TwoSum类

找前k个极值：可以考虑堆，O(nlogk)

找第开个极值，用partition

partition类

kth Largest Element

用快排的思想

```
class Solution:
    """
    @param n: An integer
    @param nums: An array
    @return: the Kth largest element
    """
    def partition(self, nums, i, left, right):
        pivot = nums[i]
        nums[i] = nums[left]
         l,r=left,right
         while l<r:
             while l<r and nums[r] <= pivot:
                 r -= 1
             nums[l] = nums[r]
             while l<r and nums[l] >= pivot:
                 l += 1
             nums[r] = nums[l]
         nums[l] = pivot
         return l
     def kthLargestElement(self, k, nums):
         # write your code here
         if not nums or k>len(nums):
             return -1

         import random
         left, right = 0,len(nums)-1
         while True:
             i = random.randint(left, right)
             pos = self.partition(nums, i, left, right)
             if pos == k-1:
                 return nums[pos]
             elif pos>k-1:
                 right = pos-1
             else:
                 left = pos+1
```

Nuts & Bolts Problem

分别给一串螺丝钉和螺母要求把它们配套起来。

-暴力搜索时间复杂度为O(n^2),

-同类之间无法比较，所以无法各自排序然后配对

可以采用快排的思想，随机选取一个螺丝钉作为pivot来partition螺母

partition的开始先遍历一遍螺母，找到和pivot相配的螺母作为下个pivot来partition螺丝钉

然后螺母和螺丝钉都有序了，然后就可以一一配对了。

```
"""
class Comparator:
    def cmp(self, a, b)
You can use Compare.cmp(a, b) to compare nuts "a" and bolts "b",
if "a" is bigger than "b", it will return 1, else if they are equal,
it will return 0, else if "a" is smaller than "b", it will return -1.
When "a" is not a nut or "b" is not a bolt, it will return 2, which is not valid.
"""

import random
class Solution:
    # @param nuts: a list of integers
    # @param bolts: a list of integers
    # @param compare: a instance of Comparator
    # @return: nothing
    def partition(self, array, cmp, pivot, left, right):
        l, r = left, right
        for i in range(l,r+1):
            if cmp(array[i], pivot)==0 or cmp(pivot, array[i])==0:
                array[i], array[l] = array[l], array[i]
                record = array[l]
                break
        while l<r:
            while l<r and (cmp(array[r], pivot)==1 or cmp(pivot, array[r])==-1):
                r-=1
            array[l] = array[r]
            while l<r and (cmp(array[l], pivot)==-1 or cmp(pivot, array[l])==1):
                l+=1
            array[r] = array[l]
        array[l] = record
        return l
    def quick_sort(self, nuts, bolts, start, end, cmp):
        if start>=end:
            return
        pos = random.randint(start, end)
        pos = self.partition(bolts, cmp, nuts[pos], start, end)
        self.partition(nuts, cmp, bolts[pos], start, end)
        self.quick_sort(nuts,bolts,start,pos-1,cmp)
        self.quick_sort(nuts,bolts,pos+1,end,cmp)

    def sortNutsAndBolts(self, nuts, bolts, compare):
        # write your code here
        self.quick_sort(nuts,bolts,0,len(nuts)-1,compare.cmp)


```

窗口类

求窗口大小

```python
def minimumSize(self, nums, s):
    # write your code here
    if not nums:
        return -1
    sum = 0
    minlen = float('inf')
    find = False
    r = 0
    for l in range(len(nums)):
        while r < len(nums) and sum < s:
            sum += nums[r]
            r += 1
        if sum >= s:
            find = True
            if r - l < minlen:
                minlen = r - l
        sum -= nums[l]
    return minlen if find else -1
```

窗口类指针移动模版

```
for i in range(n):
    while j<n:
        if 条件满足:
            j+=1
            更新j状态
        else:
            break
     更新i状态

```

和slidingwindow的区别是sliding window是固定大小的
我的算法居然没有考虑到没有重复元素的情况

而且很重要的是一定要维护一个low来指定截止目前最后一次重复了的元素位置。

```python
def lengthOfLongestSubstring(self, s):
    # write your code here
    if not s:
        return 0
    sweep = dict()
    test = []
    maxlen=0
    low = -1
    for i,elem in enumerate(s):
        if elem in sweep:
            if sweep[elem]>low:
                low = sweep[elem]
        sweep[elem] = i
        maxlen = max(maxlen, i-low)
    return maxlen
```

别人的代码思想是，把重复了的元素之前的所有元素都删掉
比如abcc,最后一个和倒数第二个重复了，那就把abc统统删除

```python
def lengthOfLongestSubstring(self, s):
    unique_chars = set([])
    j = 0
    n = len(s)
    longest = 0
    for i in range(n):
        while j < n and s[j] not in unique_chars:
            unique_chars.add(s[j])
            j += 1
        longest = max(longest, j - i)
        unique_chars.remove(s[i])

    return longest
```

前向型指针 Hash或者set记录上次访问

Minimum Window Substring

```
def minWindow(self, source , target):
    # write your code here
    if not source or not target:
        return ""

    targetHash={}
    for t in target:
        targetHash[t] = targetHash.get(t, 0)+1
    newhash={}
    distinct = len(targetHash)
    already = 0
    r=0
    minlen=len(source)+1
    minstring = ""

    for l in range(len(source)):
        while r<len(source) and already<distinct:
            if source[r] in targetHash:
                newhash[source[r]] = newhash.get(source[r], 0)+1
                if newhash[source[r]]==targetHash[source[r]]:
                    already += 1
            r+=1
        if already==distinct and minlen>r-l:
            minlen=r-l
            minstring = source[l:r]
        if source[l] in targetHash:
            if targetHash[source[l]] == newhash[source[l]]:
                already -= 1
            newhash[source[l]]-= 1
    return minstring
```

先把待查找的集合放到一个hash里面

用窗口类的模版去做题。

r的滑动当遇到在目标集合中的元素的时候就加入到自己的新的hash里面，多了也没关系，只要在就行。

等跳出了循环，判断一下是不是加够了，加够了并且长度还比之前的还要小，那就更新一下。

然后更新l，如果它在目标集合里就删掉，如果它是关键元素，那么already要减一

注意这里我们必须只维护不同元素的个数

这类问题先写出暴力的双重循环，然后看后一个指针是否需要回退