第5章 精巧求解剖析

精巧求解，包括高精度计算，是最具吸引力的亮点，也是让人望而却步的难点。

本章从互积和与嵌套根式和的巧算、同码数的整除及同码数求和规律的探索，到建模统计、分类统计、游戏中的素数概率求解，突出一个“巧”字。同时，从探索最小 0-1 串积到指定多码串积与尾数前移问题等，突显一个“精”字。

探讨并拓展著名的梅齐里亚克砝码问题与伯努利装错信封的排列问题，是“巧”与“精”结合的典范。飘逸于数学殿堂的两个“幽灵”e 和 π，凝聚着数学的精华，彰显出编程的卓越。

5.1 和与积巧算

本节探讨简单的互积和与嵌套根式和，不存在难点，但求解技巧颇具启发性。

5.1.1 互积和

探求已知整数组的互积和是涉及和与积的常规题，有较强的技巧性，常常作为数学奥赛的培训题或测试题。

有 9 个整数：$-3,-2,-1,0,1,2,4,8,16$，对这 9 个整数求任意 2 个数之积，任意 3 个数之积，……，直至所有 9 个数之积。把所有这些积之和称为这些整数的互积和。

【问题】 试求这 9 个整数的互积和 m。

【分类求解】 互积和分类是简化求解的关键。

可忽略 0，因凡含有 0 的积结果均为 0，对最后结果没有影响。

注意到所有正数和为 31，分以下 5 类情形考虑。

(1) 所有正数互积和，设为 s_1（不含单个正数）。

(2) 所有负数互积和：$s_2=2+3+6-6=5$。

(3) 含 1 个负数的互积和：$s_3=(-1-2-3)(s_1+31)=-6s_1-6\times31$。

(4) 含 2 个负数的互积和：$s_4=11\times(s_1+31)=11s_1+11\times31$。

(5) 含 3 个负数的互积和：$s_5=-6\times(s_1+31)=-6s_1-6\times31$。

以上 5 项求和，巧妙消去其中尚未算出的 s_1，得到互积和 m。

$$m=s_1+s_2+s_3+s_4+s_5=5-31=-26$$

【妙思巧解】 巧妙构造多项式化互积为多项式积。

记 8 个非零整数为 $a_1,a_2,\cdots,a_8$，构造多项式

$$f(x)=(x+a_1)(x+a_2)\cdots(x+a_8)=x^8+p_1x^7+p_2x^6+\cdots+p_7x+p_8 \tag{5-1-1}$$

其中,p_1 为 8 个数之和 25;p_2 为任意 2 个数积之和;p_3 为任意 3 个数积之和;……;p_8 为所有 8 个数之积。显然,所求互积和 $m=p_2+p_3\cdots+p_8$。

令 $x=1$,由式(5-1-1)左边知 $f(1)=0$(因有一整数为 -1);同时,由式(5-1-1)右边有 $f(1)=1+p_1+p_2+\cdots+p_8=1+25+m$,因而所求结果为 $m=p_2+p_3+\cdots+p_8=-26$。

给出的整数中有 -1,即得和 $f(1)=0$,这是减少计算量的关键所在。

即使给出的 8 个整数中没有 -1,计算式(5-1-1)左边的 8 个数之积也远比计算“互积”简单。

【概括】 以上两个求解都颇具启发性。如果按“互积”的定义,具体实施每 2 个数求积,每 3 个数求积,……,这一思路并不可行。而分类的思想和建多项式的思想,就是化解“互积”这一难点的巧妙突破点。

如果对于指定的一般 n 个整数,如何求取其互积和?

【编程拓展】 拓展至一般情形,探求 n 个整数的互积和。

对给出的 n 个整数,计算任意 2 个数之积,任意 3 个数之积,……,直到所有 n 个数之积。定义所有这些积之和为这 n 个整数的互积和 m。

从键盘输入 n 个整数,探求并输出这 n 个整数的互积和 m。

(1) 设计要点。

按上述巧妙构造多项式,化互积为多项式积。

设置存储 n 个整数的 a 数组,从键盘输入的 n 个整数存储在 $a[1],a[2],\cdots,a[n]$。

同时构造关于这 n 个整数的 n 次多项式

$$y(x)=(x+a_1)(x+a_2)\cdots(x+a_n)=x^n+p_1x^{n-1}+p_2x^{n-2}+\cdots+p_{n-1}x+p_n \tag{5-1-2}$$

