第5章 递推法

5.1 算法设计思想

递推法是利用所求解问题本身具有的性质(递推关系)来求问题解的有效方法。

具体做法是根据该问题 $N=n$ 之前的一步($n-1$)或多步($n-1$,$n-2$,$n-3$,…)的结果推导出 n 时的解：$f(n)=F(f(n-1),f(n-2),\cdots)$(称为递推关系式)；而 $N=0,1,\cdots$ 的初值 $f(0),f(1),\cdots$ 往往是直接给出或直观得出的。

递推算法的关键问题是得到相邻的数据项之间的关系，即递推关系。递推关系是一种高效的数学模型，是递推应用的核心。递推关系不仅在各数学分支中发挥着重要的作用，而且由它体现出来的递推思想在各学科领域中更是显示了独特的魅力。

在求解具体问题时，$N=n$ 的解到底与其前面的几步结果相关必须明确。假设是两步，那么在计算的过程中只需记住这前两步的结果 $R1$、$R2$，下一步的结果 R 可以由这前两步的结果推导得到，即有 $R=F(R1,R2)$，接下来进行递推：$R1=R2$，$R2=R$，向后传递，为求下一步的结果做好准备。这也正是递推法名字的由来。若问题与前三步相关，则在计算的过程中需要记住前三步的结果 $R1$、$R2$、$R3$，下一步的结果 R 可由这前三步的结果推导得到，即有 $R=F(R1,R2,R3)$，接下来进行递推：$R1=R2$，$R2=R3$，$R3=R$，向后传递。

递推法的一般步骤如下。

(1) 确定递推变量。递推变量可以是简单变量，也可以是一维或多维数组。

(2) 建立递推关系。递推关系是递推的依据，是解决递推问题的关键。

(3) 确定初始(边界)条件。根据问题最简单情形的数据确定递推变量的初始(边界)值，这是递推的基础。

(4) 控制递推过程。递推过程控制通常由循环结构实现，即递推在什么时候进行，满足什么条件结束。

递推法可分为正推法和倒推法两种。其中，正推法是一种简单的递推方式，是从小规模的问题推解出大规模问题的一种方法，也称为“正推”；倒推法则是对某些特殊问题所采用的不同于通常习惯的一种方法，即从后向前进行推导，实现求解问题的方法。

5.2 典型例题

5.2.1 兔子繁殖问题

1. 问题描述

一对兔子从出生后第三个月开始，每月生一对小兔子。小兔子长到第三个月又开始生下一代小兔子。假若兔子只生不死，一月份抱来一对刚出生的小兔子，问一年中的每个月各有多少对兔子。

2. 问题分析

寻找问题的规律性，需要通过对现实问题的具体事例进行分析，从而抽象出其中的规律。

一对兔子从出生后第三个月开始每月生一对小兔子，第三个月以后每月除上个月的兔子外，还有新生的小兔子，在下面用加号后面的数字表示。则第三个月以后兔子的对数就是前两个月兔子对数的和。过程如下：

1月　2月　　3月　　4月　　5月　　6月　…

1　　1　　1＋1＝2　2＋1＝3　3＋2＝5　5＋3＝8　…

3. 算法说明

算法说明参见表5-1。

表　5-1

类　型	名　称	代表的含义
算法	rabbit()	递推法求解兔子繁殖问题
变量	a	当前月的前两个月的兔子对数
变量	b	当前月的前一个月的兔子对数
变量	c	当前月份的兔子对数

4. 算法设计

```
#include "stdio.h"
void rabbit()
{
    int i,c,a=1,b=1;
    printf("%d,%d,",a,b);
    for(i=1;i<=10;i++)
    {
```

```
        c=a+b;                          /* 斐波那契数列规律,第三项等于前两项之和 */
        printf("%d,",c);
        a=b;                            /* 向后递推 */
        b=c;
    }
}
void main()
{
    printf("一年中每个月兔子对数如下:\n");
    rabbit();
}
```

