题目 字符串的排列

考点 分解让复杂问题简单 热点指数 45023 通过率 19.94%

具体题目

输入一个字符串,按字典序打印出该字符串中字符的所有排列。例如输入字符串abc,则打印出由字符a,b,c所能排列出来的所有字符串abc,acb,bac,bca,cab和cba。输入描述:输入一个字符串,长度不超过9(可能有字符重复),字符只包括大小写字母。

```
import java.util.ArrayList;
import java.util.List;
import java.util.Collections;
public class Solution {
    public ArrayList<String> Permutation(String str) {
        List<String> resultList = new ArrayList<>();
        if(str.length() == 0)
            return (ArrayList)resultList;
        //递归的初始值为（str数组，空的list，初始下标0）
        fun(str.toCharArray(),resultList,0);
        Collections.sort(resultList);
        return (ArrayList)resultList;
    }
```

```
    private void fun(char[] ch,List list,int i){        //这是递归的终止条件，就是i下标已经移到char数组的末尾的时候，考虑添加这一组字符串进入结果集中        if(i == ch.length-1){            //判断一下是否重复            if(!list.contains(new String(ch))){                list.add(new String(ch));                return;            }        }else{            //这一段就是回溯法，这里以"abc"为例
            //递归的思想与栈的入栈和出栈是一样的,某一个状态遇到return结束了之后，会回到被调用的地方继续执行
            //1.第一次进到这里是ch=['a','b','c'],list=[],i=0，我称为 状态A ，即初始状态            //那么j=0，swap(ch,0,0)，就是['a','b','c']，进入递归，自己调自己，只是i为1，交换(0,0)位置之后的状态我称为 状态B            //i不等于2，来到这里，j=1，执行第一个swap(ch,1,1)，这个状态我称为 状态C1 ,再进入fun函数，此时标记为T1，i为2，那么这时就进入上一个if，将"abc"放进list中            //////////////-------》此时结果集为["abc"]
            //2.执行完list.add之后，遇到return，回退到T1处，接下来执行第二个swap(ch,1,1)，状态C1又恢复为状态B
           //恢复完之后，继续执行for循环，此时j=2,那么swap(ch,1,2),得到"acb"，这个状态我称为C2,然后执行fun，此时标记为T2,发现i+1=2,所以也被添加进结果集，此时return回退到T2处往下执行            //////////////-------》此时结果集为["abc","acb"]            //然后执行第二个swap(ch,1,2)，状态C2回归状态B,然后状态B的for循环退出回到状态A
            //            a|b|c(状态A)            //              |            //             |swap(0,0)            //              |            //
 a|b|c(状态B)            //            /  \            //   swap(1,1)/    \swap(1,2)  (状态C1和状态C2)            //          /     \
  //         a|b|c   a|c|b
            //3.回到状态A之后，继续for循环，j=1,即swap(ch,0,1)，即"bac",这个状态可以再次叫做状态A,下面的步骤同上            //////////////-------》此时结果集为["abc","acb","bac","bca"]
            //            a|b|c(状态A)            //              |            //             |swap(0,1)            //              |            //
 b|a|c(状态B)            //            /  \            //   swap(1,1)/    \swap(1,2)  (状态C1和状态C2)            //          /     \
  //         b|a|c   b|c|a
            //4.再继续for循环，j=2,即swap(ch,0,2)，即"cab",这个状态可以再次叫做状态A，下面的步骤同上
//////////////-------》此时结果集为["abc","acb","bac","bca","cab","cba"]
            //            a|b|c(状态A)            //              |            //             |swap(0,2)            //              |            //
 c|b|a(状态B)            //            /  \            //   swap(1,1)/    \swap(1,2)  (状态C1和状态C2)            //          /     \
  //         c|b|a   c|a|b
            //5.最后退出for循环，结束。
            for(int j=i;j<ch.length;j++){                swap(ch,i,j);                fun(ch,list,i+1);                swap(ch,i,j);            }
    }    }
    //交换数组的两个下标的元素     private void swap(char[] str, int i, int j) {            if (i != j) {                char t = str[i];
            str[i] = str[j];                str[j] = t;            }        }    }
```

```
/* *递归算法 *于无重复值的情况 *1、定第一个字符，递归取得首位后面的各种字符串组合； *2、再把第一个字符与后面每一个字符交换，并同样递归获得首位后面的字符串组合；
*递归的出口，就是只剩一个字符的时候。 *递归的循环过程，就是从每个子串的第二个字符开始依次与第一个字符交换，然后继续处理子串。 *假如有重复值呢？ *由于全排列就是从第一个数字起，每个数分别与它后面的数字交换，我们先尝试加个这样的判断——如果一个数与后面的数字相同那么这两个数就不交换了。
*例如abb，第一个数与后面两个数交换得bab，bba。然后abb中第二个数和第三个数相同，就不用交换了。 *由于这里的bba和开始第一个数与第三个数交换的结果相同了，因此这个方法不行。
*换种思维，对abb，第一个数a与第二个数b交换得到bab，然后考虑第一个数与第三个数交换，此时由于第三个数等于第二个数， *所以第一个数就不再用与第三个数交换了。再考虑bab，它的第二个数与第三个数交换可以解决bba。此时全排列生成完毕！ */ import java.util.ArrayList; import java.util.Arrays; import java.util.Collections; import java.util.Set; import java.util.HashSet; public class Solution {     public ArrayList Permutation(String str) {
ArrayList list = new ArrayList();         if(str != null && str.length()>0) {
PermutationHelp(str.toCharArray(),0,list);             Collections.sort(list);         }
        return list;     }
    public void PermutationHelp(char[] chars, int i, ArrayList list) {         if(i == chars.length-1) {
list.add(String.valueOf(chars));         } else {             Set set = new HashSet();//去掉重复交换的字符             for(int
j=i; j<chars.length; j++) {                 if(j==i || !set.contains(chars[j])) {                     set.add(chars[j]);
swap(chars,i,j);//交换第一个字符与其它每个字符的位置                     PermutationHelp(chars,i+1,list);
swap(chars,i,j);//为了防止重复的情况，还需要将begin处的元素重新换回来                 }             }         }     }
    private void swap(char[] chars, int i, int j) {         char temp = chars[i];         chars[i] = chars[j];         chars[j] = temp;
    }
}
```