CHAPTER 3

第3章 穷 举 法

穷举法也称为枚举法、暴力法或蛮力法，它是算法设计中最常用的方法之一。穷举法的基本思想是对问题的所有可能状态一一测试，直到找到解或将全都可能状态都测试为止。本章介绍采用穷举法设计思想和相关示例。

3.1 穷举法概述

穷举法是一种简单直接地解决问题的方法，通常直接基于问题的描述和所涉及的概念定义，找出所有可能的解，然后选择其中的一种或多种解，若该解不可行则试探下一种可能的解。

穷举法是基于计算机运算速度快这一特性，在解决问题时采取的一种“懒惰”策略。这种策略不经过（或者说经过很少）思考，把问题所有情况或所有过程交给计算机去一一尝试，从中找出问题的解。

穷举法的优点如下：

- 逻辑清晰，编写程序简洁。
- 可以用来解决广阔领域的问题。
- 对于一些重要的问题，它可以产生一些合理的算法。
- 可以解决一些小规模的问题。
- 可以作为其他高效算法的衡量标准。

穷举法的主要缺点是设计的大多数算法的效率都不高。

使用穷举法通常有如下几种情况。

（1）搜索所有的解空间：问题的解存在于规模不大的解空间中。解决这类问题一般是要求找出某些特定的解，这些解满足某些特征或要求。使用穷举搜索的方法就是把所有可能的解都列出来，看这些解是否满足特定的条件或要求，从中选出符合要求的解。

（2）搜索所有的路径：这类问题中不同的路径对应不同的解，需要找出特定解。采用穷举搜索就是把所有可能的路径都搜索一遍，计算出所有路径对应的解，找出特定解。

(3) 直接计算：按照基于问题的描述和所涉及的概念定义，直接进行计算。往往是一些简单的题，不需要算法技巧。

(4) 模拟和仿真：按照求解问题的要求直接模拟或仿真即可。

从算法实现的角度看，采用穷举法设计算法分为两类，一类是采用基本穷举法，即直接采用穷举思想设计算法，另一类是在穷举中应用递归，即采用递归方法穷举搜索解空间，前者相对直接简单，后者需要结合递归算法设计方法，相对复杂些。

3.2 穷举法的基本应用

3.2.1 直接采用穷举法的一般格式

在直接采用穷举法设计算法中，主要是使用循环语句和选择语句，循环语句用于穷举所有可能的情况，而选择语句则判定当前的条件是否为所求的解。其基本格式如下：

```
for (循环变量 x 取所有可能的值)
{   ⋮
    if (x 满足指定的条件)
        输出 x;
    ⋮
}
```

【例 3.1】 编写一个程序，输出 2～1000 之间的所有完全数。所谓完全数，是指这样的数，该数的各因子(除该数本身外)之和正好等于该数本身，例如：

```
6 = 1 + 2 + 3
28 = 1 + 2 + 4 + 7 + 14
```

解：先考虑对于一个整数 m，如何判断它是否为完全数。从数学知识可知：一个数 m 的除该数本身外的所有因子都在 $1\sim m/2$ 之间。算法中要取得因子之和，只要在 $1\sim m/2$ 之间找到所有整除 m 的数，将其累加起来即可。如果累加和与 m 本身相等，则表示 m 是一个完全数，可以将 m 输出。其循环格式为：

```
for (m = 2;m <= 1000;m++)
{   求出 m 的所有因子之和 s;
    if (m == s) 输出 s;
}
```

对应的程序如下：

```
#include <stdio.h>
void main()
{   int m,i,s;
    for (m = 2;m <= 1000;m++)
    {    s = 0;
        for (i = 1;i <= m/2;i++)
            if (m % i == 0) s += i;                    //i 是 m 的一个因子
        if (m == s)
            printf("%d ",m);
```

```
    }
    printf("\n");
}
```

本程序的执行结果如下：

```
6 28 496
```

【例 3.2】 编写一个程序，求这样 4 位数：该 4 位数的千位上的数字和百位上的数字都被擦掉了，知道十位上的数字是 1，个位上的数字是 2，又知道这个数如果减去 7 就能被 7 整除，减去 8 就能被 8 整除，减去 9 就能被 9 整除。

解：设这个数为 $ab12$，则 $n=1000\times a+100\times b+10+2$，且有 $0\leqslant a\leqslant 9$，$0\leqslant b\leqslant 9$。采用穷举法求解，其循环格式为：

```
for (a = 0;a <= 9;a++)
    for (b = 0;b <= 9;b++)
    {    n = 1000 * a + 100 * b + 10 + 2;
         if (n 满足题中给定条件) 输出 n;
    }
```

对应的程序如下：

```
#include <stdio.h>
void main()
{   int n,a,b;
    for (a = 0;a <= 9;a++)
        for (b = 0;b <= 9;b++)
        {    n = 1000 * a + 100 * b + 10 + 2;
             if ((n - 7) % 7 == 0 && (n - 8) % 8 == 0 && (n - 9) % 9 == 0)
             {   printf("n = %d\n",n);
                 break;
             }
        }
}
```

本程序的执行结果如下：

```
n = 1512
```

【例 3.3】 在象棋算式里，不同的棋子代表不同的数，有如图 3.1 所示的算式，设计一个算法求这些棋子各代表哪些数字。

```
    兵炮马卒
+   兵炮车卒
------------
  车卒马兵卒
```

图 3.1　象棋算式

解：采用逻辑推理时，先从“卒”入手，卒和卒相加，和的个位数仍是卒，这个数只能是 0，确定卒是 0 后，所有是卒的地方，都为 0。这时，会看到“兵＋兵＝车 0”，从而得到兵为 5，车是 1。进一步得到“马＋1＝5”，所以，马＝4，又有“炮＋炮＝4”，从而，炮＝2。

最后的结果是：兵＝5，炮＝2，马＝4，卒＝0，车＝1。

采用穷举法时，设兵、炮、马、卒和车的取值分别为 a、b、c、d、e，则有：

a、b、c、d、e 的取值范围为 0～9 且均不相等（即 $a==b \| a==c \| a==d \| a==e \| b==c \| b==d \| b==e \| c==d \| c==e \| d==e$ 不成立）。

