基础知识和算法题，题目都比较简答，基础知识linux命令、计算机网络、数据库会问一些，算法题一般就是字符串转整数这种（atoi）

string.atoi(str)

def myAtoi(self, str):

return max(min(int(\*re.findall('^[\+\-]?\d+', str.lstrip())), 2\*\*31 - 1), -2\*\*31)

**一、参考：**

基本数据结构：<https://blog.csdn.net/qq_41560183/article/details/82390337>

栈：<https://blog.csdn.net/wjl31802/article/details/81985275?utm_source=blogxgwz0>

基本数据结构操作：<https://www.cnblogs.com/shengzhongqiu/p/7222003.html>

**二、简单排序**

1.现有字典d={‘a’:24，’g’:52，’l’:12，’k’:33}请按字典中的value值进行排序？

sorted(d.items()，key=lambda x:x[1])

2.请按alist中元素的age由大到小排序:

sorted(alist，key=lambda x:x['age']，reverse=True)

3、重新排序数组中的正值和负值

lists.sort(key=lambda x: (x < 0, abs(x)))

1. **数组**

**基本操作：insert、get、delete、size**

1、 通过dicts的value取key：

list(a.keys())[list(a.values()).index('a')]

2、 请写出一段 Python 代码实现删除一个 list 里面的重复元素?

l2 = list(set(l1))

l2 = sorted(set(l1)，key=l1.index) 按原来排序

A、B 中相同元素：print(set(A)&set(B))

A、B 中不同元素：print(set(A)^set(B))

3、寻找数组中第二小的元素：初始化2个最小值：firstmin,secondmin。遍历所有元素,假如当前元素小于firstmin, 将更新firstmin,secondmin.如果小于secondmin直接更新secondmin。import numpy as np

arr = [2, 3, 4, 1, 7, 6, 5]

print("# arr中最大的数为{}，位于第{}位".format(np.max(arr), np.argmax(arr)+1))

4、找到数组中第一个不重复出现的整数

def findNoDupMany(data):

single = 0

Map = {}

for i in range(len(data)):

if data[i] in Map:

Map[data[i]] += 1

else:

Map[data[i]] = 1

for key in Map:

if Map[key] == 1:

single = key

break

return single

5、合并两个有序数据

def merge(self, nums\_1, m, nums\_2, n):

nums\_1[m:m+n] = nums\_2

return nums\_1.sort()

1. 整形数组中除了两个数外,其它的都出现了两次。

def FindNumsAppearOnce(self, array):

temp = []

for a in array:

if a in temp:

temp.remove(a)

else:

temp.append(a)

return temp

7、全排列组合

def str\_sort(s=''):

   if len(s) <= 1:

     return [s]

   str\_list = []

   for i in range(len(s)):

     for j in str\_sort(s[0:i] + s[i + 1:]):

       str\_list.append(s[i] + j)

   return str\_list

**四、栈**

基本操作：push、pop、isEmpty、top 先进后出， x.append() x.pop() 取出

1. **队列**

基本操作：Enqueue()  在尾部插入元素、Dequeue()  移除头部的元素、isEmpty() 如果为空返回true、 Top()  返回的第一个元素,先进先出: x.pop(0)

队列和堆栈：可以用数组来实现（下标取出），也可以用链表实现（通过head、next取出）

1. **链表：**

链表就像一个节点链，其中每个节点包含着数据和指向后续节点的指针。 链表还包含一个头指针，它指向链表的第一个元素，但当列表为空时，它指向null或无具体内容。一般用于实现文件系统、哈希表和邻接表。

1、查找单链接倒数第k个数：

查找单链接倒数第k个数：

class listNode:

def \_\_init\_\_(self,val,next):

self.val = val

self.next = next

class Solution:

def FindKthToTail(self, head, k):

L = []

while head:

L.append(head)

head=head.next

if k>len(L) or k<1:

return None

return L[-k]

1. **数：**

树形结构是一种层级式的数据结构，由顶点（节点）和连接它们的边组成。 树类似于图，但区分树和图的重要特征是树中不存在环路。

1、二叉排序树又称为二叉查找树。它或者是一颗空树，或者是具有下列性质的二叉树：

若它的左子树不为空，则左子树上所有节点的值均小于它的根结构的值；

若它的右子树不为空，则右子树上所有节点的值均大于它的根结构的值；

它的左、右子树也分别为二叉排序树。

class BSTNode:

"""

定义一个二叉树节点类。以讨论算法为主，忽略了一些诸如对数据类型进行判断的问题。

"""

def \_\_init\_\_(self, data, left=None, right=None):

"""

初始化

:param data: 节点储存的数据

:param left: 节点左子树

:param right: 节点右子树

"""

self.data = data

self.left = left

self.right = right

class BinarySortTree:

"""

基于BSTNode类的二叉排序树。维护一个根节点的指针。

"""

def \_\_init\_\_(self):

self.\_root = None

def is\_empty(self):

return self.\_root is None

def search(self, key):

"""

关键码检索

:param key: 关键码

:return: 查询节点或None

"""

bt = self.\_root

while bt:

entry = bt.data

if key < entry:

bt = bt.left

elif key > entry:

bt = bt.right

else:

return entry

return None

def \_\_iter\_\_(self):

"""

实现二叉树的中序遍历算法,

展示我们创建的二叉排序树.

