leetcode分类总结

Xiang Li

Published with GitBook

Table of Contents

算法概念

编程面试的10大算法概念

面试资源，经验帖子

- 一些面试资源
- 分析算法题目过程

分类

- leetcode分类
- 原leetcode难度及频率
- leetcode题解收集
- 算法之美

Reference

- 烟客旅人
    - simulation, math, hash, bit operation
    - linked list and array
    - search, sum and tree
    - backtracking, greedy and DP
    - k sum
- code_ganker
    - 树的遍历
    - 一维DP
    - Binary Search
    - 各种题解
- leetcode summary
    - graph
    - math
    - sorting
    - tree bfs

  - tree dfs
  - tree recursion
  - stack
- 一丝微凉
- code career
  - Data Structure
  - Linked List
  - Binary Tree
  - String
  - Array and Matrix
  - Stack and Queue
  - DP
  - Search and Sort
  - Backtracking
  - Bit manipulation
  - Company related
- grandyang
- felix的系列总结
  - binary tree
  - 全排列
  - bucket sort ....

求和问题总结

题目列表

- 2 sum
- **2 sum II - input array is sorted**
- **2 sum III - data structure design**
- 3 sum
- 3 sum closest
- **3 sum smaller**
- 4 sum
- k sum
- k sum II

问题描述

一般是给一组n个数字，给1个target, 求出k个数字的sum为target. 有变化的题目就是求closest，求个数，求组合等等

注意事项

- 可能有重复项，注意去掉重复的组合, 除了2sum的题目最好先sort一下，这样可以方便去掉重复的项
- sort方法枚举的时候注意不要重复，就像subsets一样

2 sum 解法

方法1 - brute force

枚举所有的k-subset, time complexity就是从N中选出k个，`O(n^k)`

方法2 - 先sort,再two pointer

O(NlogN + N) = O(nlogn), 要注意的是sort了之后就改变了原来的顺序

方法3 - hashmap O(n)

对于2 sum来说，其实就是对于每一个element `nums[i]` , 在数组中找是否存在 `target - nums[i]`

用hashmap保存访问过的value, 对每个 `num[i]` ,检查hashmap中是否有 `target - nums[i]` ,扫完一遍就能够得到结果。属于用空间换时间。

time complexity - `O(n)` space complexity - `O(n)`

后续题目

对于two sum的题目

- 如果是要返回index,那么优先用hashmap的做，因为不会改变原来array的顺序
- 如果是返回元素的value,那么可以用先sort然后two pointer的方法

对于3sum, 3sum closest, 4sum等题目，因为大部分都是根据two sum two pointer做法的延伸，所以都是要求return value

summary

two pointer

two pointer做法有利于跳过重复元素，用来计算closest, smaller等等不等于target的题目，所以优先使用

3 sum 解法

3 sum 可以退化为2 sum， 先取出一个数i, 然后在剩下的数组中找sum为 `target - i` 的就可以了。

这里要注意的是不管采取sort还是hashmap的方法，时间复杂度其实都是 `O(n^2)` . hashmap的大家都知道，排序的解释如下：

排序

sort只sort一遍 O（nlogn), 然后每一个取出一个数，再two pointer寻找的复杂度是O(n^2)

总的复杂度是 O(nlogn + n^2) = O(n^2)

3 sum closest解法

3 sum closest的解法为采取类似3 sum, 但是不要用hashmap, 用 `sort + two pointer` 的方法可以方便的找到closest

4 Sum

4 sum退化为3 sum， 可以用两个for loop内部再2 sum的方法来做 `O(n^3)`

~~###hashmap pair(似乎不对，再研究)~~

~~还有一种hashmap存pair的方法~~

~~- 首先两个for loop,将所有(i, j)的sum作为key, (i, j)作为value存在hashmap里。 ~~ ~~- 然后4 sum问题就变成了在这个hashmap里找2 sum的问题~~

~~>这种方法要注意重复的index~~

K sum

K sum也一步步退化，最终变为2 sum

K sum的时间复杂度是 `O(n^(k-1))`

Reference

- hackersum007
- leon_cai
- sigmainfy 烟客旅人
    - 非常详细，有各个题目的各种情况分析

Two sum(input array is sorted)

题目描述

array已经sorted

解题方法

已经sorted就不用担心改变顺序，直接使用two pointer方法做

Solution

Two pointer

```
class Solution(object):
    def twoSum(self, numbers, target):
        """
        :type numbers: List[int]
        :type target: int
        :rtype: List[int]
        """
        length = len(numbers)
        if length < 2:
            return []
        #sort to allow two-pointer algorithm and skip duplicate
        numbers.sort()
        start, end = 0, length - 1
        while start < end:
            #skip duplicate elements
            if start != 0 and numbers[start] == numbers[start-1]:
                start += 1
            elif end != length - 1 and numbers[end] == numbers[end
                end -= 1
            else:
                curSum = numbers[start] + numbers[end]
                if curSum == target:
                    return [start + 1, end + 1]
                elif curSum < target:
                    start += 1
                else:
                    end -= 1
        return []
```

can do a binary search

```
class Solution(object):
    def twoSum(self, numbers, target):
        """
        :type numbers: List[int]
        :type target: int
        :rtype: List[int]
        """
        length = len(numbers)
        if length < 2:
            return [0, 0]

        index1 = 0
        index2 = 0
        for i, num in enumerate(numbers):
            foundPair, index = self.search(i + 1, length - 1, numbe
            if foundPair:
                index1 = i + 1
                index2 = index + 1
                return [index1, index2]

        return [index1, index2]

    def search(self, start, end, numbers, t):
        while start + 1 < end:
            mid = start + (end - start) / 2
            if numbers[mid] == t:
                return True, mid
            elif numbers[mid] < t:
                start = mid
            else:
                end = mid
        if numbers[start] == t:
            return True, start
        if numbers[end] == t:
            return True, end

        return False, 0
```

Two Sum Data structure

Question

Design and implement a TwoSum class. It should support the following operations: add and find.

add - Add the number to an internal data structure. find - Find if there exists any pair of numbers which sum is equal to the value.

Thoughts

The original two sum problem is solved by hashmap, actually here we can directly use a hashmap to store the numbers

Solution

```
class TwoSum(object):

    def __init__(self):
        """
        initialize your data structure here
        """
        self.nums = {}

    def add(self, number):
        """
        Add the number to an internal data structure.
        :rtype: nothing
        """
        if number in self.nums:
            self.nums[number] += 1
        else:
            self.nums[number] = 1

    def find(self, value):
        """
        Find if there exists any pair of numbers which sum is equal
        :type value: int
        :rtype: bool
        """

        for i in self.nums:
            b = value - i
            if b in self.nums and (b != i or self.nums[i] > 1):
                return True
        return False


# Your TwoSum object will be instantiated and called as such:
# twoSum = TwoSum()
# twoSum.add(number)
# twoSum.find(value)
```

3sum

题目描述

Given an array S of n integers, are there elements a, b, c in S such that a + b + c = 0? Find all unique triplets in the array which gives the sum of zero.

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

```
    For example, given array S = {-1 0 1 2 -1 -4},

    A solution set is:
    (-1, 0, 1)
    (-1, -1, 2)
```

解题方法

这里的思想主要是，取出一个元素i，然后再用two sum的方法求其他两个，它们的sum为target - i

遇到题目没有思路的时候都可以考虑先 `sort` 一下~

注意点

- 不能有重复的set, 所以都要将数组**sort**一下，方便在code中跳过重复的数
- two sum可以用hashmap, 也可以用two pointer的方法

时间复杂度O(nlogn + n * n) = O(n^2)

Solution

hashmap的方法

```
class Solution(object):
    def threeSum(self, nums):
        """
        :type nums: List[int]
        :rtype: List[List[int]]
        """
        numbers = nums
        if len(numbers) < 3:
            return []
        numbers.sort()
        results = []
        for idx, num in enumerate(numbers):
            if idx != 0 and num == numbers[idx-1]:
                #跳过重复元素
                continue
            target = 0 - num
            #only start with afterward elements
            twoSumResults = self.twoSum(numbers[idx+1:], target)
            if twoSumResults:
                for comb in twoSumResults:
                    #index should start from idx + 1
                    result = [numbers[idx], numbers[comb[0] + idx -
                    result.sort()
                    results.append(result)
        return results

    def twoSum(self, numbers, target):
        # need to find all combinations now
        results = []
        found = False
        for idx, num in enumerate(numbers):
            dic[num] = idx
        for idx, num in enumerate(numbers):
            #跳过重复元素
            if idx != 0 and num == numbers[idx-1]:
                continue
            t = target - num
            if t in dic and dic[t] > idx:
                #该元素在idx的后面
                found = True
                results.append([idx, dic[t]])
        if not found:
            return None
        return results
```

two pointer的方法

```
class Solution(object):
    def threeSum(self, nums):
        """
        :type nums: List[int]
        :rtype: List[List[int]]
        """
        numbers = nums
        if len(numbers) < 3:
            return []
        numbers.sort()
        results = []
        for idx, num in enumerate(numbers):
            if idx != 0 and num == numbers[idx-1]:
                continue
            target = 0 - num
            #only start with afterward elements
            twoSumResults = self.twoSum(numbers[idx+1:], target)
            if twoSumResults:
                for comb in twoSumResults:
                    #index should start from idx + 1
                    result = [numbers[idx], numbers[comb[0] + idx -
                    result.sort()
                    results.append(result)
        return results

    def twoSum(self, numbers, target):
        # need to find all combinations now
        dic = {}
        index1 = None
        index2 = None
        results = []
        found = False
        start, end = 0, len(numbers) - 1
        while start < end:
            if start != 0 and numbers[start] == numbers[start-1]:
                start += 1
            elif end != len(numbers) - 1 and numbers[end] == number
                end -= 1
            else:
                curSum = numbers[start] + numbers[end]
                if curSum == target:
                    found = True
                    results.append([start, end])
                    start += 1
                    end -= 1
                elif curSum < target:
                    start += 1
                else:
                    end -= 1


        if not found:
            return None
        return results
```

3 Sum Closest

题目描述

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

```
    For example, given array S = {-1 2 1 -4}, and target = 1.

    The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).
```

解题方法

这道题和3 sum 类似，但是这里只需要找到一个组合并且return他们的sum，并且不用考虑重复的问题

所以每道题的需求和情况要分析好，并且这道题有pass一个target的变量，直接用3sum的code就会改变了target得到错误的解……而不是类似的题目就把code改一改……

Solution

```
class Solution(object):
    def threeSumClosest(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: int
        """
        numbers = nums
        if len(numbers) < 3:
            return []
        numbers.sort()
        result = sys.maxint
        minDiff = sys.maxint
        for idx, num in enumerate(numbers):
            t = target - num
            #only start with afterward elements
            curDiff, twoSumResults = self.twoSumClosest(numbers[idx
            if minDiff > curDiff:
               minDiff = curDiff
               result = num + numbers[idx+1+twoSumResults[0]] + nur
        return result

    def twoSumClosest(self, numbers, target):
        # need to find all combinations now
        index1 = None
        index2 = None
        #record min Difference
        minDiff = sys.maxint
        start, end = 0, len(numbers) - 1
        while start < end:
                curSum = numbers[start] + numbers[end]
                curDiff = abs(curSum-target)
                if curDiff < minDiff:
                    minDiff = curDiff
                    index1 = start
                    index2 = end
                if curSum < target:
                    start += 1
                elif curSum > target:
                    end -= 1
                else:
                    break

        return minDiff, [index1, index2]
```

3 Sum Smaller

题目描述

Given an array of n integers nums and a target, find the number of index triplets i, j, k with 0 <= i < j < k < n that satisfy the condition nums[i] + nums[j] + nums[k] < target.

```
For example, given nums = [-2, 0, 1, 3], and target = 2.

Return 2. Because there are two triplets which sums are less than 2

[-2, 0, 1]
[-2, 0, 3]
```

解题方法

sort, sort, sort,重要的事情说三遍

这一题要找三个和小于target的数

3sum可以用hashmap或者two-pointer方法 3 sum closest用two-pointer方法

这一题也是用two-pointer的方法更好，对于num i，找另外两个数的和 < target - i 如果不用two pointer的话就要O（n * n^2) = O(n^3) 如果sort了之后用two-pointer就可以到 `O(n^2)`

Solution

```
class Solution(object):
    def threeSumSmaller(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: int
        """
        numbers = nums
        if len(numbers) < 3:
            return 0
        numbers.sort()
        result = 0
        for idx, num in enumerate(numbers):
            t = target - num
            #only start with afterward elements
            twoSumResults = self.twoSum(numbers[idx+1:], t)
            result += twoSumResults
        return result

    def twoSum(self, numbers, target):
        # need to find all combinations now
        if len(numbers) < 2:
            return 0
        result = 0
        start, end = 0, len(numbers) -1
        while start < end:
            while end > start and numbers[start] + numbers[end] >=
                end -= 1
            if start < end:
                result += end - start
                start += 1
            else:
                break

        return result
```

4 sum

题目描述

4 elements sum to target

解题方法

两个for loop,里面2 sum的two pointer方法

将two pointer写在for-loop里，可以减少function call导致的时间

Solution

```
class Solution(object):
    def fourSum(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: List[List[int]]
        """
        nums.sort()
        length = len(nums)
        results = []

        for i in range(len(nums)-3):
            if i != 0 and nums[i] == nums[i-1]:
                continue
            for j in range(i+1, len(nums)-2):
                if j!= i+1 and nums[j] == nums[j-1]:
                    continue

                left = j + 1
                right = length - 1
                while left < right:
                    curSum = nums[i] + nums[j] + nums[left] + nums[
                    if curSum > target:
                        right -= 1
                    elif curSum < target:
                        left += 1
                    else:
                        results.append([nums[i], nums[j], nums[left
                        left += 1
                        right -= 1
                        while(left < right and nums[left] == nums[
                            left += 1
                        while(left < right and nums[right] == nums[
                            right -= 1

        return results
```

Math

题型总结

第一种类型是最简单的， 就是对整数进行直接操作， 一般来说就是逐位操作， 比如反转， 比较等。

第二种题型是算术运算的题目， 比如乘除法， 阶乘， 开方等， LeetCode中这类题目有Sqrt(x)， Pow(x, n)和Divide Two Integers。 这种题目有时候看似复杂， 其实还是有几个比较通用的解法的， 下面主要介绍三种方法：

1. 二分法
2. 牛顿法
3. 位移法

第三种题目是解析几何的题目， 一般来说解析几何题目的模型都比较复杂， 而且实现细节比较多， 在面试中并不常见， LeetCode中也只有Max Points on a Line是属于这种题型。

题目列表

二分法

- sqrt(x)
- pow(x, n)

basic concept

- palindrome number
- reverse integer

- add binary

- add digits
- plus one

- multiply strings

- trailing zeros
- max points on a line
- ugly number
- ugly number II
- happy number
- Gray code (递归的规律)
- valid number(hard,状态转化图重要)

string 2 number, number 2 string

- atoi
- roman to integer
- integer to roman

其他

- gray code
- permutation sequence

Reference

- Code Ganker
- 数学相关的题目

Math without Operator

类型描述

这类题目一般是指进行数学运算，但不要使用operator， 比如除法不要使用/, , %; 加法不要使用+,/,,-

题目列表

解题方法

这种类型的题目，一般都往 `bit manipulation` 方向想

- 对于乘除的题目， `<< 1` 是乘以2， `>> 1` 是除以2。
- 对于加减的题目， `a ^ b` 是sum， `a & b` 是得到carry

注意点

- 正负数的问题
- 是否会overflow的问题（虽然对于Python不会，但是可以跟面试官提一下）

signed 32int integer的range是 `-2147483648 - 2147483647`

题目列表

- Divide Two Integers
- A + B Problem
- multiply two integers

Divide Two Integers

题目描述

Divide two integers without using `multiplication` , `division` and `mod` operator.

If it is overflow, return `MAX_INT` .

解决方法

方法一

不断地减去被除数，直到小于被除数，可以得到结果，但是会超时

方法二，类似二分法

- 左移是乘以2， 我们可以想到不断地将divisor乘以2的结果保存为变量a，直到a大于dividend
- 将dividend减去 `a >> 1` 还小于dividend的值，并且记录下这里将divisor乘以了多少倍， `dividend = divident - a`
- 再继续loop, 直到 `divident < divisor` 结束

Solution

```
class Solution(object):
    def divide(self, dividend, divisor):
        """
        :type dividend: int
        :type divisor: int
        :rtype: int
        """
        sign = 1
        if (dividend < 0) ^ (divisor < 0):
            sign = -1

        if dividend == 0:
            return 0
        if divisor == 0:
            return

        dividend = abs(dividend)
        divisor = abs(divisor)

        result = 0

        while dividend >= divisor:
            i = 0
            a = divisor
            while a <= dividend:
                a = a << 1
                i += 1
            result += (1 << (i-1))
            dividend -= (a >> 1)

        result = result * sign
        if result > 2147483647:
            return 2147483647
        else:
            return result
```

注意点

- 决定正负的时候，dividend和divisor有且只有一个为负才为负数，可以用 `^`
- overflow的range
- 对于中间记录i的时候，因为我是从 `i=0` 看来是，并且i表示乘以了几个2， 所以 `result += (1 << (i-1))`

Reference

- 博客园

Multiply Two Integers

题目描述

Multiply two integers without using multiplication, division

解题方法

不能使用bit wise operator, no loops

这种情况下不能使用任何loop, bit operation, 那只能采用recursion的方法。

```
int multiply(int x, int y)
{
   /* 0  multiplied with anything gives 0 */
   if(y == 0)
     return 0;

   /* Add x one by one */
   if(y > 0 )
     return (x + multiply(x, y-1));

  /* the case where y is negative */
   if(y < 0 )
     return -multiply(x, -y);
}
```

可以使用bit operation, loop

这里就可以采取和divide类似的方法，相当于用n除以1的二分法，在过程中也计算结果

```
class Solution(object):

    def multiply(self, m, n):
        if m == 0 or n == 0:
            return 0

        sign = 1
        if (m < 0) ^ (n < 0):
            sign = -1

        divisor = 2
        result = 0
        while n >= divisor:
            i = 1
            a = 2
            while a <= n:
                a = a << 1
                i += 1

            result += m * ( 1 << (i-1) )
            n -= (1 << (i-1))

        if n:
            result += m

        return result * sign
```

A Plus B

题目描述

Write a function that add two numbers `A` and `B`. You should not use `+` or any arithmetic operators.

解题方法

if you know binary arithmetic or how to add numbers in binary format

- you may be familiar with fact than sum of two numbers can be obtained by using XOR operation
- carry, by using AND operation.

Iterative solution

```
public static int addTwoIntegers(int a, int b){
    while (b != 0){
        int carry = (a & b); // Carry is AND of two bits
        a = a ^ b // sum of tw obits is A XOR B
        b = carry << 1 //shifts carry to 1 bit to calculate sum
    }
}
```

Recursive Solution

```
public static int addTwoIntegers(int a, int b){
    if (b == 0){
        return a;
    }
    int sum = a ^ b;
    int carry = (a & b) << 1;
    return addTwoIntegers(sum, carry);
}
```

bit manipulation的解释

1. a & b能够得到现在的carry

因为只有在同一位都是1的时候才会产生carry, 所以a & b能够得到每一位是否会产生carry.

下面的 `<< 1` 就是将carry左移一位放到下一次的计算中去

1. a ^ b 能够得到除了carry的sum
2. 0 + 1 = 1
3. 1 + 1 = 0 + carry
4. 1 + 0 = 1
5. 0 + 0 = 0

由上面的公式可以看出XOR可以得到当前位的sum的结果

这个方法对于python不行，因为python不会overflow而省去,不过可以加些code使它work

Reference

- how to add two integers

A Minus B

题目描述

Solution

将它变成a+(-b)来做

Ugly Number

题目描述

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

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

Note that 1 is typically treated as an ugly number.

解题方法

- 如果number能被2,3或者5整除，那么就除以可以整除的
- 如果不能被2，3，5中的一个整除，说明不是ugly number
- ugly number应该可以一直被除到1，此时可以结束说明是ugly number

Solution

```python
class Solution(object):
    def isUgly(self, num):
        """
        :type num: int
        :rtype: bool
        """
        if num < 1:
            return False
        while num != 1:
            if num % 2 == 0:
                num /= 2
            elif num % 3 == 0:
                num /= 3
            elif num % 5 == 0:
                num /= 5
            else:
                return False
        return True
```

Reference

Ugly Number II

题目描述

Write a program to find the n-th ugly number.

Ugly numbers are positive numbers whose prime factors only include `2, 3, 5`. For example, `1, 2, 3, 4, 5, 6, 8, 9, 10, 12` is the sequence of the first `10` ugly numbers.

Note that `1` is typically treated as an ugly number.

解题方法

丑陋数序列可以拆分为下面3个子列表：

```
(1) 1×2, 2×2, 3×2, 4×2, 5×2, …
(2) 1×3, 2×3, 3×3, 4×3, 5×3, …
(3) 1×5, 2×5, 3×5, 4×5, 5×5, …
```

我们可以发现每一个子列表都是丑陋数本身(1, 2, 3, 4, 5, …) 乘以 2, 3, 5

接下来我们使用与归并排序相似的合并方法，从3个子列表中获取丑陋数。每一步我们从中选出最小的一个，然后向后移动一步。

三个指针p2,p3,p5分别代表下一次乘以2,3,5的来比较ugly number的index,如果用过了就指向下一个ugly number

Solution

一开始错误的作法！

```
class Solution(object):
    def nthUglyNumber(self, n):
        """
        :type n: int
        :rtype: int
        """
        if n < 0:
            return -1

        if n == 1:
            return 1
        p = 1
        result = [1]
        p2, p3, p5 = 0, 0, 0
        while p < n:
            m2 = result[p2] * 2
            m3 = result[p3] * 3
            m5 = result[p5] * 5
            m = min(m2, m3, m5)
            result.append(m)
            if m == m2:
                p2 += 1
            elif m == m3:
                p3 += 1
            elif m == m5:
                p5 += 1
            p += 1
        return result[-1]
```

这种做法是错误的，因为m2,m3,m5可能会有相同的，会导致重复的数重复加到数组内，其实应该跳过 比如n等于7时

```
2 3 5  n = 2
2
4 3 5  n =3
3
4 6 5  n = 4
4
6 6 5  n = 5
5
6 6 10  n = 6
6
8 6 10  n = 7
6
```

可以看到当m2,m3的指向的数分别乘以2，3时，得到的都是6，6只能加到数组中一次，所以应该对它们都跳过！！

正确的作法

```
class Solution(object):
    def nthUglyNumber(self, n):
        """
        :type n: int
        :rtype: int
        """
        if n < 0:
            return -1

        if n == 1:
            return 1
        p = 1
        result = [1]
        p2, p3, p5 = 0, 0, 0
        while p < n:
            m2 = result[p2] * 2
            m3 = result[p3] * 3
            m5 = result[p5] * 5
            m = min(m2, m3, m5)
            result.append(m)
            if m == m2:
                p2 += 1
            if m == m3:
                p3 += 1
            if m == m5:
                p5 += 1
            p += 1
        return result[-1]
```

Reference

Add Digits (Digit root)

题目描述

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

For example:

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

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

解题方法

brute force

将每个数字加起来，放到stack里，然后进行跟题目一样的运算

找规律

```
0       0
1       1
2       2
3       3
...
10     1
11     2
12     3
...
17     8
18     9
```

可以看出其实是不断重复有规律的，0是0，9的倍数是9，其余就是% 9

Solution

```
class Solution(object):
    def addDigits(self, num):
        """
        :type num: int
        :rtype: int
        """
        if not num:
            return 0
        result = num % 9
        if result:
            return result
        else:
            return 9
```

Add Binary

题目描述

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

For example, a = `"11"` b = `"1"` Return `"100"` .

解题方法

从后往前不断地加，其实有点像merge two list。。。

Solution

```
class Solution(object):
    def addBinary(self, a, b):
        """
        :type a: str
        :type b: str
        :rtype: str
        """
        if not a:
            return b
        if not b:
            return a
        result = ""
        carry = 0
        lengthA = len(a)
        lengthB = len(b)
        p1 = lengthA - 1
        p2 = lengthB - 1
        while p1 >= 0 and p2 >= 0:
            curSum = int(a[p1]) + int(b[p2]) + carry
            if curSum == 0:
                result = "0" + result
                carry = 0
            elif curSum == 1:
                result = "1" + result
                carry = 0
            elif curSum == 2:
                result = "0" + result
                carry = 1
            elif curSum == 3:
                result = "1" + result
                carry = 1
            p1 -= 1
            p2 -= 1
        while p1 >= 0:
            curSum = int(a[p1]) + carry
            if curSum == 0:
                result = "0" + result
                carry = 0
            elif curSum == 1:
                result = "1" + result
                carry = 0
            elif curSum == 2:
                result = "0" + result
                carry = 1
            elif curSum == 3:
                result = "1" + result
                carry = 1
            p1 -= 1
        while p2 >= 0:
            curSum = int(b[p2]) + carry
            if curSum == 0:
                result = "0" + result
                carry = 0
            elif curSum == 1:
                result = "1" + result
                carry = 0
            elif curSum == 2:
                result = "0" + result
                carry = 1
            elif curSum == 3:
                result = "1" + result
                carry = 1
            p2 -= 1

        if carry == 1:
            result = "1" + result

        return result
```

Reference

Trailing Zeros

题目描述

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

Note: Your solution should be in logarithmic time complexity.

解题方法

我们可以注意到有多少个0只跟有多少2和5有关，而2的数量一定比5多，所以只要关心有多少5就可以。

比如说 `5!` 有一个5， `11!` 有两个5 而 `28!` 有6个5，5, 10, 15, 20, 25(俩个)

我们可以发现对于5的Power也需要考虑加上它们里面的5，直到不再含有更大的power

所以解题的方法就是

- n / 5, 得到所有5的一次方的数量，加到结果
- 然后n/ (5^2), 得到所有5的二次方的数量，又多了这么多的5，加到结果里
- 5的三次方。。。
- 直到n不大于下一个5的k次方就可以停止了，说明已经不会再有5了

时间复杂度

取决于n大概是5的多少次方，所以是 `Math.round(Log5(n))`

Solution

```python
class Solution(object):
    def trailingZeroes(self, n):
        """
        :type n: int
        :rtype: int
        """
        if n < 5:
            return 0
        result = 0
        fivePower = 1
        numberOfFive = n / (5 ** fivePower)
        while numberOfFive:
            result += numberOfFive
            fivePower += 1
            numberOfFive = n / (5 ** fivePower)

        return result
```

Reference

- geeksforgeeks

Max Points on a Line

题目描述

Given n points on a 2D plane, find the maximum number of points that lie on the same straight line.

解题方法

最**naive**的解法

正常表示一条直线应该是用

```
y= mx + b
m - slope
b - y-intercept
```

但是这样找出在同一条直线上的点太过复杂，需要 `O(n^3)` 的时间

- 确定起点i, 终点j，得出上面的公式 O(N^2)
- 对其他的每一点，用公式判断它是否在直线上 O(N)

改进解法

用hashtable记录对同一个起点出现过的斜率，用空间换时间

- 对每一点i, 确定它为起点，计算每一点对它的斜率slope，用hashmap记录出现过的斜率的点数
- 如果与点i重合，那么应该和所有斜率在同一条直线上
- 如果斜率出现过，那么斜率相同的点必然在同一条直线上
- 对起点i记录下以它为起点的一条直线上最多的点

loop每一个点作为起点，再loop除了它以为之后的点（前面的点作为起点已经算过），时间复杂度 `O(n^2)`

Solution

```
# Definition for a point.
# class Point(object):
#     def __init__(self, a=0, b=0):
#         self.x = a
#         self.y = b

class Solution(object):
    def maxPoints(self, points):
        """
        :type points: List[Point]
        :rtype: int
        """
        if not points:
            return 0
        length = len(points)
        maxPoints = 1
        for i in range(length):
            slopeDic = {}
            same = 0
            for j in range(i+1,length):
                if points[i].x == points[j].x and points[i].y == po
                    same += 1
                else:
                    slope = self.getSlope(points[i], points[j])
                    if slope in slopeDic:
                        slopeDic[slope] += 1
                    else:
                        slopeDic[slope] = 1
            if slopeDic:
                curMax = max(slopeDic.values()) + same + 1
            else:
                curMax = same + 1
            maxPoints = curMax if maxPoints < curMax else maxPoints
        return maxPoints

    def getSlope(self, p1, p2):
        if p1.x == p2.x:
            return "vertical"
        else:
            return 1.0 * (p2.y - p1.y) / (p2.x - p1.x)
```

Reference

- max points on a line

Palindrome Number

题目描述

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

解题方法

extra space

变换成array, 然后方便的得到左右两边的digit

no extra space

因为不能用额外空间，所以不能转化成array再做

尝试用最naive的方法去做， 用数学方法不断得到最大的digit和最小的digit，比较之后将它们去掉，除数的大小也要调整

Solution

```
class Solution(object):
    def isPalindrome(self, x):
        """
        :type x: int
        :rtype: bool
        """
        if x < 0:
            return False
        largeD = 1
        if x < 10:
            return True

        while x >= largeD:
            largeD *= 10
        largeD /= 10

        while largeD > 1:
            firstDigit = x / largeD
            lastDigit = x % 10
            if firstDigit != lastDigit:
                return False
            x = x % largeD
            x = x / 10
            largeD /= 100

        return True
```

Reference

atoi

题目描述

Implement atoi to convert a string to an integer.

Hint: Carefully consider all possible input cases. If you want a challenge, please do not see below and ask yourself what are the possible input cases.

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

解题方法

string to integer， 遇到不是number的character就停止，将之前的变为int

1. 首先要去掉两边的whitespace, python string提供了 `strip()` 函数，如果不用的话可以用指针解决
2. 要判断是否有sign， `sign` 也只能在第一个，所以判断之后去掉
3. 对于后面的就比较简单了，遇到非Number的就停止，其余的不断加到result里
4. 最后在处理一下overflow的问题

Solution

```python
class Solution(object):
    def myAtoi(self, str):
        """
        :type str: str
        :rtype: int
        """
        if not str:
            return 0
        str = str.strip()
        length = len(str)
        isNegative = False
        pointer = 0
        if pointer == length:
            return 0

        if str[pointer] == "-":
            isNegative = True
            pointer += 1
        elif str[pointer] == "+":
            pointer += 1

        result = 0
        while pointer < length:
            if not str[pointer].isdigit():
                break
            else:
                result *= 10
                result += int(str[pointer])
                pointer += 1

        if isNegative and result > 2147483648:
            return -2147483648
        elif not isNegative and result > 2147483647:
            return 2147483647

        if isNegative:
            return -1 * result
        else:
            return result
```

Reference

Count Primes

题目描述

解题方法

最简单的做法

对小于n的每一个数，检查它是否能被小于它的数整除， `O(n^2)`

倍数去除法

- 先建立一个都是True的长度为n的array, 代表Index为i的是否是prime
- 从2开始到N-1,可以分别不断地将它们的倍数设为False

优化1

这里就有技巧了，去掉2的所有倍数，再去掉3的所有倍数，当去掉4的倍数时，因为4已经被2去掉，所以4的倍数也肯定是2的倍数。 所以我们在去掉k的倍数时，先check k是否还为prime, 如果已经不是，说明k已经被小于他的因数去掉，就可以跳过了

优化2

Let's write down all of 12's factors:

```
2 × 6 = 12
3 × 4 = 12
4 × 3 = 12
6 × 2 = 12
```

As you can see, calculations of 4 × 3 and 6 × 2 are not necessary. Therefore, we only need to consider factors up to √n because, if n is divisible by some number p, then n = p × q and since p ≤ q, we could derive that p ≤ √n. Yes, the terminating loop condition can be p < √n, as all non-primes ≥ √n must have already been marked off.

sieve of eratosthenes

The Sieve of Eratosthenes is one of the most efficient ways to find all prime numbers up to n. But don't let that name scare you, I promise that the concept is surprisingly simple.
http://en.wikipedia.org/wiki/Sieve_of_Eratosthenes#Algorithm_complexity

Solution

```
class Solution(object):
    def countPrimes(self, n):
        """
        :type n: int
        :rtype: int
        """
        if n <= 2:
            return 0
        isPrime = [True for i in range(n)]
        isPrime[0] = False
        isPrime[1] = False
        i = 2
        while i ** 2 < n:
            if isPrime[i]:
                j = i + i
                while j < n :
                    isPrime[j] = False
                    j+=i
            i += 1

        result = 0
        for bl in isPrime:
            if bl:
                result += 1
        return result
```

Reference

Reverse Integer

题目描述

Reverse digits of an integer.

```
Example1: x = 123, return 321
Example2: x = -123, return -321
```

解题方法

跟reverse bits一样，只不过这里是signed value, 每次module 10更快

注意点

- sign
- overflow的问题， 32位integer range `-2147483648 to 2147483647`

Solution

```
class Solution(object):
    def reverse(self, x):
        """
        :type x: int
        :rtype: int
        """
        if not x:
            return 0
        sign = 1
        if x < 0:
            sign = -1
        x = abs(x)

        result = 0
        while x:
            curNum = x % 10
            result = result * 10 + curNum
            x = x / 10

        result = sign * result
        if result > 2**31 - 1 or result < -1 * (2 ** 31):
            return 0
        else:
            return result
```

Reference

Roman to Integer

题目描述

Given a roman numeral, convert it to an integer.

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

解题方法

这两道题目考的频率还挺高的，主要是要记得罗马数字的概念, 字符对应的数字。将罗马数字转换为整数其实就是按顺序读出来

- 如果前面的数比后面的数大，那么就是additional notation的，就加到结果了
- 如果前面的数比后面的数小，那么是substractive notation的，而substractive notation的只能是两位，所以只要判断前后两位就可以

从后往前读

从倒数第二位往前读，如果当前的字符s[i]代表的数字比s[i+1]代表的数字小，说明它们两个是substractive的，应当减去当前的value

从前往后读

用一个variable pre保存前一个字符，没遇到一个新的都先加到result上，如果发现是substractive的，就应该减去它的两倍（因为之前加过一次）

Solution

```
class Solution(object):
    def romanToInt(self, s):
        """
        :type s: str
        :rtype: int
        """
        if not s:
            return 0
        dic = {
            "I": 1,
            "V": 5,
            "X": 10,
            "L": 50,
            "C": 100,
            "D": 500,
            "M": 1000
        }
        length = len(s)
        result = 0
        result += dic[s[-1]]
        for i in range(length - 2, -1, -1):
            if dic[s[i]] >= dic[s[i+1]]:
                result += dic[s[i]]
            else:
                result -= dic[s[i]]
        return result
```

Reference

Integer to Roman

题目描述

Given an integer, convert it to a roman numeral.

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

解题方法

1、罗马数字共有7个，即I（1）、V（5）、X（10）、L（50）、C（100）、D（500）和M（1000）。

2、左减右加逻辑，以及右加不能超过三位，左减只能有1位。

- addtion notation的表达方法基本就是分隔的将单独的value加起来
- substract notation的表达方法是两个字符代表一个数，能够代表的数也是有限的

所以我们可以将能够单独表达的数字列出来

```
1000 900 500 400 100 90 50 40 10 9 5 4 1
```

罗马数字基本是从左到右，从大到小排列的，所以按照这个value array来做greedy判断，就可以得到结果

Solution

```
class Solution(object):
    def intToRoman(self, num):
        """
        :type num: int
        :rtype: str
        """
        if not num:
            return ""
        values = [1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 
        symbols = ["M", "CM", "D", "CD", "C", "XC", "L", "XL", "X",
        index = 0
        result = ""
        while num:
            cur = num / values[index]
            for j in range(cur):
                result += symbols[index]
            num %= values[index]
            index += 1
        return result
```

Reference

Multiply Strings

题目描述

Given two numbers represented as strings, return multiplication of the numbers as a string.

Note: The numbers can be arbitrarily large and are non-negative.

解题方法

就是用笔算乘法的方法

- 用一个array先将num1的低i位和num2的第j位存起来，i+j位相乘的积存在 新array的第i+j位
- 然后再将这个array按照从低位到高位相加，carry处理的方法得到结果

注意点

- 最后如果前面是0的话要去掉，但是如果只剩“0”的话就要返回

Solution

```
class Solution(object):
    def multiply(self, num1, num2):
        """
        :type num1: str
        :type num2: str
        :rtype: str
        """
        if num1 == 0 or num2 == 0:
            return "0"
        length1 = len(num1)
        length2 = len(num2)

        tmp = [0 for i in range(length1 + length2)]
        result = ""

        for i in range(length1):
            d1 = int(num1[length1-1-i])
            for j in range(length2):
                d2 = int(num2[length2-1-j])
                tmp[length1 + length2 - 1 - i - j] += d1 * d2

        carry = 0
        for i in range(length1+length2):
            cur = tmp[length1+length2-1-i] + carry
            digit = cur % 10
            carry = cur / 10
            result = str(digit) + result

        if carry != 0:
            result = str(carry) + result
        while result[0] == "0":
            if result == "0":
                break
            result = result[1:]
        return result
```

Reference

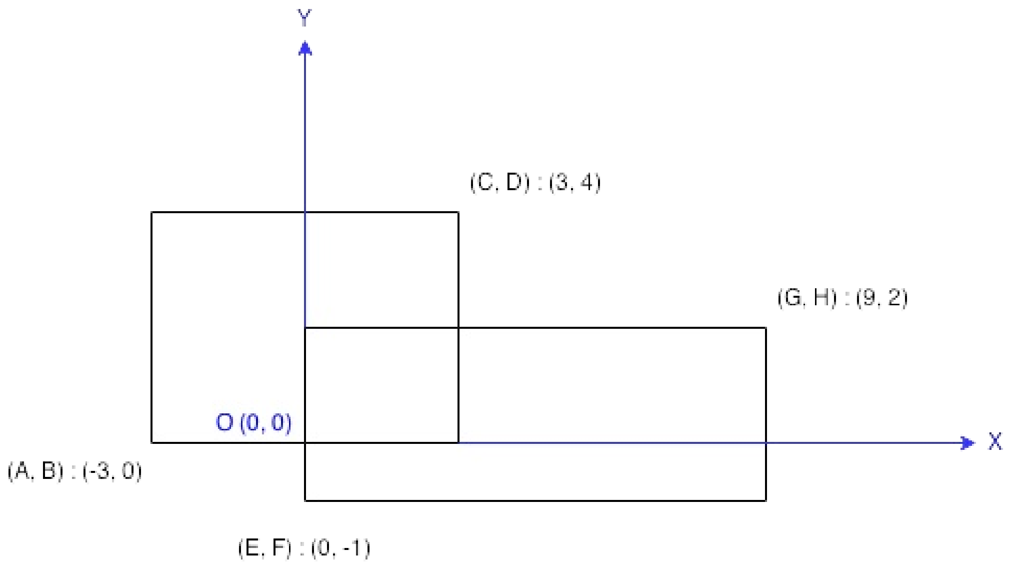
- 大整数string的相乘

Rectangle Area

题目描述

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

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



解题方法

主要就是考查是否能够考虑到各种情况，在做这种题目的时候最好画出各种图来处理

- 不重叠的情况，在左右
- 重叠的面积的计算，两个长方形的面积 - 重叠部分的面积

在discuss里看到了一个很巧妙的做法, assume两个bottom-left的点都在right-up点的左下，这样才可能重叠。 直接求出如果重叠的话那么边界的坐标，如果边界左边不能组成一个rectangle的时候，说明并不重叠。

Solution

```
        leftX = max(A, E)
        leftY = max(B, F)
        rightX = min(C, G)
        rightY = min(D, H)

        totalArea = (C-A) * (D-B) + (G-E) * (H-F)
        if rightX < leftX or rightY < leftY:
            # means they are not overlapped
            return totalArea
        else:
            overlapArea = (rightX - leftX) * (rightY - leftY)
            return totalArea - overlapArea
```

Reference

Perfect Square

题目描述

Given a positive integer n, find the least number of perfect square numbers (for example, 1, 4, 9, 16, ...) which sum to n.

