Table of Contents

My leetcode Book

这本小书主要有：

各种小轮子

几种主要类型的leetcode题目和答案

This book is for those people who want to learn some basic data structures and algorithms to crack the coding interviews. This book show how to implement basic data structures like linked list and trie. It also have some typical questions from leetcode with answers and explanations.

Dynamic Programming

主要是整理九章算法强化班的Dynamic Programming的题目解法。

老师讲的作用毕竟有限，还是要宝宝们动手实现一遍。科科

198 House Robber

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

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

Example Given [3, 8, 4], return 8.

思路： dp[i]代表 nums[0:i]最多能抢多少钱， dp[i] = max(dp[i-1], dp[i-2]+ nums[i])。就是一个根据前面2个状态决定当前这个房子抢还是不抢。

LC: https://leetcode.com/problems/house-robber/

```
class Solution(object):
    def rob(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        if not nums:
            return 0

        # if house smaller than 2.
        if len(nums)<=2:
            return max(nums)

        # init first 2
        dp = [0]+[nums[0]]

        # do the dp loop
        for i in range(1, len(nums)):
            dp.append(max(dp[-2]+nums[i], dp[-1]))

        return dp[-1]

so = Solution()
print so.rob([2,3,5,2,1])
```

Follow up: The dp array can be replaced by 2 variables. second_last, and last

213 House Robber2

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

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

思路：跟上题大致相同，重点是怎么把环形拆成array来做，其实，array[0:n-1]和array[1:n]的dp结果最大的就是最优解。 可以理解为最优解要么包含这个房子，要么不包含这个房子。包含这个房子时，肯定不包含相邻的房子。所以直接从array里面删掉2个相邻的，拆成array来解两次，得到最优就行了。

LC: https://leetcode.com/problems/house-robber-ii/

```python
class Solution(object):
    def rob(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        if not nums:
            return 0

        # if house smaller than 2.
        if len(nums)<=3:
            return max(nums)

        # do rob from nums[1:n]
        dp = [0]+[nums[1]]

        # do the dp loop
        for i in range(2, len(nums)):
            dp.append(max(dp[-2]+nums[i], dp[-1]))
        res1 = dp[-1]

        # do rob from nums[0:n-1]
        dp = [0]+[nums[0]]

        # do the dp loop
        for i in range(1, len(nums)-1):
            dp.append(max(dp[-2]+nums[i], dp[-1]))
        res2 = dp[-1]

        return max(res1, res2)

so = Solution()
print so.rob([2,3,5,2,1])
```

221 Maximal Square

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

For example, given the following matrix:

```
1 0 1 0 0
1 0 1 1 1
1 1 1 1 1
1 0 0 1 0
```

Return 4.

思路： 非常简单，公式 if matrix==1 那么 dp[i][j] = min(dp[i-1][j], dp[i][j-1], dp[i-1][j-1])+1 就很清楚了

LC: https://leetcode.com/problems/maximal-square/

jiuzhang: http://www.lintcode.com/en/problem/maximal-square/

```
class Solution(object):
    def maximalSquare(self, matrix):
        """
        :type matrix: List[List[str]]
        :rtype: int
        """
        if not matrix or not matrix[0]:
            return 0
        r, c = len(matrix), len(matrix[0])

        #init dp 2D array
        dp = [[0 if matrix[i][j]=='0' else 1 for j in range(c)]
for i in range(r)]

        for i in range(1, r):
            for j in range(1, c):
                if matrix[i][j]=='1':
                    dp[i][j] = min(dp[i-1][j], dp[i][j-1], dp[i-1
][j-1])+1
                else:
                    dp[i][j] = 0

        # find the longest square
        res = max([max(i) for i in dp])
        return res**2


so = Solution()
print so.maximalSquare([[1]])
```

4 Longest Increasing Subsequence

Given an unsorted array of integers, find the length of longest increasing subsequence.

For example, Input [10, 9, 2, 5, 3, 7, 101, 18]

LIS is [2, 3, 7, 101].

Return 4

Your algorithm should run in O(n2) complexity.

思路： dp[i]代表nums[0:i]的LIS。有点像1维DP。其实是2D DP。如果第i个数，比第k个数大，那么dp[i]=dp[k]+1, 可是如果比前面好多数大怎么办。那么就需要取前面所有dp里面值最大的+1。毕竟+1秒我们要加在最长者的地方。

递推公式是：dp[i] = max(dp[k] for k in [0:i-1] if nums[i]>nums[k]) + 1秒

LC https://leetcode.com/problems/longest-increasing-subsequence/

jiuzhang http://www.lintcode.com/en/problem/longest-increasing-continuous-subsequence/

```
class Solution(object):
    def lengthOfLIS(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        if not nums:
            return 0

        dp = [1]*len(nums)

        for i in range (1, len(nums)):
            for j in range(i):
                if nums[i] >nums[j]:
                    dp[i] = max(dp[i], dp[j]+1)

        return max(dp)
```

Follow up：为什么1D array你说是 n^2的复杂度。 答：因为每更新一个，都要把前面的全部扫一遍。n(n-1)/2 = O(n^2)

5 Longest Increasing Path in a Matrix

Given an integer matrix, find the length of the longest increasing path.

From each cell, you can either move to four directions: left, right, up or down. You may NOT move diagonally or move outside of the boundary (i.e. wrap-around is not allowed).

Example 1:

```
nums = [
  [9,9,4],
  [6,6,8],
  [2,1,1]
]
```

[1, 2, 6, 9]

Return 4

思路：dp[i][j]代表matrix中以i, j结尾的情况，LIP的长度。 递推公式当然就是dp[i][j] = max(dp[i-1][j], dp[i+1][j], dp[i][j-1], dp[i][j+1]) + 1 。然后我们发现这个dp[i][j]居然是由4个不同方向的决定的，如果用for loop来实现就会很日狗。所以我们就用memo，把算过的存起来。整体实现方法是dfs，遍历matrix中的每一个点。如果算过就直接返回长度，如果没算过就继续dfs下去。复杂度嘛。O(m*n)因为不会重复计算，每个点访问一次。

LC https://leetcode.com/problems/longest-increasing-path-in-a-matrix/

jiuzhang http://www.lintcode.com/en/problem/longest-increasing-continuous -subsequence-ii/

```
class Solution(object):
    def longestIncreasingPath(self, matrix):
        """
        :type matrix: List[List[int]]
        :rtype: int
        """
        if not matrix or not matrix[0]: return 0
        self.m, self.n = len(matrix), len(matrix[0])
        self.memo = [[0]*self.n for _ in range(self.m)]
        return max(self.dfs(i, j, matrix) for i in range(self.m)
for j in range(self.n))

    def dfs(self, i, j, matrix):
        if not self.memo[i][j]:
            val = matrix[i][j]
            self.memo[i][j] = 1+ max(
                self.dfs(i+1, j, matrix) if i < self.m - 1 and v
al < matrix[i+1][j] else 0,
                self.dfs(i-1, j, matrix) if i and val < matrix[i
-1][j] else 0,
                self.dfs(i, j+1, matrix) if j < self.n - 1 and v
al < matrix[i][j+1] else 0,
                self.dfs(i, j-1, matrix) if j and val < matrix[i
][j-1] else 0)
        return self.memo[i][j]
```

另外一一种写法 直接dfs暴力解：

```
def longestIncreasingPath(matrix):
    if not matrix or not matrix[0]: return 0
    n, m = len(matrix), len(matrix[0])
    count = 0

    visited = [[0]*m for i in range(n)]

    for i in range(n):
        for j in range(m):
            count = max(dfs(i, j, 1, matrix), count)
    return count

def dfs(i, j, count, matrix):
    tmp = [(i+1, j), (i-1, j), (i, j+1), (i, j-1)]
    neib = [matrix[w][c] for w, c in tmp if 0<=w<len(matrix) and
0<=c<len(matrix[0]) ]

    if matrix[i][j]>=max(neib):
        return count
    result = []
    for w, c in [(i+1, j), (i-1, j), (i, j+1), (i, j-1)]:
        if 0<=w<len(matrix) and 0<=c<len(matrix[0]):
            if matrix[i][j]<matrix[w][c]:
                result.append(dfs(w, c, count+1, matrix))
    return max(result)
```

Coins in a Line 1

There are n coins in a line. Two players take turns to take one or two coins from right side until there are no more coins left. The player who take the last coin wins.

Could you please decide the first play will win or lose?

Example n = 1, return true.

n = 2, return true.

n = 3, return false.

n = 4, return true.

n = 5, return true.

思路： dp[i]代表i个硬币是能不能赢，所以递推公式是 dp[i] = (!dp[i-1]) or (!dp[i-2])。 而i个的硬币能赢代表i-1个硬币，i-2个硬币时，你的对手都不能赢。反着想一下，从递推的公式可以看出，由i和i+1能够推出i+2的结果。所以一个for loop搞定。

http://www.lintcode.com/en/problem/coins-in-a-line/

```
class Solution(object):
    def coin1(self, n):
        if n==0:
            return False
        if n==1 or n==2:
            return True

        dp = [1, 1]
        for i in range(2, n):
            # play can win only when dp[-1] and dp[-2] can't win.

            if not dp[-1] or not dp[-2]:
                new = 1
            else:
                new = 0
            dp.append(new)

        return dp[-1]==1

so = Solution()
print so.coin1(6)
```

Coins in a Line II

There are n coins with different value in a line. Two players take turns to take one or two coins from left side until there are no more coins left. The player who take the coins with the most value wins.

Could you please decide the first player will win or lose?

Example Given values array A = [1,2,2], return true.

Given A = [1,2,4], return false.

思路： 改成从右边拿硬币好理解一些， dp[i]代表从coins[0:i]能拿到的最大价值数。所以核心思想是你要minimize的对手的拿到的价值。 dp[i] = sum[i] - min(dp[i-1], dp[i-2]). 所以又是一个能由i 和 i+1推到出i+2的问题。初始化前两个，然后就可以for loop啦。

http://www.lintcode.com/en/problem/coins-in-a-line-ii/

```
class Solution(object):
    def coin1(self, nums):
        """
        :type nums: List[int]
        :rtype: boolean
        """
        # reverse the list
        nums = nums[::-1]

        if len(nums)==0:
            return -1

        if len(nums)==1 or len(nums)==2:
            return True

        if len(nums)==3:
            return sum(nums[1:])>nums[0]

        dp = [0]+ [nums[0]]

        for i in range(1,len(nums)):
            cur = sum(nums[:i+1]) - min(dp[-1], dp[-2])
            dp.append(cur)

        return dp[-1] > max(dp[-2], dp[-3])

so = Solution()
print so.coin1([1,2,4])
```

Coins in a Line III

There are n coins in a line. Two players take turns to take a coin from one of the ends of the line until there are no more coins left. The player with the larger amount of money wins.

Could you please decide the first player will win or lose?

Have you met this question in a real interview? Yes Example Given array A = [3,2,2], return true.

Given array A = [1,2,4], return true.

Given array A = [1,20,4], return false.

jiuzhang http://www.lintcode.com/en/problem/coins-in-a-line-iii

Stone Game

There is a stone game.At the beginning of the game the player picks n piles of stones in a line.

The goal is to merge the stones in one pile observing the following rules:

At each step of the game,the player can merge two adjacent piles to a new pile. The score is the number of stones in the new pile. You are to determine the minimum of the total score.

Example For [4, 1, 1, 4], in the best solution, the total score is 18:

1. Merge second and third piles => [4, 2, 4], score +2
2. Merge the first two piles => [6, 4]，score +6
3. Merge the last two piles => [10], score +10 Other two examples: [1, 1, 1, 1] return 8 [4, 4, 5, 9] return 43

Tags Related Problems Code editor is not visible on small screen.

思路：dp[i][j]代表从第i到j个石头，最小的得分。递推公式：

```
dp[i][j]= sum(num[i:j]) + min([dp[i][k]+dp[k+1][j] for k in rang
e(i,j)])。
```

公式可以这样理解，从i到j石头的最小得分等于把这堆石头分成两堆的所有得可能里面取最小的。因为最后还要合并一次，所以需要加sum(nums[i:j])。因为是从上往下拆分，所以不难理解用dfs来做。并且用memo来存储已经计算过的结果。代码也非常好理解呢！

jiuzhang http://www.lintcode.com/en/problem/stone-game/

```
class Solution(object):
    def stoneGame(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        if not nums: return 0
        self.memo = [[0]*len(nums) for _ in range(len(nums))]
        return self.dfs(nums, 0, len(nums)-1)

    def dfs(self, nums, i, j):
        if i==j: return 0

        if not self.memo[i][j]:
            tmp = [self.dfs(nums, i, k) + self.dfs(nums, k+1, j)
for  k in range(i, j)]
            self.memo[i][j] = sum(nums[i:j+1]) + min(tmp)

        return self.memo[i][j]

so = Solution()
print so.stoneGame([4, 1, 1, 4])
```

10 Burst Balloons

Given n balloons, indexed from 0 to n-1. Each balloon is painted with a number on it represented by array nums. You are asked to burst all the balloons. If the you burst balloon i you will get nums[left] *nums[i]* nums[right] coins. Here left and right are adjacent indices of i. After the burst, the left and right then becomes adjacent.

Find the maximum coins you can collect by bursting the balloons wisely.

Note: (1) You may imagine nums[-1] = nums[n] = 1. They are not real therefore you can not burst them. (2) 0 ≤ n ≤ 500, 0 ≤ nums[i] ≤ 100

Example:

Given [3, 1, 5, 8]

Return 167

```
nums = [3,1,5,8] --> [3,5,8] -->   [3,8]   -->  [8]  --> []
  coins =  3*1*5      +  3*5*8    +  1*3*8      + 1*8*1   = 167
```

https://leetcode.com/problems/burst-balloons/

思路：Top down 递推公式是 dp[l,r] = max(dp[l,i]+ dp[i+1,r] + nums[i]*nums[l]*nums[r]) for i in [l+1, r-1]。 因为是top down嘛。所以要用个memo来存一下。

```
class Solution(object):
    def maxCoins(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        self.memo = {}
        nums = [1] + nums + [1]
        res = self.dfs(nums, 0, len(nums)-1)
        return res

    def dfs(self, nums, l, r):

        if (l,r)  not in self.memo:
            if l+1 == r: return 0

            res = 0
            for i in range(l+1, r):
                tmp = self.dfs(nums, l, i) + self.dfs(nums, i, r
) + nums[i]*nums[l]*nums[r]
                res = max(res, tmp)
            self.memo[(l, r)] = res

        return self.memo[(l, r)]
```

mianjing

Snapchat

这里面是我整理snapchat的所有面经，祝大家好运。

sc

P1.从东北往西南打印矩阵

input:

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

output:

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

```
class Solution(object):
    def so(self, matrix):
        """
        :type matrix: List[List[int]]
        :retype: List[List[int]]
        """
        if not matrix or not matrix[0]:
            return

        end = len(matrix) + len(matrix[0])-1
        res = []

        for num in range(end):
            tmp = []
            for i in range(num+1):
                if i<len(matrix) and (num-i)<len(matrix[0]):
                    tmp.append(matrix[i][num-i])
            res.append(tmp)
        return  res


so = Solution()
print so.so([[1,2,3,4],[5,6,7,8]])
```

P2. Merge K sorted list https://leetcode.com/problems/merge-k-sorted-lists/

```
class ListNode(object):
    def __init__(self, x):
        self.val = x
        self.next = None

import heapq
class Solution(object):
    def mergeKLists(self, lists):
        """
        :type lists: List[ListNode]
        :rtype: ListNode
        """
        dummy = ListNode(None)
        head = dummy
        q = []

        for node in lists:
            if node:
                heapq.heappush(q, (node.val, node))

        while q:
            tmp = heapq.heappop(q)
            dummy.next = tmp[1]
            dummy = dummy.next
            if tmp[1].next:
                heapq.heappush(q, (dummy.next.val, dummy.next))

        return head.next
```

P3.valid sudoku https://leetcode.com/problems/valid-sudoku/

```
class Solution(object):
    def isValidSudoku(self, board):
        """
        :type board: List[List[str]]
        :rtype: bool
        """
        res = set()
        for row in range(len(board)):
            for col in range(len(board)):
                if board[row][col]=='.': continue

                num = board[row][col]
                if (row,'row', num) in res: return False
                else: res.add((row, 'row', num))

                if (col,'col',num) in res: return False
                else: res.add((col, 'col', num))

                if (row//3, col//3, num) in res: return False
                else: res.add((row//3, col//3,num))


        return True
```

p4. 给定数组，只有正数，使用＋和＊和（）。求最大值。使用DP解决。dp[j] = max(dp[m - 1] + dp[m][j], dp[m - 1] * dp[m][j]);

```
class Solution(object):
    def findmax(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        self.memo = {}
        res = self.dfs(nums, 0, len(nums)-1)
        return res

    def dfs(self, nums, l, r):

        if (l, r) in self.memo:
            return self.memo[l, r]

        if l == r:
            return nums[l]

        res = 0
        for i in range(l, r):
            left = self.dfs(nums, l, i)
            right = self.dfs(nums, i+1, r)
            tmp = max(left + right, left * right)
            res = max(res, tmp)

        self.memo[(l, r)] = res
        return res


nums = [1,2,3]
so = Solution()
a = so.findmax(nums)
print(a)
```

P5. 有序的rotated array 二分查找 https://leetcode.com/problems/find-minimum-in-rotated-sorted-array/

```
class Solution(object):
    def findMin(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        l, r = 0, len(nums)-1

        while l<r:

            mid = (l+r)//2

            if nums[mid]<nums[r]:
                r = mid
            else:
                l = mid+1

        return nums[l]

nums = [3, 1, 2]
so = Solution()
a = so.findMin(nums)
print(a)
```

如过有重复的数字的话，那么要判断mid是不是跟右边相等，相等的话,l-=1

```
class Solution(object):
    def findMin(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        l, r = 0, len(nums)-1
        while l<r:
            mid = (l+r)//2
            if nums[mid]<nums[r]:
                r = mid
            elif nums[mid]>nums[r]:
                l = mid+1
            else:
                r -= 1
        return nums[l]
```

P6. 实现一个trie leetcode

https://leetcode.com/problems/implement-trie-prefix-tree/

```
class node(object):
    def __init__(self):
        self.dic = {}
        self.isword = False

class Solution(object):

    def __init__(self):
        self.root = node()

    def buildTrie(self, words):
        for word in words:
            tmp = self.root
            for c in word:
                if c not in tmp.dic:
                    tmp.dic[c] = node()
                tmp = tmp.dic[c]

            tmp.isword = True
        return self.root



words = ['abc', 'ab', 'abcd']
so = Solution()
root = so.buildTrie(words)
print root.dic['a'].dic['b'].isword
```

P7. 用Binary Search Tree实现实现Map，包括以下操作： put(key, value), get(key), size()。其实就是实现BST的insert和get。然后让我写test case测试自己的代码。然后问了下用BST实现Map的好处，跟HashMap比的话

Delete实在不想写了。如果考到了，那我就跪地求饶好了。

```
class TreeNode(object):
    def __init__(self, val):
        self.val = val
        self.left = None
        self.right = None
```

```
class BST(object):

    def __init__(self):
        self.root = None

    def insert(self, root, val):
        """
        :type root: TreeNode
        :type val: int
        :rtype: None
        """
        node = TreeNode(val)
        if not root:
            self.root = node
        else:
            if root.val < val:
                if not root.right:
                    root.right = node
                else:
                    self.insert(root.right, val)
            else:
                if not root.left:
                    root.left = node
                else:
                    self.insert(root.left, val)

    def delete(self, val):
        """
        :type val: int
        :rtype: None
        """
        pass


    def search(self, root, val):
        """
        :type root: TreeNode
        :type val: int
        :rtype: bool
```

```
        """
        if not root:
            return False

        if root.val == val:
            return True

        if root.val > val:
            return self.search(root.left, val)
        else:
            return self.search(root.right, val)




bst = BST()
bst.insert(bst.root, 2)
bst.insert(bst.root, 1)
bst.insert(bst.root, 3)
bst.insert(bst.root, 4)
print bst.root.val
print bst.search(bst.root, 6)
```

今天心情好，再写一遍，感觉这个版本代码风格更好一点。

```
class Hash(object):

    def __init__(self):
        self.root = None

    def add(self, key, val):
        node = TreeNode(key, val)
        if not self.root:
            self.root = node
        else:
            self.addHelper(self.root, node)

    def addHelper(self, root, node):
        if node.key < root.key:
```

```
            if not root.left:
                root.left = node
            else:
                self.addHelper(root.left, node)
        elif node.key > root.key:
            if not root.right:
                root.right = node
                return
            else:
                self.addHelper(root.right, node)

    def get(self, key):
        return self.getHelper(self.root, key)


    def getHelper(self, root, key):
        if not root:
            return
        elif root.key==key:
            return root.val
        elif key<root.key:
            return self.getHelper(root.left, key)
        else:
            return self.getHelper(root.right, key)

h = Hash()
h.add(2,'haha')
h.add(1,'heihei')
h.add(3,'hoho')
print h.get(1)
```

P8. Given a list of integers: {4, 3, 1}; // assume no duplicates and a target sum: 5 print out all sequences of integers from the list that sum to target

https://leetcode.com/problems/combination-sum/

```
class Solution(object):
    def combinationSum(self, candidates, target):
        """
        :type candidates: List[int]
        :type target: int
        :rtype: List[List[int]]
        """
        res = []
        candidates.sort()
        self.dfs(candidates, target, [], res)
        return res

    def dfs(self, nums, target, path, res):
        if target < 0:
            return  # backtracking
        if target == 0:
            res.append(path)
            return
        for i in xrange(len(nums)):
            self.dfs(nums[i:], target-nums[i], path+[nums[i]], r
es)
```

P9 实现ArrayList。

```
class ArrayList(object):
    def __init__(self, capa):
        """
        :type capa: int
        :rtype: None
        """
        self.array = []
        self.capa = capa

    def add(self, val):
        """
        :type val: int
        :rtype: None
        """
        if len(self.array) == self.capa:
```

```
            self.capa = 2*self.capa
        self.array.append(val)
        return

    def delete(self, val):
        """
        :type val: int
        :rtype: None
        """
        if val in self.array:
            self.array.remove(val)
            if len(self.array) == self.capa/2:
                self.capa = self.capa/2
        return

a = ArrayList(5)
a.add(1)
a.add(2)
a.add(3)
print a.array, a.capa
for i in range(10):
    a.add(1)
print a.array, a.capa
for i in range(10):
    a.delete(1)
print a.array, a.capa
```

P10 问了一个字符串比较问题，说很多用户名都会重复，通过后面的数字来区分，但是在排序的时候严格按照字符串排序就会出现 abc10 排在 abc2 前面（因为‘1’比‘2’要小），但是事实上他们想要达到的效果是 abc10 排在 abc2 后面（10比2要大），于是让写一个字符串比较函数.