5. 运行结果

```
一年中每个月兔子对数如下:
1,1,2,3,5,8,13,21,34,55,89,144,
```

6. 算法优化

1）优化说明

1	2	3	4	5	6	7	8	…
a	b	$a=a+b$	$b=b+a$	$a=a+b$	$b=b+a$	$a=a+b$	$b=b+a$	…

因此，可以归纳出用“$a=a+b$，$b=b+a$”做循环不变式，这样一来，循环其实是递推了 2 步，循环次数自然减少。

2）算法说明

算法说明参见表 5-1。

3）算法设计

```
#include "stdio.h"
void rabbit()
{
    int i,a=1,b=1;
    printf("%d,%d,",a,b);
    for(i=1;i<=5;i++)
    {
        a=a+b;
        b=b+a;
        printf("%d,%d,",a,b);
    }
}
void main()
{
    rabbit();
}
```

5.2.2 最大公约数问题

1. 问题描述

给出任意两个正整数，求它们的最大公约数。

2. 问题分析

求两个数的最大公约数，最简单的方法就是用最大公约数的定义来求。

设 m，n 为两个正整数，d 取 m 和 n 中较小的一个，若 $d \neq 0$，则判断 d 是否同时整除 m，n，如果是，则 d 为最大公约数；否则，$d = d - 1$ 重复上述过程，直到满足为止。因为最后 $d = 1$ 时，一定会满足条件。

3. 算法说明

算法说明参见表 5-2。

表 5-2

类　型	名　称	代表的含义
算法	f(int m,int n)	由定义求最大公约数
形参变量	m,n	输入的两个数
变量	d	保存最大公约数

4. 算法设计

```
#include "stdio.h"
int f(int m,int n)
{
    int d;
    if(m<n)
        d=m;
    else
        d=n;
    while(d>0)
    {
        if(m%d==0 && n%d==0)
            return(d);
        d--;
    }
}
void main()
{
    int m,n;
    printf("请输入两个正整数:");
    scanf("%d%d",&m,&n);
    printf("最大公约数为:%d\n",f(m,n));
}
```

5. 运行结果

```
请输入两个正整数:12 3
最大公约数为: 3
```

6. 算法优化

1）优化说明

下面应用递推法求解此问题。

在数学中，求最大公约数有一个很有名的方法叫辗转相除法，辗转相除法体现了递推法的基本思想。

设 m，n 为两个正整数，且 n 不为零，辗转相除法的过程如下。

(1) 将问题转化为数学公式，即 $r=m\%n$，r 为 m 除以 n 的余数。

(2) 若 $r=0$，则 n 即为所求的最大公约数，输出 n。

(3) 若 $r!=0$，则令 $m=n$，$n=r$，继续递归，再重复前面的(1)和(2)步骤。

其中第(3)步即为递推部分。

2）算法说明

算法说明参见表 5-3。

表　5-3

类　　型	名　　称	代表的含义
算法	f(int m,int n)	辗转相除法求最大公约数
形参变量	m,n	输入的两个数
变量	r	两个数相除的余数

3）算法设计

```
#include "stdio.h"
int f(int m,int n)
{
    int r=n;
    while(r!=0)
{
    r=m%n;                          /* 利用 r=m%n,将余数赋给 r */
    m=n;
    n=r;
}
    return m;
}
void main()
{
    int m,n;
    printf("Input two numbers:");
    scanf("%d%d",&m,&n);
```

```
    printf("%d\n",f(m,n));
}
```

5.2.3 猴子吃桃问题

1. 问题描述

一只小猴子摘了若干个桃子，每天吃现有桃子的一半多一个，到第 10 天时只剩一个桃子，求小猴子最初摘了多少个桃子？

2. 问题分析

这道题可以用倒推法来解决。因为猴子每天吃的桃子数取决于前一天的桃子数，所以可以用一个递推变量代替桃子数。设 a 为递推变量，代表今天剩下的桃子数，那么就可以推导出昨天剩下的桃子数，即 $a=(a+1)\times 2$，这就是本题的递推关系式，找到这个关系式，问题基本就解决了。

