同时也意味这这个矩形的结束。具体代码中，为了在排序后的顶点列表中区分左右顶点，左顶点的值是正数，而右顶点值则存的是负数。  
  
注意  
堆中先加入一个零点高度，帮助我们在只有最矮的建筑物时选择最低值

### 224. Basic Calculator

这道题让我们实现一个基本的计算器来计算简单的算数表达式，而且题目限制了表达式中只有加减号，数字，括号和空格，没有乘除，那么就没啥计算的优先级之分了。于是这道题就变的没有那么复杂了。我们需要一个栈来辅助计算，用个变量sign来表示当前的符号，我们遍历给定的字符串s，如果遇到了数字，由于可能是个多位数，所以我们要用while循环把之后的数字都读进来，然后用sign\*num来更新结果res；如果遇到了加号，则sign赋为1，如果遇到了符号，则赋为-1；如果遇到了左括号，则把当前结果res和符号sign压入栈，res重置为0，sign重置为1；如果遇到了右括号，结果res乘以栈顶的符号，栈顶元素出栈，结果res加上栈顶的数字，栈顶元素出栈。

### 226. Invert Binary Tree

Bug free

### 227. Basic Calculator II

这道题是之前那道[Basic Calculator 基本计算器](http://www.cnblogs.com/grandyang/p/4570699.html)的拓展，不同之处在于那道题的计算符号只有加和减，而这题加上了乘除，那么就牵扯到了运算优先级的问题，好在这道题去掉了括号，还适当的降低了难度，估计再出一道的话就该加上括号了。不管那么多，这道题先按木有有括号来处理，由于存在运算优先级，我们采取的措施是使用一个栈保存数字，如果该数字之前的符号是加或减，那么把当前数字压入栈中，注意如果是减号，则加入当前数字的相反数，因为减法相当于加上一个相反数。如果之前的符号是乘或除，那么从栈顶取出一个数字和当前数字进行乘或除的运算，再把结果压入栈中，那么完成一遍遍历后，所有的乘或除都运算完了，再把栈中所有的数字都加起来就是最终结果了。

### **234. Palindrome Linked List**

判断一个linkedlist是不是回文的，难点在followup上只要用O（1）的extra space，O（n) time非常复杂！！！重做！

O（1）的case非常复杂，要重做！！第一种找midpoint的时候注意，第二reverse最好reverse后半个，这样不用改中间的next=None，第三最后reverse back

### 235. Lowest Common Ancestor of a Binary Search Tree

Bug free

### 236. Lowest Common Ancestor of a Binary Tree

普通二叉树求公共祖先。看过<剑指Offer>以后知道这道题应该形成一系列问题。比如是不是二叉树，是不是BST。假如是BST的话我们可以用上题的方法，二分搜索。有没有指向父节点的link，假如有指向父节点的link我们就可以用intersection of two lists的方法找到两个linked list相交的地方。 对这道题目，我们使用后续遍历来做：  
  
定义两个辅助节点，使用后续遍历来遍历整个树  
当root的值等于p或者q时，找到一个符合条件的节点，返回这个root  
先遍历左子树  
再遍历右子树  
当left，right均找到时返回此root  
只找到left时返回left.(left是递归输入left以后输出的结果)  
只找到right时返回right（right是递归right以后输出的结果）不是说left 子树和right子树！！  
否则返回null

### 237. Delete Node in a Linked List

Bug free

### 238. Product of Array Except Self

用res来保存。先从后往前算乘积。 然后从前往后，用临时变量p保存之前的乘积，然后乘以res[i+1]即可

最后注意res[0] and res[-1] 的边界情况。总之就是res[i] = p\*res[i+1]

### 239. Sliding Window Maximum

这道题给定了一个数组，还给了一个窗口大小k，让我们每次向右滑动一个数字，每次返回窗口内的数字的最大值，而且要求我们代码的时间复杂度为O(n)。提示我们要用双向队列deque来解题，并提示我们窗口中只留下有用的值，没用的全移除掉。果然Hard的题目我就是不会做，网上看到了别人的解法才明白，解法又巧妙有简洁，膜拜啊。大概思路是用双向队列保存数字的下标，遍历整个数组，如果此时队列的首元素是i - k的话，表示此时窗口向右移了一步，则移除队首元素。然后比较队尾元素和将要进来的值，如果小的话就都移除，然后此时我们把队首元素加入结果中即可

### 240. Search a 2D Matrix II

1. 复杂度要求——O(m+n) time and O(1) extra space，同时输入只满足自顶向下和自左向右的升序，行与行之间不再有递增关系，与上题有较大区别。时间复杂度为线性要求，因此可从元素排列特点出发，从一端走向另一端无论如何都需要m+n步，因此可分析对角线元素。
2. 首先分析如果从左上角开始搜索，由于元素升序为自左向右和自上而下，因此如果target大于当前搜索元素时还有两个方向需要搜索，不太合适。
3. 如果从右上角开始搜索，由于左边的元素一定不大于当前元素，而下面的元素一定不小于当前元素，因此每次比较时均可排除一列或者一行元素（大于当前元素则排除当前行，小于当前元素则排除当前列，由矩阵特点可知），可达到题目要求的复杂度。

### **242. Valid Anagram**

判断两个word是不是anagram。 用hashtable。注意最后要检查一遍每个都是0. Hashtable的遍历：

for key, value in d.items():

如果通过index，val 遍历，用

For i,v in enumerate(d)

如果忘了怎么遍历，做一个26长度的list，手工直接索引。

### 253. Meeting Rooms II

输入一串interval，都是所有会议的开始结束时间。 问一共需要几间meeting room.

给定一个interval数组，求最少需要多少间教室。初始想法是扫描线算法sweeping-line algorithm，先把数组排序，之后维护一个min-oriented heap。遍历排序后的数组，每次把interval[i].end加入到heap中，然后比较interval.start与pq.peek()，假如interval[i].start >= pq.peek()，说明pq.peek()所代表的这个meeting已经结束，我们可以从heap中把这个meeting的end time移除，继续比较下一个pq.peek()。比较完毕之后我们尝试更新maxOverlappingMeetings。 像扫描线算法和heap还需要好好复习， 直线，矩阵的相交也可以用扫描线算法。

### 258. Add Digits

考虑数字和是9 的倍数。 簡單說就是判斷一個數是否為9的倍數，可以從每個位數相加是否能被9整除直接判斷，運用這樣的想法，直接取這個數除9的餘數

但是要注意如果被9整除，除了是0的时候，都应该返回9.

### 260. Single Number III

然后给出整个算法的具体思路，假设数组中两个不同的数字为 A 和 B；

1. 通过遍历整个数组并求整个数组所有数字之间的 XOR，根据 XOR 的特性可以得到最终的结果为 AXORB = A XOR B；
2. 通过某种特定的方式，我们可以通过 AXORB 得到在数字 A 和数字 B 的**二进制**下某一位不相同的位；**因为A 和 B 是不相同的，所以他们的二进制数字有且至少有一位是不相同的**。我们将这一位设置为 1，并将所有的其他位设置为 0，我们假设我们得到的这个数字为 bitFlag；
3. 那么现在，我们很容易知道，数字 A 和 数字 B 中必然有一个数字与上 bitFlag 为 0；因为bitFlag 标志了数字 A 和数字 B 中的某一位不同，那么在数字 A 和 B 中的这一位必然是**一个为 0，另一个为 1**；而我们在 bitFlag 中将其他位都设置为 0，那么**该位为 0 的数字与上 bitFlag 就等于 0，而该位为 1 的数字与上 bitFlag 就等于 bitFlag**
4. 现在问题就简单了，我们只需要在循环一次数组，将与上 bitFlag 为 0 的数字进行 XOR 运算，与上 bitFlag 不为 0 的数组进行独立的 XOR 运算。那么最后我们得到的这两个数字就是 A 和 B。

### 263. Ugly Number

检查数字是不是只是2，3，5的倍数。如果是就是ugly number

这题用递归来做比较好。 注意考虑0和负数。

### **268. Missing Number**

输入一个数组包括n个不同的从0-n里面抽出的int。求缺的那个。要求O（1）space，O（n） time

用求和公式先求出0-n的和，然后求数组的和，差就是缺的那个数字。

### 269. Alien Dictionary

这道题也是卡了很久，原因在于题意理解不明确。方法还是Topological Sorting，这题的意思是 - Words are sorted， 跟每个Word里面的letter顺序并没有什么关系。就比如"apple"排在"banana"前面，但并不是"apple"里面的"a"就排在"p"前面。所以之前花了不少时间根据每个单词内部的顺序构造edges，结果都是错的。 这种东西归根结底还是我自己的英语不好。写了一大堆总是通不过，发现题目真正含义以后就好很多了，以后面试的话也要多交流多沟通，免得题意弄不清楚白浪费时间做题。另外，对于图来说，图的三种表达方式, list of edges, ajacency matrix和ajacency lists一定要搞清楚，多弄明白。

Kahn's Algorithm: 先把输入words转化为Adjacency Lists或者list of edges，然后使用Course Schedule II的方法进行Topological Sorting。这里我们使用Kahn's Algorithm，先计算inDegree，然后维护一个Queue来进行BFS。

### 272. Closest Binary Search Tree Value II

这道题是之前那道[Closest Binary Search Tree Value](http://www.cnblogs.com/grandyang/p/5237170.html)的拓展，那道题只让我们找出离目标值最近的一个节点值，而这道题让我们找出离目标值最近的k个节点值，难度瞬间增加了不少，我最先想到的方法是用中序遍历将所有节点值存入到一个一维数组中，由于二分搜索树的性质，这个一维数组是有序的，然后我们再在有序数组中需要和目标值最近的k个值就简单的多

还有一种解法是直接在中序遍历的过程中完成比较，当遍历到一个节点时，如果此时结果数组不到k个，我们直接将此节点值加入res中，如果该节点值和目标值的差值的绝对值小于res的首元素和目标值差值的绝对值，说明当前值更靠近目标值，则将首元素删除，末尾加上当前节点值，反之的话说明当前值比res中所有的值都更偏离目标值，由于中序遍历的特性，之后的值会更加的遍历，所以此时直接返回最终结果即可，

### 273. Integer to English Words

把123转成one hundred twenty three

这道题让我们把一个整型数转为用英文单词描述，就像在check上写钱数的方法，我最开始的方法特别复杂，因为我用了几个switch语句来列出所有的单词，但是我看网上大神们的解法都是用数组来枚举的，特别的巧妙而且省地方，膜拜学习中。题目中给足了提示，首先告诉我们要3个一组的进行处理，而且题目中限定了输入数字范围为0到231 - 1之间，最高只能到billion位，3个一组也只需处理四组即可，那么我们需要些一个处理三个一组数字的函数，我们需要把1到19的英文单词都列出来，放到一个数组里，还要把20,30，... 到90的英文单词列出来放到另一个数组里，然后我们需要用写技巧，比如一个三位数n，百位数表示为n/100，后两位数一起表示为n%100，十位数表示为n%100/10，个位数表示为n%10，然后我们看后两位数是否小于20，小于的话直接从数组中取出单词，如果大于等于20的话，则分别将十位和个位数字的单词从两个数组中取出来。然后再来处理百位上的数字，还要记得加上Hundred。主函数中调用四次这个帮助函数，然后中间要插入"Thousand", "Million", "Billion"到对应的位置，最后check一下末尾是否有空格，把空格都删掉，返回的时候检查下输入是否为0，是的话要返回'Zero'

### 279. Perfect Squares

给定一个正整数n，求相加等于n的完全平方数（例如 1, 4, 9, 16, ...）的最小个数。例如，给定n = 12，返回3，因为12 = 4 + 4 + 4；给定n = 13，返回2，因为13 = 4 + 9。

下面是最简单的dp，核心的递推公式是对于j= 1 , j \* j <= i来说， tmp = Math.min(tmp, tmp[i - j \* j] + 1);

280. Wiggle Sort

先sort，然后从1 开始，swap i and i+1, 每次向后移动2位

nums=sorted(nums) 不能inplace sort，所以不对

要想inplace，必须要nums.sort()

### 283. Move Zeroes

Inplace 把所有0移到数组后面

Bug free

### 287. Find the Duplicate Number

给定一个包含n + 1个整数的数组，其中每一个整数均介于[1, n]之间，证明其中至少有一个重复元素存在。假设只有一个数字出现重复，找出这个重复的数字。

注意：

1. 不可以修改数组（假设数组是只读的）
2. 只能使用常数空间
3. 运行时间复杂度应该小于O(n2)
4. 数组中只存在一个重复数，但是可能重复多次

这道题给了我们n+1个数，所有的数都在[1, n]区域内，首先让我们证明必定会有一个重复数，这不禁让我想起了小学华罗庚奥数中的抽屉原理(又叫[鸽巢原理](https://zh.wikipedia.org/wiki/%E9%B4%BF%E5%B7%A2%E5%8E%9F%E7%90%86)), 即如果有十个苹果放到九个抽屉里，如果苹果全在抽屉里，则至少有一个抽屉里有两个苹果，这里就不证明了，直接来做题吧。题目要求我们不能改变原数组，即不能给原数组排序，又不能用多余空间，那么哈希表神马的也就不用考虑了，又说时间小于O(n2)，也就不能用brute force的方法，那我们也就只能考虑用二分搜索法了，我们在区别[1, n]中搜索，首先求出中点mid，然后遍历整个数组，统计所有小于等于mid的数的个数，如果个数大于mid，则说明重复值在[mid+1, n]之间，反之，重复值应在[1, mid-1]之间，然后依次类推，直到搜索完成，此时的low就是我们要求的重复值