For example, given `n = 12`, return 3 because `12 = 4 + 4 + 4`; given `n = 13`, return `2` because `13 = 4 + 9`.

解题方法

数论

数论的方法，任意一个数都可以用不多于4个平方和表示出来

DP

- 建立一个长度为n+1的array `dp`
- 对于是平方和的数， `dp[i] = 1`
- 对于i从1到n，只要`i + k ^ 2 <= n`，就尝试更新`dp[i+k^2]`的值

Solution

数论

```
class Solution(object):
    def numSquares(self, n):
        """
        :type n: int
        :rtype: int
        """
        if not n:
            return 0

        while n % 4 == 0:
            n /= 4

        if n % 8 == 7:
            return 4

        a = 0
        while (a ** 2) <= n:
            b = int(math.sqrt(n - a ** 2))
            if ((a**2) + (b**2)) == n:
                if a == 0:
                    return 1
                else:
                    return 2
            a += 1

        return 3
```

python DP, 过不了时间

```
class Solution(object):
    def numSquares(self, n):
        """
        :type n: int
        :rtype: int
        """
        if not n:
            return 0

        dp = [sys.maxint for i in range(n+1)]
        root = 1
        while (root ** 2) <= n:
            dp[root**2] = 1
            root += 1

        for i in range(1, n+1):
            root = 1
            while i + (root ** 2) <= n:
                dp[i+(root**2)] = min(dp[i+(root**2)], dp[i] + 1)
                root += 1

        return dp[n]
```

Reference

- 书影博客
- 数论解释

Valid Number

题目描述

Validate if a given string is numeric.

Some examples:

- "0" => true
- " 0.1 " => true
- "abc" => false
- "1 a" => false
- "2e10" => true

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

解题方法

- 应该只能最多有一个 `.`
- 检查是否有 `e` ,如果有分为两部分分别check, 后面的部分不应该有 `.`
- 检查是否有符号 `+` or `-`

Solution

```
class Solution(object):
    def isNumber(self, s):
        """
        :type s: str
        :rtype: bool
        """
        s = s.strip()

        if not s:
            return False
        if s[-1] == "e":
            return False

        arr = s.split("e")
        if len(arr) < 0 or len(arr) > 2:
            return False

        valid = self.check(arr[0], False)

        if len(arr) == 2 and valid:
            return self.check(arr[1], True)

        return valid


    def check(self, s, hasDot):
        if len(s) == 0:
            return False
        if s[0] == "+" or s[0] == "-":
            s = s[1:]
        # check if anything left after sign
        if len(s) == 0:
            return False
        if s == ".":
            return False
        for i in range(len(s)):
            if s[i] == ".":
                if hasDot:
                    return False
                hasDot = True
            elif s[i] not in "0123456789":
                return False

        return True
```

Reference

Excel Sheet Column Number

题目描述

Related to question Excel Sheet Column Title

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

For example:

```
    A -> 1
    B -> 2
    C -> 3
    ...
    Z -> 26
    AA -> 27
    AB -> 28
```

解题方法

这一题的解法是将每一个独立的字母分开来看，相同于这是一个26进制的数， 每一个字母的number通过与 `A` 的ascii码的差来得到。

Solution

```python
class Solution(object):
    def titleToNumber(self, s):
        """
        :type s: str
        :rtype: int
        """
        if not s:
            return 0

        length = len(s)
        result = 0
        for i in range(length):
            result = result * 26 + ord(s[i]) - ord('A') + 1

        return result
```

Reference

Excel Sheet Column Title

题目描述

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

For example:

```
    1 -> A
    2 -> B
    3 -> C
    ...
    26 -> Z
    27 -> AA
    28 -> AB
```

解题方法

这个应该参照进制转化的方法，不断地除26取余数

- 如果余数是1-25， 就按照A-Y取
- 如果余数是0， 那么该位就应该是Z，并且下一次N要减去**1**来计算，因为下一位应该是从""开始而不是"A",index应该减1
    - 这个可以理解成，虽然Z的时候26的倍数多了一个，但是并没有进位，所以要减去1

Solution

```python
class Solution(object):
    def convertToTitle(self, n):
        """
        :type n: int
        :rtype: str
        """
        if not n:
            return ""

        result = ""
        while n > 0:
            r = n % 26
            n = n / 26
            if r:
                result = chr(ord('A') + r - 1) + result
            else:
                result = "Z" + result
                n -= 1

        return result
```

Reference

- excel sheet column title

Bit Manipulation

题目列表

XOR

- Single Number
- Single Number II
- Single Number III
- Missing Number

&

- Number of 1 Bits
- Power of Two

Shift

- Reverse Bits
- Reverse Integer
- Divide 2 Integers

Math

- Repeated DNA Sequence
- Bitwise AND of Numbers Range

问题描述

这类题目都涉及到bit的基本操作

- `&`
- `|`
- `^`

- `~`
- shift `>>` and `<<`

math中的mathwithoutOperator也往往是使用bit manipulation：

- 乘法，除法 - `shft`
- 加法，减法 - `^` 得到carry, `^` 的到sum， carry再shift

Reference

Single Number

题目描述

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

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

解题方法

记录每个bit的出现次数

用一个array或者hashmap记录每个bit 1的出现次数，如果是奇数次就是single number里的

XOR

Solution

```
class Solution(object):
    def singleNumber(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        result = 0
        for num in nums:
            result ^= num

        return result
```

Reference

Single Number II

题目描述

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

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

解题方法

记录出现次数

要有extra memory

```java
public class Solution {
    public int singleNumber(int[] nums) {
        int length = nums.length;
        int[] count = new int[32];
        int result = 0;
        for (int i = 0; i < 32; i++){
            for (int j = 0; j < length; j++) {
                if (((nums[j] >> i) & 1) == 1){
                    count[i]++;
                }
            }
            result |= ((count[i] % 3) << i);
        }
        return result;
    }
}
```

这里可能是signed value,所以对Java有效，python的解不对

利用出现3次的特性

如果对于ith bit的出现次数将它module 3的话，每个bit只能是0或者1，因为除去single number要不就是出现3次，要不0次，module 3 都是0， 再加上single number，只会是0或1

可以用3个 bitmask variables

- `ones` 来代表ith bit had appeared once
- `twos` 来代表ith bit had appeared twice
- `threes` 来代表ith bit had appeared three times

当ith bit出现第三次的时候，将ones和twos里的ith都清0。

Solution

```
class Solution(object):
    def singleNumber(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        ones = 0
        twos = 0
        threes = 0
        for num in nums:
            # twos depends on ones and current bit
            # 如果ith bit, ones为1, A[i]也为1, 就设为1, 否则就保持原值
            twos |= ones & num
            # ones根据现在的i来判断, 如果ones本来就是1, 又出现, 就应该设two
            # 当下一次ones从0变为1的时候, 才设threes
            ones ^= num
            # threes根据ones和twos,当twos为1, ones又一次从0变成1后, thr
            threes = ones & twos
            # 根据threes将ones和twos对应的Bit清零
            ones &= ~threes
            twos &= ~threes

        #最后返回的是Ones
        #因为出现3次的应该都被清零了, 剩下就是single number出现了一次的
        return ones
```

Reference

Single Number III

题目描述

Given `2*n + 2` numbers, every numbers occurs twice except two, find them.

Example Given `[1,2,2,3,4,4,5,3]` return `1` and `5`

Challenge `O(n)` time, `O(1)` extra space.

解题方法

- 全部异或，然后对结果找到第一个为1的bit, 这个bit必然是两个single number不一样的Bit
- 对于这个bit,将原来的数组分为两组
- 然后再按照single number的方法分别找到两个数

不用**extra space**的做法

我们已经知道消去从右到左第一个1 bit的方法是 `x & (x-1)` , 那么我们将 `x-1` 的bit flip一下变为 `~(x-1)` , 再 `x & (~(x-1))` ,就可以 得到一个只有这一个bit位1的mask, 然后通过这个mask来直接在loop中分组做xor, 就像single number i一样可以找到两个single number

Solution

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

        xor = 0
        for num in nums:
            xor ^= num

        a = 0
        b = 0
        bitMask = xor & (~(xor-1))
        for num in nums:
            if num & bitMask:
                a ^= num
            else:
                b ^= num

        return [a, b]
```

Reference

- 书影博客
- without extra memory
- single number i ii iii

Single Number IV

题目描述

一个数组中有三个数字a、b、c只出现一次，其他数字都出现了两次。请找出三个只出现一次的数字。

解题方法

根据上一道single number III, 只要能找出三个其中的一个， 另外两个就能够用上题的方法找出

Solution

Reference

Missing Number

题目描述

Given an array containing n distinct numbers taken from 0, 1, 2, ..., n, find the one that is missing from the array.

For example, Given `nums = [0, 1, 3]` return `2` .

解题方法

XOR

- 1-n的XOR
- 现有array的XOR

Solution

```
class Solution(object):
    def missingNumber(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        length = len(nums)
        full = 0
        for i in range(length+1):
            full = full ^ i

        XOR = nums[0]
        for i in range(1, length):
            XOR = XOR ^ nums[i]

        return full ^ XOR
```

Reference

Count of 1 bits

题目描述

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

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

chanllenge

if number n has m 1 bits, can you do it in `O(m)`

解题方法

shift and count 1 ones

最简单的方法，check每一个Bit是否是1，不断地shift到下一个bit 这样就要检查每一个bit, `O(logn)`

bitwise操作，直接每次找到一个1bits

这个具体操作过程是这样的，当我们从一个integer里减去1的时候，会将从右往左的所有0变成1，将最右边的1变为0

比如

`110100` , 减去1，变为 `110011` , 这时候我们再对这两个数做一个 `&` , 变为 `110000` 。

所以每次的 `n & (n-1)` 可以将n的最后一个1 bit消去，用 `while` 判断是否已经消去所有的1

Solution

```
class Solution(object):
    def hammingWeight(self, n):
        """
        :type n: int
        :rtype: int
        """
        if n == 0:
            return 0
        result = 0
        while n:
            result += 1
            n = n & (n-1)

        return result
```

Reference

- geeksforgeeks

Power of Two

题目描述

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

解题方法

2的Power都是只有1个bit上为1的，所以利用 `n & (n-1)` 消去最后一个为1的bit，然后再check是否为0，如果已经是0的话 那就是2的Power

Solution

```
class Solution(object):
    def isPowerOfTwo(self, n):
        """
        :type n: int
        :rtype: bool
        """
        if not n:
            return False
        n = n & (n-1)
        if n:
            return False
        else:
            return True
```

Reference

Reverse Bits

题目描述

Reverse bits of a given 32 bits unsigned integer.

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

Follow up: If this function is called **many times**, how would you **optimize** it?

解题方法

bit shift

- 在一个while loop循环中，因为已经定了是32位的整数，所以要Loop32次
- 将当前的bit加到另一个变量result上
- n不断右shfit, result不断左shift

swap 2 bits at a time

每次swap对应的两位

```
public int reverseBits(int n) {
    for (int i = 0; i < 16; i++) {
        n = swapBits(n, i, 32 - i - 1);
    }

    return n;
}

public int swapBits(int n, int i, int j) {
    //find the value of these 2 bits
    int a = (n >> i) & 1;
    int b = (n >> j) & 1;

    //if they are not the same, swap them
    if ((a ^ b) != 0) {
        //因为它们肯定不同，所以直接用^,将0变成1，将1变成0
        return n ^= (1 << i) | (1 << j);
    }

    return n;
}
```

follow up优化

根据leetcode上的一个discuss而来，思想就是将每4位直接建立一个Lookup table

```
value -> reverse

0 ------> 0

1 ------> 8

... ------> ...

15 ------> 15
```

然后每次将这4位找到对应的reverse,每次shift4位来计算

> 其实这种思想可以继续优化，建立更大的Lookup table就可以

Solution

```python
class Solution(object):
    def reverseBits(self, n):
        """
        :type n: int
        :rtype: int
        """
        if not n:
            return 0
        result = 0
        times = 0
        while times < 32:
            curBit = n & 1
            result = (result << 1) + curBit
            n = n >> 1
            times += 1

        return result
```

Reference

- 书影博客
- programcreek
- follow up 优化

Repeated DNA Sequences

题目描述

All DNA is composed of a series of nucleotides abbreviated as A, C, G, and T, for example: "ACGAATTCCG". When studying DNA, it is sometimes useful to identify repeated sequences within the DNA.

Write a function to find all the 10-letter-long sequences (substrings) that occur more than once in a DNA molecule.

For example,

```
Given s = "AAAAACCCCCAAAAACCCCCCAAAAAGGGTTT",

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

解题方法

原来的解法是因为A,C,G,T的ascii码只有后三位不同，所以用这后三位来区分，每次取30位，不断shift,放到hashmap里看 是否有重复的。

因为python可以用string做key,所以也可以直接用10-letter-long的字符串做key,就很简单了。

注意点

- 去掉重复的
- Index range
- 长度小于10的

Solution

```
class Solution(object):
    def findRepeatedDnaSequences(self, s):
        """
        :type s: str
        :rtype: List[str]
        """
        dic = {}
        length = len(s)
        result = []
        if length < 10:
            return []
        for i in range(length - 9):
            subStr = s[i:i+10]
            if subStr in dic:
                if dic[subStr] == 1:
                    result.append(subStr)
                dic[subStr] += 1
            else:
                dic[subStr] = 1

        return result
```

Reference

- grandyang

Bitwise AND of Numbers Range

题目描述

Given a range [m, n] where `0 <= m <= n <= 2147483647`, return the bitwise AND of all numbers in this range, inclusive.

For example, given the range `[5, 7]`, you should return `4`.

解题方法

brute force

显然最简单的就是将每个and一次，但是时间复杂度太高

找规律

我们已经知道 `x & (x-1)` 会消去x最右边为1的bit, 那么对于一个数y, `y & (y+1)` 是同样的效果， `y & ( y + 2)` 其实就能够bit 2设为0

那么这道题其实并不难，我们先从题目中给的例子来分析，[5, 7]里共有三个数字，分别写出它们的二进制为：

```
101    110    111
```

相与后的结果为100，仔细观察我们可以得出，最后的数是该数字范围内所有的数的左边共同的部分，如果上面那个例子不太明显，我们再来看一个范围[26, 30]，它们的二进制如下：

```
11010    11011    11100    11101    11110
```

发现了规律后，我们只要写代码找到左边公共的部分即可，然后每次向左移一位，比较m和n是否相同，不同再继续左移一位，直至相同， 然后再将m右移回去得到结果

Solution

```python
class Solution(object):
    def rangeBitwiseAnd(self, m, n):
        """
        :type m: int
        :type n: int
        :rtype: int
        """
        numDiff = 0
        while m != n:
            m >>= 1
            n >>= 1
            numDiff += 1
        return m << numDiff
```

Reference

- codeganker
- 书影博客

Linked list

题目列表

别人总结的列表

- Rotate List
- Copy list with random pointers
- Convert sorted list to Binary Search Tree
- Remove Duplicates
- Remove Duplicates II
- Add Two Numbers

- insertion sort list use a dummy node to add node to new list one at a time

reverse类

- reverse linked list
- reverse print list
- reverse nodes in k group
- swap nodes in pairs
- palindrome list
- reorder list

Merge类

- merge linked list
- merge K sorted List

快慢指针类

- list cycle I, II
- get kth node (get middle node)
- intersection of two lists
- remove nth node from end of list

题目类型

解题技巧

dummy node

当head不确定是否要保留时，就应该用一个dummy node， 一般用法

```
dummy = ListNode(0)
dummy.next = head
cur = dummy
```

merge linked list

这个是很多题目的基础，一定要熟

reverse linked list

最简单的就是建一个Dummy node, 然后不断地将原来List的Node插入到dummy node的后面， 但是这样需要了额外的空间。

更好的方法是用一个variable `pre` 保存前一个node, 一个 `cur` 保存现在的Node, 不断地改变这两个node 的指针关系，并且将 `pre` 和 `cur` 更新向下两个点

tmp

操作linked list的时候，我们经常会改变某些Node的下一个节点， 如果要用到会被改变的node, 要记得用tmp存起来

查找 kth node

对于node之间的距离判断应该弄清楚，比如 `1st node` 和 `nth node` 之间的距离是 `n-1` , 所以如果从head开始到 `nth` node需要移动 `n-1` 步。

while loop从1开始的时候，每移动1步加1，那么终止条件就应该是 `p < n` , 也可以理解为p代表的是 现在指向的是第几个Node。

Reference

- 面试链表大总结
- 面试大总结之一：Java搞定面试中的链表题目
- 轻松搞定面试中的链表题目

- linked list总结1

- linked list总结2

Add Two Numbers

题目描述

You are given two linked lists representing two non-negative numbers. The digits are stored in reverse order and each of their nodes contain a single digit. Add the two numbers and return it as a linked list.

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

解题方法

code的结构类似Merge two sorted list, 只不过变成了加法还有carry的判断。

- dummy node用来保存头的位置
- pre用来跟随head, 并且如果最后一位是0时删掉最后一个node

注意点

- 最终还剩下的carry还要判断是否为1

Solution

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

class Solution(object):
    def addTwoNumbers(self, l1, l2):
        """
        :type l1: ListNode
        :type l2: ListNode
        :rtype: ListNode
        """
        if not l1:
            return l2
        if not l2:
            return l1
        carry = 0
        head = ListNode(0)
        dummy = ListNode(0)
        dummy.next = head
        pre = dummy
        while l1 and l2:
            s = l1.val + l2.val + carry
            head.next = ListNode(0)
            if s < 10:
                head.val = s
                carry = 0
            else:
                head.val = s - 10
                carry = 1
            l1 = l1.next
            l2 = l2.next
            head = head.next
            pre = pre.next

        while l1:
            head.next = ListNode(0)
            s = l1.val + carry
            if s < 10:
                head.val = s
                carry = 0
            else:
                head.val = s - 10
                carry = 1
            head = head.next
            l1 = l1.next
            pre = pre.next

        while l2:
            head.next = ListNode(0)
            s = l2.val + carry
            if s < 10:
                head.val = s
                carry = 0
            else:
                head.val = s - 10
                carry = 1
            head = head.next
            l2 = l2.next
            pre = pre.next

        if carry == 1:
            head.val = 1
        else:
            pre.next = None

        return dummy.next
```

Reference

Remove Duplicates

题目描述

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

For example,

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

解题方法

这一题要保留一个重复的元素，不过我们也可以用Dummy node的方法来解。 因为要判断dummy的后两个Node是否重复，所以要涉及两个元素，如果后面只剩下 `cur.next` ,那后面也就不会有重复了。所以 while loop的条件是 `while cur.next and cur.next.next`

Solution

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

class Solution(object):
    def deleteDuplicates(self, head):
        """
        :type head: ListNode
        :rtype: ListNode
        """
        if not head:
            return None
        dummy = ListNode(0)
        dummy.next = head
        cur = dummy
        while cur.next and cur.next.next:
            if cur.next.val == cur.next.next.val:
                cur.next = cur.next.next
            else:
                cur = cur.next
        return dummy.next
```

Reference

Remove Duplicates II

题目描述

Given a sorted linked list, delete all nodes that have duplicate numbers, leaving only distinct numbers from the original list.

For example, Given `1->2->3->3->4->4->5`, return `1->2->5`. Given `1->1->1->2->3`, return `2->3`.

解题方法

这一题要去除所有重复的元素，head不能确定是否保留，所以要用Dummy node。与1不同的是，要去掉所有重复的元素，记录下这个重复的点的value, 那么就要改重复元素的前一点的next, 而且要不断地向后判断重复元素结束的地方，再设next。

Solution

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

class Solution(object):
    def deleteDuplicates(self, head):
        """
        :type head: ListNode
        :rtype: ListNode
        """
        if not head:
            return None
        dummy = ListNode(0)
        dummy.next = head
        cur = dummy
        while cur.next and cur.next.next:
            if cur.next.val == cur.next.next.val:
                # keep the duplicates value
                curVal = cur.next.val
                # find all the duplicates
                # set the cur.next to the node after these duplicat
                while cur.next and cur.next.val == curVal:
                    cur.next = cur.next.next
            else:
                cur = cur.next
        return dummy.next
```

Reference

Merge Two Sorted Lists

题目描述

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

解题方法

这一题是很多题目的基础，所以要熟练掌握

Solution

```
class Solution(object):
    def mergeTwoLists(self, l1, l2):
        """
        :type l1: ListNode
        :type l2: ListNode
        :rtype: ListNode
        """
        if not l1:
            return l2
        if not l2:
            return l1

        dummy = ListNode(0)
        head = dummy
        while l1 and l2:
            if l1.val < l2.val:
                head.next = l1
                head = head.next
                l1 = l1.next
            else:
                head.next = l2
                head = head.next
                l2 = l2.next

        # 注意这里是if, 直接将后面的连上
        if l1:
            head.next = l1

        if l2:
            head.next = l2

        return dummy.next
```

Reference

Merge K Sorted Lists

题目描述

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

解题方法

方法1

这一题最简单的方法就是利用merge 2 sorted lists的方法，不断地两两比较， 这样的 worst case是第一个list很长，后面的list很短但是都要加在后面，这样第一个List的每个元素都要 遍历 n 次（n为list的数量）， 那么时间复杂度就会为 `O(n^2)`

方法2

上面一种最简单的方法时间复杂度太高，那么我们就应该想如何减少时间复杂度，方法1基本上是从头到尾遍历一遍，对于每个list来说是 `O(n)` 的复杂度，如果 `O(n)` 的复杂度不能满足， 那我们就应该往 `O(logn)` 的方法想。

而 `O(logn)` 最经典的就是二分法了，如何对于这个lists，我们也采取二分法，这样对于每个 list它只需要跟 `LogN` 的的list来比较，这样必然降低了时间复杂度。

方法3 heap

方法2是在不能有extra space的时候的解法，如果可以有extra space, 因为它们已经都是sorted list, 所以可以建一个大小为n（n为list的数量）的heap, 将每个list的head也就是最小值放到Heap里， 然后不断地将heap的最小值加入Dummy node的后边，并且更新被加入的List的head。 就可以做到 `O(n)` 的复杂度。

Solution

主要的code, 省略了merge 2 sorted list的code

```python
class Solution(object):
    def mergeKLists(self, lists):
        """
        :type lists: List[ListNode]
        :rtype: ListNode
        """
        if not lists:
            return None

        length = len(lists)
        return self.mergeHelper(lists, 0, length - 1)

    def mergeHelper(self, lists, start, end):
        if start == end:
            return lists[start]

        mid = start + (end - start) / 2
        leftList = self.mergeHelper(lists, start, mid)
        rightList = self.mergeHelper(lists, mid + 1, end)
        return self.mergeTwoLists(leftList, rightList)
```

Reference

List Cycle

题目描述

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

Follow up:

Can you solve it without using extra space?

解题方法

快慢指针two pointer, 注意点是快指针在前每次移动两个，所以只要判断 `p2` and `p2.next` 存在就可以继续loop

Solution

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

class Solution(object):
    def hasCycle(self, head):
        """
        :type head: ListNode
        :rtype: bool
        """
        if not head or not head.next:
            return False
        p1 = head
        p2 = head
        while p2 and p2.next:
            p1 = p1.next
            p2 = p2.next.next
            if p1 == p2:
                return True

        return False
```

Reference

List Cycle II

题目描述

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

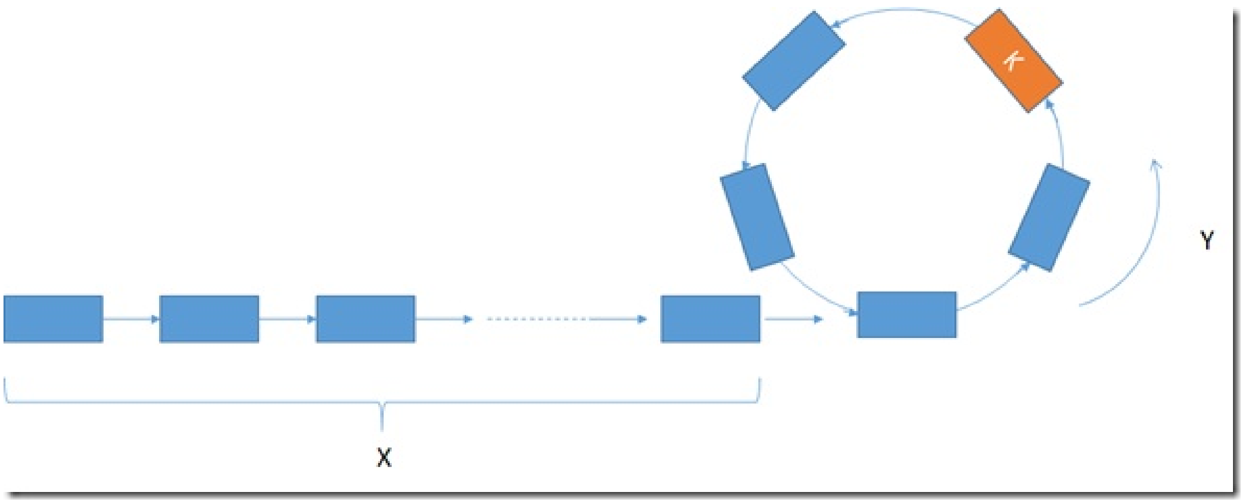
Note: Do not modify the linked list.

Follow up:

Can you solve it without using extra space?

解题方法

画图来解释能够更好地理解题目



依然是快慢指针的思路，那么快指针走的路是慢指针的两倍， 假设如图起点到开头的距离是x, 相遇点距离起点的位置是k, 圈的长度为L

- `x + m*L + k = d`
- `x + n * L + k = 2d`

（m, n分别代表走的圈数）

那么可得 `x + k = (n-2m) * L` , 也就是 `x = (L - k) + (n-2m-1) * L` , 也就是说从开头走到cycle的start, 与从相遇点走到cycle的start的距离的差距是整数倍个圈的长度,

所以这时候再从开头出发， 另一个指针从相遇点出发，它们相遇的点就是cycle的start.

Solution

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

class Solution(object):
    def detectCycle(self, head):
        """
        :type head: ListNode
        :rtype: ListNode
        """
        if not head:
            return None
        p1 = head
        p2 = head
        p3 = head
        while p2 and p2.next:
            p1 = p1.next
            p2 = p2.next.next
            if p1 == p2:
                break

        if not p2 or not p2.next:
            return None

        while p1 != p3:
            p1 = p1.next
            p3 = p3.next

        return p1
```

Reference

- 水中的鱼
- LeetCode Linked List Cycle II 和I 通用算法和优化算法

Remove Nth Node From End

题目描述

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

For example,

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

    After removing the second node from the end, the linked list bec
```

Note:

Given n will always be valid. Try to do this in one pass.

解题方法

遇到linked list的问题首先问自己

- head确定能保留吗？那么需要dummy node吗？

这一题head也不一定能够保留，所以要用dummy node来做，两个快慢指针的做法，快指针先跑出去n步

我们要删去 nth node from end, nth node到tail的距离是n-1, 而我们要获得是Nth node前面一个， 那么它跟tail的距离是n, 所以快指针应该先走出n步。

Solution

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

class Solution(object):
    def removeNthFromEnd(self, head, n):
        """
        :type head: ListNode
        :type n: int
        :rtype: ListNode
        """
        if not head:
            return
        dummy = ListNode(0) # dummy node
        dummy.next = head
        p1 = dummy
        p2 = dummy
        for i in range(n):
            p2 = p2.next

        while p2 and p2.next:
            p1 = p1.next
            p2 = p2.next
        p1.next = p1.next.next

        return dummy.next
```

Reference

Partition List

题目描述

Given a linked list and a value x, partition it such that all nodes less than x come before nodes greater than or equal to x.

You should preserve the original relative order of the nodes in each of the two partitions.

For example, Given `1->4->3->2->5->2` and `x = 3`, return `1->2->2->4->3->5`.

解题方法

如果用类似swap的方法太麻烦，可以新建一个list, 用来存原list中小于x的node, 并且将原Node从原list中删去，最后将原List接到新list的后面。

Solution

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

class Solution(object):
    def partition(self, head, x):
        """
        :type head: ListNode
        :type x: int
        :rtype: ListNode
        """
        if not head:
            return None

        smallHead = ListNode(0)
        dummy = ListNode(0)
        dummy.next = head
        pre = dummy
        s = smallHead

        while pre.next:
            if pre.next.val < x:
                s.next = pre.next
                pre.next = pre.next.next
                s = s.next
            else:
                pre = pre.next

        s.next = dummy.next

        return smallHead.next
```

Reference

Reverse Linked List

题目描述

Reverse a singly linked list.

Hint:

A linked list can be reversed either iteratively or recursively. Could you implement both?

解题方法

比较Intuitive的题目，用 `pre` 记录当前node的前一个Node, 用 `cur` 记录当前的node, 每次只影响两个node的关系

Solution

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

class Solution(object):
    def reverseList(self, head):
        """
        :type head: ListNode
        :rtype: ListNode
        """
        if not head:
            return None
        pre = None
        cur = head
        while cur:
            tmp = cur.next
            cur.next = pre
            pre = cur
            cur = tmp
        return pre
```

Reference

Reverse Linked List II

题目描述

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

For example: Given `1->2->3->4->5->NULL` , `m = 2` and `n = 4` ,

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

Note: Given `m, n` satisfy the following condition: `1 ≤ m ≤ n ≤ length` of list.

解题方法

这一个的关键是在写loop的时候的终止条件判断。

- 首先找到第n-1个和第m+1个Node, 这里的条件判断比较简单
- 然后reverse `nth` to `mth` , 这其间有 `m-n+1` 个node，所以p从0开始，终止条件应该是 `while p <= n - m and cur is not None`

Solution

```
class Solution(object):
    def reverseBetween(self, head, m, n):
        """
        :type head: ListNode
        :type m: int
        :type n: int
        :rtype: ListNode
        """
        if not head:
            return None
        if m == n:
            return head

        dummy = ListNode(0)
        dummy.next = head

        p = 1
        preM = dummy
        while p < m:
            preM = preM.next
            p += 1
        # now preM points to m - 1th node
        mth = preM.next

        if not mth:
            return dummy.next

        p = 1
        preN = dummy
        while p <= n:
            preN = preN.next
            p += 1
        # now preN points to n - 1th node
        nth = preN
        afterN = nth.next

        pre = afterN
        cur = mth
        p = 0
        # reverse between
        while p <= n - m and cur:
            tmp = cur.next
            cur.next = pre
            pre = cur
            cur = tmp
            p += 1

        preM.next = pre

        return dummy.next
```

Reference

Palindrome List

题目描述

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

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

解题方法

简单方法

建一个新的reverse list, 然后比较，但是需要 `O(n)` 的空间复杂度

follow up解法

- 求得长度
- reverse前半部分
- 然后再从两个头比较

注意点

- 长度的奇偶性

Solution

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

class Solution(object):
    def isPalindrome(self, head):
        """
        :type head: ListNode
        :rtype: bool
        """
        if not head:
            return True
        length = self.getLength(head)
        half = length / 2
        p = 0
        pre = None
        cur = head
        # reverse first half
        while p < half:
            tmp = cur.next
            cur.next = pre
            pre = cur
            cur = tmp
            p += 1
        if length % 2 != 0:
            cur = cur.next
        while pre and cur:
            if pre.val != cur.val:
                return False
            pre = pre.next
            cur = cur.next

        return True


    def getLength(self, head):
        length = 0
        while head:
            length += 1
            head = head.next
        return length
```

Reference

Swap Nodes in Pairs

题目描述

Given a linked list, swap every two adjacent nodes and return its head.

For example, Given `1->2->3->4`, you should return the list as `2->1->4->3`.

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

解题方法

相同于每次reverse两个node, 每次影响的是3个node, `pre`, `node1` 和 `node2`

所以不断地改变这三个node的关系，并且向后移，要注意是的判断结束的点.

- 每次两个node为一组，只要node2不为None, 就应该继续swap
- 当node2为None时，说明这一组的node1是tail node, 有奇数个node, 不需要再swap了
- 当node2.next为None时，说明已经到了结尾，有偶数个node

Solution

```
class Solution(object):
    def swapPairs(self, head):
        """
        :type head: ListNode
        :rtype: ListNode
        """
        if not head or not head.next:
            return head

        dummy = ListNode(0)
        dummy.next = head
        pre = dummy

        cur1 = head
        cur2 = head.next
        while cur2:
            tmp1 = cur2.next
            pre.next = cur2
            cur2.next = cur1
            cur1.next = tmp1

            if not tmp1:
                break
            else:
                pre = cur1
                cur1 = tmp1
                cur2 = cur1.next

        return dummy.next
```

Reference

Reorder List

题目描述

解题方法

基本思想也不难，就是

- 找到List的中点，将原list分成两份 (一定要记得将前半部分的结尾设成None)
- 将后半部分reverse
- 然后再merge two list

Solution

```
class Solution(object):
    def reorderList(self, head):
        """
        :type head: ListNode
        :rtype: void Do not return anything, modify head in-place i
        """
        if not head:
            return

        p1 = head
        p2 = head

        while p2 and p2.next:
            p1 = p1.next
            p2 = p2.next.next

        mid = p1.next
        p1.next = None # must set to None, otherwise the result wou

        # reverse second half
        pre = None
        cur = mid
        while cur:
            tmp = cur.next
            cur.next = pre
            pre = cur
            cur = tmp

        q = pre
        p = head
        while p and q:
            tmp = q.next
            q.next = p.next
            p.next = q

            p = p.next.next
            q = tmp
```

Reference

Rotate List

题目描述

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

For example: Given `1->2->3->4->5->NULL` and `k = 2`, return `4->5->1->2->3->NULL`.

解题方法

two pointer方法，找到第 `length-k` 的个，然后将它的next设为None, 并且将原来list的end的next设为head, 再return原来的第 `length-k + 1` 个

注意点

- k有可能大于length

Solution

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

class Solution(object):
    def rotateRight(self, head, k):
        """
        :type head: ListNode
        :type k: int
        :rtype: ListNode
        """
        if not head:
            return None
        length = self.getLength(head)
        k = k % length
        q = length - k
        p = 0
        pointer1 = head
        while p < q - 1:
            pointer1 = pointer1.next
            p += 1
        pointer2 = head
        while pointer2.next:
            pointer2 = pointer2.next
        pointer2.next = head
        result = pointer1.next
        pointer1.next = None
        return result

    def getLength(self, head):
        length = 0
        while head:
            length += 1
            head = head.next
        return length
```

Reference

Copy List with Random Pointer

题目描述

A linked list is given such that each node contains an additional random pointer which could point to any node in the list or null.

Return a **deep copy** of the list.

解题方法

方法1

所有的类似deep copy的题目，传统做法是用一个hashtable

- 先按照正常的方法复制，将Label设为正确的值
- 这一次新node的random pointer是指向旧node的random pointer指向的地方
- 并且用复制的时候用hashtable记录下random pointer， 旧node为key,新node为value， 这样在下一次遍历时就可以将新node的random pointer指到新的node
- 在second loop中根据Map里面保存的value来设正确的random pointer

time conplexity: `O(n)` , space complexity: `O(n)`

方法2

- `first loop` , copy每一个Node, 并且加在原Node的后面，random pointer指向原来的node
- `second loop` , 将偶数位置的node的random pointer指向原来node的后面一位
- `third loop` , split the list，将偶数位置的点变成一个新的list

Solution

方法1

```
class Solution(object):
    def copyRandomList(self, head):
        """
        :type head: RandomListNode
        :rtype: RandomListNode
        """
        if not head:
            return

        dummy = RandomListNode(0)
        pre = dummy
        nodeMap = {}

        # first loop
        # set label
        # set random pointer point to the same old node
        # save {old node: new node} pair in map, for use in next lo
        while head:
            newNode = RandomListNode(head.label)
            newNode.random = head.random
            nodeMap[head] = newNode
            pre.next = newNode
            pre = pre.next
            head = head.next

        pre = dummy
        # second loop
        # set random pointer
        while pre.next:
            if pre.next.random:
                pre.next.random = nodeMap[pre.next.random]
            pre = pre.next

        return dummy.next
```

Reference

- copy list with random pointer

Convert Sorted List to Binary Search Tree

题目描述

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

解题方法

这道题因为list没有random access，所以需要每次找到中点，然后再对两边recursive的构建，所以比较繁琐。

也可以采用bottom-up的方法，不断地构建左子树，让list pointer的位置随着左子树的构建而变化，这样能够一次遍历构建

Solution

bottom-up

```
class Solution(object):
    def sortedListToBST(self, head):
        """
        :type head: ListNode
        :rtype: TreeNode
        """
        if not head:
            return

        length = 0
        cur = head
        while cur:
            length += 1
            cur = cur.next

        root, cur = self.helper(head, 0, length - 1)
        return root

    def helper(self, cur, start, end):
        if start > end:
            return None, cur

        mid = start + (end - start) / 2
        # cur doesn't change with function call cause it's not a cl
        left, cur = self.helper(cur, start, mid - 1)
        root = TreeNode(cur.val)
        cur = cur.next
        right, cur = self.helper(cur, mid + 1, end)

        root.left = left
        root.right = right

        return root, cur
```

Reference

- convert 2 BST

Intersection of Two Linked List

题目描述

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

For example, the following two linked lists:

```
A:          a1 → a2
                   ↘
                     c1 → c2 → c3
                   ↗
B:     b1 → b2 → b3
```

begin to intersect at node `c1`.

解题方法

reverse list

如果可以有extra space, 可以将两个List都reverse一下，然后找最后的相同点。

no extra space

两个list在Intersection之后的长度是一样的，而在Intersection之前的长度是不一样的，那么找出 长度的差diff，并且将长的那个list的pointer先往前移diff的距离，然后在同时往前， 就可以找到intersection的点。

Solution

```
# Definition for singly-linked list.
# class ListNode(object):
```

```
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution(object):
    def getIntersectionNode(self, headA, headB):
        """
        :type head1, head1: ListNode
        :rtype: ListNode
        """
        if not headA or not headB:
            return None

        lA = self.getLength(headA)
        lB = self.getLength(headB)

        diff = abs(lA - lB)
        if lA < lB:
            p = 0
            while p < diff:
                headB = headB.next
                p += 1
        else:
            p = 0
            while p < diff:
                headA = headA.next
                p += 1

        while headA != headB:
            headA = headA.next
            headB = headB.next

        return headA

    def getLength(self, head):
        if not head:
            return 0

        l = 0
        while head:
            l += 1
            head = head.next

        return l
```

Reference

Reference

- greedy总结

Jump Game

题目描述

Given an array of non-negative integers, you are initially positioned at the first index of the array.

Each element in the array represents your maximum jump length at that position.

Determine if you are able to reach the last index.

For example: `A = [2,3,1,1,4]` , return `true` .

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

解题方法

brute force

jump game是从头开始的,那么就是一个从头遍历的过程, 新建一个array `canReach` , `canReach[i]` 表示第i个是否能够跳到。

- 对于index `i`
  - 如果 `canReach[i] == True` ， 那么将i能够跳到的index都设为true
  - 如果 `canReach[i] == False` , 跳过进入下一个循环
- 最终看 `canReach[n-1]` 是否为True
- 时间复杂度 worst case `O(n^2)`
- 空间复杂度 `O(n)`

优化算法

用一个变量 `maxReach` 记录当前能够reach到的最远的地方，如果 `i > maxReach` , 说明这个reach不到，就可以跳过了； 否则就用 `A[i]` 更新 `maxReach` 。

Solution

