**哈希表的总结**

（1）定义：哈希表（Hash table，也叫散列表），是根据关键码值(Key value)而直接进行访问的数据结构。也就是说，它通过把关键码值映射到表中一个位置来访问记录，以加快查找的速度。这个映射函数叫做散列函数，存放记录的数组叫做散列表。

（2）举例：查字典，如果我想要获取“按”字详细信息，我肯定会去根据拼音an去查找 拼音索引（当然也可以是偏旁索引），我们首先去查an在字典的位置，查了一下得到“安”，结果如下。这过程就是键码映射，在公式里面，就是通过key去查找f(key)。其中，按就是关键字（key），f（）就是字典索引，也就是哈希函数，查到的页码4就是哈希值。



（3）出现背景：

数组的特点是：寻址容易，插入和删除困难；

而链表的特点是：寻址困难，插入和删除容易。

那么我们能不能综合两者的特性，做出一种寻址容易，插入删除也容易的数据结构？

（4）存储与查找：

当存储记录时，通过散列函数计算出记录的散列地址

当查找记录时，我们通过同样的是散列函数计算记录的散列地址，并按此散列地址访问记录

（5）优缺点：

优点：高效（一对一的查找）

不论哈希表中有多少数据，查找、插入、删除（有时包括删除）只需要接近常量的时间即O(1)的复杂度。实际上，这只需要几条机器指令。

哈希表运算得非常快，在计算机程序中，如果需要在一秒种内查找上千条记录通常使用哈希表（例如拼写检查器)哈希表的速度明显比树快，树的操作通常需要O(N)的时间级。哈希表不仅速度快，编程实现也相对容易。

如果不需要有序遍历数据，并且可以提前预测数据量的大小。那么哈希表在速度和易用性方面是无与伦比的。

缺点：散列冲突

一个关键字可能对应多个散列地址；需要查找一个范围时，效果不好。

它是基于数组的，数组创建后难于扩展，某些哈希表被基本填满时，性能下降得非常严重，所以程序员必须要清楚表中将要存储多少数据（或者准备好定期地把数据转移到更大的哈希表中，这是个费时的过程）。

（6）常用散列函数

好的散列函数=计算简单+分布均匀（计算得到的散列地址分布均匀）

1. 除法散列法（取余）

如：index = value % 16

1. 平方散列法

求index是非常频繁的操作，而乘法的运算要比除法来得省时（对现在的CPU来说，估计我们感觉不出来），所以我们考虑把除法换成乘法和一个位移操作。公式：

index = (value \* value) >> 28 （右移，除以2^28。记法：左移变大，是乘。右移变小，是除。）

1. 斐波那契散列法

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233,377, 610， 987, 1597, 2584, 4181, 6765, 10946，…

1. time33算法

unsigned long hash(const char\* key){

unsigned long hash=0;

for(int i=0;i<strlen(key);i++){

hash = hash\*33+str[i];

}

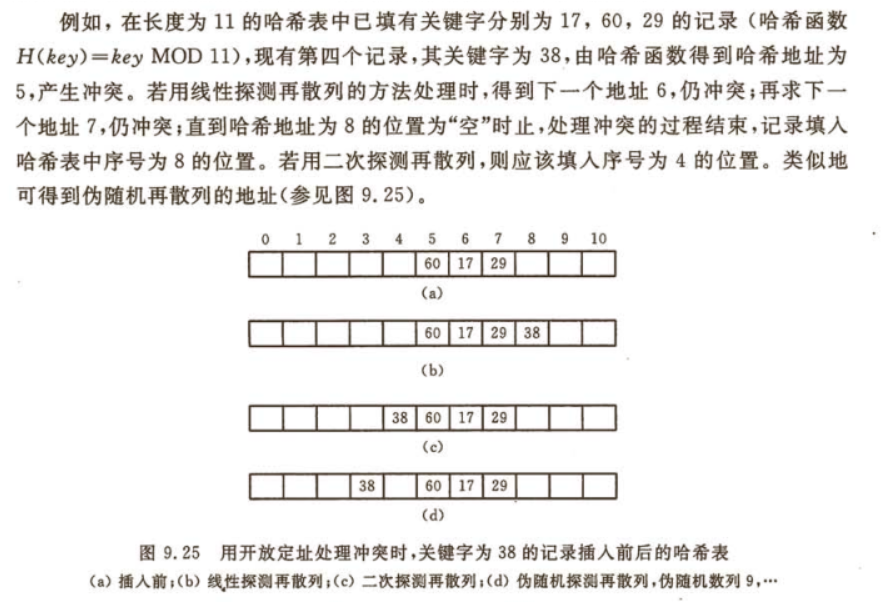
return hash;