3. 算法说明

算法说明参见表 5-4。

表 5-4

类　型	名　称	代表的含义
算法	tao()	递推法求解猴子吃桃问题
变量	a	桃子数
变量	i	天数

4. 算法设计

```
#include "stdio.h"
void tao()
{
    int i,a;
    a=1;
    for(i=9;i>=1;i--)                    /*利用倒推法求解*/
        a=(a+1)*2;
    printf("sum=%d\n",a);
}
void main()
{
    tao();
}
```

5. 运行结果

```
sum=1534
```

6. 算法优化

1）优化说明

这道题不但可以用倒推法解决，还可以用递归法求解。

由于猴子每天吃的桃子数取决于前一天的桃子数，因此可定义一个递归函数来求桃子数，设 tao(n)代表第 n 天剩下的桃子数，则有递归关系式：

$$\text{tao}(n)=(\text{tao}(n+1)+1)\times 2 \quad \text{当 } n=1,2,3,\cdots,9 \text{ 时}$$

递归终止条件是：当 $n=10$ 时，tao(n)＝1。

2）算法说明

算法说明参见表 5-5。

表　5-5

类　　型	名　　称	代表的含义
算法	tao(int n)	求解猴子吃桃问题算法
形参变量	n	天数

3）算法设计

```
#include "stdio.h"
int tao(int n)
{
    if (n==10) return 1;
    else
    return (tao(n+1)+1) * 2;
}
void main()
{
    printf("sum=%d\n",tao(1));
}
```

5.2.4　杨辉三角形问题

1. 问题描述

输出如图 5-1 所示的杨辉三角形(打印出 10 行)。

2. 问题分析

由图 5-1 所示的杨辉三角形可以看出，中间的数据等于其上一行左上、右上的数据和，第 i 层有 i 列需要求解 i 个数据。可以用二维数组 array[][]存储杨辉三角形。

为了便于表示，可以将杨辉三角形变一下形状，即从第一列开始放置，则可得到一个下三角矩阵，而且很有规律：第一列都为 1，主对角线都为 1，从第三行起，中间(除第一个和对角线之外)位置元素的值等于其上一行对应位置元素及其前一个元素之和，这就是从当前行

1
1　1
1　2　1
1　3　3　1
1　4　6　4　1
…

图　5-1

推导到下一行的递推关系式，由此便可求出杨辉三角形的任一行。

3. 算法说明

算法说明参见表 5-6。

表 5-6

类　型	名　称	代表的含义
算法	yanghui()	递推法求解杨辉三角形问题
二维数组	array	存储杨辉三角形
变量	i	行
变量	j	列

4. 算法设计

```
#include "stdio.h"
void yanghui()
{
    int array[10][10],i,j;
    for(i=0;i<10;i++)
    {
        array[i][i]=array[i][0]=1;
        for(j=1;j<=i;j++)
            array[i+1][j]=array[i][j]+array[i][j-1];
    }
    for(i=0;i<10;i++)
    {
        for(j=0;j<=i;j++)
        {
            printf("%5d",array[i][j]);
            printf(" ");
        }
    printf("\n");
    }
}
void main()
{
    yanghui();
}
```

5. 运行结果

```
1
1    1
1    2    1
1    3    3    1
1    4    6    4    1
1    5   10   10    5    1
1    6   15   20   15    6    1
1    7   21   35   35   21    7    1
1    8   28   56   70   56   28    8    1
1    9   36   84  126  126   84   36    9    1
```

