

精巧求解剖析

精巧求解与高精计算,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、同码数的整除与求和规律的探索,到建模统计、图形分类统计到数据转换,突出一个“巧”字。

同时,从探索最小 0-1 串积、指定多码串积到尾数前移,到多规格万年历构建,突显一个“精”字。

探索并拓展著名的梅齐里亚克砝码问题、拉马努金无穷嵌套平方根式与伯努利装错信封的排列问题,无疑是“巧”与“精”结合的典范。

飘逸于数学殿堂的两个神奇“幽灵”自然常数 e 与圆周率 π ,凝聚着数学的博大精深,彰显出编程的非凡卓著。

5.1 互积和与尾数前移

本节探讨简单的互积和与精妙的尾数前移,案例的趣味性颇强,求解技巧很有启发性。

5.1.1 互积和巧算

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: $s_3 = (-1 - 2 - 3)(s_1 + 31) = -6s_1 - 6 \times 31$ 。

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

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

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 $a_1, a_2, \dots, a_8$, 构造多项式

$$f(x) = (x + a_1)(x + a_2) \cdots (x + a_8) = x^8 + p_1 x^7 + p_2 x^6 + \cdots + p_7 x + p_8 \quad (5-1-1)$$

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; $\dots$; 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, $\dots$, 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, $\dots$, 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 $a[1], a[2], \dots, a[n]$ 中。

同时, 构造关于这 n 个整数的 n 次多项式

$$y(x) = (x + a_1)(x + a_2) \cdots (x + a_n) = x^n + p_1 x^{n-1} + p_2 x^{n-2} + \cdots + p_{n-1} x + p_n \quad (5-1-2)$$

其中, p_1 为 n 数之和; p_2 为任意 2 个数积之和; $\dots$; 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

$$m = y(1) - p_1 - 1 \quad (5-1-3)$$

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%d 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%d 个整数为%d",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%d 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”难点转换为在循环中求一个积,也非常精巧。

5.1.2 精妙的尾数前移

《数学通报》曾发表过以下新颖的计算趣题。

整数 n 的尾数是 9, 把尾数 9 移到其前面(成为最高位)后所得的数为原整数 n 的 3 倍, 原整数 n 至少为多大?

这一趣题开启了“尾数前移”趣味计算的先河。

第 4 届国际中学生数学竞赛也有以下较为简单的尾数前移计算趣题。

【问题】 求性质如下的最小自然数 n : 用十进制表示时, 它的最后一位数字是 6。将这个数字 6 移到其余数字的最前面, 所得的新数是原数的 4 倍。

【探求】 尝试应用实施竖式除法与竖式乘法两种方法求解。

(1) 实施竖式除法。

① 开始时被除数为 6(即前移的尾数字), 除数为 4(即前移后的倍数)。

② 每试商所得的商(一个数字)即作为被除数的下一个数字。如图 5-1(a)所示, 商的第一个数字 1 移作被除数的第 2 个数字(见图 5-1(a)中的斜线), 商的第 2 个数字 5 移作被除数的第 3 个数字, 以此类推。

③ 除法结束的条件: 所得的商为 6, 即原整数 n 的尾数; 同时试商的余数为 0。

(2) 实施竖式乘法。

① 开始时所求整数 n 的个位数字为 6(即尾数字), 另一个乘数为 4(即倍数)。

② 每试乘所得的积的个位数字作为 n 的高一位数字。如图 5-1(b) 中箭头所示, 积的个位数字 4 移作 n 的十位数字, 积的十位数字 8 移作 n 的百位数字, …… , 以此类推。

③ 乘法结束的条件: 所得的积只为一个数字 (即没有进位), 且这个数字为 6, 即原整数 n 的尾数。

答案: $n=153846$ 。这就是第 4 届国际中学生数学竞赛尾数前移趣题的答案。

【编程拓展】

以上尾数前移问题的尾数为 1 位, 事实上可把前移的尾数拓展为多位。

整数 n 的尾数 q (可为多位) 移到 n 的前面所得的数为 n 的 p 倍, 记为 $n(q, p)$ 。这里正整数 p 不大于前移尾数 q 的首位。

对于指定的尾数 q 与倍数 p , 求解 $n(q, p)$ 。

1. 模拟设计要点

试应用模拟竖式除法设计探求。

设置 d, e 数组, 存储所求 n 及指定尾数 q 的各位数字。

尾数前移后为 n 的 p 倍, 即前移后的整数能被 p 整除, 以此实施模拟竖式除探求整数 n 的各位数。

(1) 应用逐位求余求得尾数 q 的位数 k 及各位数字 $d[j] (j=k, \dots, 1)$ 。

(2) 对 q 的 k 位实施模拟竖式除法, 求得 n 的前 k 位。

(3) 设计模拟竖式除法循环, 求出 n 的第 $i+k$ 位 ($i=1, 2, \dots$)。

循环条件为 $c \neq 0 \parallel b \neq q$, 这里 c 为试商的余数, b 为 $d[i+1], d[i+2], \dots, d[i+k]$ 这 k 个数字组成的整数。

