**前言：**第一篇Leetcode的日记，希望以后能养成习惯，渐渐的我会用英文取代，一个过程吧，记录自己的学习历程！先前用python刷了很多题，已经基本掌握了python各种数据结构和内置函数的用法，但是今天经过反复思考，感觉以后还是会入坑java，干脆早点转战，但第一天真的好累啊，啥函数都不会！

下面有几点感悟

1. String没有reverse函数，数组有，如果要对字符串倒置，可以现转换成StringBuilder，这个类有实现reverse函数
2. 有时用hashMap并没有很快，因为他会随着不停的put以后不断rehash，造成运算时间上升，所以如果有简单的O(n)循环解决的就不用hashmap了
3. String又一个A.startsWith(B)方法，可以判断字符串A是否有前缀B，这样就不用自己循环一个个判断了。

**正文：**

Leetcode 14 Longest Common Prefix

最长公共前缀，最基本的解法是暴力，每个字符串都从第i=0个字符开始比较，如果不相等或i已经超过了字符串的长度就可以直接退出，这时0--i-1之间的位置就是LCP。

然后在题解中看到了分治法求解，感觉个人对这个方法一直不够理解，只知道是归并排序使用的方法，所以好好地分析一下这个解法

* 源代码如下

class Solution {

    public String longestCommonPrefix(String[] strs) {

        //分治法

        if(strs.length==0)return "";

        return longestCommonPrefix(strs,0,strs.length-1);

}

    private String longestCommonPrefix(String[] strs,int left,int right)

    {

        if(left==right)return strs[left];

        int mid=(left+right)/2;

        String leftCP=longestCommonPrefix(strs,left,mid);

        String rightCP=longestCommonPrefix(strs,mid+1,right);

        return CommonPrefix(leftCP,rightCP);

}

    private String CommonPrefix(String leftCP,String rightCP)

    {

        int minlen=Math.min(leftCP.length(),rightCP.length());

        for(int i=0;i<minlen;i++)

        {

            if(leftCP.charAt(i)!=rightCP.charAt(i))return leftCP.substring(0,i);

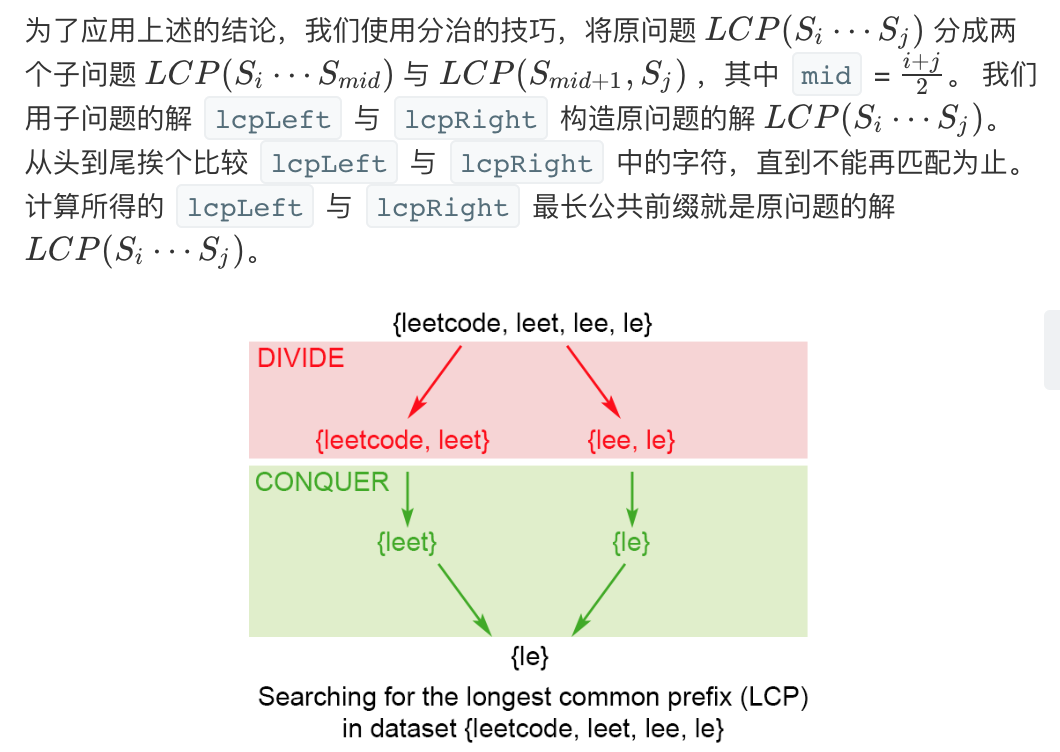
        }

        return leftCP.substring(0,minlen);

    }

}

* 具体思路



也就是用mid位置来确定LCP到哪个位置结束。先Divide，也就是不听将原数组一分为二，不停的Divide，直到left==right时表示时单个字符串了，直接返回该字符串（因为只有一个字符串时LCP肯定就是该字符串本身）。然后在左右两个字符串之间比较LCP，这部分要进入 CommonPrefix（）函数中比较，也就是左边的LCP和右边的LCP，Merge 生成新的LCP，这个过程不赘述。不停的Merge，最后会回到最原始的数组长度得到公公的LCP。

* 时间复杂度分析

注意n是字符串的数量，m是每个字符串的长度（都相同时复杂度最多，所以假设都相同）当前的时间T(n)=2\*(T(n/2))+O(m) 后者是Merge时，m次循环比较。这个递归运算最后得到

T(n)=n+n\*m所以O(n\*m)所有字符的长度

Leetcode 21 Merge Sorted Lists

此题是标准的用链表求解，但是因为太久每接触，刚碰到，思路还有但是有些无从下手，好在看了题解后马上能懂。但是在题解中遇到了递归解法，发现一个久远的问题！！！！那就是.....递归问题怎么求时间复杂度啊，所以我对这个递归解法进行以下分析

* 源代码如下：

class Solution {

    public ListNode mergeTwoLists(ListNode l1, ListNode l2) {

    //使用递归

    if(l1==null)return l2;

    if(l2==null)return l1;

    if(l1.val<=l2.val){

        l1.next=mergeTwoLists(l1.next,l2);

        return l1;

    }

    else {

        l2.next=mergeTwoLists(l1,l2.next);

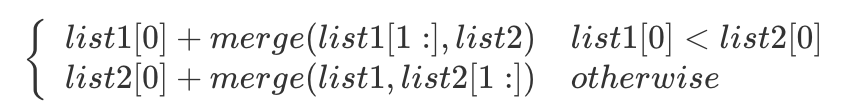
        return l2;

    }

    }

}

* 具体思路就如下：



这样最后返回的无论是l1还是l2都是头节点。

* 时间复杂度分析

因为每次递归都只移动了一个链表的一个节点，所以最后因该是O(n+m)表示两个链表的节点数。

* 空间复杂度分析

因为递归的时候，系统会自动开辟一个栈让当前变量存入其中，然后进入到下一个递归，所以进入n+m次递归，每次递归都存储一个变量，l1.next或l2.next 所以空间复杂度是O(n+m)