一、排列与组合的基本概念

1．两个原理：

(1)**加法原理（分类计数原理）：**完成一件事，有n类办法，如果在第 1 类办法中有m_1种不同的方法，在第 2 类办法中有m_2种不同的方法，…，在第n类办法中有m_n种不同的方法，那么完成这件事共有：$N=m_1+m_2+\cdots\cdots+m_n$种不同的方法。

(2)**乘法原理（分步计数原理）：**完成一件事，需要分成n个步骤，如果做第 1 步有m_1种不同的方法，做第 2 步有m_2种不同的方法，……，做第n步有m_n种不同的方法，那么完成这件事共有：$N=m_1\times m_2\times\cdots\times m_n$种不同的方法。

(3)**两个原理的区别：**一个与分类有关，一个与分步有关；加法原理是“分类完成”，乘法原理是“分步完成”。

例题一：

从北京到天津火车有 10 个车次，汽车有 12 个班次，飞机有 2 个航班，从天津到上海火车有 10 个车次，汽车有 8 个班次，飞机有 8 个航班，轮船有 2 个班次，

(1)问：从北京到天津有多少种不同的到达方法？24

(2)问：从北京经天津到上海有多少种不同的到达方法。 672

2．排列与排列数

(1)**排列：**从 n 个不同的元素中取出 m(m≤n)个元素，按照一定的顺序排成一列，叫做从 n 个不同的元素中取出 m 个元素的一个排列。相同的排列是指元素相同且顺序相同。

(2)**排列数：**从 n 个不同的元素中取出 m(m≤n)个元素的所有排列的个数，叫做从 n 个不同的元素中取出 m 个元素的排列数，用符号A_n^m表示。

排列数公式：$A_n^m=n(n-1)(n-2)\cdots(n-m+1)(n,m\in N^*)$ =n!/(n-m)!

(3)**全排列：**把 n 个不同的元素全部取出(从 n 个不同的元素中取出 n 个元素)，按照一定的顺序排成一列，叫做 n 个不同的元素的一个全排列，全排列的个数叫做 n 个元素的全排列数，用符号A_n^n表示。此时，A_n^n=n(n-1)(n-2)……3·2·1=n！n!表示正整数 1 到 n 的连乘，叫做 n 的阶乘。

因此：$A_n^m=\dfrac{n!}{(n-m)!}$，规定：0!＝1。

例题一：

7 人站在一排，

(1)甲站在中间的不同排法有_____种；

(2)甲、乙相邻的不同排法有_____种；

(3)甲、乙不相邻的不同排法有_____种；

(4)甲站在乙的左边的不同排法有_____种；

(5)甲不站在左端，乙不站在右端的不同排法有_____种。

解析：

求满足条件的排列数需要从特殊条件的元素入手，先排好特殊元素，对于没有要求的元素进行全排列即可。

(1)先排甲：$A_1^1\cdot A_6^6=720$(此时的中间指正中间)；

(2)先排甲、乙：$A_2^2\cdot A_6^6$＝1440(相邻的问题采用“捆绑”的方法，把甲、乙二人排好后看作一人，再与其他五人，共六人全排列)；

(3)先排甲、乙：$A_6^2\cdot A_5^5$＝3600(不相邻的问题采用插空的方法，没有要求的五个人排好后出现六个空，甲、乙二人站在其中的两个空中)；

(4)由于七个人站好以后，甲在乙的左边，与甲在乙的右边的情况是一样的，因此满足条件的不同排法为：$\frac{1}{2}A_7^7$＝2520 种；

(5)由于甲站不站在右端对乙有影响，因此满足条件的站法被分为两类：甲站右端，甲不站右端，甲站右端：A_6^6=720；甲不站右端：$A_5^1\cdot A_5^1\ A_5^5$=3000，共有 3720 种不同的站法。