### 289. Game of Life

这道题是有名的[康威生命游戏](https://zh.wikipedia.org/wiki/%E5%BA%B7%E5%A8%81%E7%94%9F%E5%91%BD%E6%B8%B8%E6%88%8F), 而我又是第一次听说这个东东，这是一种细胞自动机，每一个位置有两种状态，1为活细胞，0为死细胞，对于每个位置都满足如下的条件：

1. 如果活细胞周围八个位置的活细胞数少于两个，则该位置活细胞死亡

2. 如果活细胞周围八个位置有两个或三个活细胞，则该位置活细胞仍然存活

3. 如果活细胞周围八个位置有超过三个活细胞，则该位置活细胞死亡

4. 如果死细胞周围正好有三个活细胞，则该位置死细胞复活

由于题目中要求我们用置换方法in-place来解题，所以我们就不能新建一个相同大小的数组，那么我们只能更新原有数组，但是题目中要求所有的位置必须被同时更新，但是在循环程序中我们还是一个位置一个位置更新的，那么当一个位置更新了，这个位置成为其他位置的neighbor时，我们怎么知道其未更新的状态呢，我们可以使用状态机转换：

状态0： 死细胞转为死细胞

状态1： 活细胞转为活细胞

状态2： 活细胞转为死细胞

状态3： 死细胞转为活细胞

最后我们对所有状态对2取余，那么状态0和2就变成死细胞，状态1和3就是活细胞，达成目的。我们先对原数组进行逐个扫描，对于每一个位置，扫描其周围八个位置，如果遇到状态1或2，就计数器累加1，扫完8个邻居，如果少于两个活细胞或者大于三个活细胞，而且当前位置是活细胞的话，标记状态2，如果正好有三个活细胞且当前是死细胞的话，标记状态3。完成一遍扫描后再对数据扫描一遍，对2取余变成我们想要的结果。

### 292. Nim Game

有一堆石头。每次可以取1到3个。你是先取的。拿到最后一个石头的获胜。 假设你们都用optimal的策略。求函数返回你是否能胜。

考慮到下面幾總情況

石頭數量小於等於3，直接獲勝

1. 石頭數量等於4個，輸
2. 石頭數量5個，先拿走1個，獲勝
3. 由以上可以推論(5-1=4, 6-2=4, 7-3=4) 5~7都可以獲勝
4. 石頭數量8個，不管怎樣都會輸

以上可以得到結論，只要一開始的石頭數量為4的倍數，就會輸。

### 295. Find Median from Data Stream

设计一种data structure，可以不停的加入number，而且还可以返回当前所有number的median。

在Data stream中找到median。这道题是Heap的经典应用，需要同时维护一个最大堆和一个最小堆， 最大堆和最小堆的size <= 当前数字count / 2。在学习heap数据结构的时候一般都会讲到这一题，很经典。

寻找datastream或者array中的中位数， 可以用两个heap来实现，一个max heap，一个min heap。 每次加一个数先放到max heap，然后把max heap中poll一个数出来放到min heap。 同时要检查下max heap中的元素比min heap中要多；取中位数的时候，如果两个heap的size相同，则各poll一个元素出来算平均，否则max heap中poll出的就是median.

### 297. Serialize and Deserialize Binary Tree

解法I：二叉树的先序遍历

解法II：二叉树的层次遍历

### 300. Longest Increasing Subsequence

输入一串乱序的数字，给出最长的连续增长的subsequence

用记忆化recursion。

或者动态规划

我们维护一个一维dp数组，其中dp[i]表示以nums[i]为结尾的最长递增子串的长度，对于每一个nums[i]，我们从第一个数再搜索到i，如果发现某个数小于nums[i]，我们更新dp[i]，更新方法为dp[i] = max(dp[i], dp[j] + 1)，即比较当前dp[i]的值和那个小于num[i]的数的dp值加1的大小，我们就这样不断的更新dp数组，到最后dp数组中最大的值就是我们要返回的LIS的长度

### 301. Remove Invalid Parentheses

输入一串括号，去除最少的字符使得括号valide。输出所有的结果

对于一个字符串，在任何时候如果')'的个数多于左括号，则说明从开始到现在位置必然可以删除一个')'．而这段子串可能包含多个')'，删除哪一个呢？当然删除任何一个都可以．例如对于()())()，从开头到s[4]位置构成的子串多了一个右括号，因此我们需要删掉一个，而这个子串有三个右括号，但是只会产生２个结果，也就是会有一个重复值．所以在删除括号的时候，为保证不会产生重复值，需要记录一个最后删除的位置，这样可以使得在接下来删除的时候只删除这个位置之后的值．这样我们可以使得当前这一段子串不再包含多余的右括号了．这样我们可以删除了一个右括号之后合法的子串与后面还没有检查过的子串组成一个新的字符串重新开始检查．直到不再含有非法的右括号．

但是还有一种情况是包含了多余的左括号，一种直观的方法是从右向左再按照上面的方法处理一遍左括号．但是将数组逆置之后就可以重用上面的算法了．

所以总的思路就是先对字符串进行处理使得其不再含有非法右括号，然后将其翻转以后再检查是否含有非法的左括号．最后左右括号都检查完之后都合法就是我们要的答案了．

时间复杂度应该是O(n^2)

### 303. Range Sum Query - Immutable

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

设置sums 为len+1,否则很容易错

### 312. Burst Balloons

这道题提出了一种打气球的游戏，每个气球都对应着一个数字，我们每次打爆一个气球，得到的金币数是被打爆的气球的数字和其两边的气球上的数字相乘，如果旁边没有气球了，则按1算，以此类推，求能得到的最多金币数。像这种求极值问题，我们一般都要考虑用动态规划Dynamic Programming来做，我们维护一个二维动态数组dp，其中dp[i][j]表示打爆区间[i,j]中的所有气球能得到的最多金币。题目中说明了边界情况，当气球周围没有气球的时候，旁边的数字按1算，这样我们可以在原数组两边各填充一个1，这样方便于计算。这道题的最难点就是找递归式，如下所示：

dp[i][j] = max(dp[i][j], nums[i - 1]\*nums[k]\*nums[j + 1] + dp[i][k - 1] + dp[k + 1][j]) ( i ≤ k ≤ j )

有了递推式，我们可以写代码，我们其实只是更新了dp数组的右上三角区域，我们最终要返回的值存在dp[1][n]中，其中n是两端添加1之前数组nums的个数。

### 319. Bulb Switcher

N 给灯泡。第一次把所有打开。第二次把所有2的倍数的关掉。第三次把3的倍数的打开。最后第n次switch 最后一个。求一共多少个还亮着

要想亮着，必须只switch了奇数次。所以必须是完全平方数。 所以返回int(n\*\*(0.5)) 即可

### 322. Coin Change

这道题给我们了一些可用的硬币值，又给了一个钱数，问我们最小能用几个硬币来找零。仔细一想，这是一个背包问题。 根据题目中的例子可知，不是每次都会给全1,2,5的硬币，有时候没有1分硬币，那么有的钱数就没法找零，需要返回-1。这道题跟CareerCup上的那道[9.8 Represent N Cents 美分的组成](http://www.cnblogs.com/grandyang/p/4840713.html)有些类似，那道题给全了所有的美分，25,10,5,1，然后给我们一个钱数，问我们所有能够找零的方法，而这道题只让我们求出最小的那种，对于求极值问题，我们还是主要考虑动态规划Dynamic Programming来做，我们维护一个一维动态数组dp，其中dp[i]表示钱数为i时的最小硬币数的找零，递推式为：

dp[i] = min(dp[i], dp[i - coins[j]] + 1);

其中coins[j]为第j个硬币，而i - coins[j]为钱数i减去其中一个硬币的值，剩余的钱数在dp数组中找到值，然后加1和当前dp数组中的值做比较，取较小的那个更新dp数组。

### 326. Power of Three

看是不是3的power。

Consider 0 and negative case。 可以用Log或者integer最大的范围来做。注意log的话需要有个epsilon

### 328. Odd Even Linked List

很简单的，用两个指针即可，一个指针p指向当前遍历的奇数节点的最后一个位置，另一个指针q指向待提取的奇数节点的前一个位置。

然后把q.next 的节点删除，插入到p.next的位置即可

### 336. Palindrome Pairs

这道题给我们了许多单词，让我们找出回文对，就是两个单词拼起来是个回文字符串，我最开始尝试的是brute force的方法，每两个单词都拼接起来然后判断是否是回文字符串，但是通过不了OJ，会超时，可能这也是这道题标为Hard的原因之一吧，那么我们只能找别的方法来做，通过学习大神们的解法，发现如下两种方法比较好，其实两种方法的核心思想都一样，写法略有不同而已，那么我们先来看第一种方法吧，要用到哈希表来建立每个单词和其位置的映射，然后需要一个set来保存出现过的单词的长度，算法的思想是，遍历单词集，对于遍历到的单词，我们对其翻转一下，然后在哈希表查找翻转后的字符串是否存在，注意不能和原字符串的坐标位置相同，因为有可能一个单词翻转后和原单词相等，现在我们只是处理了bat和tab的情况，还存在abcd和cba，dcb和abcd这些情况需要考虑，这就是我们为啥需要用set，由于set是自动排序的，我们可以找到当前单词长度在set中的iterator，然后从开头开始遍历set，遍历比当前单词小的长度，比如abcdd翻转后为ddcba，我们发现set中有长度为3的单词，然后我们dd是否为回文串，若是，再看cba是否存在于哈希表，若存在，则说明abcdd和cba是回文对，存入结果中，对于dcb和aabcd这类的情况也是同样处理，我们要在set里找的字符串要在遍历到的字符串的左边和右边分别尝试，看是否是回文对，这样遍历完单词集，就能得到所有的回文对，

### 341. Flatten Nested List Iterator

一个元素可以是list，也可以是integer。要求把nested list flatten 成全integer的list

因为是iterator，所以用stack来做

1. 考虑[[]]第一个元素是空list的情况

2. 第二是考虑第一个元素是三层空list 情况

3. 从前面pop是pop（0）

4. 代码不能重复

### 

### 344. Reverse String

Too easy

### 345. Reverse Vowels of a String

很简单。但是要注意vowel也可以是是大写的。

### 347. Top K Frequent Elements

输入一串数字，给出top k个频次高的elements

解这个题的关键在于控制时间复杂度“小于O(nlogn)”这个条件。

思路一

按照我的思维习惯，看到这个题，首先想到的是下面的思路：先用dict得到所有不同数的个数；再对个数排序，取前k个个数最多的对应的数即可。代码如下，然而使用了内置sorted()函数，只能说时间复杂度小于等于O(nlogn)，不太满足题意的样子，仅供参考。

思路二

在上面思路的基础上，通过改进排序步骤改善时间复杂度。考虑使用时间复杂度只有O(n)的桶排序（bucket sort），同时消耗空间复杂度O(n)。

思路三

还是沿着思路一，除了桶排序，优先队列也是可以满足要求的解法。很多语言都有内建优先队列结构，在Python里有Queue.PriorityQueue，也有更高效的heapq（用list模拟heap），这里使用heapq。【时间复杂度O(nlogk)】

注：由于heapq默认是最小堆，代码中在堆的push时给权重加了负号，这样堆顶部对应的实际上是出现次数最多的数。

### 348. Design Tic-Tac-Toe

我们首先来O(n2)的解法，这种方法的思路很straightforward，就是建立一个nxn大小的board，其中0表示该位置没有棋子，1表示玩家1放的子，2表示玩家2。那么棋盘上每增加一个子，我们都扫描当前行列，对角线，和逆对角线，看看是否有三子相连的情况，有的话则返回对应的玩家，没有则返回0，

Follow up中让我们用更高效的方法，那么根据提示中的，我们建立一个大小为n的一维数组rows和cols，还有变量对角线diag和逆对角线rev\_diag，这种方法的思路是，如果玩家1在第一行某一列放了一个子，那么rows[0]自增1，如果玩家2在第一行某一列放了一个子，则rows[0]自减1，那么只有当rows[0]等于n或者-n的时候，表示第一行的子都是一个玩家放的，则游戏结束返回该玩家即可，其他各行各列，对角线和逆对角线都是这种思路

### **349. Intersection of Two Arrays**

忘了set的遍历 直接用for i in set如果用enumerate，一定要用i，v，否则输出是一个tuple

i是index，不是key！

### 364. Nested List Weight Sum II

这道题是之前那道[Nested List Weight Sum](http://www.cnblogs.com/grandyang/p/5340305.html)的拓展，与其不同的是，这道题的深度越深，权重越小，和之前刚好相反。但是解题思路没有变，还可以用DFS来做，那么由于遍历的时候不知道最终的depth有多深，则不能遍历的时候就直接累加结果，我最开始的想法是在遍历的过程中建立一个二维数组，把每层的数字都保存起来，然后最后知道了depth后，再来计算权重和，比如题目中给的两个例子，建立的二维数组分别为：

[[1,1],2,[1,1]]：

1 1 1 1

2

[1,[4,[6]]]：

1

4

6

这样我们就能算出权重和了

### 371. Sum of Two Integers

不用+，- 求两个数的和

这道题让我们实现两数相加，但是不能用加号或者其他什么数学运算符号，那么我们只能回归计算机运算的本质，位操作Bit Manipulation，我们在做加法运算的时候，每位相加之后可能会有进位Carry产生，然后在下一位计算时需要加上进位一起运算，那么我们能不能将两部分拆开呢，我们来看一个例子759+674

