链滴

leetcode解题报告-数组

作者：Zing
原文链接：https://ld246.com/article/1458304008661
来源网站：链滴

leetcode中数组问题是占比重最大的问题。数组题目的特点是灵活多变，可考察的点非常多。主要涉的考点有：递归回溯、排序、二分查找、动态规划、贪心、与图和树结合、与实际问题结合等等。而其中可以玩味可优化的时间和空间复杂度非常多。

以下是解题报告。

##1.Two Sum　（数学）

- 难度：Medium
- 题意：

给定一个整数数组，要求找出数组中两数和为指定数字，返回这两个数的索引。 假定有且只有一个。

- 思路：

这道题的关键在于如何快速找到taget-n的索引，直接扫肯定太慢，当然用哈希表最快。这道题的陷在于，注意同一个数不能用两次，需要判断索引是否相同。若同一个数出现多次，记录最后出现的索位置即可，因为有上面那个判断，可以区分出这个一个数字出现多次的情况。

这种解法的时间复杂度为O(n)，遍历一次数组，循环中查哈希表复杂度为1。

- 代码：

```
class Solution:
    # @param {integer[]} nums
    # @param {integer} target
    # @return {integer[]}
        def twoSum(self, nums, target):
            dic={}
            for i,n in enumerate(nums):
                dic[n]=i
            for i,n in enumerate(nums):
                if target-n in dic and dic[target-n]!=i:
                    return[i,dic[target-n]]
```

##15.3Sum　（数学）

- 难度:Medium
- 题意：

给定一个整数数组，找出所有能使a+b+c=0三元组。注意，元组中按正序排列，不允许重复元组出。

思路1：3个数和为0，那么必有以下推断：有1个数字<=0,有1个数>=0。因此，我采用以下解法。先数组排序（nlogn）,找到<=0 和 >0的分界点dis。第一个数字firstn遍历[0,dis),第二个数字secondn环遍历(firstn,len-1),然后在数组（secondn,len-1）中二分查找第三个数字。看似有2重循环+二分查，但是由于我们经过了大量的剪枝，所有实际上复杂度没有那么高。

- 代码：

```
class Solution:
```

```
    # @param {integer[]} nums
    # @return {integer[][]}
    def threeSum(self, nums):
        #print(datetime.datetime.now())
        result = []
        if len(nums)<3:
            return result
        nums.append(1)
        nums.sort()
        dis = self.binarySearch(nums,0,len(nums)-1,1)
        nums.remove(1)
        for firstn in range(dis):
            for secondn in range(firstn+1,len(nums)-1):
                thirdn = self.binarySearch(nums,secondn+1,len(nums)-1,0-(nums[firstn]+nums[sec
ndn]))
                if thirdn != -1:
                    if[nums[firstn],nums[secondn],nums[thirdn]] not in result:
                        result.append([nums[firstn],nums[secondn],nums[thirdn]])
        return result


    #binary search
    def binarySearch(self,nums,i,j,n):
        if n<nums[i] or n>nums[j] or i>j:
            return -1
        mid = (i+j)//2
        if n == nums[mid]:
            return mid
        elif n < nums[mid]:
            return self.binarySearch(nums,i,mid-1,n)
        else:
            return self.binarySearch(nums,mid+1,j,n)
```

- 思路2：

这道题作为2sum的延伸，当然我们的第一想法应该是上面那么复杂（由于没有系统地训练，这两道时间间隔比较大）。联系2sum，我们可以很容易想到使用哈希表，这次哈希表记录的不是数字的索，而是数字出现的次数，或者说数字可以使用的次数。第一个数字，我们遍历整个数组，同时把选中第一个数字的哈希值-1.第二个数字，遍历整个数组，需要满足哈希值>0，把选中的数字的哈希值-1.三个数字，查找哈希表中满足0-a-b,且哈希值>0的数字。这样复杂度可以降低到(n*n)。 由于没有剪，所有两重循环是实打实的。

这里有个奇怪的问题，这份代码提交报了超时，但是超时的用例，使用leetcode的测试功能（最近加的新功能），却能正确得到结果，且时间平均在80ms，比思路1快了10倍。

- 代码：

```
class Solution:
    # @param {integer[]} nums
    # @return {integer[][]}
    def threeSum(self, nums):
        dic={}
        result=[]
        for n in nums:
            if n not in dic:dic[n]=1
```

```
            else: dic[n]+=1
        for a in nums:
            dic[a]-=1
            for b in nums:
                if dic[b]>0:
                    dic[b]-=1
                    if 0-a-b in dic and dic[0-a-b]>0 and sorted([a,b,0-a-b]) not in result:
                        result.append(sorted([a,b,0-a-b]))
                    dic[b]+=1
            dic[a]+=1
        return result
```

##16.3Sum Closest （数学）

- 难度:Medium
- 题意：

给定一个整数数组，求a,b,c满足a+b+c最接近target。假设有且仅有1个解。

- 思路：

这道题的思路就和3Sum比较接近了，遍历前2个数字，但是求第三个数字时，需要对二分查找进行修，在查找不到时返回和该数最接近的数字。

- 代码：

```
class Solution:
    # @param {integer[]} nums
    # @param {integer} target
    # @return {integer}
    def threeSumClosest(self, nums, target):
        sumClosest = 2**32-1
        nums.sort()
        for firstn in range(len(nums)-2):
            for secondn in range(firstn+1,len(nums)-1):
                thirdn = self.binarySearch(nums[secondn+1::],0,len(nums[secondn+1::])-1,target-(
ums[firstn]+nums[secondn]))
                stmp = nums[firstn]+nums[secondn]+nums[secondn+1::][thirdn]
                if  stmp == target:
                    return target
                else:
                    if abs(stmp-target)<abs(sumClosest-target):
                        sumClosest = stmp
        return sumClosest



    #binary search
    #if no such num return the nearlyst
    def binarySearch(self,nums,i,j,n):
        if n<nums[i]:
            if i == 0:return i
```

```
            else:
                if abs(nums[i]-n)<abs(nums[i-1]-n):return i
                else:return i-1
        if n>nums[j]:
            if j==len(nums)-1:return j
            else:
                if abs(nums[j]-n)<abs(nums[j+1]-n):return j
                else:return j+1
        if i>j:
            if abs(nums[i]-n)<abs(nums[j]-n):return i
            else:return j
        mid = (i+j)//2
        if n == nums[mid]:return mid
        elif n < nums[mid]:
            return self.binarySearch(nums,i,mid-1,n)
        else:
            return self.binarySearch(nums,mid+1,j,n)
```

##18.4Sum （数学）

- 难度:Medium
- 题意：

给定整数数组，求出所有a,b,c,d,满足a+b+c+d=target。要求元组中升序排列，元组不重复。

- 思路：若按照2Sum和4Sum的思路，用3重循环遍历前3个数，再找到满足要求的第4个数，肯定是超时的（基本上在OJ中3重循环就肯定会超时了），而且剪枝几乎不可能实现，4个数可能出现的情太多。

因此另寻出路：把4Sum变成2个2Sum问题。建立一个哈希表，用于记录｛两数之和:[[索引a,索引b],...]...｝,记录索引有一个好处就是可以解决重复数字的问题。遍历a和b,在哈希表中查找target-a-b，注意除1个数字使用多次的情况。把新的a+b，加入哈希表中。如此这般时间复杂度仅有（n*n）。

- 代码：

```
class Solution:
    # @param {integer[]} nums
    # @param {integer} target
    # @return {integer[][]}
    def fourSum(self, nums, target):
        result = []
        sumdir = {}
        if len(nums)<4:
            return result
        nums.sort()
        for i in range(len(nums)-1):
            for j in range(i+1,len(nums)):
                sumlist = sumdir.get(target - (nums[i]+nums[j]))
                if sumlist:
                    for na,nb in sumlist:
                        if i not in(na,nb) and j not in(na,nb):
                            tmp = sorted([nums[i],nums[j],nums[na],nums[nb]])
                            if tmp not in result:
```

```
                result.append(tmp)
            if  not sumdir.get(nums[i]+nums[j]):
                sumdir[nums[i]+nums[j]] = []
            sumdir[nums[i]+nums[j]].append([i,j])
    return result
```

##39. Combination Sum （数学）

- 难度：Medium
- 题意：

给定一个候选数据集合C，和一个目标值T。从C找到所有不同的组合使和为T。每个数字可以无限制用多次。注：所有数字均非负，组合内按升序排列。结果不包含相同组合。

- 思路：

使用递归回溯就可以轻松解决这道题。回溯的时候记得恢复现场。这里突然想到，传递的时候并不需传递整个数组，只需要传递开始的index，往后寻找就可以了，因为1，2，5和1，5，2是一样的，这还可以减少一样的结果。

