

# 排列类算法问题大总结



六尺帐篷 (/u/f8e9b1c246f1)  
2017.03.26 19:57 字数 1988 阅读 263 评论 0 喜欢 13  
(/u/f8e9b1c246f1)

编辑文章 (/writer#/notebooks/7106551/notes/10628721)

- 全排列
- 带重复元素的排列
- 下一个排列
- 上一个排列
- 第 k 个排列
- 排列序号
- 排列序号II

## 全排列

给定一个数字列表，返回其所有可能的排列。

注意事项

你可以假设没有重复数字。

样例

给出一个列表[1,2,3]，其全排列为：

```
[
  [1,2,3],
  [1,3,2],
  [2,1,3],
  [2,3,1],
  [3,1,2],
  [3,2,1]
]
```

## 分析

可以用递归和非递归解决

首先递归法，也是利用了回溯法和深度优先搜索。

我们考虑一个一个将数组元素加入到排列中，递归求解，就好像下面的解答树：



Paste\_Image.png

添加的时候排除掉相同的元素即可，回溯法我们经常会有一个已访问标识数组，来表示数组被访问过，但这里不用这样，因为如果list里面已经包含就说明已经访问过了，所以只要判断，跳过已有的元素即可。

再考虑递归的结束条件，当元素都添加足够就结束了，添加足够的意思就是，元素个数等于数组的长度。

```
class Solution {  
    /**  
     * @param nums: A list of integers.  
     * @return: A list of permutations.  
     */  
    public List<List<Integer>> permute(int[] nums) {  
        List<List<Integer>> res = new ArrayList<>();  
        if(nums == null)  
            return res;  
        if(nums.length == 0)  
        {  
            res.add(new ArrayList<Integer>());  
            return res;  
        }  
  
        ArrayList<Integer> list = new ArrayList<>();  
        dfs(res, list, nums);  
        return res;  
    }  
  
    private void dfs(List<List<Integer>> res, ArrayList<Integer> list, int[] nums) {  
  
        int n = nums.length;  
        if(list.size() == n)  
        {  
            res.add(new ArrayList<Integer>(list));  
            return;  
        }  
  
        for(int i = 0; i < n; i++) {  
            if(list.contains(nums[i]))  
                continue;  
            list.add(nums[i]);  
            dfs(res, list, nums);  
            list.remove(list.size() - 1);  
        }  
    }  
}
```

#### 非递归实现

思路是这样的，就是高中的排列组合知识，运用插入法即可，假设有*i*个元素的排列组合，那么对于*i+1*个元素，可以考虑就是将*i+1*的元素插入到上述的排列的每一个位置即可。



```
class Solution {  
    /**  
     * @param nums: A list of integers.  
     * @return: A list of permutations.  
     */  
    public List<List<Integer>> permute(int[] nums) {  
        List<List<Integer>> res = new ArrayList<List<Integer>>();  
        if ( nums == null)  
            return res;  
        if( nums.length == 0)  
        {  
            res.add(new ArrayList<Integer>());  
            return res;  
        }  
        List<Integer> list = new ArrayList<>();  
        list.add(nums[0]);  
        res.add(new ArrayList<Integer>(list));  
  
        for(int i=1;i<nums.length;i++) {  
            int size1 = res.size();  
            for(int j=0;j<size1;j++) {  
                int size2 = res.get(j).size();  
                for(int k=0;k<=size2;k++) {  
                    ArrayList<Integer> temp = new ArrayList<>(res.get(j));  
                    temp.add(k,nums[i]);  
                    res.add(temp);  
                }  
                res.remove(j);  
            }  
        }  
        return res;  
    }  
}
```

