【后缀数组，字符串】 20180322_寻找一个字符串中的重复子串LRS

首先这是一个单字符串问题。子字符串 R 在字符串 L 中至少出现两次，则称 R 是 L 的重复子串。比如字符串 `abcdeabcd` 的LRS的长度是2，LRS是abcd

```
Longest Repeated Substring in GEEKSFORGEEKS is: GEEKS
Longest Repeated Substring in AAAAAAAAAA is: AAAAAAAAA
Longest Repeated Substring in ABCDEFG is: No repeated substring
Longest Repeated Substring in ABABABA is: ABABA
Longest Repeated Substring in ATCGATCGA is: ATCGA
Longest Repeated Substring in banana is: ana
Longest Repeated Substring in abcpqrabpqpq is: ab (pq is another LRS here)
```

一些定义：

后缀：后缀是指从某个位置i 开始到整个串末尾结束的一个特殊子串。字符串r 的从第i 个字符开始的后缀表示为Suffix(i)，也就是 `Suffix(i)=r[i..len(r)]` 。

后缀数组：后缀数组SA 是一个一维数组，它保存1..n 的某个排列 `SA[1]，SA[2]，...，SA[n]，` 并且保证 `Suffix(SA[i]) < Suffix(SA[i+1])，1≤i<n。` 也就是将S 的n 个后缀从小到大进行排序之后把排好序的后缀的开头位置顺次放入SA 中。 后缀数组最典型的是寻找一个字符串的重复子串

字符串大小比较： 字符串大小比较是指“字典顺序”，也就是对于两个字符串u、v。 令i 从1 开始顺次比较u[i]和v[i]，如果u[i]=v[i]则令i 加1，否则若u[i]v[i]则认为u>v（也就是vlen(u)或者i>len(v)仍比较不出结果，那么若len(u)v。

1- 方法一： 暴力破解

`O(n^3)` 直接子串和子串比较，查看所有字符串 比如 字符串 abcdabd, 先遍历第一个元素a,从第二个开始找，找到某个跟他一样的，开始游标k++,j++...然后记录相等子串长度；然后遍历第二个元素b开始...

```
public class LRS {
    private static int statLen(String X, int k, int j) {
        int cur_len = 0;

        while(k<X.length() && j<X.length() && X.charAt(k) == X.charAt(j)){
            k++;
            j++;
            cur_len++;
        }
        return cur_len;

    }
```

```
    // O(n^3)
    public static int naiveLRS(String x){
        int maxlen = 0;
        int length = x.length();
        for(int i=0;i<length;i++){
            int len = 0;
            int k = i;//第一个游标 k
            //第二个游标j
            for(int j=i+1;j<length;j++){
                len = statLen(x, k, j);
                if(maxlen<len){
                    maxlen = len;
                }
            }
        }
        return maxlen;
    }
    public static void main(String[] args) {
        String X = "ask not what your country can do for you,but what you can do for your
country";
        //String Y = "acaccbabb";
        System.out.println(naiveLRS(X));
    }

}
```

方法2

使用后缀数组降低时间复杂度为 `O(n^2)` 后缀数组的定义：一个字符数组，定义了一个字符串每个后缀子串的起始地址，如下图是banana这个字符串的后缀数组：

a[0]:banana

a[1]:anana

a[2]:nana

a[3]:ana

a[4]:na

a[5]:a

可以使用Treeset 保存后缀数组的String们..

算法的主要思想

- 利用 `后缀数组` 实现记录下来所有可能子串的起始地址，(用后缀数组的好处就是把所有起始位置相同的字符都放一起，就不用移动第二个指针 来找哪些字符与第一个指针指向的字符相同了 降低了一个n的维度的复杂度)
- 然后利用排序，把起始地址一致的字符放在一起：
  这样就简化了暴力法当中通过移动j来控制下一个相同元素的过程，（也就是少了一层循环，同利用后缀数组

的好处）因为相同的字符被放在了相邻的位置，对以后缀数组的元素开始的子串两两比较就知道了重复子串的长度。

For example

- 比如上面那个banana例子，对banana的一次循环遍历，得到后缀数组如上图所示.
- 然后对其排序，我们就得到了:

a[0]:a

a[1]:ana

a[2]:anana

a[3]:banana

a[4]:na

a[5]:nana

这时候，遍历后缀数组，每相邻的子串就按照（下第二个图comlen函数）的方法，计算两个子串的公共长度。注意，在C++语言中，后缀数组记录的（也就是数组中的元素）是一个字符指针，指向的是一个子串的起始字符。Java语言没有指针，实现起来可能比较麻烦，看到别人的一种实现方式是**利用Treeset，直接记录所有后缀数组对应的子串，但是感觉空间开销太大。**

```
Set<String> postfix = new TreeSet<String>();//存储后缀数组，并排序
for(int i = 0; i < s.length(); i++){
    postfix.add(s.substring(i));
}
int comlen(char * p, char * q)
{
    int len = 0;
    while(*p && *q && *p++ == *q++
    {
        ++len;
    }
    return len;
}
```

复杂度分析：

第一次遍历原始数组，得到后缀数组的复杂度是O（n），对后缀数组的排序复杂度是O(nlogn)， 计算最长重复子串，遍历后缀数组O（n）,每两个相邻元素的比较O（n），所以复杂度是O(n)+O(nlogn)+O(n^2) = O(n^2), 比暴力法的O（n^3）复杂度小，其主要的原因就是利用后缀数组这个数据结构缓存了每一个子串的起始位置，通过排序，避免了暴力法中通过j来定位的又一层循环~

注意：

可以在TreeSet里存放后缀字符串的起始位置，然后给TreeSet传入一个自定义的Comparator，就可以省去后缀字符串的开销了。