- 代码：

```
class Solution:
    # @param {integer[]} candidates
    # @param {integer} target
    # @return {integer[][]}


    def combinationSum(self, candidates, target):
        self.result = []
        candidates.sort()
        self.choseEle(candidates,target,[])
        return self.result

    def choseEle(self, candidates, target, ele0):
        ele = ele0[:]
        for i in range(len(candidates)):
            s = sum(ele)+candidates[i]
            if s == target:
                ele.append(candidates[i])
                ele.sort()
                if ele not in self.result:
                    self.result.append(ele)

            elif s < target:
                ele.append(candidates[i])
                self.choseEle(candidates, target, ele)
                ele.pop()
            else:
                break
```

##40. Combination Sum II （数学）

- 难度：Medium
- 题意：

给定一个候选数据集合C，和一个目标值T。从C找到所有不同的组合使和为T。每个数字只能用一次
注：所有数字均非负，组合内按升序排列。结果不包含相同组合。

- 思路：

和39的思路一样，也是用递归回溯，由于每个函数只能用1次，因此把剩余可用的候选集也作为参数递。

- 代码：

```
class Solution:
    # @param {integer[]} candidates
    # @param {integer} target
    # @return {integer[][]}


    def combinationSum2(self, candidates, target):
        self.result = []
        candidates.sort()
        self.choseEle(candidates,target,[])
        return self.result

    def choseEle(self, candidates0, target, ele0):
        ele = ele0[:]
        candidates = candidates0[:]

        for i in range(len(candidates)):
            s = sum(ele)+candidates[i]
            if s == target:
                ele.append(candidates[i])
                ele.sort()
                if ele not in self.result:
                    self.result.append(ele)

            elif s < target:
                ele.append(candidates[i])
                candidates.remove(candidates[i])
                self.choseEle(candidates, target, ele)
                candidates.insert(i,ele.pop())
            else:
                break
```

##216. Combination Sum III（数学）

- 难度：Medium
- 题意：

从 1 到 9 中选取 k 个数字使和为 n，每个数字只能用一次。

- 思路：

和前几题差别不不大，递归回溯，限制递归深度必须为k。

- 代码：

```
class Solution:
    # @param {integer[]} candidates
    # @param {integer} target
    # @return {integer[][]}


    def combinationSum3(self, k, n):
        self.result = []
        maxn = (9+10-k)*k/2
        if n<maxn:
            self.choseEle(list(range(1,10)),n,[],k)
        elif n == maxn:
            self.result.append(list(range(10-k,10)))
        return self.result

    def choseEle(self, candidates0, target, ele0, deep):
        ele = ele0[:]
        candidates = candidates0[:]
        if deep <= 0:
            return

        for i in range(len(candidates)):
            s = sum(ele)+candidates[i]
            if s == target and deep == 1:
                ele.append(candidates[i])
                ele.sort()
                if ele not in self.result:
                    self.result.append(ele)
            elif s < target:
                ele.append(candidates[i])
                candidates.remove(candidates[i])
                self.choseEle(candidates, target, ele ,deep-1)
                candidates.insert(i,ele.pop())
            else:
                break
```

##4. Median of Two Sorted Arrays （中位数）

- 难度：Hard
- 题意：

要求找到两个已经排序数组的中位数。要求时间复杂度不超过O（log（m+n））

- 思路：

先求出中位数的位置（注意奇和偶）x，由于两个数组已经正序，我们可以归并排序，直到找到第x个。所谓归并，就是比较两个数组第一个元素，取比较小那个。

● 代码：

```
class Solution:
    # @param {integer[]} nums1
    # @param {integer[]} nums2
    # @return {float}
    def findMedianSortedArrays(self, nums1, nums2):
        len1,len2 = len(nums1),len(nums2)
        if not(len1 or len2):return 0.0
        ln = rn = 0
        if (len1+len2)%2 == 0:
            lmid = (len1+len2)/2
            rmid = lmid + 1
        else:
            lmid = rmid = (len1+len2+1)/2

        i = j = 0
        while True:
            if j==len2 or (i < len1 and nums1[i] <= nums2[j]):
                num,i = nums1[i],i+1
            elif i == len1 or(j < len2 and nums2[j] <= nums1[i]):
                num,j = nums2[j],j+1
            if  i+j == lmid:
                ln = num
            if i+j == rmid:
                rn = num
                return (ln+rn)/2.0
```

##11. Container With Most Water （实际问题）

难度：Medium

题意：

给定一个非负整数数组，每个数字代表一个坐标(i,ai)， 每个点和对应的(i,0)，构成一条直线。任何两直线和x轴构成1个容器，求可以盛最多水的容器能盛多少水。

思路：

这里需要先弄清楚题意，容器不一定是由相邻的直线构成，另外容器的容积是受到两侧的短板，以及个直线的距离的影响。我们采用类似于贪心算法，先选取距离最长的两个板（第一个和最后一个），后每次去掉两个板中较短那个，选取下一个板。

代码：

```
class Solution:
    # @param {integer[]} height
    # @return {integer}
    def maxArea(self, height):
        h,t,result = 0,len(height)-1,0
        while h<t:
            result = max(result,min(height[h],height[t])*(t-h))
            if height[h]<height[t]:h+=1
```

```
        else:t-=1
    return result
```

##42. Trapping Rain Water （实际问题）

- 难度：Hard
- 题意：

给定一个非负数组代表一张海拔地图，每个数字对应的横向宽度为1，求这个山脉能困住的水的总量

- 思路：

每次向右探索找到第一个比当前海拔高的第一个点，若没有比当前海拔高的则返回右侧最高的。然后算面积。跳到右侧的这个点。

- 代码：

```
class Solution:
    # @param {integer[]} height
    # @return {integer}
    def trap(self, height):
        i=result=0
        while i<len(height):
            if height[i] == 0:i += 1
            else:
                right,index = self.findRight(height,i)
                if right:
                    space = (index-i-1)*min(right,height[i])-sum(height[i+1:index])
                    result += space
                    i = index
                else:
                    return result
        return result

    def findRight(self,nums,i):
        if len(nums)-i <=1:return 0,0
        ##return the first whilch bigger than nums[i]
        for index in range(i+1,len(nums)):
            if nums[index]>nums[i]:
                return nums[index],index
        ##no num bigger than nums[i],return the biggest
        max_right = max(nums[i+1::])
        return max_right,nums[i+1::].index(max_right)+i+1
```

##84. Largest Rectangle in Histogram （实际问题）

- 难度：Hard
- 题意：

给定n个非负数代表n个宽度为1的直方图的高，找到直方图中最大的矩形面积。

- 思路：

用栈记录下标，若高度是递增的，则持续入栈，直到当前高度小于栈顶，持续出栈结算计算面积，直高于栈顶再次入栈。仔细思考这个过程，很值得玩味。

- 代码：

```
class Solution(object):
    def largestRectangleArea(self, height):
        """
        :type height: List[int]
        :rtype: int
        """
        stack=[]
        height.append(0)
        i=largest=0
        while i < len(height):
            if len(stack)==0 or  height[i]>height[stack[-1]]:
                stack.append(i)
                i+=1
            else:
                tmp = stack.pop(-1)
                largest = max(largest,height[tmp]*(i if len(stack)==0 else i-stack[-1]-1))
        return largest
```

##26. Remove Duplicates from Sorted Array （删除）

- 难度：Easy
- 题意：

给定一个数组，要求就地删除重复的数组，使每个元素只出现1次，返回处理后的数组长度。不允许请新的空间。

- 思路：

常规思路，遍历数组，若当前数跟前一个树相同则删除，否则长度+1。也可以用滑动窗口法，探索重的元素，然后一次性跳过。

- 代码：

```
class Solution:
    # @param {integer[]} nums
    # @return {integer}
    def removeDuplicates(self, nums):
        if not nums:
            return 0
        lenth = i = 1
        while i < len(nums):
            if nums[i-1] != nums[i]:
                lenth += 1
                i += 1
            else:
                nums.remove(nums[i])
        return lenth
```

##80. Remove Duplicates from Sorted Array II （删除）

- 难度：Medium
- 题意：

给定一个数组，要求去除数组中重复超过2次的数字。返回新数组的的长度。不管长度后面是什么

- 思路：

题目中不管超过长度后面是什么其实已经很明显地提醒了我们，我们只需要把超过2次的交换到数组部就可以了。但是这里我的实现不是基于这个。更加简单地用一个dic记录每个数字出现次数，超过2直接remove。这样实现起来简单易懂，但是效率会低一些。