也可：$A_7^7-A_6^6-A_6^6+A_5^5$＝3720(用七个人的全排列减去甲在左端，再减去乙在右端，再加上甲在左端且乙在右端)。

3．组合与组合数：

1．组合：从 n 个不同的元素中取出 m(m≤n)个元素并成一组，叫做从 n 个不同的元素中取出 m 个元素的一个组合。

2．组合数：从 n 个不同的元素中取出 m(m≤n)个元素的所有组合的个数，叫做从 n 个不同的元素中取出 m 个元素的组合数，用符号 C_n^m 表示。根据分步计数原理得到：$A_n^m = C_n^m \cdot A_m^m$。

组合数公式：$C_n^m = \frac{A_n^m}{A_m^m} = \frac{n(n-1)(n-2)\cdots(n-m+1)}{m!} = \frac{n!}{m!(n-m)!}(n,m \in N^*)$

4．组合数的性质：

(1) $C_n^m = C_n^{n-m}$，规定：$C_n^0 = 1$；(2) $C_{n+1}^m = C_n^m + C_n^{m-1}$（从 n+1 个中取出一个 X 余下 n 个，X 不放入 m 中则为 C（n,m）,它放入 m 中则为 c(n,m-1) ）。

例题分析：

例 1．6 本不同的书，

(1)分成三堆，一堆一本，一堆两本，一堆三本，有_____分法；

(2)分给甲、乙、丙三人，一人一本，一人两本，一人三本，有_____ 分法；

(3)分成三堆，每堆两本，有_____分法；

(4)分给甲、乙、丙三人，每人两本，有_____ 分法。

解：(1)三堆书的本数各不相同：$C_6^1 \cdot C_5^2 \cdot C_3^3$=60 种(分组，没有顺序)；

(2)相当于(1)中三堆书再分给三个人：$C_6^1 \cdot C_5^2 \cdot C_3^3 \cdot A_3^3$=360 种；

(3)三堆书的本数相同(平均分组的问题)：$\frac{C_6^2 \cdot C_4^2 \cdot C_2^2}{A_3^3}$ =15 种；

(4)相当于(3)中三堆书再分给三个人：$\frac{C_6^2\ C_4^2 \cdot C_2^2}{A_3^3} \cdot A_3^3 = C_6^2 \cdot C_4^2 \cdot C_2^2 = 90$。

二、排列及其生成算法

（一）、几种排列

1、相异元素的不重复排列

从 n 个相异元素中取出 m（m<=n）个不同元素，按一定顺序排成一列：当 m<n 时叫选排列，当 m=n 的时候，叫全排列。计算公式：

$$P_n^m = n(n-1)(n-2)\cdots(n-m+1) = \frac{n!}{(n-m)!}$$

【例 1】从 1 2 3 4 这 4 个相异元素中取出 2 个的排列数是 $P_4^2 = 4 \times 3 = 12$；这 12 种排列

12　13　14　21　23　24　31　32　34　41　42　43

2、相异元素的可重复排列

从 n 个相异元素中，可以重复地取出 m 个元素的排列叫着相异元素的可重复选排列，起排列总数为 n^m

【例 2】从 1 2 3 4 这 4 个相异元素中可重复取出 2 个的排列数是 $4^2 = 4 \times 4 = 16$；这 12 种排列为：

11　12　13　14　21　22　23　24　31　32　33　34　41　42　43　44

3、不全相异元素的排列

如果在n个元素中，有n_1个元素彼此相同，有n_2个元素彼此相同，……，有n_m个元素彼此相同，并且$n_1+n_2+\cdots n_m=n$，则这n个元素的全排列叫作不全相异元素的全排列，其排列数为：

$$\frac{n!}{(n_1! \times n_2! \cdots \times n_m!)}$$

如果$n_1+n_2+\cdots n_m=r<n$，则这r个元素的排列叫作不全相异元素的选排列；$\frac{P_n^r}{(n_1!n_2!\cdots n_m!)\cdot}$

