第5章

递推算法

递推算法是一种按照规律分步解决复杂问题的方法，利用计算机速度和不断重复的特点把复杂过程转化为简单过程的多次重复。递归是从小规模问题求解大规模问题的常用工具。本章介绍递推和递归、递推和倒推、递推方程的求解方法和常用问题的递推算法。

视频讲解

5.1 递推算法概述

5.1.1 递推

递推又叫正推，是从小规模的问题推解出大规模问题的一种方法，是迭代算法最基本的表现形式。

例 1：计算数列$\{a_n\}$的前 n 项和。

数列$\{a_n\}$的前 n 项和$S_n=a_1+a_2+\cdots+a_n$。又$S_{n-1}=a_1+a_2+\cdots+a_{n-1}$。所以，递推公式$S_n=S_{n-1}+a_n$成立，在求出前 $n-1$ 项和的基础上推出前 n 项和。

例 2：计算 $n!$。

$n!=1\times2\times\cdots\times n$，又 $(n-1)!=1\times2\times\cdots\times(n-1)$。所以，递推公式 $n!=n(n-1)!$成立，在求出$(n-1)!$的基础上推出 $n!$。

算法的控制结构有顺序、选择、循环和模块调用，模块调用又分为模块间的调用和模块自身的直接与间接调用(递归)。递推经常使用递归和迭代实现，本例的递归和循环实现如下。

递归实现 $F(n)=nF(n-1)=n!$：

输入：n。

输出：$n!$。

```
F(n) {
1.   if(n<2) then
2.        return n
3.   else return n * F(n - 1)
4.   }
```

循环实现 $s=s*i$：

输入：n。

输出：s。

```
1.   s = 1
2.   for i = 2 to n do
3.        s = s * i
4.   return s
```

递归的计算过程是 $F(n)=nF(n-1)=n(n-1)F(n-2)=\cdots=n\times(n-1)\times\cdots\times2\times F(1)$。循环的计算过程是 $s=1\times2$，$s=1\times2\times3$，…，$s=1\times2\times3\times\cdots\times n$，计算的时间复杂度都是 $O(n)$。

5.1.2 递推与递归

直接或间接调用自身的算法称为递归算法。用函数自身给出定义的函数称为递归函数。递归函数有两个要素，分别是边界条件与递归方程。具备这两个要素，才能在有限次计算后得出结果，这也是递归设计的要点。

递归算法的执行过程分递推和回归两个阶段。递推阶段将大规模问题分解为小规模问题求解；回归阶段将小规模问题的解逐级返回，依次计算大规模问题的解，最后返回整个问题的解。

递归一般用于解决如下三类问题。

(1) 数据的定义是按递归定义的，如斐波那契函数、n 的阶乘等。

(2) 问题的解法按递归实现，如回溯算法。

(3) 数据的结构形式是按递归定义的，如二叉树的遍历、图的搜索等。

例 3：一对兔子出生两个月后，每月生一对小兔子。小兔子两个月后又开始生下一代小兔子。假若兔子只生不死，一月初抱来一对刚出生的小兔子，一年中每个月各有多少对兔子？

问题分析：每月的兔子对数如表 5-1 所示，小兔对数加成兔对数等于当月的总对数。从表 5-1 中分析可以得到，当月的兔子对数＝本月成兔对数＋本月小兔对数＝上月的兔子对数＋上上月的兔子对数。因为上月的兔子，不管是成兔还是小兔，当月都是成兔，所以本月成兔对数等于上月兔子对数。上上月的一对兔子，本月都会生一对小兔子，因此本月小兔对数等于上上月兔子对数。

表 5-1 每月的兔子对数

月份	一月	二月	三月	四月	五月	六月	…	十二月
对数	1	1	1＋1＝2	2＋1＝3	3＋2＝5	5＋3＝8	…	?
递推 1	*a*	*b*	*c*	*c*	*c*	*c*	…	*c*
递推 2	*a*	*b*	*c*	*a*	*b*	*c*	…	*c*
递推 3	*a*	*b*	*a*	*b*	*a*	*b*	…	*b*