- 代码：

```
class Solution:
    # @param {integer[]} nums
    # @return {integer}
    def removeDuplicates(self, nums):
        dic = {}
        length=i= 0
        while i <len(nums):
            n = nums[i]
            if not dic.get(n):
                dic[n] = 0
            dic[n] += 1
            if dic.get(n)>2:
                nums.remove(n)
            else:
                i += 1
                length += 1
        return length
```

##27. Remove Element （删除）

- 难度：Easy
- 题意：

给定一个数组和一个值，就地删除数组中所有等于这个值的元素，返回新的数组长度。数组中元素的序可以改变。

- 思路:

题目已经暴露了思路，维护两个下标i,j分别表示i左侧均不等于value，j右侧均等于value，i右侧到j是探索区域。就是当nums[i]值等于value时，和nums[j]交换位置。有点类似于快排的思想。

- 代码：

```
class Solution:
    # @param {integer[]} nums
    # @param {integer} val
    # @return {integer}
    def removeElement(self, nums, val):
        if not nums: return 0
```

```
    lenth,i,j= len(nums),0,len(nums)-1
    while i <= j:
        if nums[i] == val:
            nums[i],nums[j] = nums[j],nums[i]
            j -= 1
            lenth -=1
        else:
            i += 1
    return lenth
```

##31. Next Permutation （排序）

- 难度：Medium
- 题意：

实现Next Permutation方法，重新排列数组，使数组的下一排列的字典序比当前数组大“1”。如果前数组字典序最大，则变化为最小。要求转换是就地的。

- 思路：

这道题干想很难想出答案，需要动手写几个例子，然后从其中找到规律：从后往前找到第一个小于后数的数的下标记为begin，再从后往前找到第一个比begin对应数字大的数的下标记为end。然后翻转egin到end之间的数字。

- 代码：

```
class Solution:
    # @param {integer[]} nums
    # @return {void} Do not return anything, modify nums in-place instead.
    def nextPermutation(self, nums):
        if  nums and len(nums)!=1:
            begin=-1
            for i in range(len(nums)-1,0,-1):
                if nums[i-1]<nums[i]:
                    begin = i-1
                    break
            if begin == -1:
                nums.sort()
            else:
                for i in range(len(nums)-1,begin,-1):
                    if nums[i]>nums[begin]:
                        end = i
                        break
                nums[begin],nums[end] = nums[end],nums[begin]
                begin,end = begin+1,len(nums)-1
                while begin<=end:
                    nums[begin],nums[end] = nums[end],nums[begin]
                    begin+=1
                    end-=1
```

##33. Search in Rotated Sorted Array （旋转数组）

- 难度：Hard
- 题意：

一个正序数组，可能从以某个节点进行翻转（123456可能变成456123），假定数组中不存在重复数字给定一个值，求其index。

- 思路：

找到异常点i，nums[0:i]和nums[i:len]都是递增的，这样便可确定value落在哪个范围内，然后进行分查找即可

- 代码：

```
class Solution:
    # @param {integer[]} nums
    # @param {integer} target
    # @return {integer}
    def search(self, nums, target):
        flag = -1
        for i in range(len(nums)-1):
            if nums[i]>nums[i+1]:
                flag = i
                break

        if flag == -1:
            return self.binarySearch(nums,0,len(nums)-1,target)
        else:
            if target == nums[0]: return 0
            elif nums[0]<target<nums[flag]:
                return self.binarySearch(nums,0+1,flag-1,target)
            elif target == nums[flag]:return flag
            else:
                return self.binarySearch(nums,flag+1,len(nums)-1,target)

    #binary search
    def binarySearch(self,nums,i,j,n):
        if n<nums[i] or n>nums[j] or i>j:
            return -1
        mid = (i+j)//2
        if n == nums[mid]:
            return mid
        elif n < nums[mid]:
            return self.binarySearch(nums,i,mid-1,n)
        else:
            return self.binarySearch(nums,mid+1,j,n)
```

- 思路2：

思路1其实把问题退化了，有点儿取巧。我们也可以直接进行二分查找，只不过在判断target在左空还是右空间的时候，情况复杂了一点儿，需要先判断mid在旋转数组的哪个位置。

- 代码2：

```
class Solution:
    # @param {integer[]} nums
    # @param {integer} target
```

```
    # @return {integer}
    def search(self, nums, target):
        low,high = 0,len(nums)-1
        while low<=high:
            mid = (low+high)//2
            if nums[mid]==target:return mid
            #在旋转点的左侧
            if nums[mid]>nums[low]:
                if nums[low]<=target<=nums[mid]:
                    high = mid-1
                else:
                    low = mid+1
            #在旋转点右侧
            elif nums[mid]<nums[low]:
                if nums[mid]<=target<=nums[high]:
                    low = mid+1
                else:
                    high = mid-1
            else:low+=1
        return -1
```

##81. Search in Rotated Sorted Array II (旋转数组)

- 难度：Medium
- 题意：

一个正序数组，可能从以某个节点进行翻转（123456可能变成456123）,假定数组中可能存在重复数，给定一个值，求其index。

- 思路：

33题两种思路均可以处理重复问题。

- 代码：

```
class Solution:
    # @param {integer[]} nums
    # @param {integer} target
    # @return {integer}
    def search(self, nums, target):
        flag = -1
        for i in range(len(nums)-1):
            if nums[i]>nums[i+1]:
                flag = i
                break

        if flag == -1:
            return self.binarySearch(nums,0,len(nums)-1,target)
        else:
            if target == nums[0]: return True
            elif nums[0]<target<nums[flag]:
                return self.binarySearch(nums,0+1,flag-1,target)
            elif target == nums[flag]:return True
```

```
        else:
            return self.binarySearch(nums,flag+1,len(nums)-1,target)

    #binary search
    def binarySearch(self,nums,i,j,n):
        if n<nums[i] or n>nums[j] or i>j:
            return False
        mid = (i+j)//2
        if n == nums[mid]:
            return True
        elif n < nums[mid]:
            return self.binarySearch(nums,i,mid-1,n)
        else:
            return self.binarySearch(nums,mid+1,j,n)
```

##153. Find Minimum in Rotated Sorted Array（旋转数组）

- 难度：Medium
- 题意：

给定一个从某个点旋转的正序数组，找到其中的最小元素。数组中不存在重复数字。

- 思路：

非递增点出现的地方为最小元素。虽然找非递增点一定程度减少了复杂度，但是平均下来也是O(n)。道题考察的目的是直接对数组进行二分查找。其实只要画图相信一下就会发现，二分可能出现的情况具体分析我在代码注释中给出。

- 代码：

```
class Solution:
    # @param {integer[]} nums
    # @return {integer}
    def findMin(self, nums):
        low,hight=0,len(nums)-1
        while low<=hight:
            ##最后指向同一个元素，或指向一个正序区间
            if nums[low]<=nums[hight]:
                return nums[low]
            else:
                mid=(low+hight)//2
                ##旋转点在mid右侧
                if nums[mid]>=nums[low]:
                    low=mid+1
                ##旋转点恰好是mid或在mid左侧
                else:
                    if nums[mid]<nums[mid-1]:
                        return nums[mid]
                    hight=mid-1
```

##154. Find Minimum in Rotated Sorted Array II （旋转数组）

- 难度：Hard
- 题意：

给定一个从某个点旋转的正序数组，找到其中的最小元素。数组中可能存在重复数字。

- 思路：

这道题情况就要比不重复的复杂得多。

  - 若nums[mid]<nums[high]：旋转点在mid左侧
  - 若nums[mid]>nums[high]：旋转点在mid右侧
  - 若nums[mid]==nums[high]:无法判断在哪一侧，只能两侧递归二分查找

但是最坏情况（都是重复数字）时间复杂度也是O(n)，所以我选择线性探索。

- 代码：

```
class Solution:
    # @param {integer[]} nums
    # @return {integer}
    def findMin(self, nums):
        if not nums:return
        for i in range(len(nums)-1):
            if nums[i]>nums[i+1]:
                return nums[i+1]
        return nums[0]
```

##189. Rotate Array （旋转数组）

- 难度：Easy
- 题意：

对一个有n个元素的数组从右边第k个元素开始旋转。

- 思路：

比较简单，拆分成两个数组再组合。或者循环从最后弹出再插入到最前。

- 代码：

```
class Solution:
    # @param {integer[]} nums
    # @param {integer} k
    # @return {void} Do not return anything, modify nums in-place instead.
    def rotate(self, nums, k):
        for i in range(k):
            nums.insert(0,nums.pop(-1))
```