```
class Solution(object):
    def canJump(self, nums):
        """
        :type nums: List[int]
        :rtype: bool
        """
        if not nums:
            return False
        maxReach = 0
        for i in range(len(nums)):
            if i <= maxReach:
                maxReach = max(maxReach, i + nums[i])

        return maxReach >= len(nums) - 1
```

Reference

- jump game 1, 2

Jump Game II

题目描述

Given an array of non-negative integers, you are initially positioned at the first index of the array.

Each element in the array represents your maximum jump length at that position.

Your goal is to reach the last index in the minimum number of jumps.

For example: Given array `A = [2,3,1,1,4]`

解题方法

方法1，DP

就是用一个array记录目前的最小jump数，然后不断往前更新

- 时间复杂度 `O(n^2)`
- 空间复杂度 `O(n)`

方法2

因为这里是从头开始，并且要求的是最小的jump次数，所以在上面的DP中其实有很多的重复运算，如果已经求出最小的jump次数，那么后面的其实就不用算了。

我们可以将思路换为，每次进一个step，计算出这个step能够达到的范围，如果这个范围 不包含结尾，就之后再前进一个step,再计算。 这样不断从最小step推进，就可以省去重复计算的 过程。

Solution

DP, 超时

```
class Solution:
    # @param A, a list of integers
    # @return an integer
    def jump(self, A):
        # write your code here
        nums = A

        if not nums:
            return 0

        length = len(nums)
        minJumps = [sys.maxint for i in range(length)]
        minJumps[0] = 0
        for i in range(length):
            jumpNum = nums[i]
            farest = min(length - 1, i + jumpNum)
            for j in range(i+1, farest + 1):
                if minJumps[j] > minJumps[i] + 1:
                    minJumps[j] = minJumps[i] + 1

        return minJumps[length - 1]
```

`O(n)` 的方法

```
class Solution(object):
    def jump(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        length = len(nums)
        if length <= 1:
            return 0

        curStart = 0
        end = 0
        curStep = 0

        while curStart <= end:
            curEnd = end
            for i in range(curStart, curEnd + 1):
                if i + nums[i] >= length - 1:
                    return curStep + 1
                end = max(end, i + nums[i]) # 更新下一个step能达到的er
            curStep += 1
            curStart = curEnd + 1 # 下一个step的开头

        return sys.maxint
```

Reference

Gas Station

题目描述

There are N gas stations along a circular route, where the amount of gas at station i is gas[i].

You have a car with an unlimited gas tank and it costs cost[i] of gas to travel from station i to its next station (i+1). You begin the journey with an empty tank at one of the gas stations.

Return the starting gas station's index if you can travel around the circuit once, otherwise return `-1`.

解题方法

1. 如果总油量小于消耗量，那么无解，否则必然有一个解
2. 解法
3. 遍历各个index, 用 `total` 记录到现在积累的油量
4. 对于 `i` , 如果积累的油量 `total` + 加油站的油量 `gas[i]` < `cost[i]` ，说明无法到达下一个stop,那么就将
5. station设为 `i+1` , 因为i之前的stop都不能当做起点，否则无法达到 `i+1`

Solution

```python
class Solution(object):
    def canCompleteCircuit(self, gas, cost):
        """
        :type gas: List[int]
        :type cost: List[int]
        :rtype: int
        """
        if not gas or not cost:
            return -1

        if sum(gas) < sum(cost):
            return -1

        length = len(gas)
        total = 0
        station = 0

        for i in range(length):
            if total + gas[i] < cost[i]:
                total = 0
                station = i + 1
            else:
                total += gas[i] - cost[i]

        return station
```

Reference

Candy

题目描述

There are N children standing in a line. Each child is assigned a rating value.

You are giving candies to these children subjected to the following requirements:

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

解题方法

- 先将每个人都初始为1
- 从左到右扫一遍，使满足如果比左邻居高就大的条件
- 从右到左扫一遍，是满足如果比右邻居高就大的条件

Solution

```
class Solution(object):
    def candy(self, ratings):
        """
        :type ratings: List[int]
        :rtype: int
        """
        if not ratings:
            return 0
        length = len(ratings)

        result = [1 for i in range(length)]

        for i in range(1, length):
            if ratings[i] > ratings[i-1]:
                result[i] = result[i-1] + 1

        for i in range(length-2, -1, -1):
            if ratings[i] > ratings[i+1]:
                result[i] = max(result[i], result[i+1]+1)

        return sum(result)
```

Reference

Product of Array Except Self

题目描述

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

Solve it without division and in `O(n)`.

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

解题方法

- 建立一个array作为result array， `result`
- 先从左到右扫一遍， `result[i]` 表示从左边开头到 `i-1` 的乘积
- 再从右到左扫一遍，对于indexi, 用一个 `right` 变量来记录从右边到 `i+1` 的乘积，因为这里只需要用前一个值，所以不需要再建array
- 最后得到的 `result` array就是结果

Solution

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

        length = len(nums)
        if length == 1:
            return 0
        result = [0 for i in range(length)]
        result[0] = 1

        # 从左到右
        for i in range(1, length):
            result[i] = result[i-1] * nums[i-1]

        # right 变量记录右边到目前为止的乘积
        right = 1
        # 从右到左
        for i in range(length - 1, -1, -1):
            result[i] *= right
            right *= nums[i]

        return result
```

Reference

Binary Tree

Reference

- 心随风飞

Binary Tree遍历

题目列表

- pre-order
- in-order
- post-order

问题描述

有三种解法

- recursive
- iterative用stack
- Morris解法，不用stack,非递归，O(1)的空间（这个需要理解一下）

我们知道正常的遍历时间复杂度是 `O(n)` .空间复杂度是递归栈（或者自己维护的栈）的大小 `O(logn)`

有的时候这不能满足面试官，就要用到moriss的方法，这种方法不需要栈来维护，所以只需要 `O(1)` 的空间

真实面经

有人被问到如果用 `O(1)` 的空间复杂度进行层遍历，我们已经知道了 `O(1)` 的pre-order, in-order, post-order的，对moriss的方法再进行改编一下

Reference

- code ganker
- Moriss Traversal二叉树
- 二叉树遍历-编程之美

Binary Tree Preorder Traversal

题目描述

解题方法

这道题可以用递归解，也可以用iterative的方法来解，一般面试中考察iterative的方法

Solution

- 递归
- iterative stack
- morris方法

递归

因为在递归中的None的返回值与一开始root为None的返回值不一样，所以我们要用一个helper function, 在helper function中对result这个 `mutable` 的list进行修改。

```
# 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 preorderTraversal(self, root):
        """
        :type root: TreeNode
        :rtype: List[int]
        """
        if not root:
            return []
        result = []
        self.helper(result, root)

        return result

    def helper(self, result, root):
        if not root:
            return
        result.append(root.val)
        if root.left:
            self.helper(result, root.left)
        if root.right:
            self.helper(result, root.right)
        return result
```

iterative, 用stack

用一个stack来模拟这个过程，因为是Preorder，所以可以在一开始就将root加到stack上，然后再将right, left `push` 到stack上。

```
class Solution(object):
    def preorderTraversal(self, root):
        """
        :type root: TreeNode
        :rtype: List[int]
        """
        if not root:
            return []
        result = []
        stack = [root]
        while stack:
            cur = stack.pop()
            result.append(cur.val)
            if cur.right:
                stack.append(cur.right)
            if cur.left:
                stack.append(cur.left)

        return result
```

Reference

- preorder traversal分析

Inorder Traversal

题目描述

解题方法

- 递归
- stack
- morris

stack

跟preorder不同的是，这里root不是先visit的

- 我们要用一个cur来保存当前到达的节点（不是记录到result里的节点，而是要Push到stack上的）
- 因为stack是后进后出的，所以一开始可以先将root加到stack,并且不断地将left child加到stack
- 只有当curr为None的时候，说明当前已经肯定没有**left child**，这时候才从stack里pop出来一个加到result数组里, 然后继续往right child走（如果有的话）
- 如果没有right child, 那么cur还是None，stack又会pop，就又到了上面的node

Morris (google onsite的时候被问到了)

Solution

递归

```python
# 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 inorderTraversal(self, root):
        """
        :type root: TreeNode
        :rtype: List[int]
        """
        if not root:
            return []
        result = []
        self.helper(result, root)
        return result

    def helper(self, result, root):
        if not root:
            return

        self.helper(result, root.left)
        result.append(root.val)
        self.helper(result, root.right)
```

stack

```
class Solution(object):
    def inorderTraversal(self, root):
        """
        :type root: TreeNode
        :rtype: List[int]
        """
        if not root:
            return []
        cur = root
        stack = []
        result = []
        while cur or stack:
            if cur:
                stack.append(cur)
                cur = cur.left
            else:
                cur = stack.pop()
                result.append(cur.val)
                cur = cur.right

        return result
```

Reference

- code ganker
    - morris方法需要理解一下

postOrder Traversal

题目描述

解题方法

- 递归
- iterative, stack
- morris方法

stack

postorder比inorder, preorder复杂一些

Solution

递归

```
# 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 postorderTraversal(self, root):
        """
        :type root: TreeNode
        :rtype: List[int]
        """
        if not root:
            return []
        result = []
        self.helper(result, root)
        return result

    def helper(self, result, root):
        if not root:
            return

        self.helper(result, root.left)
        self.helper(result, root.right)
        result.append(root.val)
```

iterative stack

- 用一个cur来track当前到达的点（不是postorder result的顺序，而是当前visit的）
- 用一个pre来track之前遍历的那个点（**postorder**中的，所以只在加到result数组的时候更新），对每一个root,依然是不断将left加到stack上到底
- 当None的时候，是否要往右边走需要判断是否已经visit过right,所以Pre在这里就有了用处

具体步骤是

1. 一开始也是不断地将left child放到stack上,用cur记录当前的node,用pre来记录上一个node（这时候不加到result数组）
2. 当cur为None时，有几种情况
    - 如果当前stack顶元素tmp的右节点存在，并且没有访问过(prev != tmp.right)，就应该往tmp的右边走
    - 否则的话，说明右节点是空的，或者右节点已经访问过了，那么就可以将当前的pop出来加到result里了，这个时候再更新prev

```
# 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 postorderTraversal(self, root):
        """
        :type root: TreeNode
        :rtype: List[int]
        """
        if not root:
            return []
        result = []
        stack = []
        pre = None
        cur = root
        while cur or stack:
            if cur:
                stack.append(cur)
                cur = cur.left
            else:
                tmp = stack[-1] #need to compare with pre, so don't
                if tmp.right and pre != tmp.right:
                    #right exists and is unvisited, so go right
                    cur = tmp.right
                else:
                    top = stack.pop()
                    result.append(top.val)
                    pre = top

        return result
```

Morris

Reference

Binary Tree Level Order Traversal

题目描述

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

For example:

Given binary tree `{3,9,20,#,#,15,7}`,

```
    3
   / \
  9  20
    /  \
   15   7
```

return its level order traversal as:

```
[
  [3],
  [9,20],
  [15,7]
]
```

解题方法

- 用两个变量curNum和nextNum分别记录当前层和下一层的node数量，同时用另一个curList保存当前层的node
- 当current level的node数量减为0的时候，就表示这一层已经遍历完，讲curList加到result数组里，curList = nextNum, nextNum = 0, curList = []

Solution

```python
from collections import deque
# 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 levelOrder(self, root):
        """
        :type root: TreeNode
        :rtype: List[List[int]]
        """
        if not root:
            return []
        queue = deque()
        result = []
        curList = []
        curNum = 1
        nextNum = 0
        queue.append(root)
        while queue:
            cur = queue.popleft()
            curNum -= 1
            curList.append(cur.val)
            if cur.left:
                queue.append(cur.left)
                nextNum += 1
            if cur.right:
                queue.append(cur.right)
                nextNum += 1
            if curNum == 0:
                curNum = nextNum
                nextNum = 0
                result.append(list(curList))
                curList = []

        return result
```

空间 `O(1)` 的做法

```
from collections import deque
# 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 levelOrder(self, root):
        """
        :type root: TreeNode
        :rtype: List[List[int]]
        """
        if not root:
            return []
        levelNum = self.maxDepth(root)
        results = []
        for i in range(levelNum):
            curLevel = []
            levelI = self.getNodesAtLevel(root, i, curLevel)
            results.append(curLevel)
        return results

    def getNodesAtLevel(self, root, level, curLevel):
        if not root or level < 0:
            return False
        if level == 0 and root:
            curLevel.append(root.val)
            return True
        left = self.getNodesAtLevel(root.left, level - 1, curLevel
        right = self.getNodesAtLevel(root.right, level - 1, curLeve
        return left or right

    def maxDepth(self, root):
        if not root:
            return 0
        return 1 + max(self.maxDepth(root.left), self.maxDepth(root
```

上面是先求了 `depth` ，但是这样就多了遍历求depth的时间，可以不求depth, 在 `getNodesAtLevel` 里，当某一层最终 `return False` 时，说明 这一层已经没有node,就可以break了

```
i = 0
while True:
    curLevel = []
    levelI = self.getNodesAtLevel(root, i, curLevel)
    if not levelI:
        break
    i += 1
    results.append(curLevel)
```

递归

还有递归的方法，贴上C++的参考

```
// 递归版,时间复杂度 O(n),空间复杂度 O(n) class Solution {
public:
      vector<vector<int> > levelOrder(TreeNode *root) {
          vector<vector<int>> result;
          traverse(root, 1, result);
          return result;
}
      void traverse(TreeNode *root, size_t level, vector<vector<int
          if (!root) return;
          if (level > result.size())
              result.push_back(vector<int>());
          result[level-1].push_back(root->val);
          traverse(root->left, level+1, result);
          traverse(root->right, level+1, result);
} };
```

Reference

- TREE层遍历

Binary Tree Level Order Traversal II

题目描述

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

For example:

Given binary tree `{3,9,20,#,#,15,7}`,

```
    3
   / \
  9  20
    /  \
   15   7
```

return its bottom-up level order traversal as:

```
[
  [15,7],
  [9,20],
  [3]
]
```

解题方法

将level order的解法改一下，在最后reverse，或者在插入到result里的时候插入到最前面

Solution

```
from collections import deque
# 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 levelOrderBottom(self, root):
        """
        :type root: TreeNode
        :rtype: List[List[int]]
        """
        if not root:
            return []
        queue = deque()
        result = []
        curList = []
        curNum = 1
        nextNum = 0
        queue.append(root)
        while queue:
            cur = queue.popleft()
            curNum -= 1
            curList.append(cur.val)
            if cur.left:
                queue.append(cur.left)
                nextNum += 1
            if cur.right:
                queue.append(cur.right)
                nextNum += 1
            if curNum == 0:
                curNum = nextNum
                nextNum = 0
                result.append(list(curList)) # or insert(0, list(cu
                curList = []

        result.reverse()  # if insert at 0, no need
        return result
```

Reference

Zigzag Level Order Traversal

题目描述

Given a binary tree, return the zigzag level order traversal of its nodes' values. (ie, from left to right, then right to left for the next level and alternate between).

For example: Given binary tree `{3,9,20,#,#,15,7},

```
    3
   / \
  9  20
    /  \
   15   7
```

return its zigzag level order traversal as:

```
[
  [3],
  [20,9],
  [15,7]
]
```

解题方法

同level order,用一个变量记录当前的层数即可

Solution

```
from collections import deque
# 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 zigzagLevelOrder(self, root):
        """
        :type root: TreeNode
        :rtype: List[List[int]]
        """
        if not root:
            return []
        queue = deque()
        result = []
        curList = []
        level = 1
        curNum = 1
        nextNum = 0
        queue.append(root)
        while queue:
            cur = queue.popleft()
            curNum -= 1
            curList.append(cur.val)
            if cur.left:
                queue.append(cur.left)
                nextNum += 1
            if cur.right:
                queue.append(cur.right)
                nextNum += 1
            if curNum == 0:
                curNum = nextNum
                nextNum = 0
                if level % 2 == 0:
                    curList.reverse()
                result.append(list(curList))
                curList = []
                level += 1

        return result
```

Reference

Binary Tree Recursion

题目列表

遍历

- pre order
- in order
- post order

- Max Depth
- Min Depth
- Is Balanced Tree

- Same Tree

- Symmetric Tree
- Validate Binary Search Tree

- Recover Binary Search Tree

- Construct Binary Tree from pre-order and in-order
- Construct Binary Tree from post-order and in-order
- Convert Sorted Array to BST
- Convert Sorted LinkedList to BST

问题描述

Reference

Max Depth of Binary Tree

题目描述

Given a binary tree, find its maximum depth.

The maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

解题方法

recursion,因为是取最大值，所以在当前node为None时可以直接返回0

Solution

```
# 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 maxDepth(self, root):
        """
        :type root: TreeNode
        :rtype: int
        """
        if not root:
            return 0

        return 1 + max(self.maxDepth(root.left), self.maxDepth(root
```

Reference

Minimum Depth of Binary Tree

题目描述

Given a binary tree, find its minimum depth.

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

解题方法

- Min Depth的recursion和max depth有所不同，因为是要选最小值，所以当前点为None时这并不是一个可以返回0的最小值，如果返回0 的话会被 `min()` 函数所取，而是应该在比较时将它省去，因为它不是一个叶子节点，所以返回 `最大整数` ，而这又和一开始root为None要 返回0矛盾，所以要用一个 `helper` 函数。
- 叶子节点时返回1，否则继续向下面子树寻找叶子节点

Solution

```
class Solution(object):
    def minDepth(self, root):
        """
        :type root: TreeNode
        :rtype: int
        """
        if not root:
            return 0
        return self.helper(root)

    def helper(self, root):
        if not root:
            return sys.maxint
        if not root.left and not root.right:
            return 1
        return 1 + min(self.helper(root.left), self.helper(root.rig
```

Reference

Balanced Binary Tree

题目描述

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

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

解题方法

这一题也是用recursive的方法做

- 对于每一个root,分别get左右两边子树的max depth
- 如果balanced，就继续recursive的检查左右子树

Solution

```
# 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 isBalanced(self, root):
        """
        :type root: TreeNode
        :rtype: bool
        """
        if not root:
            return True
        leftDepth = self.getDepth(root.left)
        rightDepth = self.getDepth(root.right)
        diff = abs(leftDepth - rightDepth)
        if diff > 1:
            return False
        else:
            return self.isBalanced(root.left) and self.isBalanced(

    def getDepth(self, root):
        if not root:
            return 0
        return 1 + max(self.getDepth(root.left), self.getDepth(roo
```

Reference

Symmetric Tree

题目描述

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

For example, this binary tree is symmetric:

```
    1
   / \
  2   2
 / \ / \
3  4 4  3
```

But the following is not:

```
    1
   / \
  2   2
   \   \
   3    3
```

解题方法

这一题一开始理解错了，并不是symmetric tree的subtree也要是symmetric tree，而是

- `left.val == right.val`
- if exists, `left.right.val == right.left.val`
- if exists, `left.left.val == right.right.val`

将这个过程recursive地apply到每应该对称的Node上

所以做题前要想清楚，最好画出几个图来看清楚

Solution

```
# 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 isSymmetric(self, root):
        """
        :type root: TreeNode
        :rtype: bool
        """
        if not root:
            return True
        return self.sym(root.left, root.right)

    def sym(self, n1, n2):
        if not n1 and not n2:
            return True
        if n1 and n2 and n1.val == n2.val:
            return self.sym(n1.left, n2.right) and self.sym(n1.rig

        return False
```

Reference

- symmetric tree

Valid Binary Search Tree

题目描述

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

Assume a BST is defined as follows:

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

解题方法

将子树的最大值和最小值作为一个范围

Solution

```
# 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 isValidBST(self, root):
        """
        :type root: TreeNode
        :rtype: bool
        """
        if not root:
            return True
        value = root.val
        return self.isValid(root.left, -sys.maxint, value) and self

    def isValid(self, root, low, high):
        if not root:
            return True
        if root.val > low and root.val < high:
            curValue = root.val
            return self.isValid(root.left, low, curValue) and self.
        else:
            return False
```

Reference

Flatten Binary Tree to Linked List

题目描述

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

For example, Given

```
         1
        / \
       2   5
      / \   \
     3   4   6
```

The flattened tree should look like:

```
   1
    \
     2
      \
       3
        \
         4
          \
           5
            \
             6
```

Hints:

If you notice carefully in the flattened tree, each node's right child points to the next node of a pre-order traversal.

解题方法

解法1：preorder traversal

从题目的例子马上发现其实就是preorder transverse整个tree，然后根据这个访问顺序，后访问到的节点成为先访问到的节点的右子节点。那么最不动脑的方法就是preorder transverse的时候把整个节点序列保存在一个vector中。最后再依次连起来。这个解法的空间复杂度包括了递归深度O(log n)和vector的空间O(n)，所以总的空间复杂度是O(n)。

解法2

- store the right child (we call R)
- find the right-most node of left child
- set R as the right-most node's right child.
- set left child as the right child
- set the left child NULL
- set current node to current node's right child.
- iterate these steps until all node are flattened.

Solution

```
# 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 flatten(self, root):
        """
        :type root: TreeNode
        :rtype: void Do not return anything, modify root in-place i
        """
        if not root:
            return
        left = root.left
        right = root.right
        if left:
            root.right = left
            root.left = None
            leftRight = left
            while leftRight.right:
                leftRight = leftRight.right
            leftRight.right = right

        self.flatten(root.right)
```

Reference

- yu's garden
- 喜刷刷

Lowest Common Ancestor of a Binary Tree

题目描述

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

folloup up 带parent节点的情况

解题方法

方法1

对于每一个node

- 如果和p或者q相等，那么这就是LCA
- 检查p和q在它的哪一边，如果是在两边的话，这个就是LCA
- 如果在同一边的话，就往左或者往右

这个算法worst case的时间复杂度是 `O(n^2)`

方法2

bottom-up找，直接到左右字数里找LCA，然后往上级返回，这里不断recursive其实找到就是 它们本来的两个node，如果左右都找到就说明当前node是LCA,只需要 `O(n)`

经验总结

- 如果有多次重复的问题，那么找出每次重复之间的子问题关系，将重复的量去掉
- 树的recursive不一定要先判断当前的node，如果子树的信息对当前node有用，可以先判断子树变为bottom-up的做法
- 树的bottom-up和top-down的主要差别就是： 先处理当前节点还是先处理子树

follow up

如果带parent,最直观的是从p,q分别建造到root的路径，然后找出LCA,但是这样就需要extra space来保存路径

可以参考intersection of 2 lists, 先找出两个node depth的差距，然后让更深的那个先往上走diff步，然后再同步 往上，就可以通过 `O(1)` 的space找到LCA

Solution

方法1

O(n^2)

```python
class Solution(object):
    def lowestCommonAncestor(self, root, p, q):
        """
        :type root: TreeNode
        :type p: TreeNode
        :type q: TreeNode
        :rtype: TreeNode
        """
        if not root or not p or not q:
            return None

        if root == p or root == q:
            return root

        left1 = self.find(root.left, p)
        left2 = self.find(root.left, q)

        if left1 and left2:
            return self.lowestCommonAncestor(root.left, p, q)
        if not left1 and not left2:
            return self.lowestCommonAncestor(root.right, p, q)

        return root

    def find(self, root, node):
        if not root:
            return False
        if root == node:
            return True

        leftFind = False
        rightFind = False
        if root.left:
            leftFind = self.find(root.left, node)
        if root.right:
            rightFind = self.find(root.right, node)
        return leftFind or rightFind
```

方法2

```python
class Solution(object):
    def lowestCommonAncestor(self, root, p, q):
        """
        :type root: TreeNode
        :type p: TreeNode
        :type q: TreeNode
        :rtype: TreeNode
        """
        if not root or not p or not q:
            return None

        if root == p or root == q:
            return root

        left = self.lowestCommonAncestor(root.left, p, q)
        right = self.lowestCommonAncestor(root.right, p, q)

        if left and right:
            return root

        return left if left else right
```

Reference

- LCA in Binary Tree

Binary Tree Longest Consecutive Sequence

题目描述

Given a binary tree, find the length of the longest consecutive sequence path.

The path refers to any sequence of nodes from some starting node to any node in the tree along the parent-child connections. The longest consecutive path need to be from parent to child (cannot be the reverse).

For example,

```
   1
    \
     3
    / \
   2   4
        \
         5
```

Longest consecutive sequence path is `3-4-5`, so return `3`.

```
   2
    \
     3
    /
   2
  /
 1
```

Longest consecutive sequence path is `2-3`,not `3-2-1`, so return `2`.

解题方法

- recursion,在参数中包含当前的连续seq长度
- 如果left, right child的value是连续的，那么就将长度+1传入下一个call

Solution

```
class Solution(object):
    def longestConsecutive(self, root):
        """
        :type root: TreeNode
        :rtype: int
        """
        if not root:
            return 0

        self.result = 0
        self.helper(root, 1)

        return self.result

    def helper(self, root, curLen):
        self.result = curLen if curLen > self.result else self.resu
        if root.left:
            if root.left.val == root.val + 1:
                self.helper(root.left, curLen + 1)
            else:
                self.helper(root.left, 1)
        if root.right:
            if root.right.val == root.val + 1:
                self.helper(root.right, curLen + 1)
            else:
                self.helper(root.right, 1)
```

Reference

Recover Binary Search Tree

题目描述

Two elements of a binary search tree (BST) are swapped by mistake.

Recover the tree without changing its structure.

Note: A solution using `O(n)` space is pretty straight forward. Could you devise a constant space solution?

解题方法

`O(n)` space的方法

做一个in-order traversal, 然后发现顺序错误的两个node，将它们swap回去就可以了

`O(1)` space

由上面的解法我们可以发现，其实我们并不需要整个in-order的结果，只需要找到这两个在In-order里顺序错乱的node就可以了。 所以我们可以还是做in-order traversal,在这个过程中记录下两个错位的node,然后再将他们swap。但是如果找到这两个点呢， 就像在in-order array中一样，我们需要将它们和 `prev` 比较，根据比较结果来判断。

所以， 我们需要一个变量 `last` 来计算上一个visit的node

Solution

```
class Solution(object):
    def __init__(self):
        self.first = None
        self.second = None
        self.last = None

    def recoverTree(self, root):
        """
        :type root: TreeNode
        :rtype: void Do not return anything, modify root in-place i
        """
        self.traverse(root)

        tmp = self.first.val
        self.first.val = self.second.val
        self.second.val = tmp

    def traverse(self, root):
        if not root:
            return

        self.traverse(root.left)
        if not self.first and self.last and root.val < self.last.va
            self.first = self.last

        if self.first and self.last and root.val < self.last.val:
            self.second = root

        self.last = root
        self.traverse(root.right)
```

Reference

- yu's garden
- yu's coding garden
- 水中的鱼

这到底是不是同一个人……

Count Complete Tree Nodes

题目描述

Given a complete binary tree, count the number of nodes.

Definition of a complete binary tree from Wikipedia:

In a complete binary tree every level, except possibly the last, is completely filled, and all nodes in the last level are as far left as possible. It can have between `1` and `2h` nodes inclusive at the last level `h`.

解题方法

1. 首先计算leftmost和rightmost的高度
2. 如果rightmost和leftmost一样高，说明这棵树是full complete binary tree,那个node的数量就是2 ^ 高度 - 1
3. 如果不是，就recursive计算左右字数的number, 再加上1(root)

Solution

```
# 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 countNodes(self, root):
        """
        :type root: TreeNode
        :rtype: int
        """
        if not root:
            return 0
        hL = 1
        hR = 1
        head = root
        #get leftmost height
        while head.left:
            hL += 1
            head = head.left
        head = root
        #get rightmost height
        while head.right:
            hR += 1
            head = head.right

        if hL == hR:
            return 2 ** hL - 1

        leftNum = self.countNodes(root.left)
        rightNum = self.countNodes(root.right)

        return leftNum + rightNum + 1
```

Reference

- program creek
- 递归解法

Binary Tree Max Path Sum

题目描述

Given a binary tree, find the maximum path sum.

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

For example: Given the below binary tree,

```
       1
      / \
     2   3
Return 6.
```

解题方法

对于数的问题，很多都可以用recursive的思想来解，其实也就是divide and conquer, 将大问题不断地分解成小问题。 我们可以通过发现root的结果与左右child的结果有什么关系 来入手。

对于每一个root, 这个max path sum有三种可能：

- 不通过root, 在左子树里
- 不通过root, 在右子树里
- 通过root,与左右两边的child连接

对于每个root都要比较这三种情况，从而得到最大的path sum, 所以在child往上返回 的时候，不仅仅要返回最大的值，也要返回以这个child自身为一个end的最大path sum. 有一点像DP里面global, local的关系。

所以每个recursive call会返回两个值，再进行比较判断。在判断比较的时候也要注意， 选取global最大结果的时候要取max,而在取通过自己的single path sum的时候，还要与0比较， 因为如果结果是负的，在上面连接这一个child的时候就应该将

它舍去，就是0.

recursive返回的条件：

当节点为None的时候，因为节点value也有可能是负数，所以maxpathsum应该是最小整数，对于Python就是 `-sys.maxint` ,而通过它自申的最大path sum是0

Solution

```python
# 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 maxPathSum(self, root):
        """
        :type root: TreeNode
        :rtype: int
        """
        maxPathSum, single = self.PathSumHelper(root)
        return maxPathSum

    def PathSumHelper(self, root):
        if not root:
            return -sys.maxint, 0

        leftMax, leftSingle = self.PathSumHelper(root.left)
        rightMax, rightSingle = self.PathSumHelper(root.right)
        globalMax = max(leftMax, rightMax, root.val + leftSingle +
        singleMax = max(leftSingle + root.val, rightSingle + root.v

        return globalMax, singleMax
```

Reference

Construct Binary Tree from Inorder and Preorder

题目描述

Given inorder and preorder traversal of a tree, construct the binary tree.

Note: You may assume that duplicates do not exist in the tree.

解题方法

递归

过不了Leetcode OJ

Solution

递归

python的递归过不了OJ， java的可以……

```python
# 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 not preorder or not inorder:
            return
        root = TreeNode(preorder[0])
        indexIn = inorder.index(root.val)
        root.left = self.buildTree(preorder[1:indexIn+1], inorder[
        root.right = self.buildTree(preorder[indexIn+1:], inorder[
        return root
```

Reference

Construct Binary Tree from Inorder and Postorder

题目描述

Given inorder and postorder traversal of a tree, construct the binary tree.

Note: You may assume that duplicates do not exist in the tree.

解题方法

Solution

递归

过不了leetcode OJ

```
# 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, inorder, postorder):
        """
        :type inorder: List[int]
        :type postorder: List[int]
        :rtype: TreeNode
        """
        if not postorder or not inorder:
            return
        root = TreeNode(postorder[-1])
        indexIn = inorder.index(root.val)
        root.left = self.buildTree(inorder[:indexIn], postorder[:ir
        root.right = self.buildTree(inorder[indexIn+1:], postorder|
        return root
```

Reference

Unique Binary Search Tree

题目描述

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

For example, Given `n = 3`, there are a total of `5` unique BST's.

```
   1         3     3      2      1
    \       /     /      / \      \
     3     2     1      1   3      2
    /     /       \                 \
   2     1         2                 3
```

解题方法

这种求数量的，可以往DP的方向想。

当以 `i` 为root时，左子树就是 `[1:i]` , 右子树就是 `[i+1:n+1]` 构成，而左子树和右子树的root又可以从每一个Index遍历的选一遍。

`count[i]` 表示到正整数i为止的unique bst的数量，其实就是root为0-i都选一遍,可以写几个数的结果来找找感觉。

```
n = 0: 只能为空,count[0] = 1
n = 1: 一种,count[1] = 1
n = 2: 1可以做root,2也可以做root
    count[2] = count[0] * count[1](1为root) + count[1] * count[0](2
n = 3: 1,2,3都可以做root
    count[3] = count[0] * count[2](1为root) + count[1] * count[1](2
```

可以发现的规律是对于 `count[i]`

```
for j in range(0, i):
    # 做root的也占据了一个数字,所以是0到i-1
    count[i] += count[j] * count[i-1-j]
```

Solution

```
class Solution(object):
    def numTrees(self, n):
        """
        :type n: int
        :rtype: int
        """
        if n == 0:
            return 1
        count = [0 for i in range(n+1)]
        count[0] = 1

        for i in range(1, n+1):
            for j in range(0, i):
                count[i] += count[j] * count[i-j-1]

        return count[n]
```

Reference

- unique binary search tree

Unique Binary Search Tree II

题目描述

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

For example, Given `n = 3`, your program should return all `5` unique BST's shown below.

```
   1         3     3      2      1
    \       /     /      / \      \
     3     2     1      1   3      2
    /     /       \                 \
   2     1         2                 3
```

解题方法

跟上一题的思想类似，但是这里用到递归的方法来获得具体的子树，然后对于左右两边的所有组合可能都添加到结果里

注意点

- 边界条件的处理
  - 当 `start < end` 时，正常处理
  - 当 `start == end` 时，说明只剩下一个数字可用，也就只有一种子树了
  - 当 `start > end` 时，说明就是上个recusive call里选了 `start` 或者 `end` 做root,那么左/右 子树就应该是None,返回 `None`
- python变量，每次都应该建一个新的加到结果里，不然Python会复用同一变量，最终得到的结果就不对了，就像subsets题目里一样

Solution

```
# 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 generateTrees(self, n):
        """
        :type n: int
        :rtype: List[TreeNode]
        """
        return self.generate(1, n)

    def generate(self, start, end):
        result = []
        if start > end:
            # means no possible subtrees
            # should return None to above to make the left/right ch
            result.append(None)
            return result

        for i in range(start, end+1):
            leftSubs = self.generate(start, i-1)
            rightSubs = self.generate(i+1, end)

            for l in leftSubs:
                for r in rightSubs:
                    root = TreeNode(i) # should make a root everyt
                    root.left = l
                    root.right = r
                    result.append(root)

        return result
```

Reference

Tree

Reference

- code flavor

树的遍历

reference

- code ganker

BFS

题目列表

- Binary Tree Level Order Traversal
    - 两个queue
    - 递归
- Binary Tree Level Order Traversal II
- Binary Tree ZigZag level order Traversal
- Binary Tree Right Side View

- leetcode bfs

模板

```
from collections import deque
def bfs(self, root):
        """
        :type root: TreeNode
        :rtype: List[int]
        """
        if not root:
            return []
        queue = deque()
        result = []
        queue.append(root)
        while queue:
            size = len(queue) # the size if the num of node in curr
            for i in range(size):
                cur = queue.popleft()
                if cur.left:
                    queue.append(cur.left)
                if cur.right:
                    queue.append(cur.right)
        return result
```

Reference

- amc之家

Binary Tree Right Side View

题目描述

Given a binary tree, imagine yourself standing on the right side of it, return the values of the nodes you can see ordered from top to bottom.

For example: Given the following binary tree,

```
    1            <---
  /   \
 2     3         <---
  \     \
   5     4       <---
```

You should return `[1, 3, 4]`.

解题方法

一开始我以为只要不断地优先找右边的child就可以了，其实不对，因为如果只是不断地看右子树，会错过左子树depth大于右子树的情况，比如

```
    1            <---
  /   \
 2     3         <---
  \
   5             <---
```

所以这种简单的想法并不对，应该多画几个例子来看看有什么规律和没考虑到的地方

BFS level-order

我们可以很快想到之前做过的BFS level-order的方法，这里只要总是输出最右边的数就可以了。所以还是用BFS的方法做。

Solution

```
from collections import deque
# 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 rightSideView(self, root):
        """
        :type root: TreeNode
        :rtype: List[int]
        """
        if not root:
            return []
        queue = deque()
        result = []
        queue.append(root)
        while queue:
            size = len(queue)
            for i in range(size):
                cur = queue.popleft()
                if i == size - 1:
                    result.append(cur.val)
                if cur.left:
                    queue.append(cur.left)
                if cur.right:
                    queue.append(cur.right)
        return result
```

Reference

- bfs programcreek
- geeks for geeks
- bfs + recursive
- python code

Populating Next Right Pointers in Each Node

题目描述

Given a binary tree

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

Populate each next pointer to point to its next right node. If there is no next right node, the next pointer should be set to NULL.

Initially, all next pointers are set to `NULL`.

Note:

You may only use constant extra space. You may assume that it is a perfect binary tree (ie, all leaves are at the same level, and every parent has two children). For example, Given the following perfect binary tree,

```
         1
       /  \
      2    3
     / \  / \
    4  5  6  7
```

After calling your function, the tree should look like:

```
         1 -> NULL
       /  \
      2 -> 3 -> NULL
     / \  / \
    4->5->6->7 -> NULL
```

解题方法

BFS

完全二叉树，所以只需要将level order/right side view的code稍做改动，最右边的next指向None, 其他的都指向queue里面的下一个

空间 `O(1)`

两个节点node负责换层(node=node->left), cross负责把左右子树连起来，以及右子树和corss->next的左子树连起来。

Solution

```
from collections import deque
# Definition for binary tree with next pointer.
# class TreeLinkNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
#         self.next = None

class Solution(object):
    def connect(self, root):
        """
        :type root: TreeLinkNode
        :rtype: nothing
        """
        if not root:
            return
        queue = deque()
        result = []
        queue.append(root)
        while queue:
            size = len(queue)
            for i in range(size):
                cur = queue.popleft()
                if i == size - 1:
                    cur.next = None
                else:
                    cur.next = queue[0]
                if cur.left:
                    queue.append(cur.left)
                if cur.right:
                    queue.append(cur.right)
```

space `O(1)`

- 用node变量来遍历每一层，node一直是每层的最左边的node
- 然后在每一层都将下一层的next连接起来，用一个cross一直往右来连接

```python
class Solution(object):
    def connect(self, root):
        """
        :type root: TreeLinkNode
        :rtype: nothing
        """
        if not root:
            return
        node = root
        while node:
            cross = node
            while cross:
                if cross.left:
                    cross.left.next = cross.right
                if cross.right and cross.next:
                    cross.right.next = cross.next.left
                cross = cross.next
            node = node.left
```

Reference

Populating Next Right Pointers in Each Node II

题目描述

Follow up for problem "Populating Next Right Pointers in Each Node".

What if the given tree could be any binary tree? Would your previous solution still work?

Note:

You may only use constant extra space. For example, Given the following binary tree,

```
         1
       /  \
      2    3
     / \    \
    4   5    7
```

After calling your function, the tree should look like:

```
         1 -> NULL
       /  \
      2 -> 3 -> NULL
     / \    \
    4-> 5 -> 7 -> NULL
```

解题方法

不再是完全二叉树，但是bfs level order的code依然work

Solution

```python
from collections import deque
# Definition for binary tree with next pointer.
# class TreeLinkNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
#         self.next = None

class Solution(object):
    def connect(self, root):
        """
        :type root: TreeLinkNode
        :rtype: nothing
        """
        if not root:
            return
        queue = deque()
        result = []
        queue.append(root)
        while queue:
            size = len(queue)
            for i in range(size):
                cur = queue.popleft()
                if i == size - 1:
                    cur.next = None
                else:
                    cur.next = queue[0]
                if cur.left:
                    queue.append(cur.left)
                if cur.right:
                    queue.append(cur.right)
```

O(1)

```python
from collections import deque
# Definition for binary tree with next pointer.
# class TreeLinkNode(object):
```

```
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
#         self.next = None

class Solution(object):
    def connect(self, root):
        """
        :type root: TreeLinkNode
        :rtype: nothing
        """
        if not root:
            return
        # prev level's head
        lastHead = root
        # prev level's pointer
        lastCurrent = None
        # current level's head
        curHead = None
        # current level's pointer
        cur = None
        while lastHead:
            lastCurrent = lastHead # start another level loop
            while lastCurrent:
                if lastCurrent.left:
                    if curHead is None:
                        # start from the beginning
                        curHead = lastCurrent.left
                        cur = lastCurrent.left
                    else:
                        cur.next = lastCurrent.left
                        cur = cur.next
                if lastCurrent.right:
                    if curHead is None:
                        # start from the beginning
                        curHead = lastCurrent.right
                        cur = lastCurrent.right
                    else:
                        cur.next = lastCurrent.right
                    cur = cur.next
                lastCurrent = lastCurrent.next
            lastHead = curHead
            curHead = None
```

```
public void connect(TreeLinkNode root) {
    if(root == null)
        return;

    TreeLinkNode lastHead = root;//prevous level's head
    TreeLinkNode lastCurrent = null;//previous level's pointer
    TreeLinkNode currentHead = null;//currnet level's head
    TreeLinkNode current = null;//current level's pointer

    while(lastHead!=null){
        lastCurrent = lastHead;

        while(lastCurrent!=null){
            //left child is not null
            if(lastCurrent.left!=null)    {
                if(currentHead == null){
                    currentHead = lastCurrent.left;
                    current = lastCurrent.left;
                }else{
                    current.next = lastCurrent.left;
                    current = current.next;
                }
            }

            //right child is not null
            if(lastCurrent.right!=null){
                if(currentHead == null){
                    currentHead = lastCurrent.right;
                    current = lastCurrent.right;
                }else{
                    current.next = lastCurrent.right;
                    current = current.next;
                }
            }
```

```
                lastCurrent = lastCurrent.next;
            }

            //update last head
            lastHead = currentHead;
            currentHead = null;
        }
    }
```

Reference

- program creek

Word Ladder

题目描述

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

Only one letter can be changed at a time Each intermediate word must exist in the word list For example,

Given: beginWord = `"hit"` endWord = `"cog"` wordList = `["hot","dot","dog","lot","log"]` As one shortest transformation is `"hit" -> "hot" -> "dot" -> "dog" -> "cog"`, return its length `5`.