先写一个version number compare吧。

https://leetcode.com/problems/compare-version-numbers/

```
class Solution(object):
    def compareVersion(self, version1, version2):
        """
        :type version1: str
        :type version2: str
        :rtype: int
        """
        v1 = self.helper(version1)
        v2 = self.helper(version2)

        if v1==v2:
            return 0
        return 1 if v1>v2 else -1

    def helper(self, v):
        res = map(int, v.split('.'))
        while len(res)>1 and res[-1]==0:
            res.pop()
        return res

so = Solution()
v1 = '0.1'
v2 = '11.2'
print so.compareVersion(v1,v2)
```

再写一个字符串compare。

```
class Solution(object):
    def compareVersion(self, s1, s2):
        """
        :type s1: str
        :type s2: str
        :rtype: int
        """
        v1 = self.splitstr(s1)
        v2 = self.splitstr(s2)

        if v1==v2:
            return 0
        return 1 if v1>v2 else -1

    def splitstr(self, s):
        res = []
        index = 0
        while index<len(s):
            index += 1
            if s[index] in '0123456789':
                break
        return [s[:index],int(s[index:])]

s1 = 'abc10'
s2 = 'abc1'

so = Solution()
print so.compareVersion(s1, s2)
```

sc 11-20

P11: Trapping rain water。

https://leetcode.com/problems/trapping-rain-water/

思路：左右两个pointer 然后每次动矮的那一个。然后分别记录左右最高的

```
class Solution(object):
    def trap(self, height):
        """
        :type height: List[int]
        :rtype: int
        """
        if not height or len(height)<3:
            return 0

        l ,r = 0, len(height)-1
        left_h, right_h = height[l], height[r]

        res = 0
        while l<r:
            if left_h<right_h:
                res += max(0, left_h-height[l+1])
                l = l+1
                left_h = max(left_h, height[l])
            else:
                res += max(0, right_h-height[r-1])
                r = r-1
                right_h = max(right_h, height[r])
        return res
```

P12: 给定字符串A，B， 判断A中是否存在子串是B的anagram 。

https://leetcode.com/problems/valid-anagram/

leet有道简单版的。再上一道难的。

```
class Solution(object):
    def isAnagram(self, s, t):
        """
        :type s: str
        :type t: str
        :rtype: bool
        """
        return collections.Counter(s) == collections.Counter(t)
```

难的版本

```
class Solution(object):
    def isAnagram(self, a, b):
        """
        :type a: str
        :type b: str
        :rtype: bool
        """
        if len(b)> len(a):
            return False

        b = sorted(b)

        for i in range(len(a)-len(b)+1):
            if sorted(a[i:i+len(b)]) == b:
                return True

        return False

s = "sadaanagramsad"
t = "nagaram"
so = Solution()
print so.isAnagram(s,t)
```

更优化的版本用sliding window + hash来做。

```
import collections
class Solution(object):
```

```
    def isAnagram(self, a, b):
        """
        :type a: str
        :type b: str
        :rtype: bool
        """
        da = collections.defaultdict(int)
        db = collections.defaultdict(int)
        if len(b)> len(a):
            return False

        for c in b:
            db[c] += 1

        for i in range(len(b)):
            da[a[i]] += 1

        if da == db:
            return True

        for i in range(1, len(a) - len(b) + 1):
            da[a[i-1]] -= 1
            da[a[i+len(b)-1]] += 1
            if self.compare(db, da):
                return True

        return False

    def compare(self, d1, d2):
        for k in d1:
            if k not in d2 or d1[k]!=d2[k]:
                return False
        return True



s = "sadaanagramsad"
s = "aasdadnagaram"
t = "nagaram"
so = Solution()
```

```
print so.isAnagram(s,t)
```

P13. 第一轮是一个妹子, 好像是某硬件组的, 具体不记得了, 就记得妹子长得还不错 lol 我记得开始就问了下我背景和做过什麽, 看起来很有兴趣的样子。然后做题, 先是建一个event, 里面有start time和end time, 然后check这两个event有没有conflict, 各种if else, 然后升级, 给一个list of events, 直接sort他们。再升级, 建一个schedule, 给几个office, 问怎么样的solution才是optimal的。

leetcode meeting room I//meeting room II

https://leetcode.com/problems/meeting-rooms/

https://leetcode.com/problems/meeting-rooms-ii/

思路： 写了下难的那道。扫描线算法。然后存avaliable和房间数。

```
class Solution(object):
    def minMeetingRooms(self, intervals):
        """
        :type intervals: List[List[int]]
        :rtype: int
        """

        # put all the times together and sort them.
        times = []
        for i in intervals:
            times.append((i[0], 1))
            times.append((i[1], 0))
        times.sort()

        count = 0
        availiable = 0
        for t in times:
            # if we need a room
            # avaliabe room -1.
            if t[1]:
                availiable -= 1
                if availiable<0:
                    availiable += 1
                    count += 1
            # if end a meeting
            # avaliable + 1
            else:
                availiable += 1
        return count

intervals = [[0, 30],[5, 10],[15, 20]]
so = Solution()
print so.minMeetingRooms(intervals)
```

P14. 首先， 一个array的range是1-N，这个array里有1+N个数, 找出这个array里的duplicate。(ex. [1, 1, 3, 5]) HashMap 解决然后条件升级, 问不能用extra space. 想了一下, 先sort, 再找再升级, 不能改变array, 不能有extra space。

287 Find the Duplicate Number

https://leetcode.com/problems/find-the-duplicate-number/

最后跟linked list 找circle一样的。

```
class Solution(object):
    def findDuplicate(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        slow = nums[0]
        fast = nums[nums[0]]
        while (slow != fast):
            slow = nums[slow]
            fast = nums[nums[fast]]

        fast = 0;
        while (fast != slow):
            fast = nums[fast]
            slow = nums[slow]

        return slow
```

P15. binary tree to doubly linked list.

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

思路：这道题是数flatten成单项链表，面试题目相当于是加强版。就是preorder里面加点料。需要注意的地方是我们这个 self.pre = root这里很精妙，相当于在flatten右边树之前，把左树的最后一个节点给存起来了。

```
class Solution(object):
    pre = None
    def flatten(self, root):
        """
        :type root: TreeNode
        :rtype: void
        """
        if root:
            self.pre = root
            self.flatten(root.left)

            tmp = root.right
            root.right = root.left
            root.left = None
            self.pre.next = root.right

            self.flatten(tmp)
```

加强版：因为我们需要双向链表。所以要记录这个node的parent.因为Input变成了2个，所以我们需要新写一个helper function呢。

```
class TreeNode(object):
    def __init__(self, x):
        self.val = x
        self.left = None
        self.right = None

class Solution(object):
    pre = None
    def flatten(self, root):
        """
        :type root: TreeNode
        :rtype: void
        """
        self.helper(root, None)


    def helper(self, root, p):
```

```
        if root:

            self.pre = root
            self.helper(root.left, root)

            tmp = root.right
            root.right = root.left
            root.left = p
            self.pre.next = root.right

            self.helper(tmp, self.pre)


a = TreeNode(1)
b = TreeNode(2)
c = TreeNode(3)

a.left, a.right = b, c

so = Solution()
so.flatten(a)
print a.val, a.right.val, a.right.left.val
```

P16. search in rotated array 。

https://leetcode.com/problems/search-in-rotated-sorted-array/

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

思路：这道题跟一般的之前那个rotated sorted array找最小的值不一样，这个是要search。先找出带顺序的那一半，然后再判断在不在里面。(睡了一觉刚醒，起来做这个就是思路清晰，corner case毫无破绽！

```
class Solution(object):
    def search(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: int
        """
        if not nums:
            return -1

        l, r = 0, len(nums)-1

        while l<r:
            mid = (l+r)//2

            if nums[mid] == target:
                return mid

            if nums[mid]<nums[r]:
                if nums[mid]<target<=nums[r]:
                    l = mid+1
                else:
                    r = mid
            else:
                if nums[l]<=target<nums[mid]:
                    r = mid
                else:
                    l = mid+1

        return l if nums[l]==target else -1

nums = [4, 5, 6, 7, 0, 1, 2]
so = Solution()
print so.search(nums, 0)
```

再写个加强版，带重复的，其实就是多了一个如果mid == r 然后r--而已。 跟P5的思想一样。

```
class Solution(object):
    def search(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: bool
        """
        if not nums:
            return -1

        l, r = 0, len(nums)-1

        while l<r:
            mid = (l+r)//2

            if nums[mid] == target:
                return True

            if nums[mid]<nums[r]:
                if nums[mid]<target<=nums[r]:
                    l = mid+1
                else:
                    r = mid
            elif nums[mid]==nums[r]:
                r -= 1
            else:
                if nums[l]<=target<nums[mid]:
                    r = mid
                else:
                    l = mid+1

        return True if nums[l]==target else False
```

P17.上来面试官先介绍了一下他做的工作，人很nice，中国帅哥，然后让我介绍了我的projects 和 背景。之后出了一道题目： // Give a stream of numbers // Write a method to return the medium of the numbers you have received // void add(int num); // int getMedium();

https://leetcode.com/problems/find-median-from-data-stream/

```
import heapq
class MedianFinder:
    def __init__(self):
        """
        Initialize your data structure here.
        """
        self.min_heap = []
        self.max_heap = []


    def addNum(self, num):
        """
        Adds a num into the data structure.
        :type num: int
        :rtype: void
        """
        if len(self.min_heap)==len(self.max_heap):
            heapq.heappush(self.min_heap, num)
            heapq.heappush(self.max_heap, -heapq.heappop(self.mi
n_heap))
        else:
            heapq.heappush(self.max_heap, -num)
            heapq.heappush(self.min_heap, -heapq.heappop(self.ma
x_heap))


    def findMedian(self):
        """
        Returns the median of current data stream
        :rtype: float
        """
        if len(self.min_heap)==len(self.max_heap):
            return (self.min_heap[0] - self.max_heap[0])/2.0
        else:
            return -self.max_heap[0]
```

P18. 我一开始就想当然的用了个二维int矩阵做参数。 然后他memory overhead是什么，我就说我mn4。 问能不能省，我就说其实不需要用int存状态，用两个bit就可以了。 又问如果还要再省一点呢？我就说，如果要操作的点不太多的话，那就不要

存矩阵了，就存存要修改的点的坐标。 然后又问怎么做，我才意识到他是想说参数其实不必用矩阵，就用一个list of coordinates，然后就说如果是sparse矩阵的话就只扫一下live的点，然后看看它和它neighbor在下一轮是什么状态就好了。 但是这个时候时间已经差不多了，就没有再写实现。

https://leetcode.com/problems/game-of-life/

前天才写了一遍，今天就不写了。

```
class Solution(object):
    def gameOfLife(self, board):
        """
        :type board: List[List[int]]
        :rtype: void Do not return anything, modify board in-pla
ce instead.
        """
        if not board or not board[0]:
            return

        for i in range(len(board)):
            for j in range(len(board[0])):
                cnt = 0
                for m, n in [[i-1, j],[i+1, j], [i, j-1], [i, j+1
], [i-1, j-1], [i+1, j+1], [i+1, j-1], [i-1, j+1]]:
                    if 0<=m<len(board) and 0<=n<len(board[0]):
                        if board[m][n]&1 ==1:
                            cnt += 1

                if board[i][j]==1:
                    if 3<=board[i][j]+cnt<=4:
                        board[i][j] = board[i][j]^2
                else:
                    if cnt==3:
                        board[i][j] = board[i][j]^2
                print cnt

        for i in range(len(board)):
            for j in range(len(board[0])):
                board[i][j] = board[i][j]>>1
```

P19. 1. permutation i。leetcode原题，没有任何改变。

https://leetcode.com/problems/permutations-ii/

Permutation I 太简单，我们还是写permutaiont II好了

```
class Solution(object):
    def permuteUnique(self, nums):
        """
        :type nums: List[int]
        :rtype: List[List[int]]
        """
        res=[]
        self.size=len(nums)
        nums.sort()
        self.dfs(nums,[],res)
        return res

    def dfs(self,nums,cur,res):
        if len(cur)==self.size:
            res.append(cur)
            return
        for i in range(len(nums)):
            if i and nums[i]==nums[i-1]:
                continue
            self.dfs(nums[:i]+nums[i+1:], cur+[nums[i]], res)

nums = [1,2,3]
so = Solution()
print so.permuteUnique(nums)
```

P20. 24点游戏。给你几个数字，判断他们做加减乘除运算是否可以得到24，顺序可以是任意的。dfs搜索搞定。。。但是这里要注意一些细节，每次计算得到新的数之后最好加入数组做下一次搜索，不然容易出错。

```
class Solution(object):
    def game24(self, nums):
        """
        :type nums: List[int]
```

```
        :rtype: bool
        """
        return self.dfs(nums, 0, '')


    def dfs(self, nums, cur, path):
        """
        :type nums: List[int]
        :type cur: int
        :type path: str
        :rtype: bool
        """

        if not nums and cur==24:
            print path
            return True


        for i in range(len(nums)):
            if self.dfs(nums[:i]+nums[i+1:], cur+nums[i], path+'
+'+str(nums[i])):
                return True

            if self.dfs(nums[:i]+nums[i+1:], cur-nums[i], path+'
-'+str(nums[i])):
                return True

            if self.dfs(nums[:i]+nums[i+1:], cur*nums[i], path+'
*'+str(nums[i])):
                return True

            if self.dfs(nums[:i]+nums[i+1:], cur/nums[i], path+'
/'+str(nums[i])):
                return True

        return False



nums = [4,2,1,3]
```

```
so = Solution()
print so.game24(nums)
```

sc 21-30

P21. 24点游戏。给你几个数字，判断他们做加减乘除运算是否可以得到24，顺序可以是任意的。dfs搜索搞定。。。但是这里要注意一些细节，每次计算得到新的数之后最好加入数组做下一次搜索，不然容易出错。

思路：就是DFS，这个问题好有趣。我加入了Path 来打印出结果，方便理解。

```
class Solution(object):
    def game24(self, nums):
        """
        :type nums: List[int]
        :rtype: bool
        """
        return self.dfs(nums, '')

    def dfs(self, nums, path):
        """
        :type nums: List[int]
        :type cur: int
        :type path: str
        :rtype: bool
        """

        if len(nums)==1 and nums[0]==24:
            print path
            return True


        for i in range(len(nums)-1):
            for j in range(i+1,len(nums)):

                tmp = nums[i]+nums[j]
                if self.dfs(nums[:i]+nums[i+1:j]+nums[j+1:] + [t
mp], path + ' ' + str(nums[i])+'+'+str(nums[j])):
                    return True

                tmp = nums[i]-nums[j]
```

```
                if self.dfs(nums[:i]+nums[i+1:j]+nums[j+1:] + [t
mp], path + ' ' + str(nums[i])+'-'+str(nums[j])):
                    return True

                tmp = nums[i]*nums[j]
                if self.dfs(nums[:i]+nums[i+1:j]+nums[j+1:] + [t
mp], path + ' ' + str(nums[i])+'*'+str(nums[j])):
                    return True

                tmp = nums[i]/nums[j]
                if self.dfs(nums[:i]+nums[i+1:j]+nums[j+1:] + [t
mp], path + ' ' + str(nums[i])+'/'+str(nums[j])):
                    return True

        return False

nums = [1,3,1,5]
so = Solution()
print so.game24(nums)
```

P22. 一个是 word break 2，给个字典，如何break Ilovenyc .

```
class Solution(object):
    def wordBreak(self, s, wordDict):
        """
        :type s: str
        :type wordDict: Set[str]
        :rtype: List[str]
        """
        memo = {len(s): ['']}
        def sentences(i):
            if i not in memo:
                memo[i] = [s[i:j] + (tail and ' ' + tail)
                           for j in range(i+1, len(s)+1)
                           if s[i:j] in wordDict
                           for tail in sentences(j)]
            return memo[i]
        return sentences(0)

    def dfs(s, wordDict, cur, res):
        if not s: return [None]
        if s in wordDict:
            return res.append((cur + ' ' + s).strip())

        for word in wordDict:
            l = len(word)
            if word == s[:l]:
                dfs(s[l:], wordDict, cur+' '+ s[:l], res)
```

P23. 判断一个图是不是 bipartite: https://en.wikipedia.org/wiki/Bipartite_graph

思路：DFS,BFS都能解，到下一个node的时候换另一个颜色。然后check下访问过的node的颜色就行了。

```
class Solution():
    def isBipartite(self, graph):
        """
        :type graph: dict[List]
        :rtype: bool
        """

        s = [1]
        visited = {1:0}

        while s:
            tmp = s.pop()
            children = graph[tmp]
            color = visited[tmp]
            for c in children:
                if c in visited:
                    if visited[c] + color!=1:
                        return False
                else:
                    s.append(c)
                    visited[c] = 1 if color==0 else 0

        print visited
        return True

graph = {1:[2,3], 4:[2,3], 2:[1,4], 3:[1,4]}
so = Solution()
print so.isBipartite(graph)
```

P24. 一面国人小哥，人非常好。题目是面经里提过的 toddle() 和 getToddleList()，具体可以看这个帖子:

http://www.1point3acres.com/bbs/thread-160016-1-1.html

```
class Node():
    def __init__(self, val):
        self.val = val
        self.left = None
        self.right = None
```

```
class Solution():

    def __init__(self):
        self.root = None
        self.tail = None
        self.table = {}

    def toggle(self, val):
        if val not in self.table:
            node = Node(val)
            self.table[val] = node

            if not self.root:
                self.root = node
                self.tail = node
            else:
                node.left = self.tail
                self.tail.right = node
                self.tail = node
        else:
            node = self.table[val]
            pre = node.left
            cur = node.right
            if pre:
                pre.right = cur
            if cur:
                cur.left = pre

    def getList(self):
        res = []
        node = self.root
        while node:
            res.append(node.val)
            node = node.right
        return res


so = Solution()
so.toggle(1)
```

```
so.toggle(2)
so.toggle(3)
so.toggle(2)
print so.getList()
```

P25. 3sum

https://leetcode.com/problems/3sum/

```
class Solution():
    def threeSum(self, nums):
        """
        :type nums: List[int]
        :rtype: List[List[int]]
        """

        res = []
        nums.sort()

        for i in range(len(nums)-2):
            if i>0 and nums[i-1]==nums[i]:
                continue
            a = nums[i]
            l, r = i+1, len(nums)-1
            while l<r:
                if a + nums[l] + nums[r] == 0:
                    res.append([a, nums[l], nums[r]])
                    while l < r and nums[l] == nums[l+1]:
                        l += 1
                    while l < r and nums[r] == nums[r-1]:
                        r -= 1
                    l += 1
                    r -= 1
                elif a + nums[l] + nums[r]<0:
                    l += 1
                else:
                    r -= 1
        return res

nums = [-1, 0, 1, 2, -1, -4]
so = Solution()
print so.threeSum(nums)
```

P26. 题目是用前序和中序数组重建二叉树，LC 原题。

https://leetcode.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal/

https://leetcode.com/problems/construct-binary-tree-from-inorder-and-postorder-traversal/

思路: 就是用Inorder来做参照，如果inorder没了，说明不能继续recurse下去了。

```
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution(object):
    def buildTree(self, preorder, inorder):
        """
        :type preorder: List[int]
        :type inorder: List[int]
        :rtype: TreeNode
        """
        if inorder:
            num = preorder.pop(0)
            root = TreeNode(num)
            index = inorder.index(num)
            root.left = self.buildTree(preorder, inorder[:index]
)
            root.right = self.buildTree(preorder, inorder[index+1
:])
            return root
```

再来一个

```
class Solution(object):
    def buildTree(self, inorder, postorder):
        """
        :type inorder: List[int]
        :type postorder: List[int]
        :rtype: TreeNode
        """
        if inorder:
            num = postorder.pop()
            root = TreeNode(num)
            index = inorder.index(num)
            root.right = self.buildTree(inorder[index+1:], posto
rder)
            root.left = self.buildTree(inorder[:index], postorde
r)
            return root
```

P27. 分享个楼主惨痛的snapchat电面。OA就不说了，简单的数独问题， 做完当天下午约电面，电面是国人大叔。在面之前也是跟其他人一样，把地里的题都刷了一遍，什么bigInteger， DP， Zigzag print等等。 结果电面不是这些啊，并且楼主成功被自己蠢哭，第一题就没搞定。干货： 一个dictionary里面存String单词，有millions这么多，给一个char array， 让返回所有的单词，这些单词的所有字母必须在char array里面。比如：char array = {a,c,a,r,t}, dictionary= {"a","aa","cat","cats","aaa","cc"} 那么你返回的list里面包含的单词应该是：a, aa, cat这类的。其实我并不知道为啥aa也算单词， 不包含的像cats， aaa, cc这样的。

思路：刚开始我说：将char array处理成map这样形式，然后去扫dictionary里面单词，如果包含在map里面就加入list，不包含就不加。大叔说，可以，但是复杂度太高，有没有优化，想两三分钟，估计看我没动静，问我看看需不需要用什么data structure， 突然想到用trie， 就说了。 果然是用trie， 然后又让我说了下用trie怎么去遍历。

悲剧就此时发生，楼主对trie里面的成员变量理解的不够深刻。想DFS去遍历，但是不会写呜呜呜呜呜~哎~都别问我为什么，因为不知道怎么写循环，而且那会没想清楚就开始写代码，写到一半发现，不会用trie里的成员变量呢，就慌了。一个小时硬是没做出来，我也是醉醉的。面试结束，自己静下心来，20分钟竟然写出来了，我擦擦擦擦擦擦。

我的思路: 我怎么觉得这道题不应该用Trie啊

```
class Solution(object):
    def validWord(self, charArray, words):
        """
        :type charArray: List[char]
        :type words: List[str]
        :rtype: List[str]
        """
        dic = set(charArray)
        res = []
        for w in words:
            tmp = set(w)
            if len(tmp| dic)==len(dic):
                res.append(w)
        return res

charArray = ['a', 'c', 'a', 'r', 't']
words= ["a","aa","cat","cats","aaa","cc"]

so = Solution()
a = so.validWord(charArray, words)
print a
```

P28.We obtained a log file containing runtime information about all threads and mutex locks of a user program. The log file contains N lines of triplets about threads acquiring or releasing mutex locks. The format of the file is: The first line contains an integer N indicating how many more lines are in the file. Each of the following N lines contains 3 numbers separated by space. The first number is an integer representing thread_id (starting from 1). The second number is either 0 (acquiring) or 1 (releasing). The third number is an integer representing mutex_id (starting from 1). Now we want you to write a detector program that reads the logs line by line and output the line number of the trace where a deadlock happens. If there is no deadlock after reading all log traces, output 0.

