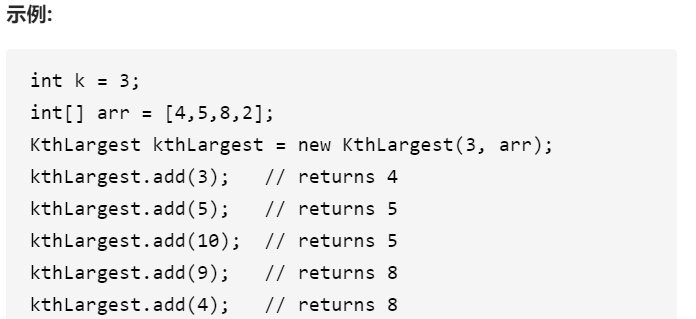
**第K大的元素**

设计一个找到数据流中第K大元素的类（class）。注意是排序后的第K大元素，不是第K个不同的元素。

你的 KthLargest 类需要一个同时接收整数 k 和整数数组nums 的构造器，它包含数据流中的初始元素。每次调用 KthLargest.add，返回当前数据流中第K大的元素。



解题思路：

1、heapq.heapify可以原地把一个list调整成堆[小顶堆] 而 heapq.nlargest 会调成大顶堆

2、heapq.heappop可以弹出堆顶，并重新调整

3、heapq.heappush可以新增元素到堆中，不会调整

4、heapq.heapreplace可以替换堆顶元素，并调整下

5、为了维持为K的大小，初始化的时候可能需要删减，后面需要做处理就是如果不满K个就新增，否则做替换；

6、heapq其实是对一个list做原地的处理，第一个元素就是最小的，直接返回就是最小的值

PY代码实现：

from heapq import \*

class KthLargest(object):

def \_\_init\_\_(self, k, nums):

"""

:type k: int

:type nums: List[int]

"""

self.k = k

self.nums = nums

heapify(self.nums)

while len(self.nums)>self.k: #cut heap to size:k

heappop(self.nums)

def add(self, val):

"""

:type val: int

:rtype: int

"""

if len(self.nums) < self.k:

heappush(self.nums, val)

heapify(self.nums) #cation

else:

top = float('-inf')

if len(self.nums) > 0:

top = self.nums[0]

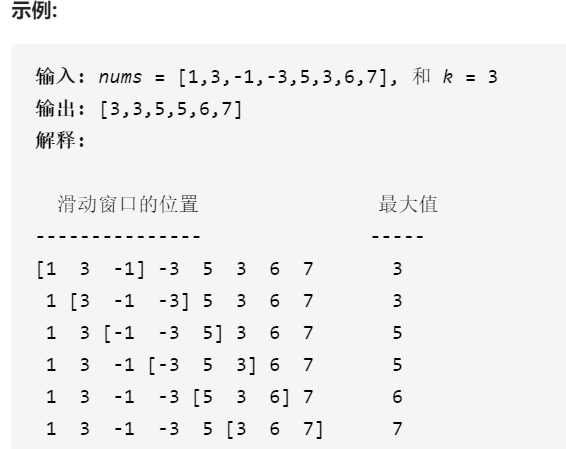
if top < val:

heapreplace(self.nums, val)

return self.nums[0]

**滑动窗口的最大值**

给定一个数组 nums 和滑动窗口的大小 k，请找出所有滑动窗口里的最大值。



PY代码实现：

class Solution {

public:

vector<int> maxSlidingWindow(vector<int>& nums, int k) {

vector<int> ans;

deque<int> deq;

for (int i = 0; i != nums.size(); ++i)

{

while (!deq.empty() && nums[deq.back()] <= nums[i])//维护递减序列

deq.pop\_back();

while (!deq.empty() && deq.front() < i - k + 1)//维护范围

deq.pop\_front();

deq.push\_back(i);

if (i >= k - 1) ans.push\_back(nums[deq.front()]);

}

return ans;

}

};

**最近的请求次数**

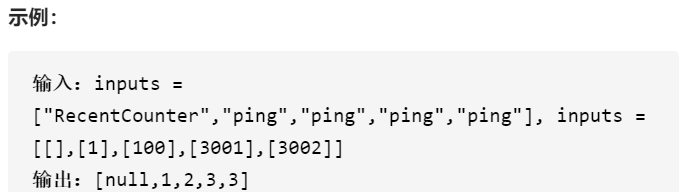
写一个 RecentCounter 类来计算最近的请求。

它只有一个方法：ping(int t)，其中 t 代表以毫秒为单位的某个时间。

返回从 3000 毫秒前到现在的 ping 数。

任何处于 [t - 3000, t] 时间范围之内的 ping 都将会被计算在内，包括当前（指 t 时刻）的 ping。

保证每次对 ping 的调用都使用比之前更大的 t 值。



题解：

我们只会考虑最近 3000 毫秒到现在的 ping 数，因此我们可以使用队列存储这些 ping 的记录。当收到一个时间 t 的 ping 时，我们将它加入队列，并且将所有在时间 t - 3000 之前的 ping 移出队列。