其中,p_1 为 n 数之和;p_2 为任意 2 个数积之和;……;p_{n-1} 为任意 $n-1$ 个数积之和;p_n 为所有 n 个数之积。显然,$m=p_2+\cdots+p_n$。

令 $x=1$,按式(5-1-2)左边通过循环求积,可简单得出 $y(1)$;按式(5-1-2)右边有 $y(1)=1+p_1+p_2+\cdots+p_{n-1}+p_n$;同时在循环中求出 n 个整数之和 p_1,显然所求互积和为

$$m=y(1)-p_1-1 \tag{5-1-3}$$

循环结束,即按式(5-1-3)输出所求 n 个整数的互积和 m。

(2) 程序设计。

```
//求 n 个整数的互积和 m
#include <stdio.h>
void main()
{  long  k,m,n,s,y,a[100];
   printf("  请确定整数的个数 n:");scanf("%ld",&n);
   s=0;y=1;
```

```
    for(k=1;k<=n;k++)                                    //逐个输入 n 个整数
    {  printf("  请输入第%ld 个整数:",k);
       scanf("%ld",&a[k]);
       s+=a[k]; y*=(1+a[k]);                             //计算整数和 s 及 f(1)
    }
    m=y-1-s;                                             //计算互积和 m
    printf("  输入的%ld 个整数为%ld",n,a[1]);
    for(k=2;k<=n;k++)                                    //集中输出 n 个整数
      printf(", %ld",a[k]);
    printf("\n  以上%ld 个整数的互积和 m=%ld。\n",n,m);    //输出结果
}
```

(3) 程序运行示例与说明。

```
请确定整数的个数 n:8
输入的 8 个整数为-5, -3, -2, 3, 5, 8, 10, 13
以上 8 个整数的互积和 m=-266142。
```

运行程序时,输入 n 个整数的先后顺序对互积和结果没有影响。

以上程序把求互积(若干积)转化为求一个积,是简化复杂运算的技巧所在。

同时,程序设置在输入循环中输入与计算(和与积)同步进行,输入循环结束,计算也随之完成。

如果运行程序,输入 9 个整数－3,－2,－1,0,1,2,4,8,16,即可得到互积和为－26,其中的整数－1 是一个简化因子。

即使没有－1 这一简化整数,以上程序实现把“互积”难点转化为在循环中求一个积,也非常精巧。

5.1.2　嵌套根式和

试求以下两个嵌套根式和

$$s_1=\sqrt{1+\sqrt{2+\sqrt{3+\cdots+\sqrt{n}}}} \tag{5-1-4}$$

$$s_2=\sqrt{5+\sqrt{3+\sqrt{5+\sqrt{3+\cdots+\sqrt{5+\sqrt{3}}}}}} \quad \text{(式中有 } n \text{ 个 5 与 } n \text{ 个 3)} \tag{5-1-5}$$

输入正整数 n,输出根式和 s_1 与 s_2(精确到小数点后 8 位)。

1. 设计要点

对于给定的正整数 n,在 s_1 中涉及 n 层开方,而 s_2 涉及 $2n$ 层开方,新颖少见。

根式和式(5-1-4)与式(5-1-5)存在根号嵌套,处理根号嵌套这一难点是设计的关键。

按通常由起点 1 到终点 n 设置循环,不便于处理根号嵌套这一难点。但若反过来逆向设置循环,解决根号嵌套就顺理成章。

(1) 实现 s_1 求和。

设置 a(n－1～1)循环枚举整数 a,循环外赋初值“s1＝n;”,循环内求累加和。

```
s1=a+sqrt(s1);
```

赋值表达式中的 sqrt(s1),即可实现 s_1 的根号嵌套。

当 $a=n-1$ 时,"s1=(n-1)+sqrt(n);"实现最内 1 层根号。

当 $a=n-2$ 时,"s1=(n-2)+sqrt((n-1)+sqrt(n));"实现最内 2 层根号。

……

当 $a=1$ 时,s_1 实现式(5-1-4)除最外层之外的其他所有 $n-1$ 层根号。

最后一层根号在输出结果时完成。

(2) 实现 s_2 求和。

同样设置 $a(n-1\sim1)$循环,循环外赋初值"s2=5+sqrt(3);",循环 $n-1$ 次。

```
s2=5+sqrt(3+sqrt(s2));
```