##34. Search for a Range （查找）

- 难度：Medium

- 题意：

给定一个正序数组和值v，求v的起始和结束index。要求时间复杂度在O(logn)内。

- 思路：

这个时间复杂度用二分查找正好，先用二分查找找到value的位置，然后分别向左和向右查找value（是使用二分）直到找不到为止。

- 代码：

```
class Solution:
    # @param {integer[]} nums
    # @param {integer} target
    # @return {integer[]}
    def searchRange(self, nums, target):
        if not nums:return [-1,-1]
        begin = end = -1
        flag = self.binarySearch(nums,0,len(nums)-1,target)
        i = flag
        while i !=-1:
            begin = i
            i = self.binarySearch(nums,0,i-1,target)
        i = flag
        while i!=-1:
            end = i
            i = self.binarySearch(nums,i+1,len(nums)-1,target)
        return [begin,end]


    #binary search
    def binarySearch(self,nums,i,j,n):
        if i>j or n<nums[i] or n>nums[j]:
            return -1
        mid = (i+j)//2
        if n == nums[mid]:
            return mid
        elif n < nums[mid]:
            return self.binarySearch(nums,i,mid-1,n)
        else:
            return self.binarySearch(nums,mid+1,j,n)
```

##35. Search Insert Position （查找）

- 难度：Medium
- 题意：

给一个正序数组和目标value，如果目标值存在数组中，返回其索引，否则，返回一个插入点，使插后依然正序。假定数组中没有重复的值。

- 思路：

这道题是二分查找的升级版本，稍微改动一下返回即可。若找到返回index，若没有找到，返回小于value的最大值的index

- 代码：

```
class Solution:
    # @param {integer[]} nums
    # @param {integer} target
    # @return {integer}
    def searchInsert(self, nums, target):
        return self.binarySearch(nums,0,len(nums)-1,target)

    #binary search
    def binarySearch(self,nums,i,j,n):
        if i > j:return i
        if n<nums[i]:return i
        if n>nums[j]:return j+1
        mid = (i+j)//2
        if n == nums[mid]:
            return mid
        elif n < nums[mid]:
            return self.binarySearch(nums,i,mid-1,n)
        else:
            return self.binarySearch(nums,mid+1,j,n)
```

##41. First Missing Positive （查找）

- 难度：Hard
- 题意：

给定一个未排序数组，求第一个丢失的正整数。要求时间复杂度为O（n），空间复杂度为O（1）

- 思路：

这道题的难度在于时间和空间复杂度的限制，不能排序，不能使用额外的空间。但是我们有更好的办，遍历数组把数字交换到其对应的位置（value-1）,最后遍历一遍就可以发现缺失的数字。

- 代码：

```
class Solution:
    # @param {integer[]} nums
    # @return {integer}
    def firstMissingPositive(self, nums):
        if not nums:return 1
        index = 0
        while index<len(nums):
            if nums[index]>0 and nums[index]-1!=index and nums[index]-1<len(nums) and nums
index]!=nums[nums[index]-1]:
                indexb = nums[index]-1
                nums[index],nums[indexb] = nums[indexb], nums[index]
            else:
                index += 1
        for i in range(len(nums)):
            if i+1 != nums[i]:
                return i+1
```

```
        return len(nums)+1
```

##55. Jump Game （贪心）

- 难度：Medium
- 题意：

给定一组非负数，最开始处于数组的开始位置，每个数字代表从该位置出发你可以跳跃的最远距离，问是否能到达最后一个索引。

- 思路：

贪心算法，我们并不需要知道确切的跳跃路径，我们只需要知道能不能跳到这里就可以了。遍历数组同时记录能到达的最远索引，若当前索引大于最远索引，则意味着不可能到达该未知，因此更不可能到最后一个索引。

- 代码：

```
class Solution:
    # @param {integer[]} nums
    # @return {boolean}
    def canJump(self, nums):
        if not nums:return False
        maxstep = 0
        for i,n in enumerate(nums):
            if i>maxstep:return False
            maxstep = max(maxstep,i+n)
        return True
```

##45. Jump Game II （贪心）

- 难度：Hard
- 题意：

给定一组非负数，最开始处于数组的开始位置，每个数字代表从该位置出发你可以跳跃的最远距离，问到达最后一个索引的最少跳跃次数。

- 思路：

同样的也是贪心算法，这道题让你以为要求出确切的路径（当然用dp是可以求的），然而并不需要。们只知道每次要跳跃就跳到最远，然后进入下一个区间再跳到最远即可。因此使用maxPos记录能达的最远索引，用rightPost记录当前这次跳跃能到达最右侧索引，若i>rightPost，表明需要再次跳跃跳跃到maxPos，step+1。

- 代码：

```
class Solution:
    # @param {integer[]} nums
    # @return {integer}
    def jump(self, nums):
        step = maxPos = rightPos =0
        for i,n in enumerate(nums):
```

```
        if i>rightPos:
            step,rightPos = step+1,maxPos
        maxPos = max(maxPos,i+n)
    return step
```

##48. Rotate Image （矩阵）

- 难度：Medium
- 题意：

给定一个n*n 矩阵，将矩阵顺时针旋转90度，最好能就地解决。

- 思路：

这道题处理好(i,j)和旋转后(i,j)的映射关系就可以了。我没有就地解决。。。

- 代码：

```
class Solution:
    # @param {integer[][]} matrix
    # @return {void} Do not return anything, modify matrix in-place instead.
    def rotate(self, matrix):
        tmp = [[matrix[i][j] for i in range(len(matrix)-1,-1,-1)] for j in range(len(matrix))]
        for i in range(len(matrix)):
            matrix[i]=tmp[i]
```

##54. Spiral Matrix （矩阵）

- 难度：Medium
- 题意：

给定一个m*n矩阵（m行，n列），以螺旋顺序返回所有元素

- 思路：

结合矩阵旋转那道题，我们可以比较快想到，每次从左到右读第一行，然后去掉第一行再逆时针旋转9度。如此循环。

- 代码：

```
class Solution:
    # @param {integer[][]} matrix
    # @return {integer[]}
    def spiralOrder(self, matrix):
        result = []
        while matrix:
            for i in range(len(matrix[0])):
                result.append(matrix[0][i])
            matrix = matrix[1:]
            matrix = self.rotate(matrix)
        return result
```

```
    def rotate(self, matrix):
        if not matrix: return
        tmp = [[matrix[i][j] for i in range(len(matrix))] for j in range(len(matrix[0])-1,-1,-1)]
        return tmp
```

##59. Spiral Matrix II （矩阵）

- 难度：Medium
- 题意：

给定一个正整数，产生一个矩阵将1到n^2以螺旋形式填充其中。

- 思路：

这道题敲好是54的逆过程（洋葱剥皮和重新包皮）。我们使用两个矩阵result是最终的结果。matrix录横纵坐标。每次区matrix第一行的坐标，按顺序往result对应的坐标内填入数字。然后去掉matrix一行，再逆时针旋转90度。如此循环。

- 代码：

```
class Solution:
    # @param {integer} n
    # @return {integer[][]}
    def generateMatrix(self, n):
        result= [[0 for i in range(n)] for j in range(n)]
        matrix= [[(i,j) for j in range(n)] for i in range(n)]
        count = 1
        while matrix:
            for i in range(len(matrix[0])):
                result[matrix[0][i][0]][matrix[0][i][1]]=count
                count += 1
            matrix = matrix[1:]
            matrix =self.rotate(matrix)
        return result

    def rotate(self, matrix):
        if not matrix: return
        tmp = [[matrix[i][j] for i in range(len(matrix))] for j in range(len(matrix[0])-1,-1,-1)]
        return tmp
```

##53.Maximum Subarray （子数组）

- 难度：Medium
- 题意：

从给定数组中找到和最大的联系子数组（至少包含1个数字）

- 思路：

典型的动态规划题目。我们定义dp[i]=s，表示以i为子数组结尾的子数组最大和为s。着有以下关系：

  - 若dp[i-1]>0，则在此基础上加上nums[i]:dp[i]=dp[i-1]+nums[i]

- 若dp[i-1]<=0，则另立门户，从nums[i]开始:dp[i]=nums[i]
- 代码：

```
class Solution:
    # @param {integer[]} nums
    # @return {integer}
    def maxSubArray(self, nums):
        if not nums: return 0
        result = [nums[0]]
        for i in range(1,len(nums)):
            result.append(nums[i] if result[i-1]<=0 else result[i-1]+nums[i])
        return max(result)
```

##56. Merge Intervals （间隔）

- 难度：Hard
- 题意：

给一组间隔，合并所有重叠的间隔

- 思路：

典型间隔问题。使用一个列表results存储不重叠的间隔。循环从原间隔中取出间隔，和results中的间比较，若存在重叠，将重叠取出合并，再循环对比，直到不存在重叠，将间隔放入results中。