JAVA代码实现：

class RecentCounter {

Queue<Integer> q;

public RecentCounter() {

q = new LinkedList();

}

public int ping(int t) {

q.add(t);

while (q.peek() < t - 3000)

q.poll();

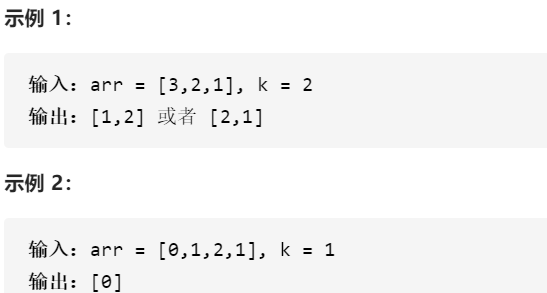
return q.size();

}

}

**最小的第K个数**

输入整数数组 arr ，找出其中最小的 k 个数。例如，输入4、5、1、6、2、7、3、8这8个数字，则最小的4个数字是1、2、3、4。



题解：堆

我们用一个大根堆实时维护数组的前 kk 小值。首先将前 kk 个数插入大根堆中，随后从第 k+1k+1 个数开始遍历，如果当前遍历到的数比大根堆的堆顶的数要小，就把堆顶的数弹出，再插入当前遍历到的数。最后将大根堆里的数存入数组返回即可。在下面的代码中，由于 C++ 语言中的堆（即优先队列）为大根堆，我们可以这么做。而 Python 语言中的对为小根堆，因此我们要对数组中所有的数取其相反数，才能使用小根堆维护前 kk 小值。

C++代码实现：

class Solution {

public:

vector<int> getLeastNumbers(vector<int>& arr, int k) {

vector<int>vec(k, 0);

if (k == 0) return vec; // 排除 0 的情况

priority\_queue<int>Q;

for (int i = 0; i < k; ++i) Q.push(arr[i]);

for (int i = k; i < (int)arr.size(); ++i) {

if (Q.top() > arr[i]) {

Q.pop();

Q.push(arr[i]);

}

}

for (int i = 0; i < k; ++i) {

vec[i] = Q.top();

Q.pop();

}

return vec;

}

};

**最后还一块石头的重量**

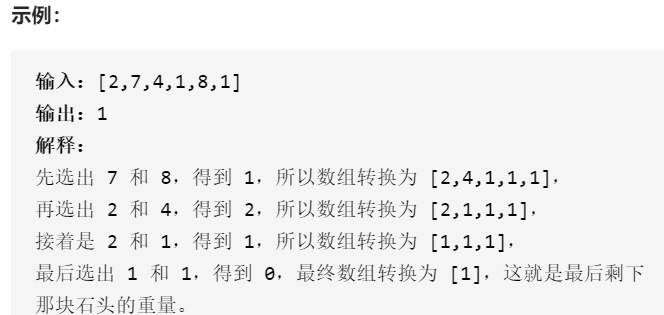
有一堆石头，每块石头的重量都是正整数。

每一回合，从中选出两块 最重的 石头，然后将它们一起粉碎。假设石头的重量分别为 x 和 y，且 x <= y。那么粉碎的可能结果如下：

如果 x == y，那么两块石头都会被完全粉碎；

如果 x != y，那么重量为 x 的石头将会完全粉碎，而重量为 y 的石头新重量为 y-x。

最后，最多只会剩下一块石头。返回此石头的重量。如果没有石头剩下，就返回 0。



题解：

初始化最大堆

循环（直至Math.max(stones[1], stones[2])==0）

按要求修改堆顶两个元素

重新调整最大堆

JAVA代码实现：

public int lastStoneWeight(int[] stones) {

int len = stones.length;

if (len==1) {

return stones[0];

}else if (len==2) {

return Math.abs(stones[0]-stones[1]);

}else {

for (int i = len/2-1; i >= 0 ; i--) {

maxHeap(stones, i, len-1);

}

int temp;

while ((temp=Math.max(stones[1], stones[2]))!=0) {

stones[0]-=temp;

if (stones[1]>stones[2]) {

stones[1]=0;

}else {

stones[2]=0;

}

for (int i = 2; i >= 0 ; i--) {

maxHeap(stones, i, len-1);

}

}

return stones[0];