【例 3】把 2 个红色球、1 个兰色求和 3 个白色球放到 10 个编号不同的盒子中去，有多少种方法？

解：本题属于不全相异元素的选排列问题的典型模型，此处 n=10，r=6，n1=2，n2=1；n3=3，所以，方法总数是：$\frac{P_{10}^6}{(2! \times 3!)\cdot} = 12600$

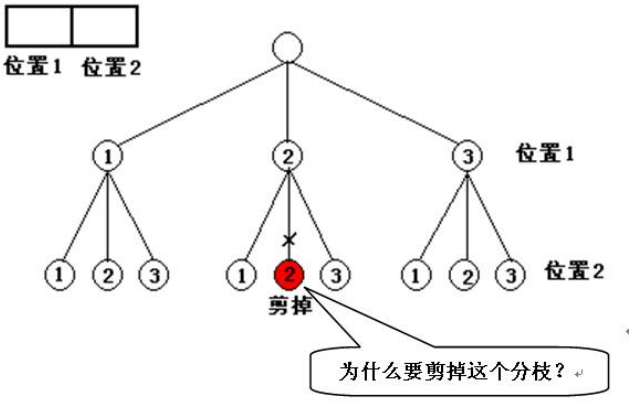
三、排列的回溯算法

信息学中对于排列的运用，不会太多关心排列数的问题，而是要求利用计算机的快速特性，列举出每种符合条件的排列，即排列的生成问题。

我们通过前面的讲述知道，“相异元素的有限重复排列”是一个具有普遍性的数学模型，所以这种排列的生成算法也就是很多搜索题目的原始数学模型。下面我们来研究排列生成问题：

排列生成算法的过程就是用深度搜索模拟我们手工书写所有排列的过程，我们知道：任何深度搜索模型都可以用“解答树”来描述：

比如从 1 1 2 3 3 这 5 个数字中选择 2 个的排列，其解答树为：



有了上面的解答树，我们只须深度遍历这棵树就可以得到所有排列的方案。

数据结构定义： int used[10],a[100],n,m;//n 个相异元素中有限重复地取 m 个的排列

比如从 1 1 2 3 3 这 5 个数字中选择 2 个的排列：

参数：dep 表示树的深度；

used[c]表示元素 c 还可以使用多少次；

程序代码：

```
#include<iostream>
using namespace std;
int used[20],a[100],m,n,k,i;
void print()
{  int j;
   for(j=1;j<=m;j++)cout<<a[j]<<" ";
   cout<<endl;
}
void f(int dep)
{  int i;
   if(dep>m){print();return;}
   for(i=0;i<=n;i++)
     if(used[i]>0)
      {
         a[dep]=i;
         used[i]--;
         f(dep+1);
         used[i]++;
      }
}
int main()
{  cin>>n>>m;
   memset(used,0,sizeof(used));
   for(i=1;i<=n;i++)
     {cin>>k;used[k]++;}
   f(1);
   system("pause");
```

```
   return 0;
}
```

（三）由当前排列生成下一个排列

当 N=4 时的排列：

1234 1243 1324 1342 1423 1432
2134 2143 2314 2341 2413 2431
3124 3142 3214 3241 3412 3421
4123 4132 4213 4231 4312 4321

从上面的排列可以发现以下的规律：

（1）偶数序列是由奇数序列的最后两位交换而来，如 1234 的最后两位交换后为 1243，4123 的最后两位交换后为 4132；

（2）除第一个排列 1234 是给定的外，其它奇数排列也可由前一个偶数序列通过以下的交换得来：对前一个序列从第一个数字开始找出比左边数大的最右数，如 1243 为 4，这个数所在的位置记为 i，1243 的 i=3，然后再从第 i 个数开始找比第 i-1 个数大的最右数，1243 为 3，它的位置记为 j，1243 的 j=4，交换 a[i-1]与 a[j]，此时 1243 变为 1342，跟 1324 不同的是从第 i 位直到第 n 位顺序相反，因此对第 i 位到第 n 位进行逆序处理则得到结果；