设：$m=a\times1000+b\times100+c\times10+d$

$n=a\times1000+b\times100+e\times10+d$

$s=e\times10000+d\times1000+c\times100+a\times10+d$

则满足的条件转换为：$m+n==s$。

对应的完整程序如下：

```
#include <stdio.h>
void fun()
{   int a,b,c,d,e,m,n,s;
    for (a=1;a<=9;a++)
        for (b=0;b<=9;b++)
            for (c=0;c<=9;c++)
                for (d=0;d<=9;d++)
                    for (e=0;e<=9;e++)
                        if (a==b||a==c||a==d||a==e||b==c||b==d
                                ||b==e||c==d||c==e||d==e)
                            continue;
                        else
                        {   m=a*1000+b*100+c*10+d;
                            n=a*1000+b*100+e*10+d;
                            s=e*10000+d*1000+c*100+a*10+d;
                            if (m+n==s)
                                printf("兵:%d 炮:%d 马:%d 卒:%d 车:%d\n",
                                    a,b,c,d,e);
                        }
}
void main()
{
    fun();
}
```

程序的输出结果如下：

```
兵:5 炮:2 马:4 卒:0 车:1
```

【例 3.4】 有 $n(n\geqslant4)$ 个正整数，存放在数组 a 中，设计一个算法从中选出 3 个正整数组成周长最长的三角形，输出该最长三角形的周长，若无法组成三角形则输出 0。

解：采用穷举法，用 i、j、k 3 重循环，让 $i<j<k$ 避免正整数被重复选中，设选中的 3 个正整数 $a[i]$、$a[j]$和 $a[k]$之和为 len，其中最大正整数为 ma，能组成三角形的条件是两边之和大于第 3 边，即 ma<len－ma。对应的完整程序如下：

```
#include <stdio.h>
int max3(int a,int b,int c)                    //求 3 个整数中的最大者
{   int d = a;
    if (d < b) d = b;
    if (d < c) d = c;
    return d;
}
void solve(int a[],int n)
{   int i,j,k,len,ma,maxlen = 0;
    for (i = 0;i < n;i++)
        for (j = i + 1;j < n;j++)
            for (k = j + 1;k < n;k++)
            {   len = a[i] + a[j] + a[k];
                ma = max3(a[i],a[j],a[k]);
                if (ma < len - ma)             //a[i]、a[j]、a[k]能组成一个三角形
                {    if (len > maxlen)         //比较求最长的周长
                        maxlen = len;
                }
            }
    if (maxlen > 0)
        printf("最长三角形的周长 = %d\n",maxlen);
    else
        printf("0\n");
}
void main()
{   int a[] = {4,5,8,20};
    int n = 4;
    solve(a,n);
}
```

程序的输出结果如下：

```
最长三角形的周长 = 17
```

3.2.2　简单选择排序和冒泡排序

问题描述：对于给定的含有 n 个元素的数组 a，对其按元素值递增排序。

第 2 章介绍过采用递归方法设计简单选择排序和冒泡排序算法，这里讨论采用穷举法设计这两种算法。

1. 简单选择排序

假设整型数组 a 中有 10 个元素，简单选择排序是将整个元素分为有序区和无序区，有序区的所有元素均小于无序区的元素，然后针对无序区的每个位置 $i(0 \leqslant i \leqslant n-2)$，从无序区挑选第 i 小的元素放在该位置，挑选过程采用穷举方法，用 k 记录无序区中最小元素的下标，依次通过无序区中所有元素的比较来实现，当 k 不等于 i 时，将 $a[i]$与 $a[k]$元素交换。

例如，图 3.2 为 $i=3$ 的一趟简单选择排序过程，其中 $a[0..2]$是有序的，从 $a[3..9]$中挑选最小元素 $a[5]$，将其与 $a[3]$进行交换，从而扩大有序区，减小无序区。

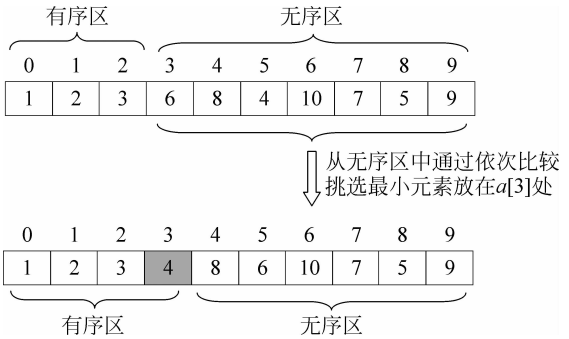

图 3.2　一趟简单选择排序过程

简单选择排序的完整程序如下：

```
#include <stdio.h>
void SelectSort(int a[],int n)              //对 a[0..n-1]元素进行递增简单选择排序
{   int i,j,k;
    int tmp;
    for (i=0;i<n-1;i++)                     //进行 n-1 趟排序
    {   k=i;                                //用 k 记录每趟无序区中最小元素的位置
        for (j=i+1;j<n;j++)                 //在 a[i+1..n-1]中采用穷举法找最小元素 a[k]
            if (a[j]<a[k])
                k=j;
        if (k!=i)                           //若 a[k]不是最小元素
        {    tmp=a[i];                      //将 a[k]与 a[i]交换
            a[i]=a[k]; a[k]=tmp;
        }
    }
}
void disp(int a[],int n)                    //输出 a 中所有元素
{   int i;
    for (i=0;i<n;i++)
        printf("%d",a[i]);
    printf("\n");
}
void main()
{   int n=10;
    int a[]={2,5,1,7,10,6,9,4,3,8};
    printf("排序前:"); disp(a,n);
    SelectSort(a,n);
    printf("排序后:"); disp(a,n);
}
```

在含有 n 个元素的数组 a 中采用穷举法找出最小元素，需要进行 $n-1$ 次比较，则在 $a[i+1..n-1]$中找最小元素 $a[k]$需要进行 $n-i-1$ 次比较，所以算法总的比较次数为 $\sum_{i=0}^{n-2}(n-i-1)=\frac{n(n-1)}{2}=O(n^2)$。简单选择排序的主要时间花费在元素比较上，所以该算法的时间复杂度为 $O(n^2)$。

2. 冒泡排序

冒泡排序也是将整个元素分为有序区和无序区，有序区的所有元素均小于无序区的元素，然后针对无序区的每个位置 $i(0\leqslant i\leqslant n-2)$，从无序区通过交换方式将第 i 小的元素放在该位置，交换过程也是采用穷举方法，从无序区尾部开始，当相邻的两个元素逆序时将两者交换。当某一趟没有元素交换时说明无序区已经有序了，所有元素均有序，算法结束。