}

（7）哈希冲突

不同关键字映射到同一个哈希值

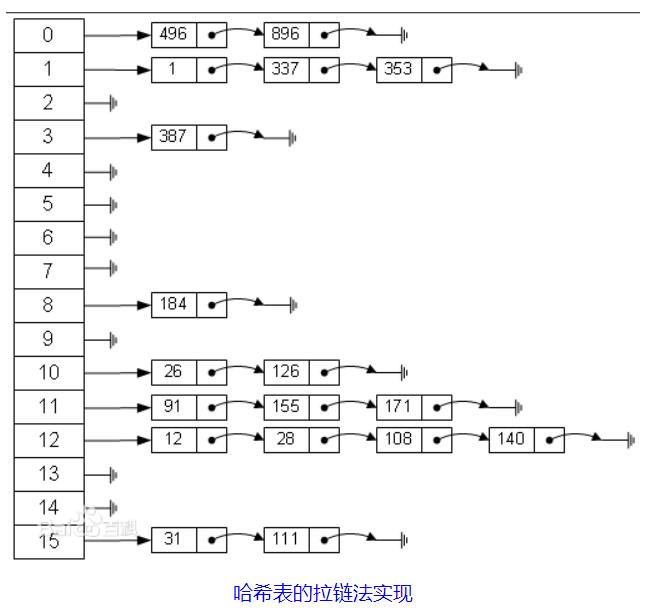
1. 开发地址法：找剩余的地址：



1. 链地址法

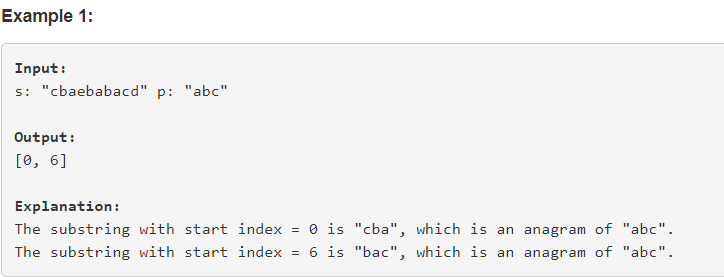
上面所说的开发定址法的原理是遇到冲突的时候查找顺着原来哈希地址查找下一个空闲地址然后插入，但是也有一个问题就是如果空间不足，那他无法处理冲突也无法插入数据，因此需要装填因子(空间/插入数据)>=1。

那有没有一种方法可以解决这种问题呢？链地址法可以，链地址法的原理时如果遇到冲突，他就会在原地址新建一个空间，然后以链表结点的形式插入到该空间。我感觉业界上用的最多的就是链地址法。下面从百度上截取来一张图片，可以很清晰明了反应下面的结构。比如说我有一堆数据{1,12,26,337,353...}，而我的哈希算法是H(key)=key mod 16，第一个数据1的哈希值f(1)=1，插入到1结点的后面，第二个数据12的哈希值f(12)=12，插入到12结点，第三个数据26的哈希值f(26)=10，插入到10结点后面，第4个数据337，计算得到哈希值是1，遇到冲突，但是依然只需要找到该1结点的最后链结点插入即可，同理353。



