全排列问题FullSort

# 全排列问题分析

## 全排列的定义和公式：

对于给定的集合A{a1,a2,...,an},其中的n个元素互不相同，如何输出这n个元素的所有排列（全排列）。

从n个数中选取m（m<=n）个数按照一定的顺序进行排成一个列，叫作从n个元素中取m个元素的一个排列。由排列的定义，显然不同的顺序是一个不同的排列。从n个元素中取m个元素的所有排列的个数，称为**排列数**。

**从n个元素取出n个元素的一个排列，称为一个全排列**。全排列的排列数公式为n!，通过乘法原理可以得到。

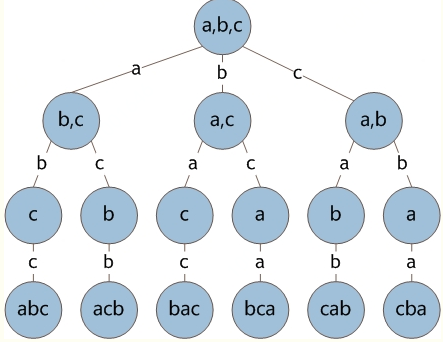
## 时间复杂度：

n个数（字符、对象）的全排列一共有n!种，所以**全排列算法至少时间O(n!)的**。如果要对全排列进行输出，那么输出的时间要**O(n∗n!)**，因为每一个排列都有n个数据。所以实际上，全排列算法对大型的数据是无法处理的，而一般情况下也不会要求我们去遍历一个大型数据的全排列。

## 全排列的思想

**<一>**

这里以A{a,b,c}为例，来说明全排列的生成方法，对于这个集合，其包含3个元素，所有的排列情况有3!=6种，对于每一种排列，其第一个元素有3种选择a,b,c，对于第一个元素为a的排列，其第二个元素有2种选择b,c；第一个元素为b的排列，第二个元素也有2种选择a，c，……，依次类推，我们可以将集合的全排列与一棵多叉树对应。如下图所示



在此树中，每一个从树根到叶子节点的路径，就对应了集合A的一个排列。通过递归算法，可以避免多叉树的构建过程，直接生成集合A的全排列，代码如下。

**<二>**

解决一个算法问题，我比较习惯于从**基本的想法**做起，我们先回顾一下我们自己是如何写一组数的全排列的：1，3，5，9（为了方便，下面我都用数进行全排列而不是字符）。

【1，3，5，9】（第一个）

首先保持第一个不变，对【3，5，9】进行全排列。

同样地，我们先保持3不变，对【5，9】进行全排列。

保持5不变，对9对进行全排列，由于9只有一个，它的排列只有一种：9。

故排列为【1，3，5，9】

接下来不能以5打头了，5与9相互交换，得到

【1，3，9，5】

此时5，9的情况都写完了，不能以3打头了，得到

1，5，3，9

1，5，9，3

1，9，3，5

1，9，5，3

这样，我们就得到了1开头的所有排列，这是我们一般的排列数生成的过程。再接着是以3、5、9打头，得到全排列。

我们现在做这样的一个假设，**假设给定的一些序列中第一位都不相同，那么就可以认定说这些序列一定不是同一个序列**，这是一个很显然的问题。有了上面的这一条结论，我们就可以同理得到如果在第一位相同，可是第二位不同，那么在这些序列中也一定都不是同一个序列。

那么，这个问题可以这样来看。对

T=【**x1**,x2,x3,x4,x5,........xn−1,xn】

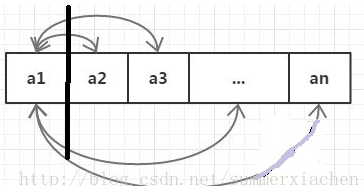
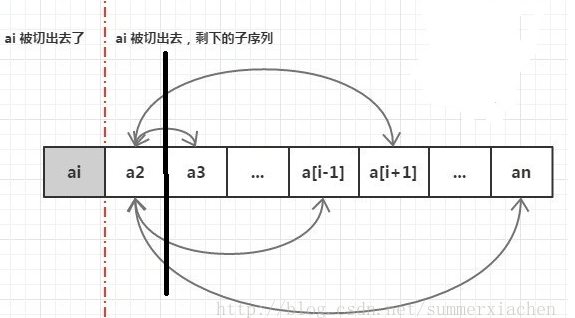
我们获得了在第一个位置上的所有情况之后(注：是所有的情况)，对每一种情况，抽去序列T中的第一个位置，那么对于剩下的序列可以看成是一个全新的序列

T1=【x2,x3,x4,x5,........xn−1,xn】

序列T1可以认为是与之前的序列毫无关联了。同样的，我们可以对这个T1进行与T相同的操作，直到T中只一个元素为止。这样我们就获得了所有的可能性。所以很显然，这是一个递归算法。

第一位的所有情况：**无非是将x1与后面的所有数x2,x3,.......xn依次都交换一次。**

**示意图：**

## Java代码

/\*\*

\* 全排列

\* @param nums 数组

\* @param start 开始索引

\* @param lists 结果集合

\*/

public void fullSortArray(int[] nums,int start,ArrayList<ArrayList<Integer>> lists){

**if(start == nums.length){**

ArrayList<Integer> list = new ArrayList<Integer>();

for(int num : nums)

list.add(num);

lists.add(list);

// System.out.println(Arrays.toString(nums));

}

for(int i = start;i<nums.length;i++){

**swap(nums,i,start);//交换位置**

**fullSortArray(nums,start + 1,lists);//递归调用**

**swap(nums,i,start);//复位**

}

}

//交换数组中两个位置的元素

public void swap(int[] nums,int i,int j){

int temp = nums[i];

nums[i] = nums[j];

nums[j] = temp;

}

测试：

@Test

public void test(){

// int[] nums = {1,2,3};

int[] nums = {1,2,3,4};

ArrayList<ArrayList<Integer>> lists = new ArrayList<ArrayList<Integer>>();

**fullSortArray(nums,0,lists);**

//一共应该是n!种情况

System.out.println("lists.size() = " + lists.size());

System.out.println("lists = " + lists);

}

**【注意】**swap(nums,i,start);//复位交换回来

我们来仔细推敲一下循环体里的代码，当我们对序列进行**交换之后**，就将交换后的序列**除去第一个元素**放入到下一次递归中去了，**递归完成了再进行下一次循环**。这是某一次循环程序所做的工作，这里有一个问题，那就是在进入到下一次循环时，序列是被改变了。可是，如果我们要假定第一位的所有可能性的话，那么，**就必须是在建立在这些序列的初始状态一致的情况下,所以每次交换后，要还原，确保初始状态一致**。