Note:

- Return 0 if there is no such transformation sequence.
- All words have the same length.
- All words contain only lowercase alphabetic characters.

解题方法

求最短路径一般是用BFS的方法，这里就是不断的BFS搜查当前word的neighbours（在wordList里的）， 如果遍历字典里的 所有word，那么会随着字典的变大而时间复杂度太大，我们可以用26个字母去替换，就得到固定的时间复杂度。

注意点

- 一开始要将 `endWord` 加入到 `wordList` 里，这样最后达到的时候好判断
- 只要已经是 `endWord` 了就可以立刻返回，因为只要求最小的长度，BFS最先得到的必然是最小长度
- BFS将neighbours加入到queue之后要将它们从 `wordList` 中删去，不然会得到重复的loop

Solution

```python
from collections import deque
from string import ascii_lowercase

class Solution(object):
    def ladderLength(self, beginWord, endWord, wordList):
        """
        :type beginWord: str
        :type endWord: str
        :type wordList: Set[str]
        :rtype: int
        """
        if not beginWord or not endWord or not wordList:
            return
        wordList.add(endWord)

        level = 1
        queue = deque()
        queue.append((beginWord, level))
        while queue:
            curWord, curLevel = queue.popleft()
            if curWord == endWord:
                return curLevel

            removeArr = []
            ns = self.getNeighbours(curWord, wordList)
            for word in ns:
                queue.append((word, curLevel + 1))
                removeArr.append(word)

            for w in removeArr:
                wordList.remove(w)

        return 0

    def getNeighbours(self, curr, wordList):
        neighbours = []
        for i in range(len(curr)):
            for c in ascii_lowercase:
                n = curr[:i] + c + curr[i+1:]
                if n in wordList and n != curr:
                    neighbours.append(n)

        return neighbours
```

Reference

Word Ladder II

题目描述

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

Only one letter can be changed at a time Each intermediate word must exist in the word list For example,

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

Return

```
  [
    ["hit","hot","dot","dog","cog"],
    ["hit","hot","lot","log","cog"]
  ]
```

解题方法

依然是BFS, 但是这里要找到所有的最短路径，就不能直接退出，并且也不能立刻从字典里删去用过的word, 因为这一层的其他word也有可能以它为下一个neighbour, 所以要在BFS的一层node都结束后再从字典里删去 用过的数。

并且要求能够最后重建路径，所以我们用一个hashtable来存每个word的prev是什么，这样才可以重建路径。

Solution

memory limit exceeded....

```
from collections import deque
from string import ascii_lowercase
```

```
class Solution(object):
    def findLadders(self, beginWord, endWord, wordlist):
        """
        :type beginWord: str
        :type endWord: str
        :type wordlist: Set[str]
        :rtype: List[List[int]]
        """
        if not beginWord or not endWord or not wordlist:
            return
        wordlist.add(endWord)

        level = 1
        curLevelNum = 1
        nextLevelNum = 0

        prevHash = {}
        prevHash[beginWord] = None

        queue = deque()
        queue.append((beginWord, level))
        wordlist.remove(beginWord)

        found = False
        while queue:
            curWord, curLevel = queue.popleft()
            curLevelNum -= 1

            if curWord == endWord:
                found = True

            removeArr = []
            ns = self.getNeighbours(curWord, wordlist)
            for word in ns:
                queue.append((word, curLevel + 1))
                nextLevelNum += 1
                removeArr.append(word)
                if word in prevHash:
                    if curWord not in prevHash[word]:
                        prevHash[word].append(curWord)
                else:
                    prevHash[word] = [curWord]

            if curLevelNum == 0:
                for w in removeArr:
                    wordlist.remove(w)
                curLevelNum = nextLevelNum
                nextLevelNum = 0

                if found:
                    results = []
                    self.getPathes(results, beginWord, endWord, pre
                    return results

        return [[]]

    def getNeighbours(self, curr, wordList):
        neighbours = []
        for i in range(len(curr)):
            for c in ascii_lowercase:
                n = curr[:i] + c + curr[i+1:]
                if n in wordList and n != curr:
                    neighbours.append(n)

        return neighbours

    def getPathes(self, results, beginWord, endWord, prevHash):

        queue = deque()
        queue.append((endWord, [endWord]))

        while queue:
            cur, path = queue.popleft()
            if cur == beginWord:
                results.append(path[::-1])
            else:
                for pre in prevHash[cur]:
                    path.append(pre)
                    newPath = list(path)
                    queue.append((pre, newPath))
                    path.pop()

    // another方法
    def getPathes(self, results, beginWord, curr, prevHash, curList
        if curr == beginWord:
            newList = list(curList)
            newList.reverse()
            results.append(newList)
        else:
            for parent in prevHash[curr]:
                curList.append(parent)
                self.getPathes(results, beginWord, parent, prevHash
                curList.pop()
```

优化的可以过code

```
def buildpath(path, word):
            if len(prevMap[word])==0:
                path.append(word); currPath=path[:]
                currPath.reverse(); result.append(currPath)
                path.pop();
                return
            path.append(word)
            for iter in prevMap[word]:
                buildpath(path, iter)
            path.pop()

        result=[]
        prevMap={}
        length=len(start)
        for i in dict:
            prevMap[i]=[]
        candidates=[set(),set()]; current=0; previous=1
        candidates[current].add(start)
        while True:
            current, previous=previous, current
            for i in candidates[previous]: dict.remove(i)
            candidates[current].clear()
            for word in candidates[previous]:
                for i in range(length):
                    part1=word[:i]; part2=word[i+1:]
                    for j in 'abcdefghijklmnopqrstuvwxyz':
                        if word[i]!=j:
                            nextword=part1+j+part2
                            if nextword in dict:
                                prevMap[nextword].append(word)
                                candidates[current].add(nextword)
            if len(candidates[current])==0: return result
            if end in candidates[current]: break
        buildpath([], end)
        return result
```

Reference

DFS

题目列表

问题描述

遇到要求所有组合，可能，排列等解集的问题，一般都是用DFS/BFS + backtracking来做

Reference

- DFS总结
- DFS有关的题目

Path Sum

题目描述

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

For example: Given the below binary tree and `sum = 22`,

```
              5
             / \
            4   8
           /   / \
          11  13  4
         /  \      \
        7    2      1
```

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

解题方法

DFS

DFS, 在stack里放一个tuple `(node, curSum)`

- node就是当前的treeNode
- curSum是到这个node为止的路径的sum

如果node是叶子节点，并且 `curSum == sum` 的时候，就说明有这样一条路径

recursion

recursion的方法就是将sum减去当前node的value,然后不断向下传递

Solution

```
# 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 hasPathSum(self, root, sum):
        """
        :type root: TreeNode
        :type sum: int
        :rtype: bool
        """
        if not root:
            return False

        stack = []
        stack.append((root, root.val))
        while stack:
            curNode, curSum = stack.pop()
            if not curNode.left and not curNode.right and curSum ==
                return True
            if curNode.left:
                stack.append((curNode.left, curSum + curNode.left.v
            if curNode.right:
                stack.append((curNode.right, curSum + curNode.right

        return False
```

recursion

```
# 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 hasPathSum(self, root, sum):
        """
        :type root: TreeNode
        :type sum: int
        :rtype: bool
        """
        if not root:
            return False

        if not root.left and not root.right and root.val == sum:
            return True

        return self.hasPathSum(root.left, sum-root.val) or self.has
```

Reference

Path Sum II

题目描述

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

For example: Given the below binary tree and `sum = 22`,

```
          5
         / \
        4   8
       /   / \
      11  13  4
     /  \    / \
    7    2  5   1
```

return

```
[
   [5,4,11,2],
   [5,8,4,5]
]
```

解题方法

和 `Path Sum I` 一样的做法，只不过在这里stack的tuple再加上目前为止的path，如果找到了就将这个path加到结果里

Solution

```
# 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 pathSum(self, root, sum):
        """
        :type root: TreeNode
        :type sum: int
        :rtype: List[List[int]]
        """
        if not root:
            return []

        result = []
        stack = []
        stack.append((root, root.val, [root.val]))
        while stack:
            curNode, curSum, curPath = stack.pop()
            if not curNode.left and not curNode.right and curSum ==
                result.append(curPath)
            if curNode.left:
                stack.append((curNode.left, curSum + curNode.left.v
            if curNode.right:
                stack.append((curNode.right, curSum + curNode.right

        return result
```

Reference

Sum Root to Leaf Numbers

题目描述

Given a binary tree containing digits from `0-9` only, each root-to-leaf path could represent a number.

An example is the root-to-leaf path `1->2->3` which represents the number `123`.

Find the total sum of all root-to-leaf numbers.

For example,

```
    1
   / \
  2   3
```

The root-to-leaf path `1->2` represents the number `12`. The root-to-leaf path `1->3` represents the number `13`.

Return the `sum = 12 + 13 = 25`

解题方法

DFS

和 `path sum` 一样的解法，只不过这里记录的是value组成的string, 当遇到leaf的时候再转换成Int加到结果里

Solution

```
# 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 sumNumbers(self, root):
        """
        :type root: TreeNode
        :rtype: int
        """
        if not root:
            return 0

        stack = []
        result = 0
        stack.append((root, str(root.val)))
        while stack:
            curNode, curSum = stack.pop()
            if not curNode.left and not curNode.right:
                result += int(curSum)
            if curNode.left:
                stack.append((curNode.left, curSum + str(curNode.le
            if curNode.right:
                stack.append((curNode.right, curSum + str(curNode.r
        return result
```

Reference

Word Break II

题目描述

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"]` .

解题方法

这一题与1一样，也可以用DP的方法来做，但是在要求所有解的问题中，DP与DFS的效果其实差不多。

如果用DP的话，就要记录下i可以由之前的哪一个index break而来，最后重建的时候code也比较复杂。

我们可以用DFS + backtracking的方法，并且减去一些不必要的计算。

注意点

- 在DFS中加入先判断是否能够word break的逻辑，这样可以避免不必要的计算

Solution

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

        self.helper(results, s, 0, wordDict, [])
        return results


    def helper(self, results, s, start, wordDict, curList):
        if self.check(s[start:], wordDict):
            if start >= len(s):
                newList = " ".join(curList)
                results.append(newList)
                return

            for i in range(start+1, len(s)+1):
                if s[start:i] in wordDict:
                    curList.append(s[start:i])
                    self.helper(results, s, i, wordDict, curList)
                    curList.pop()

    def check(self, s, wordDict):
        """
        :type s: str
        :type wordDict: Set[str]
        :rtype: bool
        """
        if not wordDict:
            return False

        length = len(s)

        canBreak = [False for i in range(length+1)]
        canBreak[0] = True

        for i in range(1, length+1):
            for j in range(i):
                if canBreak[i]:
                    continue
                if (s[j:i] in wordDict) and canBreak[j]:
                    canBreak[i] = True

        return canBreak[length]
```

Reference

- word break 2

Copy Graph

题目描述

Clone an undirected graph.

Each node in the graph contains a `label` and a list of its `neighbors` .

解题方法

这种deep copy的题目一般的做法就是要用到hashmap，先存下old/new的pair，然后在 遍历的时候check是否已经copy过，如果copy过就从hashmap中取出，如果没有就新建一个。

遍历graph的过程是bfs, 用一个queue来implement。

Solution

```python
from collections import deque
# Definition for a undirected graph node
# class UndirectedGraphNode(object):
#     def __init__(self, x):
#         self.label = x
#         self.neighbors = []

class Solution(object):
    def cloneGraph(self, node):
        """
        :type node: UndirectedGraphNode
        :rtype: UndirectedGraphNode
        """
        if not node:
            return None

        copiedMap = {}
        queue = deque()
        queue.append(node)
        newNode = UndirectedGraphNode(node.label)
        copiedMap[node] = newNode

        while queue:
            cur = queue.popleft()
            copiedCur = copiedMap[cur]
            for neighbor in cur.neighbors:
                if neighbor in copiedMap:
                    # if already copied, means it should already be
                    # no need to add into queue again
                    copiedN = copiedMap[neighbor]
                    copiedCur.neighbors.append(copiedN)
                else:
                    # first time encounter this node, copy it and a
                    newN = UndirectedGraphNode(neighbor.label)
                    copiedMap[neighbor] = newN
                    queue.append(neighbor)
                    copiedCur.neighbors.append(newN)
        return newNode
```

Reference

Binary Search

二分查找法的O(log n)让它成为十分高效的算法。不过它的缺陷却也是那么明显的。就在它的限定之上：

必须 `sorted` ，我们很难保证我们的数组都是有序的。当然可以在构建数组的时候进行排序，可是又落到了第二个瓶颈上：它必须是数组。 数组读取效率是O(1)，可是它的插入和删除某个元素的效率却是 `O(n)` 。因而导致构建有序数组变成低效的事情。

解决这些缺陷问题更好的方法应该是使用BST了，最好自然是自平衡二叉查找树了，自能高效的（O(n log n)）构建有序元素集合，又能如同二分查找法一样快速（O(log n)）的搜寻目标数。

Reference

- 心随风飞
- code ganker
- 二分查找法相关

Binary Search

题目列表

- leetcode

问题描述

`T(n) = T(n/2) + O(1) = O(log n)`

什么时候使用binary search

面试中如果需要优化 `O(n)` 的时间复杂度，一般只能是 `O(logn)的二分法`

常见问题

binary search可以有多种写法，如果每次都写的不一样，就会造成对于结束条件，指针变化的混乱，所以我们要找到一种固定的模板，通过修改模板来解决题目。这里我选择了九章算法提供的模板。

四点要素:

1. `start + 1 < end` 这样就不用考虑两个指针的前后，最后结束时一定是相邻的
2. `start + (end - start) / 2` 虽然对python来说不重要，但是对于Java等可以防止溢出
3. `A[mid] ==, <, >` 这个会根据题目的不同来调整
4. `A[start] A[end] ? target`

```python
class Solution:
    # @param nums: The integer array
    # @param target: Target number to find
    # @return the first position of target in nums, position start
    def binarySearch(self, nums, target):
        if len(nums) == 0:
            return -1

        start, end = 0, len(nums) - 1
        while start + 1 < end:
            mid = (start + end) / 2
            if nums[mid] < target:
                start = mid
            else:
                end = mid

        if nums[start] == target:
            return start
        if nums[end] == target:
            return end
        return -1
```

最终状态都是要将start和end位置的值与target来比较，根据题意得到想要的结果

题型

- 第一类
- 第二类
- 第三类
- 第四类

Reference

- 九章算法
- code ganker

第一类

- search insert position
- search for a range
- isBadVersion
- Closest Binary Search Tree Value
- Find Peak Element
- Median of Two Sorted Arrays

这两道题目是考察基本用法，但是因为题目没有告诉你是否有重复的值，所以并不能mid = target就停下来，而是要继续查找

Search insert position

基本二分法的思想

- `mid >= target` 时 `end = mid` 继续搜
- `mid < target` 是 `start = mid` 继续搜
- 最终用start和end来比较，如果有 `>=` 的，就插入在这里，如果都小于，说明应该插入在list的结尾

```
class Solution(object):
    def searchInsert(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: int
        """
        if len(nums) == 0:
            return -1

        start, end = 0, len(nums) - 1
        while start + 1 < end:
            mid = start + (end - start) / 2
            if nums[mid] >= target:
                #can not stop when equal, should continue search
                end = mid
            else:
                #start = mid + 1 is also fine
                start = mid

        #start, end, compare them with target, to find the right po
        if nums[start] >= target:
            return start
        if nums[end] >= target:
            return end
        #reach the end of array
        return len(nums)
```

Search for a range

找左右边界的关键是当mid == target时移动start还是移动end

```
class Solution(object):
    def searchRange(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: List[int]
        """
        left = -1
        right = -1
        if len(nums) == 0:
            return [left, right]

        start, end = 0, len(nums) - 1

        #search left
        #找左边界是一个不断将搜索区间的end不断左移的过程，所以当mid >= targ
        while start + 1 < end:
            mid = start + (end - start) / 2
            if nums[mid] >= target:
                end = mid
            else:
                start = mid

        if nums[start] == target:
            left = start
        elif nums[end] == target:
            left = end
        else:
            #not found
            return [-1,-1]

        start, end = 0, len(nums) - 1
        #search right
        #找右边界是一个不断将搜索区间start右移的过程，当mid <= target时，将
        while start + 1 < end:
            mid = start + (end-start)/2
            if nums[mid] <= target:
                start = mid
            else:
                end = mid

        if nums[end] == target:
            right = end
        elif nums[start] == target:
            right = start
        else:
            #not found
            return [-1, -1]

        return [left, right]
```

isBadVersion

就是binary search

Closest Binary Search Tree Value

一开始看没找到感觉，还以为left <= target, right >= target是我们要找的值。用一个例子来走一遍找找感觉。

对于每一个root,先求得当前的diff, 如果target比root大，说明root左边的值的diff肯定比当前的大，可以舍去左边一半。 右边同理。 就相当于一个binary search.

- refrence

```
class Solution(object):
    def closestValue(self, root, target):
        """
        :type root: TreeNode
        :type target: float
        :rtype: int
        """
        if not root:
            return None

        result, diff = self.helper(root, target)
        return result

    def helper(self, root, target):
        result = root.val
        diff = abs(root.val - target)

        if target >= root.val:
            if root.right:
                rightResult, rightDiff = self.helper(root.right, ta
                if diff > rightDiff:
                    result = rightResult
                    diff = rightDiff
        else:
            if root.left:
                leftResult, leftDiff = self.helper(root.left, targe
                if diff > leftDiff:
                    result = leftResult
                    diff = leftDiff
        return result, diff
```

recursion解法可以转换为while loop，更高效

```
class Solution(object):
    def closestValue(self, root, target):
        """
        :type root: TreeNode
        :type target: float
        :rtype: int
        """
        if not root:
            return None

        result = None
        minDiff = sys.maxint
        while root:
            diff = abs(root.val - target)
            if diff < minDiff:
                result = root.val
                minDiff = diff

            if diff == 0:
                break

            if target >= root.val:
                root = root.right
            else:
                root = root.left


        return result
```

Find Peak Element

link

其实也就是二分法，如果Mid不是peak的话，那么大于它的那一边一定有peak,以为如果一直递增的话，到了边界也一定会有一个peak. 所以每次可以舍去二分之一。

```
class Solution(object):
    def findPeakElement(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        length = len(nums)
        if length == 0:
            return

        if length == 1:
            return 0

        start, end = 0, length - 1
        while start + 1 < end:
            mid = start + (end - start) / 2
            if nums[mid] > nums[mid - 1] and nums[mid] > nums[mid -
                return mid
            elif nums[mid] < nums[mid - 1]:
                end = mid
            else:
                start = mid

        if nums[start] > nums[end]:
            return start
        else:
            return end
```

Median of Two Sorted Arrays

方法1 count， brute force

两个指针，一边比较一遍计数

方法2，每次比较两个array的中间值

利用**find kth Number**的思想

Reference

- 九章

第二类

与数学相关

- sqrt(X)
- pow(x, n)
- fastPower

sqrt(x)

这一题也是考察基本的，要注意的是，判断相等的条件是 `mid ** 2 <= target and (mid+1) ** 2 > target`

```python
class Solution(object):
    def mySqrt(self, x):
        """
        :type x: int
        :rtype: int
        """
        if x == 0 or x == 1:
            return x

        start, end = 0, x
        while start + 1 < end:
            mid = start + (end - start) / 2
            sqrt = mid ** 2
            sqrtPlus = (mid + 1) ** 2
            if sqrt <= x and sqrtPlus > x:
                return mid
            elif sqrt < x:
                start = mid
            else:
                end = mid

        if start ** 2 >= x:
            return start
        else:
            return end
```

pow(x,n)

这一题主要是递归类似二分法，每次递归求n/2的结果，再乘以自己，如果是奇数再乘以一个x 注意点

- n 是负数的情况
- n 是奇数的情况

```
class Solution(object):
    def myPow(self, x, n):
        """
        :type x: float
        :type n: int
        :rtype: float
        """
        if x == 0:
            return 0
        if n == 0:
            return 1

        if n < 0:
            ret = self.myPow(x, -n)
            return 1 / ret

        ret = self.myPow(x, n / 2)
        ret *= ret

        if n % 2 != 0:
            ret *= x

        return ret
```

第三类

在二维上运用binary search

- Search a 2d Matrix
- Search a 2d Matrix II

Search a 2D Matrix

2d matrix的二维二分查找，其实可以将二维的坐标变为一维，就又变成了普通的二分查找

```
class Solution(object):
    def searchMatrix(self, matrix, target):
        """
        :type matrix: List[List[int]]
        :type target: int
        :rtype: bool
        """
        row = len(matrix)
        col = len(matrix[0])

        start, end = 0, col * row - 1

        while start + 1 < end:
            mid = start + (end - start) / 2
            x = mid / col
            y = mid % col
            if matrix[x][y] == target:
                return True
            elif matrix[x][y] > target:
                end = mid
            else:
                start = mid

        startX = start / col
        startY = start % col
        if matrix[startX][startY] == target:
            return True
        endX = end / col
        endY = end % col
        if matrix[endX][endY] == target:
            return True

        return False
```

Search a 2D Matrix II

这个matrix的排序比较特别，是从左到右递增，从上到下递增 可以从第一行的最右边开始查找，如果大于可以排除左边的，如果小于可以排除下面的

```
class Solution(object):
    def searchMatrix(self, matrix, target):
        """
        :type matrix: List[List[int]]
        :type target: int
        :rtype: bool
        """
        row = len(matrix)
        col = len(matrix[0])

        x = 0
        y = col - 1
        while x < row and y >= 0:
            if matrix[x][y] == target:
                return True
            elif matrix[x][y] > target:
                y -= 1
            else:
                x += 1

        return False
```

第四类

部分sorted的array或者rotated的array

- search in rotated sorted array
- search in rotated sorted array II(duplicate allowed)
- find min in rotated sorted array
- find min in rotated sorted array II(duplicate allowed)

虽然rorate,但是只rotate了一次，如果二分必然有一半是sorted的有一半不是。通过判断中心点和边缘，比如mid和end,就可以知道哪边有序，然后就能够知道往左右哪边走是对的。

对于有duplicate的rotated数组，比如 `[3,1,2,3,3,3,3]` ,中心和边缘都是3，无法判断往哪边走，所以只能将边界start或者end移动一步，这样就可能没法切去一半，复杂度就是 `o(n)`

注意点

- 应该先判断是否rotated, 如果nums[0] < nums[length-1]说明没有rotate
- 判断array的长度: 0, 1, 2

Search In Rotated Sorted Array

```
class Solution(object):
    def search(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: int
        """
        length = len(nums)
        if length == 0:
            return -1

        start, end = 0, length - 1
        while start + 1 < end:
            mid = start + (end - start) / 2
            if target == nums[mid]:
                return mid
            if nums[start] < nums[mid]:
                #left is sorted
                if nums[start] <= target and target < nums[mid]:
                    end = mid
                else:
                    start = mid
            else:
                #right is sorted
                if target > nums[mid] and target <= nums[end]:
                    start = mid
                else:
                    end = mid

        if nums[start] == target:
            return start
        if nums[end] == target:
            return end
        return -1
```

Search Rotated Array II

对于有duplicate的rotated数组，比如 `[3,1,2,3,3,3,3]` ,中心和边缘都是3，无法判断往哪边走

会影响时间复杂度，加上下面这个判断，只能将start挪一步再判断，worst case 是 `O(n)`

因为这一题是要找到target这个值，所以我们判断哪一边是sorted的，这里我们一直采取比较left和mid的办法，如果left和Mid相等，我们就将left + 1,直到能够判断哪一边是sorted的，将这个范围与target比较，就能够知道该往那边走。

```
elif nums[start] == nums[mid]:
                start += 1
```

Find Minimum in Rotated Sorted Array

如果是sorted： nums[start] < nums[end]应该直接返回左边

```
class Solution(object):
    def findMin(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        length = len(nums)
        #length check
        if length == 0:
            return -1

        if length == 1:
            return nums[0]

        if length == 2:
            return min(nums)

        start, end = 0, length - 1

        while start + 1 < end:
            #rotate check
            if nums[start] < nums[end]:
                return nums[start]
            mid = start + (end - start) / 2
            if nums[mid] < nums[mid-1] and nums[mid] < nums[mid+1]
                return nums[mid]
            if nums[start] < nums[mid]:
                #left is sorted
                    start = mid
            else:
                #right is sorted
                    end = mid

        if nums[start] < nums[end]:
            return nums[start]
        return nums[end]
```

Find Minimum in Rotated Sorted Array II

这一题不同于search in rotated sorted array ii, 因为这里不是要找一个target,而是要找minimum。

- 如果已经sorted,直接返回左边，这个应该每次都check,因为在相等drop掉一些值时，有可能已经变成了sorted的array

```python
class Solution(object):
    def findMin(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        length = len(nums)
        #length check
        if length == 0:
            return -1

        if length == 1:
            return nums[0]

        if length == 2:
            return min(nums)

        start, end = 0, length - 1

        while start + 1 < end:
            #rotate check
            if nums[start] < nums[end]:
                return nums[start]
            mid = start + (end - start) / 2
            if nums[mid] < nums[mid-1] and nums[mid] < nums[mid+1]:
                return nums[mid]
            if nums[start] < nums[mid]:
                #left is sorted
                start = mid
            #add this condition
            elif nums[start] == nums[mid]:
                start += 1
            else:
                #right is sorted
                end = mid

        if nums[start] < nums[end]:
            return nums[start]
        return nums[end]
```

Find Kth Number

题目描述

2个array，找第k大的元素

题目列表

- median of two sorted array
- find kth largest elemtn of array

Median of two sorted array

借用find kth的思想， `偶数` 个就要将中间两个平均， `奇数` 就返回中间那个

- 每次在nums1, nums2两个array中各取k/2位置a和b,因为**index是0-based,**所以**index为k/2-1**
- 如果有任意一个array的长度小于k/2,那么就可以省去另一个array的前k/2个
- 如果a < b, 那么可以丢掉nums1的前K/2, 反之也是
- 如果a == b,不可以直接返回**a**，因为也要确定k的奇偶，所以将它和a >= b一起处理

注意点

- 这里a == b的情况
    - 如果k为偶数的话，这种情况下可以return
    - 如果k为奇数的话，这里的a和b都不是第k的元素，而应该是下一个
    - 这里为了code简单，直接传到下一个call中解决

```
class Solution(object):
    def findMedianSortedArrays(self, nums1, nums2):
        """
        :type nums1: List[int]
        :type nums2: List[int]
        :rtype: float
        """
        n = len(nums1) + len(nums2)
        if n % 2 == 1:
            return self.findKth(nums1, nums2, n / 2 + 1)
        else:
            #偶数个, 要取两个值的平均值
            smaller = self.findKth(nums1, nums2, n / 2)
            bigger = self.findKth(nums1, nums2, n / 2 + 1)
            return (smaller + bigger) / 2.0

    def findKth(self, nums1, nums2, k):
        if len(nums1) == 0:
            return nums2[k-1]

        if len(nums2) == 0:
            return nums1[k-1]

        if k == 1:
            return min(nums1[0], nums2[0])

        if len(nums1) < k / 2:
            a = None
        else:
            a = nums1[k/2 - 1]

        if len(nums2) < k / 2:
            b = None
        else:
            b = nums2[k/2 - 1]


        if not a:
            return self.findKth(nums1, nums2[k/2:], k - k/2)
        if not b:
            return self.findKth(nums1[k/2:], nums2, k - k/2)
        if a < b:
            return self.findKth(nums1[k/2:], nums2, k - k/2)
        else:
            return self.findKth(nums1, nums2[k/2:], k - k/2)
```

Longest Increasing Subsequence

题目描述

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

For example, Given `[10, 9, 2, 5, 3, 7, 101, 18]` , The longest increasing subsequence is `[2, 3, 7, 101]` , therefore the length is `4` . Note that there may be more than one LIS combination, it is only necessary for you to return the length.

Your algorithm should run in `O(n^2)` complexity.

Follow up:

Could you improve it to `O(n log n)` time complexity?

解题方法

DP `O(n^2)`

一开始我想用dp[i]来表示在[:i+1]里面的最长LIS, 但是这样很难就要像global, lobal一样建两个array, local用来表示以i为终点的LIS的长度，这样最后只要找出array中的最大值就可以了.

function

```
for k in range(0, i):
    if nums[i] > nums[k] and dp[i] < dp[k] + 1:
        dp[i] = dp[k] + 1
```

这种方法对于每个i都要遍历0~i-1， 所以需要 `O(n^2)` 的复杂度

二分法

实际上，在对于i往前查找的过程中，只要比较nums[i]和之前LIS的最小结尾就可以， 比如

```
                0  1  2  3  4  5  6  7
nums[]:    2  5  6  2  3  4  7  4
dp[]:         1  2  3  1  2  3  4  3
```

对于nums[6], 前面最大的LIS长度为3，并且有两个，但是我们只需要比较小的那个结尾，就可以确定 nums[6]是否可以接到后面。 如果不行的话，就再找小1个长度的LIS的最小结尾比较。

所以我们要建一个新的array, last_min, last_min[i]表示目前长度为i的LIS的最小结尾。 并且用一个variable `curMax` 记录目前最长的LIS， 这样便于比较。 而在查找这个i的过程可以使用二分法， 所以时间复杂度可以降到 `O(nlogn)`

Solution

`O(n^2)`

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

        dp = [1 for i in range(len(nums))]
        for i in range(1, len(nums)):
            for j in range(0, i):
                if nums[i] > nums[j] and dp[i] < dp[j] + 1:
                    dp[i] = dp[j] + 1

        return max(dp)
```

`O(nlogn)`

Reference

- Longest Increasing Subsequence详细解释

Smallest Rectangle Enclosing Black Pixels

题目描述

An image is represented by a binary matrix with 0 as a white pixel and 1 as a black pixel. The black pixels are connected, i.e., there is only one black region. Pixels are connected horizontally and vertically. Given the location `(x, y)` of one of the black pixels, return the area of the smallest (axis-aligned) rectangle that encloses all black pixels.

For example, given the following image:

```
[
  "0010",
  "0110",
  "0100"
]
```

and `x = 0, y = 2` , Return `6` .

解题方法

根据已有的black pixel的左边，分别找到上下左右含有1的边界，利用binary search查找。

在discuss里有很pythonic的写法，可以学习一下。 这里我根据我记忆的binary search的模板进行了修改。 基本上就是利用search for a range的写法。

Solution

```
class Solution(object):
    def minArea(self, image, x, y):
        """
        :type image: List[List[str]]
        :type x: int
        :type y: int
        :rtype: int
        """
        if not image:
            return 0

        top = self.searchTop(image, 0, x)
        bottom = self.searchBottom(image, x, len(image) - 1)
        left = self.searchLeft(image, 0, y)
        right = self.searchRight(image, y, len(image[0]) - 1)
        return (right - left + 1) * (bottom - top + 1)

    def searchTop(self, image, start, end):
        while start + 1 < end:
            mid = start + (end - start) / 2
            if ("1" in image[mid]) == True:
                end = mid
            else:
                start = mid
        if ("1" in image[start]) == True:
            return start
        elif ("1" in image[end]) == True:
            return end
        return end

    def searchBottom(self, image, start, end):
        while start + 1 < end:
            mid = start + (end - start) / 2
            if ("1" in image[mid]) == True:
                start = mid
            else:
                end = mid
        if ("1" in image[end]) == True:
            return end
        elif ("1" in image[start]) == True:
            return start
        return start

    def searchLeft(self, image, start, end):
        while start + 1 < end:
            mid = start + (end - start) / 2
            if any(image[k][mid] == "1" for k in range(len(image)))
                end = mid
            else:
                start = mid
        if any(image[k][start] == "1" for k in range(len(image))) =
            return start
        elif any(image[k][start] == "1" for k in range(len(image)))
            return end
        return end

    def searchRight(self, image, start, end):
        while start + 1 < end:
            mid = start + (end - start) / 2
            if any(image[k][mid] == "1" for k in range(len(image)))
                start = mid
            else:
                end = mid
        if any(image[k][end] == "1" for k in range(len(image))) ==
            return end
        elif any(image[k][start] == "1"for k in range(len(image)))
            return start
        return start
```

Reference

- leetcode discuss

非基于比较排序

基于比较的排序算法是不能突破O(NlogN)的。简单证明如下：

N个数有N!个可能的排列情况，也就是说基于比较的排序算法的判定树有N!个叶子结点，比较次数至少为log(N!)=O(NlogN)(斯特林公式)。

非基于比较的排序：

- 计数排序（counting sort)
- 桶排序 （bucket sort)
- 基数排序（radix sort）

计数排序

如果输入的元素的n个0到k之间的证书时，时间复杂度是 `O(n+k)` ，所以不适合排序数据范围很大的数组，需要大量时间和内存。

1. 找出待排序的数组中最大和最小的元素
2. 统计数组中每个值为i的元素出现的次数，存入数组C的第i项
3. 对所有的计数累加（从C中的第一个元素开始，每一项和前一项相加）
4. 反向填充目标数组：将每个元素i放在新数组的第C(i)项，每放一个元素就将C(i)减去1

当k不是很大时，这是一个很有效的线性排序算法。更重要的是，它是一种稳定排序算法，即排序后的相同值的元素原有的相对位置不会发生改变(表现在Order上)，这是计数排序很重要的一个性质，就是根据这个性质，我们才能把它应用到基数排序。

animation

桶排序

假设有一组长度为N的待排关键字序列K[1....n]。

- 首先将这个序列划分成M个的子区间(桶) 。
- 然后基于某种映射函数 ，将待排序列的关键字k映射到第i个桶中(即桶数组B的

下标 i)，那么该关键字k就作为B[i]中的元素(每个桶B[i]都是一组大小为N/M的序列)。
- 接着对每个桶B[i]中的所有元素进行比较排序(可以使用快排)。
- 然后依次枚举输出B[0]....B[M]中的全部内容即是一个有序序列。

例子：假如待排序列K= {49、 38 、 35、 97 、 76、 73 、 27、 49 }。这些数据全部在1—100之间。因此我们定制10个桶，然后确定映射函数f(k)=k/10。则第一个关键字49将定位到第4个桶中(49/10=4)。依次将所有关键字全部堆入桶中，并在每个非空的桶中进行快速排序。

这里每个桶表示一个10的区间，如果第1个桶，就表示 `1-10` 这个区间。

对N个关键字进行桶排序的时间复杂度分为两个部分：

1. 循环计算每个关键字的桶映射函数，这个时间复杂度是 `O(N)` 。
2. 利用先进的比较排序算法对每个桶内的所有数据进行排序，其时间复杂度为 `∑ O(Ni*logNi)` 。其中Ni 为第i个桶的数据量。

很显然，第(2)部分是桶排序性能好坏的决定因素。尽量减少桶内数据的数量是提高效率的唯一办法(因为基于比较排序的最好平均时间复杂度只能达到O(N*logN)了)。因此，我们需要尽量做到下面两点：(1) 映射函数f(k)能够将N个数据平均的分配到M个桶中，这样每个桶就有[N/M]个数据量。 (2) 尽量的增大桶的数量。极限情况下每个桶只能得到一个数据，这样就完全避开了桶内数据的“比较”排序操作。 当然，做到这一点很不容易，数据量巨大的情况下，f(k)函数会使得桶集合的数量巨大，空间浪费严重。这就是一个时间代价和空间代价的权衡问题了。

当N=M时，即极限情况下每个桶只有一个数据时。桶排序的最好效率能够达到O(N)。

基数排序 radix sort

是在bucket sort上的拓展，基数排序基于多关键字排序的思想，即把一个逻辑关键字拆分成多个关键字。

比如字符串 `“abcd” “aesc” "dwsc" "rews"` 就可以把每个字符看成一个关键字。另外还有整数 `425、321、235、432` 也可以每个位上的数字为一个关键字。

基数排序的思想就是将待排数据中的每组关键字依次进行桶分配。比如下面的待排序列： `278、109、063、930、589、184、505、269、008、083` 我们将每个数值的个位，十位，百位分成三个关键字： `278 -> k1(个位)=8 ，k2(十位)=7 ，k3=(百位)=2` 。 然后从最低位个位开始(从最次关键字开始)，对所有数据的 `k1` 关键字进行桶分配(因为，每个数字都是 `0-9` 的，因此桶大小为 `10`)，再依次输出桶中的数据得到下面的序列。 `930、063、083、184、505、278、008、109、589、269` 再对上面的序列接着进行针对k2的桶分配，输出序列为： `505、008、109、930、063、269、278、083、184、589` 最后针对k3的桶分配，输出序列为： `008、063、083、109、184、269、278、505、589、930`

基数排序有两种实现方式：

- 第一种叫做最高位优先（MSD），即先按最高位排成若干子序列，再对每个子序列按次高位排。
- 第二种叫做最低位优先（LSD），这种方式不必分成子序列，每次排序全体元素都参与。最低位可以优先这样进行，不通过比较，而是通过“分配”和“收集”。 由于不需要分堆并对每堆单独排序，LSD方法往往比MSD简单而开销小。

性能分析

很明显，基数排序的性能比桶排序要略差。每一次关键字的桶分配都需要O(N)的时间复杂度，而且分配之后得到新的关键字序列又需要O(N(的时间复杂度。假如待排数据可以分为d个关键字，则基数排序的时间复杂度将是O(d*2N),当然d要远远小于N，因此基本上还是线性级别的。

基数排序的空间复杂度为O(N+M)，其中M为桶的数量。一般来说N>>M，因此额外空间需要大概N个左右。

Wiggle Sort

题目描述

Given an unsorted array nums, reorder it in-place such that `nums[0] <= nums[1] >= nums[2] <= nums[3]....`

For example, given `nums = [3, 5, 2, 1, 6, 4]`, one possible answer is `[1, 6, 2, 5, 3, 4]`.