例如，如图 3.3 所示是 $i=3$ 的一趟冒泡排序过程，其中 $a[0..2]$是有序的，从 $a[3..9]$中通过交换将最小元素放在 $a[5]$处，从而扩大有序区，减小无序区。

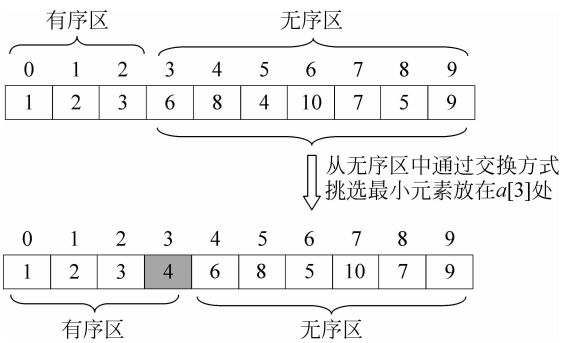

图 3.3　一趟冒泡排序过程

冒泡排序的完整程序如下：

```
#include <stdio.h>
void BubbleSort(int a[],int n)      //对 a[0..n-1]按递增有序进行冒泡排序
{   int i,j; int tmp;
    bool exchange;
    for (i=0;i<n-1;i++)             //进行 n-1 趟排序
    {   exchange=false;             //本趟排序前置 exchange 为 false
        for (j=n-1;j>i;j--)         //无序区元素比较,找出最小元素
            if (a[j]<a[j-1])        //当相邻元素反序时
            {   tmp=a[j];           //a[j]与 a[j-1]进行交换,3 次元素移动将最小元素前移
                a[j]=a[j-1];
                a[j-1]=tmp;
                exchange=true;      //本趟排序发生交换置 exchange 为 true
            }
        if (exchange==false)        //本趟未发生交换时结束算法
            return;
    }
}
void disp(int a[],int n)            //输出 a 中所有元素
{   int i;
    for (i=0;i<n;i++)
        printf("%d",a[i]);
    printf("\n");
}
void main()
{   int n=10;
    int a[]={2,5,1,7,10,6,9,4,3,8};
```

```
    printf("排序前:"); disp(a,n);
    BubbleSort(a,n);
    printf("排序后:"); disp(a,n);
}
```

冒泡排序的主要时间花费在元素比较和交换上，当初始数据正序时，只需通过一趟排序，此时呈现最好的时间复杂度为 $O(n)$。当数据反序时呈现最坏情况，元素比较次数为 $\sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1}1=\sum_{i=0}^{n-2}(n-i-1)=\frac{n(n-1)}{2}$，元素移动次数为其 3 倍，最坏时间复杂度为 $O(n^2)$。该算法的平均时间复杂度也为 $O(n^2)$。

3.2.3 求解幂集问题

问题描述：对于给定的正整数 $n(n\geqslant 1)$，求 $1\sim n$ 构成的集合的所有子集(幂集)。

解法 1：采用直接穷举法求解，将 $1\sim n$ 的存放在数组 a 中，求解问题变为构造集合 a 的所有子集。设集合 $a[0..2]=\{1,2,3\}$，其所有子集对应的二进制位及其十进制数如表 3.1 所示。

表 3.1 所有子集对应的二进制位

子　集	对应的二进制位	对应的十进制数
{}	—	—
{1}	001	1
{2}	010	2
{3}	011	3
{1,2}	100	4
{1,3}	101	5
{2,3}	110	6
{1,2,3}	111	7

因此对于含有 $n(n\geqslant 1)$ 个元素的集合 a，求幂集的过程如下：

```
for (i = 0;i < 2n;i++)              //穷举 a 的所有子集并输出
{   将 i 转换为二进制数 b;
    输出 b 中为 1 的位对应的 a 元素构成一个子集;
}
```

采用穷举法求 $a=\{1,2,3\}$ 的幂集的完整程序如下：

```
#include <stdio.h>
#include <math.h>
#define MaxN 10
void change(int b[],int n);
void pset(int a[],int b[],int n)
{   int i,k;
    int pw = pow(2,n);              //求 2n
    printf("集合 a 的所有子集:");
    for(i = 0;i < pw;i++)           //执行 2n 次
    {    printf("{ ");
```

```
        for (k = 0;k < n;k++)        //执行 n 次
            if(b[k])
                printf(" %d ",a[k]);
        printf("} ");
        change(b,n);                 //b 表示的二进制数增 1
    }
    printf("\n");
}
void change(int b[], int n)          //改变 b 数组,将 b 表示的二进制数增 1
{   int i;
    for(i = 0;i < n;i++)             //遍历数组 b
    {   if(b[i])                     //将元素 1 改为 0
            b[i] = 0;
        else                         //将元素 0 改为 1 并退出 for 循环
        {   b[i] = 1;
            break;
        }
    }
}
void main()
{   int n = 3,i;
    int a[MaxN],b[MaxN];
    for (i = 0;i < n;i++)
    {   a[i] = i + 1;                //a 初始化为{1,2,3}
        b[i] = 0;                    //b 初始化为{0,0,0}
    }
    pset(a,b,n);
}
```

程序的输出结果如下：

```
集合 a 的所有子集:{ } { 1 } { 2 } { 1 2 } { 3 } { 1 3 } { 2 3 } { 1 2 3 }
```

显然该算法的时间复杂度为 $O(n\times 2^n)$，属于指数级的算法。

解法 2：采用增量穷举法求解 1～n 的幂集，当 $n=3$ 时的求解过程如图 3.4 所示，先产生一个空子集{}，在此基础上添加 1 构成一个子集{1}，然后在{}和{1}各子集中添加 2 产生子集{2}、{1,2}，再在前面所有子集中添加 3 产生子集{3}、{1,2}、{2,3}、{1,2,3}，从而生成{1,2,3}的所有子集。