赋值表达式中的 sqrt(s2)即可实现 s_2 的根号嵌套。

当 $a=n-1$ 时,"s2=5+sqrt(3+sqrt(5+sqrt(3)));"实现最内 3 层根号。

当 $a=n-2$ 时,"s2=5+sqrt(3+sqrt(5+sqrt(3+sqrt(5+sqrt(3)))));"实现最内 5 层根号。

……

当 $a=1$ 时,s_2 实现式(5-1-5)除最外层之外的其他所有 $2n-1$ 层根号。

最后一层根号在输出结果时完成。

2. 程序设计

```
//探求两个根式和
#include <stdio.h>
#include <math.h>
void main()
{  double a,n,s1,s2;
   printf("  请输入正整数 n(n>1): "); scanf("%lf",&n);
   s1=n;s2=5+sqrt(3);
   for(a=n-1;a>=1;a--)                              //枚举 n-1~1 的整数 a
     {  s1=a+sqrt(s1);
        s2=5+sqrt(3+sqrt(s2));
     }
   printf("  s1=%.8f\n",sqrt(s1));                  //最后的和 s1 与 s2 需开平方
   printf("  s2=%.8f\n",sqrt(s2));
}
```

3. 程序运行示例与变通

```
请输入正整数 n(n>1): 100
  s1=1.75793276
  s2=2.71870969
```

在所设置的 a 循环中,应用"s1=a+sqrt(s1);"及"s2=5+sqrt(3+sqrt(s2));"是实

现根号嵌套的技巧所在。

变通：容易修改以上程序，探求区间[c,d]中的根式和

$$s_1=\sqrt{c+\sqrt{(c+1)+\sqrt{(c+2)+\cdots+\sqrt{d}}}} \tag{5-1-6}$$

$$s_2=\sqrt{c+\sqrt{(c+1)+\sqrt{c+\sqrt{(c+2)+\cdots+\sqrt{c+\sqrt{d}}}}}} \tag{5-1-7}$$

输入区间上下限正整数 c，$d(c<d)$，输出根式和 s_1 与 s_2。

5.2　同码数汇趣

由同一个数码组成的数称为同码数。

例如，555，0.7777 都是同码数，前者是同码整数，而后者则是同码小数。

如果说优美数要求数字不重复体现和谐美，那么同码数强调数码相同则是奇异美的象征。本节探讨同码数整除问题，探求同码整数与同码小数之和。从同码整数之和与同码小数之和可发现一些有趣的规律与特点。

5.2.1　同码数整除

先看一个简单的同码数整除问题。

【命题】　存在正整数 $n\leqslant 2019$，使得 n 个 1 组成的同码整数 m 能被 2019 整除。

【证明】　根据余数的抽屉原理来证。

分别由 $n=1\sim 2019$ 个由 1 组成的同码整数除以 2019，其余数 r 不外乎 0，1，…，2018。

如果存在某一 n 值，n 个 1 组成的同码数除以 2019，余数 $r=0$，则命题成立。

假设分别由 $n=1\sim 2019$ 个由 1 组成的同码整数除以 2019，余数中不存在 0，这 2019 个余数 r 不外乎 1，2，…，2018。根据抽屉原理，2019 个余数中必存在至少两个余数是相同的。

不妨设 $1\leqslant n_1<n_2\leqslant 2019$，由 n_1 个 1 组成的同码整数 m_1 与由 n_2 个 1 组成的同码整数 m_2，这两个同码数除以 2019 的余数相同。于是其差 $m=m_2-m_1$ 能被 2019 整除，注意到

$$m=m_2-m_1=\overbrace{11\cdots}^{n_2-n_1}\overbrace{100\cdots}^{n_1}0=\overbrace{11\cdots}^{n_2-n_1}1\times\overbrace{100\cdots}^{n_1}0 \tag{5-2-1}$$

整数 m 是两个同码整数之积，后者只有 2 与 5 因数，不能被 2019 整除，只有前者被 2019 整除。注意到 $1\leqslant n_2-n_1<2019$，即存在 n_2-n_1 个 1 组成的同码整数能被 2019 整除，与假设矛盾。