Example:

```
4
1 0 1
2 0 2
2 0 1
1 0 2.
```

Output:

```
4
```

中文描述:其实就是维护一个图，然后每新读一个record就找一次有没有环的说。因为mutex如果有环才会死锁。中间那个数为0就加一条边，中间那个数为1就减少一条边。代码如下。还是比较简单的。

```
import collections
class Solution(object):
    def lockDetector(self, log):
        """
        :type charArray: List[char]
        :type words: List[str]
        :rtype: List[str]
        """
        graph = collections.defaultdict(set)
        for i in range(len(log)):
            [a, b, c] = log[i]
            if b:
                graph[a].remove(c)
            else:
                if a!=c:
                    graph[a].add(c)
                self.visited = set()
                if self.findCircle(a, graph):
                    return i+1
        return -1

    def findCircle(self, a, graph):
        s = [a]

        while s:
            tmp = s.pop()
            children = graph[tmp]
            if tmp not in self.visited:
                self.visited.add(tmp)
            else:
                return True
            for c in children:
                s.append(c)
        return False

log = [[1,0,1], [2,0,2], [2,0,1], [1,0,2]]
so = Solution()
a = so.lockDetector(log)
print(a)
```

P29 上来说implement big number library的时候 我吓了一身冷汗 然后发现是 two strings 加减乘除 才放心开始写 但是要在codepad上run 发现自己写zero-bug code 的能力还是不太行。。。 改了改bug 写了unit test 时间就差不多没了。。所以没写完全部operation 哭。。。

```
class Solution(object):
    def add(self, num1, num2):
        """
        :type num1: str
        :type num2: str
        :rtype: str
        """
        num1 = list(num1)[::-1]
        num2 = list(num2)[::-1]

        if len(num1)<len(num2):
            num1, num2 = num2, num1

        carry = 0
        index = 0
        while index<len(num1):
            n2 = int(num2[index]) if index<len(num2) else 0
            tmp = n2 + int(num1[index]) + carry
            cur = tmp%10
            carry = tmp//10
            num1[index] = str(cur)
            index += 1

        if carry: num1.append('1')

        return ''.join(num1[::-1])

    def multiply(self, num1, num2):
        """
        :type num1: str
        :type num2: str
        :rtype: str
        """
        num1 = num1[::-1]
```

```
        num2 = num2[::-1]

        res = [0]*(len(num1) + len(num2))
        for i in range(len(num2)):
            for j in range(len(num1)):
                tmp = int(num2[i])*int(num1[j])
                res[j+i] += tmp
                res[j+i+1] += res[j+i]//10
                res[j+i] = res[j+i]%10

        while len(res)>1 and res[-1]==0:
            res.pop()

        return ''.join(map(str,res))[::-1]

num1 = '124'
num2 = '5679'
so = Solution()
a = so.add(num1, num2)
print(a)
a = so.multiply(num1, num2)
print(a)
```

P30.准备的面经题一道没有。面的是secretSanta, 简单的说就是，每个人都必须给别人礼物，每个人都必须收礼物。自己不能给自己礼物。 string[] name = {mary, alice, mike}; output： mary -> mike mike -> alice, alice -> marry 错误情况是：marry ->mike, mike -> marry, alice ->

```
class Solution():
    def secretSanta(self, names):
        res = []
        self.dfs(names, names, [], res)
        return res


    def dfs(self, giver, receiver, cur, res):

        if not giver:
            cur.sort()
            res.append(cur)

        for j in range(len(receiver)):
            if giver[0]!=receiver[j]:
                new_giver = giver[1:]
                new_receiver = receiver[:j] + receiver[j+1:]
                record = giver[0] + '->' + receiver[j]
                self.dfs(new_giver, new_receiver, cur + [record]
, res)


names =  ['A', 'B', 'C']
so = Solution()
a = so.secretSanta(names)
print a
print len(a)
```

sc 31-40

P31 给你一个立着的棋盘，然后每一次落子都会因为重力掉到最低的没有棋子的那一格。如果vertical, horizontal, diagonal有三个同一选手的子（就是有一个三连），就算胜利。

要求写一个isvalid来检查这个棋盘是不是valid的（即没有悬空的子的情况。）

然后要求写一个nextMove。来输出下一子的位置。 nextMove要尽量保证你的对手不能赢

例子如下，0代表空着的位置，1代表1号选手下的位置，2代表2号选手下的位置。

```
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 2 | 0 |
| 0 | 2 | 1 | 1 | 2 |
---------------------
```

思路：其实，tic-tac-toe的带重力加强版.valid就是每一列挨个检查有没有悬空子的情况。nextMove就是我把对手每个可能的下一步都判断一遍，如果那一步能赢，我就要堵住。由于带重力，所以每一列只可能有一个位置。还是很好检查的。nextP这个array代表了每一列能下的位置。我只写了判断vertical horizontal的，diagonal的一样的写法。懒得写了哈。

```
class Connect3(object):
    def __init__(self, grid):
        self.grid = grid
        self.nextP = [0]*len(grid[0])

    def isValid(self):
        """
        :rtype: Bool
        """
        for j in range(len(grid[0])):
            i = 0
            # for every column down to the first not 0 point
```

```
            while i<len(self.grid) and self.grid[i][j]==0:
                i += 1
            self.nextP[j] = i-1
            # then check if there is 0 in side.
            while i<len(self.grid):
                if grid[i][j]==0:
                    return False
                i += 1
        return True

    def nextMove(self, player):
        """
        :type player: Int
        :rtype: List[List[int]]
        """
        opponent = 1 if player==2 else 2
        res = []
        for i in range(len(self.nextP)):
            r, c = self.nextP[i], i
            if self.checkWin(r, c, opponent):
                res.append([r,c])
        return res

    def checkWin(self, i, j, player):
        """
        :type i: Int
        :type j: Int
        :type player: Int
        :rtype: Bool
        """
        # check vertical
        up, down = i+1, i-1
        count = 0
        while 0<=up<len(self.grid):
            if self.grid[up][j] == player:
                count += 1
            up += 1
        while 0<=down<len(self.grid):
            if self.grid[down][j] == player:
                count += 1
```

```
            down -= 1
        if count == 2:
            return True
        # check horizontal
        left, right = j-1, j+1
        count = 0
        while 0<=left<len(self.grid[0]):
            if self.grid[i][left] == player:
                count += 1
            left -= 1
        while 0<=right<len(self.grid[0]):
            if self.grid[i][right] == player:
                count += 1
            right += 1
        if count == 2:
            return True

grid = [[0,0,0],[0,1,0],[0,1,0]]
so = Connect3(grid)
an = so.isValid()
print an
print so.nextP
res = so.nextMove(2)
print res
```

sc onsite

P1 找Amicable Number Pairs 就是 数A 的所有因数(包括1，不包括A) 之和 等于 B。B的所有因数之和又刚好为A。 且 A != B. 求[1, N] 中所有符合条件的pair

思路：先用一个array把每个数的因数和全部存起来。然后再遍历这个array找pair就行了。i = record[record[i]]就可以了。

```
class Solution(object):
    def amiPair(self, n):
        record = [0]*(n+1)
        res = []
        # first we record sum
        for i in range(1, n):
            record[i] = sum(e for e in range(1,i//2+1) if i%e==0
)
        # then we find the pair
        for i in range(1, n):
            if i<record[i] and record[i]<n and i == record[recor
d[i]]:
                res.append([i, record[i]])
        return res

n = 1000
so = Solution()
a = so.amiPair(n)
print a
```

P2 coding题目是一个矩阵里某些位置有墙，给一个出发点及目的地，求最短距离。 如果只需能否通过，那就DFS。要计算距离，那就BFS。 follwup是现在墙可以打破，没打破一个cost为1，求cost最小的路线。 需要把每个node再加个成员变量cost，在终点取最小的cost，注意中途遇到访问过而且cost小的才不加。一定要画画图。

```
class Solution(object):
    def findPath(self, matrix, start, end):
        """
        :type matrix: List[List[int]]
        :type start: List[int]
        :type end: List[int]
        :rtype: Bool
        """
        if not matrix or not matrix[0]:
            return False

        s = [start]
        visited = set()

        while s:
            tmp = s.pop()
            for dp in [(1, 0), (-1, 0), (0, 1), (0, -1)]:
                i, j = tmp[0]+dp[0],tmp[1]+dp[1]
                print i,j
                if [i, j] == end:
                    return True
                if 0<=i<len(matrix) and 0<=j<len(matrix[0]) and
(i,j) not in visited and matrix[i][j]!=1:
                    visited.add((i,j))
                    s.append([i,j])
        return False

so = Solution()
matrix=[[0,1],[1,0]]
s, e = [0,0], [1,1]
a = so.findPath(matrix, s, e)
print a
```

P3 有一堆数, 比如{2,3,5,8}. 每次带2个过去, 比如我选了3和5, cost是最大的那个值. 一旦过去, 还要带回来一个数. 比如带回3, 花销是3. 所以这一趟把5运了过去, 总花销是3+5. 求找出最小的花销运过去所有数字.

思路：天下武功，唯暴力不破。直接暴力dfs吧。优化的解法如下：
http://blog.csdn.net/morewindows/article/details/7481851 不过提供面经这个哥们暴力dfs也拿到offer了。所以。还是装作不会最优解吧。科科

```
class Solution(object):
    def lowCost(self, nums):
        self.res = float('inf')
        self.dfs(nums, [],[], 0)
        return self.res

    def dfs(self, nums, end, path, cost):

        if len(nums)==2:
            if cost+max(nums)<self.res:
                self.res = cost+max(nums)

        for i in range(len(nums)-1):
            for j in range(i+1, len(nums)):
                # get 2 num from nums
                next_nums = nums[:]
                next_end = end[:]
                next_cost = cost
                a, b = next_nums[i], next_nums[j]
                next_nums.remove(a)
                next_nums.remove(b)
                next_cost += max(a,b)
                next_end += [a,b]
                # get 1 from end
                c = min(next_end)
                next_cost += c
                next_end.remove(c)
                next_nums.append(c)
                self.dfs(next_nums, next_end, path+[[a,b,c]], ne
xt_cost)

nums = [2,3,5,8]
so = Solution()
a = so.lowCost(nums)
print a
```

p4. 给定数组，只有正数，使用＋和＊和（）。求最大值。使用DP解决。dp[j] = max(dp[m - 1] + dp[m][j], dp[m - 1] * dp[m][j]);

```
class Solution(object):
    def findmax(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        self.memo = {}
        res = self.dfs(nums, 0, len(nums)-1)
        return res

    def dfs(self, nums, l, r):

        if (l, r) in self.memo:
            return self.memo[l, r]

        if l == r:
            return nums[l]

        res = 0
        for i in range(l, r):
            left = self.dfs(nums, l, i)
            right = self.dfs(nums, i+1, r)
            tmp = max(left + right, left * right)
            res = max(res, tmp)

        self.memo[(l, r)] = res
        return res


nums = [1,2,3]
so = Solution()
a = so.findmax(nums)
print(a)
```

P5 3d grid，每个点有高度，给起点和终点，求一个直升机起点到终点高度和最小的路径，注意一点是直升机只能上升不能下降

思路：这道题用我写的 Dijkstra's Algorithm 就行了。你把高度看成cost。想当与就是求（0，0）到（2，2）的最短路径嘛。这只是最简单的Dijkstra，里面的那个Q可以用ordered dict来实现。或者用hashed Priority Queue来优化。面试的时候我觉得写成这样就差不多可以过了。

```
class Solution(object):
    def flyChicken(self, m, start, end):
        D = {}
        Q = {}

        Q[start] = 0

        while Q:
            v = self.get_min(Q)
            D[v] = Q[v]
            for dp in [(-1,0), (1,0), (0,1), (0,-1)]:
                i, j = v[0]+dp[0], v[1]+dp[1]
                if 0<=i<len(m) and 0<=j<len(m[0]):
                    dist = Q[v] + m[i][j]
                    if (i, j) in D:
                        if dist<D[(i,j)]:
                            raise ValueError
                    else:
                        if (i,j) not in Q or dist<Q[(i,j)]:
                            Q[(i,j)] = dist
            del Q[v]

        return D[(2,2)]

    def get_min(self, Q):
        m = (None, float('inf'))
        for k in Q:
            if Q[k]<m[1]:
                m = (k, Q[k])
        return m[0]

# example, CLR p.528
Matrix = [[0,1,2],
          [1,2,3],
          [1,1,4]]

so = Solution()
print so.flyChicken(Matrix, (0,0), (2,2))
```

P6 E.g. storeBadIps(["7.0.0.0/8", "10.3.0.0/16", "6.7.8.134/32"]) In the example "7.0.0.0/8", "7.0.0.0" is ip ranges from 0.0.0.0 to 255.255.255.255, "8" is how many bits from the beginning it needs to match. in this example any "7.x.x.x" would be a bad ip. In the example "10.3.0.0/16", any "10.3.x.x" would be a bad ip. In the example "6.7.8.134/32", only exact match "6.7.8.134" would be a bad ip.

思路：这题没什么好说的呀，就是建立trie嘛，然后新来的ip.就找在不在trie里面。

```
class Node(object):
    def __init__(self):
        self.children = {}
        self.isEnd = False

class Solution(object):
    def __init__(self, list):
        self.root = Node()
        for ip in list:
            [ip, length] = ip.split('/')
            ip = ip.split('.')
            head = self.root
            for i in range(0, int(length), 8):
                if ip[i//8] not in head.children:
                    head.children[ip[i//8]] = Node()
                head = head.children[ip[i//8]]
            head.isEnd = True

    def badIp(self, ip):
        ip = ip.split('.')
        head = self.root
        for i in range(0, 32, 8):
            if ip[i//8] not in head.children:
                return False
            head = head.children[ip[i//8]]
            if head.isEnd==True:
                return True
        return False

l = ["7.0.0.0/8", "10.3.0.0/16", "6.7.8.134/32"]
so = Solution(l)
ans = so.badIp("6.7.8.x")
print ans
```

P7. N 个person和N 个bike在一个2维度数组里配对，一个person配对一个bike，要求所有配对的距离加起来和最小，面试官只要求讲思路。并提示greedy算法， 这题目我只给出了bruteforce的枚举法思路，时间复杂度是O(n!)，面试官说还有greedy

算法可以优化成O(n^2）复杂度，但事后我觉得此题用greedy 算法只能近似求解，面试官给出的解法根本就无法求解全局最优解嘛。 不知道有没有高手可以用O(n^2)解出来，打我的脸。

思路：我觉得其实面试官就想要一个greedy的次优解法。然后我就按照面试官的要求写了一个。

```
import math
class Solution(object):
    def assignBike(self, p, b):
        ps = set(p)
        bs = set(b)

        dis_list = []
        for i in p:
            for j in b:
                dis = math.sqrt((i[0]-j[0])**2+(i[1]-j[1])**2)
                dis_list.append((dis,i,j))
        dis_list.sort(key=lambda x: x[0])

        cost = 0
        res = []
        while ps and bs:
            [dis, i, j] = dis_list.pop(0)
            if i in ps and j in bs:
                ps.remove(i)
                bs.remove(j)
                cost += dis
                res.append((i,j))
        return cost, res

p = [(1,1), (1,2)]
b = [(2,1), (2,2)]
so = Solution()
ans = so.assignBike(p, b)
print ans
```

p8 小哥说话挺慢的，先让自我介绍，问了why snapchat, 实习经历里面最exciting的事是啥以及实习过程中有没有不开心的事 题目问了XML parser, LZ不熟悉XML，就改成了HTML Parser, 本质上一样 输入是一个tokenizer对象，可以调用其getNextToken()函数获得下一个token, token结构包括name和tag，name就是具体的文字，tag有open,close,text三种，让把所有的token建成一棵树 比如：

```
<html>
    <user>
        <id>aa</id>
        <meta>bb</meta>
    </user>
</html>
```

得到的token就是 ("html","open") ("user","open") ("id","open") ("aa","text") ("id","close") ("meta","open") ("bb","text") ("meta","close") ("user","close") ("html","close")

思路：是不是有点过于简单了，简直是放水嘛。还是还是我想错了。

```
class Node(object):
    def __init__(self):
        self.text = None
        self.children = {}

class Solution(object):
    def xmlParser(self, tokens):
        root = Node()
        s = [root]
        for t in tokens:
            print t
            if t[1]=='open':
                tmp = Node()
                s[-1].children[t[0]] = tmp
                s.append(tmp)
            elif t[1]=='close':
                s.pop()
            elif t[1]=='text':
                s[-1].text = t[0]
        return root
```

P9 一个2D平面有一堆雷达（雷达有x, y坐标，以及能探测到的范围r半径）然后又一辆小车要从y=0和y=1的区间里面通过并且不能被雷达探测到。让写一个function看小车能不能通过 我用dfs写了一个。判断2D grid能不能从一边到另外一边的。 下次用union find写一下。

```
class Solution(object):
    def canPass(self, sensor, matrix):
        """
        :type sensor: List[tuple]
        :type matrix: List[List[int]]
        :rtype: Bool
        """

        # process the grid
        for s in sensor:
            x, y, r = s[0], s[1], s[2]
            for i in range(x-r, x+r+1):
```

```
                for j in range(y-r, y+r+1):
                    if 0<=i<len(matrix[0]) and 0<=j<len(matrix):
                        dist = ((x-i)**2 + (y-j)**2)**(1.0/2)
                        if dist<=r:
                            matrix[i][j]=1

        # build the start points and end points
        s, end = [], []
        for j in range(len(matrix[0])):
            if matrix[0][j]==0:
                s.append((0,j))
            if matrix[len(matrix)-1][j]==0:
                end.append((len(matrix)-1,j))

        # dfs
        visited = set(s)
        while s:
            tmp = s.pop()
            if tmp in end:
                return True
            for dp in [(-1,0), (1,0), (0,1), (0,-1)]:
                i, j = tmp[0]+dp[0], tmp[1]+dp[1]
                if 0<=i<len(matrix[0]) and 0<=j<len(matrix)\
                and matrix[i][j]==0 and (i,j) not in visited:
                    visited.add((i,j))
                    s.append((i,j))
        return False


matrix = [[0]*4 for _ in range(4)]
sensor = [(1,0,1), (3,3,1)]
so = Solution()
ans = so.canPass(sensor, matrix)
print ans
```

P10 题目： ternary expression paser （三目运算符）， input是一个string，其中 'T' 表示true， 'F' 表示false， 要求返回最后运算的结果。

乍一看题目很直接， bool expression ? first result : second result ，但实际上我们通常都用的都是非常简单的形式，但每一部分都可以无限嵌套。例如： T ? T : F ? T : 1 : 2 : F ? 3 : 4; 原本妹子没让我考虑bool expression部分也嵌套的情况，结果我本着把问题分析清楚的原则，成功把问题弄的复杂了，她后来觉得这可以作为follow up。。。.1point3acres缃 说了两三个简单的solution，然后举例发现有问题。在妹子的提示下，写出了“不考虑 bool expression”的情况，但是因为已经说出口了，妹子指出我还需要继续考虑。最终时间不够，没有运行代码。 过后发现，确实应该多在纸上找一下规律，理清逻辑在写解法。string的变式题对我来说比较tricky，完全在脑子里思考容易变成一团浆糊。。。。妹子对我我说，没事她比较看重逻辑。我心里只能说谢谢你的安慰。。T.T

```
class Solution(object):
    def boolExpression(self, s):
        ops=[]
        nums=[]

        for i in s:
            if i in '0123456789':
                nums.append(i)
                self.applyOp(ops, nums)
            elif i in 'TF':
                nums.append(i)
            elif i in ':?':
                ops.append(i)
        return nums[-1]


    def applyOp(self, ops, nums):
        while len(ops)>1 and ops[-1]==':' and ops[-2]=='?':
            x, y = nums.pop(), nums.pop()
            b = nums.pop()
            res = y if b=='T' else x
            nums.append(res)
            ops.pop()
            ops.pop()



s = 'T?F?1:3:2'
so = Solution()
a = so.boolExpression(s)
print(a)
```

P10 unique paths I + II

https://leetcode.com/problems/unique-paths/

原题没什么好说的。

```
class Solution(object):
    def uniquePaths(self, m, n):
        """
        :type m: int
        :type n: int
        :rtype: int
        """
        dp = [[0]*n for _ in range(m)]

        for i in range(m):
            for j in range(n):
                if i==0 or j==0:
                    dp[i][j]=1
                else:
                    dp[i][j] = dp[i-1][j]+dp[i][j-1]

        return dp[-1][-1]
```

2

```
class Solution(object):
    def uniquePathsWithObstacles(self, obstacleGrid):
        """
        :type obstacleGrid: List[List[int]]
        :rtype: int
        """
        if not obstacleGrid or not obstacleGrid[0]:return -1
        m, n = len(obstacleGrid), len(obstacleGrid[0])
        dp = [[0]*n for _ in range(m)]

        for i in range(m):
            for j in range(n):
                if obstacleGrid[i][j]==1:
                    dp[i][j]==0
                else:
                    if i==0 and j==0:
                        dp[i][j]=1
                    elif i == 0:
                        dp[i][j] = dp[i][j-1]
                    elif j ==0:
                        dp[i][j] = dp[i-1][j]
                    else:
                        dp[i][j] = dp[i-1][j] + dp[i][j-1]

        return dp[-1][-1]
```

sc onsite 11-20

P11 Alien dict

具体参考 Topo Sort 其实可以试下另外一种写法的，有点懒，就算了哈。

```
import collections
class Solution(object):
    def alienOrder(self, words):
        """
        :type words: List[str]
        :rtype: str
        """

        # build graph
        indegree = collections.defaultdict(set)
        outdegree = collections.defaultdict(set)
        for i in range(len(words)-1):
            x, y = words[i], words[i+1]
            index = 0
            while index<min(len(x),len(y)) and x[index]==y[index
]:
                index +=1
            if index<min(len(x),len(y)):
                indegree[y[index]].add(x[index])
                outdegree[x[index]].add(y[index])

        s = [k for k in outdegree if k not in indegree]
        res = []
        while s:
            tmp = s.pop()
            res.append(tmp)
            chidren = outdegree[tmp]
            for c in chidren:
                indegree[c].remove(tmp)
                if not indegree[c]:
                    s.append(c)
```

```
        chars = set(''.join(words))
        if len(res) == len(chars):
            return ''.join(res)
        else:
            return ''

words = [
  "wrt",
  "wrf",
  "er",
  "ett",
  "rftt"
]
so = Solution()
so.alienOrder(words)
```