1. 如果我们不考虑进位，可以得到323

2. 如果我们只考虑进位，可以得到1110

3. 我们把上面两个数字假期323+1110=1433就是最终结果了

然后我们进一步分析，如果得到上面的第一第二种情况，我们在二进制下来看，不考虑进位的加，0+0=0， 0+1=1, 1+0=1， 1+1=0，这就是异或的运算规则，如果只考虑进位的加0+0=0, 0+1=0, 1+0=0, 1+1=1，而这其实这就是与的运算，而第三步在将两者相加时，我们再递归调用这个算法，终止条件是当进位为0时，我们直接返回第一步的结果

### 374. Guess Number Higher or Lower

Binary search. Bug free

### 380. Insert Delete GetRandom O(1)

这道题让我们在常数时间范围内实现插入删除和获得随机数操作，如果这道题没有常数时间的限制，那么将会是一道非常简单的题，我们直接用一个set就可以搞定所有的操作。但是由于时间的限制，我们无法在常数时间内实现获取随机数，所以只能另辟蹊径。此题的正确解法是利用到了一个一维数组和一个哈希表，其中数组用来保存数字，哈希表用来建立每个数字和其在数组中的位置之间的映射，对于插入操作，我们先看这个数字是否已经在哈希表中存在，如果存在的话直接返回false，不存在的话，我们将其插入到数组的末尾，然后建立数字和其位置的映射。删除操作是比较tricky的，我们还是要先判断其是否在哈希表里，如果没有，直接返回false。由于哈希表的删除是常数时间的，而数组并不是，为了使数组删除也能常数级，我们实际上将要删除的数字和数组的最后一个数字调换个位置，然后修改对应的哈希表中的值，这样我们只需要删除数组的最后一个元素即可，保证了常数时间内的删除。而返回随机数对于数组来说就很简单了，我们只要随机生成一个位置，返回该位置上的数字即可

### 

### 383. Ransom Note

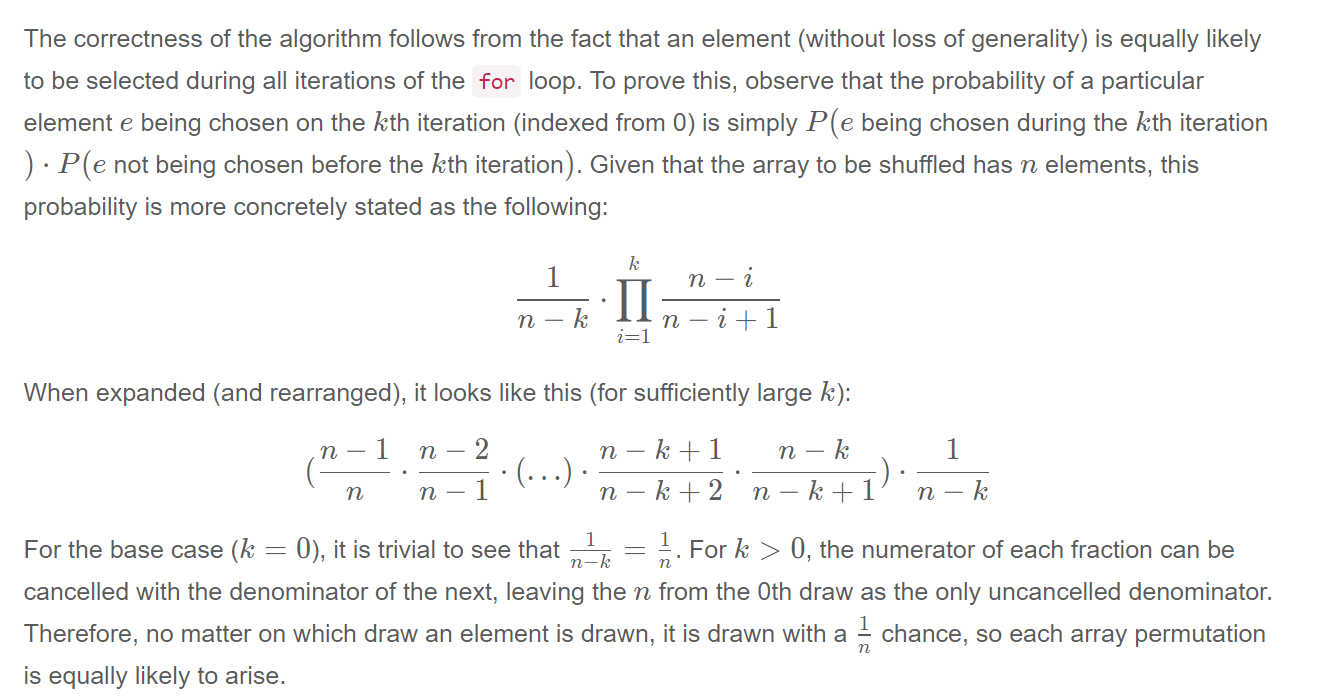
Very easy. Use Ord

### 384. Shuffle an Array

要求shuffle一个array，使得排列顺序完全随机。

每次随机从array里面抽一个，慢慢组成新的array。需要证明

要用random.randrange(len(aux))



### 387. First Unique Character in a String

Hashtable. Very easy. Use enumerate and ord

### 402. Remove K Digits

Given a non-negative integer *num* represented as a string, remove *k* digits from the number so that the new number is the smallest possible.

输入用string表示的非负整数num， 删去k个digit，使得新的数字最小。

可以借鉴其中的思路，如果n是num的长度，我们要去除k个，那么需要剩下n-k个，我们开始遍历给定数字num的每一位，对于当前遍历到的数字c，进行如下while循环，如果res不为空，且k大于0，且res的最后一位大于c，那么我们应该将res的最后一位移去，且k自减1。当跳出while循环后，我们将c加入res中，最后我们将res的大小重设为n-k。根据题目中的描述，可能会出现"0200"这样不符合要求的情况，所以我们用一个while循环来去掉前面的所有0，然后返回时判断是否为空，为空则返回“0”，

如果这样得到的是一个空串那就手动返回０．还有一个需要注意的是字符串首字符为0的话需要去掉0.

### 403. Frog Jump

终于等到青蛙过河问题了，一颗赛艇。题目中说青蛙如果上一次跳了k距离，那么下一次只能跳k-1, k, 或k+1的距离，那么青蛙跳到某个石头上可能有多种跳法，由于这道题只是让我们判断青蛙是否能跳到最后一个石头上，并没有让我们返回所有的路径，这样就降低了一些难度。我们可以用递归来做，我们维护一个哈希表，建立青蛙在pos位置和拥有jump跳跃能力时是否能跳到对岸。为了能用一个变量同时表示pos和jump，我们可以将jump左移很多位并或上pos，由于题目中对于位置大小有限制，所以不会产生冲突。我们还是首先判断pos是否已经到最后一个石头了，是的话直接返回true；然后看当前这种情况是否已经出现在哈希表中，是的话直接从哈希表中取结果。如果没有，我们就遍历余下的所有石头，对于遍历到的石头，我们计算到当前石头的距离dist，如果距离小于jump-1，我们接着遍历下一块石头；如果dist大于jump+1，说明无法跳到下一块石头，m[key]赋值为false，并返回false；如果在青蛙能跳到的范围中，我们调用递归函数，以新位置i为pos，距离dist为jump，如果返回true了，我们给m[key]赋值为true，并返回true。如果结束遍历我们给m[key]赋值为false，并返回false，

### 406. Queue Reconstruction by Height

How to srot list of list:

people = sorted(people, key=lambda x: x[1])

按照多维排序：

pass1 = sorted(people, key=lambda x: (x[0], -x[1]), reverse=True)。 先按第一个参数，如果一样就按第二个。

最后插入按照x[1]一次插入就可以

python的list，如果list.insert(pos,val)可以在那个index插入，然后把后面的全部往后移动一位

### 412. Fizz Buzz

Bug free

### 442. Find All Duplicates in an Array

Bug free

### 445. Add Two Numbers II

这道题是之前那道[Add Two Numbers](http://www.cnblogs.com/grandyang/p/4129891.html)的拓展，我们可以看到这道题的最高位在链表首位置，如果我们给链表翻转一下的话就跟之前的题目一样了，这里我们来看一些不修改链表顺序的方法。由于加法需要从最低位开始运算，而最低位在链表末尾，链表只能从前往后遍历，没法取到前面的元素，那怎么办呢？我们可以利用栈来保存所有的元素，然后利用栈的后进先出的特点就可以从后往前取数字了，我们首先遍历两个链表，将所有数字分别压入两个栈s1和s2中，我们建立一个值为0的res节点，然后开始循环，如果栈不为空，则将栈顶数字加入sum中，然后将res节点值赋为sum%10，然后新建一个进位节点head，赋值为sum/10，如果没有进位，那么就是0，然后我们head后面连上res，将res指向head，这样循环退出后，我们只要看res的值是否为0，为0返回res->next，不为0则返回res即可

### 448. Find All Numbers Disappeared in an Array

Very easy. abs(n), use negative etc

### 453. Minimum Moves to Equal Array Elements

Bug free

### 457. Circular Array Loop

### 非高频medium

### 458. Poor Pigs

1000个桶，一个有毒。 60分钟，每个猪在15分钟会死。 要找出那个有毒的。问最少需要几个猪。

正交试验法。每个猪可以做四次试验，加上剩下的没有试过的，一共相当于五次试验。那么正交来算，n个猪就是5\*\*n次方。 5^n>=B n>=logB/log5

最后注意返回的要转换成int，ceil返回的是float

### 461. Hamming Distance

求两个数字的hamming， 第一是用x^y

第二知道一个数字，python里面count binary的1的个数用：

bin(x).count("1")

### **463. Island Perimeter**

给出一个2D数组。1表示island，0表示水。如果在边上也算是perimeter。求perimeter

注意不要溢出数组。此外用双循环时候，判断不要溢出的条件的时候j==len(grid[0])-1 不要忘了0

O（n）复杂度。

### 476. Number Complement

设个i为1，然后当i都小于num就不停左移并且把个位设为1

然后返回i^num

### **479. Largest Palindrome Product**

输入n作为位数。 然后输出两个n位数的乘积里面最大的是回文数的那个。

这题很难，不是easy！ 需要构造回文数，然后再检验是不是被数字整除。 重做。

### 500. Keyboard Row

save original case, not lower case

### 518. Coin Change 2

### **非高频medium**

### **535. Encode and Decode TinyURL**

注意python class的写法

Constructor是 def \_\_init\_\_(self)

引用自己class的variable是 self.abc

### 540. Single Element in a Sorted Array

看见logn用二分

两个易错点，第一个是，不能简单盘低mid==mid-1 or mid==mid+1. 必须要结合mid=mid-1 and mid-left%2==0 这样看

第二个易错点，移动的时候，如果往右移动，而mid==mid+1， 那么要移动两个。否则的话长度计算就不对了

### 543. Diameter of Binary Tree

给定一颗二叉树，寻找这棵树的直径，直径定义为这棵树中任意两个节点的路径长度最大值。如上面的给定的那棵树，明显最长路径是3，也即4 -> 2 -> 1 -> 3。当然有一点要注意，最长路径可能不经过根节点。

思路

这道题是[104. Maximum Depth of Binary Tree](https://leetcode.com/problems/maximum-depth-of-binary-tree/#/description)的运用题目。刚拿到题目我一开始是不敢做的， 因为潜意识里认为这题很难。但是看了一下题目居然是Easy难度，仔细思考了一下发现难度其实不大，主要是考查树的遍历。最长路径有两种：

其一，这条路径经过了根节点，那么只需要找出根节点下的两棵子树的深度然后相加即可。

其二，如果这条路径没有经过根节点，那么只需要找出根节点左子树内或者根节点右子树内的最深度即可。自底向上，递归查找子树的深度，如果发现左子树深度与右子树深度之和大于当前纪录的直径，那么进行相应的值记录，然后返回左子树和右子树深度较大的那个来进行下一轮的比对。递归完成之后即可找出直径。

整个树进行了一次递归调用，因此时间复杂度是O(n)，n是树的节点数，同时，只使用了一个数来记录直径，因此空间复杂度是O(1)

### 547. Friend Circles

用union find做

### 556. Next Greater Element III

这道题给了我们一个数字，让我们对各个位数重新排序，求出刚好比给定数字大的一种排序，如果不存在就返回-1。这道题给的例子的数字都比较简单，我们来看一个复杂的，比如12443322，这个数字的重排序结果应该为13222344，如果我们仔细观察的话会发现数字变大的原因是左数第二位的2变成了3，细心的童鞋会更进一步的发现后面的数字由降序变为了升序，这也不难理解，因为我们要求刚好比给定数字大的排序方式。那么我们再观察下原数字，看看2是怎么确定的，我们发现，如果从后往前看的话，2是第一个小于其右边位数的数字，因为如果是个纯降序排列的数字，做任何改变都不会使数字变大，直接返回-1。知道了找出转折点的方法，再来看如何确定2和谁交换，这里2并没有跟4换位，而是跟3换了，那么如何确定的3？其实也是从后往前遍历，找到第一个大于2的数字交换，然后把转折点之后的数字按升序排列就是最终的结果了。最后记得为防止越界要转为长整数型，然后根据结果判断是否要返回-1即可

### 557. Reverse Words in a String III

Bug free

### 560. Subarray Sum Equals K

本题是为了求连续数组和为k的个数。本题需要注意的是可能存在负数。首先第一种方法暴力搜索法，两个循环，但是由于负数的存在不能在碰到一个之后就break

另外一种思路就是使用map来保存当前数组的求和值，然后每到一个数都将res进行更新

### 561. Array Partition I

2n个数字分成n pair. 每pair里面取小的，让这些小的总和最大。

这道题让我们分割数组，两两一对，让每对中较小的数的和最大。这题难度不大，用贪婪算法就可以了。由于我们要最大化每对中的较小值之和，那么肯定是每对中两个数字大小越接近越好，因为如果差距过大，而我们只取较小的数字，那么大数字就浪费掉了。明白了这一点，我们只需要给数组排个序，然后按顺序的每两个就是一对，我们取出每对中的第一个数即为较小值累加起来即可，