1. **寻找字符串里颠倒而成的字母（题号438，Find All Anagrams in a String）**

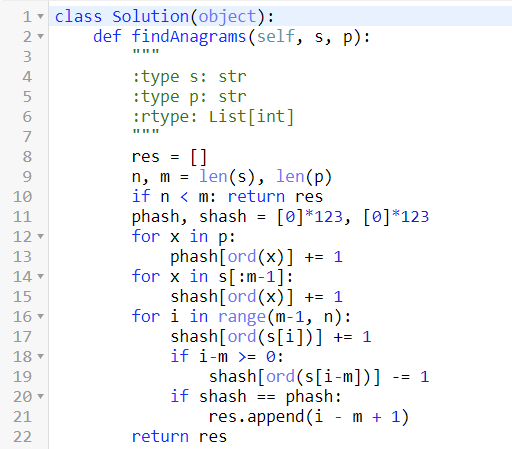
题目：给定非字符串s与p，找出所有p在s中颠倒形成的字母的起始位置



思路：先用hash表统计p中字母出现的次数，然后在s中制造滑框，当该滑框中的hash值与p的一样时，代表此滑框是p中的所有元素的颠倒组合，输出此时滑框的起点位置

关键点在于滑框的制造和统计次数的更新：首先统计s中前p的长度-1个字母出现的次数，然后对于m-1之后的数据，在统计出现次数之后，若i已经大于等于p的长度m，则调整s中之前字母出现的次数，相当于在后边的位置重新计数，更新滑块

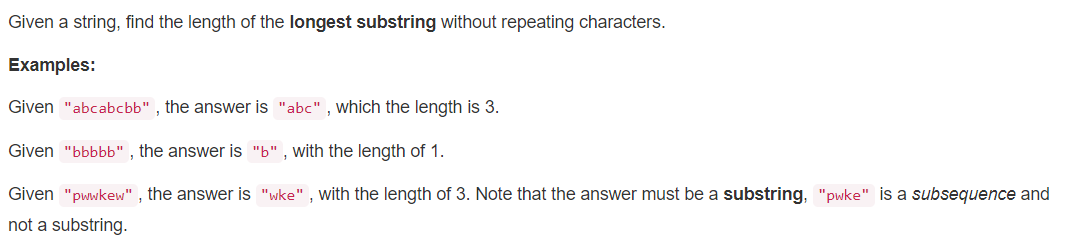
未想通：为何要从m-1处断开



Ord用于返回字母的ascii码

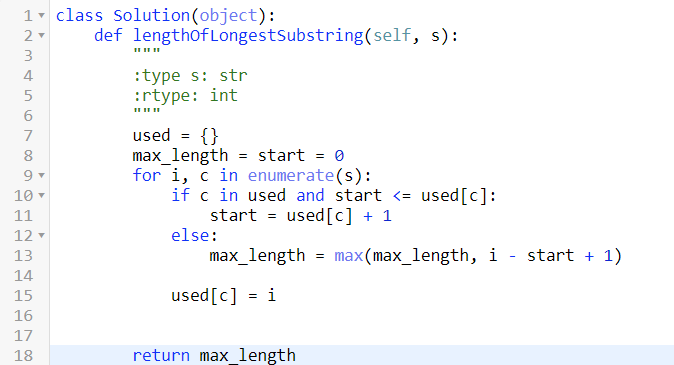
1. **寻找最长连续的不重复字符（题号：3 Longest Substring Without Repeating Characters）**

题目：给出一个字符串，寻找其最长的连续不重复字符的个数



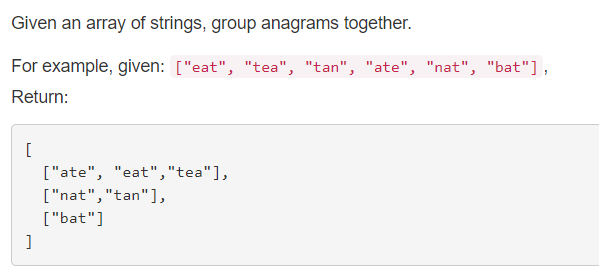
思路：创建一个字典，依次把没见过的字符放在里边为键，并以它的索引值为值，当遇到重复字符时，即键在字典里出现过，并且索引值大于起点值，则以它下一个索引值作为起点，重新开始创建字典，若没有重复值，则返回最大长度值，并且把这个字符与它的索引值放入字典