假设第 n 个月的兔子对数使用 $F(n)$ 表示，则有 $F(n)=F(n-1)+F(n-2)$。$F(n)$ 构成的数列又称为斐波那契数列或兔子序列，该数列在兔子繁殖、上楼方式、树枝、蜂房、声音、花瓣等问题中得到广泛应用。

递推可以很容易使用递归实现，递推式转换为递归方程，边界条件为 $F(0)=0$，$F(1)=1$。

斐波那契数列的递归实现：

输入：n。

输出：$F(n)$。

```
F(n) {
1.   if (n == 0) then return 0
```

```
2.   if (n == 1) then return 1
3.   else return F(n - 1) + F(n - 2)
     }
```

$F(5)$的递归树如图 5-1 所示，节点旁的数字表示递归调用的次序，实箭头线表示递推过程的分解关系，虚箭头线表示回归过程的求值关系，虚箭头线上的数字表示求值结果。算法的实现类似于二叉树生成的过程，因此算法的时间复杂度为 $O(2^n)$。

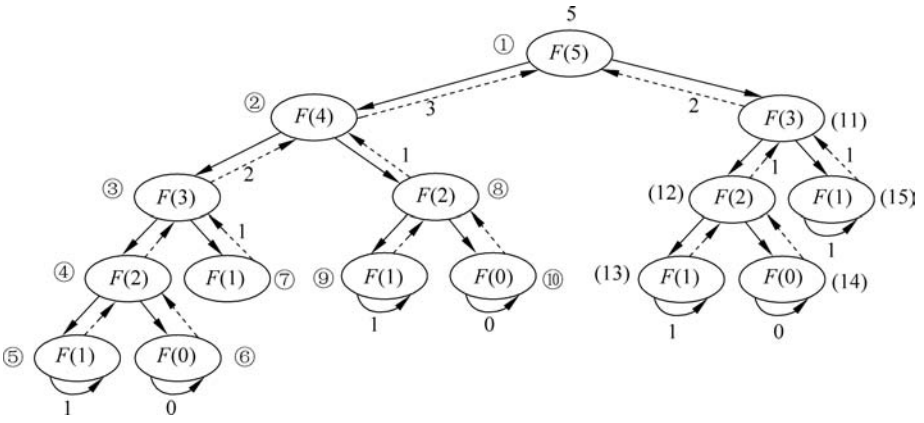

图 5-1　斐波那契数列递归计算过程

斐波那契数列的递推实现：

输入：n。

输出：F。

```
1.   F[0] = 0
2.   F[1] = 1
3.   for i = 2 to n do
4.        F[i] = F[i - 1] + F[i - 2]
5.        print(F[i])
```

算法的时间复杂度为 $O(n)$。实际上，斐波那契数列可以直接推导出结果，即

$$F(n)=\frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n-\left(\frac{1-\sqrt{5}}{2}\right)^n\right]$$

递推与递归的比较如下。

(1) 递归是从问题的最终目标出发，逐渐将复杂问题化为简单问题，先解决简单问题，然后回溯求得问题的解。递推是从简单问题出发，一步一步向前发展，最终求得问题解。

(2) 递归是逆向的，递推是正向的。

(3) 递归表现为自己调用自己，递推则没有这样的形式。

(4) 一般来说，递推的效率高于递归，因此一般将递归转化为递推。

(5) 每个迭代算法原则上总可以转换为与它等价的递归算法；反之不然。

5.1.3　递推与循环

循环用于重复性的工作。循环体的特点是“以不变应万变”。所谓“不变”是指循环体内

运算的表现形式(变量表示的循环条件、循环体)是不变的,而数据变化用变量表示,变量取值不同则每次具体的执行内容不尽相同。循环设计主要是从建好的数据模型中选择合适的变量,构造不变的循环条件和循环体——循环不变式。例如,例 2 中 $s=s\times i$ 就是循环不变式。变量 i 表示循环次数,也表示当次循环的乘数;变量 s 保存前次和当次循环的累乘结果。初始 $s=1$,第 1 次循环后 $s=s\times 2=1\times 2$,第 2 次循环后 $s=s\times 3=1\times 2\times 3$,……,第 $n-1$ 次循环后 $s=s\times n=1\times 2\times\cdots\times n$。