解题方法

先sort的方法

遇到这种问题，如果没有思路就先sort一下，sort之后后面的数肯定比前面的大，所以<=的条件都满足。 而>=的条件可以通过将两个元素swap一下来满足。

不需要sort的方法

其实并不需要sort, 直接遍历，在遍历的过程中根据Index的奇偶，和相邻两个数的大小比较，来决定是否swap

Solution

先sort的方法

`O(nlogn`

```
class Solution(object):
    def wiggleSort(self, nums):
        """
        :type nums: List[int]
        :rtype: void Do not return anything, modify nums in-place i
        """
        if not nums:
            return
        nums.sort()
        length = len(nums)
        p = 1
        while p < length:
            if p + 1 >= length:
                break
            tmp = nums[p+1]
            nums[p+1] = nums[p]
            nums[p] = tmp
            p += 2
```

不需要**sort**, 直接**O(n)**的做法

```
class Solution(object):
    def wiggleSort(self, nums):
        """
        :type nums: List[int]
        :rtype: void Do not return anything, modify nums in-place i
        """
        if not nums:
            return
        length = len(nums)
        p = 0
        while p < length:
            if p + 1 >= length:
                break
            if p % 2 == 1:
                if nums[p] < nums[p+1]:
                    tmp = nums[p+1]
                    nums[p+1] = nums[p]
                    nums[p] = tmp
            else:
                if nums[p] > nums[p+1]:
                    tmp = nums[p+1]
                    nums[p+1] = nums[p]
                    nums[p] = tmp
            p += 1
```

Reference

- google面经
- greedy算法

Sort Colors

题目描述

Given an array with n objects colored red, white or blue, sort them so that objects of the same color are adjacent, with the colors in the order red, white and blue.

Here, we will use the integers 0, 1, and 2 to represent the color red, white, and blue respectively.

解题方法

naive 方法

- 遍历一遍，记录下各自的数量
- 再遍历一遍，设value

quick sort的partition方法

类似qucik sort的partition方法，这里的pivot就是1

Solution

```
class Solution(object):
    def sortColors(self, nums):
        """
        :type nums: List[int]
        :rtype: void Do not return anything, modify nums in-place i
        """
        def swap(p1, p2):
            tmp = nums[p2]
            nums[p2] = nums[p1]
            nums[p1] = tmp

        length = len(nums)
        if length == 0:
            return
        left = 0
        right = length - 1
        p = 0
        while p <= right:
            if nums[p] < 1:
                # the value at left must be equal or smaller than 1
                swap(left, p)
                p += 1
                left += 1
            elif nums[p] == 1:
                p += 1
            else:
                # don't increment p here, cause you don't know the
                swap(right, p)
                right -= 1
```

Reference

Sort List

题目描述

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

解题方法

Solution

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

class Solution(object):
    def sortList(self, head):
        """
        :type head: ListNode
        :rtype: ListNode
        """
        if not head or not head.next:
            return head
        p1 = head
        p2 = head
        while p2.next and p2.next.next:
            p1 = p1.next
            p2 = p2.next.next

        #first half
        head1 = head
        #second half
        head2 = p1.next
        p1.next = None
        leftList = self.sortList(head1)
        rightList = self.sortList(head2)
        return self.merge(leftList, rightList)

    def merge(self, list1, list2):
        dummy = ListNode(0)
        pre = dummy
        while list1 and list2:
            if list1.val < list2.val:
                pre.next = list1
                list1 = list1.next
            else:
                pre.next = list2
                list2 = list2.next
            pre = pre.next
        if list1:
            pre.next = list1
        if list2:
            pre.next = list2
        return dummy.next
```

Reference

- sort list

Insertion Sort List

题目描述

Sort a linked list using insertion sort.

解题方法

- dummy head
- 每当插入一个新的Node时，就用Dummy开始找它应该在的位置

Solution

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode insertionSortList(ListNode head) {
        ListNode dummy = new ListNode(0);
        while (head != null){
            ListNode pre = dummy;
            while(pre.next != null && pre.next.val < head.val){
                pre = pre.next;
            }
            ListNode tmp = head.next;
            head.next = pre.next;
            pre.next = head;
            head = tmp;
        }
        return dummy.next;
    }
}
```

python的相同code过不了，囧......

Reference

Maximum Gap

题目描述

解题方法

naive的方法，先sort

没想法就先跟面试官说sort,让他知道你是有办法做出来的，但是sort的时间复杂度是 `O(nlogn)`

bucket sort

`O(nlogn)` 的方法无法满足我们，那我们就应该想 `o(n)` ，甚至 `o(logn)` 的方法。

能够达到 `O(n)` 的sort的方法有

- bucket sort
- counting sort
- radix sort

这里我们可以用Bucket sort的方法，找出最大值和最小值确定范围，然后有n个数，最小的gap也就是平均分就是 `(max-min) / (n-1)` ， 所以我们可以有n个bucket。在同一个bucket内的数肯定不会是最大gap，应该找两个相邻的Bucket里的前面的最大值和后面bucket的最小值的gap, 有可能是结果。

Solution

sort的方法

```
class Solution(object):
    def maximumGap(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        length = len(nums)
        if length < 2:
            return 0
        nums.sort()

        maxGap = 0
        for i in range(1, length):
            gap = nums[i] - nums[i-1]
            if gap > maxGap:
                maxGap = gap

        return maxGap
```

bucket sort

```
class Solution(object):
    def maximumGap(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        length = len(nums)
        if length < 2:
            return 0

        minValue = sys.maxint
        maxValue = -sys.maxint
        for num in nums:
            if num > maxValue:
                maxValue = num
            if num < minValue:
                minValue = num
        bucketsNum = length + 1 # 有length + 1个bucket
        buckets = [[] for i in range(bucketsNum)]
        bucketSize = (maxValue - minValue) / bucketsNum + 1 # bucke
        for num in nums:
            index = (num - minValue) / bucketSize # 注意要减去minVal
            buckets[index].append(num)

        p = 0
        maxGap = 0
        while p < bucketsNum:
            if not buckets[p]:
                p += 1
            nextP = p + 1
            while nextP < bucketsNum and not buckets[nextP]:
                nextP += 1
            if nextP >= bucketsNum:
                break
            gap = min(buckets[nextP]) - max(buckets[p])
            if gap > maxGap:
                maxGap = gap
            p = nextP

        return maxGap
```

Reference

Meeting Point

题目描述

A group of two or more people wants to meet and minimize the total travel distance. You are given a 2D grid of values 0 or 1, where each 1 marks the home of someone in the group. The distance is calculated using Manhattan Distance, where distance(p1, p2) = |p2.x - p1.x| + |p2.y - p1.y|.

For example, given three people living at (0,0), (0,4), and (2,2):

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

The point (0,2) is an ideal meeting point, as the total travel distance of 2+2+2=6 is minimal. So return 6.

Hint:

Try to solve it in one dimension first. How can this solution apply to the two dimension case?

follow up:

There can be multiple people in the same room.

解题方法

这一题距离的计算方法其实是x于y分开的，所以我们可以把它分为2个1-D的结果的和。

在X轴的1-D问题上，最佳的meeting point就是所有X坐标的median中间点

- 如果是偶数个，只要是中间两个点之间就可以

- 如果是奇数个，那就应该是中间那个点

Solution

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

        row = len(grid)
        col = len(grid[0])

        rowArr = []
        colArr = []

        for i in range(row):
            for j in range(col):
                if grid[i][j] == 1:
                   rowArr.append(i)
                   colArr.append(j)
        colArr.sort()

        result = 0
        medianRow = rowArr[len(rowArr)/2]
        medianCol = colArr[len(colArr)/2]

        for r in rowArr:
            result += abs(r-medianRow)
        for c in colArr:
            result += abs(c-medianCol)

        return result
```

Reference

- meeting point

Meeting Rooms

题目描述

Given an array of meeting time intervals consisting of start and end times `[[s1,e1],[s2,e2],...] (si < ei)`, determine if a person could attend all meetings.

For example, Given `[[0, 30],[5, 10],[15, 20]]`, return `false`.

解题方法

没有思路就先sort一下，这一题可以用comparator先根据起始时间sort一下, 这样后面的必然在前面之后开始，只需要考虑结束时间就可以。

然后我们再看怎样的情况下会有重叠的intervals，如果后面任意一个的start time早于前面的end time, 那么就代表重叠了。 我们需要比较后面的每一个吗？不需要，因为起始时间是有序的，只要后面第一个的start time晚于当前这一个的end time，就表示后面的肯定与当前这一个没有重叠。

所以我们Loop through每一个interval，比较intervals[i]的end time和intervals[i+1]的start time就可以。

comparator先把intervals按照start time sort一下

Solution

```
# Definition for an interval.
# class Interval(object):
#     def __init__(self, s=0, e=0):
#         self.start = s
#         self.end = e

class Solution(object):
    def canAttendMeetings(self, intervals):
        """
        :type intervals: List[Interval]
        :rtype: bool
        """
        def compare(a, b):
            return a.start - b.start

        intervals = sorted(intervals, cmp=compare)

        length = len(intervals)
        if length == 1:
            return True

        for i in range(1, length):
            if intervals[i].start < intervals[i-1].end:
                return False
        return True
```

Reference

Meeting Rooms II

题目描述

Given an array of meeting time intervals consisting of start and end times [[s1,e1],[s2,e2],...] (si < ei), find the minimum number of conference rooms required.

For example, Given `[[0, 30],[5, 10],[15, 20]]`, return `2`.

解题方法

有点类似airline的那道题

Solution

```
from heapq import *
# Definition for an interval.
# class Interval(object):
#     def __init__(self, s=0, e=0):
#         self.start = s
#         self.end = e

class Solution(object):
    def minMeetingRooms(self, intervals):
        """
        :type intervals: List[Interval]
        :rtype: int
        """
        if not intervals:
            return 0

        length = len(intervals)
        def compare(a, b):
            return a.start - b.start

        intervals = sorted(intervals, cmp=compare)

        curNum = 0
        maxNum = 0
        endTimes = []
        for i in range(length):
            curNum += 1
            heappush(endTimes, intervals[i].end)
            while endTimes and endTimes[0] <= intervals[i].start:
                heappop(endTimes)
                curNum -= 1
            if curNum > maxNum:
                maxNum = curNum
        return maxNum
```

Reference

Insert Intervals

题目描述

Given a set of non-overlapping intervals, insert a new interval into the intervals (merge if necessary).

You may assume that the intervals were initially sorted according to their start times.

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

解题方法

方法1，利用merge

先插到最后，然后用merge的方法过一遍

方法2，遍历，找到插入位置

Solution

方法1

```
class Solution(object):
    def insert(self, intervals, newInterval):
        """
        :type intervals: List[Interval]
        :type newInterval: Interval
        :rtype: List[Interval]
        """
        if not newInterval:
            return intervals
        if not intervals:
            return [newInterval]
        intervals.append(newInterval)

        result = self.merge(intervals)

        return result

    def merge(self, intervals):
        """
        :type intervals: List[Interval]
        :rtype: List[Interval]
        """
        if not intervals:
            return []

        def compare(a, b):
            return a.start - b.start

        intervals = sorted(intervals, cmp=compare)
        length = len(intervals)

        result = [intervals[0]]
        for i in range(1, length):
            cur = result[-1]
            if cur.end >= intervals[i].start:
                newInterval = Interval(cur.start, max(cur.end, inte
                result[-1] = newInterval
            else:
                result.append(intervals[i])
        return result
```

方法2

```
# Definition for an interval.
# class Interval(object):
#     def __init__(self, s=0, e=0):
#         self.start = s
#         self.end = e

class Solution(object):
    def insert(self, intervals, newInterval):
        """
        :type intervals: List[Interval]
        :type newInterval: Interval
        :rtype: List[Interval]
        """
        if not newInterval:
            return intervals
        if not intervals:
            return [newInterval]
        result = []
        insertPos = 0
        length = len(intervals)
        for i in range(length):
            cur = intervals[i]
            if cur.end < newInterval.start:
                result.append(cur)
                insertPos += 1
            elif cur.start > newInterval.end:
                result.append(cur)
            else:
                newStart = min(cur.start, newInterval.start)
                newEnd = max(cur.end, newInterval.end)
                newInterval.start = newStart
                newInterval.end = newEnd
        result.insert(insertPos, newInterval)
        return result
```

Reference

Merge Intervals

题目描述

Given a collection of intervals, merge all overlapping intervals.

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

解题方法

遇到interval类似的题目总是可以将它们先用comparator sort一下，这里的解法也是这样。

- comparator按照start time sort一下
- 先将第一个interval放到result array里，然后后面的interval不断地与result的末尾Interval merge或者加到末尾

Solution

```
# Definition for an interval.
# class Interval(object):
#     def __init__(self, s=0, e=0):
#         self.start = s
#         self.end = e

class Solution(object):
    def merge(self, intervals):
        """
        :type intervals: List[Interval]
        :rtype: List[Interval]
        """
        if not intervals:
            return []

        def compare(a, b):
            return a.start - b.start

        intervals = sorted(intervals, cmp=compare)
        length = len(intervals)

        result = [intervals[0]]
        for i in range(1, length):
            cur = result[-1]
            if cur.end >= intervals[i].start:
                newInterval = Interval(cur.start, max(cur.end, inte
                result[-1] = newInterval
            else:
                result.append(intervals[i])
        return result
```

Reference

Largest Number

题目描述

Given a list of non negative integers, arrange them such that they form the largest number.

For example, given `[3, 30, 34, 5, 9]`, the largest formed number is `9534330`.

Note: The result may be very large, so you need to return a string instead of an integer.

解题方法

对于哪个数放在前面的问题，其实对于a和b两个数

如果`ab > ba`，那么a就应该放在b的前面，根据这一条件写一个comparator,就可以了

Solution

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

        def compare(a, b):
            ab = str(a) + str(b)
            ba = str(b) + str(a)
            ab = int(ab)
            ba = int(ba)
            if ab < ba:
                return - 1
            elif ab == ba:
                return 0
            else:
                return 1

        nums = sorted(nums, cmp=compare)
        if nums[-1] == 0:
            return "0"

        result = ""
        for num in nums:
            result = str(num) + result

        return result
```

Reference

First Missing Positive

题目描述

Given an unsorted integer array, find the first missing positive integer.

For example, Given `[1,2,0]` return `3`, and `[3,4,-1,1]` return `2`.

解题方法

要达到`O(n)`的时间复杂度，肯定就不能先把array sort一下了。

而在`O(n)`的sort方法中有

- radix sort (不适合)
- bucket sort
- counting sort

而counting sort和bucket sort理论上都需要`O(n)`的space

如果转换思路的话，其实可以将原array当做bucket：

- 因为是找first missing positive, 那么不可能大于array的长度
- 如果value是`i`, 就放到`nums[i-1]`中去，这里可以做in-place swap
- 然后再从头遍历，发现第一个`nums[i-1] != i`的，就是first missing positive

Solution

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

        length = len(nums)
        i = 0
        while i < length:
            if nums[i] <= 0:
                i += 1
            else:
                if nums[i] <= length:
                    if nums[i] == i+1 or nums[nums[i]-1] == nums[i
                        # no need to swap
                        i += 1
                        continue
                    tmp = nums[nums[i]-1]
                    nums[nums[i]-1] = nums[i]
                    nums[i] = tmp
                else:
                    i += 1

        i = 0
        while i < length:
            if nums[i] != i + 1:
                return i + 1
            i += 1

        return length + 1  # no missing until length
```

Reference

sorted matrix的搜索

Reference

- 心随风飞

Dynamic Programming

动态规划的4点要素

1. 状态 State 灵感，创造力，存储小规模问题的结果
2. 方程 Function 状态之间的联系，怎么通过小的状态，来算大的状态
3. 初始化 Intialization 最极限的小状态是什么, 起点
4. 答案 Answer 最大的那个状态是什么，终点

常见的四种DP类型

- Sequence DP
- 2 Sequence DP
- Matrix DP
- Others
    - 背包类
    - 区间类

什么情况下可能使用DP

- 求 max/min
- yes/no 求能否达到
- count(*) 求数量

注意点

- 多开一位，把0位空出来

Reference

- code ganker
- 面试会考的DP

- 区间DP
- DP总结
- 关于动态规划的一点心得

一维DP

题目列表

- leetcode

- Climbing Stairs

- Decode Ways
- Unique Binary Search Trees
- Maximum Subarray
- Maximum Product Subarray
- Best Time to Buy and Sell Stock

什么情况下可能使用DP

- 求 max/min
- yes/no 求能否达到
- count(*) 求数量

什么情况下可能不是DP

- 要求出具体方案而不是数量
    - palindrome partitioning
- 输入的是集合而不是序列
    - longest consecutive sequence

问题描述

动态规划就是利用存储历史信息来减少重复计算，从而降低时间复杂度，用空间换时间的算法思路

动态规划的基本思路

1. state，也就是dp表示的状态，确定递推量具体含义，也会定下维度

2. function， 连接方程，这是最难的地方
3. init, 初始化状态
4. result, 最后的result是哪一个状态

常见优化：

有的时候对于历史信息的要求仅限于前面几步，就可以减少存储的空间

类型1

题目

- climbing chairs
- jump game
- decode ways
- unique binary search trees

思路

类型1比较简单，按照general思路来想，找到递推式

类型2 global, local

题目

- maximum subarray
- maximum product subarray
- best time to but and sell stock

思路

当看到一个题目，对于要求的结果可以选择 `包含` 还是 `不包含` 当前 `value[i]` 时，就可以往这个方向想了

这种类型的题目要维护两个DP：

- `global` 全局最优 （到目前为止最好的结果，不一定包含当前值）

- `local` 局部最优 （包含当前值的最优结果）

递推式： 根据 `local` 和 `global` 还有当前 `value[i]` 的关系，找出他们的递推式，最终结果是看 `global`

题目

- submatrix sum

这道题跟subarray sum类似，但是是二维的，但是如果我们将行的上下边界固定，将每一列的和当做一个元素的话，就和array的问题类似。

！pic

- subarray sum和submatrix sum的总结

Triangle

题目描述

Given a triangle, find the minimum path sum from top to bottom. Each step you may move to adjacent numbers on the row below.

For example, given the following triangle

```
[
     [2],
    [3,4],
   [6,5,7],
  [4,1,8,3]
]
```

The minimum path sum from top to bottom is `11 (i.e., 2 + 3 + 5 + 1 = 11)`.

Note:

Bonus point if you are able to do this using only O(n) extra space, where n is the total number of rows in the triangle.

解题方法

这种求最小的的题目，而且上下层之间是有关联的，一般可以往dp的方向想。 dp的四要素

- state
- function
- init state
- result

这里 `dp[i][j]` 代表的是以triangle[i][j]为终点的路径的最小值，因为dp[i][j]的上面一层最多只有两种选择，（在左边界和右边界时只有一种选择），所以就可以推出function是

```
dp[i][j] = min(dp[i-1][j-1], dp[i-1][j]) + triangle[i][j]
```

follow up

因为只有两层之间有联系，所以可以只用两个n为长度的array就可以

Solution

```
class Solution(object):
    def minimumTotal(self, triangle):
        """
        :type triangle: List[List[int]]
        :rtype: int
        """
        if not triangle:
            return 0
        rowNum = len(triangle)
        dp = [[sys.maxint for i in range(rowNum)]for j in range(row
        dp[0][0] = triangle[0][0]
        for i in range(1, rowNum):
            for j in range(i+1):
                if j == 0:
                    dp[i][0] = dp[i-1][0] + triangle[i][0]
                elif j == i:
                    dp[i][j] = dp[i-1][j-1] + triangle[i][j]
                else:
                    dp[i][j] = min(dp[i-1][j-1], dp[i-1][j]) + tria

        minResult = sys.maxint
        for i in range(rowNum):
            if dp[rowNum-1][i] < minResult:
                minResult = dp[rowNum-1][i]

        return minResult
```

空间优化

注意，不能直接 `dp1=dp2` ，因为这样它们指向的是同一个地址，那么dp2在变化的时候dp1指向的array也会 随着变化，就会导致结果不对，需要新建一个array，用 `dp1 = list(dp2)`

```
class Solution(object):
    def minimumTotal(self, triangle):
        """
        :type triangle: List[List[int]]
        :rtype: int
        """
        if not triangle:
            return 0
        rowNum = len(triangle)
        if rowNum == 1:
            return triangle[0][0]
        dp1 = [sys.maxint for i in range(rowNum)]
        dp2 = [sys.maxint for i in range(rowNum)]
        dp1[0] = triangle[0][0]
        for i in range(1, rowNum):
            for j in range(i+1):
                if j == 0:
                    dp2[0] = dp1[0] + triangle[i][0]
                elif j == i:
                    dp2[j] = dp1[j-1] + triangle[i][j]
                else:
                    dp2[j] = min(dp1[j-1], dp1[j]) + triangle[i][j
            dp1 = list(dp2)

        return min(dp2)
```

Reference

Pascal's Triangle

题目描述

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

For example, given `numRows = 5`, Return

```
[
     [1],
    [1,1],
   [1,2,1],
  [1,3,3,1],
 [1,4,6,4,1]
]
```

解题方法

这一题和triangle类似，每一层元素是它上面两个元素的和

Solution

```
class Solution(object):
    def generate(self, numRows):
        """
        :type numRows: int
        :rtype: List[List[int]]
        """
        if numRows == 0:
            return []

        result = [[1]]
        if numRows == 1:
            return result
        cur = [1]

        for i in range(1, numRows):
            tmp = [0 for i in range(i+1)]
            for j in range(i+1):
                if j == 0:
                    tmp[0] = 1
                elif j == i:
                    tmp[j] = 1
                else:
                    tmp[j] = cur[j-1] + cur[j]
            result.append(tmp)
            cur = list(tmp)

        return result
```

Reference

Pascal's Triangle II

题目描述

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

For example, given `k = 3`, Return `[1,3,3,1]`.

Note: Could you optimize your algorithm to use only O(k) extra space?

解题方法

和triangle一样的思路，follow up就是只用上一层和这一层两个array，这样就只需要 `O(k)` 的space

Solution

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

        cur = [1]
        if rowIndex == 0:
            return cur

        for i in range(1, rowIndex+1):
            tmp = [0 for i in range(i+1)]
            for j in range(i+1):
                if j == 0:
                    tmp[0] = 1
                elif j == i:
                    tmp[j] = 1
                else:
                    tmp[j] = cur[j-1] + cur[j]
            cur = list(tmp)

        return tmp
```

Reference

House Robber

题目描述

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

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

解题方法

dp[i]表示打劫到第i间房间时候的最大值（包含或者不包含i），其实可以更清晰地用global和local来表示是否包含第i间房子。

`dp[i] = max(dp[i - 1], dp[i - 2] + num[i - 1])`

还可以简化将空间复杂度变为 `O(1)`

Solution

```
class Solution(object):
    def rob(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        if not nums:
            return 0
        length = len(nums)
        globalDP = [0 for i in range(length)]
        localDP = [0 for i in range(length)]

        globalDP[0] = nums[0]
        localDP[0] = nums[0]
        for i in range(1, length):
            if i == 1:
                localDP[i] = nums[1]
                globalDP[i] = max(nums[0], nums[1])
            else:
                localDP[i] = globalDP[i-2] + nums[i]
                globalDP[i] = max(localDP[i], localDP[i-1])

        return globalDP[-1]
```

Reference

- house robber

House Robber II

题目描述

This time, all houses at this place are **arranged in a circle**.

解题方法

连成圈也就是说明第一间和最后一件不能同时抢，一开始想如何在转换方程里体现出来，并没有什么思路。 后来看了别人的Blog，其实就将它们分成不包含第一间和不包含最后一间的两个array就可以了。

Solution

```
class Solution(object):
    def rob(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        if not nums:
            return 0
        if len(nums) == 1:
            return nums[0]
        return max(self.rob1(nums[:-1]), self.rob1(nums[1:]))
```

Reference

- house robber 2

Best Time to Buy and Sell Stock

题目描述

解题方法

Solution

Reference

- stock 1,2,3,4

Best time to buy and sell stock with cool down

题目描述

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

Design an algorithm to find the maximum profit. You may complete as many transactions as you like (ie, buy one and sell one share of the stock multiple times) with the following restrictions:

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

Example:

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

解题方法

这一题跟2的区别就是有一天的cooldown不能买stock。4的做法是用 `local` 和 `global` 来区别在第i天是否有sell的行为， 而这里第i天sell和buy的行为对后面的影响是不同的，所以我们要用两个数组 `sells` 和 `buys` 来区别

- `buys[i]` 表示在第i天买或者不买达到的最大收益
- `sell[i]` 表示在第i天卖或者不卖达到的最大收益

初始状态

```
buys = [0 for i in range(length)]
sells = [0 for i in range(length)]

sells[0], sells[1] = 0, max(0, prices[1] - prices[0])
buys[0], buys[1] = -prices[0], max(-prices[0], -prices[1])
```

function

```
sells[i] = max(sells[i-1], buys[i-1] + prices[i])
buys[i] = max(buys[i-1], sells[i-2] - prices[i])
```

Solution

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

        length = len(prices)

        if length < 2:
            return 0

        buys = [0 for i in range(length)]
        sells = [0 for i in range(length)]

        sells[0], sells[1] = 0, max(0, prices[1] - prices[0])
        buys[0], buys[1] = -prices[0], max(-prices[0], -prices[1])

        for i in range(2, length):
            sells[i] = max(sells[i-1], buys[i-1] + prices[i])
            buys[i] = max(buys[i-1], sells[i-2] - prices[i])

        return sells[-1]
```

Reference

- cooldown

Word Break

题目描述

Given a string s and a dictionary of words dict, determine if s can be segmented into a space-separated sequence of one or more dictionary words.

For example, given `s = "leetcode"`, `dict = ["leet", "code"]`.

Return true because `"leetcode"` can be segmented as `"leet code"`.

解题方法

只求最终是否能够的状态，并且前后子问题都有联系，可以往DP方向想。

1. state

`canBreak[i]` 表示前i个字符是否能够做word break

1. function

`canBreak[i] = True if canBreak[j] and (s[j:i] in wordDict) for j < i`

1. result

`canBreak[len(s)]`

Solution

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

        length = len(s)

        canBreak = [False for i in range(length+1)]
        canBreak[0] = True

        for i in range(1, length+1):
            for j in range(i):
                if canBreak[i]:
                    continue
                if (s[j:i] in wordDict) and canBreak[j]:
                    canBreak[i] = True

        return canBreak[length]
```

Reference

- word break

Dungeon Game

题目描述

The demons had captured the princess (P) and imprisoned her in the bottom-right corner of a dungeon. The dungeon consists of M x N rooms laid out in a 2D grid. Our valiant knight (K) was initially positioned in the top-left room and must fight his way through the dungeon to rescue the princess.

The knight has an initial health point represented by a positive integer. If at any point his health point drops to 0 or below, he dies immediately.

Some of the rooms are guarded by demons, so the knight loses health (negative integers) upon entering these rooms; other rooms are either empty (0's) or contain magic orbs that increase the knight's health (positive integers).

In order to reach the princess as quickly as possible, the knight decides to move only rightward or downward in each step.

解题方法

这道题目有两种解法

Binary search + 验证

从0到 `sys.maxint` ，不断地取中间值，然后验证时候能够存活，不能的话二分舍去一半再找。

利用的Binary search的思想，但是时间复杂度很高

DP

这种**max, min**的问题我们要想到可以用DP做，但是难点是在于如果定下dp代表的状态和连接function。

因为这道题跟unique path不同的地方在于这里要求每一步都不能小于0。从走的顺序推的问题在于不能确定下一步是否会为0， 而反过来推的话就可以去掉这一个烦恼，因为只要再满足现在的位置，后面的肯定可以存活，所以我们从终点往前推。

state

`dp[i][j]` 代表从(i,j)位置走到终点所需要的最小起始生命值

function

那么

```
dp[i][j] = max(  min(dp[i+1][j], dp[i][j+1]) - dungeon[i][j], 0 )
```

如果这个比较难读的话，起始应该是

```
dp[i][j] = min(dp[i+1][j], dp[i][j+1]) - dungeon[i][j]
if dp[i][j] < 0:
    dp[i][j] = 0
```

当前dungeon[i][j]如果是正的表示加血，dp[i][j]的起始生命值就应该减去，反之也是。 因为起始生命值不能为负数，从前面走到这一步也不可以是负的，所以小于0的数都应该改为0

init

初始值就是两个边的值

result

`dp[0][0] + 1`

Solution

```python
class Solution(object):
    def calculateMinimumHP(self, dungeon):
        """
        :type dungeon: List[List[int]]
        :rtype: int
        """
        if not dungeon:
            return -1
        row = len(dungeon)
        col = len(dungeon[0])

        #dp[i][j] means from i,j to row-1, col-1
        dp = [[0 for i in range(col)] for j in range(row)]
        dp[row-1][col-1] = max(0 - dungeon[row-1][col-1], 0)

        #init states
        for i in range(row-2, -1, -1):
            dp[i][col-1] = max(dp[i+1][col-1] - dungeon[i][col-1],

        for j in range(col-2, -1, -1):
            dp[row-1][j] = max(dp[row-1][j+1] - dungeon[row-1][j],

        for i in range(row-2, -1, -1):
            for j in range(col-2, -1, -1):
                dp[i][j] = min(dp[i+1][j], dp[i][j+1]) - dungeon[i]
                if dp[i][j] < 0:
                    dp[i][j] = 0
        return dp[0][0] + 1
```

Reference

- dp
- 二分法

Max Rectangle

题目描述

解题方法

这一题可以在largest rectangle in histogram基础上做

Solution

Reference

Maximal Square

题目描述

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

For example, given the following matrix:

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

Return 4.

解题方法

brute force

找出所有的square, 再check是否都是1

DP

这一道题应该通过画图来理解， `dp[i][j]` 表示以[i, j]为右下角时最大square的边长

画图就可以看出， `dp[i][j]` 和 `dp[i-1]` ， `dp[i][j-1]` 还有 `dp[i-1][j-1]` 有关

Solution

我的code

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

        rowNum = len(matrix)
        colNum = len(matrix[0])


        dp = [[0 for i in range(colNum)] for j in range(rowNum)]

        for i in range(rowNum):
            if matrix[i][0] == "1":
                dp[i][0] = 1
                maxLength = 1

        for j in range(colNum):
            if matrix[0][j] == "1":
                dp[0][j] = 1
                maxLength = 1

        for i in range(1, rowNum):
            for j in range(1, colNum):
                if matrix[i][j] == "0":
                    dp[i][j] = 0
                else:
                    dp[i][j] = min(dp[i-1][j], dp[i][j-1], dp[i-1][
                    maxLength = dp[i][j] if dp[i][j] > maxLength el

        return maxLength ** 2
```

别人更简洁的code

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

        # 学习这种写法
        dp = [[0 for x in row] for row in matrix]

        max_square_size = 0
        # 学习这种2维loop的写法
        for i, row in enumerate(matrix):
            for j, elem in enumerate(row):
                if i == 0 or j == 0:
                    dp[i][j] = 1 if matrix[i][j] == "1" else 0
                else:
                    dp[i][j] =  0 if matrix[i][j] == "0" else min(

                if dp[i][j] > max_square_size:
                    max_square_size = dp[i][j]

        return max_square_size**2
```

Reference

Edit Distance

题目描述

Given two words word1 and word2, find the minimum number of steps required to convert word1 to word2. (each operation is counted as 1 step.)

You have the following 3 operations permitted on a word:

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

解题方法

这种求minimum的问题，并且前后子问题之间有关系，可以直觉地往ＤＰ方面去想。

state

`dp[i][j]` 表示 `word[:i]` 和 `word[:j]` 的minimum edit distance

init status

- `dp[i][0] = i`
- `dp[0][j] = j`

这种情况下只有insert相应个数的字符

function

DP的关键就是找前后两种状态之间的关系，那么 `dp[i][j]` 之前的关系也是跟三种操作有关

- word1 insert one character

    - `dp[i][j] = dp[i-1][j] + 1`
- word2 delete one character(it's actually the same as insert)
    - 'dp[i][j] = dp[i][j-1] + 1'
- replace a character +　在考虑replace的时候，因为我们是找与之前的状态的关系，所以我们只用考虑 当期新增的character的改变，而不需要考虑之前的edit
    - 所以也就有两种可能，取决于word1[i-1]是否等于word2[j-1]

Solution

```
class Solution(object):
    def minDistance(self, word1, word2):
        """
        :type word1: str
        :type word2: str
        :rtype: int
        """
        length1 = len(word1)
        length2 = len(word2)

        dp = [[sys.maxint for i in range(length2+1)] for j in range
        dp[0][0] = 0

        for i in range(length1+1):
            dp[i][0] = i

        for j in range(length2+1):
            dp[0][j] = j

        for i in range(1, length1+1):
            for j in range(1, length2+1):
                a = dp[i-1][j] + 1
                b = dp[i][j-1] + 1
                if word1[i-1] == word2[j-1]:
                    c = dp[i-1][j-1]
                else:
                    c = dp[i-1][j-1] + 1
                dp[i][j] = min(a, b, c)

        return dp[length1][length2]
```

Reference

- 喜刷刷

Regular Expression Matching

题目描述

Implement regular expression matching with support for `'.'` and `'*'`.

`'.'` Matches any single character. `'*'` Matches zero or more of the preceding element.

The matching should cover the entire input string (not partial).

The function prototype should be:

`bool isMatch(const char *s, const char *p)`

Some examples:

```
isMatch("aa","a") → false
isMatch("aa","aa") → true
isMatch("aaa","aa") → false
isMatch("aa", "a*") → true
isMatch("aa", ".*") → true
isMatch("ab", ".*") → true
isMatch("aab", "c*a*b") → true
```

解题方法

backtracking + dfs

这道题目最直观的想法就是，应该根据 `p` 的情况不断地去试，比较难处理的地方就是有 `.` 和 `*` 的地方，而这种不断去试的题目 又让我想到了permutation和subset这几题，也就是backtracking + dfs的方法。而且这个思路我们可以去往下想，根据 `.` 和 `*` 来 进行分情况讨论。

并且我们可以注意到，如果这其中的小问题之间是有联系的，如果之前的部分已经match, 那么我们可以直接recursive call the function on 剩下的string和pattern. 所以这是一个很自然的recursive的方法。

我们可以对 `p` 的前两位进行分析，因为只要分析这两位中的 `.` 和 `*` 情况，就可以进行下一个recursive call了。

首先是 `len(p) == 0` 的情况先考虑，这时候如果 `s` 也为empty string才match

1. `len(p) == 1 or p[1] != "*"`

因为p的第一位不能是 `*` ，所以我们只需要考虑 `p[1]` 是否为 `*` ， 因为只有为 `*` 时，才会影响到后面的match, 否则就只需要 单独考虑这一位就可以。

- `p` 只剩下一位，那么只需要判断s和这一位的关系就可以了
- `p[1] != "*"` ， 那么说明 `p[0]` 不会match后面的char, 所以可以单独考虑和第一位的关系了

- `p[1] == "*"`

这时候前两位就会对后面的match产生影响了， `*` 代表可以match zero or more characters，而到底是多少我们不知道， 所以需要一个一个地去试，这里就可以写一个loop，最短match **0**位，最长match 整个string来试。

超出了leetcode的time limit

DP

这里的子问题直接是有联系的，并且只要求能与不能，我们又可以往DP的方向想。

state

`dp[i][j]` 表示 `s[:i+1]` 可以由 `p[j+1]` match

init status

就是s长度为0和p长度为0的时候

function

这里的function比较复杂，也需要分情况讨论， `dp[i][j]` 肯定是由之前的状态得来

1. `p[j-1] != "." and p[j-1] == "*"`

之前一个不是特殊字符，那么只需要check当前这一位就可以了。

```
if dp[i-1][j-1] and s[i-1] == p[j-1]:
    dp[i][j] = True
```

1. `p[j-1] == "."`

当前这一位可以match任何字符，并且必须match一位，那么 `dp[i][j] = dp[i-1][j-1]`

1. `p[j-1] == "*"`

这时候又有了很多种情况， `p[j-2]` 可以match zero or more characters, 而match次数>=2的时候可以再退回到0或1的情况

- `if dp[i][j-2] == True`，说明match 0的情况可以成功，就可以将 `dp[i][j]` 设为 `True`
- `if dp[i][j-1] == True`，说明match 1的情况可以成功
- match 2次以上的情况，可以退回到 `dp[i-1][j]`， 最终可以退到0次或1次
    - `if dp[i-1][j] and (s[i-1] == p[j-2] or p[j-2] == "."): dp[i][j] = True`
    - 这个能想到很重要，否则match 2次以上很难处理。可以试着找找前后的关系找灵感

注意点

- `dp[i][j]`代表的是前i个，前j个字符，而不是Index, 在初始化dp的时候要比长度多一位

two pointer方法

Solution

backtracking + dfs

```
class Solution(object):
    def isMatch(self, s, p):
        """
        :type s: str
        :type p: str
        :rtype: bool
        """
        if len(p) == 0:
            return len(s) == 0

        if len(p) == 1 or p[1] != "*":
            if len(s) == 0 or ( s[0] != p[0] and p[0] != "."):
                return False
            return self.isMatch(s[1:], p[1:])
        else:
            i = - 1 # 因为要试验match 0位，所以这里从-1开始，考虑match 0
            # i < 0 -- match 0
            # p[0] == "." or p[0] == s[i]都表示可以match当前位
            while i < len(s) and ( i < 0 or p[0] == "." or p[0] ==
                if self.isMatch(s[i+1:], p[2:]):
                    return True
                i += 1
            return False
```

DP

```
class Solution(object):
    def isMatch(self, s, p):
        """
        :type s: str
        :type p: str
        :rtype: bool
        """
        if len(p) == 0:
            return len(s) == 0

        lengthS = len(s)
        lengthP = len(p)

        dp = [[False for i in range(lengthP + 1)] for j in range(le

        dp[0][0] = True

        # init state, match no characters
        # last char in pattern has to be "*", which match 0 charact
        for i in range(2, lengthP + 1):
            if p[i-1] == "*":
                dp[0][i] = dp[0][i-2]


        for i in range(1, lengthS + 1):
            for j in range(1, lengthP + 1):
                if p[j-1] != "." and p[j-1] != "*":
                    if dp[i-1][j-1] and s[i-1] == p[j-1]:
                        dp[i][j] = True
                elif p[j-1] == ".":
                    dp[i][j] = dp[i-1][j-1]
                elif j > 1: # "*" can't be first char
                    if dp[i][j-2]:
                        # match 0
                        dp[i][j] = True
                    elif dp[i][j-1]:
                        # match 1
                        dp[i][j] = True
                    elif dp[i-1][j] and (s[i-1] == p[j-2] or p[j-2]
                        dp[i][j] = True

        return dp[lengthS][lengthP]
```

Reference

- python的题解
- 喜刷刷
- leetcode上的two pointer解法

Wildcard Matching

题目描述

Implement wildcard pattern matching with support for `'?'` and `'*'`.

```
'?' Matches any single character.
'*' Matches any sequence of characters (including the empty sequenc

The matching should cover the entire input string (not partial).

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

Some examples:
isMatch("aa","a") → false
isMatch("aa","aa") → true
isMatch("aaa","aa") → false
isMatch("aa", "*") → true
isMatch("aa", "a*") → true
isMatch("ab", "?*") → true
isMatch("aab", "c*a*b") → false
```

解题方法

backtracking + dfs

和regular expression的做法差不多，转变一下 `*` 的处理情况

DP

首先我们想到这一题和regular rexpression matching差不多，也可以用DP来解，主要就是将"*"的情况改变一下

two pointer

上面两种做法都过不了leetcode,因为time limit和stack limit，我们只能另想办法。看了各路大神的解法，找到了将 recursive改为two-pointer的做法，思路还是遇到 `*` 时不断从0个开始往后试探匹配，如果不行的话就回朔，关键就是 回朔时候的pointer reset到哪里的问题。

pointer reset到哪里的问题

- 首先我们有两个pointer分别对应 `s` 和 `p`

当处理 `*` 时，我们需要从匹配0个开始不断地试探匹配，当匹配不成功的话，我们就应该回朔，尝试再用 `*` 多匹配 一位继续尝试看是否能够成功，这个reset的过程我们对两个Pointer都要reset

- 假如我们上一次已经尝试用 `*` 匹配了n个字符，那么 `s` 的pointer应该设置到上一次匹配n个字符的后一位
- 而 `p` 的pointer呢，我们应该还是从 `*` 后面开始新一次的尝试

所以我们要有两个变量

- `star` 记录 `*` 出现的位置
- `lastMatch` 记录上一次对 `s` 匹配到的位置

而根据这个文章分析，我们只需要回去尝试最后一个 `*` 就可以。

Solution

DP

超时

```
class Solution(object):
    def isMatch(self, s, p):
        """
        :type s: str
        :type p: str
        :rtype: bool
        """
        if len(p) == 0:
            return len(s) == 0

        lengthS = len(s)
        lengthP = len(p)

        dp = [[False for i in range(lengthP + 1)] for j in range(le

        dp[0][0] = True

        # init state, match no characters
        # last char in pattern has to be "*", which match 0 charact
        for i in range(1, lengthP + 1):
            if p[i-1] == "*":
                dp[0][i] = True
            else:
                break

        for i in range(1, lengthS + 1):
            for j in range(1, lengthP + 1):
                if p[j-1] != "?" and p[j-1] != "*":
                    if dp[i-1][j-1] and s[i-1] == p[j-1]:
                        dp[i][j] = True
                elif p[j-1] == "?":
                    dp[i][j] = dp[i-1][j-1]
                else: # "*" can't be first char
                    if dp[i][j-1]:
                        # match 0
                        dp[i][j] = True
                    elif dp[i-1][j-1]:
                        # match 1
                        dp[i][j] = True
                    elif dp[i-1][j]:
                        # match 2+, 往前退回
                        dp[i][j] = True

        return dp[lengthS][lengthP]
```

backtracking + dfs

```
def isMatch(self, s, p):
        # write your code here
        if len(p) == 0: return len(s) == 0

        if p[0] != "*":
            if len(s) == 0 or (s[0] != p[0] and p[0] != "?"):
                return False
            return self.isMatch(s[1:], p[1:])
        else:
            for i in range(0, len(s)+1):
                if self.isMatch(s[i:], p[1:]):
                    return True

        return False
```

Two-pointer 做法

```
class Solution(object):
    def isMatch(self, s, p):
        """
        :type s: str
        :type p: str
        :rtype: bool
        """
        pI = 0
        sI = 0
        lastMatch = -1
        star = -1

        while sI < len(s):
            if pI < len(p) and (p[pI] == "?" or p[pI] == s[sI]):
                # 当是"?"或者是匹配字符时，不要考虑别的，直接都往后一位
                pI += 1
                sI += 1
            elif pI < len(p) and p[pI] == "*":
                # 当是 * 的时候， 就应该从匹配0个开始尝试
                # start 记录 * 的位置以便下次reset
                # lastMatch 设为当前匹配到的s的位置
                # 只前进p的指针，表示先尝试对s匹配0个
                star = pI
                lastMatch = sI
                pI += 1
            elif star != -1:
                # 前面都没有满足或者j已经超出范围了，无法匹配了，就应该重设
                # p的指针重设到*的后面
                # s的指针重设到上一次尝试匹配n个字符的后一个
                pI = star + 1
                lastMatch += 1
                sI = lastMatch
            else:
                # 没有*,已经无法匹配，直接返回 False
                return False

        # 如果p的指针还没有到底的话，只能是将*的跳过
        while pI < len(p) and p[pI] == "*":
            pI += 1

        # 如果p的指针后面还有多余的非*的字符，说明无法匹配
        # 如果到底了，说明成功匹配
        return pI == len(p)
```

Reference

- wildcard matching
- yu's garden

Find the two repeating elements

题目描述

You are given an array of `n+2` elements. All elements of the array are in range 1 to n. And all elements occur once except two numbers which occur twice. Find the two repeating numbers.

