**Lab6**

## ● 要点

1. 理解并实现康托展开与其逆运算，理解其精髓所在。
2. 实现阶乘函数，加强对 java method 的编写能力。
3. 灵活运用数组，并与循环结合。
4. 锻炼将算法描述转化为代码的能力。

## ● 注意事项

1.请勿抄袭

2.遇到问题请立即联系 TA

## ● 背景介绍

**康托展开简介**

康托展开是一个全排列到一个自然数的双射。康托展开的实质是计算当前排列在所有由小到大全排列中的顺序，因此是可逆的。

以下称第 x 个全排列是都是指由小到大的顺序。

**公式**

---

X=a[n]*(n-1)!+a[n-1]*(n-2)!+...+a[i]*(i-1)!+...+a[1]*0!

其中,a[i]为整数,并且 0<=a[i]<i,1<=i<=n。

a[i]的意义参见举例中的解释部分

**举例**

例如，3 5 7 4 1 2 9 6 8 展开为 98884。因为 X=2*8!+3*7!+4*6!+2*5!+0*4!+0*3!+2*2!+0*1!+0*0!=98884.

**解释**

排列的第一位是 3，比 3 小的数有两个，以这样的数开始的排列有 8!个，因此第一项为 2*8! 排列的第二位是 5，比 5 小的数有 1、2、3、4，由于 3 已经出现，因此共有 3 个比 5 小的数，这样的排列有 7!个，因此第二项为 3*7! 以此类推，直至 0*0!

最后，因为 X 是比该序列小的所有序列的个数，所以算出的该序列的位置应该是 X+1。

**用途**

---

显然，n 位（0~n-1）全排列后，其康托展开唯一且最大约为 n!，因此可以由更小的空间来储存这些排列。由公式可将 X 逆推出对应的全排列。

**用途举例请参见附录**

**康托展开的逆运算**

既然康托展开是一个双射，那么一定可以通过康托展开值求出原排列，即可以求出 n 的全排列中第 x 大排列。

如 n=5,x=96 时：

```
首先用 96-1 得到 95，说明 x 之前有 95 个排列.(将此数本身减去！)
用 95 去除 4！得到 3 余 23，说明有 3 个数比第 1 位小，所以第一位是 4.
用 23 去除 3！得到 3 余 5，说明有 3 个数比第 2 位小，所以是 4，但是 4 已出现过，因此是 5.
用 5 去除 2!得到 2 余 1，类似地，这一位是 3.
用 1 去除 1!得到 1 余 0，这一位是 2.
最后一位只能是 1.
所以这个数是 45321.
```

按以上方法可以得出通用的算法。

- **具体内容：**
  1. 实现阶乘函数, public static int factorial(int number)
  2. 实现康托展开函数，public static int cantorExpansion(String arrangement), 其中 arrangement 为某全排列，例如” 357412968”。
  3. 实现康托展开的逆运算函数，public static String cantorInverseOperation(int n,int x)，n 为全排列的长度，x 为排列在全排列中的位置。

- **检查内容**

  检查前两个函数的正确编写，代码风格，一共 10 分。

- **截止时间**

  前两部分内容当堂检查，并将代码提交到 work_upload\lab6 目录下。第三部分为选作。

## ● 附录：用途举例

例如我们常玩的挪动拼图游戏，就是格子中有一个为空，可以将周围的某个图挪到空地，之后该图原位置成为空地。

假设我们原始拼图给每个图片编号按照最终的完成状态

1 ，2 ，3， 4

5 ，6 ，7， 8

9(空),10，11，12

13，14，15，16

这样的方式对应。

如果游戏有个要求，给用户以实时提示，当玩家新走出的某种图的状态是以前曾经经历过的，给用户以提示，那么我们需要在程序中给每个玩家经历过的状态以记录。

如下面这个图，我们可以

1.用字符串”5,3,2,10,11,7,8,12,1,4,16,14,15,9,13,6”来保存，当玩家经历每一个新的状态时，新产生的字符串与以前每个字符串做一一比较，若发现重复，则提示。这样随着状态的增多，需要比对的状态越来越多，需要的存储空间越来越大。

2.开一个 boolean[][][][][][][][][][][][][][][][] label=new boolean [16][16][16][16][16][16][16][16][16][16][16][16][16][16][16][16] 数组，每一个状态经历过后，将数组中对应的位置置为 true，方便以后与之比较。

而我们采用康托展开则只需要 16! 这样多的空间即可保存这些状态。

我们可以采用 boolean[] label=new boolean[ 16! ];与 16^16 相比，所需空间为原来的 1/20922789888000。(当然 16!这么多的空间其实计算机也是无法承受的)