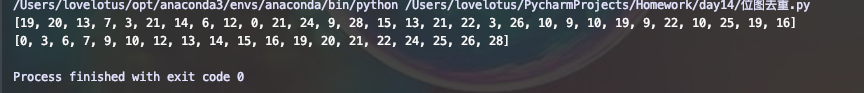
1. 上课已写
2. 上课已写
3. 通过自己写的hash函数，实现昨天的圣经前10的词频统计

*from* operator *import* itemgetter  
*class* Hash\_str(object):  
 *def \_\_init\_\_*(self, *len*):  
 self.hash\_len = *len* self.hash\_map = [*None*] \* *len  
  
 def* hash\_elf\_algorithm(self,*str*):  
 h = 0  
 g = 0  
 *for* i *in str*:  
 h = (h << 4)+ ord(i) #ASCII  
 g = h & 0xf00000000  
 *if* g:  
 h ^= g >>24  
 h &= ~g  
 *return* h % self.hash\_len #取余使得其落在len范围  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
  
 hash1 = Hash\_str(10000)  
 f = open('The\_Holy\_Bible.txt')  
 text = f.read()  
 *for* i *in* text.replace('\n', ' ').split(' '):  
 *if* i != '':  
 hash1.hash\_map[hash1.hash\_elf\_algorithm(i)] = [i, 0]  
 *for* i *in* text.replace('\n', ' ').split(' '):  
 *if* i != '' *and* i *in* hash1.hash\_map[hash1.hash\_elf\_algorithm(i)]:  
 hash1.hash\_map[hash1.hash\_elf\_algorithm(i)][1] += 1  
 f.close()  
 list1 = []  
 *for* i *in* hash1.hash\_map:  
 *if* i != *None*:  
 list1.append(i)  
 list2 = sorted(list1, key=itemgetter(1), reverse=*True*)  
 *for* i *in* range(10):  
 print(list2[i], end=' ')



1. 自行编写位图，实现1千个随机数（0-1万）之间的去重

# 不会用位图实现，还是参照自己的方法了  
# 等自己再学一下位图吧  
*import* random  
*def* count\_sort(*list*):  
 count\_value\_list = [0] \* (max(*list*)+1)  
 # 如果有某个元素，则计1，这样只计一次实现去重  
 *for* i *in list*:  
 count\_value\_list[i] = 1  
 list1 = []  
 *for* i *in* range(max(*list*)+1):  
 *for* j *in* range(count\_value\_list[i]):  
 list1.append(i)  
 *return* list1  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 a = []  
 *for* i *in* range(30):  
 a.append(random.randint(0,30))  
 print(a)  
 print(count\_sort



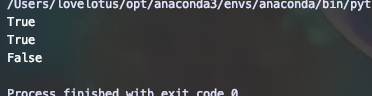
难度作业：

1. 完成两个有序数组找出第K大的数

# 第一个函数如果列表中有重复元素就有点问题  
# 第一个函数实现在两个升序列表中查找第k小的元素  
  
*import* random  
  
  
*def* Binary\_find\_Kth(*list1*, *len1*, *list2*, *len2*, *k*):  
 # 始终认为len1 <=len2  
 *if len1* > *len2*:  
 *return* Binary\_find\_Kth(*list2*, *len2*, *list1*, *len1*, *k*)  
 *if len1* == 0:  
 *return list2*[*k* - 1]  
 *if k* == 1:  
 *return* min(*list1*[0], *list2*[0])  
  
 k1 = min(*k* // 2, *len1*)  
 k2 = *k* - k1  
 *if list1*[k1 - 1] > *list2*[k2 - 1]:  
 *for* i *in* range(k2):  
 *list2*.pop(0)  
 *return* Binary\_find\_Kth(*list1*, *len1*, *list2*, *len2* - k2, *k* - k2)  
 *elif list1*[k1 - 1] < *list2*[k2 - 1]:  
 *for* i *in* range(k1):  
 *list1*.pop(0)  
 *return* Binary\_find\_Kth(*list1*, *len1* - k1, *list2*, *len2*, *k* - k1)  
 *else*:  
 *return list1*[k1 - 1]  
  
  
# 直接归并两个列表了,如果不是分机查找就不用上面的方法  
*def* search(*list1*, *list2*, *k*):  
 list\_result = []  
 i = 0  
 j = 0  
 *while* i < len(*list1*) *and* j < len(*list2*):  
 *if list1*[i] <= *list2*[j]:  
 list\_result.append(*list1*[i])  
 i += 1  
 *else*:  
 list\_result.append(*list2*[j])  
 j += 1  
 *if* i < len(*list1*):  
 *for* h *in* range(i, len(*list1*)):  
 list\_result.append(*list1*[h])  
 *if* j < len(*list2*):  
 *for* h *in* range(j, len(*list2*)):  
 list\_result.append(*list2*[h])  
  
 *return* list\_result[len(list\_result) - *k*]  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 list1, list2 = [], []  
 *for* i *in* range(10):  
 list1.append(random.randint(0, 100))  
 *for* j *in* range(10):  
 list2.append(random.randint(0, 100))  
 print(sorted(list1))  
 print(sorted(list2))  
 print(  
 Binary\_find\_Kth(  
 sorted(list1),  
 len(list1),  
 sorted(list2),  
 len(list2),  
 6))  
 print(search(sorted(list1), sorted(list2), 1))

1. 实现字典树的新增，查找，删除

*class* TriNode(object):  
 *def \_\_init\_\_*(self, *value*=*None*, *count*=0, *parent*=*None*):  
 self.val = *value* # 词频统计  
 self.count = *count* self.p = *parent* # 子节点，val:TriNode  
 self.child = {}  
  
  
*class* Trie(object):  
 *def \_\_init\_\_*(self):  
 self.root = TriNode()  
  
 *def* insert(self, *str*):  
 cur = self.root  
 *for* i *in str*:  
 *if* i *not in* cur.child:  
 child = TriNode(value=i, count=1, parent=cur)  
 cur.child[i] = child  
 cur = child  
 *else*:  
 cur = cur.child[i]  
 cur.count += 1  
  
 *def* search(self, *str*):  
 cur = self.root  
 flag = *True  
 for* i *in str*:  
 *if* i *not in* cur.child:  
 flag = *False  
 break  
 else*:  
 cur = cur.child[i]  
 # 希望查找的是完整的序列  
 *if* cur.child:  
 flag = *False  
 return* flag  
  
 *def* delete(self, *str*):  
 flag = *False  
 if* self.search(*str*):  
 flag = *True* cur = self.root  
 *for* i *in str*:  
 cur.child[i].count -= 1  
 *if* cur.child[i].count == 0:  
 cur.child.pop(i)  
 *break  
 else*:  
 cur = cur.child[i]  
  
 *return* flag  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 trie = Trie()  
 str = 'abcdefghijk'  
 *for* i *in* str:  
 trie.insert(i)  
 print(trie.search('a'))  
 print(trie.search('f'))  
 trie.delete('h')  
 print(trie.search('h'))



day12的答案上课都写了，day14-day17的作业答案上课都写了