6. 算法优化

1）优化说明

前面以二维数组为存储，使用递推法求解杨辉三角形问题，那么，是否可以用一维数组为存储来解决本问题呢？回答当然是肯定的。

使用一维数组时注意以下两个问题：一是不能保存完整的杨辉三角形，只能求出一行输出一行；二是在利用递推法从上一行推导下一行时，如果按照通常从前向后推导的话，则原有已知数据将被覆盖，怎么办？只好倒过来从后向前逐项计算，这也是一种倒推法。

下面举例说明。设有一维数组 $a[]$，现已存储第三行数据，则有 $a[1]=1$，$a[2]=2$，$a[3]=1$，那么，下一行的求值顺序是：先求 $a[4]$，已知所有数组元素初值均为 0，因此，$a[4]=a[3]+a[4]=1+0=1$；再求 $a[3]=a[2]+a[3]=2+1=3$；以此类推。

2）算法说明

算法说明参见表 5-7。

表　5-7

类　　型	名　　称	代表的含义
算法	yanghui()	递推法求解杨辉三角形问题
一维数组	a	存储杨辉三角形
变量	i	杨辉三角形的行数
变量	j	数组下标

3）算法设计

```
#include "stdio.h"
void yanghui()
{
    int n,i,j,a[20];
    printf(" 1\n");
    a[1]=a[2]=1;
    printf(" %-4d  %-4d\n",a[1],a[2]);
    for(i=3;i<=10;i++)
    {
        a[1]=a[i]=1;
        for(j=i-1;j>1;j--)
            a[j]=a[j]+a[j-1];
        for(j=1;j<=i;j++)
            printf(" %-4d ",a[j]);
        printf("\n");
    }
}
void main()
{
    yanghui();
}
```

5.2.5 伯努利装错信封问题

1. 问题描述

某人写了 n 封信，这 n 封信对应有 n 个信封。求把所有的信都装错了信封的情况共有多少种？

这是组合数学中有名的错位问题。著名数学家伯努利(Bernoulli)曾最先考虑此题。后来，欧拉对此题产生了兴趣，称此题是“组合理论的一个妙题”，独立地解出了此题。

2. 问题分析

为叙述方便，把某一元素在自己相应位置(如“2”在第2个位置)称为在自然位；某一元素不在自己相应位置称为错位。

事实上，所有全排列分为以下三类。

(1) 所有元素都在自然位，实际上只有一个排列。当 $n=5$ 时，即 12345。

(2) 所有元素都错位。当 $n=5$ 时，如 24513。

(3) 部分元素在自然位，部分元素错位。当 $n=5$ 时，如 21354。

装错信封问题求解实际上是求 n 个元素全排列中的“每一元素都错位”的子集。

当 $n=2$ 时显然只有一个解：21(“2”不在第2个位置且“1”不在第1个位置)。

当 $n=3$ 时，有 231，312 两个解。

通常，可在实现排列过程中加上“限制取位”的条件。

设置一维数组 $a[]$。$a[i]$在 $1\sim n$ 中取值，出现数字相同 $a[j]=a[i]$或元素在自然位 $j=a[j]$时返回($j=1,2,\cdots,n-1$)。

当 $i<n$ 时，还未取 n 个数，i 增1后 $a[i]=1$ 继续；

当 $i=n$ 且最后一个元素不在自然位 $a[n]!=n$ 时，输出一个错位排列，并设置变量 s 统计错位排列的个数。

当 $a[i]<n$ 时，$a[i]$增1继续。

当 $a[i]=n$ 时，回溯或调整，直到 $i=1$ 且 $a[1]=n$ 时结束。

3. 算法说明

算法说明参见表 5-8。

表 5-8

类　型	名　称	代表的含义
算法	bernoulli(int n)	回溯法求解伯努利装错信封问题
形参变量	n	信封个数
一维数组	a	标记信和信封是否错位
变量	s	统计全错位总和

4. 算法设计

```
#include "stdio.h"
int bernoulli(int n)
```