### 564. Find the Closest Palindrome

这道题给了我们一个数字，让我们找到其最近的回文数，而且说明了这个最近的回文数不能是其本身。比如如果给你个131，那么就需要返回121。而且返回的回文数可能位数还不同，比如当n为100的时候，我们就应该返回99，或者给了我们99时，需要返回101。那么实际上最近回文数是有范围的，比如说n为三位数，那么其最近回文数的范围在[99, 1001]之间，这样我们就可以根据给定数字的位数来确定出两个边界值，要和其他生成的回文数进行比较，取绝对差最小的。

下面我们来看如何求一般情况下的最近回文数，我们知道回文数就是左半边和右半边互为翻转，奇数情况下中间还有个单独的值。那么如何将一个不是回文数的数变成回文数呢，我们有两种选择，要么改变左半边，要么改变右半边。由于我们希望和原数绝对差最小，肯定是改变低位上的数比较好，所以我们改变右半边，那么改变的情况又分为两种，一种是原数本来就是回文数，这种情况下，我们需要改变中间的那个数字，要么增加1，要么减小1，比如121，可以变成111和131。另一种情况是原数不是回文数，我们只需要改变右半边就行了，比如123，变成121。那么其实这三种情况可以总结起来，分别相当于对中间的2进行了-1, +1, +0操作，那么我们就可以用一个-1到1的for循环一起处理了，先取出包括中间数的左半边，比如123就取出12，1234也取出12，然后就要根据左半边生成右半边，为了同时处理奇数和偶数的情况，我们使用一个小tricky，在反转复制左半边的时候，我们给rbegin()加上len&1，当奇数时，len&1为1，这样就不会复制中间数了；为偶数时，len&1为0，这就整个翻转复制了左半边。我们把每次生成的回文串转为转为数字后加入到一个集合set中，把之前的两个边界值也同样加进去，最后我们在五个candidates中找出和原数绝对差最小的那个返回，记得别忘了在集合中删除原数，因为如果原数时回文的话, i=0时就把自己也加入集合了，

### 609. Find Duplicate File in System

**Follow-up beyond contest:**

1. Imagine you are given a real file system, how will you search files? DFS or BFS?
2. If the file content is very large (GB level), how will you modify your solution?
3. If you can only read the file by 1kb each time, how will you modify your solution?
4. What is the time complexity of your modified solution? What is the most time-consuming part and memory consuming part of it? How to optimize?
5. How to make sure the duplicated files you find are not false positive?

算法上的难度不大，我觉得主要在于如何很好地回答follow up问题。所以我首先简单地总结一下思路，然后重点探讨如何回答follow up（自己的回答不一定准确，仅供参考）。

算法上关键还是在于建立一个哈希表，建立从fn\_content到文件全路径（folder\_path + "/" + fn）数组之间的映射。对于paths中的每一个string，我们用istringstream进行解析，其第一个string就是folder\_name，之后的每个string就是该文件夹下面的一个fn + fn\_content。解析出来之后，再进一步解析fn和fn\_content，然后将对应数据加入哈希表中。最后遍历一遍哈希表，找出同样内容出现在多个文件中的项目，加入结果集中即可。

关于follow-up：

1）如果是一个真实的文件系统，那么其实DFS和BFS搜索都是可以的。不同点在于用DFS搜索时，如果文件深度太深，那么可能会需要占用较大的内存栈空间；用BFS时，如果每个文件夹下面的文件夹数量太多，则队列可能会比较长，因此也会占用较大的内存空间。

2）如果文件内容过大，那么显然就不能将文件内容直接作为哈希表的key了。我觉得有几种可能的改进方法：a）用MD5算法（或者其它类似的算法）生成文件内容对应的key，然后用这个key作为哈希表的key；2）采用多次哈希的方法，例如第一次用fn\_content的长度来作为key，第二次处理的时候，在长度相同的文件列表中，再用MD5算法判断文件内容是不是相同。这样做的好处在于第一次哈希的时候，就排除了大量内容大小不同的文件，从而使得需要用MD5算法的文件数目大大减少，从而提高效率。

3）那就每读进来1kb的数据，做一次MD5吧，最后把所有的MD5结果生成一个向量，或者再次MD5，用最终结果来作为哈希表的key。

4）如果认为平均每个文件夹下面有常数个文件（夹），那么修改后的算法的时间复杂度就是O(file\_counts \* avg\_file\_length)，其中file\_counts表示文件系统中所有的文件个数；avg\_file\_length表示平均文件长度。最耗时的有可能是两部分：a）遍历文件的部分（如果文件数量特别大）；b）读取每个文件内容的部分（如果每个文件特别大）。要优化，那么除了采用3）中的方法之外，还可以采用并行化方法来处理。例如遍历文件的时候，采用多线程或者map reduce的思路进行并行，最后再合成所有的结果。读取每个文件内容的时候，也可以采用多线程（需要硬盘支持了，有这样的硬盘吗？）。

5）再牛逼的哈希算法也不能保证完全消除冲突，那么为了make sure，就只好最后对结果采用最笨的方法进行double check了。。。

### 617. Merge Two Binary Trees

这道题给了我们两个二叉树，让我们合并成一个，规则是，都存在的结点，就将结点值加起来，否则空的位置就由另一个树的结点来代替。那么根据过往经验，处理二叉树问题的神器就是递归，那么我们来看递归函数如何去写。我们首先判断，如果t1不存在，则直接返回t2，反之，如果t2不存在，则直接返回t1。如果上面两种情况都不满足，那么以t1和t2的结点值之和建立新结点t，然后对t1和t2的左子结点调用递归并赋给t的左子结点，再对t1和t2的右子结点调用递归并赋给t的右子结点，返回t结点即可

### 621. Task Scheduler

### **非高频medium**

### **628. Maximum Product of Three Numbers**

输入一个数组，输出三个数乘积最大的。可以有负数

从小到大sort

最大数字必然是max（num[-1]\*num[-2]\*num[-3], num[0]\*num[1]\*num[-1])

Bug free

### 636. Exclusive Time of Functions

给定一组函数调用日志，格式为"function\_id:start\_or\_end:timestamp"，函数可以递归。  
  
求每个函数调用的总时长

栈（Stack）

栈中保存元素格式为函数ID，时间戳 [fid, tmp]  
  
当日志为'start'时：  
  
 若栈非空，记栈顶元素为topFid, topTmp；将fid的时间累加tmp - topTmp；  
  
 将[fid, tmp]压栈  
  
否则：  
  
 将栈顶元素的时间累加tmp - topTmp + 1  
  
 弹出栈顶元素  
  
 若栈非空，将栈顶元素topTmp更新为tmp + 1

### 642. Design Search Autocomplete System

设计搜索自动完成系统

包含如下两个方法：

**构造方法：**

AutocompleteSystem(String[] sentences, int[] times): 输入句子sentences，及其出现次数times

**输入方法：**

List<String> input(char c): 输入字符c可以是26个小写英文字母，也可以是空格，以'#'结尾。返回输入字符前缀对应频率最高的至多3个句子，频率相等时按字典序排列。

解题思路：

可以用trie树来解决这个问题。

由于要返回前3个搜索次数最多的句子，我们可以用priority\_queue来存储所返回的所有的句子和它的次数的键值对。

首先构造trie tree，主要为trieNode的结构以及insert 方法。

构造完trieNode类后， 这个系统实际上主要为一个巨大的trietree，我们需要一个树的根节点。

由于我们每次都要输入一个字符，我们可以用一个私有的Node：curNode来追踪当前我们节点。

curNode初始化为root，在每次输入完一个句子时，即输入的字符为‘#’时，我们需要将其置为root

同时需要一个string类型stn来表示当前的搜索的句子。

需要注意的是我们priority\_queue中存储的为pair<string,int>我们需要给它重写比较器。

所以我们每输入一个字符，首先检查是不是结尾标识“#”，如果是的话，将当前句子加入trie树，重置相关变量，返回空数组。

如不是，检查当前TrieNode对应的child是否含有c的对应节点。如果没有，将curNode置为NULL并且返回空数组。

若存在，将curNode 更新为c对应的节点，并且对curNode进行dfs。

dfs时，我们首先检查当前是不是一个完整的句子，如果是，将句子与其次数同时加入priority\_queue中，然后对其child中可能存在的子节点进行dfs。

进行完dfs后，我们需要取出前三个，需要注意的是，可能可选择的结果不满3个，所以要在while中多加入检测q为空的条件语句。

最后要将q中的所有元素都弹出。

### 647. Palindromic Substrings

动态规划。

这个回文字符串的题母，千篇一律啊。在[5. Longest Palindromic Substring](http://blog.csdn.net/fuxuemingzhu/article/details/79573621)中，当遇到回文字符串时，我们是使用maxL记录下来最长的那个。稍微修改，这个题我们只要做统计个数就可以。因此代码基本一样。

### 654. Maximum Binary Tree

调用自身递归做子树的时候，输入的要是nums[:maxi], 而不是nums[:i] !!!

### 657. Judge Route Circle

Bug free

### 669. Trim a Binary Search Tree

不要忘记包括L R 本身。 此题简单。

### 681. Next Closest Time

这道题给了我们一个时间点，让我们求最近的下一个时间点，规定了不能产生新的数字，当下个时间点超过零点时，就当第二天的时间。为了找到下一个时间点，我们肯定是从分钟开始换数字，而且换的数字要是存在的数字，那么我们最先要做的就是统计当前时间点中的数字，由于可能有重复数字的存在，我们把数字都存入集合set中，这样可以去除重复数字，并且可以排序，然后再转为vector。下面就从低位分钟开始换数字了，如果低位分钟上的数字已经是最大的数字了，那么说明要转过一轮了，就要把低位分钟上的数字换成最小的那个数字；否则就将低位分钟上的数字换成下一个数字。然后再看高位分钟上的数字，同理，如果高位分钟上的数字已经是最大的数字，或则下一个数字大于5，那么直接换成最小值；否则就将高位分钟上的数字换成下一个数字。对于小时位上的数字也是同理，对于小时低位上的数字情况比较复杂，当小时高位不为2的时候，低位可以是任意数字，而当高位为2时，低位需要小于等于3。对于小时高位，其必须要小于等于2

### 683. K Empty Slots

这道题给了我们这样一个场景，说是花园里有N个空槽，可以放花，每天放一朵开着的花，而且一旦放了就会一直开下去。不是按顺序放花，而是给了我们一个数组flowers，其中flowers[i] = x表示第i天放的花会在位置x。其实题目这里有误，数组是从0开始的，而天数和位置都是从1开始的，所以正确的应该是第i+1天放的花会在位置x。然后给了我们一个整数k，让我们判断是否正好有两朵盛开的花中间有k个空槽，如果有，返回当前天数，否则返回-1。博主刚开始想的是先用暴力破解来做，用一个状态数组，如果该位置有花为1，无花为0，然后每增加一朵花，就遍历一下状态数组，找有没有连续k个0，结果TLE了。这说明，应该等所有花都放好了，再来找才行，但是这样仅用0和1的状态数组是不行的，我们得换个形式。

我们用一个days数组，其中days[i] = t表示在i+1位置上会在第t天放上花，那么如果days数组为[1 3 2]，就表示第一个位置会在第一天放上花，第二个位置在第三天放上花，第三个位置在第二天放上花。我们想，在之前的状态数组中，0表示没放花，1表示放了花，而days数组中的数字表示放花的天数，那么就是说数字大的就是花放的时间晚，那么在当前时间i，所有大于i的是不是也就是可以看作是没放花呢，这样问题就迎刃而解了，我们来找一个k+2大小的子数组，除了首尾两个数字，中间的k个数字都要大于首尾两个数字即可，那么首尾两个数字中较大的数就是当前的天数。left和right是这个大小为k+2的窗口，初始化时left为0，right为k+1，然后i从0开始遍历，这里循环的条件时right小于n，当窗口的右边界越界后，循环自然需要停止。如果当days[i]小于days[left]，或者days[i]小于等于days[right]的时候，有两种情况，一种是i在[left, right]范围内，说明窗口中有数字小于边界数字，这不满足我们之前限定的条件，至于days[i]为何可以等于days[right]，是因为当i遍历到right到位置时，说明中间的数字都是大于左右边界数的，此时我们要用左右边界中较大的那个数字更新结果res。不管i是否等于right，只要进了这个if条件，说明当前窗口要么是不合题意，要么是遍历完了，我们此时要重新给left和right赋值，其中left赋值为i，right赋值为k+1+i，还是大小为k+2的窗口，继续检测。最后我们看结果res，如果还是INT\_MAX，说明无法找到，返回-1即可，

### 695. Max Area of Island

DFS 和number of island 那题的方法一样。走过的1用要给新set保存或者把原来的改成#。 然后每次遍历记录最大面积。

### 697. Degree of an Array

**这题不要录音。**

**输入一个非负int组成的array。 degree表示里面出现频次最大的元素的频次。 要求输出最短的包括同意频度的subarray**

**要用left，right指针。然后同时记录left，right，freq。然后往右走。**

### 716. Max Stack

这道题让我们实现一个最大栈，包含一般栈的功能，但是还新加了两个功能peekMax()和popMax()，随时随地可以查看和返回最大值。之前有一道很类似的题[Min Stack](http://www.cnblogs.com/grandyang/p/4091064.html)，所以我们可以借鉴那道题的解法，使用两个栈来模拟，s1为普通的栈，用来保存所有的数字，而s2为最大栈，用来保存出现的最大的数字。