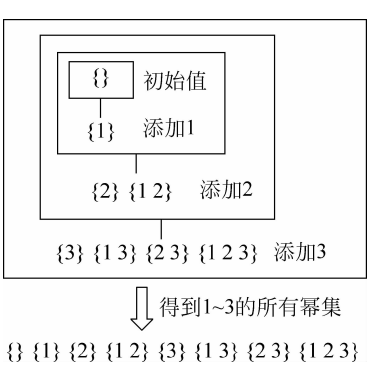

图 3.4　求 1～3 的幂集的过程

这种思路也是穷举法方法，即穷举 1～n 的所有子集。先建立一个空子集，对于 i(1≤i≤n)，每次都是在前面已建立的子集上添加元素 i 而构成若干个子集，对应的过程如下：

```
void f(int n)                    //求 1～n 的幂集 ps
{   置 ps = {{}};                //在 ps 中加入一个空子集元素
    for (i = 1;i <= n;i++)
        在 ps 的每个元素中添加 i 而构成一个新子集;
}
```

用一个二维数组 data 存放所有幂集，data[i]存放第 i 个子集，其中 data[i][0]存放该子集中的元素个数(子集长度)，例如，用 data[3]存放子集{1,2}，则 data[3][0]＝2，data[3][1]＝1，data[3][2]＝2。用 n 记录幂集中子集的个数。为此将 data 数组和 n 构成一个结构体类型 PSetType。对应的求解 1～3 集合的幂集的完整程序如下：

```
#include <stdio.h>
#define Maxn 10                          //最大的 n 值
#define MaxSize 1000                     //最大的子集个数
typedef struct                           //定义幂集类型
{   int data[MaxSize][Maxn];             //data[i][0]表示该子集的长度
    int n;                               //子集个数
} PSetType;
void copy(int a[],int b[],int m)         //将 a[0..m]复制到 b[0..m]
{   int i;
    for (i = 0;i <= m;i++)
        b[i] = a[i];
}
void pset(int n,PSetType &p)             //求 1～n 的幂集 p
{   int i,j,m;
    int a[Maxn];                         //a 用于保存一个子集
    p.data[0][0] = 0; p.n = 1;           //置初值空集{}
    for (i = 1;i <= n;i++)               //循环添加 1～n
    {   m = p.n;                         //求原幂集中的子集个数
        for (j = 0;j<m;j++)              //将每个子集添加 i 后插入到幂集中
        {    copy(p.data[j],a,p.data[j][0]);
            a[0]++;                      //子集的长度增 1
            a[a[0]] = i;                 //子集尾添加 i
            copy(a,p.data[p.n],a[0]);    //该子集插入到幂集中
            p.n++;                       //幂集中子集个数增 1
        }
    }
}
void disp(PSetType p)                    //输出幂集 p
{   int i,j;
    for (i = 0;i<p.n;i++)
    {   printf("{ ");
        for (j = 1;j <= p.data[i][0];j++)
            printf("%d ",p.data[i][j]);
        printf("} ");
    }
}
```

```
void main()
{    int n = 3;
     PSetType p;
     pset(n,p);
     printf("1～%d 的幂集如下:",n);
     disp(p); printf("\n");
}
```

程序的输出结果如下：

```
1～3 的幂集如下:{ } { 1 } { 2 } { 1 2 } { 3 } { 1 3 } { 2 3 } { 1 2 3 }
```

对于给定的 n，每一个子集都要处理，有 2^n 个，所以上述算法的时间复杂度为 $O(n\times 2^n)$。

3.2.4　求解 0/1 背包问题

问题描述：有 n 个重量分别为 $\{w_1,w_2,\cdots,w_n\}$ 的物品，它们的价值分别为 $\{v_1,v_2,\cdots,v_n\}$，给定一个容量为 W 的背包。设计从这些物品中选取一部分物品放入该背包的方案，每个物品要么选中要么不选中，要求选中的物品不仅能够放到背包中，而且具有最大的价值。并对表 3.2 所示的 4 个物品求出 $W=7$ 时的所有解和最佳解。

表 3.2　4 个物品的信息

物品编号	重量	价值
1	5	4
2	3	4
3	2	3
4	1	1

问题求解：对于 n 个物品、容量为 W 的背包问题，采用前面求幂集的方法求出所有的物品组合，对于每一种组合，计算其总重量 sumw 和总价值 sumv，当 sumw 小于等于 W 时，该组合是一种解，并通过比较将最佳方案保存在 maxsumw 和 maxsumv 中，最后输出所有的解和最佳解。

对于表 3.2 所示的 4 个物品，当 $W=7$ 时，求解所有解和最佳解的完整程序如下：

```
#include <stdio.h>
#define Maxn 10                          //最大的 n 值
#define MaxSize 1000                     //最大的子集个数
typedef struct                           //定义幂集类型
{    int data[MaxSize][Maxn];            //data[i][0]表示该子集的长度
     int n;                              //子集个数
} PSetType;
void copy(int a[],int b[],int m)         //将 a[0..m]复制到 b[0..m]
{    int i;
     for (i = 0;i <= m;i++)
          b[i] = a[i];
}
void pset(int n,PSetType &p)             //求 1～n 的幂集 p
```

```
{   int i,j,m;
    int a[Maxn];                                //a用于保存一个子集
    p.data[0][0] = 0; p.n = 1;                  //置初值空集{}
    for (i = 1;i <= n;i++)                      //循环添加1～n
    {   m = p.n;                                //求原幂集中的子集个数
        for (j = 0;j < m;j++)                   //将每个子集添加i后插入到幂集中
        {   copy(p.data[j],a,p.data[j][0]);
            a[0]++;                             //子集的长度增1
            a[a[0]] = i;                        //子集尾添加i
            copy(a,p.data[p.n],a[0]);           //该子集插入到幂集中
            p.n++;                              //幂集中子集个数增1
        }
    }
}
void knap(PSetType p,int w[],int v[],int W)     //求所有的方案和最佳方案
{   int i,j;
    int sumw,sumv;
    int maxi,maxsumw = 0,maxsumv = 0;
    printf("  序号\t选中物品\t总重量\t总价值\t能否装入\n");
    for (i = 0;i < p.n;i++)
    {   printf("   %d\t",i + 1);
        sumw = sumv = 0;
        printf("{ ");
        for (j = 1;j <= p.data[i][0];j++)
        {   printf(" %d ",p.data[i][j]);
            sumw += w[p.data[i][j] - 1];
            sumv += v[p.data[i][j] - 1];
        }
        printf("}\t\t%d\t%d\t",sumw,sumv);
        if (sumw <= W)
        {   printf("能\n");
            if (sumv > maxsumv)
            {    maxsumw = sumw;
                 maxsumv = sumv;
                 maxi = i;
            }
        }
        else printf("否\n");
    }
    printf("最佳方案为 ");
    printf("选中物品:");
    printf("{ ");
    for (j = 1;j <= p.data[maxi][0];j++)
        printf(" %d ",p.data[maxi][j]);
    printf("},");
    printf("总重量:%d,总价值:%d\n",maxsumw,maxsumv);
}
void main()
{   int n = 4,W = 7;
    int w[] = {5,3,2,1};
    int v[] = {4,4,3,1};
```

```
    PSetType p;
    pset(n,p);
    printf("0/1 背包的求解方案\n",n);
    knap(p,w,v,W); printf("\n");
}
```