- 代码：

```
# Definition for an interval.
# class Interval:
#     def __init__(self, s=0, e=0):
#         self.start = s
#         self.end = e

        class Solution:
            # @param {Interval[]} intervals
            # @return {Interval[]}
            def merge(self, intervals):
                results = []
                for interval in intervals:
                    flag = True
                    while flag:
                        flag = False
                        for result in results:
                            if (interval.end-interval.start)+(result.end-result.start)>=max(interval.end,re
ult.end)-min(interval.start,result.start):
                                results.remove(result)
                                interval.start = min(interval.start,result.start)
                                interval.end = max(interval.end,result.end)
                                flag = True
                                break
                    results.append(interval)
```

```
        return results
```

##57. Insert Interval （间隔）

- 难度：Hard
- 题意：

给定一个间隔列表，间隔不重叠，以间隔开始位置正序，给定一个新的间隔，插入到该列表中，若有要合并间隔使间隔不重合。

- 题意：

和56思路一致，循环比较，若存在重叠将重叠的间隔拿出与插入间隔合并，再次循环比较直到不存在叠为止。需要注意插入后，可能导致间隔排序乱掉。

- 代码：

```
# Definition for an interval.
# class Interval:
#     def __init__(self, s=0, e=0):
#         self.start = s
#         self.end = e

class Solution:
    # @param {Interval[]} intervals
    # @param {Interval} newInterval
    # @return {Interval[]}
    def insert(self, intervals, newInterval):
        flag = True
        while flag:
            flag = False
            for interval in intervals:
                if (newInterval.end-newInterval.start)+(interval.end-interval.start)>=max(newInterv
l.end,interval.end)-min(newInterval.start,interval.start):
                    intervals.remove(interval)
                    newInterval.start = min(newInterval.start,interval.start)
                    newInterval.end = max(newInterval.end,interval.end)
                    flag = True
                    break
        intervals.append(newInterval)
        intervals.sort(key= lambda x:x.start)
        return intervals
```

##62. Unique Paths （非降路径）

- 难度：Medium
- 题意：

一个机器人位于m*n表格的左上角，机器人每次只能向右或者向下移动一格，机器人尝试达到表格的下角。求问有多少不同的路径？

- 思路：

非降路径数学题。答案是：C((m+n-2),(m-1))。当然也可以使用动态规划，下一题我们再展示非降路的动态规划。

- 代码：

```
class Solution:
    # @param {integer} m
    # @param {integer} n
    # @return {integer}
    def uniquePaths(self, m, n):
        a = b = 1
        for i in range(n,m+n-1):
            a *= i
        for i in range(1,m):
            b *= i
        return a//b
```

##63. Unique Paths II （非降路径）

- 难度：Medium
- 题意：

一个机器人位于m*n表格的左上角，机器人每次只能向右或者向下移动一格，机器人尝试达到表格的下角。但是路径上有一些节点是不允许通过的，我们使用0表示能通过，1表示不能通过，求问有多少同的路径？

- 思路：

使用动态规划，我们定义dp[i][j]表示到达(i,j)的路径数量。由于只能向右或者向下移动，因此dp[i][j]=p[i-1][j]+dp[i][j-1]

- 代码：

```
class Solution:
    # @param {integer[][]} obstacleGrid
    # @return {integer}
    def uniquePathsWithObstacles(self, obstacleGrid):
        result = [[0 for j in range(len(obstacleGrid[0]))] for i in range(len(obstacleGrid))]
        for i,row in enumerate(obstacleGrid):
            for j,n in enumerate(row):
                if n==1:
                    if i==0 and j==0:
                        return 0
                    else:
                        result[i][j] = 0
                else:
                    if i==0 and j==0:
                        result[i][j] = 1
                    elif i==0:
                        result[i][j] = result[i][j-1]
                    elif j==0:
```

```
                result[i][j] = result[i-1][j]
            else:
                result[i][j] = result[i-1][j]+result[i][j-1]
    return result[-1][-1]
```

##64. Minimum Path Sum （非降路径）

- 难度：Medium
- 题意：

给定一个m*n的表格，每个格填非负整数，找到从左上角到右下角路径中数字和最小的路径。每次只向下或向右移动一格。

- 思路：

还是dp，跟62、63思路一致。

- 代码：

```
class Solution:
    # @param {integer[][]} grid
    # @return {integer}
    def minPathSum(self, grid):
        m,n=len(grid),len(grid[0])
        result = [[grid[i][j] for j in range(n)] for i in range(m)]
        for i in range(m):
            for j in range(n):
                if i==0 and j==0:
                    result[i][j] = grid[i][j]
                elif j==0:
                    result[i][j] += result[i-1][j]
                elif i == 0:
                    result[i][j] += result[i][j-1]
                else:
                    result[i][j] += min(result[i-1][j],result[i][j-1])
        return result[-1][-1]
```

##66. Plus One （数学）

- 难度：Easy
- 题意：

给定一个用数组表示的非负整数，高位在左。求这个数字加1后的值，同样也是使用数组表示。

- 难度：

从后往前遍历，记录进位，若最高位有进位需要在最前面插入。

- 代码：

```
class Solution:
    # @param {integer[]} digits
```

```
    # @return {integer[]}
    def plusOne(self, digits):
        if not digits:return digits
        c,digits[-1] = 0,digits[-1]+1
        for i in range(len(digits)-1,-1,-1):
            digits[i] += c
            if digits[i] == 10:
                digits[i],c = 0,1
            else:
                c = 0
                break
        if c == 1:
             d = [1]
             d.extend(digits)
             digits = d
        return digits
```

##73. Set Matrix Zeroes （矩阵）

- 难度：Medium
- 题意：

给定一个m*n矩阵，如果某个位置元素为0，则设置整行和整列为0，要求原地解决。

- 思路：

这道题的难度在于，若你检查到0就去设置行和列的话，后面遍历的时候就无法判断出现的0是原来的是后置的。但是当发现0的时候只需要把第0行对应列号的位置设为0，把第0列队列行号的位置设置为，就可以了。但是这样还有一个问题，若第0行或第0列原来就出现0呢？我们再用两个变量记录就可了。

- 代码：

```
class Solution:
    # @param {integer[][]} matrix
    # @return {void} Do not return anything, modify matrix in-place instead.
    def setZeroes(self, matrix):
        row0 = cel0 = False
        for i in range(len(matrix)):
            for j in range(len(matrix[0])):
                if matrix[i][j] == 0:
                    if i == 0:
                        row0 = True
                    if j == 0:
                        cel0 = True
                    matrix[i][0] = 0
                    matrix[0][j] = 0
        for i in range(1,len(matrix)):
            if matrix[i][0] == 0:
                for j in range(1,len(matrix[0])):
                    matrix[i][j] = 0
        for j in range(1,len(matrix[0])):
            if matrix[0][j] == 0:
```

```
            for i in range(1,len(matrix)):
                matrix[i][j] = 0
        if row0:
            for j in range(1,len(matrix[0])):
                matrix[0][j] = 0
        if cel0:
            for i in range(1,len(matrix)):
                matrix[i][0] = 0
```

##74. Search a 2D Matrix（查找）

- 难度：Medium
- 题意：

写一个在m*n二维数组中高效查找的算法。这个二维矩阵有如下规律：

  - 每一行都是正序的
  - 每一个行的第一个数字也是正序的

- 思路：

先用二分查找确定行，注意这里的二分查找是要返回<=value的最大值。再用二分查找在行中搜索。

- 代码：

```
class Solution:
# @param {integer[][]} matrix
# @param {integer} target
# @return {boolean}
def searchMatrix(self, matrix, target):
    row = self.binaryRows(matrix,0,len(matrix)-1,target)
    if not row:
        return False
    else:
        return self.binaryInRow(row,0,len(row)-1,target)

# find out which row may be fit
# return a[i] which a[i][0]<=target and a[i+1][0]> target
def binaryRows(self,matrix,i,j,target):
    if not matrix or matrix[i][0]> target or i > j:
        return None
    mid = (i+j)//2
    if matrix[mid][0] == target:
        return matrix[mid]
    elif matrix[mid][0]< target:
        if mid+1 < len(matrix):
            if matrix[mid+1][0]>target:
                return matrix[mid]
            else:
                return self.binaryRows(matrix,mid+1,j,target)
        else:
            return matrix[mid]
```

```
        elif matrix[mid][0]> target:
            return self.binaryRows(matrix,i,mid-1,target)

    def binaryInRow(self,row,i,j,target):
        if not row or row[i]>target or row[j]<target or i>j:
            return False
        mid = (i+j)//2
        if row[mid] == target:
            return True
        elif row[mid] < target:
            return self.binaryInRow(row,mid+1,j,target)
        else:
            return self.binaryInRow(row,i,mid-1,target)
```