因而证得存在正整数 $n\leqslant 2019$，使得 n 个 1 组成的同码整数 m 能被 2019 整除。

【问题】　至少需要多少个 1 组成的同码整数 m 才能被 2019 整除。

【求解】　试实施竖式除法实验探求。

探求由 1 组成的同码整数 m，至少需要多少个 1 才能被 2019 整除，可实施竖式除法，如图 5-1 所示。

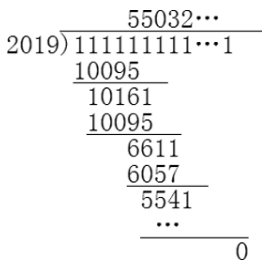

图 5-1　实施竖式除法示意图

只要有一定的耐心与时间，总可以把竖式除法做下去，找到最小的整数 n，使得 n 个 1 组成的同码数 m 能被 2019 整除。

可以肯定，除法不可能无限制地进行下去，因为有 $n \leqslant 2019$。至于 n 至少为多大，则必须要等以图 5-1 所示的试商结果出炉。

【编程探求】　探求同码数能被 p 整除。

给定正整数 p（约定整数 p 为个位数字不是 5 的奇数），探求最小的正整数 n，使得 n 个 1 组成的同码数能被 p 整除。

为什么要约定整数 p 为个位数字不是 5 的奇数呢？因为偶数的积只能是偶数，个位数字为 5 的奇数的乘积个位数字只能是 0 或 5，都不可能为 1。

（1）竖式除法模拟设计要点。

设整数竖式除法每次试商的被除数为 a，除数为 p（即给定正整数），每次试商余数为 c。

以余数 c≠0 作为条件设置条件循环，循环外赋初值："c＝1；n＝1；"或"c＝11；n＝2；"等。

被除数 a＝c＊10＋1，试商余数 c＝a％p。每商一位，设置统计 1 的个数的变量 n 增加 1。

若余数 c＝0，结束循环，输出结果。

否则，继续下一轮试商，直到 c＝0 为止。

（2）程序设计。

```
//探求整数至少多少个 1 能被指定整数 p 整除
#include <stdio.h>
void main()
{  int a,c,p,n;
   printf("  请输入整数 p: "); scanf("%d",&p);
   if(p%2==0 || p%10==5)
     { printf("  不存在同码数整除%d。",p); return; }
   n=1; c=1;                                          //确定初始值 n,c
   while(c!=0)
     { a=c*10+1; c=a%p; n++;}                         //实施除乘竖式计算模拟

   printf("  至少需%d 个 1 的整数才能被%d 整除。\n",n,p);
}
```

（3）程序运行示例与说明。

```
请输入整数 p: 2019
至少需 672 个 1 的整数才能被 2019 整除。
```

输出结果是至少需 672 个 1 的整数才能被 2019 整除，靠人工实施竖式除法，想得出

这一结果还是比较繁复的。

这还不算，试试“至少需多少个 1 的整数才能被 2017 整除”，就更能切身体会到人工计算与程序计算的效益差别。

5.2.2　同码数求和

本节探索 n 个同码数求和，包括同码整数求和与同码小数求和。设和式

$$s(d,n)=d+dd+ddd+\cdots+dd\cdots d(n\text{ 个 }d) \tag{5-2-2}$$

$$f(d,n)=0.d+0.dd+0.ddd+\cdots+0.dd\cdots d(\text{小数点后 }n\text{ 个 }d) \tag{5-2-3}$$