P12 ugly number 写一个臭数字全家福 哈哈哈

https://leetcode.com/problems/ugly-number/

```
class Solution(object):
    def isUgly(self, num):
        """
        :type num: int
        :rtype: bool
        """
        for p in 2,3,5:
            while num!=0 and num%p==0:
                num = num//p
        return num==1

so = Solution()
a = so.isUgly(6)
print(a)
```

https://leetcode.com/problems/ugly-number-ii/

```
class Solution(object):
    def nthUglyNumber(self, n):
        """
        :type n: int
        :rtype: int
        """

        res = [1]
        i2, i3, i5= 0, 0 ,0
        while len(res)<n:
            n2, n3, n5 = 2*res[i2], 3*res[i3], 5*res[i5]
            val = min([n2, n3, n5])
            if val==n2:
                i2 += 1
            if val==n3:
                i3 += 1
            if val==n5:
                i5 += 1
            res.append(val)

        return res[-1]

n = 10
so = Solution()
ans = so.nthUglyNumber(n)
print(ans)
```

https://leetcode.com/problems/super-ugly-number/

```
import heapq
class Solution(object):
    def nthSuperUglyNumber(self, n, primes):
        """
        :type n: int
        :type primes: List[int]
        :rtype: int
        """
        res = [1]
        q = []
        for p in primes:
            heapq.heappush(q, (p, 0, p))

        while len(res)<n:
            val, i, p = q[0]
            res.append(val)
            while q and q[0][0]==val:
                val, i, p = heapq.heappop(q)
                heapq.heappush(q, (p*res[i+1], i+1, p))
        return res[-1]
```

P13 merge k sorted list

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

import heapq
class Solution(object):
    def mergeKLists(self, lists):
        """
        :type lists: List[ListNode]
        :rtype: ListNode
        """

        dummy =  ListNode(None)
        head = dummy
        q = [(n.val ,n) for n in lists if n]
        heapq.heapify(q)

        while q:
            val, n = heapq.heappop(q)
            head.next = n
            head = head.next
            if n.next:
                heapq.heappush(q, (n.next.val,n.next))

        return dummy.next
```

好像这里用heapreplace要快一点。不过也没快多少。

```
import heapq
class Solution(object):
    def mergeKLists(self, lists):
        """
        :type lists: List[ListNode]
        :rtype: ListNode
        """

        dummy =  ListNode(None)
        head = dummy
        q = [(n.val ,n) for n in lists if n]
        heapq.heapify(q)

        while q:
            val, n = q[0]
            head.next = n
            head = head.next
            if not n.next:
                heapq.heappop(q)
            else:
                heapq.heapreplace(q, (n.next.val, n.next))

        return dummy.next
```

P14 8*8的棋盘 走K步从一个点走到另一个点有多少不同路径，之前在地里看到过但下面的都不对，先写了个dfs， 然后说简化，我说两头dfs， 然后说再简化，然后经他提示用dp做出来了， 思路是第K步走到一个格子一共有多少不同路径， 那么K+1步走到他周围八个点就是多少路径，解法类似Game of life开个buff数组

怎么越做他们家的题越觉得他们家的bar越低啊。什么菜题都拿出来考人。快把我招进去，我帮他们出两道酷酷的题目。我觉得这道题dp也没有简化复杂度。只是优化了空间。

```
class Solution(object):
    def UniquePath(self, size, start, end, k):

        res = []
        self.dfs(start, end, [], k, size, res)
        return res

    def dfs(self, cur, end, path, k, size, res):

        if not k:
            if cur==end:
                res.append(path)
            return

        for dp in [(1,0) ,(-1,0), (0,1), (0,-1)]:
            i, j = cur[0]+dp[0], cur[1]+dp[1]
            if 0<=i<size and 0<=j<size:
                self.dfs((i,j), end, path+[(i,j)], k-1, size, re
s)

boardsize = 8
start = (0,0)
end = (3,3)
so = Solution()
ans = so.UniquePath(boardsize, start, end, 6)
print(ans)
```

P15 安排会议室 又是瞬间秒掉 https://leetcode.com/problems/meeting-rooms-ii/

```
class Solution(object):
    def minMeetingRooms(self, intervals):
        """
        :type intervals: List[List[int]]
        :rtype: int
        """
        times = []
        for s, e in intervals:
            times.append((s,1))
            times.append((e,0))

        times.sort()

        rooms, avaliable = 0, 0
        for t in times:
            if t[1]:
                if not avaliable:
                    rooms += 1
                else:
                    avaliable -= 1
            else:
                avaliable += 1
        return rooms

intervals = [[0, 30],[5, 10],[15, 20]]
so = Solution()
print so.minMeetingRooms(intervals)
```

我们加强一下这道题。除了计算房间的个数，还要给每个会议室分配房间号码。

```
class Solution(object):
    def minMeetingRooms(self, intervals):
        """
        :type intervals: List[List[int]]
        :rtype: int
        """
        times = []
        room_num = {}
        for s, e in intervals:
            times.append((s,1,e))
            times.append((e,0,s))

        times.sort()

        rooms, avaliable = 0, []
        for t in times:
            if t[1]:
                if not avaliable:  # make a new room num
                    rooms += 1
                    room_num[t[0]] = rooms
                else: # use a empty room num
                    room_num[t[0]] = avaliable.pop(0)
            else:   # return a room
                avaliable.append(room_num[t[2]])

        return room_num

intervals = [[0, 30],[5, 10],[15, 20]]
so = Solution()
print so.minMeetingRooms(intervals)
```

P16 感觉是word search 简易版，一个字典，给个单词，判断单词在不在字典里。和第二轮一样，一下子就觉得应该用trie做，本来以为没啥问题，结果写search的时候脑抽了，写了个bug，改了一下，又跑了另一个test case，结果还有一个bug，还好及时改好了，估计这里减分了。follow up就是如果单词里有问号，代表任意字母，判断字典里是不是有这个pattern的单词。用dfs挺快写好了，跑了几个test case也没啥问题。他好像说挺好的，具体也没听清了，估计也是安慰安慰我。后来就聊了聊天，我问了问在这里和大公司的区别之类的。

那我们就先写一个简单的版本吧。

```
class Node(object):
    def __init__(self):
        self.dic = {}
        self.isWord = False

class Solution(object):
    def findWords(self, words, s):

        root = Node()
        # Build trie
        for w in words:
            tmp = root
            for c in w:
                if c not in tmp.dic:
                    tmp.dic[c] = Node()
                tmp = tmp.dic[c]
            tmp.isWord = True

        ans = [i for i in root.dic]
        print(ans)

        tmp = root
        for c in s:
            if c not in tmp.dic:
                return False
            tmp = tmp.dic[c]

        return tmp.isWord


words = ["oath","pea","eat","rain"]
s = "oath"
so = Solution()
ans = so.findWords(words, s)
print(ans)
```

然后我们再写一个带星号的。

```
class Node(object):
    def __init__(self):
        self.dic = {}
        self.isWord = False

class Solution(object):
    def findWords(self, words, s):

        root = Node()
        # Build trie
        for w in words:
            tmp = root
            for c in w:
                if c not in tmp.dic:
                    tmp.dic[c] = Node()
                tmp = tmp.dic[c]
            tmp.isWord = True

        res = []
        self.findAll(root, s, '', res)
        return res

    def findAll(self, node, s, path, res):

        if not s:
            if node.isWord==True:
                res.append(path)
            return

        if s[0]=='*':
            for n in node.dic:
                self.findAll(node.dic[n], s[1:], path+n, res)
        else:
            if s[0] in node.dic:
                self.findAll(node.dic[s[0]], s[1:], path+s[0], r
es)

words = ["oath","pea","eat","rain"]
s = "*a*h"
so = Solution()
```

```
ans = so.findWords(words, s)
print(ans)
```

P17 给inorder和postorder建树

```
class Solution(object):
    def buildTree(self, inorder, postorder):
        """
        :type inorder: List[int]
        :type postorder: List[int]
        :rtype: TreeNode
        """
        if inorder:
            num = postorder.pop()
            root = TreeNode(num)
            index = inorder.index(num)
            root.right = self.buildTree(inorder[index+1:], posto
rder)
            root.left = self.buildTree(inorder[:index], postorde
r)
            return root
```

P18 lc Word Search II word search变形，输出字典里面每一个单词出现的次数。follow up是怎么scale，都要求把代码写下来然后运行

https://leetcode.com/problems/word-search-ii/

```
class Node(object):
    def __init__(self):
        self.dic = {}
        self.isWord = False

class Solution(object):
    def findWords(self, board, words):
        """
        :type board: List[List[str]]
        :type words: List[str]
        :rtype: List[str]
        """
```

```
        root = Node()

        for w in words:
            cur = root
            for c in w:
                if c not in cur.dic:
                    cur.dic[c] = Node()
                cur = cur.dic[c]
            cur.isWord = True

        res = []
        for i in range(len(board)):
            for j in range(len(board[0])):
                self.bt(i, j, board, '', root, res)
        return res

    def bt(self, i, j, board, path, root, res):
        if root.isWord:
            res.append(path)
            root.isWord = False

        if 0<=i<len(board) and 0<=j<len(board[0]):
            c  = board[i][j]
            if c in root.dic:
                board[i][j]='#'
                self.bt(i-1, j, board, path+c, root.dic[c], res)
                self.bt(i+1, j, board, path+c, root.dic[c], res)
                self.bt(i, j+1, board, path+c, root.dic[c], res)
                self.bt(i, j-1, board, path+c, root.dic[c], res)
                board[i][j]=c
```

P19

317 Shortest Distance from All Buildings

今天状态好，居然又写了一个更简单的版本，鼓掌！撒花！

```
class Solution(object):
    def shortestDistance(self, grid):
        """
        :type grid: List[List[int]]
        :rtype: int
        """
        if not grid or not grid[0]:
            return -1

        cost = [[0]*len(grid[0]) for _ in range(len(grid))]
        cnt = 0 # count how many building we have visited
        for i in range(len(grid)):
            for j in range(len(grid[0])):
                if grid[i][j]==1:
                    self.bfs((i,j), grid, cost, cnt)
                    cnt -= 1

        res = [cost[i][j] for i in range(len(grid)) for j in ran
ge(len(grid[0])) if grid[i][j]==cnt]
        return min(res) if res else -1

    def bfs(self, start, grid, cost, cnt):

        q = [(start, 0)]
        while q:
            p, dist = q.pop(0)
            for dp in [(1,0), (-1,0), (0,1), (0,-1)]:
                i, j = p[0]+dp[0], p[1]+dp[1]
                # only the position be visited by cnt times will
 append to queue
                if 0<=i<len(grid) and 0<=j<len(grid[0]) and grid
[i][j]==cnt:
                    cost[i][j] += dist+1
                    q.append(((i,j), dist+1))
                    grid[i][j] -= 1
```

296 Best Meeting Point 原来直接算meduim就行了？我觉得自己是个傻逼啊！

```
class Solution(object):
    def minTotalDistance(self, grid):
        """
        :type grid: List[List[int]]
        :rtype: int
        """
        if not grid or not grid[0]:
            return

        cost = [[0]*len(grid[0]) for _ in range(len(grid))]

        for i in range(len(grid)):
            for j in range(len(grid[0])):
                if grid[i][j]==1:
                    self.bfs((i,j), cost, grid)
        res = [i for row in cost for i in row if i!=0]
        return min(res) if res else -1

    def bfs(self, start, cost, grid):

        q = [(start, 0)]
        visited = {start}

        while q:
            po, dist = q.pop(0)
            for dp in [(1,0), (-1,0), (0,1), (0,-1)]:
                i, j = po[0]+dp[0], po[1]+dp[1]
                if 0<=i<len(grid) and 0<=j<len(grid[0])\
                and (i,j) not in visited:
                    cost[i][j] += dist+1
                    q.append(((i, j),dist+1))
                    visited.add((i,j))

grid = [[1,0,0,0,1],[0,0,0,0,0],[0,0,1,0,0]]
so = Solution()
ans = so.minTotalDistance(grid)
print ans
```

大神的解法

```
class Solution(object):
    def minTotalDistance(self, grid):
        """
        :type grid: List[List[int]]
        :rtype: int
        """
        if not grid:
            return 0
        r, c = len(grid), len(grid[0])
        sumr = [i for i in xrange(r) for j in xrange(c) if grid[
i][j]]
        sumc = [j for i in xrange(r) for j in xrange(c) if grid[
i][j]]
        sumr.sort()
        sumc.sort()
        mid_row = sumr[len(sumr)/2]
        mid_col = sumc[len(sumc)/2]
        return sum([abs(r-mid_row) for r in sumr]) + sum([abs(c-
mid_col) for c in sumc])
```

P20 Big Integer加减，可正可负，听他描述完，有点想抽自己，看到有人说了类似题目的面经，但是说的是电面，就没准备，没想撞上了。题目大意： 传入一个String, 自己定定义BigInteger class的内部变量, 实现加减； 想了一下，本来想直接char arrar处理，发现把符号信息单独拿出来会简单很多， 所以就是BigInteger{int sign, int[] nums}, 然后就是实现， 两个数的符号 ++, --, +-, -+, 分情况写，没有全部写完，就写好了＋＋， －－, 小哥说没时间了，你写几个testcase 先跑一下这个吧，testcase 通过， done.

不带小数的加法和乘法。

```
class Solution(object):
    def add(self, num1, num2):
        """
        :type num1: str
        :type num2: str
        :rtype: str
        """
        num1 = list(num1)[::-1]
```

```
        num2 = list(num2)[::-1]

        if len(num1)<len(num2):
            num1, num2 = num2, num1

        carry = 0
        index = 0
        while index<len(num1):
            n2 = int(num2[index]) if index<len(num2) else 0
            tmp = n2 + int(num1[index]) + carry
            cur = tmp%10
            carry = tmp//10
            num1[index] = str(cur)
            index += 1

        if carry: num1.append('1')

        return ''.join(num1[::-1])

    def multiply(self, num1, num2):
        """
        :type num1: str
        :type num2: str
        :rtype: str
        """
        num1 = num1[::-1]
        num2 = num2[::-1]

        res = [0]*(len(num1) + len(num2))
        for i in range(len(num2)):
            for j in range(len(num1)):
                tmp = int(num2[i])*int(num1[j])
                res[j+i] += tmp
                res[j+i+1] += res[j+i]//10
                res[j+i] = res[j+i]%10

        while len(res)>1 and res[-1]==0:
            res.pop()

        return ''.join(map(str,res))[::-1]
```

```
num1 = '124'
num2 = '5679'
so = Solution()
a = so.add(num1, num2)
print(a)
a = so.multiply(num1, num2)
print(a)
```

sc onsite 21-30

P21 regular和wildcard match准备一波 regular expression 这两道题真的是把本宝宝写哭了。就是各种corner case！ 其实基本思想就是dp.

遇到.和相同的时候，应该选左上角进行dp,如果是星星，那么可以继承上面和上上面的结果，作用分别是消除掉前面那个char和相做不存在。然后还可以不停地copy前一个，这里就需要当前一个与s中的字母相同时，可以dp s前的前一个char的结果，总之很绕。

```
class Solution(object):
    def isMatch(self, s, p):
        """
        :type s: str
        :type p: str
        :rtype: bool
        """
        dp = [[False]*(len(s)+1) for _ in range(len(p)+1)]
        dp[0][0] = True
        for i in range(1, len(p)):
                dp[i + 1][0] = dp[i - 1][0] and p[i] == '*'

        for i in range(len(p)):
          for j in range(len(s)):
            if p[i]=='.':
              dp[i+1][j+1] = dp[i][j]
            elif p[i]==s[j]:
              dp[i+1][j+1] = dp[i][j]
            elif p[i]=='*':
              dp[i+1][j+1] = dp[i][j+1] or dp[i-1][j+1]
              if i>0 and p[i-1]=='.' or p[i-1]==s[j]:
                dp[i+1][j+1] = dp[i+1][j+1] or dp[i+1][j]
          print i+1, dp[i+1]
        return dp[-1][-1]
```

wildcard matching https://leetcode.com/problems/wildcard-matching/ 我还是觉得2D dp要好理解得多，面试如果要我优化成1D的DP。那我可是要跪地求饶给他看！

```
class Solution(object):
    def isMatch(self, s, p):
        """
        :type s: str
        :type p: str
        :rtype: bool
        """
        dp = [[False]*(len(s)+1) for _ in range(len(p)+1)]
        dp[0][0] = True

        for i in range(len(p)):
            for j in range(len(s)):
                if p[i]=='?':
                    dp[i+1][j+1] = dp[i][j]
                elif p[i]==s[j]:
                    dp[i+1][j+1] = dp[i][j]
                elif p[i]=='*':
                    dp[i+1][j+1] = dp[i][j+1] or dp[i][j] or dp[
i+1][j]
            print dp[i+1]
        return dp[-1][-1]

s = 'aaa'
p = 'aa'
so = Solution()
ans = so.isMatch(s, p)
print(ans)
```

P22 1. 1. find all amicable numbers.输入一个正整数，找出所有小于这个数的amicable pairs。 看之前的面经有时间复杂度O(nlogn)，空间复杂度O(n)的算法。感觉对于每一个数字求解是sqrt(n). 总的复杂度是n * log(n)。求解答怎么得到nlogn的算法 下面有大神回答：跟count prime类似的思想，通过n log n的复杂度求因子和

我没太听懂，不过还是写一遍count prime好了。aicable number之前写过一次就不写了

```
class Solution(object):
    def countPrimes(self, n):
        """
        :type n: int
        :rtype: int
        """
        if n<2:
            return 0
        p = [True]*n
        p[0], p[1]=False, False

        for i in range(2, (int(n**0.5)+1)):
            if not p[i]:
                pass
            else:
                for j in range(i*i, n, i):
                    p[j]=False
        return sum(p)
```

P23 第一轮题目是给一堆会议时间，选出从早7点到晚7点的free时间，比如给了 8-11, 12:30-17:00, 15:00-17:30,就输出7-8, 11-12:30, 5:30-19:00，我想了好一阵，然后说用stack一个一个处理，结果面试官说为啥要用stack，我才意识到用一个结束时间就可以，一开始设置成7:00，处理一个时间就更新一些结束时间，然后输出就行了，不过写完了有一个bug，然后面试小哥说这是一个serious bug，我当时感觉他就有点不满意，囧，加了if else判断处理掉bug

整体不难吧。主要是corner case处理比较麻烦？

```
class Solution(object):
    def findFreeTime(self, d, ms):
        """
        :type d: str
        :type m: List[str]
        :rtype: list[str]
        """
        # split times
        meetings = []
        for m in ms:
            s,e = m.split('-')
            meetings.append([s,e])

        # merge times
        mergedMeetings = [meetings[0]]
        for i in range(1,len(meetings)):
            if mergedMeetings[-1][1]>meetings[i][0]:
                mergedMeetings[-1][1]=meetings[i][1]
            else:
                mergedMeetings.append(meetings[i])

        # build result
        s, e = d.split('-')
        res = []
        for m in mergedMeetings:
            if m[0]>s:
                res.append([s, m[0]])
                s = m[1]
        if s<e:
            res.append([s, e])

        return res

day = '7:00-19:00'
meetings = ['8:00-11:00', '12:30-17:00', '15:30-17:30']
so = Solution()
ans = so.findFreeTime(day, meetings)
print ans
```

P24 中国小哥 + shadow， 聊天， 问project， 问的问题是longest incresing subsequence， 似乎是lintcode上的题？还有一题是大数乘法。

大数乘法如下 https://qiuzhihui.gitbooks.io/r-book/content/big_integer.html LIS如下https://leetcode.com/problems/longest-increasing-subsequence/ 中国大哥就是给力！

```
class Solution(object):
    def lengthOfLIS(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """

        if not nums:
            return 0

        dp = [1]*len(nums)

        for i in range(len(nums)):
            for j in range(i):
                if nums[i]>nums[j]:
                    dp[i] = max(dp[i],dp[j]+1)

        return max(dp)

nums = [10, 9, 2, 5, 3, 7, 101, 18]
so = Solution()
ans = so.lengthOfLIS(nums)
print ans
```

P25 第四轮，国人小哥。我也不知道他后来有没有放水，反正是挂了。但过程中他给了很多的提示。题目是这样的：一棵树，代表上司和员工的关系，然后每个节点都有一个对应的权值。然后公司开了一个party，为了让员工们更好的交流，有个规定，如果上司去参加party，那么他的直接下属（直接孩子节点)就不去（同理，如果下属去了，直接上司，父节点就不去）。然后设计一个算法，参加party的人的权

值总和最高。这是一道动态规划题，思路是每次计算一个员工去的权值之和与不去的权值之和，从叶子开始。 这不就是house robber 3么？
https://leetcode.com/problems/house-robber-iii/

思路：这道题很像dp。就是一直返回最优。根据取不去root的值可以分成2类情况来讨论，因为要返回两个情况的最优，所以需要新建一个helper function。这里提一句，什么时候需要helper function呢？ 一般是我们想到这道题需要递归，原来的fuction不能满足这种递归的结构。 dfs也是其中的一种原方程不能满足结构的情况。

Top to bot with memo

```
class Solution(object):
    def rob(self, root):
        """
        :type root: TreeNode
        :rtype: int
        """
        self.memo = {}
        return self.helper(root)

    def helper(self, root):
        if not root:   # if not root
            return 0

        if root not in self.memo:  # if it is not in memo
            val = 0
            if root.left:  # we calculate not have left
                val += self.helper(root.left.left) + self.helper
(root.left.right)
            if root.right: # we calculate not have right
                val += self.helper(root.right.left) + self.helpe
r(root.right.right)
            f = root.val+val    # max include root
            s = self.helper(root.left) + self.helper(root.right)
 # max not include root
            self.memo[root] = max(f, s)
        return self.memo[root]
```

bot to top solution

```
class Solution(object):
    def rob(self, root):
        """
        :type root: TreeNode
        :rtype: int
        """
        return max(self.helper(root))

    def helper(self, root):

        if not root:
            return [0, 0]

        l = self.helper(root.left)
        r = self.helper(root.right)

        f = root.val + l[1] + r[1]
        s = max(l[0]+r[1], l[1]+r[0], l[0]+r[0], l[1]+r[1])

        return [f, s]
```

P26 第一轮是一个大胡子白人，扎着马尾辫。看起来还是挺严肃的。先聊简历，聊了很多。他问到有个我做过的项目，如果想再做一遍的话，问我想怎样改进。编程题是给定一个有向关系图，在给定2个名字，求出这两个人的共同朋友（即2个点能达到的共同节点，类似树的共同祖先）

思路：就是dfs然后求一个交集？