##75. Sort Colors （排列）

- 难度：Medium
- 题意：

给定一个数组，有n各元素被涂颜色：红、白、蓝，对他们进行排序时相同的颜色是相邻的，且颜色顺序是红、白、蓝。现在我们使用0、1、2代表红、白、蓝。不允许使用内置库中的排序方法解决这问题。

- 思路：

有多少种排序方法，这道题就有多少种解法。然而，切合这道题，有几种比较方便的解放，如计数法。这里我采用了一种类似插入排序的方法。由于只有3种颜色，我维护i,j,k把数组切分成0，1，2及未索4部分。遍历的时候插入到对应的位置即可。

- 代码：

```
class Solution:
    # @param {integer[]} nums
    # @return {void} Do not return anything, modify nums in-place instead.
    def sortColors(self, nums):
        if not nums or len(nums)<2:return
        i,j,k = -1,-1,0
        while k<len(nums):
            if nums[k] == 0:
                i,j = i+1,j+1
                if i!=j:
                    nums[k],nums[i],nums[j] = nums[j],nums[k],nums[i]
                else:
                    nums[k],nums[i] = nums[i],nums[k]
            elif nums[k] == 1:
                j+=1
                nums[j],nums[k] = nums[k],nums[j]
            k+=1
```

##78. Subsets （子集）

- 难度：Medium
- 题意：

给定一个集合包含不一样的数字，求所有可能的子集。要求子集中数字是正序的，且子集之间必须不同。

- 思路：

使用递归就可以了。和求和的思路是一样的。对这类题，我总是把剩下的集合传递过去，其实完全没必要，只需要传递过去当前选取数字的index即可，有右遍历。这样还能避免相同的子集出现。

- 代码：

```
class Solution:
    # @param {integer[]} nums
    # @return {integer[][]}
    def subsets(self, nums):
        self.result = [[]]
        nums.sort()
        self.reduce(nums,[])
        return self.result

    def reduce(self,nums0,ele0):
        nums = nums0[:]
        ele = ele0[:]
        if ele:
            self.result.append(ele)
        for i,n in enumerate(nums):
            ele.append(nums.pop(i))
            self.reduce(nums[i:],ele)
            nums.insert(i,n)
            ele.pop(-1)
```

##90. Subsets II （子集）

- 难度：Medium
- 题意：

给定一个数组代表的一个集合，可能包含重复数字，返回所有可能的子集。

- 思路：

和上一题的思路一模一样，同样是递归回溯就可以解决了。

- 代码：

```
class Solution:
    # @param {integer[]} nums
    # @return {integer[][]}
    def subsetsWithDup(self, nums):
        self.result = []
        self.dic = {}
        nums.sort()
        self.reduce(nums,[])
```

```
        return self.result

    def reduce(self,nums0,ele0):
        nums = nums0[:]
        ele = ele0[:]
        if not self.dic.get(str(ele)):
            self.dic[str(ele)] = 1
            self.result.append(ele)
        for i,n in enumerate(nums):
            ele.append(nums.pop(i))
            self.reduce(nums[i:],ele)
            nums.insert(i,n)
            ele.pop(-1)
```

##79. Word Search （矩阵）

- 难度：Medium
- 题意：

给定一个填满字母矩阵，求问单词是否存在矩阵中。相邻的字母可以构成单词，相邻包括上下左右，同的位置只能使用1次。

- 思路：

使用递归，使用过的位置用特殊符号标记。

- 代码：

```
class Solution:
    # @param {character[][]} board
    # @param {string} word
    # @return {boolean}
    def exist(self, board, word):
        board = [list(board[i]) for i in range(len(board))]
        if self.reduce(board,word,-1,-1):
            return True
        else:
            return False

    def reduce(self,board,word,i,j):
        if not word:
            return True
        candidate = self.search(board,word[0],i,j)
        for i,j in candidate:
            board[i][j] = '.'
            if self.reduce(board,word[1:],i,j):
                return True
            board[i][j] = word[0]

    def search(self,board,char,i,j):
        result = []
        if i==j==-1:
            ##first search
```

```
        for i in range(len(board)):
            for j in range(len(board[0])):
                if board[i][j] == char:
                    result.append((i,j))
    else:
        if i-1>=0 and board[i-1][j] == char:
            result.append((i-1,j))
        if j-1>=0 and board[i][j-1] == char:
            result.append((i,j-1))
        if i+1<=len(board)-1 and board[i+1][j] == char:
            result.append((i+1,j))
        if j+1<=len(board[0])-1 and board[i][j+1] == char:
            result.append((i,j+1))
    return result
```

##212. Word Search II （字典树）

- 难度：Hard
- 题意：

题意完全和79一直，不过输入是一组字符串，要求输出存在的字符串。

- 思路：

这道题和79最大的区别在于，若输入规模很大，则会有很多重复的递归操作。完全没有考虑到输入之可以能存在关系。因此我们应该对输入单词建立字典树，是字典树进行dfs，使用字母矩阵对字典树行剪枝。

- 代码：

```
class TrieNode:
    def __init__(self):
        self.childs = dict()
        self.isWord = False

class Trie:
    def __init__(self):
        self.root = TrieNode()

    def insert(self,word):
        node = self.root
        for letter in word:
            child = node.childs.get(letter)
            if not child:
                child = TrieNode()
                node.childs[letter] = child
            node = child
        node.isWord = True

    def delete(self,word):
        node = self.root
        queue = []
        for letter in word:
```

```
            queue.append((letter,node))
            child = node.childs.get(letter)
            if not child:
                return False
            node = child
        if not node.isWord:
            return False
        if len(node.childs):
            node.isWord = False
        else:
            for letter,node in queue[::-1]:
                del node.childs[letter]
                if len(node.childs) or node.isWord:
                    break
        return True


class Solution:
    # @param {character[][]} board
    # @param {string[]} words
    # @return {string[]}
    def findWords(self, board, words):
        w, h = len(board[0]), len(board)
        trie = Trie()
        for word in words:
            trie.insert(word)

        visited = [[False for j in range(w)] for i in range(h)]
        dz = zip([1, 0, -1, 0], [0, 1, 0, -1])
        ans = []

        def dfs(word, node, x, y):
            node = node.childs.get(board[x][y])
            if node is None:
                return
            visited[x][y] = True
            for z in dz:
                nx, ny = x + z[0], y + z[1]
                if nx >= 0 and nx < h and ny >= 0 and ny < w and not visited[nx][ny]:
                    dfs(word + board[nx][ny], node, nx, ny)
            if node.isWord:
                ans.append(word)
                trie.delete(word)
            visited[x][y] = False

        for x in range(h):
            for y in range(w):
                dfs(board[x][y], trie.root, x, y)

        return sorted(ans)
```

##88. Merge Sorted Array （排序）

- 难度：Easy
- 题意：

给定两个正序数组num1和num2，将num2合并到num1中，并排序。

- 思路：

在num1中找num2中各个数字的插入点。

- 代码：

```
class Solution:
    # @param {integer[]} nums1
    # @param {integer} m
    # @param {integer[]} nums2
    # @param {integer} n
    # @return {void} Do not return anything, modify nums1 in-place instead.
    def merge(self, nums1, m, nums2, n):
        while m<len(nums1):
            nums1.pop(-1)
        i = j = 0
        while i<m and j<n:
            if nums2[j]<=nums1[i]:
                nums1.insert(i,nums2[j])
                i,j,m = i+1,j+1,m+1
            else:
                i += 1
        nums1.extend(nums2[j:n])
```

##105. Construct Binary Tree from Preorder and Inorder Traversal （树）

- 难度：Medium
- 题意：

给定二叉树的先根遍历和中根遍历数组，要求建立起该二叉树。

- 思路：

先根遍历（根左右）、中跟遍历（左根右）。因此可以利用根的位置，把中跟遍历切割成左和右两个分，然后递归建立左子树和右子树。

- 代码：

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

class Solution:
    # @param {integer[]} preorder
    # @param {integer[]} inorder
    # @return {TreeNode}
```