式(5-2-2)为 n 个同码数码 d 整数之和，和式中第 k 项有 k 个数字 $d(d=1,2,\cdots,9)$。

例如，$s(3,5)=3+33+333+3333+33333$。

式(5-2-3)为 n 项同码数码 d 小数之和，其中第 k 项小数点后有连续 k 个数字 $d(d=1,2,\cdots,9)$。

例如，$f(7,4)=0.7+0.77+0.777+0.7777$。

【问题 1】　试求同码整数和 $s(3,30)$。

【求解】　引入中间量 $s(9,30)$用以简化求和。

引入中间量 $s(9,30)$是有趣的，因为 $s(9,30)$与 $s(3,30)$直接相关，而 $s(9,30)$可以直接算出。事实上，由

$$s(3,30)=s(9,30)/9\times 3=s(9,30)/3 \tag{5-2-4}$$

而　$s(9,30)=9+99+\cdots+99\cdots9$

$=(10-1)+(100-1)+\cdots+(10^{30}-1)$

$=11\cdots10-30=11\cdots1080$（最后数中 28 个 1）

$s(3,30)=11\cdots1080/3$　　（被除数中 28 个 1）

注意到 1111/3=370，余数 1；1080/3=360，因而得

$$s(3,30)=\underbrace{370\cdots370}_{9\text{个}370}360 \tag{5-2-5}$$

有趣的是，以上 $s(3,30)$的结果式(5-2-5)中出现 9 个 370 重复节，耐人寻味。

【问题 2】　试求同码小数和 $f(6,30)$。

【求解】　引入中间量 $f(9,30)$用以简化求和。

同样引入中间量 $f(9,30)$，因为 $f(9,30)$与 $f(6,30)$直接相关，而且 $f(9,30)$可以直接算出。由

$$f(6,30)=f(9,30)/9\times 6=f(9,30)\times 2/3 \tag{5-2-6}$$

而

$$f(9,30)=0.9+0.99+\cdots+0.99\cdots9=30-0.11\cdots1(\text{后项小数共连续 30 个 1})$$

则

$$f(6,30)=(30-0.111\cdots1)\times 2/3=(60-0.222\cdots2)/3$$

因而

$$f(6,30)=59.77\cdots78/3=19+2.77\cdots78/3\ (\text{其中后项小数有连续 29 个 7})$$

注意到 2777/3=925，余数为 2；而 2778/3=926，因而得

$$f(6,30)=19.\underbrace{925\cdots925}_{9个925}926$$

同样耐人寻味的是,$f(6,30)$的结果中出现 9 个重复节 925。

【编程拓展】 探求同码整数和 s(d,n)与同码小数和 f(d,n)。

输入整数 d(1≤d≤9),及整数 n(1<n≤100),输出 s(d,n)与 f(d,n)。

1. 设计要点

设置 s 数组存储和 s(d,n),s[1]存储个位数字,s[2]存储十位数字,以此类推。

设置 f 数组存储和 f(d,n),f[0]存储小数和的整数部分,f[1]存储小数点后第 1 位,f[2] 存储小数点后第 2 位,……,f[n]存储小数点后第 n 位。

(1) 整数求和。

整数和式中共有 n 位数求和,第 i 个数有 i 位(i=1,2,…,n),每位数字为 d。

设置 i(1～n)循环,循环 n 次,实施 n 个数相加:i=1 时,个位有 n 个 d 相加,其值为 n * d;i=2 时,十位有 n－1 个 d 相加,其值为(n－1) * d;一般第 i 位,有 n－i＋1 个 d 相加,其值为(n－i＋1) * d。

完成相加后,还需从个位开始,逐位实施进位(j=1～n－1):

```
s[j+1]=s[j+1]+s[j]/10;s[j]=s[j]%10;
```

整数输出当然是从高位开始,逐位向低位输出。

(2) 小数求和。

同样设置 i(1～n)循环,循环 n 次,实施 n 个数相加:i=1 时,小数点后第 1 位有 n 个 d 相加,其值为 n * d;i=2 时,小数点后第 2 位有 n－1 个 d 相加,其值为(n－1) * d;一般小数点后第 i 位,有 n－i＋1 个 d 相加,其值为(n－i＋1) * d。

以上相加就在一个循环中实现。完成 n 个相加后,还需从小数点后第 n 位开始,逐位实施进位(j=n～1):

```
f[j-1]=f[j-1]+f[j]/10;f[j]=f[j]%10;
```

最后得到的 f[0]即为小数和的整数部分,其值可能为 1 位,也可能为多位。

小数和输出,最先输出整数部分 f[0]加带小数点,然后从小数点第 1 位开始,逐位输出到小数点后第 n 位。

2. 同码数求和程序设计

```
//求整数和 s(d,n)=d+dd+ddd+…+dd…d(n 个 d)
//求小数和 f(d,n)=0.d+0.dd+0.ddd+…+0.dd…d(n 个 d)
#include <stdio.h>
void main()
{  int d,i,j,n,s[5000],f[5000];
   printf("   请输入整数 d,n: ");  scanf("%d,%d",&d,&n);
   for(j=0;j<=n;j++) s[j]=f[j]=0;
   for(i=1;i<=n;i++)
     s[i]=f[i]=(n-i+1) * d;                    //第 i 位共 n+1-i 个 d 之和
```

```
    for(j=1;j<=n-1;j++)                          //加完 n 个整数后统一进位
      { s[j+1]=s[j+1]+s[j]/10;s[j]=s[j]%10;}
    printf("  s(%d,%d)=",d,n);
    for(j=n;j>=1;j--)                            //从高位开始逐位输出和 s(d,n)
      printf("%d",s[j]);
    for(j=n;j>=1;j--)                            //加完 n 个小数后统一进位
      { f[j-1]=f[j-1]+f[j]/10;f[j]=f[j]%10;}
    printf("\n  f(%d,%d)=%d.",d,n,f[0]);
    for(j=1;j<=n;j++)                            //从小数点后第一位开始逐位输出和
      printf("%d",f[j]);
    printf(" \n");
}
```