```
import collections
class Solution(object):
    def commonFriends(self, relation, a, b):

        if not relation:
            return []

        dic = collections.defaultdict(set)
        for r in relation:
            a, b = r
            dic[a].add(b)

        s, af = [a], set([a])
        while s:
            tmp = s.pop()
            children = dic[tmp]
            for c in children:
                if c not in af:
                    s.append(c)
                    af.add(c)

        s, bf = [b], set([b])
        while s:
            tmp = s.pop()
            children = dic[tmp]
            for c in children:
                if c not in bf:
                    s.append(c)
                    bf.add(c)

        return list(af&bf)

relation = [[1,2],[2,3],[3,4],[6,5],[5,4]]
a, b = 1, 6
so = Solution()
ans = so.commonFriends(relation, a, b)
print ans
```

P27 第二轮是一个白人小哥，人很nice。简单的聊了一下简历后就开始编程。题目是如果通讯录里从某一个名字开始翻转了，请把这个名字找出来。类似leetcode这道题：。这题我二分搜索算法犯了个低级错误，大概被扣分不少。 这道题就是find min in rotated array啊。 https://leetcode.com/problems/find-minimum-in-rotated-sorted-array/

```
class Solution(object):
    def findR(self, names):
        if not names:
            return

        l, r = 0, len(names)-1
        while l<r:
            mid = (l+r)//2
            if names[mid]<names[r]:
                r = mid
            else:
                l = mid+1
        return names[l]

names = ['case', 'doug', 'aron', 'bob']
so = Solution()
ans = so.findR(names)
print ans
```

P28 我觉得吧game of life得再写一遍

正常的

```
class Solution(object):
    def gameOfLife(self, board):
        """
        :type board: List[List[int]]
        :rtype: void Do not return anything, modify board in-pla
ce instead.
        """

        if not board or not board[0]:
            return

        for i in range(len(board)):
            for j in range(len(board[0])):
                cnt = 0
                for dp in [(1,0),(-1,0),(0,1),(0,-1),(1,1),(1,-1
),(-1,1),(-1,-1)]:
                    m, n = i+dp[0], j+dp[1]
                    if 0<=m<len(board) and 0<=n<len(board[0]):
                        if board[m][n]&1==1:
                            cnt += 1

                if board[i][j]==1 and 2<=cnt<=3:
                    board[i][j] = board[i][j]^2
                if board[i][j]==0 and cnt==3:
                    board[i][j] = board[i][j]^2


        for i in range(len(board)):
            for j in range(len(board[0])):
                board[i][j] = board[i][j]>>1
        print board

board = [[0,0,0,0],[0,1,1,0],[0,1,1,0],[0,0,0,0]]
so = Solution()
so.gameOfLife(board)
```

用hash直接来存坐标 输入和输出都不变，为了好验证

```
class Solution(object):
```

```
    def gameOfLife(self, board):
        """
        :type board: List[List[int]]
        :rtype: void Do not return anything, modify board in-pla
ce instead.
        """

        if not board or not board[0]:
            return
        dic = {}

        for i in range(len(board)):
            for j in range(len(board[0])):
                if board[i][j]==1:
                    dic[(i,j)] = [1,0]

        keys = dic.keys()

        for k in keys:
            tmp = dic[k]
            for dp in [(1,0),(-1,0),(0,1),(0,-1),(1,1),(1,-1),(-1
,1),(-1,-1)]:
                i, j = k[0] + dp[0], k[1]+dp[1]
                if (i,j) in dic:
                    dic[(i,j)][1] += 1
                else:
                    dic[(i,j)] = [0,1]

        keys = dic.keys()
        for k in keys:
            if dic[k][0]==1:
                if 2<=dic[k][1]<=3:
                    dic[k] = [1,0]
                else:
                    del dic[k]
            else:
                if dic[k][1]==3:
                    dic[k] = [1,0]
                else:
                    del dic[k]
```

```
        for i in range(len(board)):
            for j in range(len(board[0])):
                if (i,j) in dic:
                    board[i][j]==1
                else:
                    board[i][j]==0

        print board


board = [[0,0,0,0],[0,1,1,0],[0,1,1,0],[0,0,0,0]]
so = Solution()
so.gameOfLife(board)
```

P29 Reconstruct Itinerary

backtracking的经典题型，找一个path试，可以就返回，不可以就试下一个。

```
import collections
class Solution(object):
    def findItinerary(self, tickets):
        """
        :type tickets: List[List[str]]
        :rtype: List[str]
        """
        maxL = len(tickets)+1
        dic = collections.defaultdict(list)

        for f,t in tickets:
            dic[f].append(t)

        for k in dic:
            dic[k].sort()

        self.res = None
        print self.dfs(dic, 'JFK', ['JFK'], maxL)
        print self.res


    def dfs(self, dic, cur, path, maxL):

        if len(path)==maxL:
            self.res = path
            return True

        for i in range(len(dic[cur])):
            c = dic[cur].pop(0)
            if self.dfs(dic, c, path+[c], maxL):
                return True
            dic[cur].append(c)


tickets = [["JFK","SFO"],["JFK","ATL"],["SFO","ATL"],["ATL","JFK
"],["ATL","SFO"]]
so = Solution()
so.findItinerary(tickets)
```

感觉自己不是很熟悉backtracking,那我们就把N-Queen II写一遍好了。

```
class Solution(object):
    def totalNQueens(self, n):
        """
        :type n: int
        :rtype: List[List[str]]
        """
        board = [['.']*n for _ in range(n)]
        dic = set()
        res = []
        self.bt(0, 0, board, dic, [], n, res)
        return res

    def bt(self, m, n, board, dic, path, l, res):
        if len(path)==l:
            return True

        for i in range(m, len(board)):
            for j in range(len(board[0])):
                if (0,i-j) not in dic and (1,i+j) not in dic and
\
                (2,i) not in dic and (3,j) not in dic:
                    dic.add((0,i-j))
                    dic.add((1,i+j))
                    dic.add((2,i))
                    dic.add((3,j))
                    board[i][j]= 'Q'
                    if self.bt(i, j, board, dic, path+[(i,j)], l
, res):
                        res.append(map(''.join, [k for k in boar
d]))
                    dic.remove((0,i-j))
                    dic.remove((1,i+j))
                    dic.remove((2,i))
                    dic.remove((3,j))
                    board[i][j]= '.'

so = Solution()
ans = so.totalNQueens(4)
print ans
```

P30 然后就做一道类似knight move的题目，给了一个函数list nextMoves()，返回所有下一个有效的move，棋盘无穷大，输出到目的地的最短路径。bfs就可以做好，另开一个hash表记录上一步。follow-up是输出所有的最短路径。用hash表记录上一步所有可能的位置，然后dfs返回所有路径。这道题是在电脑上编译通过的。讲完follow-up时已经没时间了，问了个小问题国人小哥就走了。

感觉这道题目跟P14 很像，那就用bfs写吧。最短路径我总感觉只有唯一解啊。实在不行，我就用level order traversal的办法好了。具体代码就不写了。

sc onsite 31-40

P31 给两个节点，判断两个人是不是6度之内的朋友并且返回中间有多少人，之前面经看到，没写，真是后悔啊。一开始给了用queue和set的bfs解法，他说空间复杂度太高了，后来说用dfs，他说空间还是高，毕竟要用set存访问过的节点，我实在想不出来了，要了提示，可是感觉提示也并不是那么明显，我说能不能从两个点分别开始搜？他说你觉得这对复杂度有优化吗？我就觉得这估计也没戏。折腾了半天，好像也就剩20多分钟了，他说你先写一个你的解法，有时间我们再优化。写了那个bfs的，写完他让给test case，我忘了a和b是同一个人以及a和b中间人数多于6个的情况了，他提醒了一下加上了一两句判断。写好之后他看了觉得应该没bug了，让我问问题了，我还在纠结优化，他说10min估计不够了，优化是optional的(估计是安慰我的)，然后让我问问题了。白人小哥是搞discovery的，我提了一个使用不是很顺的情况，他也说有蛮多人有抱怨的。中国小哥还给我讲了一些隐藏功能，他说他们也在想办法弄用户手册之类的，不然真的蛮难上手的

```
import collections
class Solution(object):
    def friend6(self, relations, a, b):
        dic = collections.defaultdict(set)
        for s,e in relations:
            dic[s].add(e)
            dic[s].add(e)

        visited = set([a])
        q = [(a, 0, [])]

        while q:
            p, d, path = q.pop(0)
            if d>6: return
            if p==b: return path+[p]
            for c in dic[p]:
                if c not in visited:
                    visited.add(c)
                    q.append((c, d+1, path+[p]))

        return

relations = [[1,2], [2,3], [3,4],[4,5]]
so = Solution()
ans = so.friend6(relations, 1, 5)
print ans
```

不知道优化空间是不是这个意思。

```
import collections
class Solution(object):
    def friend6(self, relations, a, b):
        dic = collections.defaultdict(set)
        for s,e in relations:
            dic[s].add(e)
            dic[s].add(e)

        visited = set([a])
        q = [(a, 0)]
        indegree = {}

        while q:
            p, d = q.pop(0)
            if d>6: return
            if p==b: break
            for c in dic[p]:
                if c not in visited:
                    visited.add(c)
                    indegree[c] = p
                    q.append((c, d+1))
        path = []
        while b!=a:
            path = [b] + path
            b = indegree[b]
        if b==a:
            path = [a] + path
        return path



relations = [[1,2], [2,3], [3,4],[4,5]]
so = Solution()
ans = so.friend6(relations, 1, 5)
print ans
```

P32 第四轮是个国人姐姐。问了问project，她好像也做过类似的，所以稍微多问了一点点。题目挺简单的，merge two sorted list， follow up是merge k sorted list，写完test case跑过了，问了问复杂度。她看还有好久再写一个吧，给inorder和

postorder建树，我用recursion做，结果最后在子数组边界的地方加一减一的地方老是有问题，然后让她说没事，我相信你很快就能调好的，思路是对的，不用纠结了。

```
class Solution(object):
    def mergeTwoLists(self, l1, l2):
        """
        :type l1: ListNode
        :type l2: ListNode
        :rtype: ListNode
        """
        dummy = ListNode(None)
        head = dummy

        while l1 and l2:
          if l1.val<l2.val:
            dummy.next = l1
            dummy = dummy.next
            l1 = l1.next
          else:
            dummy.next = l2
            dummy = dummy.next
            l2 = l2.next

        dummy.next = l1 or l2

        return head.next
```

已经不知道写过多少遍了。

```
import heapq
class Solution(object):
    def mergeKLists(self, lists):
        """
        :type lists: List[ListNode]
        :rtype: ListNode
        """
        dummy = ListNode(None)
        head = dummy

        q = [(n.val, n) for n in lists if n]
        heapq.heapify(q)

        while q:
            val, node = heapq.heappop(q)
            dummy.next = node
            dummy = dummy.next
            if dummy.next:
                heapq.heappush(q, (dummy.next.val, dummy.next))

        return head.next
```

P33 给一个nn的*chess board*，*knight*可以跳2\3的格子的对角线，就是国际象棋的规则。求给出一个起始点，knight能否跳到给定的重点。BFS搞定。followup print出来路径，backtrace就可以了，把每个格子上个格子的方位存下来，允许使用额外空间。写完后跑了test，小哥说：you did a good job

没太懂瞎写了一个。

```
class Solution(object):
    def knightMove(self, n, start):

        board = [[False]*n for _ in range(n)]

        q = [(start,[start])]

        while q:
            cur = q.pop(0)
            tmp, path = cur[0], cur[1]

            for dp in [(1,2), (-1,2), (1,-2), (-1,-2), (2,1), (2,
-1), (-2,1),(-2,-1)]:
                i, j = tmp[0]+dp[0], tmp[1]+dp[1]
                if (i,j) == start: return path+[(i,j)]
                if 0<=i<n and 0<=j<n and not board[i][j]:
                    board[i][j] = True
                    q.append(((i,j), path+[(i,j)]))

so = Solution()
ans = so.knightMove(4, (0,0))

print ans
```

P34 给一个数组，A,B轮流从头尾处选出一个数字，求当B使用（1）贪心和（2）最优策略时，A能取到所有数字之和最大。我使用的recursive写的，优化用的是hash 存储从子数组（i, j）上能得到的最优解。写了几个test跑过了。

这道题跟我收录的dp的里面很像

B使用贪心的时候 A的解法。

```
class Solution(object):
    def maxCoin(self, nums):

        self.memo = {}
        res = self.helper(0, len(nums)-1, nums)
        print self.memo
        return res

    def helper(self, i, j, nums):

        if j-i<2:
            return max(nums[i:j+1])

        if (i, j) not in self.memo:
            l = nums[i]
            if nums[i+1]>nums[j]:
                l += self.helper(i+2,j,nums)
            else:
                l += self.helper(i+1,j-1,nums)
            r = nums[j]
            if nums[i]>nums[j-1]:
                r += self.helper(i+1,j-1,nums)
            else:
                r += self.helper(i, j-2, nums)
            self.memo[(i,j)] = max(l,r)
        return self.memo[(i,j)]

nums = [1,2,5,3]
so = Solution()
ans = so.maxCoin(nums)
print ans
```

B使用最优的时候的A的解法。

```
class Solution(object):
    def maxCoin(self, nums):

        self.memo = {}
        res = self.helper(0, len(nums)-1, nums)
        print self.memo
        return res

    def helper(self, i, j, nums):

        if j-i<2:
            return max(nums[i:j+1])

        if (i, j) not in self.memo:
            p2 = min(self.helper(i+1,j,nums), self.helper(i,j-1,
 nums))
            self.memo[(i,j)] = sum(nums[i:j+1]) - p2
        return self.memo[(i,j)]

nums = [1,9,1,3]
so = Solution()
ans = so.maxCoin(nums)
print ans
```

P35 第二轮，韩国大叔，Leetcode的symmetric tree。这题我一开始上来用了最精简的方法，然后似乎大叔不太能follow，要我从最简单的BFS做一遍，然后又从DFS做一遍。现在回过头来总结，其实面试的时候不要一开始给出最优解，给出一个相对naive但是直观的解，然后逐步改进，这样可以向面试官展现你的thinking process，一上来就最优的，很多面试官都不是很喜欢的。

```
class Solution(object):
    def isSymmetric(self, root):
        """
        :type root: TreeNode
        :rtype: bool
        """
        if not root:
            return True
        return self.helper(root.left, root.right)

    def helper(self, l, r):
        if not l and not r: return True
        if not l or not r: return False
        if l.val!=r.val: return False

        return self.helper(l.left, r.right) and self.helper(l.ri
ght, r.left)
```

老天爷啊，求面试给我一道这种题目吧！！一道就行！！

就到这里了吧。不写了。考到没做过的。就看命咯。

祝看到这里的同学都能有个好前程，

哎呀，还是再写一个word break吧，有点不放心。

先来一个暴力的哈

```
class Solution(object):
    def wordBreak(self, s, wordDict):
        """
        :type s: str
        :type wordDict: Set[str]
        :rtype: List[str]
        """
        if not s:
            return []

        res = []
        self.dfs(s, wordDict, '', res)
        return res

    def dfs(self, s, wordDict, path, res):
        if not s:
            res.append(path)

        for w in wordDict:
            if s[:len(w)] == w:
                self.dfs(s[len(w):], wordDict, path+(' ' if path
 else '')+w, res)

s = "catsanddog"
dict = ["cat", "cats", "and", "sand", "dog"]
so = Solution()
a = so.wordBreak(s, dict)
print a
```

然后在用memo优化

```
class Solution(object):
    def wordBreak(self, s, wordDict):
        """
        :type s: str
        :type wordDict: Set[str]
        :rtype: List[str]
        """
        if not s:
            return []

        self.memo = {}
        return self.dfs(0, len(s), s, wordDict)

    def dfs(self, i, j, s, wordDict):

        if (i,j) not in self.memo:
            path = []
            if s[i:j] in wordDict:
                path += [s[i:j]]

            for w in wordDict:
                if len(w)<=j-i and s[i:i+len(w)]==w:
                    res = self.dfs(i+len(w), j, s, wordDict)
                    path += [w+' '+p for p in res]
            self.memo[(i,j)] = path
        return self.memo[(i,j)]


s = "catsanddog"
dict = ["cat", "cats", "and", "sand", "dog"]
so = Solution()
a = so.wordBreak(s, dict)
print a
```

酱油版的Dijkstra再写一遍吧。

```
class Solution(object):
    def dijkstra(self, graph, start, end):
        Q = {}
```

```
        P = {}
        D = {}

        Q[start] = 0

        while Q:
            k = self.findMin(Q)
            D[k] = Q[k]
            children = graph[k]
            for c in children:
                dis = D[k]+children[c]
                if c in D:
                    if dis<D[c]:
                        return
                elif c not in Q or dis<Q[c]:
                    Q[c] = dis
                    P[c] = k
            del Q[k]

        print P
        path, res = [], D[end]
        while end in P:
            path = [end] + path
            end = P[end]
        path = [end] + path

        return path,res

    def findMin(self, Q):
        m = (None, float('inf'))
        for k in Q:
            if Q[k]<m[1]:
                m = (k, Q[k])
        return m[0]


# example, CLR p.528
G = {'s': {'u':10, 'x':5},
    'u': {'v':1, 'x':2},
    'v': {'y':4},
```

```
    'x':{'u':3,'v':9,'y':2},
    'y':{'s':7,'v':6}}

so = Solution()
print so.dijkstra(G, 's', 'v')
```

再写个topo sort吧

```
import collections
class Solution(object):
    def toposort(self, graph):
        """
        :type graph: dict[int:List[int]]
        :rtype: List[int]
        """
        indegree = collections.defaultdict(set)
        outdegree = collections.defaultdict(set)
        for k in graph:
            for c in graph[k]:
                outdegree[k].add(c)
                indegree[c].add(k)



        s = [k for k in outdegree if k not in indegree]
        res = s[:]
        while s:
            tmp = s.pop()
            if tmp in outdegree:
                children = outdegree[tmp]
                for c in children:
                    indegree[c].remove(tmp)
                    if not indegree[c]:
                        s.append(c)
                        res.append(c)

        node = set(outdegree.keys()+indegree.keys())
        return res if len(res)==len(node) else []
```

```
graph = {1:[2,3,4,5],
        2:[3,4],
        3:[5]}
so = Solution()
a = so.toposort(graph)
print a
```

```
class Solution(object):
    def toposort(self, graph):
        """
        :type graph: dict[int:List[int]]
        :rtype: List[int]
        """
        children = []
        for k in graph:
            children += graph[k]
        s = [k for k in graph if k not in set(children)]

        res = []
        while s:
            while s[-1] in graph and graph[s[-1]]:
                c = graph[s[-1]].pop()
                if c in s:  # find circle
                    return []
                if c not in res:  # have visited node
                    s.append(c)
            res.append(s.pop())

        print res[::-1]
```

不管了，不管了。拿不到奥佛就算了！

yp

里面是我整理的yp的所有的面经。 P1 String Reduction: 给一个字符串，把所有连续出现两次或以上的字符缩减到一个，比如“aaabbbccdccc”变成“3a3b2cd3c”

```
def compress(s):
    if not s:
        return None
    index = 0
    res = ''
    while index<len(s):
        cur = s[index]
        cnt = 1
        while (index+1)<len(s) and s[index+1]==s[index]:
            index += 1
            cnt += 1
        res += cur + (str(cnt) if cnt>1 else '')
        index += 1
    return res


s = 'abbdceccc'
ans = compress(s)
print(ans)
```

P2 Merge two sorted linked list.

```
class Node(object):
    def __init__(self, val=None):
        self.val = val
        self.next = None


def mergeLinkedList(l1, l2):
    dummy = Node()
    pre = dummy

    while l1 and l2:
        if l1.val<l2.val:
            dummy.next = l1
            l1 = l1.next
        else:
            dummy.next = l2
            l2 = l2.next
        dummy = dummy.next

    if l1 or l2:
        dummy.next = l1 or l2

    return pre.next

a = Node(0)
b = Node(1)
c = Node(2)
d = Node(3)
a.next = b
c.next = d
head =  mergeLinkedList(a, c)
while head:
    print head.val
    head = head.next
```

昨晚做了YELP OA。。。 地里原题，好像很简单 就是给你一堆BusinessInfo{int id, int rating}, 根据rating 从大到小排序 直接sort 刚做的Yelp OA.题目是有一个BusinessInfo class. 有两个参数 Rating(Integer). Name(String).. 输入一个List 要求对这个list进行排序根据 rating从大到小。

```
class businessObject(object):
    def __init__(self, name=None, rating = None):
        self.name = name
        self.rating = rating

def sortRes(l):

    l.sort(key=lambda x: x.rating)
    return l

l = [businessObject('a', 5),
    businessObject('b', 4),
    businessObject('c', 3),
    businessObject('d', 2),
    businessObject('e', 1)]

a = [i.name for i in sortRes(l)]
print a
```

给两个list 这俩list里面有相同元素 找到index的和是最小的对应这个元素是啥

给一个list，每个element有id和rating. Waral 输出rating的中位数 直接排序，输出中位数即可

top color: http://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=201910&extra=page%3D1%26filter%3Dsortid%26sortid%3D311%26searchoption%5B3089%5D%5Bvalue%5D%5B5%5D%3D5%26searchoption%5B3089%5D%5Btype%5D%3Dcheckbox%26searchoption%5B3046%5D%5Bvalue%5D%3D36%26searchoption%5B3046%5D%5Btype%5D%3Dradio%26sortid%3D311

之前还练习过一道排列 color的题，输入二维的list, { "red", "black","green" "white", "red", "black".

上个月HR发来的OA拖到昨天才做，15分钟，题跟地里其他的同学一样，给一个BusinessInfo{name, rating}的vector，求rating的medium。 写了个custom comparator排序做出来的

只有一道题，15分鐘。給個matrix of strings，找出現最多次的strings（可以超過一個）。答案要 alphabetically sorted。. e.g., [["blue", "blue", "black"], ["black", "red", "yellow", "green"]. 1point 3acres 璁哄潵 ["yellow", "green"]]

Answer: a vector of {"black", "blue", "green", "yellow"} 因爲都出現兩次。 . more info on 1point3acres.com 問題有些細節忘了，大家拿到還是認眞看下题。

海投的Yelp 2016 fall intern。没想到很快有hr联系我，收到OA V4 说1-2周内做完，硬是拖到了最后。还问了下有没有summer的机会，告诉我已经满了。 OA题目没有变，还是Anagram Checker，大家可以参考其他。没有简答题，只有一道编程题。但是有个奇怪的事是，我好像只有15min，刚打开题大概扫了一下，然后看时间发现只有14分多了，吓了一身冷汗。赶紧按照提前准备的写完了。 我没有处理exception，但hr反馈说Everything looks great。然后准备约电面。

有个class 是customer 有id 和 rating, input是一个 list of customer, 写了compareTo function, return double 是一个 median of rating.

大家好啊，刚刚做完yelp OA V7, 收到OA的时间是 4/15 题目： 给一个 bussinessInfo 的类，里面有id 和catagory, 是这样： class bussinessInfo{int id, String catagory}

输入： 101 japanese, sushi, resturant. 102 japanese, seafood 103 japanese, resturant, ramen . more info on 1point3acres.com 输出： 101, 103 即找出所有的id 即是 japanese 又是resturant

上周HR给发的OA5，今天晚上才有时间抽空做了一下. 地里面的资料都很详细了，OA分为两部分：1. 简答和选择：地里面的面经已经可以全部覆盖了，所以就不多说了。

1. coding题：potential palindrome，之前地里面也有人提到过。由于是在hackerrank上做，需要自己写stdin和stdout.

yp 1-10

1 139 word break

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