（3）将交换规律（2）同样对奇数序列作处理，也能得到正确的偶数序列，结果一样，因此可以统一成一个交换规律，即将（1）和（2）统一成（2）；

交换到什么时候结束呢？当已是完全由大到小时（i=1），则完成了交换构造。

算法描述：

（1）1、2……N 依次赋给 a[1]至 a[n]，输出第一种排列；

（2）构造下一种全排列，分四步完成：

第一步，i 的初值为 1，在 a[1]至 a[n]中搜索找出相应的 i，使 i 是 a[k]>a[k-1]的 k 中最大的，即 i=max{k|a[k]>a[k-1],k=2,3…n}；

第二步，在 a[1]至 a[n]中搜索找出相应的 j，使 j 是 a[k]>a[i-1]的 k 中最大的，即 j=max{k|a[k]>a[i-1],k=i,i+1…n}；

第三步，交换 a[i-1]与 a[j]形成新的序列；

第四步，对新的序列从 a[i+1]……a[n]进行逆序处理，输出相应序列。

（3）重复（2）直到 i=1 时结束

程序代码：

方法 1：

```
#include <iostream>
using namespace std;
int a[100],i,j,n,temp,k,ll;
void print()
{
   for(i=1;i<=n;i++)cout<<a[i]<<" ";
   cout<<endl;
}
int main()
{  cin>>n;
   for(i=1;i<=n;i++)cin>>a[i];
   i=n-1;
   while(i>0&&a[i]>a[i+1])i--;
   if(i>0)
    {
      j=n;
      while(a[j]<a[i])j--;
      temp=a[i];a[i]=a[j];a[j]=temp;
      k=i+1;ll=n;
      while(k<ll)
      {
        temp=a[k];a[k]=a[ll];a[ll]=temp;
```

```
        k++;ll--;
      }
    }
    print();
    return 0;
}
```

方法 2：

可以基于回溯来找一个，设置一个栈来模拟一次回溯的过程，下例中 s[]表示栈，top 跟踪栈顶，used[]表明该数是否用过。

```
void getnext()
{int i;
//初始化
 cin>>n;
 for(int i=1;i<=n;i++){cin>>s[i];used[s[i]]=1;}
 int top=n;// 栈顶
 while(top>0&&top<=n)// 栈顶不超界
 {
    used[s[top]]=0;//取出栈顶元素,设为可用
    //从其下一个开始换
   for (i=s[top]+1 ;i<=n;i++)
    if(used[i])//若可用
 {used[i]=1;s[top]=i;top++;break;}//替换栈顶,前进
 if(i>n) {s[top]=0;top--;}//无可用元素后退
 }
}
```

例题二：

【1636】:

输入：

两个自然数 m,n 1<=n<=18，1<=m<=n!

输出：

n 个数的第 m 种全排列。

样例输入：

3 2

样例输出：

1 3 2

程序代码：

```
#include<iostream>
using namespace std;
long long a[21]＝{0};
int n,m,c[21],d[21]＝{0};
void out()//输出
{
   bool b[21]＝{0};//使用标志
   int  ka;
  for(int i=1;i<=n;i++)//确定每一位的数字
      {
        //排除用过的,根据 d[i]确定数字
```

```
        ka=1;
       for(int j=1;j<=d[i];j++)
        {while(b[ka]==true) ka++; //已用的不算
         ka++;}
       b[--ka]=true;//设为已用
       cout<<ka<<" ";
       }
    return ;
}
int main()
{
    cin>>n>>m;
    m--;//从 0 开始计数
    int t=1;
    for(int i=1;i<=n;i++){ t*=i;a[i]=t;}//算阶乘
    for(int i=1;i<n;i++)//算出第 i 位
     {
    d[i]=m/a[n-i]+1;//计算每位包含的阶乘数+1
      m=m%a[n-i];//余下的数
      }
      d[n]=1;//最后一位必为 0  +1
     out();
}
```