## 带重复元素的全排列

给出一个具有重复数字的列表，找出列表所有不同的排列。

样例

给出列表 [1,2,2]，不同的排列有：

```
[  
  [1,2,2],  
  [2,1,2],  
  [2,2,1]  
]
```

Paste\_Image.png

## 代码



```

class Solution {
    /**
     * @param nums: A list of integers.
     * @return: A list of unique permutations.
     */
    public List<List<Integer>> permuteUnique(int[] nums) {

        ArrayList<List<Integer>> res = new ArrayList<>();

        if(nums == null)
            return null;

        if(nums.length == 0)
        {
            res.add(new ArrayList<Integer>());
            return res;
        }

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

        //先将数组排序，这样相同元素将会出现在一起
        Arrays.sort(nums);

        int n = nums.length;
        int[] visited = new int[n];
        for(int i=0;i<n;i++)
            visited[i] = 0; //0标识未访问

        helper(res, list, visited, nums);
        return res;
    }

    private void helper(ArrayList<List<Integer>> res, ArrayList<Integer> list, int[] visited)

        if(nums.length == list.size()) {
            res.add( new ArrayList<Integer>(list));
        }

        for(int i=0;i<nums.length;i++) {

            if(visited[i] == 1 || i!= 0 && (visited[i-1] == 0 && nums[i] == nums[i-1]))
                continue;

            /*
             上面的判断主要是为了去除重复元素影响。
             比如，给出一个排好序的数组，[1,2,2]，那么第一个2和第二2如果在结果中互换位置，
             我们也认为是同一种方案，所以我们强制要求相同的数字，原来排在前面的，在结果
             当中也应该排在前面，这样就保证了唯一性。所以当前面的2还没有使用的时候，就
             不应该让后面的2使用。
            */
            list.add(nums[i]);
            visited[i] = 1;
            helper(res, list, visited, nums);
            list.remove(list.size()-1);
            visited[i] = 0;

        }

    }
}

```

## 下一个排列

给定一个若干整数的排列，给出按正数大小进行字典序从小到大排序后的下一个排列。

如果没有下一个排列，则输出字典序最小的序列。

样例

左边是原始排列，右边是对应的下一个排列。

1,2,3 → 1,3,2

3,2,1 → 1,2,3

1,1,5 → 1,5,1



分析

这道题让我们求下一个排列顺序，有题目中给的例子可以看出来，如果给定数组是降序，则说明是全排列的最后一种情况，则下一个排列就是最初始情况，可以参见之前的博客Permutations 全排列 (<http://www.cnblogs.com/grandyang/p/4358848.html>)。我们再来看下面一个例子，有如下的一个数组

1    2    7    4    3    1

下一个排列为：

1    3    1    2    4    7

那么是如何得到的呢，我们通过观察原数组可以发现，如果从末尾往前看，数字逐渐变大，到了2时才减小的，然后我们再从后往前找第一个比2大的数字，是3，那么我们交换2和3，再把此时3后面的所有数字转置一下即可，步骤如下：

1    2    7    4    3    1

1    2    7    4    3    1

1    3    7    4    2    1

1    3    1    2    4    7

所以我们要做的就是找到第一个比peak元素大的数字，交换，然后反转

```
public class Solution {
    /**
     * @param nums: an array of integers
     * @return: return nothing (void), do not return anything, modify nums in-place instead
     */
    public int[] nextPermutation(int[] nums) {
        int i = nums.length - 2;
        while (i >= 0 && nums[i + 1] <= nums[i]) {
            i--;
        }
        if (i >= 0) {
            int j = nums.length - 1;
            while (j >= 0 && nums[j] <= nums[i]) {
                j--;
            }
            swap(nums, i, j);
        }
        reverse(nums, i + 1);
        return nums;
    }

    private void reverse(int[] nums, int start) {
        int i = start, j = nums.length - 1;
        while (i < j) {
            swap(nums, i, j);
            i++;
            j--;
        }
    }

    private void swap(int[] nums, int i, int j) {
        int temp = nums[i];
        nums[i] = nums[j];
        nums[j] = temp;
    }
}
```

上一个排列

给定一个整数数组来表示排列，找出其上一个排列。

注意事项

排列中可能包含重复的整数

样例

给出排列[1,3,2,3]，其上一个排列是[1,2,3,3]

给出排列[1,2,3,4]，其上一个排列是[4,3,2,1]

📄

🔖

🔗

分析

与求下一个排列是一样的方法，只是相应的操作变反即可

```
public class Solution {
    /**
     * @param nums: A list of integers
     * @return: A list of integers that's previous permuation
     */
    public void swapItem(ArrayList<Integer> nums, int i, int j) {
        Integer tmp = nums.get(i);
        nums.set(i, nums.get(j));
        nums.set(j, tmp);
    }
    public void swapList(ArrayList<Integer> nums, int i, int j) {
        while ( i < j) {
            swapItem(nums, i, j);
            i ++; j --;
        }
    }
    public ArrayList<Integer> previousPermuation(ArrayList<Integer> nums) {
        int len = nums.size();
        if ( len <= 1)
            return nums;
        int i = len - 1;
        while ( i > 0 && nums.get(i) >= nums.get(i-1) )
            i --;
        swapList(nums, i, len - 1);
        if ( i != 0) {
            int j = i;
            while ( nums.get(j) >= nums.get(i-1) ) j++;
            swapItem(nums, j, i-1);
        }
        return nums;
    }
}
```

第k个排列

给定 n 和 k，求123..n组成的排列中的第 k 个排列。

注意事项

$1 \leq n \leq 9$

样例

对于 n = 3, 所有的排列如下：

- 123
- 132
- 213
- 231
- 312
- 321

如果 k = 4, 第4个排列为， 231.