在push()函数中，我们先来看s2，如果s2为空，或者s2的栈顶元素小于等于x，将x压入s2中。因为s2保存的是目前为止最大的数字，所以一旦新数字大于等于栈顶元素，说明遇到更大的数字了，压入栈。然后将数组压入s1，s1保存所有的数字，所以都得压入栈。

在pop()函数中，当s2的栈顶元素和s1的栈顶元素相同时，我们要移除s2的栈顶元素，因为一个数字不在s1中了，就不能在s2中。然后取出s1的栈顶元素，并移除s1，返回即可。

在top()函数中，直接返回s1的top()函数即可。

在peekMax()函数中，直接返回s2的top()函数即可。

在popMax()函数中，先将s2的栈顶元素保存到一个变量mx中，然后我们要在s1中删除这个元素，由于栈无法直接定位元素，所以我们用一个临时栈t，将s1的出栈元素保存到临时栈t中，当s1的栈顶元素和s2的栈顶元素相同时退出while循环，此时我们在s1中找到了s2的栈顶元素，分别将s1和s2的栈顶元素移除，然后要做的是将临时栈t中的元素加回s1中，注意此时容易犯的一个错误是，没有同时更新s2，所以我们直接调用push()函数即可，

### 717. 1-bit and 2-bit Characters

给定数组中仅有0和1两种元素，且出现的元素组合只能有三种编码方式：10、11、0，判断给定的数组最后一个元素是否属于0编码，而不是10编码。

读清楚题目。

* 明白题目意图，就会发现，题目的意思是要判断最后一个0元素是属于0还是输入10；
* 遍历数组，给定指针，若当前位为1则指针+2；若当前位为0，则指针+1；
* 判断最后指针是否与bits.length-1相等，相等则为真，否则为假；其中length=1的情况也包括进去了。

### 722. Remove Comments

我们需要逐行解析源代码。 我们的状态是，我们要么在块中评论，要么不是。

如果我们开始一个块注释而我们不在一个块中，那么我们将跳过接下来的两个字符并将我们的状态更改为在注释块中。

如果我们结束一个块注释并且我们在一个块中，那么我们将跳过接下来的两个字符并将我们的状态更改为不在注释块中。

如果我们开始一个行注释而我们不在一个块中，那么我们将忽略该行的其余部分。

如果我们不在块注释中（并且它不是注释的开头），我们将记录我们所处的字符。

在每一行的末尾，如果我们不在一个块中，我们将记录该行。

### **728. Self Dividing Numbers**

求能被自身每一位数字整除的number

注意不能包括0. 因为mod 0不合法

### 739. Daily Temperatures

有一系列天的温度。求每一天到之后更高的一个温度需要等待的天数。如果之后没有更高的就返回0. 输出一个array，里面是每一天需要等的天数。 是next greater element 系列。 从最右边往左边，搞一个stack，每次放tuple，温度和天的index。然后往左移动。如果坐标小于stack里面的，就往里面填。 如果突然哪一天比之后的高了，就从stack里面依次往外拿，知道stack top的比这天高。 然后把这天的放进stack底部并计算差几天。因为类似3,2,1,5 这样的，在3之前的天，如果比3大那么肯定也比2，1大，所以不需要存2，1. Stack里面应该只存一个比一个大的天。 这就是next greater element的精髓。

### 740. Delete and Earn

给定整数数组nums，执行如下操作：

挑选任意数字nums[i]，得到nums[i]分，同时需要删除所有等于nums[i] - 1和nums[i] + 1的整数。

求最大得分。

动态规划（Dynamic Programming）

dp[x]表示删除不大于x的所有数字的最大得分。  
  
cnt[x]存储数字x的个数。

状态转移方程：

dp[x] = max(dp[x - 1], dp[x - 2] + cnt[x] \* x)

### 760. Find Anagram Mappings

Bug free

### 772. Basic Calculator III

这道题是基本计算器系列的第三道，前两道分别为[Basic Calculator](http://www.cnblogs.com/grandyang/p/4570699.html)和[Basic Calculator II](http://www.cnblogs.com/grandyang/p/4601208.html)，区别是，第一道只有加减法跟括号，第二道只有加减乘除法，而这第三道既有加减乘除法又有括号运算。其实做过前两道题的话，那么这道题也就没什么问题，因为把前两道题的解法综合一下就是这道题的解法啦。由于此题既有括号，又有乘除法，我们知道括号是优先级最高的，但是好就好在我们可以将括号里的内容当作一个整体调用递归函数来处理。而其他部分，就跟第二道一模一样了。我们还是分情况来处理遍历，我们需要几个变量，num表示当前的数字，curRes表示当前的结果，res为最终的结果，op为操作符号，初始化为'+'。当遇到数字的时候，我们将num自乘以10并加上这个数字，这是由于可能遇到多位数，所以每次要乘以10。当遇到括号的时候，这里就有一个小trick，由于表示可能会有括号嵌套括号，所以我们如果搜索右括号的话，就有可能使得括号没有正确的匹配上，所以我们用一个变量cnt，遇到左括号自增1，遇到右括号自减1，当cnt为0的时候，说明括号正好完全匹配，这个trick在验证括号是否valid的时候经常使用到。然后我们就是根据左右括号的位置提取出中间的子字符串调用递归函数，返回值赋给num。如果遇到符号，或者是最后一个位置的字符时，我们根据op的值对num进行分别的加减乘除的处理，结果保存到curRes中。然后再次判读如果op是加或减，或者是最后一个位置的字符时，将curRes加到结果res中，并且curRes重置为0。最后将当前字符c赋值给op（注意这里只有当时最后一个位置的字符时，才有可能不是运算符号，不过也不要紧了，因为遍历已经结束了），num也要重置为0

### 798. Smallest Rotation with Highest Score

我们假设A的大小为n，如果采用暴力法逐个测试，则时间复杂度为O(n^2)，应该过不了大数据测试。

我采取的方法是：首先计算使得每个元素A[i]要符合条件，需要rotate的K的集合，用线段表示；然后再扫描一遍，求出这些线段中重合最多的点，那么这个点对应的rotate次数就是题目所要计算的K。对应A[i]来讲，如果A[i] <= i，那么它向左移动到j也可能维持A[i] <= j，所以我们计算出此时它向左移动的合法区间[0, i - A[i]]。那么A[i]向右移动的合法区间是多少呢？我们知道它向右移动最多移动到n - 1，即移动n - 1 - i步；而最少需要移动max(1, A[i] - i)步，其中1表示A[i] <= 的情况。那么如果这个区间合法，就可以同样构成了一个合法的移动区间[i + 1, n - max(1, A[i] - i)]。

得到多个线段构成的合法移动区间之后，我们的任务就是求出这些区间的最大重合点。首先对segment中的各个点进行排序，然后采用扫描线的方法计算最大最大重合处。为了便于区分某个点是起点还是终点，我们定义一个pair<int, bool>来表示点，并且让起点的bool值为false，终点的bool值为true，这样就可以在扫描到某个点之后，先处理起点，再处理终点。

由于每个A[i]最多对应2个合法移动区间，所以segments大小也是O(n)量级的。这样可以得知，本算法的时间复杂度是O(nlogn)，空间复杂度是O(n)。

### 804. Unique Morse Code Words

找出一组字符串进行莫尔斯电码的编码有多少种不同情况。

找出多少种不同的情况，完全可以用len(set())的方式进行处理。所以，先得到每个字符的莫尔斯电码，然后把字符串所有进行拼接。很简单了哈

char can't minus each other in python

### 807. Max Increase to Keep City Skyline

问的是increase多少，不是综合是increase多少

### 867 transpose matrix

创建空list必须要如下

B = [[0 for \_ in range(len(A))] for \_ in range(len(A[0]))]

不能用[0]\*10!!! 因为全指向同一个array！

### 876. Middle of the Linked List

**求一个 linkedlist的midpoint**

**快慢指针，快的走2步，慢的走一步**

**注意快的走两步的话必须要检查while fast and fast.next**

**同时开始走的时候，试一下应该从pre走会如何。**

### 877. Stone Game

有一堆正整数piles

然后A和B从里面依次拿数字。A，B只能从两边拿。假设AB都用最优策略，返回A胜还是B胜

这题要用动态规划。

### 878. Nth Magical Number

**From fraction import gcd 不用自己写gcd function**

**推到一个数学公式出来**

# Appendix

## A summary: how to use bit manipulation to solve problems easily and efficiently

### **Wiki**

**Bit manipulation is the act of algorithmically manipulating bits or other pieces of data shorter than a word. Computer programming tasks that require bit manipulation include low-level device control, error detection and correction algorithms, data compression, encryption algorithms, and optimization. For most other tasks, modern programming languages allow the programmer to work directly with abstractions instead of bits that represent those abstractions. Source code that does bit manipulation makes use of the bitwise operations: AND, OR, XOR, NOT, and bit shifts.**

**Bit manipulation, in some cases, can obviate or reduce the need to loop over a data structure and can give many-fold speed ups, as bit manipulations are processed in parallel, but the code can become more difficult to write and maintain.**

### **Details**

#### **Basics**

**At the heart of bit manipulation are the bit-wise operators & (and), | (or), ~ (not) and ^ (exclusive-or, xor) and shift operators a << b and a >> b.**

**There is no boolean operator counterpart to bitwise exclusive-or, but there is a simple explanation. The exclusive-or operation takes two inputs and returns a 1 if either one or the other of the inputs is a 1, but not if both are. That is, if both inputs are 1 or both inputs are 0, it returns 0. Bitwise exclusive-or, with the operator of a caret, ^, performs the exclusive-or operation on each pair of bits. Exclusive-or is commonly abbreviated XOR.**

* **Set union A | B**
* **Set intersection A & B**
* **Set subtraction A & ~B**
* **Set negation ALL\_BITS ^ A or ~A**
* **Set bit A |= 1 << bit**
* **Clear bit A &= ~(1 << bit)**
* **Test bit (A & 1 << bit) != 0**
* **Extract last bit A&-A or A&~(A-1) or x^(x&(x-1))**
* **Remove last bit A&(A-1)**
* **Get all 1-bits ~0**

#### **Examples**

**Count the number of ones in the binary representation of the given number**

**int count\_one(int n) {  
 while(n) {  
 n = n&(n-1);  
 count++;  
 }  
 return count;  
}**

**Is power of four (actually map-checking, iterative and recursive methods can do the same)**

**bool isPowerOfFour(int n) {  
 return !(n&(n-1)) && (n&0x55555555);  
 //check the 1-bit location;  
}**

#### **^ tricks**

**Use ^ to remove even exactly same numbers and save the odd, or save the distinct bits and remove the same.**

##### **Sum of Two Integers**

**Use ^ and & to add two integers**

**int getSum(int a, int b) {  
 return b==0? a:getSum(a^b, (a&b)<<1); //be careful about the terminating condition;  
}**

##### **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. (Of course, you can do this by math.)**

**int missingNumber(vector<int>& nums) {  
 int ret = 0;  
 for(int i = 0; i < nums.size(); ++i) {  
 ret ^= i;  
 ret ^= nums[i];  
 }  
 return ret^=nums.size();  
}**

#### **| tricks**

**Keep as many 1-bits as possible**

**Find the largest power of 2 (most significant bit in binary form), which is less than or equal to the given number N.**

**long largest\_power(long N) {  
 //changing all right side bits to 1.  
 N = N | (N>>1);  
 N = N | (N>>2);  
 N = N | (N>>4);  
 N = N | (N>>8);  
 N = N | (N>>16);  
 return (N+1)>>1;  
}**

##### **Reverse Bits**

**Reverse bits of a given 32 bits unsigned integer.**

###### **Solution**

**uint32\_t reverseBits(uint32\_t n) {  
 unsigned int mask = 1<<31, res = 0;  
 for(int i = 0; i < 32; ++i) {  
 if(n & 1) res |= mask;  
 mask >>= 1;  
 n >>= 1;  
 }  
 return res;  
}**

**uint32\_t reverseBits(uint32\_t n) {  
 uint32\_t mask = 1, ret = 0;  
 for(int i = 0; i < 32; ++i){  
 ret <<= 1;  
 if(mask & n) ret |= 1;  
 mask <<= 1;  
 }  
 return ret;  
}**

#### **& tricks**

**Just selecting certain bits**

**Reversing the bits in integer**

**x = ((x & 0xaaaaaaaa) >> 1) | ((x & 0x55555555) << 1);  
x = ((x & 0xcccccccc) >> 2) | ((x & 0x33333333) << 2);  
x = ((x & 0xf0f0f0f0) >> 4) | ((x & 0x0f0f0f0f) << 4);  
x = ((x & 0xff00ff00) >> 8) | ((x & 0x00ff00ff) << 8);  
x = ((x & 0xffff0000) >> 16) | ((x & 0x0000ffff) << 16);**

##### **Bitwise AND of Numbers Range**

**Given a range [m, n] where 0 <= m <= n <= 2147483647, return the bitwise AND of all numbers in this range, inclusive. For example, given the range [5, 7], you should return 4.**

###### **Solution**

**int rangeBitwiseAnd(int m, int n) {  
 int a = 0;  
 while(m != n) {  
 m >>= 1;  
 n >>= 1;  
 a++;  
 }  
 return m<<a;   
}**

##### **Number of 1 Bits**

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

###### **Solution**

**int hammingWeight(uint32\_t n) {  
 int count = 0;  
 while(n) {  
 n = n&(n-1);  
 count++;  
 }  
 return count;  
}**