例 4:可以有不同的循环不变式,如表 5-1 所示。如果一月用 a 表示,二月用 b 表示,三月及以上用 c 表示,则有 $c=a+b$,其实现如下。

输入:n。

输出:$F(n)$。

```
1.  a = 1
2.  b = 1
3.  for i = 1 to n - 2 do
4.      c = a + b
5.      print(c)
6.      a = b
7.      b = c
8.  return c
```

算法每次循环递推 1 步,使用 3 个变量,其时间复杂度为 $O(4n)$。当 $n=12$ 时,算法需要 10 次循环。

如果一月用 a 表示,二月用 b 表示,三月用 c 表示,四月又用 a 表示,五月又用 b 表示,则有 $c=a+b$,$a=c+b$,$b=c+a$,其实现如下。

输入:n。

输出:$F(n)$。

```
1.  a = 1
2.  b = 1
3.  for i = 1 to n/3 do
4.     c = a + b
5.     a = b + c
6.     b = c + a
7.     print(c,a,b)
```

算法每次循环递推 3 步,使用 3 个变量,其时间复杂度为 $O(4n/3)$。当 $n=12$ 时,算法需要 4 次循环,算法最后输出的并不是 12 项,而是 $2+3\times 4=14$ 项。

如果一月用 a 表示,二月用 b 表示,三月又用 a 表示,则有 $a=a+b$,$b=b+a$,其实现如下。

输入:n。

输出:$F(n)$。

```
1.  a = 1
2.  b = 1
3.  for i = 1 to n/2 do
4.      a = a + b
```

```
5.      b = a + b
6.      print(a,b)
```

算法每次循环递推 2 步，使用 2 个变量，其时间复杂度为 $O(3n/2)$。当 $n=12$ 时，算法需要 5 次循环。

上述算法都是递推算法，循环不变式就是递推公式，利用计算机计算速度快和递推算法不断重复的特点把复杂过程转化为简单过程的多次重复。

5.1.4　递归与非递归

递归与循环都是解决“重复操作”的机制。递归代码简洁，结构清晰，容易理解，容易用归纳法证明，而设计循环不变式则比较困难。递归实现耗费更多的时间（调用和返回需要额外的时间）与存储空间（栈保存递归调用的变量值），这也限制了递归的深度。因此一般将递归算法转变为非递归算法。递归与非递归算法的比较如表 5-2 所示。

表 5-2　递归与非递归算法的比较

算法	可读性	代码量	时间	空间	适用范围	设计难度
递归	易	小	长	大	广	易
非递归	难	大	短	小	窄	难

递归算法转变为非递归算法的常用方法如下。

（1）采用一个用户定义的栈来模拟系统的递归调用工作栈。该方法通用性强，但本质上还是递归，只不过人工做了本来由编译器做的事情，优化效果不明显。

（2）用递推来实现递归函数。

（3）通过变换将递归转化为尾递归，从而迭代求出结果。

后两种方法在时空复杂度上均有较大改善，但其适用范围有限。尾递归中递归调用是整个函数体中最后执行的语句并且它的返回值不属于表达式的一部分，其特点是回归过程中不用做任何操作。这个特性很重要，因为大多数现代编译器会利用这个特点自动生成优化的代码。

例 2 的尾递归实现：

输入：n。

输出：$F(n,1)$。

```
F(n,a) {
1.   if(n == 0) then return 0
2.   else if(n == 1) then
3.              return a
4.        else return F(n - 1,n * a)
}
```

时间复杂度为 $O(n)$。第 1 次调用返回的结果是 $F(n-1,n)$，第 2 次调用返回的结果是 $F(n-2,n(n-1))$，……，第 n 次返回的结果是 $F(1,n(n-1)\cdots 2)$。当前的运算结果（路径）作为参数传给下层函数，函数在递归调用前完成所有计算并把结果交给子函数，子函数不需要再去创建一个栈帧，直接就用当前栈帧把原先的数据覆盖即可。一是防止栈溢出，二