直接使用python内置的列表作为一个栈。

:return: data

"""

stack = []

node = self.\_root

while node or stack:

while node:

stack.append(node)

node = node.left

node = stack.pop()

yield node.data

node = node.right

def afterTraverse(root):

'''

后序遍历

'''

if root==None:

return

afterTraverse(root.left)

afterTraverse(root.right)

if \_\_name\_\_ == '\_\_main\_\_':

lis = [62, 58, 88, 48, 73, 99, 35, 51, 93, 29, 37, 49, 56, 36, 50]

bs\_tree = BinarySortTree()

for i in range(len(lis)):

bs\_tree.insert(lis[i])

for i in bs\_tree:

print(i, end=" ")

print("\n", bs\_tree.search(4))

**八、最长回文字符串（**是否回文：st==st[::-1] 表示从后往前逐个取字符串**）：**

    def longestPalindrome(self, s):

        if len(s) <= 1:

            return s

        for j in range(len(s),0,-1):

            for i in range(0,len(s)-length+1):

                now\_s=s[i:i+j]

                if now\_s==now\_s[::-1]:

                    return now\_s

**九、字符串的所有组合问题，输入三个字符a,b,c，则它们的组合有a,b,c,ab,ac,bc,abc。**

def quanpailie(s):

if len(s) <= 1:

return [s]

total = []

lens = len(s)

for i in range(lens):

total.append(s[i])

for j in range(i + 1, lens):

##因为j只能到lens -1，所以需要j+1来取到最后一个数

total.append(s[i:j+1])

return sorted(total, key=lambda i: len(i), reverse=False)

**十、海量日志数据，提取出某日访问次数最多的那个IP：**

思路如下：

   1.对于海量数据中的每一个ip，使用hash函数计算hash(ip)%1000,输出到1000个文件中

   2.对于这1000个文件，分别找出出现最多的ip。

   3.使用外部排序，对找出来的1000个ip在进行排序。

from collections import Counter

if \_\_name\_\_ == '\_\_main\_\_':

ip\_list = ["192.168.1.2","192.168.1.3", "192.168.1.3", "192.168.1.4", "192.168.1.2","192.168.1.2"]

ip\_counter = Counter(ip\_list) #使用python内置的列表元素计数函数，进行统计

print(ip\_counter.most\_common()) #[('192.168.1.2', 3), ('192.168.1.3', 2), ('192.168.1.4', 1)]

print(ip\_counter.most\_common()[0][0])

**十一、动态规划：**



**十二、"驼峰"和"下划线"字符串之间的相互转换**



**十三、查找第一个只出现一次的字符。**

def methond3(str):

for s in str:

if str.count(s) == 1:

return s,str.index(s)

**十四、现在要处理一个大小为 10G 的文件，但是内存只有 4G.**

def get\_lines():

L = []

with open('file.txt', 'rb') as f:

data = f.readlines(60000)

L.append(data)

yield L

要考虑到的问题有：内存只有 4G 无法一次性读入 10G 的文件，需要分批读入。分批读入数据要记录每次读入数据的位置。分批每次读入数据的大小，太小就会在读取操作上花费过多时间。

read、readline 和 readlines 的区别?

read: 读取整个文件。

readline：读取下一行，使用生成器方法。

readlines：读取整个文件到一个迭代器以供我们遍历。

经过测试发现参数为"rb"时的效率是"r"的6倍。

r： 以只读方式打开文件。文件的指针将会放在文件的开头,这是默认模式。

rb：以二进制格式打开一个文件用于只读。文件指针将会放在文件的开头。   
w：表示要向文件写入数据，并截断以前的内容  
a：表示要向文件写入数据，添加到当前内容尾部  
r+：表示对文件进行可读写操作（删除以前的所有数据）  
r+a：表示对文件可进行读写操作（添加到当前文件尾部）

1. 快排

/ 表示浮点数除法，返回浮点float结果;

// 表示整数除法,返回一个不大于" / "计算结果的最大整数int，特别注意如果其中一个操作数位负数，则结果必为负数。

def quickSort(listx):

if len(listx)<=1:

return listx

pivot = listx[len(listx)//2] #取列表中中间的元素为被比较数

listl = [x for x in listx if x < pivot] #<pivot的放在一个列表

listm = [x for x in listx if x ==pivot] #=pivot的放在一个列表

listr = [x for x in listx if x > pivot] #>pivot的放在一个列表

left = quickSort(listl) #递归进行该函数

right = quickSort(listr) #递归进行该函数

return left + listm + right #整合

print(quickSort([9,3, 6, 8, 9, 19, 1, 5])) #[1, 3, 5, 6, 8, 9, 9, 19]