For example, array = `{4, 2, 4, 5, 2, 3, 1}` and `n = 5`

The above array has n + 2 = 7 elements with all elements occurring once except `2` and `4` which occur twice. So the output should be 4 2.

解题方法

1. brute force

2-for loops

Time Complexity: `O(n*n)` Auxiliary Space: `O(1)`

2. count array

count the occurances of each element

Time Complexity: O(n) Auxiliary Space: O(n)

we can get x + y and x * y `(x-y)^2 = x^2 + 2xy + y ^ 2 = (x-y)^2 - 4xy`

Time Complexity: `O(n)` Auxiliary Space: `O(1)`

3. two equations

the sum of 1 to n should be n(n+1)/2 the product of 1 to n should be n!

4. XOR

类似find two non repeating elements in a array, 和1 to n XOR，然后找到第一个不是0的bit,然后分成两组再和 1 to n中这一个Bit为1的XOR，得到其中一个，另一个类似方法得到。

5. Use array elements as index

```
Example: A[] =  {1, 1, 2, 3, 2}
i=0;
Check sign of A[abs(A[0])] which is A[1].  A[1] is positive, so mak
Array now becomes {1, -1, 2, 3, 2}

i=1;
Check sign of A[abs(A[1])] which is A[1].  A[1] is negative, so A[1

i=2;
Check sign of A[abs(A[2])] which is A[2].  A[2] is  positive, so ma
Array now becomes {1, -1, -2, 3, 2}

i=3;
Check sign of A[abs(A[3])] which is A[3].  A[3] is  positive, so ma
Array now becomes {1, -1, -2, -3, 2}

i=4;
Check sign of A[abs(A[4])] which is A[2].  A[2] is negative, so A[4
```

Solution

```
```

Reference

- geeksforgeeks

Find the Duplicate Number

题目描述

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

Note:

1. You must not modify the array (assume the array is read only).
2. You must use only constant, O(1) extra space.
3. Your runtime complexity should be less than O(n2).
4. There is only one duplicate number in the array, but it could be repeated more than once.

解题方法

find two repeating elements的方法1-4不适用，方法5可以改造到这里

- 每经过一个元素i，就将它nums[i]设为负数
- 这样如果下一次遇到元素j，如果nums[j]已经为负数，说明它是一个重复元素

时间复杂度 `O(N)` 空间复杂度 `O(1)`

另一种方法

另一种方法是可以将元素i放到它对应的位置index `i-1` 去，然后不断地swap, 最后再扫一遍，对应有两个数的位置就是重复的数

Solution

```
class Solution(object):
    def findDuplicate(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        for i, num in enumerate(nums):
            if num < 0:
                x = num * -1
            else:
                x = num
            if nums[x] >= 0:
                nums[x] = nums[x] * -1
            else:
                return x
```

Move Zeroes

题目描述

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

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

Note:

1. You must do this in-place without making a copy of the array.
2. Minimize the total number of operations.

解题方法

跟sort color方法一样

Solution

```
class Solution(object):
    def moveZeroes(self, nums):
        """
        :type nums: List[int]
        :rtype: void Do not return anything, modify nums in-place i
        """
        length = len(nums)
        if length == 0:
            return

        left, numNonZero = 0, 0
        while left < length:
            if nums[left] != 0:
                tmp = nums[numNonZero]
                nums[numNonZero] = nums[left]
                nums[left] = tmp
                numNonZero += 1
                left += 1
            else:
                left += 1
```

Remove Element

题目描述

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

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

解题方法

因为不需要保持原有array的顺序等，所以遇到要去掉的就将末尾的换过了，并且用一个variable记录去掉的数的数量。

Solution

```
class Solution(object):
    def removeElement(self, nums, val):
        """
        :type nums: List[int]
        :type val: int
        :rtype: int
        """
        if not nums:
            return 0
        numVal = 0
        length = len(nums)
        p = 0
        while p < length - numVal:
            if nums[p] == val:
                nums[p] = nums[length - 1 - numVal]
                numVal += 1
                #don't increment i here, cause you don't the new va
            else:
                p += 1

        return length - numVal
```

Reference

Spiral Matrix

题目描述

Given a matrix of `m x n` elements (m rows, n columns), return all elements of the matrix in spiral order.

For example, Given the following matrix:

```
[
 [ 1, 2, 3 ],
 [ 4, 5, 6 ],
 [ 7, 8, 9 ]
]
```

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

解题方法

这题比较像rotate image那道题，思路不难，但是code容易出错。

- 记录下左上角和右下角的坐标位置， 不断地处理最外围一圈
- 递归的继续下一个matrix的最外围一圈

注意点

m和n有可能不相同，所以最后又可能剩下的不是一个正方形matrix,而是一行或者一列，这个需要特殊处理

Solution

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

        rowNum = len(matrix)
        colNum = len(matrix[0])
        result = []
        self.helper(matrix, result, 0, 0, rowNum - 1, colNum - 1)
        return result

    def helper(self, matrix, result, startX, startY, endX, endY):
        if startX == endX:
            for i in range(startY, endY + 1):
                result.append(matrix[startX][i])
            return
        if startY == endY:
            for i in range(startX, endX + 1):
                result.append(matrix[i][startY])
            return

        # upper row
        for i in range(startY, endY):
            result.append(matrix[startX][i])

        # right column
        for i in range(startX, endX):
            result.append(matrix[i][endY])

        # bottom row
        for i in range(endY, startY, -1):
            result.append(matrix[endX][i])

        # left column
        for i in range(endX, startX, -1):
            result.append(matrix[i][startY])

        sX = startX + 1
        sY = startY + 1
        eX = endX - 1
        eY = endY - 1
        if eX >= sX and eY >= sY:
            self.helper(matrix, result, sX, sY, eX, eY)
```

Reference

Spiral Matrix II

题目描述

Given an integer n, generate a square matrix filled with elements from 1 to n2 in spiral order.

For example, Given n = 3,

You should return the following matrix:

```
[
 [ 1, 2, 3 ],
 [ 8, 9, 4 ],
 [ 7, 6, 5 ]
]
```

解题方法

和1类似，将数字一个个填进去

Solution

```
class Solution(object):
    def generateMatrix(self, n):
        """
        :type n: int
        :rtype: List[List[int]]
        """
        if n == 0:
            return []
        result = [[0 for i in range(n)] for j in range(n)]
        self.helper(result, 1, 0, 0, n-1, n-1)
        return result
    def helper(self, result, num, startX, startY, endX, endY):
        if startX == endX:
            for i in range(startY, endY + 1):
                result[startX][i] = num
                num += 1
            return
        if startY == endY:
            for i in range(startX, endX + 1):
                result[i][startY] = num
                num += 1
            return

        # upper row
        for i in range(startY, endY):
            result[startX][i] = num
            num += 1

        # right column
        for i in range(startX, endX):
            result[i][endY] = num
            num += 1

        # bottom row
        for i in range(endY, startY, -1):
            result[endX][i] = num
            num += 1

        # left column
        for i in range(endX, startX, -1):
            result[i][startY] = num
            num += 1

        sX = startX + 1
        sY = startY + 1
        eX = endX - 1
        eY = endY - 1
        if eX >= sX and eY >= sY:
            self.helper(result, num, sX, sY, eX, eY)
```

Reference

Remove Duplicates in Sorted Array

题目描述

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

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

For example, Given input array `nums = [1,1,2]` ,

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

解题方法

- 用一个pointer表示指向新的array的下一个放置distinct value的位置
- two pointer, p1和p2,来寻找重复的元素

Solution

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

        newIndex = 0
        p1 = 0
        p2 = 1
        length = len(nums)

        while p1 < length:
            p2 = p1 + 1
            while p2 < length and nums[p1] == nums[p2]:
                p2 += 1
            nums[newIndex] = nums[p1]
            newIndex += 1
            p1 = p2
        return newIndex
```

Reference

Remove Duplicates in sorted array II

题目描述

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

解题方法

用p1,p2两个指针来计算当前元素重复了几次

Solution

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

        newIndex = 0
        p1 = 0
        p2 = 1
        length = len(nums)

        while p1 < length:
            p2 = p1 + 1
            while p2 < length and nums[p1] == nums[p2]:
                p2 += 1
            times = p2 - p1
            if times >= 2:
                nums[newIndex] = nums[p1]
                newIndex += 1
                nums[newIndex] = nums[p1]
                newIndex += 1
            else:
                nums[newIndex] = nums[p1]
                newIndex += 1
            p1 = p2
        return newIndex
```

Reference

Longest Consecutive Sequence

题目描述

Given an unsorted array of integers, find the length of the longest consecutive elements sequence.

For example, Given `[100, 4, 200, 1, 3, 2]`, The longest consecutive elements sequence is `[1, 2, 3, 4]`. Return its length: `4`.

Your algorithm should run in `O(n)` complexity.

解题方法

brute force

sort一下，然后遍历找连续sequence

时间复杂度 `O(nlogn)`

hashtable

用空间换时间

- 先用hashtable保存每一个num为key, value设为False
- 然后遍历的时候，对于每一个num找上下能够达到的连续边界
    - 将用过的数在hashtable里的value设为False,这样就不会重复利用了
- 时间复杂度 `O(n)`
- 空间复杂度 `O(n)`

Solution

```python
class Solution(object):
    def longestConsecutive(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        if not nums:
            return 0
        hashMap = {}
        for num in nums:
            hashMap[num] = True

        result = 0
        for num in nums:
            if not hashMap[num]:
                continue
            curLen = 1
            k = num + 1
            while k in hashMap and hashMap[k]:
                curLen += 1
                hashMap[k] = False
                k += 1
            k = num - 1
            while k in hashMap and hashMap[k]:
                curLen += 1
                hashMap[k] = False
                k -= 1
            result = curLen if curLen > result else result

        return result
```

Reference

- 博客园

One Edit Distance

题目描述

Given two strings S and T, determine if they are both one edit distance apart.

解题方法

这道题目和edit distance是相关的，one edit有三种可能的操作

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

- 首先是长度差距不能超过 1
- 如果长度差距是 1 的花，insert和delete其实是一样的，就是长的那个string delete一个字符
- 长度相等的时候，它们相对应位置字符不同的个数不能多于 1

Solution

```
class Solution(object):
    def isOneEditDistance(self, s, t):
        """
        :type s: str
        :type t: str
        :rtype: bool
        """
        lengthS = len(s)
        lengthT = len(t)

        diff = abs(lengthS - lengthT)

        if diff > 1:
            return False
        elif diff == 1:
            if lengthS > lengthT:
                for i in range(lengthS):
                    tmp = s[0:i] + s[i+1:]
                    if tmp == t:
                        return True
            else:
                for i in range(lengthT):
                    tmp = t[0:i] + t[i+1:]
                    if tmp == s:
                        return True
            return False
        else:
            if s == t:
                return False
            diffNum = 0
            for i in range(lengthS):
                if s[i] != t[i]:
                    if diffNum == 0:
                        diffNum += 1
                    else:
                        return False
            return True
```

Reference

Majority

题目描述

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

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

解题方法

Brute Force

Always先尝试用最直接的方法解决。这一题最直接的方法就是用一个hashtable记录下每个number出现的 次数，然后再找出出现次数大于 `n/2` 的数。

时间复杂度: `O(n)` 空间复杂度: `O(n)`

Moore's voting Algorithm

利用Moore's voting algorithm可以在空间 `O(1)` 的情况下找到。

基本思路就是keep一个variable x,

- 如果等于x， count就加1
- 当count为0时，就将x设为当前的值
- 用所有不等于x的element将count减1
- 那么到最后剩下来的那个就是Majority number.
- 得到这个Number后再计算一遍它的出现次数来验证
    - (不要忘了，特别是在不保证有majority number的情况下)

Solution

```python
class Solution(object):
    def majorityElement(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        if not nums:
            return
        length = len(nums)
        if length == 1:
            return nums[0]
        x = nums[0]
        count = 1
        for i in range(1, length):
            if count == 0:
                x = nums[i]
                count = 1
            else:
                if nums[i] != x:
                    count -= 1
                else:
                    count += 1

        countX = 0
        for num in nums:
            if num == x:
                countX += 1

        if countX > length / 2:
            return x
        else:
            return
```

Reference

- moore's voting algorithm
- geeksforgeeks

Majority Number II

题目描述

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

解题方法

注意这里要的是所有的结果，而不是某一个数

brute force

hashmap, count

消去法

`N/3`，最多有两个结果。

- 模仿moore's voting algorithm,用两个变量和counter记录
- 如果当前值等于其中某一个，就将对应的counter + 1
- 如果某一个counter为0了，就将对应的变量设为当前值，counter设为1
- 如果当前num都不等于就两个counter都减去1
- 当任一一个counter为0时就将变量设为当前的num
- 最后剩下的两个value还要再loop一遍得到次数来验证是否是majority number
    - 不要忘了验证
    - 不要忘了验证
    - 不要忘了验证（重要的事情说三遍）

注意点

- 先判断等于，再判断counter，这样可以避免两个变量变成相同的数

Solution

```python
class Solution(object):
    def majorityElement(self, nums):
        """
        :type nums: List[int]
        :rtype: List[int]
        """
        length = len(nums)
        if length == 0:
            return []
        if length == 1:
            return nums
        if length == 2:
            if nums[0] == nums[1]:
                return [nums[0]]
            else:
                return nums

        a, b = None, None
        countA, countB = 0, 0
        for num in nums:
            if num == a:
                countA += 1
            elif num == b:
                countB += 1
            elif countA == 0:
                a = num
                countA = 1
            elif countB == 0:
                b = num
                countB = 1
            else:
                countA -= 1
                countB -= 1

        countA = 0
        countB = 0
        for num in nums:
            if num == a:
                countA += 1
            elif num == b:
                countB += 1
        if countA > length / 3.0 and countB > length / 3.0:
            return [a, b]
        elif countA > length / 3.0:
            return [a]
        elif countB > length / 3.0:
            return [b]
        else:
            return []
```

Reference

Majority Number III

题目描述

Given an array of integers and a number k, the majority number is the number that occurs more than `1/k of the size of the array.

解题方法

brute force

hashmap记录次数

消去法

思路是类似的，但是这里可以不要用 `k-1` 个变量和counter，可以用一个hashtable来记录变量和counter。

```
for num in nums:
    if num in counters:
        counters[num] += 1
    else:
        counters[num] = 1
    if len(counters.keys()) > :
        self.removeKey(counters)
```

这里 `ermoveKey()` 是将counters里所有的number都减去1，然后将number为0的value删掉

最后再验证一下

Solution

```
class Solution:
    """
    @param nums: A list of integers
    @param k: As described
    @return: The majority number
    """
    def majorityNumber(self, nums, k):
        # write your code here
        counters = {}
        for num in nums:
            if num in counters:
                counters[num] += 1
            else:
                counters[num] = 1
            if len(counters.keys()) > k:
                self.removeKey(counters)

        #recalculate remaining num occurances
        for num in counters:
            counters[num] = 0
        for num in nums:
            if num in counters:
                counters[num] += 1

        maxCounter = 0
        maxKey = 0
        for num in counters:
            if counters[num] > maxCounter:
                maxCounter = counters[num]
                maxKey = num

        return maxKey

    def removeKey(self, counters):
        l = []
        for num in counters:
            counters[num] -= 1
            if counters[num] == 0:
                l.append(num)
        for num in l:
            del counters[num]
```

Reference

Subarray类型的题目

题目列表

- maximum subarray sum
- maximum subarray sum II
- minimum subarray
- maximum product subarray
- product of array except itself
- Continuous Subarray Sum
- Continuous Subarray Sum II

问题描述

Reference

Maximum Subarray Sum

题目描述

Find the contiguous subarray within an array (containing at least one number) which has the largest sum.

For example, given the array [−2,1,−3,4,−1,2,1,−5,4], the contiguous subarray [4,−1,2,1] has the largest sum = 6.

解题方法

brute force

O(n^2)

DP

`local[i]` 表示包含当前index为i的值的最大sum `global[i]` 表示在index为i的之前的最大sum（不一定包含i）

no dp

其实可以不用DP， 只用一个variable保存当前的最大值，如果小于0，那么在下一个 element那里就可以舍去之前的

Solution

```
class Solution(object):
    def maxSubArray(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        if not nums:
            return 0
        length = len(nums)
        localDP = [-sys.maxint for i in range(length)]
        globalDP = [-sys.maxint for i in range(length)]

        localDP[0] = nums[0]
        globalDP[0] = nums[0]
        result = nums[0]

        for i in range(1, length):
            localDP[i] = max(localDP[i-1] + nums[i], nums[i])
            globalDP[i] = max(localDP[i], globalDP[i-1])
            if globalDP[i] > result:
                result = globalDP[i]

        return result
```

Reference

Maximum Product Subarray

题目描述

Find the contiguous subarray within an array (containing at least one number) which has the largest product.

For example, given the array `[2,3,-2,4]`, the contiguous subarray `[2,3]` has the largest `product = 6`.

解题方法

brute force

两个loop遍历所有subarray, O(n^2)

dp

- 因为最大值有可能由前一个的 `最大值/最小值 * 当前值` 得到，所以要有两个array来 保存每个Index的最大值和最小值
- 并且要与当前值比较，有可能只保留当前值

Solution

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

        minDP = [sys.maxint for i in range(length)]
        maxDP = [sys.maxint for i in range(length)]

        minDP[0] = nums[0]
        maxDP[0] = nums[0]
        maxProduct = nums[0]

        for i in range(1, length):
            minDP[i] = min(min(minDP[i-1] * nums[i], maxDP[i-1] * n
            maxDP[i] = max(max(minDP[i-1] * nums[i], maxDP[i-1] * n
            maxProduct = maxDP[i] if maxDP[i] > maxProduct else max

        return maxProduct
```

Reference

Product of Array Except Self

题目描述

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

Solve it without division and in O(n).

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

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

解题方法

Two array

两个array `left` 和 `right` , 分别存从左到i，从右到i的乘积（不包括i）

时间复杂度 `O(n)` 空间复杂度 `O(n)`

空间复杂度 `O(n)`

作为result的array不算空间，其实可以讲上一个解法中的从左到右，从右到左在同一个array进行

Solution

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

        length = len(nums)
        if length == 1:
            return 0
        result = [0 for i in range(length)]

        #from left to right
        #for first element, to make the value 1 now for easy multip
        result[0] = 1
        for i in range(1, length):
            result[i] = result[i-1] * nums[i-1]

        #from right to left
        #cause we cannot use the value already in the array to repr
        #so we use a variable to keep the value
        right = 1
        for i in range(length - 1, -1, -1):
            result[i] *= right
            right *= nums[i]

        return result
```

Reference

- programcreek

Backtracking

题目列表

- Combination
  - comb sum
- subset
  - generate parenthese
  - letter comb of phone number
- permutation
  - next permutation
  - previous permutation
  - permutation sequence
  - permutation index
- N queen
- sudoku solver
- valid sudoku
- word search
- flip game
- nim game

问题描述

backtracking采用试错的方法，尝试分步的去解决一个问题。在分步解决问题的 过程中，当它发现尝试发现现有的分布答案不能得到有效的正确解答时，就会取消上一步或者 上几步的计算。再通过其他可能尝试寻找。

最典型的就是**N queen**问题

Reference

全排列问题

题目列表

类型1

- subsets
- subset II
- permutations
- permutations II
- Combinations
- Combinations Sum
- Combinations Sum II
- N Queen
- N Queen II
- letter combination of a phone number

类型2 （基于全排列的问题）

- Next Permutation
- Previous Permutation（与next permuataion的操作类似）
- Permutation Sequence
- Permutation Index
- Permutation Index II (hard)

问题描述

排列组合的关键是顺序是否关键

- permuataions是有顺序的
- subsets是没有顺序的

Reference

- "全排列"问题系列

Subsets

题目描述

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

Note: Elements in a subset must be in non-descending order. The solution set must not contain duplicate subsets. For example, If `nums = [1,2,3]`, a solution is:

```
[
  [3],
  [1],
  [2],
  [1,2,3],
  [1,3],
  [2,3],
  [1,2],
  []
]
```

解题方法

要求 `non-descending` ,所以我们再生成subset之前应该将原list sort一下，这样 后面才可以方便的处理，包括去掉重复的数的操作也会更方便

subsets的不同顺序是相同**subsets,**这里是**non-descending**，是规定了一种顺序

将结果的顺序重新调整一下，应该是

```
[
    [],
    [1],
    [2],
    [3],
    [1,2],
    [1,3],
    [1,2,3],
    [2,3]
]
```

DFS + 递归

可以看出前后subset之间的关系，我们可以用类似dfs的方法做，对当前的List将下一个num加进去进行dfs, 得到所有的subsets,然后再pop掉，对下一个num进行DFS

iterative

其实一共有2^n个子集

```
public ArrayList<ArrayList<Integer>> subsets(int[] S) {
     ArrayList<ArrayList<Integer>> res = new  ArrayList<ArrayList<I
     res.add(new ArrayList<Integer>());
     if(S == null || S.length == 0)
        return res;
    Arrays.sort(S);
    for(int i=0;i<S.length;i++)
    {
        int size = res.size();
        for(int j=0;j<size;j++)
        {
            ArrayList<Integer> item = new ArrayList<Integer>(res.ge
            item.add(S[i]);
            res.add(item);
        }
    }
    return res;
}
```

Solution

```
class Solution(object):
    def subsets(self, nums):
        """
        :type nums: List[int]
        :rtype: List[List[int]]
        """
        results = []
        if not nums:
            return [[]]
        nums.sort()
        self.subsetsHelper(results, nums, 0, [])
        return results

    def subsetsHelper(self, results, nums, index, curList):
        tmpList = list(curList)
        results.append(tmpList)

        for i in range(index, len(nums)):
            curList.append(nums[i])
            #dfs
            self.subsetsHelper(results, nums, i + 1, curList)
            #pop, dfs for next number
            curList.pop()
```

Reference

- 两种解法
- code ganker

Subsets II

题目描述

Given a collection of integers that **might contain duplicates**, nums, return all possible subsets.

Note: Elements in a subset must be in non-descending order. The solution set must not contain duplicate subsets. For example, If `nums = [1,2,2]`, a solution is:

```
[
  [2],
  [1],
  [1,2,2],
  [2,2],
  [1,2],
  []
]
```

解题方法

与subsets 1相比，要去掉重复的subsets,因为我们已经sort过，所以可以方便地去掉。 如果当前的number在之前已经用过dfs过，那么当前的dfs结果一定被包含了，所以可以直接跳过。

Solution

```
class Solution(object):
    def subsetsWithDup(self, nums):
        """
        :type nums: List[int]
        :rtype: List[List[int]]
        """
        results = []
        if not nums:
            return [[]]
        nums.sort()
        self.subsetsHelper(results, nums, 0, [])
        return results

    def subsetsHelper(self, results, nums, index, curList):
        tmpList = list(curList)
        results.append(tmpList)

        for i in range(index, len(nums)):
            if i != index and nums[i] == nums[i-1]:
                continue
            curList.append(nums[i])
            self.subsetsHelper(results, nums, i + 1, curList)
            curList.pop()
```

Reference

Permutations

题目描述

Given a collection of numbers, return all possible permutations.

解题方法

因为没有non-descending这种限制，所以之前的数其实也可以用，所以不需要Index来track。 为了避免同一个数被重复用，用另一个 `user` array来保存用过的index。

permutations的不同顺序是不同**permutations**

Solution

```
class Solution(object):
    def permute(self, nums):
        """
        :type nums: List[int]
        :rtype: List[List[int]]
        """
        results = []
        if not nums:
            return results
        length = len(nums)
        nums.sort() #not necessary in this question, but good habit
        used = [False for i in range(length)]
        self.permuteHelper(nums, used, results, [])
        return results

    def permuteHelper(self, nums, used, results, curList):
        if len(curList) == len(nums):
            tmpList = list(curList)
            results.append(tmpList)

        for i in range(len(nums)):
            if not used[i]:
                curList.append(nums[i])
                used[i] = True
                self.permuteHelper(nums, used, results, curList)
                curList.pop()
                used[i] = False
```

Reference

Permutation II

题目描述

解题方法

Solution

```
class Solution(object):
    def permuteUnique(self, nums):
        """
        :type nums: List[int]
        :rtype: List[List[int]]
        """
        results = []
        if not nums:
            return results
        length = len(nums)
        nums.sort()
        used = [False for i in range(length)]
        self.permuteHelper(nums, used, results, [])
        return results

    def permuteHelper(self, nums, used, results, curList):
        if len(curList) == len(nums):
            tmpList = list(curList)
            results.append(tmpList)

        for i in range(len(nums)):
            if i != 0 and nums[i] == nums[i-1] and used[i-1] == Fal
                #当i不等于0, nums[i] == nums[i-1]还不够成跳过条件
                #如果前一个number已经用在这个list里, 是应该保留这个结果的
                #只有当前一个number没有被用, 如果再用这个重复的数, 就会dupl
                #所以加上 used[i-1] == False
                continue
            if not used[i]:
                curList.append(nums[i])
                used[i] = True
                self.permuteHelper(nums, used, results, curList)
                curList.pop()
                used[i] = False
```

Reference

Combinations

题目描述

Given two integers n and k, return all possible combinations of k numbers out of 1 ... n.

For example, If `n = 4` and `k = 2`, a solution is:

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

解题方法

对于combination，不同顺序是同一combination,所以应该采用subsets的方法， 在途中判断当前list的长度

Solution

```
class Solution(object):
    def combine(self, n, k):
        """
        :type n: int
        :type k: int
        :rtype: List[List[int]]
        """
        results = []
        if not n:
            return []
        self.subsetsHelper(results, n, k, 1, [])
        return results

    def subsetsHelper(self, results, n, k, index, curList):
        if len(curList) == k:
            tmpList = list(curList)
            results.append(tmpList)

        for i in range(index, n + 1):
            curList.append(i)
            self.subsetsHelper(results, n, k, i + 1, curList)
            curList.pop()
```

Reference

Combination Sum

题目描述

Given a set of candidate numbers (C) and a target number (T), find all unique combinations in C where the candidate numbers sums to T.

The same repeated number may be chosen from C **unlimited number of times**.

Note:

- All numbers (including target) will be **positive** integers.
- Elements in a combination (a1, a2, … , ak) must be in **non-descending** order. (ie, a1 ≤ a2 ≤ … ≤ ak).
- The solution set must not contain duplicate combinations.

For example, given candidate set `2,3,6,7` and target `7,

A solution set is:

```
[7]
[2, 2, 3]
```

解题方法

- 有顺序，non-descending,所以应该用subset那种方法
- 可以重复使用，所以DFS时应该继续包括当前的number
- 都是Positive,所以当当前List的和已经大于target,就可以结束了
- 重复依然要去除

Solution

```
class Solution(object):
    def combinationSum(self, candidates, target):
        """
        :type candidates: List[int]
        :type target: int
        :rtype: List[List[int]]
        """
        results = []
        if not candidates:
            return [[]]
        candidates.sort()
        self.combHelper(results, candidates, target, 0, [])
        return results

    def combHelper(self, results, candidates, target, index, curLis
        if sum(curList) == target:
            tmpList = list(curList)
            results.append(tmpList)
        elif sum(curList) > target:
            return

        for i in range(index, len(candidates)):
            if i != index and candidates[i] == candidates[i-1]:
                continue
            curList.append(candidates[i])
            self.combHelper(results, candidates, target, i, curList
            curList.pop()
```

Reference

Combination Sum II

题目描述

Given a collection of candidate numbers (C) and a target number (T), find all unique combinations in C where the candidate numbers sums to T.

Each number in C may only be used **once** in the combination.

Note:

- All numbers (including target) will be positive integers.
- Elements in a combination (a1, a2, … , ak) must be in non-descending order. (ie, a1 ≤ a2 ≤ … ≤ ak).
- The solution set must not contain duplicate combinations.

解题方法

唯一差别就是每个数只能用一次，所以直接dfs recursive call的时候index + 1就可以了

Solution

```
class Solution(object):
    def combinationSum2(self, candidates, target):
        """
        :type candidates: List[int]
        :type target: int
        :rtype: List[List[int]]
        """
        results = []
        if not candidates:
            return [[]]
        candidates.sort()
        self.combHelper(results, candidates, target, 0, [])
        return results

    def combHelper(self, results, candidates, target, index, curLis
        if sum(curList) == target:
            tmpList = list(curList)
            results.append(tmpList)
        elif sum(curList) > target:
            return

        for i in range(index, len(candidates)):
            if i != index and candidates[i] == candidates[i-1]:
                continue
            curList.append(candidates[i])
            self.combHelper(results, candidates, target, i+1, curLi
            curList.pop()
```

Reference

Combination Sum III

题目描述

Find all possible combinations of k numbers that add up to a number n, given that only numbers from 1 to 9 can be used and each combination should be a unique set of numbers.

Ensure that numbers within the set are sorted in ascending order.

Example 1:

Input: `k = 3, n = 7`

Output:

```
[[1,2,4]]
```

Example 2:

Input: `k = 3, n = 9`

Output:

```
[[1,2,6], [1,3,5], [2,3,4]]
```

解题方法

与 `combinations` 一样，不过要同时检查 `长度` 和 `sum`

Solution

```
class Solution(object):
    def combinationSum3(self, k, n):
        """
        :type k: int
        :type n: int
        :rtype: List[List[int]]
        """
        results = []
        if k == 0 or n == 0:
            return []
        self.combHelper(k, n, results, 1, [])
        return results

    def combHelper(self, k, n, results, index, curList):
        if len(curList) == k and sum(curList) == n:
            tmpList = list(curList)
            results.append(tmpList)
        if len(curList) > k:
            return
        if sum(curList) > n:
            return

        for i in range(index, 10):
            curList.append(i)
            self.combHelper(k, n, results, i+1, curList)
            curList.pop()
```

Reference

Letter Combinations of a Phone Number

题目描述

Given a digit string, return all possible letter combinations that the number could represent.

A mapping of digit to letters (just like on the telephone buttons) is given below.

Input:Digit string `"23"`

Output: `["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"]`.

解题方法

这个题目也是DFS的题，因为这里的电话号码是有顺序的，所以是subsets的问题

- 用一个array记录可以替换的characters
- 然后的code就与subsets的模板基本一样了

Solution

```
class Solution(object):
    def __init__(self):
        self.dtcMap = ["0", "1", "abc", "def", "ghi", "jkl", "mno",

    def letterCombinations(self, digits):
        """
        :type digits: str
        :rtype: List[str]
        """
        if not digits:
            return []

        results = []
        self.helper(results, digits, 0, [])

        return results

    def helper(self, results, digits, index, curList):
        if len(curList) == len(digits):
            newStr = "".join(curList)
            results.append(newStr)

        for i in range(index, len(digits)):
            cOptions = self.dtcMap[int(digits[i])]
            for j in range(len(cOptions)):
                curList.append(cOptions[j])
                self.helper(results, digits, i+1, curList)
                curList.pop()
```

Reference

Generate Parentheses

题目描述

Given n pairs of parentheses, write a function to generate all combinations of well-formed parentheses.

For example, given `n = 3`, a solution set is:

`"((()))", "(()())", "(())()", "()(())", "()()()"`

解题方法

一开始没有什么头绪，但是做到一道也是parentheses的题目，是valid parentheses,是将`(`存在stack来解决。 大概想到的思路是在从开头开始的string里，`(`的数量都应该比`)`的数量多，而且可以一直加入`(`。所以如果我们 从左往右构建的话，就应该遵循这种规则，直到用完所有的pair.

pair的数量为`n`，那么从左往右的过程中，只有两种选择，`(`或者`)`, 加入哪种的选择规则是

- 如果`(`的数量小于`n`,就还可以加入`(`
- 如果`(`的数量大于`)`, 就可以加入`)`

而为了产生所有的组合，而且这是有顺序的，就应该像subsets那样用dfs的方法做

Solution

```
class Solution(object):
    def generateParenthesis(self, n):
        """
        :type n: int
        :rtype: List[str]
        """
        if not n:
            return []

        results = []
        self.helper(n, results, 0, 0, [])
        return results

    def helper(self, n, results, leftP, rightP, curList):
        if len(curList) == n * 2:
            newStr = "".join(curList)
            results.append(newStr)

        if leftP < n:
            curList.append('(')
            self.helper(n, results, leftP+1, rightP, curList)
            curList.pop()

        if leftP > rightP:
            curList.append(')')
            self.helper(n, results, leftP, rightP + 1, curList)
            curList.pop()
```

Reference

- generate parentheses

Next Permutation

题目描述

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

If such arrangement is not possible, it must rearrange it as the lowest possible order (ie, sorted in ascending order).

The replacement must be in-place, do not allocate extra memory.

Here are some examples. Inputs are in the left-hand column and its corresponding outputs are in the right-hand column.

