leetcode解题报告- 2Sum/3Sum/4Sum

作者：Zing
原文链接：https://ld246.com/article/1456310095813
来源网站：链滴

##1.Two Sum

- 难度：Medium
- 题目大意：

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

- 难度:Medium
- 题目大意：

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。先将数组序（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循环遍历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+二分查找，但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

- 难度:Medium
- 题目大意:

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

- 难度:Medium
- 题目大意:

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，肯定是会超时的基本上在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个数可能出现的情况太多。

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
```