分析

康托展开的公式：（不用记，看形势就行，下面会有例子）

$X=a_n(n-1)!+a_{n-1}(n-2)!+...+a_i(i-1)!+...+a_21!+a_1*0!$

$a_i$ 为整数，并且 $0 \leq a_i < i(1 \leq i \leq n)$

适用范围：没有重复元素的全排列

📄

🔖

🔗

N个数的第k个排序，例子，1，2，3，4共有4! 种排列，1234,1243,1324等等。按顺序应该是

1234

1243

1324

1342

1423

1432等等

可以通过STL中next\_permutation (begin, end) ;来算下一个全排列，理论上你要算n个数的第k个排列只要调用k-1次next\_permutation()就行，但是一般来说肯定会超时的，因为next\_permutation的时间复杂度是O(n)（如果自己写出来next\_permutation时间复杂度比n大就要注意了，其中一个容易疏忽的地方是最后排序可以用reverse而不是sort）。所以如果用这个的话时间复杂度是O(N^2)。

而用康托展开只要O(n)就行，下面来说说具体怎么做：

题目：找出第16个n = 5的序列（12345）

首先第十六个也就是要前面有15个数，要调用15次next\_permutation函数。

根据第一行的那个全排列公式， $15 / 4! = 0 \dots 15 =$  有0个数比他小的数是1，所以第一位是1

拿走刚才的余数15，用 $15 / 3! = 2 \dots 3 =$  剩下的数里有两个数比他小的是4（1已经没了），所以第二位是4

拿走余数3，用 $3 / 2! = 1 \dots 1 =$  剩下的数里有一个数比他小的是3，所以第三位是3

拿走余数1，用 $1 / 1! = 1 \dots 0 =$  剩下的数里有一个数比他小的是5（只剩2和5了），所以第四位是5

所以排列是 1,4,3,5,2



```

class Solution {
    /**
     * @param n: n
     * @param k: the kth permutation
     * @return: return the k-th permutation
     */
    public String getPermutation(int n, int k) {

        StringBuilder sb = new StringBuilder();
        boolean[] used = new boolean[n];

        k = k - 1;
        int factor = 1;
        for (int i = 1; i < n; i++) {
            factor *= i;
        }

        for (int i = 0; i < n; i++) {
            int index = k / factor;
            k = k % factor;
            for (int j = 0; j < n; j++) {
                if (used[j] == false) {
                    if (index == 0) {
                        used[j] = true;
                        sb.append((char) ('0' + j + 1));
                        break;
                    } else {
                        index--;
                    }
                }
            }
        }
        if (i < n - 1) {
            factor = factor / (n - 1 - i);
        }
    }

    return sb.toString();
}
}