说明：

3421 的排列位置：首位 3,由于 3 前有 2 个数，则由 3 开始的前面排列数为 2*3!,对于第 2 位 4,因为 3 已用，则 4 是 421 中的第 3 大的数，所以前面有 2*2！，依次类推..3421=2*3!+2*2!+1*1!+0*0!=17,由于把 1234=0*3!+0*2!+0*1!+0＝0 视为第 1 个，所以最后加 1=18.本题就是该过程的逆过程。

M=18-1=17
17/3!=2 2+1=3 输出 3（第 3 小） m=17%3!=5
5/2!=2 2+1=3 输出 4 (3 已输出) m=5%2!=1
1/1!=1 1+1=2 输出 2 m=1%1!=0
0/0!=0 0+1=1 输出 1

【2004 普及】火星人 1057

题目描述:

人类终于登上了火星的土地并且见到了神秘的火星人。人类和火星人都无法理解对方的语言，但是我们的科学家发明了一种用数字交流的方法。这种交流方法是这样的，首先，火星人把一个非常大的数字告诉人类科学家，科学家破解这个数字的含义后，再把一个很小的数字加到这个大数上面，把结果告诉火星人，作为人类的回答。

火星人用一种非常简单的方式来表示数字——掰手指。火星人只有一只手，但这只手上有成千上万的手指，这些手指排成一列，分别编号为 1，2，3……。火星人的任意两根手指都能随意交换位置，他们就是通过这方法计数的。

一个火星人用一个人类的手演示了如何用手指计数。如果把五根手指—拇指、食指、中指、无名指和小指分别编号为 1，2，3，4 和 5，当它们按正常顺序排列时，形成了 5 位数 12345，当你交换无名指和小指的位置时，会形成 5 位数 12354，当你把五个手指的顺序完全颠倒时，会形成 54321，在所有能够形成的 120 个 5 位数中，12345 最小，它表示 1；12354 第二小，它表示 2；54321 最大，它表示 120。下表展示了只有 3 根手指时能够形成的 6 个 3 位数和它们代表的数字：

三进制数	123	132	213	231	312	321
代表的数字	1	2	3	4	5	6

现在你有幸成为了第一个和火星人交流的地球人。一个火星人会让你看他的手指，科学家会告诉你要加上去的很小的数。你的任务是，把火星人用手指表示的数与科学家告诉你的数相加，并根据相加的结果改变火星人手指的排列顺序。输入数据保证这个结果不会超出火星人手指能表示的范围。

输入:

输入包括三行，第一行有一个正整数 N，表示火星人手指的数目（1 <= N <= 10000）。第二行是一个正整数 M，表示要加上去的小整数（1 <= M <= 100）。下一行是 1 到 N 这 N 个整数的一个排列，用空格隔开，表示火星人手指的排列顺序。

输出:

输出只有一行，这一行含有 N 个整数，表示改变后的火星人手指的排列顺序。每两个相邻的数中间用一个空格分开，不能有多余的空格。

样例输入：

5
3
1 2 3 4 5

样例输出：

1 2 4 5 3

数据范围:

对于 30%的数据，N<=15；
对于 60%的数据，N<=50；
对于全部的数据，N<=10000；

步骤：

根据前一个排列产生下一个排列

四、组合及其生成算法

（一） 用回溯算法

```
#include <iostream>
using namespace std;
int a[101]={0},i,j,n,m,dep;
void print()
{  for(i=1;i<=m;i++)cout<<a[i]<<" ";
   cout<<endl;
}
void solve(int dep)
{ int i;
  if(dep==m+1){print();return;}
  for(i=a[dep-1]+1;i<=n-(m-dep);i++)
    {
       a[dep]=i;
       solve(dep+1);
    }
}
int main()
{
   cin>>n>>m;a[0]=0;
   solve(1);
   return 0;
}
//按升序生成
```