当试商余数 c 为 0, 且最后的 k 位数字组成的整数 b 等于尾数 q 时, 终止探索, 输出 d 数组的共 $i+k$ 位, 即所求的 n 。

2. 程序设计

```
//模拟除竖式求解多位尾数前移
#include<stdio.h>
void main()
{ int a,b,c,i,j,k,p,q,x,d[10000],e[10000];
  printf(" 请输入尾数(可多位)q,倍数p:"); scanf("%d,%d",&q,&p);
  k=0;x=q;
  while(x>0)
    { k++;d[k]=x%10;x=x/10;} //求尾数q的位数k及各位数字
  if(d[k]<p)
    { printf(" 问题无解!"); return;}
  printf(" n(%d,%d)=",q,p);
  for(j=1;j<=k;j++) e[j]=d[k+1-j];
  for(c=0,j=1;j<=k;j++)
    { a=c*10+e[j];d[j]=a/p;c=a%p;} //d[j]为所求n的第j位数字
  i=b=0;
```

$$\begin{array}{r}
 1\ 5\ 3\ 8\ 4\ 6 \\
 4 \overline{) 6\ 1\ 5\ 3\ 8\ 4} \\
 \underline{4} \\
 2\ 1 \\
 \underline{2\ 0} \\
 1\ 5 \\
 \underline{1\ 2} \\
 3\ 3 \\
 \underline{3\ 2} \\
 1\ 8 \\
 \underline{1\ 6} \\
 2\ 4 \\
 \underline{2\ 4} \\
 0
 \end{array}$$

(a) 除法

$$\begin{array}{r}
 1\ 5\ 3\ 8\ 4\ 6 \\
 \times) \\
 6\ 1\ 5\ 3\ 8\ 4
 \end{array}$$

(b) 乘法

图 5-1 实施竖式除法与竖式乘法示意图

```

while(c!=0 || b!=q)                                //试商循环, c=0 且 b=q 时结束
{
    i++; a=c*10+d[i];
    d[i+k]=a/p; c=a%p;                               //模拟整数竖式除法计算
    for(b=0, j=1; j<=k; j++)
        b=b*10+d[i+j];                               //b 为所求 n 的后 k 位数
}
for(j=1; j<=i+k; j++)
    printf("%d", d[j]);                               //输出所求 n 及其位数
printf("\n 共有%d 位。 \n", i+k);
}

```

3. 程序运行示例与说明

```

请输入尾数(可多位)q, 倍数 p: 9, 3
n(9, 3)=3103448275862068965517241379
共有 28 位。
请输入尾数(可多位)q, 倍数 p: 31, 2
n(31, 2)=15577889447236180904522613065326633165829145728643
          2160804020100502512562814070351758793969849246231
共有 99 位。

```

以上所得 $n(9, 3)$ 即为《数学通报》上所提问题的答案, 把 $n(9, 3)$ 的尾数 9 前移, 所得整数即为 $n(9, 3)$ 的 3 倍。

所得 $n(31, 2)$ 达 99 位, 还可得 $n(2019, 2)$ 达 408 位数。以上所求的数为高精度数, 应用模拟竖式除法得到快捷解决, 显出程序设计的优势。

注意: 多位尾数包含 1 位在内, 即多位尾数前移程序同样可求解 1 位尾数前移问题。

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 $n \leq 2019$, 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 $0, 1, \dots, 2018$ 。

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 $1, 2, \dots, 2018$ 。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 $1 \leq n_1 < n_2 \leq 2019$, 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

$$m = m_2 - m_1 = \overbrace{11 \cdots 1}^{n_2 - n_1} \overbrace{00 \cdots 0}^{n_1} = \overbrace{11 \cdots 1}^{n_2 - n_1} \times \overbrace{100 \cdots 0}^{n_1} \quad (5-2-1)$$

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 $1 \leq n_2 - n_1 < 2019$,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 $n \leq 2019$,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 5-2 所示。

$$\begin{array}{r} 2019 \overline{) 111111111 \cdots 1} \\ \underline{10095} \\ 10161 \\ \underline{10095} \\ 6611 \\ \underline{6057} \\ 5541 \\ \underline{} \\ 0 \end{array}$$

图 5-2 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 $n \leq 2019$ 。至于 n 至少为多大,则必须要等以图 5-2 所示的试商结果出炉。

【编程拓展】 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 $c \neq 0$ 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烦琐的。

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

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

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

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

例如, $s(3, 5) = 3 + 33 + 333 + 3333 + 33333$ 。

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

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 $s(9, 30)$ 用于简化求和。

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

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

而 $s(9, 30) = 9 + 99 + \cdots + 99 \cdots 9$

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

$$= 11 \cdots + 11 \cdots 1080 (\text{最后数中 28 个 1})$$

$$s(3, 30) = 11 \cdots 1080 / 3 \quad (\text{被除数中 28 个 1})$$

注意到 $1111/3=370$, 余数 1; $1080/3=360$, 因而得

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

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 $f(9, 30)$ 用于简化求和。

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

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

而

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

则

$$f(6,30) = (30 - 0.111\cdots 1) \times 2/3 = (60 - 0.222\cdots 2)/3$$

因而

$$f(6,30) = 59.77\cdots 78/3 = 19 + 2.77\cdots 78/3 \quad (\text{其中后项小数有连续 29 个 7})$$