```
class Solution(object):
    def wordBreak(self, s, wordDict):
        """
        :type s: str
        :type wordDict: Set[str]
        :rtype: bool
        """
        dp = [False]*len(s)

        for i in range(len(s)):
            for j in range(i,len(s)):
                if i==0 and s[i:j+1] in wordDict:
                    dp[j] = True
                elif dp[i-1]==True and s[i:j+1] in wordDict:
                    dp[j] = True

        return dp[-1]
```

```
class Solution(object):
    def wordBreak(self, s, wordDict):
        """
        :type s: str
        :type wordDict: Set[str]
        :rtype: bool
        """
        dp = [False]*(len(s)+1)
        dp[0]=True

        for i in range(len(s)+1):
            for w in wordDict:
                l = len(w)
                if l<=i and s[i-l:i]==w:
                    dp[i] = dp[i-l] or dp[i]
        return dp[-1]
```

2 merge intervals 扫描线算法，没什么好说的。

```
class Solution(object):
    def merge(self, intervals):
        """
        :type intervals: List[Interval]
        :rtype: List[Interval]
        """

        times = []
        for s,e in intervals:
            times.append((s,0))
            times.append((e,1))

        times.sort()
        s = []
        res = []
        for t in times:
            if t[1]==0:
                s.append(t)
            else:
                tmp = s.pop()
                if not s:
                    res.append([tmp[0], t[0]])
        return res


#[1,6],[8,10],[15,18]
intervals = [[1,3],[2,6],[8,10],[15,18]]
so = Solution()
a = so.merge(intervals)
print(a)
```

3 generate parenthesis

```
class Solution(object):
    def generateParenthesis(self, n):
        """
        :type n: int
        :rtype: List[str]
        """
        res = []
        self.dfs(n, n, '', res)
        return res


    def dfs(self, l, r, cur, res):

        if l<0 or r<0:
            return

        if l==0 and r==0:
            res.append(cur)

        self.dfs(l-1, r, cur+'(', res)

        if r>l:
            self.dfs(l, r-1, cur+')', res)


so = Solution()
ans = so.generateParenthesis(4)
print(ans)
```

4 ，输入一个string，全部都是小写，然后把其中的某些变成大写，输出all combinations。follow up是问有多少种可能，我答的是n的阶乘，现在一想发现是2的n次方

```
class Solution(object):
    def shift(self, s):
        """
        :type n: int
        :rtype: List[str]
        """
        res = []
        self.dfs(0, s, '', res)
        return res

    def dfs(self, i, s, cur, res):
        if i==len(s):
            res.append(cur)
            return
        self.dfs(i+1, s, cur+s[i].upper(), res)
        self.dfs(i+1, s, cur+s[i], res)


s = 'abc'
so = Solution()
a = so.shift(s)
print(a)
```

5 224 basic calculator 每天写一遍basic calculator有益于身心健康。

```
class Solution(object):
    def basicCaculator(self, s):
        """
        :type n: int
        :rtype: List[str]
        """
        num, nums, ops = 0, [0], []
        s = s + '#'

        for i in range(len(s)):
            if s[i] in '0123456789':
                num += num*10 + int(s[i])
            else:
                if i>0 and s[i-1] not in '+-*/()':
```

```
                    nums.append(num)
                    num = 0
                if s[i] in '+-*/':
                    if ops and ops[-1] in '*/':
                        y, x, op = nums.pop(), nums.pop(), ops.p
op()
                        nums.append(self.applyOps(x, y, op))
                    ops.append(s[i])
                elif s[i]=='(':
                    ops.append(s[i])
                elif s[i]==')':
                    while ops[-1]!='(':
                        y, x, op = nums.pop(), nums.pop(), ops.p
op()
                        nums.append(self.applyOps(x, y, op))
                    ops.pop()

        while ops:
            y, x, op = nums.pop(), nums.pop(), ops.pop()
            nums.append(self.applyOps(x, y, op))
        return nums[-1]

    def applyOps(self, x, y, op):

        if op=='+':
            return x + y
        elif op=='-':
            return x - y
        elif op=='*':
            return x * y
        else:
            if x//y<0 and x%y:
                return x//y+1
            else:
                return x//y

s = '1*(2-3)'
so = Solution()
a = so.basicCaculator(s)
print(a)
```

6 Integer to English Words 电面没做过的，应该现场来不及写完吧，讲道理是这样的。

```
class Solution(object):
    def __init__(self):
        self.less20 = ["","One","Two","Three","Four","Five","Six"
,"Seven","Eight","Nine","Ten","Eleven","Twelve","Thirteen","Four
teen","Fifteen","Sixteen","Seventeen","Eighteen","Nineteen"]
        self.tens = ["","Ten","Twenty","Thirty","Forty","Fifty",
"Sixty","Seventy","Eighty","Ninety"]
        self.thousands = ["","Thousand","Million","Billion"]

    def numberToWords(self, num):
        """
        :type num: int
        :rtype: str
        """
        res = ''
        index = 0
        while num:
            res = self.helper(num%1000) + ' ' + self.thousands[i
ndex] + ' ' + res
            index += 1
            num = num//1000

        return res

    def helper(self, num):

        res = ''
        if num>=100:
            res = self.less20[num//100] + ' ' + 'Hundred'
            num = num%100
        if num >=20:
            res = res + (' ' if res else '') + self.tens[num//10
] + (' ' if self.tens[num//10] else '') + self.less20[num%10]
        else:
            res = res + (' ' if res else '') + self.less20[num]
        return res
```

```
num = 2111111121
so = Solution()
a = so.numberToWords(num)
print(a)
```

7 Write a function that takes two arguments, a list of words and an integer n, and returns the nth most common word in the list.

e.g. f(['cat', 'dog', 'cat'], 1) => 'cat' (the 1st most common word) f(['cat', 'dog', 'cat'], 2) => 'dog' (the 2nd most common word)

奉献给大家

```
import collections
class Solution(object):
    def kthcommon(self, words, k):
        """
        :type wrods: List[str]
        :type k: int
        :rtype: str
        """
        return collections.Counter(words).most_common(k)[k-1][0]

words = ['cat', 'dog', 'cat']
so = Solution()
a = so.kthcommon(words, 2)
print(a)
```

8 longest-consecutive-sequence

```
import collections
class Solution(object):
    def longest_consecutive_sequence(self, nums):

        s = set(nums)
        res = 0
        for num in nums:
            if (num-1) not in s:
                tmp = 0
                while num in s:
                    tmp += 1
                    num += 1
                res = max(res, tmp)
        return res

nums = [100, 4, 200, 1, 3, 2]
so = Solution()
a = so.longest_consecutive_sequence(nums)
print(a)
```

9 电面45分钟

Input 2D平面四个点 第一问判断是不是一个square

不一定要跟x轴 y轴平行 (Square 可以旋转)

代码大致上像是:

```
struct Point {
  double x, y;
  Point(double x_, double y_) : x(x_), y(y_) {}
};

bool isSquare(con****st vector<Point>& points) {
...
}
```

不会做！碰到我就跪地求饶好啦！

```
class Solution(object):
    def ifSquare(self, points):
        q = points.pop()

        res = []
        for p in points:
            dist = (abs(p[0]-q[0])**2 + abs(p[1]-q[1])**2)**(0.5
)
            res.append((dist, p))

        m = max(res)
        res.remove(m)
        x, y = res[0], res[1]
        print(m, x, y, (x[0]**2 + y[0]**2)**(0.5))
        if x[0]!=y[0] or m[0]!=(x[0]**2 + y[0]**2)**(0.5) or\
        m[0]!=(abs(x[1][0]-y[1][0])**2 + abs(x[1][1]-y[1][1])**2
)**(0.5):
            return False
        return True
```

第二问 是 return 四个点围成的形状: "rectangle", "square", "diamond", “parallelogram",. ...

10 最后来说coding题吧，就是一个dart game，然后有一群possible scores,然后给你一个candidate score，问这个candidate score是不是可能的得分。For example:possible scores: [3,5,3,3,5] (有duplicates，unsorted，可能有很多很多possible score，不一定只有3和5) candidate score: 6, yes candidate score: 11, yes candidate score: 7, no candidate score: 0, yes 我感觉这题有点像leetcode combination sum那题，但是那题是用回溯求出所有可能解，这题是判断是否可以，想写个recursion的方法，试了几次还是弄不好，求各位大神帮忙。

```
class Solution(object):
    def dartGame(self, scores, target):
        """
        :type scores: List[int]
        :rtype: Bool
        """
        scores.sort()
        return self.dfs(scores, target)


    def dfs(self, scores, target):
        if target==0:
            return True

        if target<0 or not scores:
            return False

        for i in range(len(scores)):
            if scores[i]<=target:
                if self.dfs(scores[i+1:], target-scores[i]):
                    return True
        return False


scores = [3,5,3,3,5]
target = 7
so = Solution()
a = so.dartGame(scores, target)
print a
```

yp 11-20

P11 coding题是字符串乘法，全是bug，懒得说了。。。

```
class Solution(object):
    def multiply(self, num1, num2):
        """
        :type num1: str
        :type num2: str
        :rtype: str
        """
        num1, num2 = num1[::-1], num2[::-1]
        res = [0]*(len(num1)+len(num2))

        for i in range(len(num1)):
            for j in range(len(num2)):
                tmp = int(num1[i])*int(num2[j])
                cur = res[i+j]+tmp
                res[i+j] = cur%10
                res[i+j+1] += cur//10
                print res



        while len(res)>1 and res[-1]==0:
            res.pop()

        return ''.join(map(str, res[::-1]))


num1 = '999'
num2 = '9999'
so = Solution()
a = so.multiply(num1, num2)
print(a)
```

P12 maximum subarray

```
class Solution(object):
    def maxSubArray(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """

        pre = 0
        res = float('-inf')
        for num in nums:
            pre = max(num, num+pre)
            res = max(res,pre)

        return res


nums = [-2,1,-3,4,-1,2,1,-5,4]
so = Solution()
a = so.maxSubArray(nums)
print a
```

P13 Coin Change

```
class Solution(object):
    def coinChange(self, coins, amount):
        """
        :type coins: List[int]
        :type amount: int
        :rtype: int
        """
        self.res = float('inf')
        coins = sorted(coins, reverse=True)
        self.dfs(coins, 0, amount)
        return self.res if self.res!=float('inf') else -1

    def dfs(self, coins, cnt, amount):
        if amount<0:
            return
        if amount==0:
            self.res = min(self.res, cnt)
        for c in coins:
            if c<=amount<c*(self.res-cnt):
                self.dfs(coins, cnt+1, amount-c)

scores = [1,2,5]
target = 1000
so = Solution()
a = so.coinChange(scores, target)
print a
```

DP解法

```
class Solution(object):
    def coinChange(self, coins, amount):
        """
        :type coins: List[int]
        :type amount: int
        :rtype: int
        """
        dp = [0]*(amount+1)
        dp[0] = 0

        for i in range(amount+1):
            tmp = []
            for c in coins:
                if c<=i and (dp[i-c]!=0 or (i-c)==0):
                    tmp.append(dp[i-c]+1)
            if tmp:
                dp[i] = min(tmp)

        return dp[-1] if dp[-1] else -1

scores = [1,2,5]
target = 11
so = Solution()
a = so.coinChange(scores, target)
print a
```

yp onsite 1-10

已经跪了辣么多公司啦。这个dream company可是不能再跪啦！ 奔跑吧咸鱼！

P1 How do I find the greatest sum on a path from root to leaf (that can only touch one node per level—no going up and down) in this tree like structure? Question Quora link

```
class Solution(object):
    def findMax(self, root):
        if not root:
            return 0
        left = self.findMax(root.left)
        right = self.findMax(root.right)
        return max(left, right) + root.val
```

这道题抽空写一下吧 https://leetcode.com/problems/binary-tree-maximum-path-sum/

P2 第一轮面，上来就问 why yelp，然后问简历的project。完了之后做题目，leetcode 121 best time to buy and sell stock 原题。我一上来居然想到了trapping water那题。。。用了两个数组来存某个位置其左边最小值，和其右边最大值。在面试官引导优化了空间，最后其实不用额外空间就可以的。。。

https://leetcode.com/problems/best-time-to-buy-and-sell-stock/

P3 第二轮面，还是问 why yelp，然后问简历的project。然后是一道 smallest k elements in an array的题，用个max heap搞定。我感觉挺满意。结果面试官又来一题。简易版的 lc 290 word pattern， 比如 foo 对应 app 就是对的，要是 foo 对应 apg 那就是错啦。很简单吧。结果我上来只用了一个HashMap，写完了后，他让我写test case，我自己写了下，发现不对，赶紧又加了一个HashMap.. 险过。

P4 第三轮面，还是问 why yelp，然后问简历的project。需要提一下，每轮面试官结束后，他会和下个面试官交流个两三分钟，然后下个面试官再进来。我前两轮介绍的project都是我最熟的那个，结果第三个面试官就直接问我，你咋总是讲那一个project。。。 好吧，换个project开始侃了。侃完了后，出一道题，懵了。

你们帮我看一看。题目是：给一串movie，假定每个movie都是一个小时，并且都在整点播放；给你一个List的movies，让你安排播放时间，如果没有答案throw一个exception。

比如 电影A: 14, 15, 16, 17 电影B: 14, 15, 16 电影C: 14, 15 电影D: 14, 15, 17

比如 电影A: 14, 15, 16, 17 电影B: 14, 15, 16 电影C: 14, 15 电影D: 14, 15, 17

返回一个解就行，比如 A 14, B 16, C 15, D 17。 如果你要 A 14, B 15, 后面 C就没法看了。

P5 然后上题，设计一个Rate limitor。就是一小时内访问第六次时，就返回false。这也没啥算法啊，我没太明白，就是HashMap，存访问的IP和ArrayList\，然后同一个IP一小时内访问第六次，那就返回 false呗。面试官说就是这样的，然后我用java把这个函数写出来。然后讨论了些他做的工作啊什么的，结束了

P6 coding：设计一个函数，给定一个list，能够返回一个内部元素random permutation过后的该list。很快写出一共不到10行，随后问一些基于permutation的概率题，最后说你如何测试该函数。楼主答的一测试edge cases，二测试overall results是否uniformly distributed。由于并不是很懂伪随机的原理，可能在这里答一些伪随机的内容会更好。

P7 1轮skype的美国大哥：聊简历，coding题是给你一群nodes，每一个node里有3个invariants: unique的id，可能重名的label，其parent的id。要求将一组乱序的nodes按照如下方式print出来： Node A Node B Node C Node D Node E 这里BD的parent为A，C的为B，AE的为一个id=0的默认root（不print），也就是子node按照depth要缩进相应的空格。 这题不难，我是先遍历nodes重建了储存children关系的hash table，然后写了一个基于stack的方法，中间卡了下壳，还好大哥人很nice，提示了下，就“looks great to me”了。Follow up是“我看你写的是iteration，告诉我写成recursion的话pros和cons是啥”，我答了一些基本比较，他满意，后面就闲聊

P8 2轮中国大哥：一个很典型的干瘦中国人，搞deep learning的，问题很尖锐。聊简历：问的很深，HMM,GMM,CNN,EM等。设计题：打开yelp，输入关键词restaurant返回一个附近restaurant的候选列表，问这一功能如何实现？答的search+filter+ranking，后来问我为何你用这种ranking方法不用其他的，这应该是一个开放性问题。coding：设计一个函数，给定一个list，能够返回一个内部元素random permutation过后的该list。很快写出一共不到10行，随后问一些基于

permutation的概率题，最后说你如何测试该函数。楼主答的一测试edge cases，二测试overall results是否uniformly distributed。由于并不是很懂伪随机的原理，可能在这里答一些伪随机的内容会更好

P9 3轮美国小哥：Physics PhD帅哥。聊简历，设计题：yelp现在要给附近一些新发掘的店铺定制category label，问你如何根据user对该新店铺的review得知该店铺属于restaurant, gas station,...的哪一个。classification问题，也是开放性的。Followup是如何对应“I just had a meal nextdoor”这种容易将一般店铺误判为restaurant的review。coding题出乎意料的简单。说是yelp每次有人访问会给一个临时id，6位digits，问你如何判断这6数字里有重复的...一行搞定，第二问是给了一个当天所有临时id的log，求今天id中带有重复数字的比率是多少...套用前问结果10行搞定，最后又来permutation相关的概率题，但都很简单。

P8 4轮三哥：不好好听就提问打断，玩手机，全程站起来转椅子玩（醉了），不交流不帮助不给思考时间，楼主卡壳他就直接下一题。聊简历，coding题是leetcode原题longest palindromic substring，还问我之前见过这题没。我说没，然后想不起来O(n)的方法了就只好写了最土的从每个字符往两边搜的O(n^2)方法。Followup是我只有一个只能获得最长odd length palindrome的函数f_odd，问如何调用这个得到原函数，也就是 f (s) = g1(f_odd(g2(s)))，求g1，g2。楼主答了给原string插入#再处理，最后得到结果后每隔一个字去除#，被问原string含#怎么办，卡壳不到30秒就说算了我问你behavioral question，然后心不在焉地听了会就结束了。

P2 TinyURL

P3 RATE LIMITER

P4 121 Best Time to Buy and Sell Stock

P5 word pattern

Trie

Shortest Classifier

给出几个二进制数...如何找到最小的连续bit位数以区别这几个二进制数？比如我1000 0100 0010这三个数...我只要前两位就能区别了。

思路： 解法就是我们建一个Trie树，把所有的数都存到Trie里。在这个建树的过程中我们找到一个最晚分叉的位置和一个最早分叉的位置

怎么找这个最晚分叉的位置呢。就是我们在造树过程中，如果遇到其中一个节点，树里面没有。那么这个深度就是对于这个数来说我们需要的classify的节点深度。所有的这些深度中，最大的，就是我们要找的最小深度。（有点难理解哈。）

然后我们要找这个树最早是在哪里分叉的来得到minDepth。这样我们从minDepth取到maxDepth就可以把最短的给找出来了。

以1000 0100 0010举例。先把1000放到trie里面。 所以深度是1，因为第一个node我们就要新建。然后接着放0100，深度也是1，因为第一个Node我们要新建。接着放0010，这时候深度为2，因为0已经在Trie中了。所以maxDepth是2.而第一个就分叉了，随意minDepth是1。

```
class Solution():
    def minPrefixLength(self, nums):
        """
        :type nums: List[str]
        :rtype: int
        """
        trie = {}
        maxDepth = 0

        for num in nums:
            depth, checked = 0, False
            root = trie
            for c in num:
                # when node is not in trie
                if c not in root:
                    # only compare depth of first node we create
                    if not checked:
                        maxDepth = max(depth, maxDepth)
                        checked = True
                    root[c] = {}
                root = root[c]
                # increase depth
                depth += 1

        # get the min index of the subarray
        root = trie
        minDepth = 0
        while len(root)==1:
            minDepth += 1
            root = root.items()[0][1]

        return maxDepth+1 - minDepth

nums = ['11110100', '11111000', '11111100']
so = Solution()
ans = so.minPrefixLength(nums)
print ans
```

2 Auto fill words

Implement a auto fill class. It will take a word dictionary for init. then it will auto fill the word you are entered. You can understand this by looking at my code.