（二） 由当前组合产生下一个组合算法

步骤：根据前一个组合产生下一个组合

1.从后向前找第一个未达到当前位置最大值的数 a[i]；

2.a[i]=a[i]+1；

3.后面的数都等于前面数加 1，即 a[i]=a[i-1]+1；

4.判断 a[i]，若达到当前位的最大值则结束，否则转为 1；

```
#include <iostream>
using namespace std;
int a[101],i,j,n,m;
void print()
{
   for(i=1;i<=m;i++)cout<<a[i]<<" ";
   cout<<endl;
}
int main()
{
   cin>>n>>m;
   for(i=1;i<=m;i++)cin>>a[i];
   i=m;
   while(i>0&&a[i]==n-(m-i))i--;
   if(i>0)
    {
      a[i]++;
      for(j=i+1;j<=m;j++)a[j]=a[j-1]+1;
    }
   print();
   return 0;
}
```

无重复元素的组合

【练习试题】组合 1588

题目描述：

输入一串小些字母（无重复字母），从中取出 k 个字母，输出组合情况。

输入：

```
abcd
3
```

输出：

```
abc
abd
acd
 bcd
```

程序代码：

```
#include<iostream>
#include<cstring>
using namespace std;
string s;
int a[100]={0},n,m;
int  f(int k)
{  int i,j;
   if(k==m+1)
    {
       for(j=1;j<k;j++)cout<<s[a[j]];
       cout<<endl;
```

```
       return 0;
    }
   for(i=a[k-1]+1;i<=n-m+k;i++)
    {
      a[k]=i;
      f(k+1);
    }
}
int main()
{  cin>>s>>m;a[1]=0;
   n=s.length();
   s='0'+s;
   f(1);
   return 0;
}
```

有重复元素的组合

【练习试题】有重复元素的组合 1589

题目描述：

输入一串小些字母（小于 30 个字母，有重复字母），从中取出 k 个字母，输出组合情况。

输入：

```
aabbcc
4
```

输出：

```
1:aabb
2:aabc
3:aacc
4:abbc
5:abcc
6:bbcc
```

程序代码:

```
#include<iostream>
using namespace std;
int a[26],k,c=0;char b[31]={'a'};
void out()
{   c++;
    cout<<c<<":";
    for(int i=1;i<=k;i++)cout<<b[i];
    cout<<endl;
}
void f(int t)
{  int i;
   if(t==k+1){out();return;}
   for(i=b[t-1]-'a';i<=25;i++)
    if(a[i])
     { a[i]--;
       b[t]=i+'a';
       f(t+1);
       a[i]++;
    }
}
int main()
{  char s[31];
```

```
   cin>>s>>k;
   memset(a,0,sizeof(a));
   for(int i=0;s[i]!='\0';i++)a[s[i]-'a']++;
   f(1);
}
```

【2006 年普及组 3】Jam 的计数法 1509

题目描述：

Jam 是个喜欢标新立异的科学怪人。他不使用阿拉伯数字计数，而是使用小写英文字母计数，他觉得这样做，会使世界更加丰富多彩。在他的计数法中，每个数字的位数都是相同的（使用相同个数的字母），英文字母按原先的顺序，排在前面的字母小于排在它后面的字母。我们把这样的“数字”称为 Jam 数字。在 Jam 数字中，每个字母互不相同，而且从左到右是严格递增的。每次，Jam 还指定使用字母的范围，例如，从 2 到 10，表示只能使用{b,c,d,e,f,g,h,i,j}这些字母。如果再规定位数为 5，那么，紧接在 Jam 数字“bdfij”之后的数字应该是“bdghi”。（如果我们用 U、V 依次表示 Jam 数字“bdfij”与“bdghi”，则 U < V，且不存在 Jam 数字 P，使 U < P < V）。你的任务是：对于从文件读入的一个 Jam 数字，按顺序输出紧接在后面的 5 个 Jam 数字，如果后面没有那么多 Jam 数字，那么有几个就输出几个。