是节省了调用函数时创建栈帧的开销。

例 4 的尾递归实现：

输入：n。

输出：$F(n,1,1)$。

```
1.  F(n,ret1,ret2) {
2.  if(n == 1) then return 1
3.  else if(n == 2) then return rect2
4.  return F(n - 1,ret2,ret1 + ret2)
}
```

时间复杂度为 $O(n)$。第 1 次调用返回的结果是 $F(n-1,1,2)$，第 2 次调用返回的结果是 $F(n-2,2,3)$，第 3 次调用返回的结果是 $F(n-3,3,5)$……

前面两个例子中的函数都可以找到相应的非递归方式定义，但有些函数找不到相应的非递归方式定义，如 Ackerman 函数。

例 5：Ackerman 函数。

$$\begin{cases} A(1,0)=2 & \\ A(0,m)=1 & m \geqslant 0 \\ A(n,0)=n+2 & n \geqslant 2 \\ A(n,m)=A(A(n-1,m),m-1) & n,m \geqslant 1 \end{cases}$$

Ackerman 函数是双递归函数，函数和它的一个变量都由函数自身定义。

$m=1$ 时，$A(n,1)=A(A(n-1,1),0)=A(n-1,1)+2$，又 $A(1,1)=2$，故 $A(n,1)=2n$，$n \geqslant 1$。

$m=2$ 时，$A(n,2)=A(A(n-1,2),1)=2A(n-1,2)$，又 $A(1,2)=A(A(0,2),1)=A(1,1)=2$，故 $A(n,2)=2^n$。

$m=3$ 时，类似地可以推出 $A(n,3)=2^{2^{\cdots^2}}$，其中 2 的层数为 n。

$m=4$ 时，$A(n,4)$的增长速度非常快，以至于没有适当的数学式子来表示这一函数，因此不能转为非递归。

5.1.5 切分问题

王小二自夸刀工不错，有人放一张大的煎饼在砧板上，问他："饼不许离开砧板，切 n 刀最多能分成多少块？"

问题分析：切分块数统计如表 5-3 所示。切一次分成两块(2=1+1)，切两次分成 4 块(4=2+2)，切 3 次分成 7 块(7=3+4)……

表 5-3 切分块数统计

切分次数	0	1	2	3	4	5
切分块数	1	2	4	7	11	16

设第 n 次切分块数为 $F(n)$，根据表 5-3，有 $F(n)=F(n-1)+n$，$F(0)=1$。

切分问题的递推算法：

输入：n。

输出：$F(n)$。

```
1.  F[0] = 1
2.  for i = 1 to n do
3.      F[i] = F[i - 1] + i
4.  return F[n]
```

实际上由 $F(n)=F(n-1)+n$，可得 $F(n-1)=F(n-2)+n-1$，这样可以直接推出结果。

$$\begin{aligned} F(n) &= F(n-1)+n \\ &= F(n-2)+n-1+n \\ &\quad \vdots \\ &= F(0)+1+\cdots+n \\ &= 1+n(n+1)/2 \end{aligned}$$

5.1.6　狱吏问题

狱吏问题：国王对囚犯大赦，让狱吏 n 次通过一排锁着的 n 间牢房，每通过 1 次，按规则转动某些门锁，进行开关转换，最后门锁开着的牢房中的犯人获释。转动门锁的规则如下：第 1 次转动所有门锁；第 2 次从第 2 间牢房开始每隔 1 间转动 1 次；……；第 k 次从第 k 间牢房开始，每隔 $k-1$ 间转动 1 次。转动 n 次后哪些牢房的门锁是开的？

问题分析 1：设置长度为 n 的数组进行模拟。第 1 轮次转动 1、2、…、n 号牢房，第 2 轮次转动 2、4、…号牢房，……，第 k 轮次转动 k、$2k$、…号牢房。

输入：n。

输出：A。

```
1.  for i = 1 to n do
2.      A[i] = 1                                  //初始化门锁关
3.  for i = 1 to n do
4.      for j = i to n do
5.          A[j] = 1 - A[j]                       //转动门锁
6.          j = j + i
7.  return A
```