```
1,2,3 → 1,3,2
3,2,1 → 1,2,3
1,1,5 → 1,5,1
```

解题方法

- 从右往左，找出第一个递减的数，记它的index为i
- 在i的右边，找到比它value大的最小值，index记为j，将i和j swap
- 将[i+1:]的array reverse,变成Increasing sequence

Solution

```
class Solution(object):
    def nextPermutation(self, nums):
        """
        :type nums: List[int]
        :rtype: void Do not return anything, modify nums in-place i
        """
        if not nums:
            return
        length = len(nums)
        if length == 1:
            return
        #from right to left, find first descending number
        i = length - 2
        while i >= 0:
            if nums[i] < nums[i+1]:
                break
            i -= 1
        if i == -1:
            #no descending number, it's already largest
            nums.reverse()
            return

        j = i + 1
        while j < length - 1:
            if nums[j] > nums[i] and nums[j+1] <= nums[i]:
                break
            j += 1
        #swap i and j
        tmp = nums[j]
        nums[j] = nums[i]
        nums[i] = tmp
        nums[i+1:length] = nums[length-1:i:-1] #python的这个方法很好
```

Reference

Permutation Sequence

题目描述

The set `[1,2,3,…,n]` contains a total of n! unique permutations.

By listing and labeling all of the permutations in order, We get the following sequence (ie, for n = 3):

```
"123"
"132"
"213"
"231"
"312"
"321"
```

Given n and k, return the kth permutation sequence.

Note: Given n will be between 1 and 9 inclusive.

解题方法

这个也是一个排列问题，根据排列问题的规律和总结， n个数会有 `n!` 个排列，对于第一位的数字有 `n` 种选法，之后的 `n-1` 位 的排列有 `(n-1)!` 种，根据这几条规律我们就可以确定第一位数字，后面的数字也同理求得。

同一个数字不能重复用，所以我们应该用一个array来保存用过的数或者未用过的数，为了方便选择下一个number，我们保存 未用过的数，通过对应的index找到下一个该用的数，同时要记得将用过的数删除。

k应该减去1来计算，比如n=3,

```
1. "123"
2. "132"
```

他们 `/ (3-1)!` 的结果是不一样的，但是他们前面的数都应该是1，对于每一个 `(n-1)!` 的组合，他们的结果应该一样，所以直接减去1 变为0-based比较好计算

注意点

- 用一个array保存未用的数
- 记得删去用过的数
- k要减去1更方便计算

Solution

```
class Solution(object):
    def getPermutation(self, n, k):
        """
        :type n: int
        :type k: int
        :rtype: str
        """
        if not n:
            return ""
        result = ""
        product = 1
        nums = [] #use an array to keep all remaining potential nur
        for i in range(1, n+1):
            nums.append(i)
            product *= i
        if k > product:
            return ""
        k = k - 1

        for j in range(n, 0, -1):
            product /= j
            index = k / product
            result += str(nums[index])
            del nums[index] #should delete from numbers array
            k = k % product

        return result
```

Reference

Permutation Index

题目描述

Given a permutation which contains no repeated number, find its index in all the permutations of these numbers, which are ordered in lexicographical order. The index begins at 1.

解题方法

例如给定一个排列 356421， 因为第一位为3，因此1 和 2 开头的全排列已经经过了，以1开头的全排列个数为5!，2也是。因此该全排列的排名 > 2 * 5!

第二位为5，对于以3开头的全排列，排在35前面的有31,32,34开头的三个全排列。在356421中，5右边比5小的也正是1,2,4。

我们可以发现：序列长度为n，对于给定排列P某位上的数，假设这个数在P上从右起排第m位，我们只要看看该数右侧的位数上还有几个比它小的，就知道该数以右的部分在对应所有子序列中的排名了。

Solution

Reference

Permutation Index II

题目描述

Given a permutation which **may contain repeated numbers**, find its index in all the permutations of these numbers, which are ordered in lexicographical order. The index begins at 1.

解题方法

像排列组合的去重一样，要除去每个重复数字次数的阶乘，所以对于每一个element i,在数右边比它小的数的个数的时候， 还要用一个hashmap来存每个数字出现过的次数

Solution

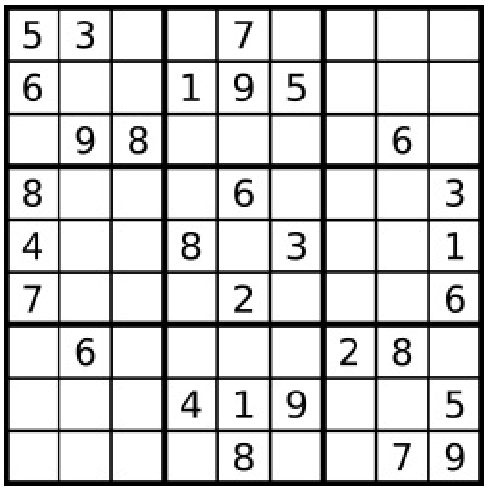
Reference

Valid Sudoku

题目描述

Determine if a Sudoku is valid, according to: Sudoku Puzzles - The Rules.

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



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

A partially filled sudoku which is valid.

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

解题方法

首先要把sudoku的规则记住

- 9行9列
- 每行只能有1-9，不能重复
- 每列只能有1-9，不能重复
- 有9个小的3*3的sub-box, 它们中也只能有1-9，不能重复

解题方法

- 用三个2-D array保存row, column, sub sodoku里面已经用过的value
- 遍历每一个格子，检查是否valid

注意点

- 建立的array是0-based的，所以1-9的数字放在array里是要减去1
- sub sodoku位置的计算，可以规定排列为

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

那么根据row和col得到的所属的sub sodoku公式就是 `(row / 3) * 3 + col / 3`

Solution

```
class Solution(object):
    def isValidSudoku(self, board):
        """
        :type board: List[List[str]]
        :rtype: bool
        """
        if not list:
            return False
        rowValid = [[False for i in range(9)] for j in range(9)]
        colValid = [[False for i in range(9)] for j in range(9)]
        subValid = [[False for i in range(9)] for j in range(9)]

        row = len(board)
        col = len(board[0])
        for i in range(row):
            for j in range(col):
                if board[i][j] == ".":
                    continue
                value = int(board[i][j]) - 1
                if rowValid[i][value]:
                    return False
                if colValid[j][value]:
                    return False
                subNum = (i / 3) * 3 + j / 3
                if subValid[subNum][value]:
                    return False
                rowValid[i][value] = True
                colValid[j][value] = True
                subValid[subNum][value] = True
        return True
```

Reference

Sudoku Solver

题目描述

Write a program to solve a Sudoku puzzle by filling the empty cells.

Empty cells are indicated by the character '.'.

You may assume that there will be only one unique solution.

解题方法

这一题大概思路就是跟permutaion, subset差不多的backtracking的方法

- 对每一个 `.` 的地方，尝试着填入1-9其中的某个数，然后再把这个暂时的结果传入下一个recursive call来计算，
- 判断是否有正确的解，如果没有就去掉填入的信息，继续用下一个数试
- 在check是否可以填入时可以用valid sodoku的方法优化，记录下现在行，列，和小九宫格里已经用过的数，遇到就可以直接跳过

`172ms`

Solution

```
class Solution(object):
    def solveSudoku(self, board):
        """
        :type board: List[List[str]]
        :rtype: void Do not return anything, modify board in-place
        """
        if not board:
            return False
        rowValid = [[False for i in range(9)] for j in range(9)]
        colValid = [[False for i in range(9)] for j in range(9)]
        subValid = [[False for i in range(9)] for j in range(9)]
        # 先保存下已经存在的数字
        for i in range(9):
            for j in range(9):
                if board[i][j] != ".":
                    subNum = (i/3)*3 + j / 3
                    rowValid[i][int(board[i][j])-1] = True
                    colValid[j][int(board[i][j])-1] = True
                    subValid[subNum][int(board[i][j])-1] = True

        self.helper(board, rowValid, colValid, subValid, 0, 0)

    def helper(self, board, rowValid, colValid, subValid, row, col
        i = row
        j = col
        while i < 9:
            if j > 8:
                #已经到了行尾，该进入下一行了
                j = 0
                i += 1
            else:
                subNum = (i/3) * 3 + j / 3
                if board[i][j] == ".":
                    for k in range(9):
                        if not rowValid[i][k] and not colValid[j][
                            board[i][j] = str(k+1)
                            rowValid[i][k] = True
                            colValid[j][k] = True
                            subValid[subNum][k] = True
                            if j < 8:
                                if self.helper(board, rowValid, co
                                    return True
                            else:
                                if self.helper(board, rowValid, co
                                    return True
                            board[i][j] = "."
                            rowValid[i][k] = False
                            colValid[j][k] = False
                            subValid[subNum][k] = False
                    return False
                else:
                    j += 1
        return True
```

Reference

- solve sodoku

Flip Game

题目描述

You are playing the following Flip Game with your friend: Given a string that contains only these two characters: + and -, you and your friend take turns to flip two consecutive `"++"` into `"--"`. The game ends when a person can no longer make a move and therefore the other person will be the winner.

Write a function to compute all possible states of the string after one valid move.

For example, given `s = "++++"`, after one move, it may become one of the following states:

```
[
  "--++",
  "+--+",
  "++--"
]
```

If there is no valid move, return an empty list `[]`.

解题方法

1比较简单，只要找到"++"，将它们被替代后的结果加到List里就可以。

Solution

```
class Solution(object):
    def generatePossibleNextMoves(self, s):
        """
        :type s: str
        :rtype: List[str]
        """
        length = len(s)
        if length == 0 or length == 1:
            return []
        result = []
        for i in range(length-1):
            if s[i:i+2] == "++":
                newStr = s[:i] + "--" + s[i+2:]
                result.append(newStr)
        return result
```

Reference

Flip Game II

题目描述

You are playing the following Flip Game with your friend: Given a string that contains only these two characters: + and -, you and your friend take turns to flip two consecutive `"++"` into `"--"`. The game ends when a person can no longer make a move and therefore the other person will be the winner.

Write a function to determine if the starting player can guarantee a win.

For example, given s = "++++", return true. The starting player can guarantee a win by flipping the middle `"++"` to become `"+--+"`.

Follow up:

Derive your algorithm's runtime complexity.

解题方法

这一题有点像 `Lintcode` 上的 `coins in a line` ，也是判断先行动的人是否能够确保最后胜利的题目，所以应该也可以用DP解。 但是这里backtracking的解法更直观，只不过之间复杂度比较高，是 `O(n^2)`

recursive

将 `canWin(self, s)` 看为当start string为 `s` 的时候第一个行动的人是否能确保赢，那么能够确保赢的条件就是，当有一个"++"被替换后 的 `s1` , 当start string为 `s1` 的时候先行动的人确保会输（这时候先行动的是第二个人了）。对于 `s` 中的所有 `++` 都尝试一下，递归就能够 找到结果。

DP

Solution

```
class Solution(object):
    def canWin(self, s):
        """
        :type s: str
        :rtype: bool
        """
        if not s:
            return False

        for i in range(len(s)-1):
            if s[i:i+2] == "++" and not self.canWin(s[:i] + "--" +
                return True

        return False
```

Reference

Nim Game

题目描述

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

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

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

Hint:

If there are 5 stones in the heap, could you figure out a way to remove the stones such that you will always be the winner?

解题方法

类似 `coins in a line I` , 找规律的题目

- 如果 `n<=3` , True
- `n == 4` , False
- 那么 `n == 5 or 6 or 7` 的时候， 都可以让下一个n为4这样自己就可以赢
- 那么 `n == 8` 时，不管取多少对手都会在5,6,7之间必胜

所以规律就是如果是4的倍数就会输

Solution

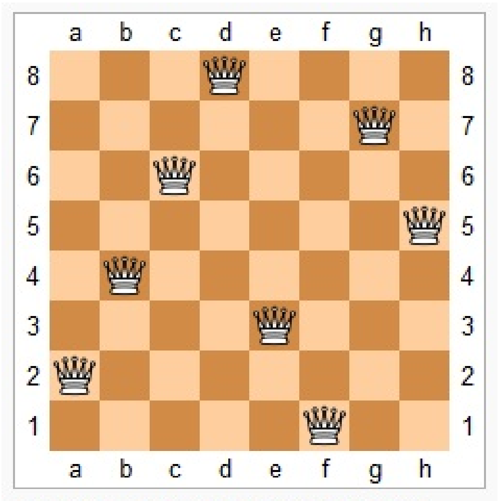
```
class Solution(object):
    def canWinNim(self, n):
        """
        :type n: int
        :rtype: bool
        """
        if not n:
            return True
        if n <= 3:
            return True

        if n % 4 == 0:
            return False
        else:
            return True
```

Reference

N Queen

题目描述

The n-queens puzzle is the problem of placing n queens on an n×n chessboard such that no two queens attack each other.



One solution to the eight queens puzzle

Given an integer n, return all distinct solutions to the n-queens puzzle.

Each solution contains a distinct board configuration of the n-queens' placement, where 'Q' and '.' both indicate a queen and an empty space respectively.

For example, There exist two distinct solutions to the 4-queens puzzle:

```
[
 [".Q..",  // Solution 1
  "...Q",
  "Q...",
  "..Q."],

 ["..Q.",  // Solution 2
  "Q...",
  "...Q",
  ".Q.."]
]
```

解题方法

N queen的规则是任意两个queen都不能攻击到彼此，而queen的攻击路线是直线或者对角线上，所以需要check的有

- 行
- 列
- 对角线

其实就类似于sudoku问题，sudoku是检查行，列和小的正方形. 这里不像sudoku那样填入到一个matrix里，当然如果我们自己 建一个matrix也是可以的，但是我们并不需要记录那么多的信息，只需要记录queen的位置，并且方便check valid就可以了。

我们可以用一个array来保存queen的信息，array的index表示行，而value就表示列，因为填入n个queen到n*n里的，其实也就是 依次每行填入一个queen,而列的选择也是在n的范围内，就变成了一个找出n的permutation的问题（这里多的条件是要满足 对角线没有queen的条件，所以要加一个check）。

- `row` array记录信息
- `used` array记录用过的column number
- 每填入一个number的时候要check对角线的条件是否满足

Solution

```
class Solution(object):
    def solveNQueens(self, n):
        """
        :type n: int
        :rtype: List[List[str]]
        """
        if not n:
            return [[]]
        row = []
        used = [False for i in range(n)]
        results = []

        self.helper(results, row, used, n)
        results = self.getMatrix(results, n)
        return results

    def helper(self, results, row, used, n):
        if len(row) == n:
            newList = list(row)
            results.append(newList)

        for j in range(n):
            if not used[j]:
                row.append(j)
                if self.checkValid(row):
                    used[j] = True
                    self.helper(results, row, used, n)
                    row.pop()
                    used[j] = False
                else:
                    row.pop()

    # check 是否斜线上有两个queen
    def checkValid(self, row):
        length = len(row)
        for i in range(length):
            for j in range(i + 1, length):
                if abs(row[i] - row[j]) == abs(i-j):
                    return False
        return True

    # 用于得到最后的".......Q."形式的结果
    def getMatrix(self, results, n):
        result = []
        for i in range(len(results)):
            string = ""
            for j in range(n):
                string += "."
            matrix = [string for j in range(n)]
            for k in range(len(results[i])):
                st = matrix[k]
                st = list(st)
                index = results[i][k]
                st[index] = "Q"
                matrix[k] = "".join(st)
            result.append(matrix)
        return result
```

Reference

Palindrome Partitioning

题目描述

Given a string s, partition s such that every substring of the partition is a palindrome.

Return all possible palindrome partitioning of `s`.

For example, given `s = "aab"`, Return

```
[
  ["aa","b"],
  ["a","a","b"]
]
```

解题方法

这种求全部组合的问题，一般都是用backtracking + dfs试错的方法来做。我们已经知道如何判断一个 string是否是palindrome,那么就不断地去backtracking去找出所有的组合就好了。

其实类似subsets,只不过这里结果里的每一部分是一个palindrome，所以要加上截取palindrome的部分。

Solution

```python
class Solution(object):
    def partition(self, s):
        """
        :type s: str
        :rtype: List[List[str]]
        """
        results = []
        self.helper(results, s, [])
        return results


    def helper(self, results, s, curList):
        if not s:
            newList = list(curList)
            results.append(newList)

        for i in range(len(s)):
            if self.isPalindrome(s, i):
                curList.append(s[:i+1])
                self.helper(results, s[i+1:], curList)
                curList.pop()


    def isPalindrome(self, s, i):
        start = 0
        while start < i:
            if s[start] != s[i]:
                return False
            start += 1
            i -= 1
        return True
```

Reference

Palindrome Partitioning II

题目描述

Given a string s, partition s such that every substring of the partition is a palindrome.

Return the minimum cuts needed for a palindrome partitioning of s.

For example, given `s = "aab"`, Return 1 since the palindrome partitioning `["aa","b"]` could be produced using 1 cut.

解题方法

DFS + backtracking

这一题用1的做法求出所有解，再找出长度最小的是可以的，不过会有很多重复计算

DP

这种找min，max的题目我们可以往DP的方向想。

1. state

   `minCut[i]` 表示 `s[:i+1]` 的最小minCut是多少

2. function

   `minCut[i] = minCut[j] + 1 if isPalindrome(s[j:i+1]) == True`

3. result

   `minCut[len(s)]`

Solution

```
class Solution(object):
    def minCut(self, s):
        """
        :type s: str
        :rtype: int
        """
        if not s:
            return

        length = len(s)

        matrix = [[False for i in range(length)] for j in range(len
        minCut = [sys.maxint for i in range(length)]
        minCut[0] = 0

        for i in range(length):
            matrix[i][i] = True

        for i in range(1, length):
            for j in range(i+1):
                if (s[i] == s[j] and i - j < 2) or (s[i] == s[j] an
                    matrix[j][i] = True
                    if j == 0:
                        minCut[i] = 0
                    else:
                        minCut[i] = min(minCut[i], minCut[j-1] + 1)

        return minCut[length-1]
```

Reference

- 喜刷刷

Palindrome Permutation

题目描述

Given a string, determine if a permutation of the string could form a palindrome.

For example, `"code"` -> `False`, `"aab"` -> `True`, `"carerac"` -> `True`.

解题方法

这一题根据提示应该按照palindrome的特征才判断，就可以达到时间 `O(n)` 的复杂度。

- 用一个hashtable来记录每个character出现的次数
- 如果string长度是偶数，那么每个char出现的次数都必须是偶数
- 如果string长度是奇数，那么必须有且仅有一个字符出现的次数是奇数

Solution

```
class Solution(object):
    def canPermutePalindrome(self, s):
        """
        :type s: str
        :rtype: bool
        """
        if len(s) == 0:
            return True

        dic = {}
        for i in range(len(s)):
            if s[i] not in dic:
                dic[s[i]] = 1
            else:
                dic[s[i]] += 1
                dic[s[i]] %= 2

        foundOdd = 0
        for key in dic:
            if dic[key] == 1:
                foundOdd += 1

        if len(s) % 2 == 0:
            if foundOdd != 0:
                return False
            else:
                return True
        else:
            if foundOdd == 1:
                return True
            else:
                return False
```

Reference

Palindrome Permutation II

题目描述

Given a string s, return all the palindromic permutations (without duplicates) of it. Return an empty list if no palindromic permutation could be form.

For example:

Given `s = "aabb"`, return `["abba", "baab"]`.

Given `s = "abc"`, return `[]`.

Hint:

- If a palindromic permutation exists, we just need to generate the first half of the string.
- To generate all distinct permutations of a (half of) string, use a similar approach from: Permutations II or Next Permutation.

解题方法

一开始没啥想法，不知道如何下手，看了提示之后顿悟，如果能够有palindrome,只需要找出前half string的所有permutation就可以了。

- 先判断是否能有palindrome,用1的方法
- 如果能够组成，找出前half string可用的所有characters(可以重复)
- permutation找出所有（如果是奇数的话注意中间的字母不可以变）

Solution

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

        canPalindrome, dic = self.canPermutePalindrome(s)
        if not canPalindrome:
            return []

        halfS = []
        for c in dic:
            for i in range(dic[c] / 2):
                halfS.append(c)

        results = []
        self.permuteUnique(results, halfS)

        if len(s) % 2 == 1:
            oddC = ""
            for c in dic:
                if dic[c] % 2 == 1:
                    oddC = c
            if not results:
                results = [oddC]
            else:
                for idx, result in enumerate(results):
                    results[idx] = result + oddC + result[::-1]
        else:
            for idx, result in enumerate(results):
                results[idx] = result + result[::-1]


        return results


    def canPermutePalindrome(self, s):
        if len(s) == 0:
            return True

        dic = {}
        for i in range(len(s)):
            if s[i] not in dic:
                dic[s[i]] = 1
            else:
                dic[s[i]] += 1

        foundOdd = 0
        for key in dic:
            if dic[key] % 2 == 1:
                foundOdd += 1

        if len(s) % 2 == 0:
            if foundOdd != 0:
                return False, dic
            else:
                return True, dic
        else:
            if foundOdd == 1:
                return True, dic
            else:
                return False, dic

    def permuteUnique(self, results, nums):
        """
        :type nums: List[int]
        :rtype: List[List[int]]
        """
        if not nums:
            return results
        length = len(nums)
        nums.sort()
        used = [False for i in range(length)]
        self.permuteHelper(nums, used, results, [])
        return results

    def permuteHelper(self, nums, used, results, curList):
        if len(curList) == len(nums):
            tmpList = "".join(curList)
            results.append(tmpList)

        for i in range(len(nums)):
            if i != 0 and nums[i] == nums[i-1] and used[i-1] == Fal
                continue
            if not used[i]:
                curList.append(nums[i])
                used[i] = True
                self.permuteHelper(nums, used, results, curList)
                curList.pop()
                used[i] = False
```

Reference

Additive Number

题目描述

Additive number is a string whose digits can form additive sequence.

A valid additive sequence should contain at least three numbers. Except for the first two numbers, each subsequent number in the sequence must be the sum of the preceding two.

For example: `"112358"` is an additive number because the digits can form an additive sequence: `1, 1, 2, 3, 5, 8.`

`1 + 1 = 2, 1 + 2 = 3, 2 + 3 = 5, 3 + 5 = 8`

`"199100199"` is also an additive number, the additive sequence is: `1, 99, 100, 199.`` ` 1 + 99 = 100, 99 + 100 = 199`

Note: Numbers in the additive sequence **cannot** have leading zeros, so sequence 1, 2, 03 or 1, 02, 3 is invalid.

Given a string containing only digits '0'-'9', write a function to determine if it's an additive number.

Follow up:

How would you handle overflow for very large input integers?

解题方法

这题根据对例子的分析，我们可以发现

- 数字的长度是不确定的
- 第一个数字和第二个数字的长度没有关系
- 如果遇到0的话它必然不是数字的开头

所以我们需要枚举所有的可能性，枚举所有的数字的可能长度，可以使用backtracking的方法，类似于permutation的解法。

而判断第三个数是否在剩下的string里，python有一个方便的method `str.startswith()`。

注意点

- 第一个数必然小于总长度的一半，否则第三个数不可能是它的和
- 结束点在第三个数正好是剩下的string
- 虽然数字开头不能是0，但是 `0` 也是一个有效数字

Solution

```python
class Solution(object):
    def isAdditiveNumber(self, num):
        """
        :type num: str
        :rtype: bool
        """
        def isValid(n):
            return len(n) == 1 or n[0] != "0"

        def dfs(a, b, c):
            d = str(int(a) + int(b))
            if not isValid(d) or not c.startswith(d):
                return False
            if c == d:
                return True
            return dfs(b, d, c[len(d):])

        length = len(num)
        if length < 3:
            return False

        for i in range(1, length/2 + 1):
            for j in range(i+1, length):
                a, b, c = num[:i], num[i:j], num[j:]
                if not isValid(a) or not isValid(b):
                    continue
                if dfs(a, b, c):
                    return True

        return False
```

Reference

- 书影博客

Two pointer

Reference

- two pointer的应用

Minimum Size Subarray Sum

题目描述

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

For example, given the array [2,3,1,2,4,3] and s = 7, the subarray [4,3] has the minimal length under the problem constraint.

Challenge If you have figured out the `O(n)` solution, try coding another solution of which the time complexity is `O(n log n)`.

解题方法

首先要问清楚面试官，如果没有解应该返回什么，比如

- [1,1], 3
- [], 2 ....

two pointer

这道题可以用two pointer解，对每一个i,将j向右移，直到sum >= s, 后面的j就不用看了，因为都是positive,所以肯定大于s

time complexity: `O(n)` space complexity: `O(1)`

二分法

这种subarray sum的题目，很多都可以采取类似的方法： 用一个sums[]， sums[i]表示nums[0:i]的sum，然后对于sums[i], 用二分法找到右边的边界，因为都是Positive number，所以sums必然是一个 递增sorted array.

Solution

```
class Solution(object):
    def minSubArrayLen(self, s, nums):
        """
        :type s: int
        :type nums: List[int]
        :rtype: int
        """
        length = len(nums)
        if length == 0:
            return 0
        p1, p2 = 0, 0
        currSum = 0
        minLen = sys.maxint
        while p1 < length:
            while currSum < s and p2 < length:
                currSum += nums[p2]
                p2 += 1
            #if currSum is larger than s, count the len
            #in case of the p2 >= length
            if currSum >= s:
                currLen = p2 - p1
                if minLen > currLen:
                    minLen = currLen
            currSum -= nums[p1]
            p1 += 1

        if minLen == sys.maxint:
            return 0
        else:
            return minLen
```

二分法

```
class Solution(object):
    def minSubArrayLen(self, s, nums):
        """
        :type s: int
        :type nums: List[int]
        :rtype: int
        """
        length = len(nums)
        if length == 0:
            return 0
        sums = [0 for i in range(length+1)]
        result = sys.maxint
        sums[1] = nums[0]
        for i in range(1, length+1):
            sums[i] = sums[i-1] + nums[i-1]

        for i in range(length):
            rightEnd = self.searchRight(i, length, sums[i] + s, sur
            if rightEnd != length + 1:
                currLen = rightEnd - i
                if currLen < result:
                    result = currLen

        if result == sys.maxint:
            return 0
        else:
            return result

    def searchRight(self, start, end, target, sums):
        while start + 1 < end:
            mid = start + (end - start) / 2
            if sums[mid] >= target:
                end = mid
            else:
                start = mid
        if sums[start] >= target:
            return start
        if sums[end] >= target:
            return end
        return len(sums) + 1
```

Valid Palindrome

题目描述

解题方法

Solution

```
class Solution(object):
    def isPalindrome(self, s):
        """
        :type s: str
        :rtype: bool
        """
        if not s:
            return True
        length = len(s)
        left = 0
        right = length - 1
        s = s.lower()
        while left < right:
            while left < right and s[left] not in "abcdefghijklmnop
                left += 1
            while left < right and s[right] not in "abcdefghijklmno
                right -= 1
            if s[left] == s[right]:
                left += 1
                right -= 1
            else:
                return False
        return True
```

Reference

Container with Most Water

题目描述

Given n non-negative integers a1, a2, ..., an, where each represents a point at coordinate (i, ai). n vertical lines are drawn such that the two endpoints of line i is at (i, ai) and (i, 0). Find two lines, which together with x-axis forms a container, such that the container contains the most water.

> Note: You may not slant the container.

解题方法

这一题跟trapping water不同，比较简单，因为它只需要找到两条竖边，然后跟x轴组成一个container，计算这个square的面积即可。 而面积是由最小的高度和两条边x轴的差决定的，用two pointer的方法从两边往中间扫就可以，每当计算完一个area的时候就将 高度小的那条边前进一个index。

Solution

```
class Solution(object):
    def maxArea(self, height):
        """
        :type height: List[int]
        :rtype: int
        """
        length = len(height)
        if not height or length < 2:
            return 0
        maxArea = 0
        left = 0
        right = length - 1
        while left < right:
            curArea = (right - left) * min(height[right],height[lef
            if curArea > maxArea:
                maxArea = curArea
            if height[right] > height[left]:
                left += 1
            else:
                right -= 1

        return maxArea
```

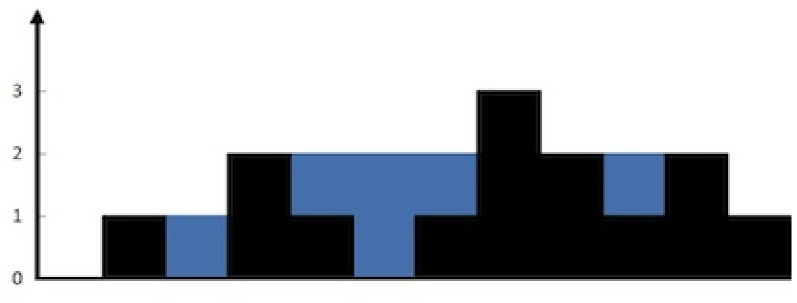
Reference

Trapping Water

题目描述

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it is able to trap after raining.

For example, Given [0,1,0,2,1,0,1,3,2,1,2,1], return 6.



解题方法

从两边往里灌水，总是从低的那一边灌，而高的那一边保持不动，同时要记录下当前可以灌到的高度 `curBar` , 及时更新

方法1

一开始直觉的方法是记录下每个位置能够灌到的位置，形成一个新的array，然后再用灌到的after的值减去原来的值

这就多了 `O(n)` 的space

方法2

其实不用新建一个array,在遍历的过程中直接将结果记录下来

Solution

第一次的方法，space `O(n)`

```
class Solution(object):
    def trap(self, height):
        """
        :type height: List[int]
        :rtype: int
        """
        if not height:
            return 0
        length = len(height)
        after = [0 for i in range(length)]
        left = 0
        right = length - 1
        curBar = min(height[left], height[right])
        while left <= right:
            curMin = min(height[left], height[right])
            if curMin > curBar:
                curBar = curMin
            after[left] = curBar
            after[right] = curBar
            if height[left] < height[right]:
                left += 1
            else:
                right -= 1

        result = 0
        for i in range(length):
            result += max(0, after[i]-height[i])
        return result
```

第二次不需要新建array的方法

```
class Solution(object):
    def trap(self, height):
        """
        :type height: List[int]
        :rtype: int
        """
        if not height:
            return 0
        length = len(height)
        left = 0
        right = length - 1
        curBar = min(height[left], height[right])
        result = 0
        while left <= right:
            curMin = min(height[left], height[right])
            if curMin > curBar:
                curBar = curMin
            result += max(0, curBar - height[left])
            result += max(0, curBar - height[right])
            if height[left] < height[right]:
                left += 1
            else:
                right -= 1


        return result
```

```
class Solution:
    # @param heights: a list of integers
    # @return: a integer
    def trapRainWater(self, heights):
        # write your code here
        if not heights:
            return 0

        p1 = 0
        p2 = len(heights) - 1
        leftBound = heights[p1]
        rightBound = heights[p2]
        result = 0

        while p1 < p2:
            if leftBound <= rightBound:
                #左边低，从左边往里灌
                p1 += 1
                if heights[p1] <= leftBound:
                    # 比现有的bound低，可以灌
                    result += leftBound - heights[p1]
                else:
                    # 比现有的bound高，需要更新左边的bound
                    leftBound = heights[p1]
            else:
                p2 -= 1
                if heights[p2] < rightBound:
                    result += rightBound - heights[p2]
                else:
                    rightBound = heights[p2]

        return result
```

Reference

Find Kth Largest Element in an Array

题目描述

Find the kth largest element in an unsorted array. Note that it is the kth largest element in the sorted order, not the kth distinct element.

For example, Given `[3,2,1,5,6,4]`` and `k = 2`, return `5`.

Note: You may assume k is always valid, `1 ≤ k ≤ array's` length.

解题方法

brute force

sort, 然后取kth largest

`O(nlogn)`

quick select

Quick select的经典应用， average case `O(n)`, worest case `O(n^2)`

要把**quick select**的方法练透，写出自己的模板

我这里直接将比pivot大的放到前面，这样直接用前面一个指针计算第n大，不用再跟 长度相减。 如果是求kth smallest,就应该将小的放前面

注意点

- pivot选择
- 大小元素的放置
- 重复元素的处理

Solution

```
class Solution(object):
    def findKthLargest(self, nums, k):
        """
        :type nums: List[int]
        :type k: int
        :rtype: int
        """
        length = len(nums)
        if length == 0:
            return
        pivot = nums[0]
        left = 0
        right = length - 1
        p = 0
        while p <= right:
            if nums[p] > pivot:
                self.swap(nums, left, p)
                p += 1
                left += 1
            elif nums[p] == pivot:
                p += 1
            else:
                self.swap(nums, right, p)
                right -= 1

        if left + 1 == k:
            return pivot
        elif left + 1< k:
            return self.findKthLargest(nums[left+1:], k - left - 1
        else:
            return self.findKthLargest(nums[:left+1], k)

    def swap(self, nums, p1, p2):
        tmp = nums[p2]
        nums[p2] = nums[p1]
        nums[p1] = tmp
```

Reference

- quickSelect的解释
- programcreek

Quick Select

说明

Quick Select是由quick sort而来，只是它只用了quick sort前面的partition的部分

Average Case `O(n)` Worst Case `O(n^2)`

模板思路

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

初始化（两个指针指向的都是下一个对应的数应该放的地方）

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

`pointer` 从0开始，遇到right结束，后面的肯定逗比pivot大

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

Reference

- quickSelect的解释
- programcreek

Find Kth Largest Element in an Array

题目描述

Find the kth largest element in an unsorted array. Note that it is the kth largest element in the sorted order, not the kth distinct element.

For example, Given `[3,2,1,5,6,4]`` and` k = 2 `, return` 5`.

Note: You may assume k is always valid, `1 ≤ k ≤ array's` length.

解题方法

brute force

sort, 然后取kth largest

`O(nlogn)`

quick select

Quick select的经典应用， average case `O(n)`, worest case `O(n^2)`

要把**quick select**的方法练透，写出自己的模板

我这里直接将比pivot大的放到前面，这样直接用前面一个指针计算第n大，不用再跟 长度相减。 如果是求kth smallest,就应该将小的放前面

注意点

- pivot选择
- 大小元素的放置
- 重复元素的处理

Solution

```python
class Solution(object):
    def findKthLargest(self, nums, k):
        """
        :type nums: List[int]
        :type k: int
        :rtype: int
        """
        length = len(nums)
        if length == 0:
            return
        pivot = nums[0]
        left = 0
        right = length - 1
        p = 0
        while p <= right:
            if nums[p] > pivot:
                self.swap(nums, left, p)
                p += 1
                left += 1
            elif nums[p] == pivot:
                p += 1
            else:
                self.swap(nums, right, p)
                right -= 1

        if left + 1 == k:
            return pivot
        elif left + 1< k:
            return self.findKthLargest(nums[left+1:], k - left - 1
        else:
            return self.findKthLargest(nums[:left+1], k)

    def swap(self, nums, p1, p2):
        tmp = nums[p2]
        nums[p2] = nums[p1]
        nums[p1] = tmp
```

Reference

- quickSelect的解释
- programcreek

Sort Color

题目描述

Given an array with n objects colored red, white or blue, sort them so that objects of the same color are adjacent, with the colors in the order red, white and blue.

Here, we will use the integers 0, 1, and 2 to represent the color red, white, and blue respectively.

解题方法

quick select 模板解决

Solution

```
class Solution(object):
    def sortColors(self, nums):
        """
        :type nums: List[int]
        :rtype: void Do not return anything, modify nums in-place i
        """
        def swap(p1, p2):
            tmp = nums[p2]
            nums[p2] = nums[p1]
            nums[p1] = tmp

        length = len(nums)
        if length == 0:
            return
        left = 0
        right = length - 1
        p = 0
        while p <= right:
            if nums[p] < 1:
                swap(left, p)
                p += 1
                left += 1
            elif nums[p] == 1:
                p += 1
            else:
                swap(right, p)
                right -= 1
```

Reference

Sort Colors II （有k种colors)

题目描述

Given an array of n objects with k different colors (numbered from 1 to k), sort them so that objects of the same color are adjacent, with the colors in the order 1, 2, ... k.

解题方法

brute force

- 先遍历一次记录下每种颜色出现的次数
- 再遍历一次将对应位置改为对应的颜色

这一题就是将sort colors的方法扩展到k种colors，我们首先应该想如何利用本来的sort color的算法

本来sort color是将3种颜色分开，其实是将两种颜色，一种放到开头，一种放到结尾。

我们如果要利用这种方法，就可以每次取出要放在开头的颜色和放在结尾的颜色，也就是Min和max。

将他们按照sort color的方法调整好后，再继续进行下面2个颜色，直到完成所有的颜色

Solution

```
class Solution:
    """
    @param colors: A list of integer
    @param k: An integer
    @return: nothing
    """
    def sortColors2(self, colors, k):
        # write your code here
        count = 0
        start = 0
        end = len(colors) - 1
        remain = list(colors)
        while count < k:
            if not remain:
                break
            minColor = min(remain)
            maxColor = max(remain)
            pColor1, pColor2 = self.sortColors(colors, minColor, ma
            start = pColor1
            end = pColor2
            count += 2
            remain = colors[start:end+1] #start和end都应该包含在下一次
        return colors

    def sortColors(self, colors, color1, color2, start, end):
        # left和right表示的是两个颜色'下一个'的位置'
        left = start
        right = end
        index = start
        while index <= right:
            if colors[index] == color1:
                colors = self.swap(colors, left, index)
                left += 1
                index += 1
            elif colors[index] == color2:
                colors = self.swap(colors, right, index)
                right -= 1
            else:
                index += 1

        return left, right


    def swap(self, nums, i, j):
        tmp = nums[j]
        nums[j] = nums[i]
        nums[i] = tmp
        return nums
```

Reference

- sort colors ii

Contains Duplicate

题目描述

解题方法

直接hashtable解决

Solution

Reference

Contains Duplicate II

题目描述

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

解题方法

- 还是hashtable，保存上一次出现的index
- 每次又遇到的时候计算与上一次的距离，再将hashtable里的value更新为新的index

Solution

```
class Solution(object):
    def containsNearbyDuplicate(self, nums, k):
        """
        :type nums: List[int]
        :type k: int
        :rtype: bool
        """
        if not nums:
            return False
        dic = {}
        for idx, num in enumerate(nums):
            if num in dic:
                dif = idx - dic[num]
                if dif <= k:
                    return True
            dic[num] = idx

        return False
```

Reference

Contains Duplicate III

题目描述

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

解题方法

从index的difference最多为k我们可以感觉有点像sliding window, 在Move的过程中增加和减少element, 而如果对于每一个value, 都要将[value-t, value + t]这个范围找一遍的话，肯定时间复杂度太高。

利用**Bucket sort**的思想

如果需要在一定等间隔范围的查找，可以利用**bucket sort**的思想减小查找比较的范围

```
如果： | nums[i] - nums[j] | <= t    式a

等价： | nums[i] / t - nums[j] / t | <= 1   式b

推出： | floor(nums[i] / t) - floor(nums[j] / t) | <= 1   式c

等价： floor(nums[j] / t) ∈ {floor(nums[i] / t) - 1, floor(nums[i] /
```

那么对于每一个Num,只要

`key = num / t`

然后搜寻(key-1, key, key + 1)是否在维护的最大size为k的map中，并且判断真实的value diff是否<= t就可以了。

快速delete开头的key/value pair

在sliding window总我们用hashheap, 在Python中有OrderedDict这个数据结构，可以用在这里

```
OrderedDict.popitem(last=True)
The popitem() method for ordered dictionaries returns and removes a
The pairs are returned in LIFO order if last is true or FIFO order
```

Solution

```
from collections import OrderedDict

class Solution(object):
    def containsNearbyAlmostDuplicate(self, nums, k, t):
        """
        :type nums: List[int]
        :type k: int
        :type t: int
        :rtype: bool
        """
        if not nums:
            return False
        if t < 0 or k < 1:
            return False
        window = OrderedDict()
        for idx, num in enumerate(nums):
            # bucket size should be at least 1
            key = num / max(1, t)
            for i in (key, key-1, key+1):
                if i in window and abs(window[i] - num) <= t:
                    return True
            window[key] = num
            if idx >= k:
                window.popitem(last=False)

        return False
```

Reference

Shortest Word Distance

题目描述

Given a list of words and two words word1 and word2, return the shortest distance between these two words in the list.

For example, Assume that `words = ["practice", "makes", "perfect", "coding", "makes"]` .

Given word1 = "coding", word2 = "practice", return 3. Given word1 = "makes", word2 = "coding", return 1.

Note: You may assume that word1 does not equal to word2, and word1 and word2 are both in the list.

解题方法

1比较简单，只需要用两个variable来记录word1和word2出现过的位置就可以，随着loop往 后，我们只需要记录最后一次出现的位置就可以得到shortest distance

Solution

```
class Solution(object):
    def shortestDistance(self, words, word1, word2):
        """
        :type words: List[str]
        :type word1: str
        :type word2: str
        :rtype: int
        """
        if not words:
            return -1
        length = len(words)
        result = sys.maxint
        index1 = -1
        index2 = -1
        for idx, word in enumerate(words):
            if word == word1:
                index1 = idx
                if index2 != -1:
                    newDis = index1 - index2
                    if newDis < result:
                        result = newDis
            elif word == word2:
                index2 = idx
                if index1 != -1:
                    newDis = index2 - index1
                    if newDis < result:
                        result = newDis

        return result
```

Reference

Shortest Word Distance 2

题目描述

This is a **follow up** of Shortest Word Distance.

The only difference is now you are given the list of words and your method will be called repeatedly many times with different parameters. How would you optimize it?

Design a class which receives a list of words in the constructor, and implements a method that takes two words word1 and word2 and return the shortest distance between these two words in the list.

解题方法

2的区别是需要不断地call, 并且每次的word也不同，所以我们肯定要记录下所有word的位置信息 以备后面使用。而当输入一个word的时候也需要快速的查询，我们可以想到用hashtable来 保存位置信息可以达到 `O(1)` 的查找时间。

因为word不一定只出现一次，所以我们hashtable的value是一个list。

两个List之间找最小distance的code有点类似merge two sorted list

Solution

```
class WordDistance(object):
    def __init__(self, words):
        """
        initialize your data structure here.
        :type words: List[str]
        """
        self.dic = {}
        for idx, word in enumerate(words):
            if word in self.dic:
                self.dic[word].append(idx)
            else:
                self.dic[word] = [idx]

    def shortest(self, word1, word2):
        """
        Adds a word into the data structure.
        :type word1: str
        :type word2: str
        :rtype: int
        """
        list1 = self.dic[word1]
        list2 = self.dic[word2]

        p1, p2 = 0, 0
        minDis = sys.maxint
        while p1 < len(list1) and p2 < len(list2):
            diff = abs(list1[p1] - list2[p2])
            minDis = minDis if minDis < diff else diff
            if list1[p1] < list2[p2]:
                p1 += 1
            else:
                p2 += 1

        return minDis
```

Reference

Shortest Word Distance 3

题目描述

This is a follow up of Shortest Word Distance. **The only difference is now word1 could be the same as word2**.

Given a list of words and two words word1 and word2, return the shortest distance between these two words in the list.

word1 and word2 may be the same and they represent two individual words in the list.