输入：

输入文件 counting.in 有 2 行，第 1 行为 3 个正整数，用一个空格隔开： s t w （其中 s 为所使用的最小的字母的序号，t 为所使用的最大的字母的序号。w 为数字的位数，这 3 个数满足：1≤s

第 2 行为具有 w 个小写字母的字符串，为一个符合要求的 Jam 数字。

所给的数据都是正确的，不必验证。

输出：

输出文件 counting.out 最多为 5 行，为紧接在输入的 Jam 数字后面的 5 个 Jam 数字，如果后面没有那么多 Jam 数字，那么有几个就输出几个。每行只输出一个 Jam 数字，是由 w 个小写字母组成的字符串，不要有多余的空格。

样例输入：

```
2 10 5
bdfij
```

Sample Output

```
bdghi
bdghj
bdgij
bdhij
befgh
```

步骤：根据前一个组合产生下一个组合

1.从后向前找第一个未达到当前位置最大值的数 a[i]；

2.a[i]=a[i]+1；

3.后面的数都等于前面数加 1，即 a[i]=a[i-1]+1；

4.判断 a[i]，若达到当前位的最大值则结束，否则转为 1；

```
#include <iostream>
using namespace std;
int main()
{  char c;
   int a[27],j,i,w,s,t;
   cin>>s>>t>>w;
   for(i=0;i<w;i++) {cin>>c;a[i]=c-'a'+1;}
   for(i=1;i<=5;i++)
     {  for(j=w-1;j>=0;j--) if(a[j]!=t-(w-j-1)) break;
        if(j==-1) break;
        a[j]++;
        for(j=j+1;j<w;j++)a[j]=a[j-1]+1;
        for(j=0;j<w;j++)cout<<char(a[j]-1+'a');
```

```
        cout<<endl;
     }
   return 0;
}
```

对计算高精度组合数的优化

【组合定义】：从 n 个不同元素中，选取 m(1<=m<=n)个元素而不考虑先后次序，叫做从 n 个不同元素中取出 m 个元素的组合；用符号 c(n,m) 表示。

【计算公式】：$C_N^M=\frac{N!}{M!(N-M)!}$

【问题描述】：已知 n(n<=10^40)和 m(0<m<=100000)，求 c(n,m)的值。

【问题分析】：

显然，我们可以直接地求解，但是由于题目中的 N<=10^40 很大，所以我们必须要用到高精度计算。而在高精度计算中，运行的时间与参与运算的数的大小有直接的关系。所以，我们要使运算的中间结果尽可能地小。如果我们先把 N~(N-M+1)这 M 个连续的自然数乘起来，再依次除以 1~M，这样既费时间也浪费空间，是一种不太明智的选择，需要进一步优化求解。

优化：我们可以先乘 N 除 1，然后乘(N-1)除 2，再乘(N-2)除 3，……最后乘(N-M+1)除 M。因为连续的 K 个自然数的积一定能被 K!整除，所以在这一过程中不会出现除不尽的情况。同时也使得中间结果比较小，从而提高了程序运行的速度。

```
a[]←1;
for(i=1;i<=m;i++)
 {
   a[]←a[]*n; //高精度乘以高精度
   a[]←a[]/i;  //高精度除以低精度
   n[]←n[]-1; //改变高精度 n 的值
}
```

【模拟试题】彩票 1640

【问题描述】现今,社会上流行着各种各样的福利彩票,彩票已经融入到了人们的日常生活之中。彩票之所以能吸引那么多的人们,玩法多是一大原因。其中有一类是从前 N 个自然数中选出 M 个(不计顺序)不同的号码,如果这 M 个号码与摇奖时摇出的 M 个中奖号码完全相符,那么就中了头奖。如现在已经有的:30 选 7,35 选 7,36 选 7,37 选 7……