}

}

public void maxHeap(int[] nums, int root, int end) {

int l=2\*root+1;

int temp=nums[root];

while (l<=end && (nums[l]>temp || (l+1<=end && nums[l+1]>temp))) {

if (l+1<=end && nums[l+1]>nums[l]) {

l++;

}

nums[root]=nums[l];

nums[l]=temp;

root=l; l=2\*root+1;

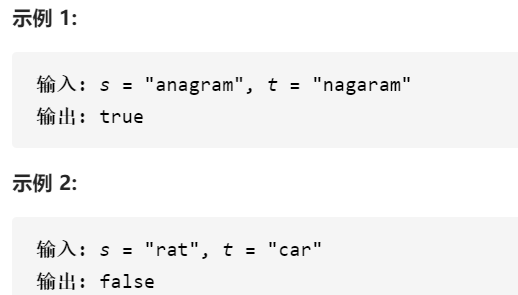
temp=nums[root];

}

}

**有效的字母异味词**

给定两个字符串 s 和 t ，编写一个函数来判断 t 是否是 s 的字母异位词。



题解：

通过将 ss 的字母重新排列成 tt 来生成变位词。因此，如果 TT 是 SS 的变位词，对两个字符串进行排序将产生两个相同的字符串。此外，如果 ss 和 tt 的长度不同，tt 不能是 ss 的变位词，我们可以提前返回。

JAVA代码实现：

public boolean isAnagram(String s, String t) {

if (s.length() != t.length()) {

return false;

}

char[] str1 = s.toCharArray();

char[] str2 = t.toCharArray();

Arrays.sort(str1);

Arrays.sort(str2);

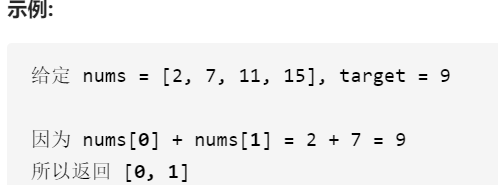
return Arrays.equals(str1, str2);

}

**两数之和**

给定一个整数数组 nums 和一个目标值 target，请你在该数组中找出和为目标值的那 两个 整数，并返回他们的数组下标。

你可以假设每种输入只会对应一个答案。但是，数组中同一个元素不能使用两遍。



题解：

找到 num2 = target - num1，是否也在 list 中，那么就需要运用以下两个方法：

num2 in nums，返回 True 说明有戏

nums.index(num2)，查找 num2 的索引

Py代码实现：

def twoSum(nums, target):

lens = len(nums)

j=-1

for i in range(lens):

if (target - nums[i]) in nums:

if (nums.count(target - nums[i]) == 1)&(target - nums[i] == nums[i]):

continue

else:

j = nums.index(target - nums[i],i+1)

break

if j>0:

return [i,j]

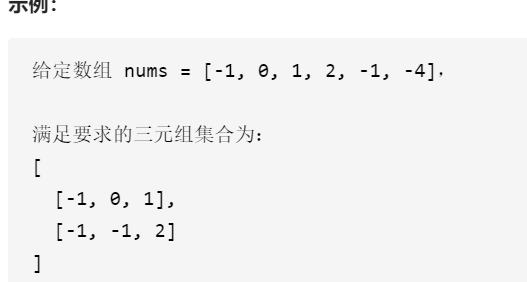
else:

return []

**三数之和**

给你一个包含 n 个整数的数组 nums，判断 nums 中是否存在三个元素 a，b，c ，使得 a + b + c = 0 ？请你找出所有满足条件且不重复的三元组。

注意：答案中不可以包含重复的三元组。



题解：

双指针法思路： 固定 33 个指针中最左（最小）数字的指针 k，双指针 i，j 分设在数组索引 (k, len(nums))(k,len(nums)) 两端，通过双指针交替向中间移动，记录对于每个固定指针 k 的所有满足 nums[k] + nums[i] + nums[j] == 0 的 i,j 组合：

1.当 nums[k] > 0 时直接break跳出：因为 nums[j] >= nums[i] >= nums[k] > 0，即 33 个数字都大于 00 ，在此固定指针 k 之后不可能再找到结果了。

2.当 k > 0且nums[k] == nums[k - 1]时即跳过此元素nums[k]：因为已经将 nums[k - 1] 的所有组合加入到结果中，本次双指针搜索只会得到重复组合。

3.i，j 分设在数组索引 (k, len(nums))(k,len(nums)) 两端，当i < j时循环计算s = nums[k] + nums[i] + nums[j]，并按照以下规则执行双指针移动：