算法的时间复杂度为 $n+n/2+n/3+\cdots+n/n=O(n\ln n)$。

问题分析 2：设 $d(n)$ 表示 n 的不重复因子个数，如 4 的因子是 1、2 和 4，正好是门锁的开关次数。牢房开始是锁的，因此因子个数为奇数的房间，就是最后门锁开的房间。

输入：n。

输出：最后门锁开的房间号。

```
1.  for i = 1 to n do
2.      d = 1
3.      for j = 2 to i do
4.          if (i mod j == 0) then d = d + 1
5.      if (d mod 2 == 1) print i
```

算法的时间复杂度为 $1+2+3+\cdots+n=O(n^2)$。

问题分析 3：设 $d(n)$表示 n 的不重复因子个数，$d(n)$为奇数的就是最后门锁开的房间。$d(n)$为奇数的房间号为 1,4,9,…，正好是完全平方数。因为因子是成对出现的，只有完全平方数的两个因子相同，计数为 1 个，因此 $d(n)$为奇数。

输入：n。

输出：最后门锁开的房间号。

```
1.  for i = 1 to n do
2.      if (i * i <= n) print i * i
```

算法的时间复杂度为 $O(n)$。

本节思考题

划分问题：n 个大小不等的圆饼，分给 m 个人。要求每个人分得的大小相等，每个人只能有一块。那么最大的分隔方案是什么？

视频讲解

5.2 倒推算法

5.2.1 倒推与应用

正推是从小规模问题推解出大规模问题的方法。倒推是从后向前推解问题的方法，是对某些特殊问题所采用的违反通常习惯的方法。倒推一般使用情况如下。

(1) 不知前提条件时，从后向前递推，从而求解问题。即由结果倒过来推解它的前提条件。

(2) 由于存储的要求，必须从后向前进行推算。

(3) 一些问题从前向后分析问题比较棘手，而采用倒推则容易理解和解决。

例 6：猴子偷桃。一只小猴子摘了若干桃子，每天吃现有桃子的一半多 1 个，到第 10 天时就只有 1 个桃子了，求原有多少个桃子？

问题分析：第 10 天剩余 1 个桃子，第 9 天的桃子数$=2(1+1)=4$，第 8 天的桃子数$=2(4+1)=10$，……

设第 i 天的桃子数为 $F[i]$，则有 $F[i]=2(F[i+1]+1)$，$F[10]=1$，$i=9,8,\cdots,1$。

猴子偷桃的递推算法：

```
1.  s = 1
2.  for i = 9 to 1 do
3.      s = (s + 1) * 2
4.  return s
```

例 7：输出如图 5-2(a)所示的杨辉三角形(限定用 1 个一维数组)。

问题分析：上下行规律明显，中间的数等于上行左上、右上两数之和。如果使用二维数组存储，对于第 i 行，$A[i,1]=A[i,i]=1$，$A[i,j]=A[i-1,j]+A[i-1,j-1]$，如图 5-2(b)所示。

图 5-2　杨辉三角形

题目中要求用 1 个一维数组完成。若求 n 层，则数组最多存储 n 个数据。对于第 i 行，$A[1]=1, A[i]=1, A[j]=A[j]+A[j-1]$。如果从左到右计算，$A[i,j]=A[i-1,j]+A[i-1,j-1]$中的 $A[i-1,j-1]$已经变为 $A[i,j-1]$，等式不一定成立。如果从右向左计算，$A[i,j]=A[i-1,j]+A[i-1,j-1]$中的 $A[i-1,j]$和 $A[i-1,j-1]$没有发生变化，等式仍然成立。

依照上面分析，采取从右向左计算，计算公式为 $A[1]=1, A[i]=1, A[j]=A[j]+A[j-1]$。

杨辉三角递推算法：

```
1.  print("1")
2.  print("\n")
3.  a[1] = a[2] = 1
4.  print(a[1],a[2])
5.  print("\n")
6.  for i = 3 to n do
7.      a[1] = a[i] = 1
8.      for j = i - 1 to 2 do
9.          a[j] = a[j] + a[j - 1]
10.     for j = 1 to i do
11.         print(a[j])
12.     print("\n")
```