**int hammingWeight(uint32\_t n) {  
 ulong mask = 1;  
 int count = 0;  
 for(int i = 0; i < 32; ++i){ //31 will not do, delicate;  
 if(mask & n) count++;  
 mask <<= 1;  
 }  
 return count;  
}**

#### **Application**

##### **Repeated DNA Sequences**

**All DNA is composed of a series of nucleotides abbreviated as A, C, G, and T, for example: "ACGAATTCCG". When studying DNA, it is sometimes useful to identify repeated sequences within the DNA. Write a function to find all the 10-letter-long sequences (substrings) that occur more than once in a DNA molecule.**

**For example,**

**Given s = "AAAAACCCCCAAAAACCCCCCAAAAAGGGTTT",**

**Return: ["AAAAACCCCC", "CCCCCAAAAA"].**

###### **Solution**

**class Solution {  
public:  
 vector<string> findRepeatedDnaSequences(string s) {  
 int sLen = s.length();  
 vector<string> v;  
 if(sLen < 11) return v;  
 char keyMap[1<<21]{0};  
 int hashKey = 0;  
 for(int i = 0; i < 9; ++i) hashKey = (hashKey<<2) | (s[i]-'A'+1)%5;  
 for(int i = 9; i < sLen; ++i) {  
 if(keyMap[hashKey = ((hashKey<<2)|(s[i]-'A'+1)%5)&0xfffff]++ == 1)  
 v.push\_back(s.substr(i-9, 10));  
 }  
 return v;  
 }  
};**

**But the above solution can be invalid when repeated sequence appears too many times, in which case we should use unordered\_map<int, int> keyMap to replace char keyMap[1<<21]{0}here.**

##### **Majority Element**

**Given an array of size n, find the majority element. The majority element is the element that appears more than ⌊ n/2 ⌋ times. (bit-counting as a usual way, but here we actually also can adopt sorting and Moore Voting Algorithm)**

###### **Solution**

**int majorityElement(vector<int>& nums) {  
 int len = sizeof(int)\*8, size = nums.size();  
 int count = 0, mask = 1, ret = 0;  
 for(int i = 0; i < len; ++i) {  
 count = 0;  
 for(int j = 0; j < size; ++j)  
 if(mask & nums[j]) count++;  
 if(count > size/2) ret |= mask;  
 mask <<= 1;  
 }  
 return ret;  
}**

##### **Single Number III**

**Given an array of integers, every element appears three times except for one. Find that single one. (Still this type can be solved by bit-counting easily.) But we are going to solve it by digital logic design**

###### **Solution**

**//inspired by logical circuit design and boolean algebra;  
//counter - unit of 3;  
//current incoming next  
//a b c a b  
//0 0 0 0 0  
//0 1 0 0 1  
//1 0 0 1 0  
//0 0 1 0 1  
//0 1 1 1 0  
//1 0 1 0 0  
//a = a&~b&~c + ~a&b&c;  
//b = ~a&b&~c + ~a&~b&c;  
//return a|b since the single number can appear once or twice;  
int singleNumber(vector<int>& nums) {  
 int t = 0, a = 0, b = 0;  
 for(int i = 0; i < nums.size(); ++i) {  
 t = (a&~b&~nums[i]) | (~a&b&nums[i]);  
 b = (~a&b&~nums[i]) | (~a&~b&nums[i]);  
 a = t;  
 }  
 return a | b;  
}  
;**

##### **Maximum Product of Word Lengths**

**Given a string array words, find the maximum value of length(word[i]) \* length(word[j]) where the two words do not share common letters. You may assume that each word will contain only lower case letters. If no such two words exist, return 0.**

**Example 1:**

**Given ["abcw", "baz", "foo", "bar", "xtfn", "abcdef"]**

**Return 16**

**The two words can be "abcw", "xtfn".**

**Example 2:**

**Given ["a", "ab", "abc", "d", "cd", "bcd", "abcd"]**

**Return 4**

**The two words can be "ab", "cd".**

**Example 3:**

**Given ["a", "aa", "aaa", "aaaa"]**

**Return 0**

**No such pair of words.**

###### **Solution**

**Since we are going to use the length of the word very frequently and we are to compare the letters of two words checking whether they have some letters in common:**

* **using an array of int to pre-store the length of each word reducing the frequently *measuring* process;**
* **since int has 4 bytes, a 32-bit type, and there are only 26 different letters, so we can just use one bit to indicate the existence of the letter in a word.**

**int maxProduct(vector<string>& words) {  
 vector<int> mask(words.size());  
 vector<int> lens(words.size());  
 for(int i = 0; i < words.size(); ++i) lens[i] = words[i].length();  
 int result = 0;  
 for (int i=0; i<words.size(); ++i) {  
 for (char c : words[i])  
 mask[i] |= 1 << (c - 'a');  
 for (int j=0; j<i; ++j)  
 if (!(mask[i] & mask[j]))  
 result = max(result, lens[i]\*lens[j]);  
 }  
 return result;  
}**

#### **Attention**

* **result after shifting left(or right) too much is undefined**
* **right shifting operations on negative values are undefined**
* **right operand in shifting should be non-negative, otherwise the result is undefined**
* **The & and | operators have lower precedence than comparison operators**

### **Sets**

**All the subsets**

**A big advantage of bit manipulation is that it is trivial to iterate over all the subsets of an N-element set: every N-bit value represents some subset. Even better, if A is a subset of B then the number representing A is less than that representing B, which is convenient for some dynamic programming solutions.**

**It is also possible to iterate over all the subsets of a particular subset (represented by a bit pattern), provided that you don’t mind visiting them in reverse order (if this is problematic, put them in a list as they’re generated, then walk the list backwards). The trick is similar to that for finding the lowest bit in a number. If we subtract 1 from a subset, then the lowest set element is cleared, and every lower element is set. However, we only want to set those lower elements that are in the superset. So the iteration step is just i = (i - 1) & superset.**

**vector<vector<int>> subsets(vector<int>& nums) {  
 vector<vector<int>> vv;  
 int size = nums.size();   
 if(size == 0) return vv;  
 int num = 1 << size;  
 vv.resize(num);  
 for(int i = 0; i < num; ++i) {  
 for(int j = 0; j < size; ++j)  
 if((1<<j) & i) vv[i].push\_back(nums[j]);   
 }  
 return vv;  
}**

**Actually there are two more methods to handle this using recursion and iteration respectively.**

### **Bitset**

**A** [**bitset**](http://www.cplusplus.com/reference/bitset/bitset/?kw=bitset) **stores bits (elements with only two possible values: 0 or 1, true or false, ...).**

**The class emulates an array of bool elements, but optimized for space allocation: generally, each element occupies only one bit (which, on most systems, is eight times less than the smallest elemental type: char).**

**// bitset::count  
#include <iostream> // std::cout  
#include <string> // std::string  
#include <bitset> // std::bitset  
  
int main () {  
 std::bitset<8> foo (std::string("10110011"));  
 std::cout << foo << " has ";  
 std::cout << foo.count() << " ones and ";  
 std::cout << (foo.size()-foo.count()) << " zeros.\n";  
 return 0;  
}**

**Always welcom new ideas and practical tricks, just leave them in the comments!**

# New aftert 9/9

1. Python3 heap/priority queue usage:

import heapq

li=[5,7,3,21,7,0]

heapq.heapify(li)

heapq.heappop(li)

heapq.heappush(li,88)

如果输入的元素是自定义的class，比如listnode, 那么不能直接比较大小。 Heapq里面简便的比较大小的方式，是元素改写为tuple，tuple的第一个元素是可以正常比较大小的。不过python 3 有一些版本删去了这个方式。3.6是可以的。python 2 都可以

heapq.heappush(li,(node.val,node))

另外一个办法是改写那个class.对list.sort and heapq.heapush()都是改写\_\_cmp\_\_

2. 看一个string是否是number: str.isnumeric() str.isdigit()

1. 看一个string是否是字母或者digit str.isalnum()
2. set可以直接用sorted，但是没有sort()
3. 查找一个var在list中的index，用 list.index(var) e.g. d.index(‘3’)

3. Collections

Import collections

|  |  |
| --- | --- |
| [namedtuple()](https://docs.python.org/3.6/library/collections.html#collections.namedtuple) | 产生一个有name的tuple |
| [deque](https://docs.python.org/3.6/library/collections.html#collections.deque) | 两边都可以快速pop push的list |
| [ChainMap](https://docs.python.org/3.6/library/collections.html#collections.ChainMap) | dict-like class for creating a single view of multiple mappings |
| [Counter](https://docs.python.org/3.6/library/collections.html#collections.Counter) | 数出现的次数 |
| [OrderedDict](https://docs.python.org/3.6/library/collections.html#collections.OrderedDict) | dict subclass that remembers the order entries were added |
| [defaultdict](https://docs.python.org/3.6/library/collections.html#collections.defaultdict) | dict subclass that calls a factory function to supply missing values |
| [UserDict](https://docs.python.org/3.6/library/collections.html#collections.UserDict) | wrapper around dictionary objects for easier dict subclassing |
| [UserList](https://docs.python.org/3.6/library/collections.html#collections.UserList) | wrapper around list objects for easier list subclassing |
| [UserString](https://docs.python.org/3.6/library/collections.html#collections.UserString) | wrapper around string objects for easier string subclassing |

Counter用法和dict一模一样，就是初始化时候a = counter(), 当成set来添加元素，然后key is 元素，val是次数

然后可以用a.most\_common(10) 来看次数

count = collections.Counter(

word.strip("!?',;.") for word in paragraph.lower().split())

Count[word] 显示word的次数

4. Strip

两端strip掉一些字符，输入的参数是set

a.strip(‘abcdedf’)就是如果两端凡是属于这些字母的全部strip掉

5. cmp

function

def \_\_cmp\_\_(self, other):

return cmp(self.intAttribute, other.intAttribute)

6. 用 a<b<c. a<b<c means a<b and b<c

7. Sort on two keys:

I have a list of lists:

[[12, 'tall', 'blue', 1], [2, 'short', 'red', 9], [4, 'tall', 'blue', 13]]

想先按tall/short sort, then blue/red if tall/short equals

s = sorted(s, key = lambda x: (x[1], x[2]))

Or

import operator s = sorted(s, key = operator.itemgetter(1, 2))

8. Bisect

bisect.bisect(list,x) 可以查找x 在list中插入的位置。

bisect.insort(list,x) 可以插入x并且keep list sorted

bisect.insort\_left(list,x), 插入x，如果有多个x，那么插在相等x的最左边

bisect.insort\_right(list,x) 插入x，多个x，插在最右边

Bisect 最大特点是sort和插入自动化，都是nlogn

1. Python heapq 只支持最小堆。要用最大堆，需要全部搞成负的，就有最大堆的效果了。

### Backtracking

class Solution(object):

def generateParenthesis(self, N):

ans = []

def backtrack(S = '', left = 0, right = 0):

if len(S) == 2 \* N:

ans.append(S)

return

if left < N:

backtrack(S+'(', left+1, right)

if right < left:

backtrack(S+')', left, right+1)

backtrack()

return ans

### Quick Select template

模板思路

Quick select average case O(n), worest case O(n^2)

每次选取一个Pivot，可以选第一个 两个variable, 一个存比pivot小的数的数量left，一个存比pivot大的数的数量right

初始化（两个指针指向的都是下一个对应的数应该放的地方）

* left=0 从头开始
* right = length-1 从尾部开始

pointer从0开始，遇到right结束，后面的肯定逗比pivot大

while pivot <= right:  
 #因为right指向的是比Pivot大的数下一个可以存放的地方，所以应该用<=  
 if nums[pivot] < pivot:  
 swap(nums, pointer, left)  
 pointer += 1 #这里Pointer可以加1，因为left <= pointer这是肯定的，所以前面换过来的数必然<= pivot, 不向前就会死循环  
 left += 1  
 elif nums[pivot] == pivot:  
 pointer += 1  
 else:  
 swap(nums, pointer, right)  
 right -= 1  
 #这里的pointer不能加1，因为不确定后面换过来的数的大小，所以要再次判断

215. Kth Largest Element in an Array

def findKthLargest(self, nums, k):

"""

:type nums: List[int]

:type k: int

:rtype: int

"""

if len(nums)==0:

return None

pivot = nums[0]

left = 0# 此处易错。从0开始。这样left总是表示已经比较过的。如果从1开始，后面left-1才是比较过的。

right = len(nums)-1

p=left+1

while p<=right:

if nums[p]>pivot:

self.swap(nums,p,left)

left+=1

p+=1

elif nums[p]==pivot:

p+=1

else:

self.swap(nums,p,right)

right-=1

#之前left从0开始就已经把pivot更换了。不需要最后再换一次。

if left+1==k:

return pivot

if left<k-1:

return self.findKthLargest(nums[left+1:],k-left-1)

else:

return self.findKthLargest(nums[:left],k)

def swap(self,nums,a,b):

tmp = nums[a]

nums[a]=nums[b]

nums[b]=tmp

### Radix Sort

首先对所有等个位数放进0-9等bucket里面排序。排完以后，一次倒出来成一个list

然后对十位数放bucket。完了再倒出来

反复这个放bucket，倒出来等操作直到最高位也做完。

动画部分见<https://visualgo.net/bn/sorting>

def radix\_sort(array, base=10):  
 def list\_to\_buckets(array, base, iteration):  
 buckets = [[] for x in range(base)]  
 for number in array:  
 # Isolate the base-digit from the number  
 digit = (number // (base \*\* iteration)) % base  
 # Drop the number into the correct bucket  
 buckets[digit].append(number)  
 return buckets  
  
 def buckets\_to\_list(buckets):  
 numbers = []  
 for bucket in buckets:  
 # append the numbers in a bucket  
 # sequentially to the returned array  
 for number in bucket:  
 numbers.append(number)  
 return numbers  
  
 maxval = max(array)  
  
 it = 0  
 # Iterate, sorting the array by each base-digit  
 while base \*\* it <= maxval:  
 array = buckets\_to\_list(list\_to\_buckets(array, base, it))  
 it += 1  
  
 return array

164. Maximum Gap. O(11\*n) is O(n)

def maximumGap(self, nums):

if len(nums)<2:

return 0

maxd = len(str(2\*\*32))

sortedNums = nums

------------------------------此处加上后面的function就是radix sort.