当s < 0时，i += 1并跳过所有重复的nums[i]；

当s > 0时，j -= 1并跳过所有重复的nums[j]；

当s == 0时，记录组合[k, i, j]至res，执行i += 1和j -= 1并跳过所有重复的nums[i]和nums[j]，防止记录到重复组合。

Py代码实现:

class Solution:

def threeSum(self, nums: [int]) -> [[int]]:

nums.sort()

res, k = [], 0

for k in range(len(nums) - 2):

if nums[k] > 0: break # 1. because of j > i > k.

if k > 0 and nums[k] == nums[k - 1]: continue # 2. skip the same `nums[k]`.

i, j = k + 1, len(nums) - 1

while i < j: # 3. double pointer

s = nums[k] + nums[i] + nums[j]

if s < 0:

i += 1

while i < j and nums[i] == nums[i - 1]: i += 1

elif s > 0:

j -= 1

while i < j and nums[j] == nums[j + 1]: j -= 1

else:

res.append([nums[k], nums[i], nums[j]])

i += 1

j -= 1

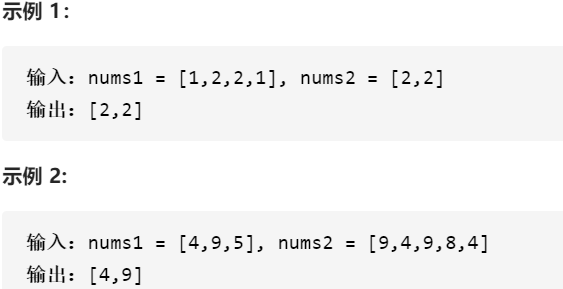
while i < j and nums[i] == nums[i - 1]: i += 1

while i < j and nums[j] == nums[j + 1]: j -= 1

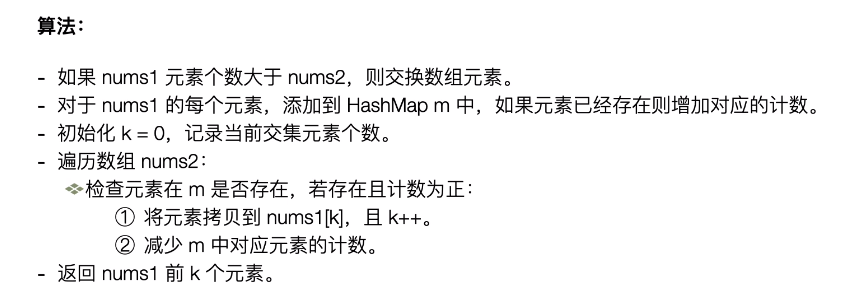
return res

**两个数组的交集2**

给定两个数组，编写一个函数来计算它们的交集



题解：



Py代码实现：

class Solution {

public int[] intersect(int[] nums1, int[] nums2) {

if (nums1.length > nums2.length) {

return intersect(nums2, nums1);

}

Map<Integer, Integer> map = new HashMap<Integer, Integer>();

for (int num : nums1) {

int count = map.getOrDefault(num, 0) + 1;

map.put(num, count);

}

int[] intersection = new int[nums1.length];

int index = 0;

for (int num : nums2) {

int count = map.getOrDefault(num, 0);

if (count > 0) {

intersection[index++] = num;

count--;

if (count > 0) {

map.put(num, count);

} else {

map.remove(num);

}

}

}

return Arrays.copyOfRange(intersection, 0, index);

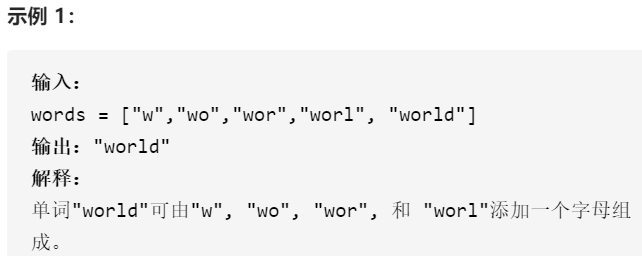
}

}

**词典中最长的单词**

给出一个字符串数组words组成的一本英语词典。从中找出最长的一个单词，该单词是由words词典中其他单词逐步添加一个字母组成。若其中有多个可行的答案，则返回答案中字典序最小的单词。

若无答案，则返回空字符串。





题解：

当我们找到一个单词它的长度更长且它的全部前缀都存在，我们将更改答案。

或者，我们可以事先将单词排序，这样当我们找到一个符合条件的单词就可以认定它 是答案。

Py代码实现：

class Solution(object):

def longestWord(self, words):

ans = ""

wordset = set(words)

for word in words:

if len(word) > len(ans) or len(word) == len(ans) and word < ans:

if all(word[:k] in wordset for k in xrange(1, len(word))):

ans = word

return ans