Hash表的解法待续

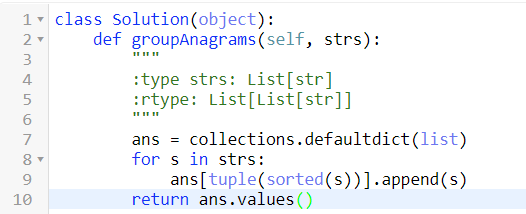


1. 寻找颠倒群组（题号49，Group Anagrams）

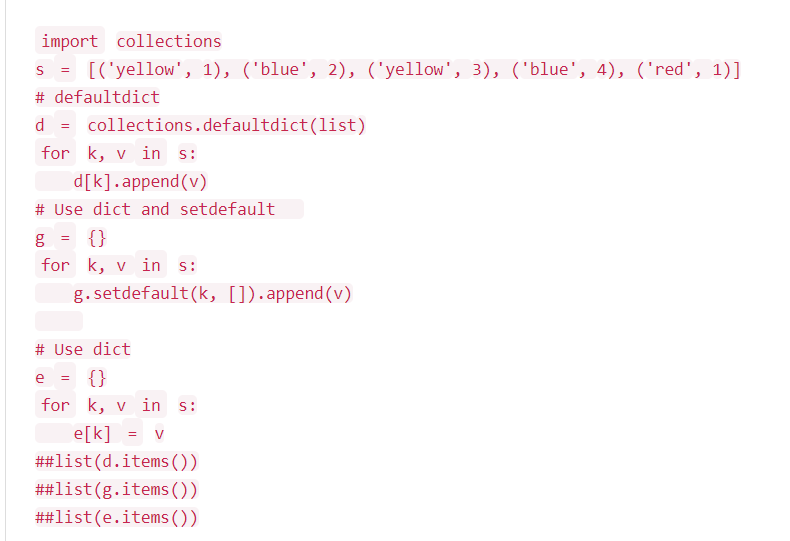
题目：给定一个字符串数组，返回它颠倒字母构成的群组

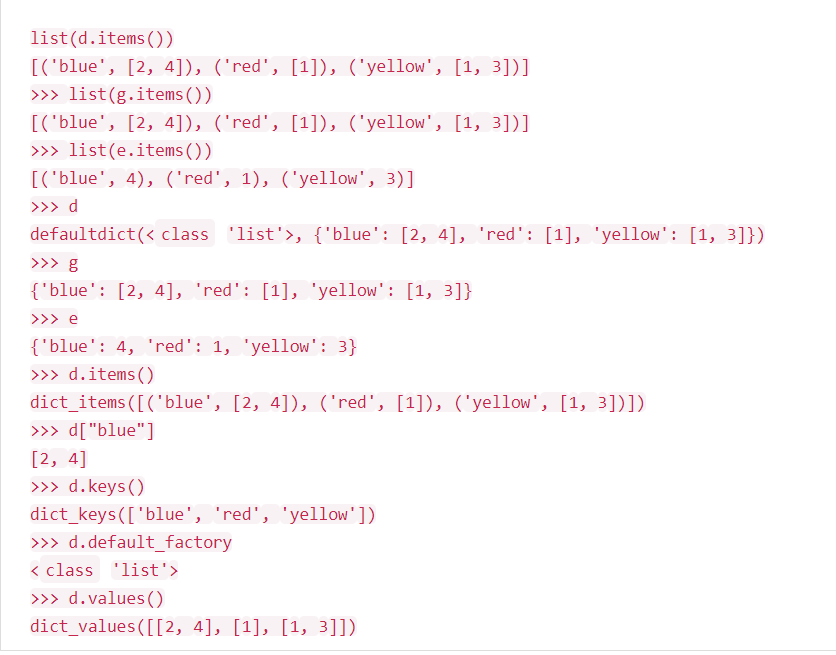


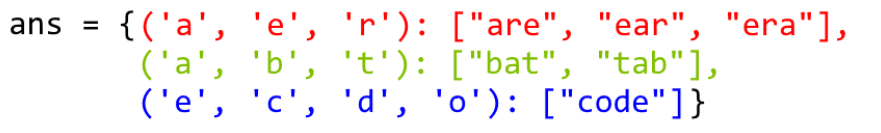
思路一：对每个字符串进行排序，是颠倒字母群的构成的排序一致



这里的defaultdict(function\_factory)构建的是一个类似dictionary的对象，其中keys的值，自行确定赋值，但是values的类型，是function\_factory的类实例，而且具有默认值。