for d in range(maxd):

sortedNums = self.radix(sortedNums,d)

-------------------------------

t=0

for i in range(1,len(sortedNums)):

if sortedNums[i]-sortedNums[i-1]>t:

t=sortedNums[i]-sortedNums[i-1]

return t

def radix(self,nums,d):

buckets = [[] for \_ in range(10)]

for n in nums:

buckets[(n//10\*\*d)%10].append(n)

res=[]

for b in buckets:

res.extend(b)

return res

### Union Find

#### 128. Longest Consecutive Sequence

class UnionFind(object):

def \_\_init\_\_(self,nums):

self.parents=dict()

for n in nums:

self.parents[n]=n

def find(self,x):

if self.parents[x]!=x:

self.parents[x] = self.find(self.parents[x])

return self.parents[x]

def union(self,x,y):

roota = self.find(x)

rootb = self.find(y)

if roota!=rootb:

self.parents[rootb]=roota

def getMaxLen(self):

maxlen=0

counts=dict()

vals=self.parents.keys()

for key in vals:

root = self.find(key) # 这里要注意，root必须用find来选，不能直接用parents里面对应的id。因为前面的扫过的元素，在两个set联合的时候，可能没有再改成同一个id！！！

if root in counts:

counts[root]+=1

else:

counts[root]=1

maxlen=max(maxlen,counts[root])

return maxlen

class Solution:

def longestConsecutive(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

nums=set(nums)

uf = UnionFind(nums)

for n in nums:

if n-1 in nums:

uf.union(n,n-1)

if n+1 in nums:

uf.union(n,n+1)

return uf.getMaxLen()

### Sweep Line

第一要注意的是，一定要 把上升的高度写成负的.

也就是说[2,5,3] 拆成[2,-3],[5,3] 这样在排序等时候才会排到前面。就不会遇到[0,3,3] [3,5,3] 这种情况输出[0,3] [3,0] [3,3] [5,0]

第二要会用defaultdict

第三一定要用dict先存要pop出等点。否则直接remove，每次都要heapify，非常慢。只有每次pop最上面的，才能不LTE。如果要pop的点不是最上面的，就在dict里面存着，直到下次一起pop出来。

from heapq import heappush,heappop

from collections import defaultdict

import time

class Solution:

def getSkyline(self, buildings):

edges = []

for b in buildings:

edges.append((b[0],-b[2]))#一定要 把上升的高度写成负的.

也就是说[2,5,3] 拆成[2,-3],[5,3] 这样在排序等时候才会排到前面。就不会遇到[0,3,3] [3,5,3] 这种情况输出[0,3] [3,0] [3,3] [5,0]

edges.append((b[1],b[2]))

edges = sorted(edges)

t2=time.time()

live=[0]

res=[]

out=defaultdict(lambda:0)

for e in edges:

if e[1]<0:

if e[1]<live[0]:

res.append([e[0],-e[1]])

heappush(live,e[1])

else:

if -e[1]==live[0]:

heappop(live)#一定要用dict先存要pop出等点。否则直接remove，每次都要heapify，非常慢。只有每次pop最上面的，才能不LTE。如果要pop的点不是最上面的，就在dict里面存着，直到下次一起pop出来（下面的while）。

while live[0] in out.keys() and out[live[0]]>0:

out[live[0]]-=1

heappop(live)

if len(res)==0 or -live[0]!=res[-1][1]:

res.append([e[0],-live[0]])

else:

out[-e[1]]+=1

print(time.time()-t2)

return res

### Topological Sort

#### class Node(object): def \_\_init\_\_(self): self.IN = set() self.OUT = set()

def toposort:

res = ""

while graph:

oldlen = len(graph)

for key in graph:

if not graph[key].IN: # to remove this

for key2 in graph[key].OUT:

graph[key2].IN.remove(key)

del graph[key]

res += key

break

if oldlen == len(graph): # if shrinking stops, solution doesn't exist

return ""

oldlen = len(graph)

return res

#### 269. Alien Dictionary

1. 如果你在loop 一个dictionary 的时候，你加上或者减去了dictionary的key，那么会出现dictionary change size in loop error. 但是，如果你加上一个break，就在处理了第一个以后，就退出来，就不会出现那个error！！ 外面再加一个while之类的循环，就可以达到修改dictionary的效果！！
2. Remove a key from dictionary 的方法

my\_dict.pop('key', None)

Or if 'key' in myDict: del myDict['key']

class Node():

def \_\_init\_\_(self):

self.In=set()

self.Out=set()

class Solution:

def alienOrder(self, words):

graph=dict()

for w in words:

for l in w:

if l not in graph.keys():

graph[l]=Node()

for pair in zip(words,words[1:]): # 此处注意，zip里面可以不一样长，以短的为限

for a,b in zip(\*pair): # 此处注意，对一个list，加上\*即可

if a!=b:# 此处格外注意，一定要a!=b

graph[a].Out.add(b)

graph[b].In.add(a)

break# break也非常重要，否则会加上后面的char，但因为之前才是关系，后面的不是真正的关系

res=[]

while graph:

glen=len(graph)

for key in graph:

if len(graph[key].In)==0:

for k in graph[key].Out:

graph[k].In.remove(key)

res.append(key)# 注意除了这个for循环，key就不存在了。所以这一句要在for里面写

del graph[key] #注意不要写成graph.remove(key),没有那个function

break# 可以解决dictionary change size in loop 的问题

if len(graph)==glen:

return ""

return "".join(res)

### DFS/BFS

#### 301. Remove Invalid Parentheses

1.用个set保存已经访问过的作为pruning否则LTE

class Solution:

def \_\_init\_\_(self):

self.minLen=0

self.visited=set()

def removeInvalidParentheses(self, s):

"""

:type s: str

:rtype: List[str]

"""

res=[]

olen = len(s)

self.minLen = olen

flag=0

self.dfs("",s,0,res,olen)

res=set(res)

result=[]

for r in res:

if len(r)==olen-self.minLen:

result.append(r)

return result

def dfs(self,left,right,flag,res,olen):

if (left,right,flag) in self.visited:

return

if flag<0:

return

if len(right)==0:

if flag!=0:

return

else:

if(olen-len(left)<=self.minLen):

self.minLen=min(self.minLen,olen-len(left))

res.append(left)

else:

return

else:

if right[0]=='(':

self.dfs(left+right[0],right[1:],flag+1,res,olen)

elif right[0]==')':

self.dfs(left+right[0],right[1:],flag-1,res,olen)

else:

self.dfs(left+right[0],right[1:],flag,res,olen)

self.dfs(left,right[1:],flag,res,olen)

self.visited.add((left,right,flag))

### HashMap

336 Palindrom pairs

这道题给我们了许多单词，让我们找出回文对，就是两个单词拼起来是个回文字符串，我最开始尝试的是brute force的方法，每两个单词都拼接起来然后判断是否是回文字符串，但是通过不了OJ，会超时，可能这也是这道题标为Hard的原因之一吧，那么我们只能找别的方法来做，通过学习大神们的解法，发现如下两种方法比较好，其实两种方法的核心思想都一样，写法略有不同而已，那么我们先来看第一种方法吧，要用到哈希表来建立每个单词和其位置的映射，然后需要一个set来保存出现过的单词的长度，算法的思想是，遍历单词集，对于遍历到的单词，我们对其翻转一下，然后在哈希表查找翻转后的字符串是否存在，注意不能和原字符串的坐标位置相同，因为有可能一个单词翻转后和原单词相等，现在我们只是处理了bat和tab的情况，还存在abcd和cba，dcb和abcd这些情况需要考虑，这就是我们为啥需要用set，由于set是自动排序的，我们可以找到当前单词长度在set中的iterator，然后从开头开始遍历set，遍历比当前单词小的长度，比如abcdd翻转后为ddcba，我们发现set中有长度为3的单词，然后我们dd是否为回文串，若是，再看cba是否存在于哈希表，若存在，则说明abcdd和cba是回文对，存入结果中，对于dcb和aabcd这类的情况也是同样处理，我们要在set里找的字符串要在遍历到的字符串的左边和右边分别尝试，看是否是回文对，这样遍历完单词集，就能得到所有的回文对，

class Solution:

def palindromePairs(self, words):

res = []

dic = {}

for i in range(len(words)):

dic[words[i]] = i

for word in words:

word\_rev = word[::-1]

if word\_rev in dic and dic[word\_rev] != dic[word]:

res.append([dic[word\_rev], dic[word]])

for i in range(len(word)):

temp1 = word[:i][::-1] #注意这里要同时考虑前面半部分连接和后面半部分连接。所以要分temp1 和temp2 两种情况。

temp2 = word\_rev[:i]

if temp1 in dic and isPal(word[i:]):

res.append([dic[word], dic[temp1]])

if temp2 in dic and isPal(word\_rev[i:]):

res.append([ dic[temp2], dic[word]])

return res

def isPal(word):

left = 0

right = len(word)-1

while left < right:

if word[left] != word[right]:

return False

left += 1

right -= 1

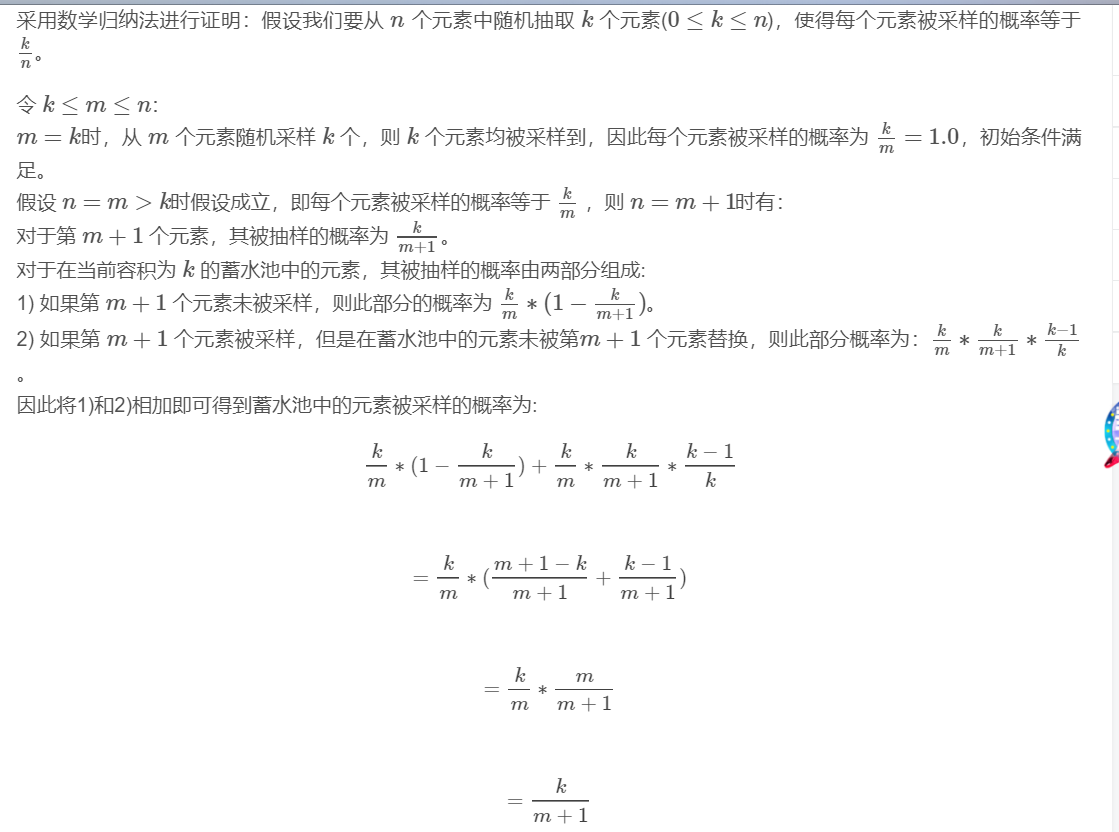
return True

Resevouir sampling

设立一个k大小的pool

然后前k个放进pool

后面的，对每个用count 分之一的概率决定是否放进pool，如果放进pool，那么随机替换pool中一个元素

证明

#### 382. Linked List Random Node

def \_\_init\_\_(self, head):

self.head=head

def getRandom(self):

p = self.head

res=p.val

count=1

while p.next:

count+=1

p=p.next

if random.randint(1,count)==1:

res=p.val# 注意这里不能用self.head来作为reseviour存储。因为lc平台每个例子之间不变，所以会报错

return res

### Reject Sampling

#### 470. Implement Rand10() Using Rand7()

当你uniform的产生1-n的数字，但你只要1-k的。那么如果大于k的都去掉。产生的数字在1-k 还是uniform的

def rand10(self):

"""