程序执行结果如下：

```
0/1 背包的求解方案
序号  选中物品    总重量  总价值  能否装入
 1    { }          0       0       能
 2    { 1 }        5       4       能
 3    { 2 }        3       4       能
 4    { 1 2 }      8       8       否
 5    { 3 }        2       3       能
 6    { 1 3 }      7       7       能
 7    { 2 3 }      5       7       能
 8    { 1 2 3 }   10      11       否
 9    { 4 }        1       1       能
10    { 1 4 }      6       5       能
11    { 2 4 }      4       5       能
12    { 1 2 4 }    9       9       否
13    { 3 4 }      3       4       能
14    { 1 3 4 }    8       8       否
15    { 2 3 4 }    6       8       能
16    { 1 2 3 4 }  11      12       否
最佳方案为 选中物品:{ 2 3 4 },总重量:6,总价值:8
```

3.2.5　求解全排列问题

问题描述：对于给定的正整数 $n(n\geqslant 1)$，求 $1\sim n$ 的所有全排列。

问题求解：这里采用增量穷举法求解。产生 1～3 全排列的过程如图 3.5 所示，这里 $n=3$，采用一个顺序栈 st，栈中每个元素存放一个排列（用数组 a 表示，为了简便，用 $a[1..m]$ 存放一个排列）及其长度（用 m 表示），另设一个栈顶指针 top。

首先将({1},1)进栈（前者表示进栈的一个排列，后者表示其长度），然后栈不空循环：退栈元素({1},1)，将 2 插入到{1}的各位上构成({1,2},2)和({2,1},2)并进栈；退栈元素({1,2},2)，将 3 插入到{1,2}的各位上构成({1,2,3},3)、({1,3,2},3)和({3,1,2},3)并进栈；退栈元素({2,1},2)，将 3 插入到{2,1}的各位上构成({2,1,3},3)、({2,3,1},3)和({3,2,1},3)并进栈。再退栈时，栈中各元素长度均为 n，将其作为一个排列输出并退栈。当栈为空时算法结束。

对应的过程如下：

```
void f(int n)
{   ({1},1)进栈;
    while (栈不空)
    {   取栈顶元素的长度 m;
        if (m == n)
            输出一个排列并退栈;
```

```
            else
            {   将栈顶元素序列复制到 c 中并退栈;
                在 c 中的 1～m+1 的位置上插入 m+1 构成一个排列(中间结果)并进栈;
            }
        }
    }
```

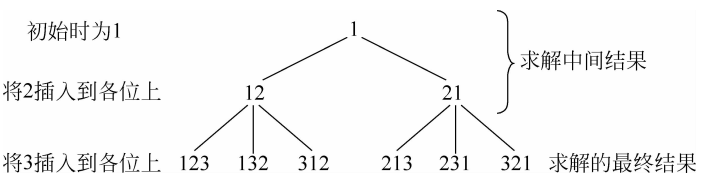

图 3.5　产生 1～3 全排列的过程

求解 1～3 的全排列的完整程序如下：

```
#include <stdio.h>
#define Maxn 10                                  //最大的 n 值
#define MaxSize 1000
typedef struct                                   //定义栈元素类型
{   struct
    {   int a[Maxn];                             //存放一个排列
        int m;                                   //存放该排序的元素个数
    } data[MaxSize];
    int top;                                     //栈顶指针
} StackType;
void insert(int a[],int m,int j,int k)           //在 a[1..m]中 a[j]处插入 k
{   int i;
    for (i=m+1;i>j;i--)                          //将 a[j..m]元素后移
        a[i]=a[i-1];
    a[j]=k;
}
void disp(int a[],int m)                         //输出 a[1..m]中元素
{   int i;
    for (i=1;i<=m;i++)
        printf("%d",a[i]);
    printf(" ");
}
void copy(int a[],int b[],int m)                 //将 a[1..m]复制到 b[1..m]
{   int i;
    for (i=1;i<=m;i++)
        b[i]=a[i];
}
void perm(int n)                                 //输出 1～n 的所有全排列
{   int j,m;
    int b[Maxn],c[Maxn];                         //用于临时存放一个排列
    StackType st;                                //定义一个顺序栈
    st.top=-1;                                   //初始化顺序栈
    st.top++;                                    //将 1 进栈
    st.data[st.top].a[1]=1; st.data[st.top].m=1;
```

```
    while (st.top!= -1)                          //栈不空循环
    {   m = st.data[st.top].m;                   //取栈顶元素的 m 值
        if (m == n)                              //找到一种全排列,输出并退栈
        {   disp(st.data[st.top].a,n);
            st.top--;                            //退栈
        }
        else                                     //找到一种部分排序
        {   copy(st.data[st.top].a,c,m);         //取出栈顶排列到 c 中
            st.top--;                            //退栈
            for (j = 1;j <= m + 1;j++)
            {   copy(c,b,m);                     //将 c[1..m]中部分排序复制到 b 中
                insert(b,m,j,m + 1);             //将 m + 1 插入到 b[j]处
                st.top++;                        //将 b 进栈
                copy(b,st.data[st.top].a,m + 1);
                st.data[st.top].m = m + 1;
            }
        }
    }
}
void main()
{   int n = 3;
    printf("1～%d 的全排序如下:",n);
    perm(n); printf("\n");
}
```

程序执行结果如下：

```
1～3 的全排序如下:123 132 312 213 231 321
```

对于给定的 n，每一种全排列都必须处理，有 $n!$ 种，所以上述算法的时间复杂度为 $O(n\times n!)$。

3.2.6　求解最大连续子序列和问题

问题描述：给定一个有 $n(n\geqslant 1)$ 个整数的序列，要求求出其中最大连续子序列的和。例如序列(−2,11,−4,13,−5,−2)的最大子序列和为 20，序列(−6,2,4,−7,5,3,2,−1,6,−9,10,−2)的最大子序列和为 16。