注意到 $2777/3=925$, 余数为 2; 而 $2778/3=926$, 因而得

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

同样耐人寻味的是, $f(6,30)$ 的结果中出现 9 个 925 重复节。

【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 $d(1 \leq d \leq 9)$ 及整数 $n(1 < n \leq 100)$, 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, $\cdots$, $f[n]$ 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 ($i=1, 2, \cdots, n$), 每位数字为 d 。

设置 $i(1 \sim n)$ 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 ($j=1 \sim n-1$):

```
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 $i(1 \sim n)$ 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 ($j=n \sim 1$):

```
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”在和中共重复 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

统计求和是趣味数学中不可或缺的基本内容,也是编程拓展的重要课题。

本节探讨区间数位和、统计方格图中的正方形个数与创新统计建模等 3 个有代表性的统计案例。统计对象有数字、也有图形,更有建模,类型丰富,内容新颖,思维开放,技巧创新,充分体现统计的智慧之光。

5.3.1 数位和与数字频数

本节探讨指定区间中所有整数的数位和,以及指定数字出现的频数。

先看两个统计的具体实例。

【问题 1】 试求区间 $[100,2026]$ 中所有整数的各位数字之和(简称为数位和)。

【思考】 该区间范围较大,有 3 位数,也有 4 位数,逐个累加将会非常烦琐。拟分为以下几个子区间首尾配对,可望简化统计求解。

(1) 区间 $[0,1999]$ 中,首尾配对共 1000 对:

$$0+1999, 1+1998, \dots, 999+1000$$

每对求和不进位,每对数位和 $1+3 \times 9 = 28$,共计数位和为: $28 \times 1000 = 28000$ 。

(2) 需去除区间 $[0,99]$,该区间首尾配对共 50 对:

$$0+99, 1+98, \dots, 49+50$$

每对求和不进位, 每对数位和为 18, 共计数位和为: $18 \times 50 = 900$ 。

(3) 区间[2000, 2019]中, 首尾配对共 10 对:

$$2000+2019, \dots, 2009+2010$$

每对求和不进位, 每对数位和 $2 \times 2 + 1 + 9 = 14$, 共计数位和为: $14 \times 10 = 140$ 。

(4) 区间[2020, 2026]中, 首尾配对共 7/2 对:

$$2020+2026, 2021+2025, 2022+2024, \text{半对 } 2023$$

每对求和不进位, 每对数位和 $4 \times 2 + 6 = 14$, 共计数位和为: $14 \times 7/2 = 49$ 。

(5) 累加: $28000 - 900 + 140 + 49 = 27289$ 。

统计结果: 区间[100, 2026]中所有整数的各位数字之和为 27289。

【问题 2】 试分别求在区间[100, 999]的所有整数中, 数字“8”与数字“0”各出现多少次?

【思考】 为防止统计出现重复或遗漏, 分数字出现在个位、十位与百位这 3 类实施统计。

(1) 个位数字出现“8”的频数。

个位数字为“8”时, 高 2 位为 10~99, 共 90 个, 即个位数字“8”出现 90 次。

(2) 十位数字出现“8”的频数。

十位数字为“8”时, 百位数字为 1~9, 对应每个百位数字的个位数字为 0~9, 相乘即得十位数字为“8”出现 90 次。

(3) 百位数字出现“8”的频数。

百位数字为“8”时, 低 2 位为 00~99, 共 100 个, 即百位数字“8”出现 100 次。

以上 3 项相加, 得区间[100, 999]的所有整数中数字“8”出现的频数为

$$90 + 90 + 100 = 280$$

把数字“8”改为数字“0”, 显然第 3 项百位为“0”不成立, 而前两项相同, 即得区间[100, 999]的所有整数中数字“0”出现 180 次。

统计结果: 区间[100, 999]的所有整数中, 数字“8”出现 280 次, 数字“0”出现 180 次。

【编程拓展】 探求指定区间[m, n]上的数位和 s, 统计区间上所有整数的数字总数, 及其中数字 j(0~9)出现的频数。

1. 编程要点

(1) 设置数组 p[j](j: 0~9)统计数字 j 的个数。

变量 s1 统计数位和, 同时设置 s2 为 $j * p[j]$ 求和, 通过 $s1 = s2$ 检验数位和;

变量 t1 统计数字个数, 同时设置 t2 为 $p[j]$ 求和, 通过 $t1 = t2$ 检验数字个数。

(2) 设计的关键在于把区间内的整数分解出它的各个数字。

注意到整数 k 的位数是变化的, 可通过以下条件循环实现数字分离:

```
while(k>0)
{ d=k%10; s1=s1+d; k=k/10;          //分离出数字 d 后, 即累加到数位和 s1
  t1++; p[d]++;                      //统计所有数字个数 t1 及数字 d 的个数
}
```

(3) 程序设计有两道检验:

```
for(j=0; j<=9; j++)
{ s2=s2+j * p[j]; t2=t2+p[j]; }    //另法统计数位和 s2 与数字总计 t2
```