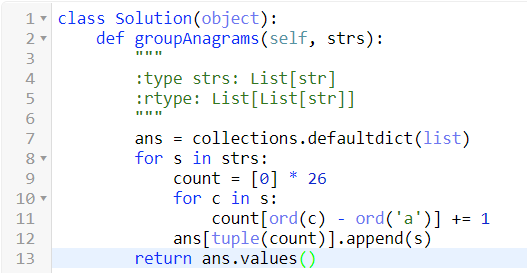






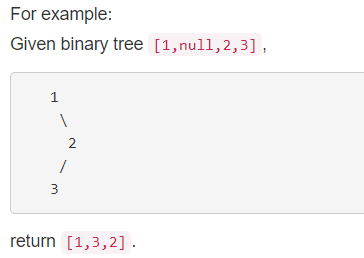
思路二：用Hash表计数

利用ascii码构造一个26个字母的hash表，计数每个字母出现的个数，一样的放一起，用元组和collections.defaultdict(list)结合



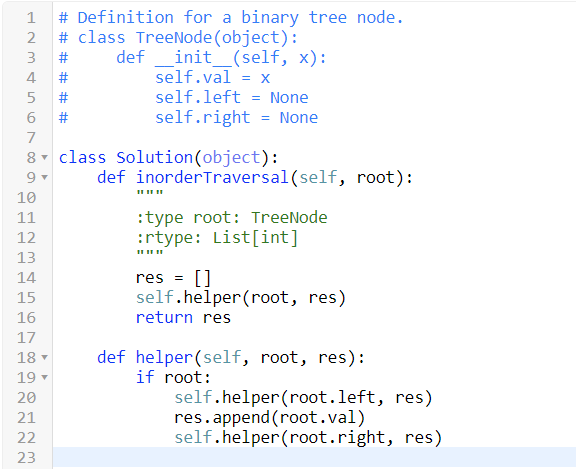
1. **二叉树中序遍历（题号94，Binary Tree Inorder Traversal）**

题目：给出一个二叉树，返回其中序遍历



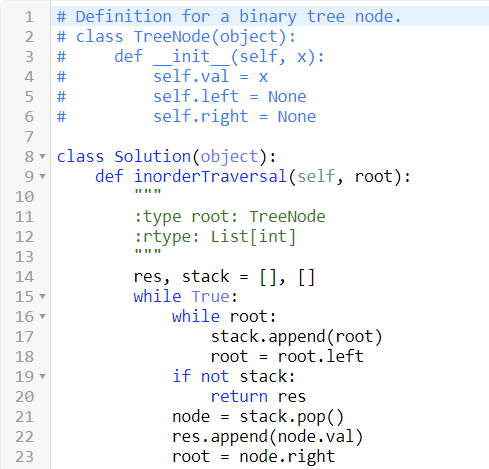
思路一：递归法

中序遍历是左根右，反复迭代结点即可（我理解的，构造root时已将左右根结点分类，相当于定义了一个类，因此后续的遍历要左有左，要右有右）



思路二：迭代法

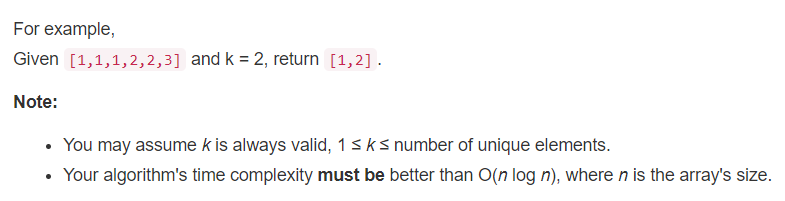
建立返回数组res[]以及栈stack,当root不为空时，让root进栈，并把root赋值为左侧结点，也就是将整个树的左侧结点都进栈，然后当无左侧结点时，开始出栈，先左边出，再中间放入res，然后把根结点放在right处，使右侧节点入栈再出栈进入res中，此时栈中剩下上一层根结点，使之出栈，然后在使根结点右侧的入栈出栈来实现遍历



理解过程可参考下述动画

https://leetcode.com/problems/binary-tree-inorder-traversal/solution/

1. 出现最频繁的前K个元素（题号347，Top K Frequent Elements）



思路：先用字典统计出每个元素出现的次数，再用堆来输出

堆结构仍需后续理解