解法 1：设含有 n 个整数的序列 $a[0..n-1]$，其中任何连续子序列 $a[i..j](i\leqslant j,0\leqslant i\leqslant n-1,i\leqslant j\leqslant n-1)$ 求出它的所有元素之和 thisSum，并通过比较将最大值存放在 maxSum 中，最后返回 maxSum。这种解法是通过穷举所有连续子序列(一个连续子序列由起始下标 i 和终止下标 j 确定)来得到，是典型的穷举法思想。

本解法对应的完整程序如下：

```
#include <stdio.h>
long maxSubSum1(int a[],int n)
{   int i,j,k;
    long maxSum = a[0],thisSum;
    for (i = 0;i < n;i++)                        //两重循环穷举所有的连续子序列
    {    for (j = i;j < n;j++)
```

```
        {   thisSum = 0;
            for (k = i;k <= j;k++)
                thisSum += a[k];
            if (thisSum > maxSum)              //通过比较求最大连续子序列之和
                maxSum = thisSum;
        }
    }
    return maxSum;
}
void main()
{   int a[] = { - 2,11, - 4,13, - 5, - 2},n = 6;
    int b[] = { - 6,2,4, - 7,5,3,2, - 1,6, - 9,10, - 2},m = 12;
    printf("a 序列的最大连续子序列的和: % ld\n",maxSubSum1(a,n));
    printf("b 序列的最大连续子序列的和: % ld\n",maxSubSum1(b,m));
}
```

maxSubSum1(a,n)算法中用了 3 重循环,所以有:

$$T(n)=\sum_{i=0}^{n-1}\sum_{j=i}^{n-1}\sum_{k=i}^{j}1=\sum_{i=0}^{n-1}\sum_{j=i}^{n-1}(j-i+1)=\frac{1}{2}\sum_{i=0}^{n-1}(n-i)(n-i+1)=O(n^3)$$

解法 2:改进前面的解法,在求两个相邻子序列之和时,它们之间是关联的,例如,$a[0..3]$子序列之和=$a[0]+a[1]+a[2]+a[3]$,$a[0..4]$子序列之和=$a[0]+a[1]+a[2]+a[3]+a[4]$,在前者计算出来后,求后者时只需在前者基础上加上 $a[4]$即可,没有必须每次都重复计算。从而提高了算法的效率。

改进后的完整程序如下:

```
#include <stdio.h>
long maxSubSum2(int a[],int n)
{   int i,j;
    long maxSum = a[0],thisSum;
    for (i = 0;i < n;i++)
    {   thisSum = 0;
        for (j = i;j < n;j++)
        {   thisSum += a[j];
            if (thisSum > maxSum)
                maxSum = thisSum;
        }
    }
    return maxSum;
}
void main()
{   int a[] = { - 2,11, - 4,13, - 5, - 2},n = 6;
    int b[] = { - 6,2,4, - 7,5,3,2, - 1,6, - 9,10, - 2},m = 12;
    printf("a 序列的最大连续子序列的和: % ld\n",maxSubSum2(a,n));
    printf("b 序列的最大连续子序列的和: % ld\n",maxSubSum2(b,m));
}
```

maxSubSum2(a,n)算法中只有两重循环,容易求出 $T(n)=O(n^2)$。尽管这样改进后降低了算法的时间复杂度,但仍采用的是穷举法思路。

解法 3：进一步改进解法 2，从头开始扫描数组 a，用 thisSum(初值为 0)记录当前子序列之和，用 maxSum(初值为 0)记录最大连续子序列和。如果扫描中遇到负数，当前子序列和 thisSum 将会减小，若 thisSum 为负数，表明前面已经扫描的子序列可以抛弃了，则放弃这个子序列，重新开始下一个子序列的分析，并置 thisSum 为 0。若这个子序列和 thisSum 不断增加，那么最大子序列和 maxSum 也不断增加。本算法仍采用穷举法的思路。

改进后的完整程序如下：

```
#include <stdio.h>
int maxSubSum4(int a[],int n)
{   int i,maxSum = 0,thisSum = 0;
    for (i = 0;i < n;i++)
    {   thisSum += a[i];
        if (thisSum < 0)          //若当前子序列和为负数,则重新开始下一个子序列
            thisSum = 0;
        if (maxSum < thisSum)     //比较求最大连续子序列和
            maxSum = thisSum;
    }
    return maxSum;
}
void main()
{   int a[] = { - 2,11, - 4,13, - 5, - 2},n = 6;
    int b[] = { - 6,2,4, - 7,5,3,2, - 1,6, - 9,10, - 2},m = 12;
    printf("a 序列的最大连续子序列的和: %ld\n",maxSubSum4(a,n));
    printf("b 序列的最大连续子序列的和: %ld\n",maxSubSum4(b,m));
}
```

显然该算法中仅扫描 a 一次，其算法的时间复杂度为 $O(n)$。

从中看出，尽管一般而言采用穷举法设计的算法效率不高，但通过精心设计，仍可以设计出高效的算法。

3.3　递归在穷举法中的应用

穷举法所依赖的基本技术是遍历技术，采用一定的策略将待求解问题的所有元素依次处理一次，从而找出问题的解。而在遍历过程中，很多求解问题都可以采用递归方法来实现，如二叉树的遍历和图的遍历等。本节通过几个经典示例介绍其算法设计方法。

3.3.1　用递归方法求解幂集问题

前面介绍了两种采用穷举法求解由 1～n 整数构成的集合的幂集的方法，这里以解法 2 为基础。

用变量 p 存放幂集，其定义见 3.2.3 小节的解法 2。初始时 $p=\{\{\}\}$，设 $f(i,n,p)$用于添加 i～n 整数(共需添加 $n-i+1$ 个整数)产生的幂集 p，显然 $f(1,n,p)$产生由 1～n 整数构成的集合的幂集 p。$f(i+1,n,p)$用于添加 $i+1$～n 整数(共需添加 $n-i$ 个整数)产生的幂集，所以 $f(i,n,p)$是“大问题”，$f(i+1,n,p)$是“小问题”，对应的递归模型如下：