随着时间的推移,越来越多的人不满足于原来的玩法。为了追求更大的刺激,可供选择的号码和每注的号码个数越来越大,88 选 8,518 选 18,8888 选 68 等等应运而生。但是,由此也衍生出了许多麻烦。由于数字越来越大,福彩中心的工作人员们已经无法用一般的计数器精确地计算出每一种彩票中头奖的概率。现在请你帮助他们,编一个程序:对于每一种玩法,能够快速准确地计算出中头奖概率的倒数。

【文件输入】文件输入 n (N<10^40)和 m (0< M<= 1000)

【文件输出】文件输出在“N 选 M”的玩法中,中头奖的概率的倒数

【样例输入】

```
5
2
```

【样例输出】

```
10
```

【题目分析】

很显然本体求解的是：从 N 个数中（不计顺序）取出 M 个不同的数的取法共有 C(N,M)种。这里 C(N,M)表示组合数C_N^M。因此,要使摇出的中奖号码与所选的号码完全相同，概率只有 1/C(N,M)。所以我们要求的值即为 C(N,M)。根据组合数的计算公式：

$$C_N^M=\frac{N!}{M!(N-M)!}=\frac{N\times(N-1)\times\cdots\times(N-M+1)}{M\times(M-1)\times\cdots\times1}$$

这时，很显然知道可以用上面所讲的优化方法求解。

【程序代码】:

```
#include<iostream>
using namespace std;
int a[100000]={0},b[100000]={0},n[100000]={0},i,m;
void init()
{ int i,j;
  string s;
  cin>>s>>m;  //输入数串 s
  memset(n,0,sizeof(n));
  n[0]=s.length();  //取得数串 s 的长度
  for(i=0;i<n[0];i++)  //把 s 对应的整数保存到数组 a 中
    {
     j=(n[0]-i+3)/4;
     n[j]=n[j]*10+s[i]-'0';
    }
   n[0]=(n[0]+3)/4;
}
void print(int p[])
{
  cout<<p[p[0]];//输出最高位
  for(i=p[0]-1;i>0;i--)
   {
     cout<<p[i]/1000;
     cout<<p[i]/100%10;
     cout<<p[i]/10%10;
     cout<<a[i]%10;
    }
  cout<<endl;
  return ;
}
void chenggao(int a[],int b[],int c[])
{ int i,j;
  for (i=1;i<=a[0];i++)
    for (j=1;j<=b[0];j++) c[i+j-1]+=a[i]*b[j];
  c[0]=a[0]+b[0];
  for(i=1;i<=c[0];i++)  //处理进位
   {c[i+1]+=c[i]/10000;
    c[i]%=10000;
   }
  while(c[0]>0&&c[c[0]]==0)c[0]--;
  return ;
}
void chudan(int a[],int b,int c[])//c=a/b,d=a%b
{
   int i,d;
   d=0;  //余数初始化
   for(i=a[0];i>=1;i--)//按照由高位到底位的顺序,逐位相除
   {d=d*10000+a[i]; //接受了来自第 i+1 位的余数
   c[i]=d/b; //计算商的第 i 位
   d=d%b;}//计算第 i 位的余数
   c[0]=a[0];
   while(c[0]>0&&c[c[0]]==0)c[0]--;//计算商的有效位数
   return ;
```

```
}
int main()
{
  int j,k;
  init();
  a[0]=a[1]=1;
  for(i=1;i<=m;i++)
   {
     memset(b,0,sizeof(b));
     chenggao(a,n,b);
     memset(a,0,sizeof(a));
     chudan(b,i,a);
     j=1;
     while(n[j]==0)j++;
     n[j]--;
     for(k=1;k<=j-1;k++)n[k]=9999;
     if(j==n[0]&&n[j]==0)n[0]--;
   }
  print(a);
  return 0;
}
```