# 康托展开

**康托展开**是一个全排列到一个[自然数](https://baike.baidu.com/item/%E8%87%AA%E7%84%B6%E6%95%B0" \t "https://baike.baidu.com/item/%E5%BA%B7%E6%89%98%E5%B1%95%E5%BC%80/_blank)的[双射](https://baike.baidu.com/item/%E5%8F%8C%E5%B0%84" \t "https://baike.baidu.com/item/%E5%BA%B7%E6%89%98%E5%B1%95%E5%BC%80/_blank)，常用于构建[哈希表](https://baike.baidu.com/item/%E5%93%88%E5%B8%8C%E8%A1%A8" \t "https://baike.baidu.com/item/%E5%BA%B7%E6%89%98%E5%B1%95%E5%BC%80/_blank)时的空间压缩。 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。

目录

1. 1 [原理介绍](https://baike.baidu.com/item/%E5%BA%B7%E6%89%98%E5%B1%95%E5%BC%80/7968428?fr=aladdin" \l "1)
2. ▪ [康托展开运算](https://baike.baidu.com/item/%E5%BA%B7%E6%89%98%E5%B1%95%E5%BC%80/7968428?fr=aladdin" \l "1_1)
3. 2 [康托展开和逆康托展开](https://baike.baidu.com/item/%E5%BA%B7%E6%89%98%E5%B1%95%E5%BC%80/7968428?fr=aladdin" \l "2)
4. ▪ [康托展开举例](https://baike.baidu.com/item/%E5%BA%B7%E6%89%98%E5%B1%95%E5%BC%80/7968428?fr=aladdin" \l "2_1)
5. ▪ [代码实现](https://baike.baidu.com/item/%E5%BA%B7%E6%89%98%E5%B1%95%E5%BC%80/7968428?fr=aladdin" \l "2_2)
6. ▪ [逆康托展开举例](https://baike.baidu.com/item/%E5%BA%B7%E6%89%98%E5%B1%95%E5%BC%80/7968428?fr=aladdin" \l "2_3)
7. ▪ [代码实现](https://baike.baidu.com/item/%E5%BA%B7%E6%89%98%E5%B1%95%E5%BC%80/7968428?fr=aladdin" \l "2_4)
8. 3 [用途](https://baike.baidu.com/item/%E5%BA%B7%E6%89%98%E5%B1%95%E5%BC%80/7968428?fr=aladdin" \l "3)

## 原理介绍

### 康托展开运算

IMG_256

其中, IMG_257 为整数,并且 IMG_258 。

IMG_259 表示原数的第i位在当前未出现的元素中是排在第几个

z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1)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。 [1]

## 康托展开和逆康托展开

### 康托展开举例

再举个例子说明。  
　　在 IMG_260 5个数的排列组合中，计算34152的康托展开值。

首位是3，则小于3的数有两个，为1和2， IMG_261 ，则首位小于3的所有排列组合为 IMG_262

第二位是4，由于第一位小于4，1、2、3中一定会有1个充当第一位，所以排在4之下的只剩2个，所以其实计算的是在第二位之后小于4的个数。因此 IMG_263 。

第三位是1，则在其之后小于1的数有0个，所以 IMG_264 。

第四位是5，则在其之后小于5的数有1个，为2，所以 IMG_265 。

最后一位就不用计算啦，因为在它之后已经没有数了，所以 IMG_266 固定为0

根据公式：

IMG_267

　　所以比34152小的组合有61个，即34152是排第62。

### 代码实现

#include<iostream>

**using** **namespace** std;

**const** **int** factorial[]={1,1,2,6,24,120,720,5040,40320,362880,3628800};//阶乘0-10

**int** cantor(**int** a[],**int** n){//cantor展开,n表示是n位的全排列，a[]表示全排列的数（用数组表示）

**int** ans=0,sum=0;

**for**(**int** i=1;i<n;i++){

**for**(**int** j=i+1;j<=n;j++)

**if**(a[j]<a[i])

                sum++;

        ans+=sum\*factorial[n-i];//累积

        sum=0;//计数器归零

    }

**return** ans+1;

}

**int** main(){

**int** sb[12],gs;

    cin>>gs;

**for**(**int** i=1;i<=gs;i++)

        cin>>sb[i];

    cout<<cantor(sb,gs);//输出该集合在全排列所在位置

**return** 0;

}

### 逆康托展开举例

一开始已经提过了，康托展开是一个全排列到一个自然数的双射，因此是可逆的。即对于上述例子，在

IMG_268

给出61可以算出起排列组合为34152。由上述的计算过程可以容易的逆推回来，具体过程如下：

用 61 / 4! = 2余13，说明

IMG_269

,说明比首位小的数有2个，所以首位为3。

用 13 / 3! = 2余1，说明

IMG_270

，说明在第二位之后小于第二位的数有2个，所以第二位为4。

用 1 / 2! = 0余1，说明

IMG_271

，说明在第三位之后没有小于第三位的数，所以第三位为1。

用 1 / 1! = 1余0，说明

IMG_272

，说明在第二位之后小于第四位的数有1个，所以第四位为5。

最后一位自然就是剩下的数2。

通过以上分析，所求排列组合为 34152。

### 代码实现

**static** **const** **int** FAC[] = {1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880};   // 阶乘

//康托展开逆运算

**void** decantor(**int** x, **int** n)

{

    vector<**int**> v;  // 存放当前可选数

    vector<**int**> a;  // 所求排列组合

**for**(**int** i=1;i<=n;i++)

        v.push\_back(i);

**for**(**int** i=n;i>=1;i--)

    {

**int** r = x % FAC[i-1];

**int** t = x / FAC[i-1];

        x = r;

        sort(v.begin(),v.end());// 从小到大排序

        a.push\_back(v[t]);      // 剩余数里第t+1个数为当前位

        v.erase(v.begin()+t);   // 移除选做当前位的数

    }

}

#include<iostream>

#include<cmath>//pow的头文件

**using** **namespace** std;

**const** **int** factorial[]={1,1,2,6,24,120,720,5040,40320,362880,3628800};//阶乘0-9

**int** gs,rank;//gs表示gs位的全排列，rank排列位置

**bool** used[11];//判断是否用过

**int** decantor(**int** x,**int** gs){//逆cantor展开，x就是rank

**int** ans=0;//存放答案

**int** sum=0;//暂时的计数

**int** quotient,remainder;//quotient商，remainder余数

**for**(**int** i=gs-1;i>=1;i--){

        quotient=x/factorial[i];

        remainder=x%factorial[i];

**for**(**int** j=1;j<=gs;j++){

**if**(!used[j])

                sum++;

**if**(sum==quotient+1){//找到该位

                ans+=j\***pow**(10,i);//pow幂运算

                sum=0;//清零

                x=remainder;

                used[j]=**true**;//标记为用过

**break**;

            }

        }

    }

**for**(**int** i=1;i<=gs;i++){

**if**(!used[i]){

            ans+=i;

**break**;

        }

    }//最后一位

**return** ans;//答案

}//逆推过程

**int** main(){//（signed main也可以）

**for**(**int** i=1;i<=10;i++)

        used[11]=**false**;//初始化为未用过

    cin>>gs;

    cin>>rank;

    rank--;//原rank前的个数

    cout<<decantor(rank,gs);//输出该排列

**return** 0;//建议加上

}

## 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逆推出唯一的一个排列。