$f(i,n,p)$≡输出幂集 p　　当 $i>n$

$f(i,n,p)$≡将整数 i 添加到 p 中原有每个子集中产生新子集；　否则

　　并将所有新子集加入到 p 中；

　　$f(i+1,n,p)$；

对应的完整的求解程序如下：

```
#include <stdio.h>
#define Maxn 10                          //最大的 n 值
#define MaxSize 1000                     //最大的子集个数
typedef struct                           //定义幂集类型
{   int data[MaxSize][Maxn];             //data[i][0]表示该子集的长度
    int n;                               //子集个数
} PSetType;
void copy(int a[],int b[],int m)         //a[0..m]复制到 b[0..m]
{   int i;
    for (i=0;i<=m;i++)
        b[i]=a[i];
}
void addi(PSetType &p,int i)             //向幂集 p 中所有子集添加 i 产生新子集并插入到 p 中
{   int j,m;
    int a[Maxn];
    m=p.n;
    for (j=0;j<m;j++)                    //处理 p 中原有的所有子集
    {   copy(p.data[j],a,p.data[j][0]);
        a[0]++;                          //子集的长度增 1
        a[a[0]]=i;                       //子集尾添加 i
        copy(a,p.data[p.n],a[0]);        //该子集插入到幂集中
        p.n++;                           //幂集中子集个数增 1
    }
}
void disp(PSetType p)                    //输出幂集
{   int i,j;
    for (i=0;i<p.n;i++)
    {   printf("{ ");
        for (j=1;j<=p.data[i][0];j++)
            printf("%d ",p.data[i][j]);
        printf("} ");
    }
}
void pset(int i,int n,PSetType p)        //输出 1～n 的幂集
{   if (i>n)                             //满足递归出口条件,输出幂集
    {   printf("1～%d 的幂集如下:",n);
        disp(p); printf("\n");
    }
    else
    {   addi(p,i);                       //将 i 插入到现有子集中产生新子集
        pset(i+1,n,p);                   //递归调用
    }
}
void main()
```

```
{   int n = 3;
    PSetType p;                          //定义幂集
    p.data[0][0] = 0; p.n = 1;           //幂集置初始空集
    pset(1,n,p);                         //输出 1～n 的幂集
}
```

3.3.2　用递归方法求解全排列问题

3.2.5 小节介绍过一种求 1～n 的全排列的方法，现在采用递归穷举法求解。

用数组 $a[0..n-1]$存放一种排列，设 $f(a,n,k)$用于求 $a[k..n-1]$的全排列，是“大问题”，则 $f(a,n,k+1)$求 $a[k+1..n-1]$的全排列，是“小问题”。对应的递归模型如下：

$f(a,n,k)$ ≡输出 a 中 n 个元素构成一个排列　　当 $k=n$ 时

$f(a,n,k)$ ≡将 1～n 不重复地插入到 $a[k]$位置上；　否则 $f(a,n,k+1)$；

采用递归穷举法求解 1～3 的全排列的完整程序如下：

```
#include <stdio.h>
#define MaxSize 10
void disp(int a[],int n)                 //输出一个排列
{   int i;
    for (i = 0;i < n;i++)
        printf("%d",a[i]);
    printf(" ");
}
void perm(int a[],int n,int k)
{   int i,j;
    if (k == n)                          //当 a 中所有位置已填数,则输出一个解
        disp(a,n);
    else
        for (i = 1;i <= n;i++)           //尝试在 a 中填入整数 i
        {   bool has = false;
            for (j = 0;j < k;j++)        //在 a[0..k]中出现过 i,置 has 为 true
                if (a[j] == i) has = true;
            if (!has)                    //若 a[0..k]中没有出现过 i,则选 i
            {   a[k] = i;
                perm(a,n,k + 1);         //继续添加下一个位置的整数
            }
        }
}
void main()
{   int n = 3;
    int a[MaxSize];
    printf("1～%d 的全排序如下:",n);
    perm(a,n,0);
    printf("\n");
}
```

程序执行结果如下：

```
1～3 的全排序如下:123 132 213 231 312 321
```

3.3.3 用递归方法求解组合问题

问题描述：求 1～n 的正整数中取 k($k \leqslant n$)个不重复整数的所有组合。

问题求解：用数组元素 $a[0..k-1]$来保存一个组合，由于一个组合中所有元素不会重复出现，规定 a 中所有元素按递增排列。设 comb(a,n,k)为从 1～n 中任取 k 个数的所有组合，它是“大问题”，comb(a,m,$k-1$)为从 1～m 中任取 $k-1$ 个数的所有组合($k-1 \leqslant m < n$)，它是“小问题”。因为 a 中元素递增排列，所以 $a[k-1]$的取值范围只能为 k～n，当 $a[k-1]$确定为 i 后，合并 comb(a,$i-1$,$k-1$)的一个结果便构成 comb(a,n,k)的一个组合结果，如图 3.6 所示。当 $k=0$ 时，a 中所有元素均已确定，输出 a 中一种组合。对应的递归模型如下：

$$\text{comb}(a,n,k) \equiv \text{输出 } a \text{ 中的一种组合} \qquad \text{当 } k=0$$

$$\text{comb}(a,n,k) \equiv \text{for } (i=k;i<=n;i++); \qquad \text{当 } k>0$$
$$\{\ a[k-1]=i;\ \text{comb}(a,i-1,k-1);\ \}$$

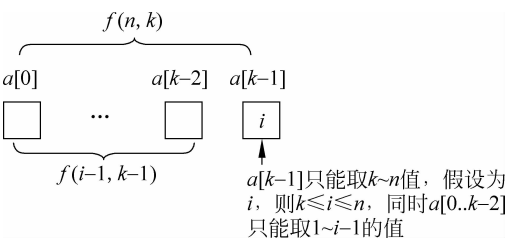

图 3.6 求 comb(a,n,k)的过程

求 1～5 中取 3 个整数组合的完整程序如下：

```
#include <stdio.h>
#define MAXK 10
int n,k;                              //全局变量
void dispcomb(int a[])                //输出一个组合
{    int j;
     for (j=0;j<k;j++)
     printf("%3d",a[j]);
     printf("\n");
}
void comb(int a[],int n,int k)        //求 1..n 中 k 个整数的组合
{   int i;
    if (k==0)                         //k 为 0 时输出一个组合
        dispcomb(a);
    else
    {   for (i=k;i<=n;i++)
        {   a[k-1]=i;                 //a[k-1]位置取 k～n 的整数
            comb(a,i-1,k-1);          //a[k-1]组合 a[0..i-1]中的 k-1 个整数产生一个组合
        }
    }
}
void main()
```

```
{     int a[MAXK];
      n = 5; k = 3;
      printf("1.. %d 中 %d 个的整数的所有组合:\n",n,k);
      comb(a,n,k);
}
```