:rtype: int

"""

a = rand7()

b = rand7()

num = 7\*(a-1)+b-1

if num>=40:

return self.rand10()

return num//4+1

### Trie

### Segment Tree

1. 用来做 Range query
2. 功能和binary indexed tree similary
3. 更强大，还可以range max, range min, not only range sum
4. Update and query 复杂度O（logn）, build tree O(n)
5. 对N个数组，ST的存在一个长度为2N的数组中
6. **注意此处我们对于segmentTree]数组的索引从1开始算起**。则对于数组中的任意结点i，其左子结点为2\*i，右子结点为2\*i + 1，其母结点为i/2。

Tree based-implementation:

注意node里面的start和end是都包括在里面的【】 形式

class Node(object):

def \_\_init\_\_(self, start, end):

self.start = start

self.end = end

self.total = 0

self.left = None

self.right = None

class NumArray(object):

def \_\_init\_\_(self, nums):

def createTree(nums, l, r):

if l > r:

return None

if l == r:

n = Node(l, r)

n.total = nums[l]

return n

mid = (l + r) // 2

root = Node(l, r)

root.left = createTree(nums, l, mid)

root.right = createTree(nums, mid+1, r)

root.total = root.left.total + root.right.total

return root

self.root = createTree(nums, 0, len(nums)-1)

def update(self, i, val):

def updateVal(root, i, val):

if root.start == root.end:

root.total = val

return val

mid = (root.start + root.end) // 2

if i <= mid:

updateVal(root.left, i, val)

else:

updateVal(root.right, i, val)

root.total = root.left.total + root.right.total

return root.total

return updateVal(self.root, i, val)

def sumRange(self, i, j):

def rangeSum(root, i, j):

if root.start == i and root.end == j:

return root.total

mid = (root.start + root.end) // 2

if j <= mid:

return rangeSum(root.left, i, j)

elif i >= mid + 1:

return rangeSum(root.right, i, j)

else:

return rangeSum(root.left, i, mid) + rangeSum(root.right, mid+1, j)

return rangeSum(self.root, i, j)

#### 273 Integer to Words

a.分类的时候要用>=1e6,>=1e3,否则1000就不会分到对的区间。

b.1000000在1e3-1e6位置是0，不能输出任何val，需要判断如果是空就不能加上thousand,否则产生1million thousand.

C. 还要注意，123%1000产生的数字是int，但是123%1e3产生的是float！123//100是int，123//1e2就是float，float不能做array的index！！

#### 146. LRU Cache

用doubly linked list and hashmap

错误点

1. Doubly linkedlist，每次移动还要注意改后面node 的prev，不像linkedlist那样只要改next
2. Dict每个key对应value, and node. 这两个用[] 包括，不要用tuple，因为tuple没法只改其中要给元素
3. Move node的function中，如果node==last,也就是说现在这个node已经是last，那么不要做任何操作。否则在修改next.prev会出现问题

Heap 插入的time complexity is O(1)

Delete is O(logn)

#### 681. Next Closest Time

1. 看一个string是否是number: str.isnumeric() str.isdigit()
2. 看一个string是否是字母或者digit str.isalnum()
3. set可以直接用sorted，但是没有sort()
4. 查找一个var在list中的index，用 list.index(var) e.g. d.index(‘3’)

#### 819. Most Common Word

1. Counter. 当成set添加元素，自动生成key 为元素，val为次数的dictionary
2. Strip 去掉两端的，输入会自动转成set。凡是出现在输入里面的全都自动去掉

#### 42. Trapping Rain water

需要重温算法。

33. Search in Rotated array

1. While left<=right in case search 1 in [1]
2. 不要专门考虑0,1,2,3,4,5,6,7这种未旋转的可能性。已经包括在后面了
3. 用 a<b<c. a<b<c means a<b and b<c

253 Meeting Rooms 2

先按第一dimension sort，如果相同按第二dimension, 也就是a first, than b if a equals.

Sort on two keys:

I have a list of lists:

[[12, 'tall', 'blue', 1], [2, 'short', 'red', 9], [4, 'tall', 'blue', 13]]

想先按tall/short sort, then blue/red if tall/short equals

s = sorted(s, key = lambda x: (x[1], x[2]))

Or

import operator s = sorted(s, key = operator.itemgetter(1, 2))

683 K Empty Slots

Bisect

bisect.bisect(list,x) 可以查找x 在list中插入的位置。

bisect.insort(list,x) 可以插入x并且keep list sorted

bisect.insort\_left(list,x), 插入x，如果有多个x，那么插在相等x的最左边

bisect.insort\_right(list,x) 插入x，多个x，插在最右边

Bisect 最大特点是sort和插入自动化，都是nlogn

17. Phone number combination

先想清楚dfs怎么写，然后注意“”的话直接返回，否则容易加入一个“”在res里面。这个其实也未必不对，看答案要求

还要注意\_\_init\_\_(self) 必须要用self argument，不可以忘记

46 Permutations

A is a list

a.remove(3)

a.append(3) 都是在原来数组上操作，返回都是Null！

所以要想产生新数组还不改变原来的， a+[1], a[:3]+a[4:]

### **543. Diameter of Binary Tree**

1. Python的 local function 对外面的参数不能直接用。所以必须要把外面的参数写成class的attribute，才能在里面的函数直接调用而不传递
2. 不能在定义一个function之前久调用他。只能写成class的method，才可以在那个method定义前，由其他method调用。

20 valid parentheses

1. If [] => false
2. If not [] => true
3. Python 里面没有switch case， 　所以要用dictionary。 dictionary可以放function。 详见<https://jaxenter.com/implement-switch-case-statement-python-138315.html>

13. Roman to integer

易错点 不要错误的把坐标i作为key，要把s[i] 作为key

Dict[s[i]] not dict[i]

12 integer to roman

细心不要把dict里面的80，90打错了

14. Longest common prefix

Zip的用法

zip(list1,list2,...) 可以放无数多个list, 不需要在list外面加【】

如果要对二维数组，或者list of string zip. 需要用\*

zip(\*list)

Filter 返回filter object，需要用list 改变成list

Map 返回也是map object, 需要list改变成list

Apply是pandas dataframe 里面的东西，不自带

a=[i for i in range(10)]

b=list(filter(lambda x:x>2,a))

c=list(map(lambda y:y\*\*2,a))

Not “” 为true

If “” 为false

a=1

Def func():

print(a)

这时候，def里面可以识别前面定义的var

但是如果是class，就必须只能识别 self.var

38. Lambda 函数，不要写return! Lambda x: return str(x) is wrong! Should be lambda x: str(x)

49. 1.return tuple(list), not '#'.join(), 2. return list(dict.values). dict.values可以返回所以values，但是要convert 成list. 3. From collections import defaultdict

67 add binary

int array 不能直接join，需要先map(lambda x:str(x),arr), 如果用recursion就不用reverse了。

69. sqrt(x)

1. x=0. 2. if mid\*mid<=x and (mid+1)\*\*2>x:

return mid

if (mid+1)\*\*2<=x:# 此处一定要是《=， 否则出错。记住如果等于的话，被放到else情况下无限循环

25. Reverse nodes in k-group

1. 注意设立pre以后，加上pre.next=head, 2. while listLen>k:

p1=p0.next# 此行和下一行放这里，不要放循环最后。否则如果不大于k会溢出

p2=p1.next

for i in range(1,k):# 此处1->k or 0->k-1, 否则会多换一个。 一定要自己走一遍。

tmp=p2.next

p2.next=p0.next

p0.next=p2

p1.next=tmp

p2=p1.next

listLen-=k

p0=p1

30. Substring with concatenation of all words:

1. Use Counter. But need to convert counter to dict every time
2. In the final loop, use for i in range(0,len(s)-len(words)\*wordLength+1). Not for i in range(len(s)).

139 Word break

注意一次搜索玩不成功不要返回false，要等全搜索完再返回false

memo=dict()

def helper(s,memo):

if not s:

return True

if s in wordDict:

return True

if s in memo and memo[s]:

return True

if s in memo and not memo[s]:

return False

for w in wordDict:

if s.startswith(w):

if helper(s[len(w):],memo):

memo[s]=True

return True

memo[s]=False# 不要在之前等if里面定false。在完成这个search以后定false

return False

return helper(s,memo)

60 permutation

k 先要减一。 ''.join(num) 需要num里面都是str，如果是int就不好

res=[]

numslist=list(range(1,10))

k-=1# 此处一定要减去一

while n>0:

times=1

for i in range(1,n):

times\*=i

res.append(str(numslist[k//times]))

numslist.remove(numslist[k//times]) # 此处不要放在k改变以后

k = k%times

n-=1

return ''.join(res)# 此处res里面必须是str，如果是int久不能join

72 edit distance

假设dp[i][j] 表示从word1[i-1] 与 word2[i-1] convert的最小distance

那么当word1i == word2 j, dp(i+1,j+1) = dp(i,j)

If word1i!=word2j, dp(i+1,j+1) =

1. dp(i+1,j) +1 从word2 j-1 插入要给字符，
2. dp(i,j) +1 这里要格外注意。 i,j 是已经相等了，这事扩展到i+1,j+1的时候，新字符不一样，所以替换也是加一。 这里并不是增加字符，所以不需要 +1 再加一
3. dp(i,j+1)+1. 从word1 插入

三者取最小值

This is a classic problem of Dynamic Programming. We define the state dp[i][j] to be the minimum number of operations to convert word1[0..i - 1] to word2[0..j - 1]. The state equations have two cases: the boundary case and the general case. Note that in the above notations, both i and j take values starting from 1.

For the boundary case, that is, to convert a string to an empty string, it is easy to see that the mininum number of operations to convert word1[0..i - 1] to ""requires at least i operations (deletions). In fact, the boundary case is simply:

1. dp[i][0] = i;
2. dp[0][j] = j.
3. dp[i][0] = i;
4. dp[0][j] = j;
5. dp[i][j] = dp[i - 1][j - 1], if word1[i - 1] = word2[j - 1];
6. dp[i][j] = min(dp[i - 1][j - 1] + 1, dp[i - 1][j] + 1, dp[i][j - 1] + 1), otherwise.

79. Word search

For i in [i-1,i+1]:

For j in [j-1,j+1]:

是不对的

应该是

ijNext = [[i+1,j],[i-1,j],[i,j+1],[i,j-1]]

for a,b in ijNext:

if self.dfs(board,word[1:],a,b):

return True

93 restore ip address

01不是valid ip 段

除了0，不能用0开头，

E.g 10.0.01.24 不是valid ip

应该是10.0.1.24

note:

List 没有prepend，而是 list.insert(0,var)

### 144 iterative preorder:

用stack，先有再左的push进去!

### 8 atoi

-+1,-1.0,42f,最大整数 2\*\*31-1，最小-2\*\*32, 因为第一位要留着存符号，负数是因为有补码

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

二分法对left 和right 对mid是否要加一减一的情况，要仔细考虑各种情况是否会死循环。

### **43. Multiply Strings**

1. map(Lambda x:str,intarray)不对，必须要str(x)
2. 如果遇到纯0， rstrip（‘0’）会输出“”。 不对

81. Search in Rotated Sorted Array II

2<x<3 在python里面可以直接写出来，不需要分开写

68 text justification

array.append() 返回none，只是在原数组上操作

要想返回新的数组，用array + []

126 word ladder II

Def …

abc=0

Def …

在这里面，不能接触到外面的abc

B. 如果要loop abcdef...z, 直接用 for c in ‘abcdefghijklmnopqrstuvwxyz’ 不用写成set

C. 如果遇到最短路径问题，用BFS，不要用DFS

### 81. Search in Rotated Sorted Array II

核心是找左边还是右边是sorted。sorted 就可以用大小比较，如果不在中间就肯定在另外一段

287  Find the Duplicate Number

还可以用快慢指针法detect cycle。Fast=nums[nums[index]]

Slow=nums[index]

如果fast=slow，那么就有cycle。

然后cycle的起点，

ptr1 = nums[0]

ptr2 = tortoise

while ptr1 != ptr2:

ptr1 = nums[ptr1]

ptr2 = nums[ptr2]

类似于142 detect cycle in linkedlist 2

# 非高频

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

非高频hard

### 156. Binary Tree Upside Down

159.

161.One Edit Distance

如果有不同字母 s[:i]+t[i]+t[i+1:] . 然后最终比较s==t or s==t[:-1]. 到-1是防止t比s长要给字母

163.要细心

170 保存一个计数器。然后比较是否存在还要比较个数是否大于1.

174

186

199 做一个level order traversal，然后每一个level取最后边的即可。

201

209 双针法解决第一个问题，follow up还没看

210 topological sort. 但是实际写程序可能不美好。建议看答案

211 trie. Can use dict to implement trie

212

213

214

216

217

219

220

221

222. Count Complete Tree Nodes

通过上面的定义，我们可以看出二者的关系是，完美二叉树一定是完全二叉树，而完全二叉树不一定是完美二叉树。那么这道题给的完全二叉树就有可能是完美二叉树，若是完美二叉树，节点个数很好求，为2的h次方-1，h为该完美二叉树的高度。这道题可以用递归和非递归两种方法来解。我们先来看递归的方法，思路是分别找出以当前节点为根节点的左子树和右子树的高度并对比，如果相等，则说明是满二叉树，直接返回节点个数，如果不相等，则节点个数为左子树的节点个数加上右子树的节点个数再加1(根节点)，其中左右子树节点个数的计算可以使用递归来计算，

223需要判断矩阵是否相交，相交的话减去重复面积(顶点相交除外)

225

228

229

230

231

232

233

241

243

244

245

246

247

248

249

250

251

252

254

255

256

257

259

261

262

264

265

266

267

270

271

274

275

276

277

278

281

282

284

285

286

288

290

291

293

294

296

298

299