例 8：穿越沙漠。用一辆吉普车穿越 1000 千米的沙漠。吉普车的总装油量为 500 加仑，耗油率为 1 加仑/千米。沙漠中没有油库，必须先用这辆车在沙漠中建立临时油库。该吉普车以最少的耗油量穿越沙漠，应在什么地方建油库，各油库的储油量是多少？

问题分析：问题要求以最少的耗油量穿越沙漠，那么到达终点时，沙漠中的各临时油库和车的装油量均为 0 加仑。这样只能从终点开始向前倒推储油点和储油量。设下一个油库距离为 x 千米，运油次数为 y。

最后一段长度为 500 千米且最后一个加油点储油为 500 加仑。

倒数第二段中为了储备油，吉普车在这段的行程必须有来回。

(1) 首先不计方向，这段应走奇数次(保证最后向前走)，即 $2y-1$ 次，相应的消耗油量为$(2y-1)x$ 加仑。

(2) 每次向前行进时吉普车是满载，即 $500y$ 加仑。

(3) 储存够下一个加油点 500 加仑油量。

根据上述分析有 $500y=500+(2y-1)x$，则 $y_{\min}=2$，$x=500(y-1)/(2y-1)=$

500/3，储油 $500y=1000$。

同样，倒数第三段有 $500y=1000+(2y-1)x$，则 $y_{\min}=3$，$x=500(y-2)/(2y-1)=500/5$，储油 $500y=1500$。

因此从终点倒推起点，倒数第一个油库距离终点为 500 千米，储油量为 500 加仑。设倒数第 k 个油库距离上一个油库的距离为 x 千米，运油次数为 y。根据上述分析有 $x=500/(2k-1)$千米，储油量为 $500k$ 加仑。

穿越沙漠递推算法：

```
1.  dis = 500
2.  oil = 500
3.  k = 1
4.  do
5.  print(k,1000dis,oil)
6.      k = k + 1
7.      dis = dis + 500/(2k - 1)
8.      oil = 500k
9.  while (dis < 1000)
10. oil = 500(k - 1) + (1000 - dis)(2k - 1)              //第一个储油点
11. print(k,0,oil)
```

5.2.2 约瑟夫问题

n 个人围成一圈，从第一个人开始顺序报数，报数 m 的将被杀掉，最后剩下一个人胜出。例如 $n=6$，$m=5$，被杀掉的顺序是：5、4、6、2、3、1 胜出。

问题分析：n 个人(编号 0～$n-1$)，从 0 开始报数，报到 $m-1$ 的退出，剩下的人继续从 0 开始报数。

第 1 个出局的人编号：$m-1$。

剩余人员：　m　$m+1$　…　$n-1$　0　1　…　$m-2$

剩余人员报数：　0　1　…　$n-1-m$　$n-m$　$n-m+1$　…

第 2 个出局人报数 $m-1$，原编号是 $(m-1+m)\%n$。

设 $F[i]$表示 i 个人游戏报数 m 时最后胜利者的编号，则有 $F(1)=0$。根据上述分析，有 $F(i)=(F(i-1)+m)\%i$，$F(n)=(F(n-1)+m)\%n$。

约瑟夫递推算法：

输入：n，m。

输出：胜利者编号。

```
1.  f = 0
2.  for i = 2 to n do
3.      f = (f + m) % i
4.  print f + 1
```

本节思考题

1. 约瑟夫问题：一个监狱有 k 个好人和 k 个坏人。监狱长接到命令，杀掉 k 个人。

监狱长比较正义，想杀掉 k 个坏人，留下 k 个好人。他让 k 个好人和 k 个坏人顺序围成一圈，从第一个人开始顺序报数，报数 m 的将被杀掉，最后剩下 k 个好人，最小的 m 是多少？

2. NIM 游戏：有一堆石子和两个玩家。每个玩家每次可以从该堆剩余石子中拿走至少 1 个、至多 m 个石子。拿走最后一个剩余石子者获胜。获胜策略是什么？

如果有 n 堆石子，给定每堆石子数和两个玩家。每个玩家每次可以从任意一堆石子中拿走任意数量的石子(最少一个，最多整堆石子)，每次拿的石子数可以不同。拿走最后一个剩余石子者获胜。获胜策略是什么？