```
    def buildTree(self, preorder, inorder):
        if not inorder:return None
        n = preorder[0]
        i = inorder.index(n)
        preorder.remove(n)
        root = TreeNode(n)
        root.left = self.buildTree(preorder,inorder[:i])
        root.right = self.buildTree(preorder,inorder[i+1:])
        return root
```

##106. Construct Binary Tree from Inorder and Postorder Traversal （树）

- 难度：Easy
- 题意：

给定二叉树的中根遍历和后根遍历数组，要求建立起该二叉树。

- 思路：

中根遍历（左根右）、后根遍历（左右根），因此和105思路一致，利用后跟遍历中根的位置，把中遍历切割成左和右两个部分，然后递归建立左子树和右子树。

- 代码：

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

class Solution:
    # @param {integer[]} inorder
    # @param {integer[]} postorder
    # @return {TreeNode}
    def buildTree(self, inorder, postorder):
        if not inorder:return None
        n = postorder[-1]
        i = inorder.index(n)
        postorder.remove(n)
        root = TreeNode(n)
        root.right = self.buildTree(inorder[i+1:],postorder)
        root.left = self.buildTree(inorder[:i],postorder)
        return root
```

##118. Pascal's Triangle （三角）

- 难度：Easy
- 题意：

给定行数numRows，生成前numRows行帕斯卡三角。

- 思路：

找规律。对于每一行，最左和最右一个数字是1，其他的数字有如下关系 res[i][j]=res[i-1][j-1]+res[i-1[i]

- 代码：

```
class Solution:
    # @param {integer} numRows
    # @return {integer[][]}
    def generate(self, numRows):
        result = []
        for row in range(numRows):
            tmp = [1]
            for i in range(1,row):
                tmp.append(result[-1][i-1]+result[-1][i])
            if row!=0:tmp.append(1)
            result.append(tmp)
        return result
```

##119. Pascal's Triangle II （三角）

- 难度：Easy
- 题意：

给定一个索引k，球帕斯卡三角的第k行。

- 思路：

和上一题思路是一样的，只不过这次只需要记录上一行即可。

- 代码：

```
class Solution:
    # @param {integer} rowIndex
    # @return {integer[]}
    def getRow(self, rowIndex):
        pre=[]
        result=[]
        for row in range(rowIndex+1):
            result = [1]
            for i in range(1,row):
                result.append(pre[i-1]+pre[i])
            if row!=0:result.append(1)
            pre = result[::]
        return result
```

##120. Triangle （三角）

- 难度：Medium
- 题意：

给定一个三角，找到从顶端到底端路径上数字之和最小的路径。

- 思路：

典型的动态规划。我们定义dp[i][j]=s为终点为(i,j)的路径之和最小为s。对于除第一个节点外的每个节都有两个相邻的前驱节点，因此有如下关系：

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

- 代码：

```
class Solution:
    # @param triangle, a list of lists of integers
    # @return an integer
    def minimumTotal(self, triangle):
        for r in range(len(triangle)-2,-1,-1):
            for i in range(0,len(triangle[r])):
                triangle[r][i] += min(triangle[r+1][i],triangle[r+1][i+1])
        return triangle[0][0]
```

##121. Best Time to Buy and Sell Stock （股票）

- 难度：Medium
- 题意：

给定一个数组其中第i个数字代表第i天股票的价格。假定，你只允许完成一次交易，设计算法找到最的收益.

- 思路：

典型动态规划。我们定义dp[i]=s 表示在第i天卖出股票的最大收益。则有以下关系：

  - 股票在第i-1天已经卖出，我们在第i-1天重新买入，在第i天重新卖出
  - 之前没有进行任何交易，在第i-1天买入，在第i天卖出

因此，dp[i]= max(dp[i-1]-prices[i-1]+prices[i],prices[i]-prices[i-1])。发现dp[i]只和dp[i-1]有关，所有只需记录前一dp的值即可，不必要用数组存储。

- 代码：

```
class Solution:
    # @param {integer[]} prices
    # @return {integer}
    def maxProfit(self, prices):
        if len(prices)<2:return 0
        profit = [0]
        maxProfit = 0
        for i in range(1,len(prices)):
            profit.append(max(profit[-1]+prices[i]-prices[i-1],prices[i]-prices[i-1]))
            maxProfit=max(maxProfit,profit[-1])
        return maxProfit
```

##122. Best Time to Buy and Sell Stock II （股票）

- 难度：Medium
- 题意：

给定一个数组其中第i个数字代表第i天股票的价格。你可以交易任意多次数，设计算法找到最大的收。

- 思路：

不限制交易次数的思路反而容易得多，只要明天涨，我今天就买入，明天直接卖出。

- 代码：

```
class Solution:
    # @param {integer[]} prices
    # @return {integer}
    def maxProfit(self, prices):
        profit=0
        for i in range(1,len(prices)):
            if prices[i]>prices[i-1]:
                profit+=prices[i]-prices[i-1]
            else:
                continue
        return profit
```

##123. Best Time to Buy and Sell Stock III （股票）

- 难度：Hard
- 题意：

给定一个数组其中第i个数字代表第i天股票的价格。你可以交易最多2次数，设计算法找到最大的收益。

- 思路：

这道题的难度就比较大了。也是使用动态规划，但是需要把问题划分为两个子问题。left[i]表示第i天出最大的收益，right[i]表示第i天后交易的最大收益。left[i]的递推公式我们在121题已经给出了。right i]如何求呢，right[i]需要重后往前遍历，维护maxPrices=price[i:]，right[i]有两种选择，在第i天买入或不在第i天买入。right[i]=max(right[i+1],maxPrices-prices[i])。最后left+right，最大值即为所求。

- 代码：

```
class Solution:
    # @param {integer[]} prices
    # @return {integer}
    def maxProfit(self, prices):
        if len(prices)<2:return 0
        left=[0]
        right=[0]
        for i in range(1,len(prices)):
            left.append(max(prices[i]-prices[i-1],prices[i]-prices[i-1]+left[-1]))
        maxPrices = prices[-1]
        for i in range(len(prices)-2,-1,-1):
            maxPrices = max(maxPrices,prices[i])
```

```
        right.insert(0,max(right[0],maxPrices-prices[i]))
    left = [left[i]+right[i] for i in range(len(prices))]
    return max(left)
```

##128. Longest Consecutive Sequence （查找）

- 难度：Hard
- 题意：

给定一个未排序数组，找到最长连续序列。

- 思路：

这道题的难度在于，对每个数向前先后搜索的复杂度太高。因此为了方便查找，我使用集合来存储这数组。每次拿到一个数字，循环向左和先有搜索，每次搜索到就从集合中删除，继续向两侧搜索。

- 代码：

```
class Solution(object):
    def longestConsecutive(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        self.s = set(nums)
        res=0
        while self.s:
            n=self.s.pop()
            res = max(self.search(n-1,True)+self.search(n+1,False)+1,res)
        return res

    def search(self,n,asc):
        res=0
        while n in self.s:
            self.s.remove(n)
            res+=1
            if asc:n-=1
            else:n+=1
        return res
```

##152. Maximum Product Subarray （子集）

- 难度：Medium
- 题意：

给定一个数组，找到乘积最大的子数组。

- 思路：

这道题第一反应就是动态规划。但是这里头注意有个坑：负负得正。因此还得记录最小值。其他的就常规了。

- 代码：

```
class Solution:
    # @param {integer[]} nums
    # @return {integer}
    def maxProduct(self, nums):
        if not nums:return 0
        if len(nums)==1:return nums[0]
        result=maxP=minP=nums[0]
        for n in nums[1:]:
            minP,maxP = min(n,minP*n,maxP*n),max(n,minP*n,maxP*n)
            result = max(result,maxP)
        return result
```

##162. Find Peak Element （查找）

- 难度：Medium
- 题意：

极值元素指一个元素大于其邻居元素。给定一个输入数组，其中num[i]!=num[i+1],找到极值并返回索引。数组可能包含多个极值，这种情况下，返回任意极值点索引即可。假定num[-1]和num[n]无限。

- 思路：

罪过，我又是线性探索了。

- 代码：

```
class Solution(object):
    def findPeakElement(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        if not nums:return None
        if len(nums)==1:return 0
        if nums[1]<nums[0]:return 0
        if nums[-2]<nums[-1]:return len(nums)-1
        for i in range(1,len(nums)-1):
            if nums[i-1]<nums[i]>nums[i+1]:
                return i
```

- 思路2：

这道题其实很神奇，由于num[-1]是无限小，则有num[-1]<num[0]。我们假定不存在极值，则有对每个i，num[i]<num[i+1]，然而题目设定num[n]无限小，则num[n-2]<num[n-1]>num[n]，出现值。则可知，该数组必定存在极值，且出现在第一个num[i]>num[i+1]的点。因此我们用二分查找即。

- 代码：

```
class Solution(object):
    def findPeakElement(self, nums):
        low,high = 0,len(nums)-1
        while low <= high:
```

```
        if low==high:
            return low
        mid = (low+high)//2
        if num[mid]<num[mid+1]:
            low+=1
        else:
            high=mid
```