3. 程序运行示例与说明

```
请输入整数 d,n: 7,30
s(7,30)=864197530864197530864197530840
f(7,30)=23.246913580246913580246913580247
```

从上述整数求和结果看，864197530 这一“重复节”在和中重复 3 次，只有最后 3 位 840 不属于重复节。

从上述小数求和结果看，小数点后除了尾部 3 位 247 之外，其余是 246913580 这一“重复节”，重复 3 次。

同码数求和结果中的“重复节”与循环小数的“循环节”有类似之处，不同的是重复节往往在前面，而循环节通常在后面。

5.3　统计的智慧

本节探讨巧妙建模、三角网格与交通方格网 3 个有代表性的统计案例，其统计思路与方法颇为新颖。

5.3.1　巧妙建模

本案例探求一个线性不定方程的解有多少组，应用建模简化了统计过程。

【问题】　对于不定方程 $x+y+z=15$，试统计：

(1) x,y,z 为非负整数解的组数；

(2) x,y,z 为正整数解的组数；

(3) x,y,z 满足 $x\geqslant -3,y\geqslant 2,z\geqslant 5$ 的整数解的组数。

【建模巧解】　拟建立投球统计模型，转化为组合计算。

(1) 把问题转化为 15 个相同的小球投放到 3 个分别标有 x,y,z 标签的盒子中的不同的投放种数。然后，把 3 个盒子“抽象”为并排设置的 2 块隔板，这 2 块隔板划分的 3 个区域相当于 3 个盒子，即 x,y,z 变量。于是把 15 个小球与这 2 块隔板进行排列，每种不

同的排列对应一种投球结果,即对应不定方程 $x+y+z=15$ 的一组解。排列种数等于15+2个位置中挑选出2块隔板位置的组合数 C(15+2,2)。因而 x,y,z 为非负整数解的组数为 C(17,2)=136。

(2) 若 x,y,z 为正整数,相当于每个盒子至少要投放一个小球,不妨每个盒子先放一个小球。然后 15−3=12 个小球与2块隔板进行排列,共有组数即为组合数 C(12+2,2)。因而 x,y,z 为正整数解的组数为 C(14,2)=91。

(3) 若 x,y,z 满足 $x\geqslant -3,y\geqslant 2,z\geqslant 5$,令 $a=x+3,b=y-2,c=z-5$,则由 $x+y+z=15$ 得 $a+b+c=11(a,b,c\geqslant 0)$,由上可知这一方程的非负整数解组数为组合数 C(11+2,2)。因而 x,y,z 满足 $x\geqslant -3,y\geqslant 2,z\geqslant 5$ 的整数解的组数为 C(13,2)=78。

【编程拓展】

对于给定的正整数和 s,对于3个变量 x,y,z 的不定方程 $x+y+z=s$,试统计 x,y,z 取自区间 $[c,d](0\leqslant c<d,3c\leqslant s)$ 整数解的组数。

(1) 设计要点。

建立 x,y,z 循环枚举区间 $[c,d]$ 上的所有整数,如果满足条件 $x+y+z=s$ 即输出方程的解并用 n 实施统计。

注意到3个变量 x,y,z 没有大小约定,因而循环区间都是 $[c,d]$。

(2) 程序设计。

```
//不定方程 x+y+z=s 求解与统计
#include <stdio.h>
#include <math.h>
void main()
{  int c,d,n,s,x,y,z;
   printf("  请确定各变量起点 c 与终点 d:"); scanf("%d,%d",&c,&d);
   printf("  请确定和 s(3c<s<3d): "); scanf("%d",&s);
   n=0;
   for(x=c;x<=d;x++)                              //设置三重循环枚举 x,y,z
   for(y=c;y<=d;y++)
   for(z=c;z<=d;z++)
   if(x+y+z==s)                                   //满足条件时统计并输出解
   {  n++;
      printf("   x=%d,y=%d,z=%d",x,y,z);
      if(n%3==0)  printf("\n");
   }
   printf("\n   共%d组解。\n",n);
}
```

(3) 程序运行示例与说明。