该程序的输出结果如下：

```
1..5 中 3 个的整数的所有组合:
  1   2   3
  1   2   4
  1   3   4
  2   3   4
  1   2   5
  1   3   5
  2   3   5
  1   4   5
  2   4   5
  3   4   5
```

求 1～5 中取 3 个整数组合的过程如图 3.7 所示。

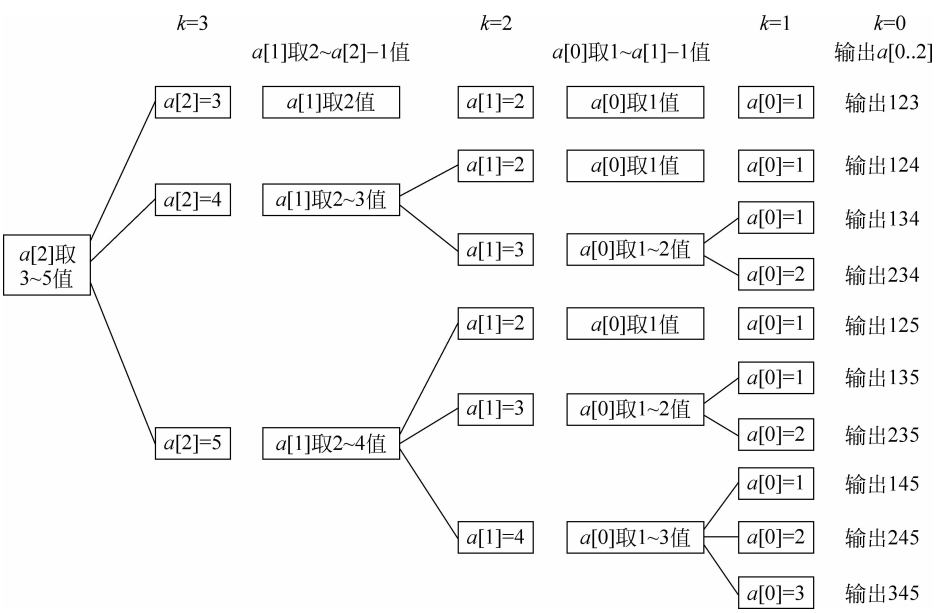

图 3.7　求 1～5 中取 3 个整数组合的过程

上机实验题 3——钱币兑换问题

钱币兑换问题：某个国家仅有 1 分、2 分和 5 分硬币，将钱 $n(n\geqslant 5)$ 兑换成硬币有很多种兑法。设计一个实验程序计算出 10 分钱有多少种兑法以及每种兑换方式。

练习题 3

1. 简述穷举法的特点。

2. 在采用穷举法求解时什么情况下使用递归?

3. 给定一个整数数组 $A=(a_0,a_1,\cdots a_{n-1})$,若 $i<j$ 且 $a_i>a_j$,则$<a_i,a_j>$就为一个逆序对。例如数组(3,1,4,5,2)的逆序对有$<3,1>$,$<3,2>$,$<4,2>$,$<5,2>$。设计一个算法采用穷举法求 A 中逆序对的个数。

4. 设计求解有重复元素的排列问题的算法,设有 n 个元素$(r_0,r_1,\cdots,r_{n-1})$,其中可能含有重复的元素,求这些元素的所有不同排列。

5. 有一群鸡和一群兔,它们的只数相同,它们的脚数都是 3 位数,且这两个 3 位数的数字分别是 0、1、2、3、4、5。设计一个算法用穷举法求鸡和兔的只数各是多少? 它们的脚数各是多少?

6. 有一个 3 位数,个位数字比百位数字大,而百位数字又比十位数字大,并且各位数字之和等于各位数字相乘之积,设计一个算法用穷举法求此 3 位数。

7. 甲、乙两数的和为 168,甲数的八分之一与乙数的四分之三的和为 76,设计一个算法用穷举法求甲、乙两数各是多少?

8. 我国古代数学问题:1 兔换 2 鸡,2 兔换 3 鸭,5 兔换 7 鹅。某人用 20 只兔换得鸡、鸭、鹅共 30 只,设计一个算法用穷举法求其中鸡、鸭、鹅各几只?

9. 某年级的同学集体去公园划船,如果每只船坐 10 人,那么多出 2 个座位;如果每只船多坐 2 人,那么可少租 1 只船,设计一个算法用穷举法求一共需要租几只船?

10. 松鼠妈妈摘松果,晴天每天可摘 20 个,雨天每天可摘 12 个。它一连几天摘了 112 个松果,平均每天摘 14 个。设计一个算法用穷举法求这些天中有几天下雨?

11. 一辆汽车共载客 50 人,其中一部分人买 A 种票,每张 0.80 元;另一部分人买 B 种票,每张 0.30 元。售票员最后统计出所卖的 A 种票比卖 B 种票多收入 18 元。设计一个算法用穷举法求买 A 种票的有多少人。

12. 设计一个穷举法算法,将 1～9 这 9 个数字组成的 3 个 3 位的平方数,要求每个数字只准使用一次。

13. 有 36 块砖,有 36 个人,男的每次可搬 4 块,女的每次可搬 3 块,2 个小孩可以每次搬 1 块。36 块砖如果要一次搬完,设计一个算法用穷举法求需要多少男的、女的和小孩?

14. 对于正整数 $k(k\leqslant 50)$,设计一个算法用穷举法找到所有的正整数 $x\geqslant y$,使得 $\frac{1}{k}=\frac{1}{x}+\frac{1}{y}$。

15. **【ACM 训练题】**已知:若一个合数的质因数分解式逐位相加之和等于其本身逐位相加之和,则称这个数为 Smith 数。如 $4937775=3\times5\times5\times65837$,而 $3+5+5+6+5+8+3+7=42$,$4+9+3+7+7+7+5=42$,所以 4937775 是 Smith 数。求给定一个正整数 N,求大于 N 的最小 Smith 数。

输入:若干个 case,每个 case 一行代表正整数 N,输入 0 表示结束。

输出：大于 N 的最小 Smith 数。
输入样本：

```
4937774
0
```

输出结果：

```
4937775
```