##169. Majority Element （众数）

- 难度：Easy
- 题意：

给定一个长度为n的数组，找出其中的众数。众数是指出现超过n/2（向下取整）的数。假定数组不为且必定存在众数。

- 思路：

抵销法，不必要记录每个数字出现的次数，只需要记录目前出现最多次的数字及次数，若出现其他数即次数-1。最后记录的数字即为众数。

- 代码：

```
class Solution:
    # @param {integer[]} nums
    # @return {integer}
    def majorityElement(self, nums):
        count=0
        for n in nums:
            if count==0:
                result,count=n,count+1
            else:
                if n==result:count+=1
                else:count-=1
        return result
```

##229. Majority Element II （众数）

- 难度：Medium
- 题意：

给定一个数组，求所有出现次数超过n/3（向下取整）的数字。要求在线性时间和常数级空间内解决。

- 思路：

出现超过n/3的数字最多只可能出现2个。我们使用a,ca,b,cb分别记录出现最多两个数字以及次数。这的次数不是总次数，而是采用和169一样抵销法后得到的。最后还需要再次确认这两个数字是否出现数超过n/3

- 代码：

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

```
    def majorityElement(self, nums):
        """
        :type nums: List[int]
        :rtype: List[int]
        """
        result=[]
        a=b=ca=cb=0
        for n in nums:
            if n==a:ca+=1
            elif n==b:cb+=1
            elif ca==0:
                a,ca=n,ca+1
            elif cb==0:
                b,cb=n,cb+1
            else:
                ca,cb=ca-1,cb-1
        ca=cb=0
        for n in nums:
            if n==a:ca+=1
            elif n==b:cb+=1
        l = len(nums)//3
        if ca > l:result.append(a)
        if cb> l:result.append(b)
        return result
```

##209. Minimum Size Subarray Sum （子数组）

- 难度：Medium
- 题意：

给定一个由n个正整数组成的数组和正整数s，求和大于s的最短子串长度。如果不存在，返回0.

- 思路：

使用两个下标的滑动窗口。维护窗口内数字之和恰好大于s。若大于s，记录最小长度，左下标右移。下标右移至大于s为止。

- 代码：

```
class Solution:
    # @param {integer} s
    # @param {integer[]} nums
    # @return {integer}
    def minSubArrayLen(self, s, nums):
        if not nums:return 0
        minL= 2**31
        total=i=j=0
        while i<=j and j<len(nums) and total<s:
            total+=nums[j]
            while i<=j and total>=s:
                minL = min(minL,j-i+1)
                total-=nums[i]
                i+=1
```

```
        j+=1
    return 0 if minL==2**31 else minL
```

##217. Contains Duplicate （查找重复）

- 难度：Easy
- 题意：

判断一个数组是否存在重复数字

- 思路：

直接用set存储就可以了。

- 代码：

```
class Solution:
    # @param {integer[]} nums
    # @return {boolean}
    def containsDuplicate(self, nums):
        numset = set()
        la = 0
        for i in range(len(nums)):
            if nums[i] in numset:
                return True
            numset.add(nums[i])
        return False
```

##219. Contains Duplicate II （查找重复）

- 难度：Easy
- 题意：

给定一个数组和正整数k，求是否存在nums[i]==nums[j]，且i和j的间隔最多为k

- 思路：

把（value，index）作为哈希表的键值对。在哈希表中查找nums[i]，取得相等数字的索引j，再进行较。

- 代码：

```
class Solution:
    # @param {integer[]} nums
    # @param {integer} k
    # @return {boolean}
    def containsNearbyDuplicate(self, nums, k):
        numDir = {}
        length = len(nums)
        for i in range(length):
            j = numDir.get(nums[i])
            if j!=None and i-j<=k:
```

```
            return True
        else:
            numDir[nums[i]]=i
    return False
```

##228. Summary Ranges

- 难度：Easy
- 题意：

给定一个数组，返回该数组的概要。如： 输入[0,1,2,4,5,7], 输出 ["0->2","4->5","7"].

- 思路：

题目就是要把相邻数字归为一组。方法比较常规，做线性探索即可。

- 代码：

```
class Solution(object):
    def summaryRanges(self, nums):
        """
        :type nums: List[int]
        :rtype: List[str]
        """
        result=[]
        if not nums:return result
        begin=None
        for i,n in enumerate(nums):
            if begin==None:
                begin=n
            else:
                if n != nums[i-1]+1:
                    result.append(self.toString(begin,nums[i-1]))
                    begin = n
        result.append(self.toString(begin,nums[-1]))
        return result

    def toString(self,begin,end):
        if begin==end:
            return str(begin)
        return str(begin)+'->'+str(end)
```

##238. Product of Array Except Self

- 难度：Easy
- 题意：

给定一个数组num，要求返回一个数组output，其中output[i]=除了num[i]之外所有数字的乘积。

- 思路：

定义up[i]=num[:i]的乘积,down[i]=down[i+1:]的乘积。因此从前和从后分别遍历一次即可求得up和

own。up*down即为题目所求。

- 代码：

```
class Solution(object):
    def productExceptSelf(self, nums):
        """
        :type nums: List[int]
        :rtype: List[int]
        """
        if not nums:return nums
        up=[1]
        down=[1]
        for i in range(1,len(nums)):
            up.append(up[-1]*nums[i-1])
        for i in range(len(nums)-2,-1,-1):
            down.append(down[-1]*nums[i+1])
        down=down[::-1]
        up = [up[i]*down[i] for i in range(len(up))]
        return up
```

##268. Missing Number (查找)

- 难度：Medium
- 题意：

给定长度为n的数组（不包含重复数字），其中数字的范围0到n。找到其中缺失的数字。

- 思路：

0到n总共n-1个数字求和，减去数组之和即为所求。

- 代码：

```
class Solution(object):
    def missingNumber(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        l,s = len(nums),sum(nums)
        ss = l*(l+1)//2
        return ss-s
```

##287. Find the Duplicate Number（查找）

- 难度：Hard
- 题意：

给定一个长度为n+1的数组，其中数字的范围是1到n，则至少有一个重复的数字。求该数组中重复的字。要求不得改变数组顺序，辅助空间只有O（1），时间复杂度必须小于O(n^2).

- 思路：

由于可能存在大于2个的重复，因此直接求和的方法就行不通了。排序不允许，哈希表也不允许。这有一种很巧妙的方法，由于数字的范围是1到n。若不存在重复数字，则大于n/2的数字有n/2个，小于/2的数字有n/2个。基于此，也可以进行类似于二分查找的方法。

- 代码：

```
class Solution(object):
    def findDuplicate(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        low,high=1,len(nums)-1
        while low<=high:
            if low==high:
                return low
            mid=(high+low-1)//2
            count=0
            for n in nums:
                if low<=n<=mid:
                    count+=1
            if count>(high-low+1)//2:
                high=mid
            else:
                low=mid+1
```

##289. Game of Life

- 难度：Medium
- 题意：

给定一个二维数组表示细胞，其中1代表活，0代表死。细胞的状态迁移受到8个邻居状态的影响，具如以下规则：

  - 活细胞邻居少于2个活细胞，则死
  - 活细胞邻居有2或3个活细胞，则活
  - 活细胞邻居有大于3个活细胞，这死
  - 死细胞邻居恰有3个活细胞，则活

要求就地求出该二维数组的下一状态。

- 思路：

由于有求就地，我们必须想办法标记各个细胞的状态。我的办法是记录以该细胞为中心的9个细胞（括自己）的活细胞数c，若该细胞是活细胞则用c代表状态，若该细胞是死细胞则用-c代表状态。然后遍历一遍，根据规则更新细胞。网上也有用00,01,10,11表示细胞现状态和下一状态的，也是很方便。

- 代码：

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

```
def gameOfLife(self, board):
    """
    :type board: List[List[int]]
    :rtype: void Do not return anything, modify board in-place instead.
    """
    n,m=len(board),len(board[0])
    for i in range(n):
        for j in range(m):
            count=0
            for ii in (-1,0,1):
                for jj in (-1,0,1):
                    if 0<=i+ii<n and 0<=j+jj<m:
                        if board[i+ii][j+jj]>0:
                            count+=1
            board[i][j]=count if board[i][j]>0 else 0-count

    for i in range(n):
        for j in range(m):
            if board[i][j] in (1,2):
                board[i][j]=0
            elif board[i][j] in (3,4):
                board[i][j]=1
            elif board[i][j]>4:
                board[i][j]=0
            elif board[i][j]==-3:
                board[i][j]=1
            else:
                board[i][j]=0
```