```
# word = ['app', 'apple', 'application', 'appeal', 'ape', approv
e', 'apply', 'applicant']

class word(object):

    def __init__(self):
        self.dic = {}
        self.isword = False

class Solution(object):

    def __init__(self, words):
        self.word_freq = {}
        self.trie = None
        self.build_freq(words)
        self.build_trie(words)

    def build_freq(self, words):
        for w in words:
            self.word_freq[w] = 0

    def build_trie(self, words):
        trie = word()
        for i in words:
            tmp = trie
            for c in i:
                if c not in tmp.dic:
                    tmp.dic[c] = word()
                tmp = tmp.dic[c]
            tmp.isword = True
        self.trie = trie

    def get_words(self, prefix):
```

```
        res = []
        tmp = self.trie
        cur = prefix
        s = [tmp]

        # search in the trie untill finish prefix
        for c in prefix:
            if c not in tmp.dic:
                return []
            else:
                tmp = tmp.dic[c]

        if tmp.isword:
            res.append(res)

        # dfs the rest trie to find all the words start from thi
s prefix
        self.dfs(tmp, prefix, res)

        # sorted res based on search times
        res = self.sort_helper(res)

        return res

    def pick(self, word):
        self.word_freq[word] += 1

    def sort_helper(self, words):
        count = [self.word_freq[word] for word in words]
        tmp = zip(count, words)
        res = sorted(tmp, key = lambda x: x[0])
        res = [i[1] for i in res]
        return res[::-1]

    def dfs(self, trieNode, cur, res):
        # if it is a word
        if trieNode.isword:
            res.append(cur)

        # No children for the node
```

```
        if not trieNode.dic:
            return

        # Go through all the keys to find the words.
        for key in trieNode.dic.keys():
            self.dfs(trieNode.dic[key], cur+key, res)


words = ['app', 'apple', 'application', 'appeal', 'ape', 'approv
e', 'apply', 'applicant', 'age', 'add', 'average', 'boy', 'bat']


s = Solution(words)
# first we get suggestuon for prefix ap
print s.get_words('ap')
# then we get suggestion for prefix b
print s.get_words('b')
# we decided to pick bat as our input
s.pick('bat')
# next time b will be ranked higher
print s.get_words('b')
s.pick('appeal')
print s.get_words('a')
```

3 Word Search II

Given a 2D board and a list of words from the dictionary, find all words in the board.

Each word must be constructed from letters of sequentially adjacent cell, where "adjacent" cells are those horizontally or vertically neighboring. The same letter cell may not be used more than once in a word.

For example, Given words =

```
["oath","pea","eat","rain"]
```

and board =

```
[
  ['o','a','a','n'],
  ['e','t','a','e'],
  ['i','h','k','r'],
  ['i','f','l','v']
]
```

Return ["eat","oath"].

https://leetcode.com/problems/word-search-ii/

思路：这其实是一道非常好的集合了2D arrar DFS和Trie的题目。看到这道题，最暴力的方法就是，直接在2D array里面一个一个单词地找。可是这样就复杂度感人了。因为首先你要loop 2D array找首字母可能的起始位置，然后从每个可能的起始位置dfs。

我们先把所有的单词建trie。然后对2D array的每一个位置根据trie来一层一层dfs。这样我们过一遍matrix就行了。在对每个位置根据trie进行dfs的时候，我们要注意不要再访问访问过的节点，然后就可以看出这里有点backtracking的意思了。我们继续dfs到其他位置的时候把这个点表成其他‘#’，其实存一个visited的矩阵更安全，因为matrix里的字符有可能就是‘#’。

最后结果会出现4次是因为，走到一个单词的结尾的时候，会继续从4个方向dfs下去。所以这时候我们在存好第一个的时候，要删除isword的flag，避免重复存储。

```
class Node(object):
    def __init__(self):
        self.dic = {}
        self.isWord = False

class Solution(object):
    def findWords(self, board, words):
        """
        :type board: List[List[str]]
        :type words: List[str]
        :rtype: List[str]
        """
        root = Node()

        for w in words:
            cur = root
            for c in w:
                if c not in cur.dic:
                    cur.dic[c] = Node()
                cur = cur.dic[c]
            cur.isWord = True

        res = []
        for i in range(len(board)):
            for j in range(len(board[0])):
                self.bt(i, j, board, '', root, res)
        return res

    def bt(self, i, j, board, path, root, res):
        if root.isWord:
            res.append(path)
            root.isWord = False

        if 0<=i<len(board) and 0<=j<len(board[0]):
            c  = board[i][j]
            if c in root.dic:
                board[i][j]='#'
```

```
                self.bt(i-1, j, board, path+c, root.dic[c], res)
                self.bt(i+1, j, board, path+c, root.dic[c], res)
                self.bt(i, j+1, board, path+c, root.dic[c], res)
                self.bt(i, j-1, board, path+c, root.dic[c], res)
                board[i][j]=c
```

For fun

Big Integer

Big Integer, 带小数的加法。 好恶心啊！先按照小数点. 来split两部分。然后先加小数，然后把小数可能的进位丢进整数部分一起算。

```
class Solution(object):
    def add(self, num1, num2):
        if '.' in num1 and '.' in num2:
            inte1, deci1 = num1.split('.')
            inte2, deci2 = num2.split('.')
        decimal, carry = self.doDeci(deci1, deci2)
        integer = self.doInt(inte1, inte2, carry)

        return '.'.join([integer, decimal])

    def doDeci(self, num1, num2):
        num1 = list(map(int, list(num1)))
        num2 = list(map(int, list(num2)))
        if len(num1)<len(num2):
            num1, num2 = num2, num1

        carry = 0
        for i in range(len(num2)-1, -1, -1):
            tmp = num2[i] + num1[i] + carry
            num1[i] = tmp%10
            carry = tmp//10

        res = list(map(str, num1))
        return ''.join(res), carry

    def doInt(self, num1, num2, carry):
        num1 = list(map(int,num1))[::-1]
        num2 =list(map(int,num2))[::-1]
        if len(num1)<len(num2):
            num1, num2 = num2, num1

        index = 0
        while index<len(num1):
```

```
            n2 = int(num2[index]) if index<len(num2) else 0
            tmp = n2 + num1[index] + carry
            num1[index] = tmp%10
            carry = tmp//10
            index += 1

        if carry: num1.append(1)
        num1 = list(map(str, num1))
        return ''.join(num1[::-1])


num1 = '20.78'
num2 = '1.5679'
so = Solution()
ans = so.add(num1, num2)
print(ans)
```

不带小数的加法和乘法。

```
class Solution(object):
    def add(self, num1, num2):
        """
        :type num1: str
        :type num2: str
        :rtype: str
        """
        num1 = list(num1)[::-1]
        num2 = list(num2)[::-1]

        if len(num1)<len(num2):
            num1, num2 = num2, num1

        carry = 0
        index = 0
        while index<len(num1):
            n2 = int(num2[index]) if index<len(num2) else 0
            tmp = n2 + int(num1[index]) + carry
            cur = tmp%10
            carry = tmp//10
```

```
            num1[index] = str(cur)
            index += 1

        if carry: num1.append('1')

        return ''.join(num1[::-1])

    def multiply(self, num1, num2):
        """
        :type num1: str
        :type num2: str
        :rtype: str
        """
        num1 = num1[::-1]
        num2 = num2[::-1]

        res = [0]*(len(num1) + len(num2))
        for i in range(len(num2)):
            for j in range(len(num1)):
                tmp = int(num2[i])*int(num1[j])
                res[j+i] += tmp
                res[j+i+1] += res[j+i]//10
                res[j+i] = res[j+i]%10

        while len(res)>1 and res[-1]==0:
            res.pop()

        return ''.join(map(str,res))[::-1]

num1 = '124'
num2 = '5679'
so = Solution()
a = so.add(num1, num2)
print(a)
a = so.multiply(num1, num2)
print(a)
```

Generate longest word

given a bag of chars, and a dictionary contains with many words, find list of maximum length word which can be generated from the bag of chars respectively.
字符集中的字符不一定全用上。

bag may contain duplicate characters.The char in bag cannot be reused. bag {a,p,l,e} cannot generate word 'apple'.

case 2: bag of chars

```
{'a', 'p', 'p', 'l', 'e', 'o'}
```

dict

```
{'apple', 'people'}
```

return

```
{'apple'}
```

because people need 2 char 'e', there is only 1 'e' in the bag.

思路：我觉得这道题用trie不好写。

因为你用单词来build trie的话。相当于你要用bag of chars去dfs每一个trie的path。然后一个一个char的删除，直到hit到word最后一个字母，或者节点不在bag里面。如果能hit到word最后一个字母，记录这个word,和深度。最后比较所有能dfs出来的word的长度。感觉不是很酷炫。

这个我们提供一个更酷炫的思路。你不是要保证你的单词所有字母在bag里面么，本宝宝直接用一个counter来存你的bag。key是字母，value是有多少个。然后我把每一个word都做成couter，直接并集。如果bag的counter不变的话，那么说明这个word就被bag给包括了。最后直接存最长那个就行了。

举例Counter object是怎么回事

```
bag = ['a','a','b','r']
counter = {'a':2, 'b':1, 'r':1}
```

代码如下：

```
import collections
class Solution(object):
    def findLongest(self, bag, words):
        """
        :type bag: List[char]
        :type words: List[str]
        :rtype: str
        """
        d = collections.Counter(bag)
        res = []
        for w in words:
            tmp = collections.Counter(w)
            if d == (d|tmp):
                if not res:
                    res.append(w)
                else:
                    if len(set(w))>len(set(res[0])):
                        res = [w]
                    elif len(set(w))==len(set(res[0])):
                        res.append(w)
        return res

bag = ['u','b','e','r','d']
words = ['uber', 'red']

so = Solution()
ans = so.findLongest(bag, words)
print(ans)

bag = ['a','p','p','l','e','o']
words = {'apple', 'people'}
ans = so.findLongest(bag, words)
print(ans)

bag = ['i','n','b','o','x','d','r','a','f','t','m','r','e']
words = {'inbox', 'draft', 'more'}
ans = so.findLongest(bag, words)
print(ans)
```

340 Longest Substring with At Most K Distinct Characters

Given a string, find the length of the longest substring T that contains at most k distinct characters.

For example, Given s = “eceba” and k = 2,

T is "ece" which its length is 3.

思路：典型的滑动窗口，双指针的问题。如果外面直接套while，太容易写错了。所以直接for loop j。然后移动i来确保每次都是valid的。用res来存最大的结果就行了。

```
import collections
class Solution(object):
    def lengthOfLongestSubstringKDistinct(self, s, k):
        d = collections.defaultdict(int)
        i, res = 0, 0
        for j in range(len(s)):
            d[s[j]] += 1
            # while have more than k, we keep removing from i
            while len(d)>k:
                d[s[i]] -= 1
                if d[s[i]] == 0:
                    del d[s[i]]
                i += 1
            res = max(res, j-i+1)
        return res


s = 'ab'
k = 1
so = Solution()
a = so.lengthOfLongestSubstringKDistinct(s, k)
print a
```

另外一种写法

```
class Solution(object):
    def lengthOfLongestSubstringKDistinct(self, s, k):
        """
        :type s: str
        :type k: int
        :rtype: int
        """
        d = {}
        low, res = 0, 0
        for i,c in enumerate(s):
            d[c] = i

            # delete a element from the dict and reset the window

            if len(d)>k:
                index = min(d.values())
                low = index + 1
                del d[s[index]]
            res = max(res, i-low+1)
        return res
```

BFS

126 Word ladder II

beginWord = "hit"

endWord = "cog"

wordList = ["hot","dot","dog","lot","log"]

As one shortest transformation is "hit" -> "hot" -> "dot" -> "dog" -> "cog", return its length 5.

https://leetcode.com/problems/word-ladder-ii/

思路： 就是一个DFS。然后存一个visited免得重走。

```
import collections
class Solution(object):
    def ladderLength(self, beginWord, endWord, wordList):
        """
        :type beginWord: str
        :type endWord: str
        :type wordList: Set[str]
        :rtype: int
        """
        # construct next possible dic
        dic = collections.defaultdict(list)
        wordList.add(endWord)
        for w in wordList:
            for i in range(len(w)):
                tmp = w[:i] + '_' + w[i+1:]
                dic[tmp].append(w)

        q = [[beginWord, [beginWord]]]

        visited = set()
        # BFS
        while q:
            k = len(q)
            for i in range(k):
```

```
                tmp = q.pop(0)
                w, path = tmp[0], tmp[1]
                # if hit the end. return
                if w==endWord:
                    return len(path)
                # else construct next possible path
                if w not in visited:
                    visited.add(w)
                    for i in range(len(w)):
                        key = w[:i] + '_' + w[i+1:]
                        if key in dic:
                            children = dic[key]
                            for c in children:
                                if c not in path:
                                    new = path + [c]
                                    q.append([c, new])
        return 0



wordList = {"hot","dot","dog","lot","log"}
beginWord = "hot"
endWord = "dog"

so = Solution()
a = so.ladderLength(beginWord, endWord, wordList)
print(a)
```

317 Shortest Distance from All Buildings

You want to build a house on an empty land which reaches all buildings in the shortest amount of distance. You can only move up, down, left and right. You are given a 2D grid of values 0, 1 or 2, where:

Each 0 marks an empty land which you can pass by freely. Each 1 marks a building which you cannot pass through. Each 2 marks an obstacle which you cannot pass through. For example, given three buildings at (0,0), (0,4), (2,2), and an obstacle at (0,2):

```
1 - 0 - 2 - 0 - 1
|   |   |   |   |
0 - 0 - 0 - 0 - 0
|   |   |   |   |
0 - 0 - 1 - 0 - 0
```

The point (1,2) is an ideal empty land to build a house, as the total travel distance of 3+3+1=7 is minimal. So return 7.

https://leetcode.com/problems/shortest-distance-from-all-buildings/

思路： 我们用一个matrix来记录能到达i,j位置的building数和距离的和。然后对每一个grid值为1的点进行bfs就行了。

```
class Solution(object):
    def shortestDistance(self, grid):
        """
        :type grid: List[List[int]]
        :rtype: int
        """
        if not grid or not grid[0]:
            return -1

        matrix = [[[0,0] for i in range(len(grid[0]))] for j in
range(len(grid))]
```

```
        cnt = 0    # count how many building we have visited
        for i in range(len(grid)):
            for j in range(len(grid[0])):
                if grid[i][j] == 1:
                    self.bfs([i,j], grid, matrix, cnt)
                    cnt += 1

        res = float('inf')
        for i in range(len(matrix)):
            for j in range(len(matrix[0])):
                if matrix[i][j][1]==cnt:
                    res = min(res, matrix[i][j][0])

        return res if res!=float('inf') else -1

    def bfs(self, start, grid, matrix, cnt):
        q = [(start, 0)]
        while q:
            tmp = q.pop(0)
            po, step = tmp[0], tmp[1]
            for dp in [(-1,0), (1,0), (0,1), (0,-1)]:
                i, j = po[0]+dp[0], po[1]+dp[1]
                # only the position be visited by cnt times will
 append to queue
                if 0<=i<len(grid) and 0<=j<len(grid[0]) and matr
ix[i][j][1]==cnt and grid[i][j]==0:
                    matrix[i][j][0] += step+1
                    matrix[i][j][1] = cnt+1
                    q.append(([i,j], step+1))

grid = [[1,0,2,0,1],[0,0,0,0,0],[0,0,1,0,0]]
so = Solution()
a = so.shortestDistance(grid)
print a
```

DFS/DP

Generate Parentheses

Given n pairs of parentheses, write a function to generate all combinations of well-formed parentheses.

For example, given n = 3, a solution set is:

```
[
  "((()))",
  "(()())",
  "(())()",
  "()(())",
  "()()()"
]
```

Leetcode 22

思路： 可以用backtracking做，也可以用dp来做。

BT：

```
class Solution(object):
    def generateParenthesis(self, n):
        """
        :type n: int
        :rtype: List[str]
        """
        left = right = n
        res = []
        self.bt('', left, right, res)
        return res

    def bt(self, cur, left, right, res):

        if left>right or left<0 or right<0:
            return

        if not left and not right:
            res.append(cur)

        self.bt(cur+'(', left-1, right, res)
        self.bt(cur+')', left, right-1, res)
```

DP:

```
class Solution(object):
    def generateParenthesis(self, n):
        """
        :type n: int
        :rtype: List[str]
        """
        dp = [[] for _ in range(n+1)]
        dp[0] = ['']

        for i in range(1, n+1):
            for j in range(i):
                dp[i] += ['(' + x + ')'+ y for x in dp[j] for y
in dp[i-j-1]]
        return dp[-1]

so = Solution()
so.generateParenthesis(3)
```

Generate play list

Get a songs list and a target number. Generate a play list have target number of songs. The songs can repeat but should not as same as the last song.

Ex: songs = [0,1,2,3,4,5,6,7] target = 10 playlist: [0,1,0,1,2,3,4,5,6,7] is a valid list [0,0,1,2,3,4,5,6,7,0] is not return number of result play list.

中文描述：假如我有8首不同的歌 我想做一个10首歌的歌单。这个歌单要包含所有的歌，所以我肯定要重复二首歌。现在又有一个条件，就是这个歌单里，如果要重复一首歌，那必须在这之前放了1首不同的歌。那么共能生成多少个这种歌单.

思路：dfs可以暴力枚举每一种可能。 简单来说，就是先取2首歌（因为顺序可能不一样，所以有8*7种可能）。然后开始用从第三首开始排列组合。每新加的歌曲可以来自于前面的老歌，也可以是新歌。代码如下。复杂度太高，所以只取了前两首直接做permutation。 写到这里我突然感觉可以用dp来做。欢迎大神指导。

```
class Solution(object):
    def playList(self, songs, target):

        self.res = 0

        for i in range(len(songs)):
            for j in range(len(songs)):
                if j!=i:
                    playlist = [songs[i], songs[j]]
                    remain_song = songs[:]
                    remain_song.remove(songs[i])
                    remain_song.remove(songs[j])
                    #self.dfs(playlist, remain_song, target - le
n(songs), target)

        playlist = songs[:2]
        remain_song = songs[2:]
        self.dfs(playlist, remain_song, target - len(songs), tar
get)

        return self.res
```

```
    def dfs(self, playlist, songs, remain, target):

        # escape condition
        if remain<0:
            return

        if not songs:
            if remain==0:
                self.res += 1
            return

        # dfs
        # pick a old song
        if remain>0:
            for repeat in playlist[:-1]:
                self.dfs(playlist + [repeat], songs, remain-1, t
arget)

        # pick a new song
        for i in range(len(songs)):
            newsong = songs[i]
            self.dfs(playlist + [newsong], songs[:i] + songs[i+1
:], remain, target)


songs = [0,1,2,3,4,5,6,7]
target = 10

so = Solution()
a = so.playList(songs, target)
print(a)
```

140 Word Break II

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

Return all such possible sentences.

For example, given s = "catsanddog", dict = ["cat", "cats", "and", "sand", "dog"].

A solution is ["cats and dog", "cat sand dog"].

思路：最简单的就是dfs。不停地用字典里面的单词尝试去切割string s，然后把剩下的继续丢进去recurse。如果s被切归干净了，就把path给存进res里面。

```
class Solution(object):
    def wordBreak(self, s, wordDict):
        """
        :type s: str
        :type wordDict: Set[str]
        :rtype: List[str]
        """
        res = []
        self.dfs(s, wordDict, '', res)
        return res

    def dfs(self, s, wordDict, cur, res):
        if not s: return [None]
        if s in wordDict:
            res.append((cur + ' ' + s).strip())

        for word in wordDict:
            l = len(word)
            if word == s[:l]:
                self.dfs(s[l:], wordDict, cur+' '+ s[:l], res)

s = "catsanddog"
dict = ["cat", "cats", "and", "sand", "dog"]
so = Solution()
a = so.wordBreak(s, dict)
print a
```

然后我们再开用cache优化一下。

思路：上一种做法代码写出来还是比较简单的，可是如果你画出树形图，你会发现，好多路径被重复计算了。所以我们准备用cache来优化。这里需要注意的是，因为要用cache，所以dfs的输入就要稍微改一改，我们不切割s而是用i和j来表示截取string的哪一部分。如果i,j没有在memo里面的话那么我们就recurse去算。如果在memo里面的话，就直接返回结果。避免了重复计算的情况。

抽象出来，我们可以想象其实我们在填一个2D DP的格子。因为j恒比i大，所以我们只用填写格子的右上半部分。然后右上角那个点就是我们要输出的值（即从dp[0,len(s)]的所有解）

```
import collections
class Solution(object):
    def wordBreak(self, s, wordDict):
        """
        :type s: str
        :type wordDict: Set[str]
        :rtype: List[str]
        """
        self.memo = collections.defaultdict(list)
        self.dfs(s, 0, len(s), wordDict)
        return self.memo[(0,len(s))]

    def dfs(self, s, i, j, wordDict):
        if (i, j) not in self.memo:
            if s[i:j] in wordDict:  # if in the dict add to memo
                self.memo[(i, j)] += [s[i:j]]
            for k in range(i+1,j):
                if s[i:k] in wordDict:  # take the first part
                    tail = self.dfs(s, k, j, wordDict)  # recurs
e rest
                    self.memo[(i,j)] += [s[i:k]+' '+t for t in t
ail]
        return self.memo[(i, j)]

s = "aaaaaaa"
dict = ["aaaa","aa","a"]
so = Solution()
a = so.wordBreak(s, dict)
print a
```

加强版：返回这个string能切割出最多个words的结果。

思路：其实就是一个top to bot的dp，不停的切割，然后只保存切割数最多的结果。memo里面存的是一个tuple，第一个int代表能切割多少个单词，后面是切割过后的结果。这样recurse下去就能得到全局最优。其实也是一个dp填格子的思想。

彩蛋：对于不能切割的情况，我们直接返回一个mission impossible哈哈（我的恶趣味），如果能有其他切割情况的话，会顶掉这个mission impossible因为我置的-1。

```
import collections
class Solution(object):
    def wordBreak(self, s, wordDict):
        """
        :type s: str
        :type wordDict: Set[str]
        :rtype: List[str]
        """
        self.memo = collections.defaultdict(list)
        self.dfs(s, 0, len(s), wordDict)
        return self.memo[(0,len(s))]

    def dfs(self, s, i, j, wordDict):
        if (i, j) not in self.memo:
            res = []
            if s[i:j] in wordDict:  # if in the dict add to memo
                res += [(1, s[i:j])]
            for k in range(i+1,j):
                if s[i:k] in wordDict:  # take the first part
                    tail = self.dfs(s, k, j, wordDict)  # recurs
e rest
                    for t in tail:
                        res += [(1+t[0],s[i:k]+' '+t[1])]
            self.memo[(i,j)] = [max(res)] if res else [(-1,'Miss
ion impossible')]
        return self.memo[(i, j)]

s = "aabaaaabaa"
dict = ["aaaa","aa","b"]
so = Solution()
a = so.wordBreak(s, dict)
print a
```

291 Word Pattern II

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

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

Examples: pattern = "abab", str = "redblueredblue" should return true. pattern = "aaaa", str = "asdasdasdasd" should return true. pattern = "aabb", str = "xyzabcxzyabc" should return false. Notes: You may assume both pattern and str contains only lowercase letters.

https://leetcode.com/problems/word-pattern-ii/

```
class Solution(object):
    def wordPatternMatch(self, pattern, str):
        """
        :type pattern: str
        :type str: str
        :rtype: bool
        """
        return self.bt(pattern, str, {})

    def bt(self, pattern, s, d):
        if not pattern and len(s)>0:
            return False
        if not pattern and not s:
            return True

        for i in range(len(s)-len(pattern)+1):
            if pattern[0] in d and d[pattern[0]]==s[:i+1]:
                if self.bt(pattern[1:], s[i+1:], d):
                    return True
            elif pattern[0] not in d and s[:i+1] not in d.values
():
                d[pattern[0]] = s[:i+1]
                if self.bt(pattern[1:], s[i+1:], d):
                    return True
                del d[pattern[0]]
        return False
```

Stack

Basic Calulator

Implement a basic calculator to evaluate a simple expression string.

The expression string contains only non-negative integers, +, -, *, /, (,) operators and empty spaces . The integer division should truncate toward zero.

You may assume that the given expression is always valid.

思路：用stack, 一个存数，一个存operation。遇到符号，就检查上一个operation是不是*或者/是的话就先算。遇到反括号‘）’,就一直算到遇到‘（’为止。最后loop完s，nums中只剩数字，ops中只剩+ or -。一个一个op地pop出来，然后算好push回去就行了。

```
class Solution(object):
    def calculate(self, s):
        """
        :type s: str
        :rtype: ints
        """
        num, nums, ops = 0, [0], []
        s = s + '#'

        for i in range(len(s)):
            if s[i] in '1234567890':    # if it is a number.
                num = num*10+int(s[i])
            else:                       # if it is not a number
                if i>0 and s[i-1] not in '+-*/()':
                    nums.append(num)
                    num = 0
                if s[i] in '+-*/':      # check if pre is */
                    if ops and ops[-1] in '*/': # if so we apply
op
                        y, x, op = nums.pop(), nums.pop(), ops.p
op()
                        nums.append(self.applyOp(x,y,op))
                    ops.append(s[i])
                elif s[i] in '(':       # heihei
```

```
                    ops.append(s[i])
                elif s[i] == ')':       # loop untill find '('
                    while ops and ops[-1]!='(':
                        y, x, op = nums.pop(), nums.pop(), ops.p
op()
                        nums.append(self.applyOp(x,y,op))
                    ops.pop()

        while ops:  # construct final value
            y, x, op = nums.pop(), nums.pop(), ops.pop()
            nums.append(self.applyOp(x,y,op))
        return nums[-1]

    def applyOp(self, x, y, op):
        """
        opertion helper
        :type x: int
        :type y: int
        :type op: str
        :rtype: int
        """
        if op == '+':
            return x+y
        elif op == '-':
            return x-y
        elif op == '*':
            return x*y
        elif op == '/':
            if x//y<0 and x%y!=0:
                return x//y + 1
            else:
                return x//y

so = Solution()
ans = so.calculate('-3')
print(ans)
ans = so.calculate('1+4*(2-3)+(4/2+1)')
print(ans)
```

Tree

Serialize and Deserialize Binary Tree

297 Serialize and Deserialize Binary Tree

https://leetcode.com/problems/serialize-and-deserialize-binary-tree/

Serialization is the process of converting a data structure or object into a sequence of bits so that it can be stored in a file or memory buffer, or transmitted across a network connection link to be reconstructed later in the same or another computer environment.

Design an algorithm to serialize and deserialize a binary tree. There is no restriction on how your serialization/deserialization algorithm should work. You just need to ensure that a binary tree can be serialized to a string and this string can be deserialized to the original tree structure.