解题方法

就是多一个word1 == word2时候的查找

Solution

```
class Solution(object):
    def shortestWordDistance(self, words, word1, word2):
        """
        :type words: List[str]
        :type word1: str
        :type word2: str
        :rtype: int
        """
        dic = {}
        for idx, word in enumerate(words):
            if word in dic:
                dic[word].append(idx)
            else:
                dic[word] = [idx]


        if word1 == word2:
            list1 = dic[word1]
            minDis = sys.maxint
            for i in range(1, len(list1)):
                diff = list1[i] - list1[i-1]
                minDis = diff if minDis > diff else minDis

            return minDis

        list1 = dic[word1]
        list2 = dic[word2]

        p1, p2 = 0, 0
        minDis = sys.maxint
        while p1 < len(list1) and p2 < len(list2):
            diff = abs(list1[p1] - list2[p2])
            minDis = minDis if minDis < diff else diff
            if list1[p1] < list2[p2]:
                p1 += 1
            else:
                p2 += 1

        return minDis
```

Reference

Happy Number

题目描述

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

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

Example: `19` is a happy number

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

解题方法

- 用一个hashmap记录出现过的数
- 如果新得到的结果已经出现过，说明进入了死循环
- 否则一直计算到结果为1

Solution

```
class Solution(object):
    def isHappy(self, n):
        """
        :type n: int
        :rtype: bool
        """
        if n == 0:
            return False
        dic = {}
        while n != 1:
            tmp = self.getSquareSum(n)
            if tmp in dic:
                return False
            else:
                dic[tmp] = True
            n = tmp
        return True

    def getSquareSum(self, n):
        result = 0
        while n:
            result += (n % 10) ** 2
            n /= 10
        return result
```

Reference

Longest Substring without Repeating Characters

题目描述

Given a string, find the length of the longest substring without repeating characters. For example, the longest substring without repeating letters for "abcabcbb" is "abc", which the length is 3. For "bbbbb" the longest substring is "b", with the length of 1.

解题方法

这种类型的题目是 `hashtable` + `two pointers` 的结合

左右两个指针 `p1` , `p2` , `p1` 是window的左边界, `p2` 是window的右边界。当左边界不动时， 右边界不断向前扩展直到不能满足条件，这时候左边界就往前推进，删掉最左边的元素。

这就是基本的 `sliding window` 的思想。

这里要没有重复元素，那么最简单的就是用hashtable来判断是否出现过。

Solution

```
class Solution(object):
    def lengthOfLongestSubstring(self, s):
        """
        :type s: str
        :rtype: int
        """
        if not s:
            return 0
        length = len(s)
        hashMap = {}
        p1 = 0
        p2 = 0
        maxLength = 0
        while p1 < length:
            while p2 < length and (s[p2] not in hashMap or hashMap|
                hashMap[s[p2]] = True
                curLength = p2 - p1 + 1
                maxLength = max(curLength, maxLength)
                p2 += 1
            hashMap[s[p1]] = False
            p1 += 1

        return maxLength
```

Reference

Longest Substring with At Most Two Distinct Characters

题目描述

Given a string, find the length of the longest substring T that contains at most 2 distinct characters.

For example, Given `s = “eceba”`,

T is `"ece"` which its length is `3`.

解题方法

也是 `hashtable` + `two pointer`，这一题其实是follow up的特殊情况，应该是 with at most **k** characters， 那么我们可以用一个variable来记录出现过的 distinct characters的数量就可以了。

在遍历过程中的情况分类

- 遇到出现过的重复元素（在hashtable里），并且出现次数不为1（说明并没有被删掉），那么就加occurance
- 没有出现过，但是num of distinct chars还 `< k`，就加入
- 否则就停止right pointer继续前进，可以推进左边界并且删掉开头的元素了

Solution

```
class Solution(object):
    def lengthOfLongestSubstringTwoDistinct(self, s):
        """
        :type s: str
        :rtype: int
        """
        if not s:
            return 0
        left = 0
        right = 0
        numDis = 0
        result = 0
        cDic = {}

        for left in range(len(s)):
            while right < len(s):
                cur = s[right]
                if cur in cDic and cDic[cur] != 0:
                    # if it's not a new distinct one, increment occ
                    cDic[cur] += 1
                    result = max(result, right - left + 1)
                    right += 1
                elif numDis < 2:
                    # if num of distinct is still less than k, add
                    numDis += 1
                    cDic[cur] = 1
                    result = max(result, right - left + 1)
                    right += 1
                else:
                    break

            # remove left char occurance with for loop incrementing
            if s[left] in cDic:
                cDic[s[left]] -= 1
                if cDic[s[left]] == 0:
                    numDis -= 1

        return result
```

Reference

- program creek

Minimum Window Substring

题目描述

Given a string S and a string T, find the minimum window in S which will contain all the characters in T in complexity O(n).

For example,

```
S = "ADOBECODEBANC"
T = "ABC"
```

Minimum window is `"BANC"`.

- sliding window maximum

解题方法

类似longest substring类型的题目，这种题目的code结构我应该记下来，类似的题目都可以用 这种模板来解。

Solution

```
class Solution(object):
    def minWindow(self, s, t):
        """
        :type s: str
        :type t: str
        :rtype: str
        """
        if not s or not t:
            return ""

        tMap = {}
        for c in t:
            if c in tMap:
                tMap[c] += 1
            else:
                tMap[c] = 1

        left = 0
        right = 0
        numT = len(tMap.keys())
        minLen = sys.maxint
        result = ""

        for left in range(len(s)):
            while left <= right and right < len(s) and numT > 0:
                if s[right] in tMap:
                    tMap[s[right]] -= 1
                    if tMap[s[right]] == 0:
                        numT -= 1
                right += 1 # don't forget

            if numT == 0:
                curLen = right - left # be carefult of the bound
                if curLen < minLen:
                    minLen = curLen
                    result = s[left: right]
            if s[left] in tMap:
                tMap[s[left]] += 1
                if tMap[s[left]] == 1:
                    numT += 1

        return result
```

Reference

Substring with Concatenation of All Words

题目描述

You are given a string, s, and a list of words, words, that are all of the same length. Find all starting indices of substring(s) in s that is a concatenation of each word in words exactly once and without any intervening characters.

For example, given: s: `"barfoothefoobarman"` words: `["foo", "bar"]`

You should return the indices: [0,9].

解题方法

Solution

```
class Solution(object):
    def findSubstring(self, s, words):
        """
        :type s: str
        :type words: List[str]
        :rtype: List[int]
        """
        if not s or not words:
            return
        result = []

        wMap = {}
        # word出现的次数也要统计
        for word in words:
            if word in wMap:
                wMap[word] += 1
            else:
                wMap[word] = 1
        found = {}
        wLength = len(words[0])
        numWord = len(words)

        left = 0
        for left in range(len(s) - wLength * numWord + 1):
            found = {}
            k = 0
            # 最长check numWord个word的长度
            while k < numWord:
                wStart = left + k * wLength
                word = s[wStart: wStart + wLength]
                if word not in wMap:
                    # this word is not in the list
                    break
                else:
                    if word in found:
                        found[word] += 1
                    else:
                        found[word] = 1
                    # 如果次数多于原来的，说明不对了就break
                    if found[word] > wMap[word]:
                        break
                k += 1
            if k == numWord:
                result.append(left)

        return result
```

Reference

Palindrome Permutation

题目描述

Given a string, determine if a permutation of the string could form a palindrome.

For example, `"code" -> False, "aab" -> True, "carerac" -> True.`

解题方法

这一题写出几个例子就会发现规律，因为是Permutation，所以characters都可以任意排列。

- 首先看这个string的长度是奇数还是偶数
  - 如果是奇数，那必然是两边对称，中间有一个独特的character
  - 如果是偶数，就应该是两边对称
- 然后数每个character出现的次数
  - 如果长度是奇数，那么出了一个出现了奇数次的数外，其他的数都应该出现偶数次
  - 如果长度是偶数，那么所有数都应该出现偶数次

优化，注意点

- 为了便于判断奇偶，我将所有的出现次数都 % 2

Solution

```python
class Solution(object):
    def canPermutePalindrome(self, s):
        """
        :type s: str
        :rtype: bool
        """
        if len(s) == 0:
            return True

        dic = {}
        for i in range(len(s)):
            if s[i] not in dic:
                dic[s[i]] = 1
            else:
                dic[s[i]] += 1
                dic[s[i]] %= 2

        foundOdd = 0
        for key in dic:
            if dic[key] == 1:
                foundOdd += 1

        if len(s) % 2 == 0:
            if foundOdd != 0:
                return False
            else:
                return True
        else:
            if foundOdd == 1:
                return True
            else:
                return False
```

Reference

Group Shift String

题目描述

Given a string, we can "shift" each of its letter to its successive letter, for example: "abc" -> "bcd". We can keep "shifting" which forms the sequence:

`"abc" -> "bcd" -> ... -> "xyz"` Given a list of strings which contains only lowercase alphabets, group all strings that belong to the same shifting sequence.

For example, given: `["abc", "bcd", "acef", "xyz", "az", "ba", "a", "z"]` ,

Return:

```
[
  ["abc","bcd","xyz"],
  ["az","ba"],
  ["acef"],
  ["a","z"]
]
```

Note: For the return value, each inner list's elements must follow the lexicographic order.

解题方法

首先最明显的是按长度分，然后我们观察几组string的变化，可以发现同一组的string他们string之间的间隔是一样的， 而如何记录每个字符之间的间隔并且保存下来呢，list是mutable的不行，我们可以采用string或者tuple的方式来保存为key.

Solution

```
class Solution(object):
    def groupStrings(self, strings):
        """
        :type strings: List[str]
        :rtype: List[List[str]]
        """
        if not strings:
            return []

        results = []
        dic = {}
        for s in strings:
            hs = self.strHash(s)
            if hs not in dic:
                dic[hs] = [s]
            else:
                dic[hs].append(s)

        for key in dic:
            ls = dic[key]
            ls.sort()
            results.append(ls)

        return results


    def strHash(self, s):
        hsList = [(ord(i) - ord(s[0])) % 26 for i in s]
        return tuple(hsList)
```

Reference

- group shift string

Unique Abbreviation

题目描述

An abbreviation of a word follows the form . Below are some examples of word abbreviations:

```
a) it                      --> it    (no abbreviation)

     1
b) d|o|g                   --> d1g

              1    1  1
     1---5----0----5--8
c) i|nternationalizatio|n  --> i18n

              1
     1---5----0
d) l|ocalizatio|n          --> l10n
```

Assume you have a dictionary and given a word, find whether its abbreviation is unique in the dictionary. A word's abbreviation is unique if no other word from the dictionary has the same abbreviation.

Example:

```
Given dictionary = [ "deer", "door", "cake", "card" ]

isUnique("dear") -> false
isUnique("cart") -> true
isUnique("cane") -> false
isUnique("make") -> true
```

解题方法

Solution

```
class ValidWordAbbr(object):
    def __init__(self, dictionary):
        """
        initialize your data structure here.
        :type dictionary: List[str]
        """
        self.dic = {}
        for word in dictionary:
            abbr = self.getAbbr(word)
            if abbr == word:
                continue

            if abbr in self.dic:
                self.dic[abbr] = False
            else:
                self.dic[abbr] = word

    def isUnique(self, word):
        """
        check if a word is unique.
        :type word: str
        :rtype: bool
        """
        if not word:
            return True
        abbr = self.getAbbr(word)

        return (abbr not in self.dic) or (self.dic[abbr] == word)

    def getAbbr(self, word):
        abbr = ""
        if len(word) <= 2:
            abbr = word
        else:
            abbr = word[0] + str(len(word) - 2) + word[-1]

        return abbr
```

Reference

Implement Stack with Queue

题目描述

Implement the following operations of a stack using queues.

- push(x) -- Push element x onto stack.
- pop() -- Removes the element on top of the stack.
- top() -- Get the top element.
- empty() -- Return whether the stack is empty.

解题方法

Version A:

- push: enqueue in queue1
- pop: while size of queue1 is bigger than 1, pipe dequeued items from queue1 into queue2 dequeue and return the last item of queue1, then switch the names of queue1 and queue2

Version B:

- push: enqueue in queue2 enqueue all items of queue1 in queue2, then switch the names of queue1 and queue2
- pop: deqeue from queue1

Solution

Version B

```
from collections import deque

class Stack(object):
    def __init__(self):
        """
        initialize your data structure here.
        """
        self.queue1 = deque()
        self.queue2 = deque()

    def push(self, x):
        """
        :type x: int
        :rtype: nothing
        """
        self.queue1.append(x)

    def pop(self):
        """
        :rtype: nothing
        """
        while len(self.queue1) > 1:
            ele = self.queue1.popleft()
            self.queue2.append(ele)
        result = self.queue1.pop()
        self.queue1, self.queue2 = self.queue2, self.queue1
        return result

    def top(self):
        """
        :rtype: int
        """
        if len(self.queue1) > 1:
            ele = self.queue1.popleft()
            self.queue2.append(ele)
        result = self.queue1[-1]
        return result

    def empty(self):
        """
        :rtype: bool
        """
        if not self.queue1 and not self.queue2:
            return True
        else:
            return False
```

Reference

- stackoverflow
- geeksforgeeks

Implement Queue with stacks

题目描述

Implement the following operations of a queue using stacks.

- push(x) -- Push element x to the back of queue.
- pop() -- Removes the element from in front of queue.
- peek() -- Get the front element.
- empty() -- Return whether the queue is empty.

解题方法

用两个stack

- Push时，push到stack1上
- 当pop或top时，将stack1的都pop到stack2上，此时stack2的top就是最早的

Solution

```
class Queue(object):
    def __init__(self):
        """
        initialize your data structure here.
        """
        self.stack1 = []
        self.stack2 = []

    def push(self, x):
        """
        :type x: int
        :rtype: nothing
        """
        self.stack1.append(x)
    def pop(self):
        """
        :rtype: nothing
        """
        if not self.stack2:
            while self.stack1:
                ele = self.stack1.pop()
                self.stack2.append(ele)
        return self.stack2.pop()

    def peek(self):
        """
        :rtype: int
        """
        if not self.stack2:
            while self.stack1:
                ele = self.stack1.pop()
                self.stack2.append(ele)
        return self.stack2[-1]

    def empty(self):
        """
        :rtype: bool
        """
        if not self.stack1 and not self.stack2:
            return True
        else:
            return False
```

Reference

Min Stack

题目描述

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

- push(x) -- Push element x onto stack.
- pop() -- Removes the element on top of the stack.
- top() -- Get the top element.
- getMin() -- Retrieve the minimum element in the stack.

解题方法

- 用另一个stack存可能的最小值 `minStack`
- 如果新Push的value大于当前的最小值，就不用存，只有当新的 `value <= current min` 时, 才push到这个 `minStack` 上
- 如果pop的value == `minStack` 的top, 才从它pop

Solution

```
class MinStack(object):
    def __init__(self):
        """
        initialize your data structure here.
        """
        self.stack = []
        self.minStack = []

    def push(self, x):
        """
        :type x: int
        :rtype: nothing
        """
        self.stack.append(x)
        if not self.minStack or x <= self.minStack[-1]:
            self.minStack.append(x)

    def pop(self):
        """
        :rtype: nothing
        """
        ele = self.stack.pop()
        if ele == self.minStack[-1]:
            self.minStack.pop()

    def top(self):
        """
        :rtype: int
        """
        return self.stack[-1]

    def getMin(self):
        """
        :rtype: int
        """
        return self.minStack[-1]
```

Reference

Valid Parenthesis

题目描述

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

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

解题方法

当前的character需要更前一个last in的character判断关系的题目，显然是用stack解决

Solution

```python
class Solution(object):
    def isValid(self, s):
        """
        :type s: str
        :rtype: bool
        """
        if not s:
            return True
        stack = []
        for c in s:
            if c in ("(", "[", "{"):
                stack.append(c)
            elif len(stack) == 0:
                return False
            elif c == ")":
                prev = stack.pop()
                if prev != "(":
                    return False
            elif c == "]":
                prev = stack.pop()
                if prev != "[":
                    return False
            elif c == "}":
                prev = stack.pop()
                if prev != "{":
                    return False
        if len(stack) != 0:
            return False

        return True
```

Reference

Evaluate Reverse Polish Notation

题目描述

Evaluate the value of an arithmetic expression in Reverse Polish Notation.

Valid operators are `+, -, *, /`. Each operand may be an integer or another expression.

Some examples:

```
["2", "1", "+", "3", "*"] -> ((2 + 1) * 3) -> 9
["4", "13", "5", "/", "+"] -> (4 + (13 / 5)) -> 6
```

解题方法

这一题的基本思路就是用stack，每次遇到operator的时候pop出最后的两个， 计算后再加入到stack上。

用python解题需要注意的是，**python**的负数出发运算与**java**存在差异。

在java中， `6/-132 = 0`, java使用的是截断。 而在python中， `6/-132 = -1`, python使用的是floor。

Solution

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

        stack = []
        for token in tokens:
            if token not in ["+", "-", "*", "/"]:
                stack.append(float(token))
            else:
                second = stack.pop()
                first = stack.pop()
                if token == "+":
                    r = first + second
                elif token == "-":
                    r = first - second
                elif token == "*":
                    r = first * second
                elif token == "/":
                    r = int(first / second)
                stack.append(r)

        return int(stack[0])
```

Reference

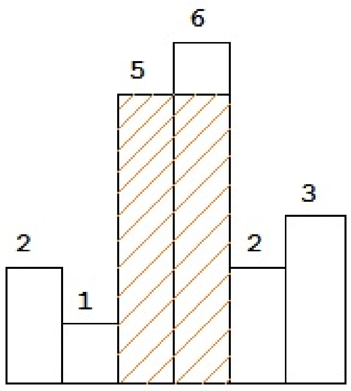
- reverse polish notation

Largest Rectangle in Histogram

题目描述

Given n non-negative integers representing the histogram's bar height where the width of each bar is 1, find the area of largest rectangle in the histogram.

given height = `[2,1,5,6,2,3]`



解题方法

brute force方法

对每个bar都遍历之前的bar来找最大rectangle的面积，这样的时间复杂度是 `O(n^2)`

stack

对于每个bar的最大rectangle面积的计算，如果以当前的bar做为最低的height, 这样我们需要知道的就是 `左边界` 和 `右边界` ，左右边界也就是第一个比这个bar高度低的bar的index。

而需要知道 `第一个` 的位置，也就是最近的index, 可以想到是需要用stack这个data structure, 才能够获得最左边第一个小于它高度的index， 而 `右边界` 可以在遍历的过程中得到。

如果我们在遍历时

- 如果当前bar的高度 >= 栈顶的高度，那么就将当前的index push到stack上
- 如果当前bar的高度 <= 栈顶的高度
    - 那么说明当前的index对于栈顶高度来说是一个 `右边界`
    - 将栈顶元素pop,而根据我们对于stack的操作，当前的栈顶index就是Pop出来高度的 `左边界`
    - 计算面积，更新最大面积

注意

- 等于的时候也要push到stack上，如果两个相等高度的bar index不一样，需要更新该高度的index

Solution

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

        stack = []
        maxArea = 0
        length = len(height)

        i = 0
        while i < length:
            if not stack or height[i] >= height[stack[-1]]:
                stack.append(i)
                i += 1
            else:
                # 将stack顶端的当成smallest bar height
                h = height[stack.pop()]
                if stack:
                    area = h * (i - stack[-1] - 1)
                else:
                    area = h * i
                maxArea = area if area > maxArea else maxArea

        # 最后一个bar还是会Push到stack上，还要计算剩下这些面积
        while stack:
            h = height[stack.pop()]

            if stack:
                area = h * (i - stack[-1] - 1)
            else:
                area = h * i
            maxArea = area if area > maxArea else maxArea

        return maxArea
```

Reference

- O(nlogn)
- O(n)

Maximal Rectangle

题目描述

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

解题方法

brute force

取任意两点为两个对角点，然后check是否包含为0的元素。

O(n^6)

左上角

只取一个左上角，然后逐行找行能达到的最右边，计算面积

利用 largest rectangle in histogram

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

如果是一个这样的matrix, 逐行扫描的话，其实就是bar的高度

- 第一行 [0, 1, 0, 1, 0]
- 第二行 [1, 2, 0, 2, 1]
- 第三行 [2, 3, 0, 3, 2] ...

那么其实就是和largest rectangle in histogram一样的题目了

时间复杂度 `O(n ^ 2)` , 空间复杂度为 `O(n)`

Solution

```python
class Solution(object):
    def maximalRectangle(self, matrix):
        """
        :type matrix: List[List[str]]
        :rtype: int
        """
        if not matrix:
            return 0
        rowNum = len(matrix)
        colNum = len(matrix[0])
        result = 0

        height = [0 for j in range(colNum)]
        for i in range(rowNum):
            for j in range(colNum):
                if matrix[i][j] == "0":
                    height[j] = 0
                else:
                    height[j] += 1
            curMax = self.largestRectangleArea(height)
            result = curMax if curMax > result else result

        return result
```

Reference

Simplify Path

题目描述

Given an absolute path for a file (Unix-style), simplify it.

For example,

```
path = "/home/", => "/home"
path = "/a/./b/../../c/", => "/c"
```

Corner Cases:**

- "/../" => "/"
- "/home//foo/" => "/home/foo"

解题方法

非stack方法

将string根据"/" split, 用一个string来记录path

- 遇到 `..` , 就将上一个dir去掉
- 遇到 `.` 或者 `` `， `` `表示` //`这种情况，不做任何事

注意点

- `/` 路径的判断！

stack做法

将 `/` 之间的都push到stack上，注意 `empty string` 不要push,

- 遇到"..", 就 `pop`
- 遇到".", 什么都不做

- 遇到others, push到stack上

Solution

```python
class Solution(object):
    def simplifyPath(self, path):
        """
        :type path: str
        :rtype: str
        """
        if not path:
            return ""
        path = path.split("/")
        cur = "/"

        for p in path:
            if p == "..":
                if cur != "/":
                    cur = cur.split("/")[:-1]
                    cur = "/".join(cur)
                    if cur == "":
                        # e.g. "/a" would become ""
                        cur = "/"
            elif p != "." and p != "":
                # take care of ″/"
                if cur == "/":
                    cur += p
                else:
                    cur += "/" + p

        return cur
```

Reference

Closest Binary Search Tree Value II

题目描述

Given a non-empty binary search tree and a target value, find k values in the BST that are closest to the target.

Note: Given target value is a floating point. You may assume k is always valid, that is: k ≤ total nodes. You are guaranteed to have only one unique set of k values in the BST that are closest to the target.

Follow up:

Assume that the BST is balanced, could you solve it in less than O(n) runtime (where n = total nodes)?

解题方法

在遍历这个tree的过程中，要不断地更新距离target的距离，时刻替换。 而一般的data structure都没有比较的作用，只有heap可以维护一个有比较的 structure, 那么用来比较的value就是跟target的距离。

follow u

利用Inorder得到pre-decessors的性质来做，逆inorder就是得到successors

- 找到这个target的closest node
- 对于这个node找到pre-decessors和successors
- 然后用类似merge sort的方法找到k个最近的值

Solution

```
class Solution(object):
    def closestKValues(self, root, target, k):
        """
        :type root: TreeNode
        :type target: float
        :type k: int
        :rtype: List[int]
        """

        heap = []
        stack = []

        stack.append(root)
        while stack:
            cur = stack.pop()
            if cur.right:
                stack.append(cur.right)
            if cur.left:
                stack.append(cur.left)

            if len(heap) < k:
                heapq.heappush(heap, (abs(cur.val - target) * -1, c
            elif (abs(cur.val - target) * -1) > heap[0][0]:
                heapq.heappop(heap)
                heapq.heappush(heap, (abs(cur.val - target) * -1, c

        return [node[1] for node in heap]
```

Reference

- 参考
- two stack方法

Sliding Window Maximum

题目描述

Given an array nums, there is a sliding window of size k which is moving from the very left of the array to the very right. You can only see the k numbers in the window. Each time the sliding window moves right by one position.

For example, Given `nums = [1,3,-1,-3,5,3,6,7]`, and `k = 3`.

```
Window position                Max
---------------               -----
[1  3  -1] -3  5  3  6  7       3
 1 [3  -1  -3] 5  3  6  7       3
 1  3 [-1  -3  5] 3  6  7       5
 1  3  -1 [-3  5  3] 6  7       5
 1  3  -1  -3 [5  3  6] 7       6
 1  3  -1  -3  5 [3  6  7]      7
```

Therefore, return the max sliding window as `[3,3,5,5,6,7]`.

Note: You may assume k is always valid, ie: `1 ≤ k ≤ input` array's size for non-empty array.

Follow up:

Could you solve it in linear time?

Hint:

How about using a ·data structure such as deque (double-ended queue)? The queue size need not be the same as the window’s size. Remove redundant elements and the queue should store only elements that need to be considered.

解题方法

heap

如果用一个maxheap的话，那么删除数据的时候比较麻烦，所以需要用hashheap以达到快速找最大值和快速删除的效果

deque

用一个Deque, 前后都可以pop, 但是如果只是按顺序把element加入的话还需要求最大值，额外增加了复杂度。我们可以只存能够成为候选最大值 的element的index，存index是方便删除的时候判断。

比如 `[1,2,3,7,5,6]` , window size `4`

1,2,3,7时，我们只需要在deque中存7的 `Index 3` 就可以，因为前面1，2，3在被删除时，并不会影响window max,也就不是最大值的候选，所以不用存。 所以方法的步骤就是：

- 遇到一个新的元素index为i，value为v, 将deque末尾小于它的value都pop掉，直到比它大的，然后将i加到Deque的末尾
- 如果size > k了，`if i - k == deque[0]: deque.popleft()`
- `deque[0]` 是当前window的最大值

Solution

```
from collections import deque

class Solution(object):
    def maxSlidingWindow(self, nums, k):
        """
        :type nums: List[int]
        :type k: int
        :rtype: List[int]
        """
        if not nums:
            return []
        if not k:
            return nums
        window = deque()
        length = len(nums)
        p = 0
        result = []
        while p < length:
            while window and nums[window[-1]] < nums[p]:
                window.pop()
            window.append(p)
            if p - k == window[0]:
                window.popleft()
            if p >= k - 1:
                result.append(nums[window[0]])
            p += 1
        return result
```

Reference

- sliding window

Count and Say

题目描述

The count-and-say sequence is the sequence of integers beginning as follows: `1, 11, 21, 1211, 111221, ...`

```
1 is read off as "one 1" or 11.
11 is read off as "two 1s" or 21.
21 is read off as "one 2, then one 1" or 1211.
Given an integer n, generate the nth sequence.
```

Note: The sequence of integers will be represented as a string.

解题方法

找出生成这种string的规律，其实就是计数，如果当前的charcter等于之前的，就加1；否则就将现在的加到结果里，重新从1开始计数

Solution

```python
class Solution(object):
    def countAndSay(self, n):
        """
        :type n: int
        :rtype: str
        """
        # init string
        result = "1"
        if not n or n == 1:
            return result

        for i in range(n-1):
            result = self.getNext(result)

        return result

    def getNext(self, result):
        prefix = 1
        nextStr = ""
        length = len(result)
        for i in range(1, length):
            if result[i] == result[i-1]:
                prefix += 1
            else:
                nextStr += str(prefix) + result[i-1]
                prefix = 1
        nextStr += str(prefix) + result[-1]
        return nextStr
```

Reference

Zigzag Conversion

题目描述

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

```
P   A   H   N
A P L S I I G
Y   I   R
```

And then read line by line: `"PAHNAPLSIIGYIR"` Write the code that will take a string and make this conversion given a number of rows:

`string convert(string text, int nRows);`

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

解题方法

建立一个2d array，按照顺序将字母加到这个array里，然后再按行输出就可以了。

Solution

```
class Solution(object):
    def convert(self, s, numRows):
        """
        :type s: str
        :type numRows: int
        :rtype: str
        """
        if numRows == 1 or numRows >= len(s):
            return s

        zigzag = [[] for i in range(numRows)]

        row, step = 0, 1
        for c in s:
            zigzag[row].append(c)
            if row == 0:
                step = 1
            elif row == numRows - 1:
                step = -1
            row += step

        result = ""
        for i in range(numRows):
            result += "".join(zigzag[i])

        return result
```

Reference

Length of Last Word

题目描述

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

If the last word does not exist, return `0`.

Note: A word is defined as a character sequence consists of non-space characters only.

For example,

```
s = "Hello World",
return 5.
```

解题方法

easy question

Solution

```
class Solution(object):
    def lengthOfLastWord(self, s):
        """
        :type s: str
        :rtype: int
        """
        if not s:
            return 0
        s = s.strip()
        length = len(s)
        if length == 0:
            return 0

        result = 0
        p = length - 1
        while p >= 0:
            if s[p] != ' ':
                p -= 1
                result += 1
            else:
                break

        return result
```

Reference

Compare Version Number

题目描述

Compare two version numbers version1 and version2. If version1 > version2 return 1, if version1 < version2 return -1, otherwise return 0.

You may assume that the version strings are non-empty and contain only digits and the . character. The . character does not represent a decimal point and is used to separate number sequences. For instance, 2.5 is not "two and a half" or "half way to version three", it is the fifth second-level revision of the second first-level revision.

Here is an example of version numbers ordering:

```
0.1 < 1.1 < 1.2 < 13.37
```

解题方法

这一个题目并不难，但是需要考虑各种case, 所以在面试的时候要跟考官交流好可能的version number有哪些.

比如会出现的有

- `01`
- `0.1.2`
- `01.2.1`

所以不一定只有一个小数点，在split之后，应该取长度更长的array来Loop，如果已经超出array长度就取0来比较

Solution

```
class Solution(object):
    def compareVersion(self, version1, version2):
        """
        :type version1: str
        :type version2: str
        :rtype: int
        """
        if not version1 or not version2:
            return
        v1 = version1.split(".")
        v2 = version2.split(".")

        length = max(len(v1), len(v2)) # 取v1, v2长度更长的那个
        for i in range(length):
            # 如果没有了就用0来比较
            num1 = int(v1[i]) if i < len(v1) else 0
            num2 = int(v2[i]) if i < len(v2) else 0
            if num1 > num2:
                return 1
            elif num1 < num2:
                return -1
            else:
                continue

        return 0
```

Reference

Reverse Words in a String

题目描述

Given an input string, reverse the string word by word.

For example, Given s = "the sky is blue", return "blue is sky the".

解题方法

注意要用 `s.split()` , 不要pass任何参数，这样连续的whitespace都会被去掉

Solution

```
class Solution(object):
    def reverseWords(self, s):
        """
        :type s: str
        :rtype: str
        """
        return " ".join(map(lambda x : x[::-1], s[::-1].split()))
```

Reference

Reverse Words in a String II

题目描述

do it in-place, no extra space

解题方法

这里的输入也不同了，是一个list, 这样才可以in-place

Solution

```
class Solution(object):
    def reverseWords(self, s):
        """
        :type s: a list of 1 length strings (List[str])
        :rtype: nothing
        """
        self.reverse(s, 0, len(s) - 1)

        first = 0
        for i in range(len(s)):
            if s[i] == " ":
                self.reverse(s, first, i-1)
                first = i + 1
            elif i == len(s) - 1:
                self.reverse(s, first, i)

    def reverse(self, s, start, end):
        while start < end:
            s[start], s[end] = s[end], s[start]
            start += 1
            end -= 1
```

Reference

Longest Palindrome Substring

题目描述

Given a string S, find the longest palindromic substring in S. You may assume that the maximum length of S is 1000, and there exists one unique longest palindromic substring.

解题方法

brute force

i,j 确定一个substring,然后再check这个substring是否是palindrome。

时间复杂度是 `O(n^2 * n) = O(n^3)`

DP

在brute中有很多重复计算，并且两个palindrome之间是有关系的，所以很容易可以往DP的方向想。

state

`dp[i][j]` 表示从i到j的substring是否为palindrome

function

```
if dp[j+1][i-1] == True and s[i] == s[j]:
    dp[i][j] = True
```

这里要求 `j < i` 并且 `dp[j+1][i-1]` 在 `dp[j][i]` 之前就已经计算出来了，所以 i应该从0到length的遍历，而j从0到i遍历，只有 `j == i - 1` 的情况需要特殊处理，因为之前 的一个loop里j达不到这个 `i-1` .

- 时间复杂度 `O(n^2)`

- 空间复杂度 `O(n^2)`

确定pivot, 两边扩展

对于每个i，确定pivot向两边扩展找最大的palindrome

注意点

- palindrome长度可以是奇数也可以是偶数
- 奇数的话pivot就是 `s[i]`
- 偶数的话pivot就应该确定为 `i-1` , `i` 的中间

- 时间复杂度 `O(n^2)`

- 空间复杂度 `O(1)`

Solution

DP

方法正确，但是过不了test case, time limit exceeds

```
class Solution(object):
    def longestPalindrome(self, s):
        """
        :type s: str
        :rtype: str
        """
        if not s:
            return ""

        result = ""
        length = len(s)
        if length == 1:
            return s

        dp = [[False for i in range(length)] for j in range(length)
        for i in range(length):
            dp[i][i] = True
            if len(result) < 1:
                result = s[i]

        for i in range(length):
            for j in range(i):
                if j == i - 1:
                    if s[i] == s[j]:
                        dp[j][i] = True
                        if len(result) < 2:
                            result = s[j:i+1]
                else:
                    if s[i] == s[j] and dp[j+1][i-1]:
                        dp[j][i] = True
                        if len(result) < i - j + 1:
                            result = s[j:i+1]

        return result
```

确定中心往两边扩展

可以过test case

```
class Solution(object):
    def longestPalindrome(self, s):
        """
        :type s: str
        :rtype: str
        """
        if not s:
            return ""

        result = ""
        length = len(s)
        if length == 1:
            return s

        for i in range(1, length):
            # center is between i - 1, i
            # 长度为偶数的palindrome
            low, high = i - 1, i
            while low >= 0 and high < length and s[low] == s[high]
                low -= 1
                high += 1

            curLen = (high - 1) - (low + 1) + 1
            if curLen > len(result):
                result = s[low+1:high]

            # center is i
            # 长度为奇数的palindrome
            low, high = i - 1, i + 1
            while low >= 0 and high < length and s[low] == s[high]
                low -= 1
                high += 1

            curLen = (high - 1) - (low + 1) + 1
            if curLen > len(result):
                result = s[low+1:high]

        return result
```

Reference

- Longest Palindromic Substring
- Longest Palindromic Substring各种解法

Shortest Palindrome

题目描述

Given a string S, you are allowed to convert it to a palindrome by adding characters in front of it. Find and return the shortest palindrome you can find by performing this transformation.

For example:

Given `"aacecaaa"`, return `"aaacecaaa"`.

Given `"abcd"`, return `"dcbabcd"`.

解题方法

这一道题可以借鉴longest palindrome substring的方法，从每个pivot搜palindrome, 如果左边能够停留在开头，说明可以通过在开头加characters找到shortest palindrome.

注意

- 这里其实只需要找一半, 因为结果的左半部分不完整，必然是在前半部分有一个pivot
- 从后往前扫，这样找到了结果就可以返回了

KMP解法

`O(n)` 的算法，需要理解一下 理解kmp kmp geeksfor geeks shortest Palindrome

Solution

利用longest palindrome substring

python超时

```
class Solution(object):
    def shortestPalindrome(self, s):
        """
        :type s: str
        :rtype: str
        """
        if not s:
            return ""

        length = len(s)
        minLength = sys.maxint
        result = ""
        if self.isPalindrome(s):
            return s


        if length % 2 == 1:
            half = length / 2 + 1
        else:
            half = length / 2

        for i in range(1, half):
            # pivot is between i-1, i
            low = i - 1
            high = i
            while low >= 0 and high < length and s[low] == s[high]
                low -= 1
                high += 1
            if low == -1:
                pLength = (high - 1) - (low + 1) - 1
                doubleLength = (length - 1) - high + 1
                if minLength > pLength + 2 * doubleLength:
                    minLength = pLength + 2 * doubleLength
                    result = s[length-1:high-1:-1] + s

            # pivot is i
            low = i - 1
            high = i + 1
            while low >= 0 and high < length and s[low] == s[high]
```

```
                low -= 1
                high += 1
            if low == -1:
                pLength = (high - 1) - (low + 1) - 1
                doubleLength = (length - 1) - high + 1
                if minLength > pLength + 2 * doubleLength:
                    minLength = pLength + 2 * doubleLength
                    result = s[length-1:high-1:-1] + s
        return result

    def isPalindrome(self, s):
        length = len(s)
        if length == 0 or length == 1:
            return True

        left, right = 0, length - 1
        while left < right:
            if s[left] == s[right]:
                left += 1
                right -= 1
            else:
                return False
        return True
```

```
class Solution(object):
    def shortestPalindrome(self, s):
        """
        :type s: str
        :rtype: str
        """
        if not s:
            return ""

        length = len(s)
        minLength = sys.maxint
        result = ""
        if self.isPalindrome(s):
            return s
        for i in range(length / 2, -1, -1):
            # pivot is between i-1, i
            if s[i] == s[i-1]:
                result = self.scanFromCenter(s, i-1, i)
                if result:
                    return result
            else:
                result = self.scanFromCenter(s, i, i)
                if result:
                    return result

    def scanFromCenter(self, s, l, r):
        while l >= 0 and r < len(s) and s[l] == s[r]:
            l -= 1
            r += 1

        if l >= 0:
            return None

        result = s[len(s)-1:r-1:-1] + s


    def isPalindrome(self, s):
        length = len(s)
        if length == 0 or length == 1:
            return True

        left, right = 0, length - 1
        while left < right:
            if s[left] == s[right]:
                left += 1
                right -= 1
            else:
                return False
        return True
```

Reference

Valid Anagram

题目描述

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

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

解题方法

sort

第一种方法是将两个string都sort一下，然后就可以判断是否相同。

- 时间复杂度 `O(nlogn)`
-

hashtable

第二种方法是统计两个string中出现过的字母的次数，然后比较是否相同。

- 时间复杂度`O(n)`
- 空间复杂度`O(n)`

> python的collections module有Counter,可以直接统计字母出现的次数，可以利用

Solution

传统写法

```python
class Solution(object):
    def isAnagram(self, s, t):
        """
        :type s: str
        :type t: str
        :rtype: bool
        """
        dic = {}
        for c in s:
            if c not in dic:
                dic[c] = 1
            else:
                dic[c] += 1

        for c in t:
            if c not in dic:
                return False
            else:
                dic[c] -= 1

        for key in dic:
            if dic[key] != 0:
                return False

        return True
```

Counter

```
class Solution(object):
    def isAnagram(self, s, t):
        """
        :type s: str
        :type t: str
        :rtype: bool
        """
        t1 = sorted(collections.Counter(s).items())
        t2 = sorted(collections.Counter(t).items())

        return t1 == t2
```

Reference

Group Anagram

题目描述

Given an array of strings, group anagrams together.

For example, given: `["eat", "tea", "tan", "ate", "nat", "bat"]` ,

Return:

```
[
  ["ate", "eat","tea"],
  ["nat","tan"],
  ["bat"]
]
```

解题方法

判断是否是anagram可以用上一题的思想，但是这里需要快速查找，那么就应该往hashtable的方向想，本来 `Counter.items()` 是list。list是不可以做dictinary的key的，因为它 不是hashable的，所以我们将list转换为tuple。

Solution

```
class Solution(object):
    def groupAnagrams(self, strs):
        """
        :type strs: List[str]
        :rtype: List[List[str]]
        """
        if not strs:
            return []
        if len(strs) == 1:
            return [strs]

        strs.sort()
        result = {}
        for i in range(len(strs)):
            hashCode = self.hashFunction(strs[i].lower())
            if hashCode in result:
                result[hashCode].append(strs[i])
            else:
                result[hashCode] = [strs[i]]

        return result.values()

    def hashFunction(self, s):
        return tuple(sorted(collections.Counter(s).items()))
```

Reference

- facebook anagram
- geeks for geeks