```

## 排列序号

给出一个不含重复数字的排列，求这些数字的所有排列按字典序排序后该排列的编号。其中，编号从1开始。

样例

例如，排列 [1,2,4] 是第 1 个排列。

## 分析

这道题是求第k个排列的反向思维

已知是n = 5，求14352是它的第几个序列？（同一道题）

用刚才的那道题的反向思维：

第一位是1，有0个数小于1，即 $0 * 4!$

第二位是4，有2个数小于4，即 $2 * 3!$

第三位是3，有1个数小于3，即 $1 * 2!$

第四位是5，有1个数小于5，即 $1 * 1!$

第五位是2，不过不用算，因为肯定是0

所以14352是 n = 5 的第  $0 + 12 + 2 + 1 + 0 = 15 + 1$ （求的是第几个，所以要加一）= 16

第16个，跟刚才那道题一样，证明对了





```
public class Solution {
    /**
     * @param A an integer array
     * @return a long integer
     */
    public long permutationIndex(int[] A) {
        // Write your code here
        HashMap<Integer, Integer> hash = new HashMap<Integer, Integer>();

        for (int i = 0; i < A.length; i++) {
            if (hash.containsKey(A[i]))
                hash.put(A[i], hash.get(A[i]) + 1);
            else {
                hash.put(A[i], 1);
            }
        }
        long ans = 0;
        for (int i = 0; i < A.length; i++) {
            for (int j = i + 1; j < A.length; j++) {
                if (A[j] < A[i]) {
                    hash.put(A[j], hash.get(A[j])-1);
                    ans += generateNum(hash);
                    hash.put(A[j], hash.get(A[j])+1);
                }
            }
            hash.put(A[i], hash.get(A[i])-1);
        }
        return ans+1;
    }

    long fac(int numerator) {
        long now = 1;
        for (int i = 1; i <= numerator; i++) {
            now *= (long) i;
        }
        return now;
    }

    long generateNum(HashMap<Integer, Integer> hash) {
        long denominator = 1;
        int sum = 0;
        for (int val : hash.values()) {
            if(val == 0 )
                continue;
            denominator *= fac(val);
            sum += val;
        }
        if(sum==0) {
            return sum;
        }
        return fac(sum) / denominator;
    }
}
```

排列序号II

给出一个可能包含重复数字的排列，求这些数字的所有排列按字典序排序后该排列在其中的编号。编号从1开始。

样例

给出排列[1, 4, 2, 2]，其编号为3。

分析

这道题基于查找不存在重复元素中排列序号的基础之上，

即 $P(n) = P(n-1)+C(n-1)$

$C(n-1)$  = (首元素为小于当前元素，之后的全排列值)

$P(1) = 1;$

而不存在重复元素的全排列值 $C(n-1) = (n-1)! * k$ ( $k$ 为首元素之后小于当前元素的个数)



在存在重复元素的排列中首先全排列的值的求法变为：

$C(n-1) = (n-1)! / (A_1! A_2! \cdots A_j! k)$  (其中  $A_i$  为重复元素的个数,  $k$  为小于首元素前不重复的个数)

```
/**
 * @param A an integer array
 * @return a long integer
 */
long fac(int numerator) {
    long now = 1;
    for (int i = 1; i <= numerator; i++) {
        now *= (long) i;
    }
    return now;
}

long generateNum(HashMap<Integer, Integer> hash) {
    long denominator = 1;
    int sum = 0;
    for (int val : hash.values()) {
        if(val == 0)
            continue;
        denominator *= fac(val);
        sum += val;
    }
    if(sum==0) {
        return sum;
    }
    return fac(sum) / denominator;
}

public long permutationIndexII(int[] A) {
    HashMap<Integer, Integer> hash = new HashMap<Integer, Integer>();

    for (int i = 0; i < A.length; i++) {
        if (hash.containsKey(A[i]))
            hash.put(A[i], hash.get(A[i]) + 1);
        else {
            hash.put(A[i], 1);
        }
    }
    long ans = 0;
    for (int i = 0; i < A.length; i++) {
        HashMap<Integer, Integer> flag = new HashMap<Integer, Integer>();

        for (int j = i + 1; j < A.length; j++) {
            if (A[j] < A[i] && !flag.containsKey(A[j])) {
                flag.put(A[j], 1);
                hash.put(A[j], hash.get(A[j])-1);
                ans += generateNum(hash);
                hash.put(A[j], hash.get(A[j])+1);
            }
        }
        hash.put(A[i], hash.get(A[i])-1);
    }
    return ans + 1;
}
```


 LintCode&&LeetCode (/nb/7106551)

© 著作权归作者所有



六尺帐篷 (/u/f8e9b1c246f1)

写了 245418 字, 被 15240 人关注, 获得了 1429 个喜欢  
(/u/f8e9b1c246f1)

 喜欢 13



更多分享

(http://cwb.assets.jianshu.io/notes/images/1062872




被以下专题收入，发现更多相似内容


投稿管理


+ 收入我的专题

 程序员 (/c/NEt52a?utm\_source=desktop&utm\_medium=notes-included-collection)

 Android知识 (/c/3fde3b545a35?utm\_source=desktop&utm\_medium=notes-included-collection)

 Android开发 (/c/d1591c322c89?utm\_source=desktop&utm\_medium=notes-included-collection)

 今日看点 (/c/3sT4qY?utm\_source=desktop&utm\_medium=notes-included-collection)

 排序/算法 (/c/84adabbfa359?utm\_source=desktop&utm\_medium=notes-included-collection)