```
   1
   / \
  2   3
     / \
    4   5
```

Convert to [1 2 # # 3 4 # # 5 # #] and then back the tree

思路： 用preorder traversal 然后用#来代表None。 这样在造树的时候。就可以停止在正确的位置。

```
class TreeNode(object):
    def __init__(self, x):
        self.val = x
        self.left = None
        self.right = None

class Codec:
    def serialize(self, root):
        if not root:
            return '# '
```

```
        res = ''
        if root:
            res += str(root.val) + ' '
            res += self.serialize(root.left)
            res += self.serialize(root.right)

        return res

    def deserialize(self, data):
        data = data.split()
        root = self.helper(data)
        return root

    def helper(self, data):

        tmp = data.pop(0)
        if tmp == '#':
            return None

        root = TreeNode(int(tmp))
        root.left = self.helper(data)
        root.right = self.helper(data)

        return root


a = TreeNode(1)
b = TreeNode(2)
c = TreeNode(3)
a.left = b
a.right = c

codec = Codec()
ans = codec.serialize(a)
print(ans)
root = codec.deserialize(ans)
print(root.val)
print(root.left.val)
print(root.right.val)
```

2 Construct Binary Tree Using Pre/In/Post Order traversal

3 Flatten A Binary Tree to Double Linked List

binary tree to doubly linked list.

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

思路：这道题是数flatten成单项链表，面试题目相当于是加强版。就是preorder里面加点料。需要注意的地方是我们这个 self.pre = root这里很精妙，相当于在flatten右边树之前，把左树的最后一个节点给存起来了。

```
class Solution(object):
    pre = None
    def flatten(self, root):
        """
        :type root: TreeNode
        :rtype: void
        """
        if root:
            self.pre = root
            self.flatten(root.left)

            tmp = root.right
            root.right = root.left
            root.left = None
            self.pre.next = root.right

            self.flatten(tmp)
```

加强版：因为我们需要双向链表。所以要记录这个node的parent.因为Input变成了2个，所以我们需要新写一个helper function呢。

```
class TreeNode(object):
    def __init__(self, x):
        self.val = x
        self.left = None
        self.right = None
```

```
class Solution(object):
    pre = None
    def flatten(self, root):
        """
        :type root: TreeNode
        :rtype: void
        """
        self.helper(root, None)


    def helper(self, root, p):

        if root:

            self.pre = root
            self.helper(root.left, root)

            tmp = root.right
            root.right = root.left
            root.left = p
            self.pre.next = root.right

            self.helper(tmp, self.pre)


a = TreeNode(1)
b = TreeNode(2)
c = TreeNode(3)

a.left, a.right = b, c

so = Solution()
so.flatten(a)
print a.val, a.right.val, a.right.left.val
```

Union Find

3 Number of Connected Components in an Undirected Graph

Given n nodes labeled from 0 to n - 1 and a list of undirected edges (each edge is a pair of nodes), write a function to find the number of connected components in an undirected graph.

Example 1:

```
     0          3
     |          |
     1 --- 2    4
```

Given n = 5 and edges = [[0, 1], [1, 2], [3, 4]], return 2.

Example 2:

```
     0           4
     |           |
     1 --- 2 --- 3
```

Given n = 5 and edges = [[0, 1], [1, 2], [2, 3], [3, 4]], return 1.

https://leetcode.com/problems/number-of-connected-components-in-an-undirected-graph/

思路：典型的union find题目。首先把所有节点add进union class。然后遇到edge就union。 加node的时候+1， union成功的话-1。最后看counter为几。

```
class Solution(object):
    def countComponents(self, n, edges):
        """
        :type n: int
        :type edges: List[List[int]]
        :rtype: int
        """
```

```
        u = union()
        for i in range(n):
            u.add(i)
        for i,j in edges:
            u.unite(i,j)
        return u.cnt

class union(object):
    def __init__(self):
        self.dic = {}
        self.sz = {}
        self.cnt = 0

    def add(self, i):
        self.dic[i] = i
        self.sz[i] = 1
        self.cnt += 1

    def find(self, i):
        while self.dic[i] != self.dic[self.dic[i]]:
            self.dic[i] = self.dic[self.dic[i]]
        return self.dic[i]

    def unite(self, i, j):
        i, j = self.find(i), self.find(j)
        if i==j:
            return
        if self.sz[i]<self.sz[j]:
            i, j = j, i
        self.sz[i] += self.sz[j]
        self.dic[j] = i
        self.cnt -= 1
```

2 Number of island I

Number of Islands II

A 2d grid map of m rows and n columns is initially filled with water. We may perform an addLand operation which turns the water at position (row, col) into a land. Given a list of positions to operate, count the number of islands after each addLand operation. An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Example:

Given m = 3, n = 3, positions = [[0,0], [0,1], [1,2], [2,1]]. Initially, the 2d grid grid is filled with water. (Assume 0 represents water and 1 represents land).

We return the result as an array: [1, 1, 2, 3]

https://leetcode.com/problems/number-of-islands-ii/

思路：这道题是union find的经典题型了。看代码就能make sense了。我只能说感谢乐神。乐神我是你的粉丝！我优化了一点点乐神的代码。一样ac了。在find的时候，其实只用compress i的parents就行了。

```
class Solution(object):
    def numIslands2(self, m, n, positions):
        """
        :type m: int
        :type n: int
        :type positions: List[List[int]]
        :rtype: List[int]
        """
        ans = []
        islands = Union()
        for p in map(tuple, positions):
            islands.add(p)
            for dp in (0, 1), (0, -1), (1, 0), (-1, 0):
                q = (p[0] + dp[0], p[1] + dp[1])
                if q in islands.dic:
                    islands.unite(p, q)
```

```
            ans += [islands.cnt]
        return ans

class Union(object):
    def __init__(self):
        self.dic = {}
        self.sz = {}
        self.cnt = 0

    def add(self, i):
        self.dic[i] = i
        self.sz[i] = 1
        self.cnt += 1

    def find(self, i):
        while self.dic[i]!= self.dic[self.dic[i]]:
            self.dic[i] = self.dic[self.dic[i]]
        return self.dic[i]

    def unite(self, p, q):
        i, j = self.find(p), self.find(q)
        if i==j:
            return
        if self.sz[i] > self.sz[j]:
            i, j = j, i
        self.dic[i] = j
        self.sz[j] += self.sz[i]
        self.cnt -= 1


so = Solution()
p = [[0,1],[1,2],[2,1],[1,0],[0,2],[0,0],[1,1]]
a = so.numIslands2(3,3,p)
print(a)
```

Data Structure

Insert, Delete and Get Most Frequent in O(1)

Design a data structure that can make insert, delete and get most frequent in O(1).

思路： 用hash+double linked List. 整体很简单，看代码应该就可以懂了。hash存每个node在链表中的位置。add的时候如果不在hash中，那么加到链表末尾，如果在hash中，update counter并与左边的node比较。如果比他大就swap。Delete的话，直接从hash找到链表位置，删除掉就好了。GetFrequent直接返回双向链表的head。

```
class Node(object):
    def __init__(self, val):
        self.val = val
        self.count = 1
        self.left = None
        self.right = None

class Solution(object):
    def __init__(self):
        self.d = {}
        self.head = None
        self.tail = None

    def insert(self, key):
        if key in self.d:   # if in dict plue on
            node = self.d[key]
            node.count += 1
            # if it is not head count larger than left node, swap

            if node!=self.head and node.count > node.left.count:
                self.swap(node.left, node)
        else:
            node = Node(key)
            self.d[key] = node
            if not self.head:
```

```
                self.head = node
                self.tail = node
            else:
                self.tail.right = node
                node.left = self.tail
                self.tail = node
        return True

    def delete(self, key):
        if key not in self.d:
            return False
        else:
            node = self.d[key]
            del self.d[ky]
            if len(self.d)==0:
                self.head == self.tail == None  # only node.
            elif not node.left:
                self.head = self.head.right   # node is head
            elif not node.right:
                self.tail = self.tail.left  # node is tail
            else:
                node.left.right = node.right
        return True

    def getMostFrequent(self):
        if self.head:
            return self.head.count

    def swap(self, l, r):
        if l == self.head:
            self.head = r
        if r == self.tail:
            self.tail = l
        l.right, l.left, r.left, r.right = r.right, r, l.left, l

so = Solution()
so.insert(1)
so.insert(2)
so.insert(2)
so.insert(3)
```

```
so.insert(2)
tmp = so.head
while tmp:
    print(tmp.val)
    tmp = tmp.right
a = so.getMostFrequent()
print(a, "times")
```

2 Min Queue enqueue, dequeue, getMin in O(1)

Implement a queue which supports enqueue, dequeue, getMin in O(1)

```
class minQueue(object):
    def __init__(self):
        self.q = []
        self.m = []

    def enqueue(self, num):
        """
        :type num: int
        :rtype: None
        """
        self.q.append(num)
        if not self.m:
            self.m.append(num)
        else:
            while self.m and self.m[-1]>num:
                self.m.pop()
            self.m.append(num)

    def dequeue(self):
        """
        :rtype: int
        """
        if self.q:
            tmp = self.q.pop(0)
            if tmp==self.m[0]:
                self.m.pop(0)
            return tmp

    def findMin(self):
        """
        :rtype: int
        """
```

```
        if self.m:
            return self.m[0]


q = minQueue()
q.enqueue(2)
q.enqueue(5)
ans = q.findMin()
print ans
ans = q.dequeue()
print ans
ans = q.findMin()
print ans
```

Graph

Topological sort

Topological sort 有很多实现方式比如维基上就列举了2种。这里我们分别实现一下吧。 https://en.wikipedia.org/wiki/Topological_sorting

1 Kahn's algorithm （push node到stack前，先检查是否已visit了所有parent）

这里我们先实现第一种Kahn's algorithm。我也不知道怎么发音，就叫他“看”算法好了。主要思想是maintain两个dict。一个是indegree一个是outdegree。只有一个node所有的indegree都被visit过了，我们才把这个node加到stack里面去。这样就能保证visit一个node的时候，在他之前的node全部都visit过了（这不就是toposort要做的事情么）。如果最后不能遍历所有的node的话，那么这个图就是有circle的，这里有点绕。因为带circle的图，node之间互为indegree，永远不可能visit这个node全部的indegree。

实现方法是，开始先找所有没有indegree的node，来init一个stack，然后pop一个node出来，然后对他的所有child的indegree移除这个parent。如果移除这个node后的child已经没有indegree了，那么就把这个child push到stack里面。

```
import collections
class Solution(object):
    def toposort(self, graph):
        """
        :type graph: dict[int:List[int]]
        :rtype: List[int]
        """
        indegree = collections.defaultdict(set)
        outdegree = graph

        # init the stack
        for key in outdegree:
            for c in outdegree[key]:
                indegree[c].add(key)
        stack = [k for k in outdegree if k not in indegree]

        # dfs
        res = []
        while stack:
```

```
            tmp = stack.pop()
            res.append(tmp)
            if tmp in outdegree:
                children = outdegree[tmp]
                for c in children:       # remove tmp for all th
e child
                    indegree[c].remove(tmp)
                    if not indegree[c]:  # if any child's all pa
rents are visited
                        stack.append(c)  # add this child to sta
ck

        node = set([k for k in outdegree] + [k for k in indegree
])

        return res if len(node)==len(res) else []

graph = {1:[2,3,4,5],
        2:[3,4],
        3:[5]}
so = Solution()
a = so.toposort(graph)
print a
```

2 DFS （找一条path一路走到黑，边走边检查是否circle，走到底就把最后node存起来。）

第二种实现方式是DFS。我偷的我女朋友的代码。主要思想是DFS。开始也是一样的，抬手先找所有没有indegree的node，来init一个stack。然后从stack最后的node开始，找一条path一直走到底，中间不断检查新的node时候在stack里面（即有没有circle）。走到底后，我们就pop出最后这个node，把他存到result里面。

```
class Solution(object):
    def toposort(self, graph):
        """
        :type graph: dict[int:List[int]]
        :rtype: List[int]
        """
        children = []
        for k in graph:
            children += graph[k]
        # find starting points
        stack  = [k for k in graph if k not in set(children)]

        res=[]
        while stack:
            while stack[-1] in graph and graph[stack[-1]]: # kee
p dig further
                tmp=graph[stack[-1]].pop()
                if tmp not in stack:   # check if node in path
                    stack.append(tmp)
                else:                  # if it in the coming pat
h, we find a circle
                    return []          # return []
            tmp=stack.pop()        # finish one path, pop last n
ode
            if tmp not in res:
                res.append(tmp)
        return res

graph = {1:[2,3,4,5],
        2:[3,4],
        3:[5]}
so = Solution()
a = so.toposort(graph)
print a
```

比较：第一种实现方式，我们一开始要建立一个indegree的dict。所以比较耗费空间，后面因为都是hash操作，所以时间复杂度很不错呀。第二种方式，因为不停要检查新的node在不在stack里面，这个操作是O（k）,k是stack的长度。所以我只能

理解为，这个时间复杂度比较高。所以两种算法的复杂度tradeoff在时间和空间上。（不一定对哈）

温故而知新，不如我们用topological sort来做道题把。

269 Alien Dictionary https://leetcode.com/problems/alien-dictionary/

```
import collections
class Solution(object):
    def alienOrder(self, words):
        """
        :type words: List[str]
        :rtype: str
        """
        greater = collections.defaultdict(set)
        less = collections.defaultdict(set)

        for i in range(len(words)-1):
            pre = words[i]
            cur = words[i+1]
            index=0
            while index<min(len(pre), len(cur)) and cur[index]==
pre[index]:
                index += 1
            if index<min(len(pre), len(cur)):
                greater[pre[index]].add(cur[index])
                less[cur[index]].add(pre[index])

        chars = set(''.join(words))
        s = [ i for i in chars if i not in less]

        res = []
        while s:
            tmp = s.pop()
            res.append(tmp)
            children = greater[tmp]
            for c in children:
                less[c].remove(tmp)
                if not less[c]:
                    s.append(c)
```

```
        if len(chars) == len(res):
            return ''.join(res)
        return []


words = [ "wrt", "wrf", "er", "ett", "rftt"]
so = Solution()
a = so.alienOrder(words)
print a
```

Dijkstra's Algoritm

很多问题都可以抽象为shortest path in a weighted graph。所以掌握一种shortest path的算法还是很重要的。比如这道面经题目(找P5那道题)我是面经。 这里我们提供一种比较简单方便实现的Dijstra algorithm。不是最优的，还有很多优化空间。

Dijkstra's algorithm的思想是。从start point开始，然后把周围能reach到的点存到Queue里面。然后每次取cost最小的那个point来继续traverse这个graph。遇到的每一个点，先检查是否在Q中，不在的话就加进去，在的话就与Q中的点进行比较，如果cost更小就更新Q里面的cost。然后再取一个最短的出来继续算。这样是可以保证最终start到每一个点的cost最小。

好吧我觉得我没有讲清楚，anyway。我们还是看代码吧，我的Q可以用hashed priority queue进行优化。这里我先不管了哈。

```
class Solution(object):
    def dijkstra(self, graph, start, end):
        D = {}  # distance of the node
        P = {}  # parent of the node
        Q = {}  # Q for get the min
        Q[start] = 0

        while Q:
            v = self.get_min(Q) # get the min vertice
            D[v] = Q[v]
            children = graph[v]
            for c in children:  # relax each child
                dist = Q[v]+children[c]
                if c in D:
                    if dist<D[c]:
                        raise ValueError
                elif c not in Q or dist<Q[c]:
                    Q[c] = dist
                    P[c] = v
            del Q[v]

        path = []
```

```
        res = D[end]
        while end in P:
            path.append(end)
            end = P[end]
        path.append(end)

        return path[::-1], res

    def get_min(self, Q):
        m = (None, float('inf'))
        for k in Q:
            if Q[k]<m[1]:
                m = (k, Q[k])
        return m[0]

# example, CLR p.528
G = {'s': {'u':10, 'x':5},
    'u': {'v':1, 'x':2},
    'v': {'y':4},
    'x':{'u':3,'v':9,'y':2},
    'y':{'s':7,'v':6}}

so = Solution()
print so.dijkstra(G, 's', 'v')
```

另外一种实现方法.

```
import sys

def min_dist(dist, visited, n):
    min_val = sys.maxint
    min_idx = 0
    for i in xrange(n):
        if not visited[i] and dist[i] <= min_val:
            min_val, min_idx = dist[i], i

    return min_idx

def dijkstra(graph, src, des):
    n = len(graph)
    dist = [sys.maxint] * n
    dist[src] = 0
    visited = [False] * n

    for _ in xrange(n):
        i = min_dist(dist, visited, n)
        visited[i] = True
        if i == des:
           return dist[i]
        for j in xrange(n):
            if (not visited[j] and graph[i][j] and dist[i] != sy
s.maxint and
                    dist[i] + graph[i][j] < dist[j]):
                dist[j] = dist[i] + graph[i][j]



graph = [[0, 3, 2],[5, 0, 2],[1, 6, 0]]
src, des = 2, 1
print("length of the shortest path from #%d to #%d: %d" \
      % (src, des, dijkstra(graph, src, des)))
```

QuadTree

1 QuadTree Structure

Read a 2D array image, buid a quad tree. 输入的矩阵如下：

```
[[1, 1, 0, 0],
 [1, 1, 0, 0],
 [1, 0, 1, 1],
 [1, 1, 1, 1]]
```

然后build出来的quadtree的root的child, 左上角应该是1， 右上是0，左下是2，右下是1

```
class Node(object):
    def __init__(self, val=None):
        self.val = val
        #[NW, NE, SE, SW]
        self.chidren = [None]*4

class Solution(object):
    def QuadTree(self, image):
        if not image or not image[0]:
            return
        i, j = (0,0), (len(image)-1,len(image[0])-1)
        root = self.build(i, j, image)
        return root

    def build(self, i, j, image):
        if i==j:
            return Node(image[i[0]][i[1]])

        root = Node()
        rmid = (i[0]+j[0])//2
        cmid = (i[1]+j[1])//2
        NW = self.build(i, (rmid, cmid), image)
        NE = self.build((i[0],cmid+1), (rmid, j[1]), image)
        SE = self.build((rmid+1, cmid+1), j, image)
        SW = self.build((rmid+1, i[1]),(j[0],cmid), image)
```

```
        #print i, (rmid, cmid), NW.val
        if (NW.val,NE.val,SE.val,SW.val) == (0,0,0,0):
            root.val = 0
        elif (NW.val,NE.val,SE.val,SW.val) == (1,1,1,1):
            root.val = 1
        else:
            root.val = 2
            root.chidren = [NW,NE,SE,SW]

        return root

image = [
[1, 1, 0, 0],
[1, 1, 0, 0],
[1, 0, 1, 1],
[1, 1, 1, 1]]

so = Solution()
root = so.QuadTree(image)
print root.val
print [c.val for c in root.chidren]
```

Binary Search

Median of Two Sorted Arrays

https://leetcode.com/problems/median-of-two-sorted-arrays/

There are two sorted arrays nums1 and nums2 of size m and n respectively.

Find the median of the two sorted arrays. The overall run time complexity should be O(log (m+n)).

Example 1:

```
nums1 = [1, 3]
nums2 = [2]

The median is 2.0
```

Example 2:

```
nums1 = [1, 2]
nums2 = [3, 4]

The median is (2 + 3)/2 = 2.5
```

思路：这道题可以用一个通用解法，就是find kth element in 2 sorted array。主要思想是每次至少砍掉一个数组的一半的数。怎么实现每次把一个数组砍半呢。

我们每次取a的中间的index叫做amid 和b中间的index叫做bmid，然后a和b就根据amid和bmid被切成了4个如下小块，a的前半部分和a的后半部分，b的前半部分和b的后半部分。

```
a[:amid], a[amid:]
b[:bmid], b[bmid:]
```

然后我们要每次要丢掉一个小块。这样每一个循环我们就都在缩小范围。然后我们要保证每次丢掉一小块的时候kth element都在剩下的块里面。

所以我们先用amid+bmid和k比较，如果amid+bmid比k大说明，k在左边三块中，我们可以把最大的那一小块给删掉。所以通过比较a[amid]和b[bmid]就可以找出较大的那一小块。如果amid+bmid比k小说明，k在右边三块中，我们可以把最小的那一小块给找出来删掉。所以通过比较a[amid]和b[bmid]中小的那个的mid value的左边就是最小的那一块。

这里要注意的是，等于情况怎么处理。因为我们的index是向下取整的。所以如果碰到amid+bmid=k的话，应该算mide在左边的三块里面，所以删出右边的那一块。

这样就保证了，每次都在减少，每次解都在剩下的数里面。感觉还是没讲清楚，我们还是来看代码吧。

```
class Solution(object):
    def findMedianSortedArrays(self, nums1, nums2):
        """
        :type nums1: List[int]
        :type nums2: List[int]
        :rtype: float
        """
        l = len(nums1) + len(nums2)
        if l%2==1:
            return self.kth(nums1, nums2, l//2)
        else:
            return (self.kth(nums1, nums2, l//2-1) + self.kth(nu
ms1,nums2, l//2))//2


    def kth(self, a, b, k):
        if not a:
            return b[k]
        if not b:
            return a[k]

        # mid index of a and b
        ma, mb = len(a)//2, len(b)//2

        if ma+mb>=k: # mid is on the left
            if a[ma]>b[mb]:
                return self.kth(a[:ma], b, k)
            else:
                return self.kth(a, b[:mb], k)
        else:   # mid is on the right
            if a[ma]>b[mb]:
                return self.kth(a, b[mb+1:], k-mb-1)
            else:
                return self.kth(a[ma+1:], b, k-ma-1)
```