如果 NIM 游戏中，每次最多只能取 k 个，怎么处理？

视频讲解

5.3　递推求解

5.3.1　快速排序

C. A. R. Hoare 于 1962 年提出快速排序并于 1980 年获得图灵奖。快速排序算法思想如下：选择一个基准数，通过一次排序将要排序的数据分割成独立的两部分；其中一部分的所有数据都比另外一部分的所有数据小。然后，再按此方法对这两部分数据分别进行快速排序，整个排序过程可以递归进行，以此达到整个数据变成有序序列。

快速排序算法：

输入：数组 A。

输出：数组 A，使 $i<j$ 时 $a_i \leqslant a_j$，$1 \leqslant i < j \leqslant n$。

```
Call quickSort(A,1,n)
quickSort(A,p,r) {
1.   if (p < r) then
2.        q = partion(A,p,r)
3.        quickSort(A,p,q - 1)
4.        quickSort(A,q + 1,r)
5.   }
partion(A,p,r) {
1.   x = A[p]
2.   i = p
3.   j = r + 1
4.   while(true) do
5.        repeat j = j - 1     until A[j] <= x
6.        repeat i = i + 1     until A[i] > x
7.        if (i < j) then Swap(A[i],A[j])
8.        else Swap(A[j],A[p])  break
9.   return j
10 }
```

partion(A，p，r) 的操作过程如表 5-4 所示。

表 5-4 partion 操作示例

6	7	5	2	5	8	初始序列
6	7	5	2	**5**	8	$j=j-1$
6	**7**	5	2	5	8	$i=i+1$
6	**5**	5	2	**7**	8	交换
6	5	5	**2**	7	8	$j=j-1$
6	5	5	2	**7**	8	$i=i+1$
2	5	5	**6**	7	8	交换并划分

$j=j-1$ 和 $i=i+1$ 的操作次数不超过数组长度，因此划分操作的时间复杂度为 $O(n)$。记录的比较和交换是从两端向中间进行的，关键字较大的记录一次就能交换到后面单元，关键字较小的记录一次只能交换到前面单元，记录每次移动的距离较大，因而总的比较和移动次数较少。

设快速排序的时间复杂度是 $T(n)$。当 $n=1$ 时，$T(1)=0$。partion$(A,1,n)$划分成的 2 个子数组大小不一，影响快速排序的时间复杂度。

最坏情况下，序列已经排好序，partion 操作后分成的两个子数组，一个为空，一个有 $n-1$ 个元素，则 $T(n)=T(0)+T(n-1)+n-1$。

最好情况下，划分的两个子数组包含的元素数相等，则 $T(n)=2T(n/2)+n-1$。

平均情况下，假设划分的子数组长度的概率相同，则

$$T(n)=\frac{T(0)+T(n-1)+T(1)+T(n-2)+\cdots+T(n-1)+T(0)}{n}+n-1$$

$$=\frac{2(T(0)+T(1)+\cdots+T(n-1))}{n}+n-1$$

$$=\frac{2}{n}\sum_{i=1}^{n-1}T(i)+n-1$$

5.3.2 递推方程求解

递推方程求解的常用方法有：迭代法、递归树、归纳法和主定理法。

1. 迭代法

迭代法又分为直接迭代、差消迭代和换元迭代。

(1) 直接迭代：直接迭代从原始递推方程出发，反复将对应方程左边的函数用右边等式代入，直至得到初值，然后将所得的结果化简。

最坏情况下，快速排序的时间复杂度为 $T(n)=T(n-1)+n-1$，$T(1)=0$。则有 $T(n-1)=T(n-2)+n-2$，代入原式，有

$$\begin{aligned}T(n)&=T(n-1)+n-1\\&=[T(n-2)+n-2]+n-1\\&=T(n-2)+(n-2)+(n-1)\\&=[T(n-3)+n-3]+(n-2)+(n-1)\\&\cdots\\&=T(1)+1+2+\cdots+(n-2)+(n-1)\end{aligned}$$

$$=1+2+\cdots+(n-2)+(n-1)$$
$$=\frac{n(n-1)}{2}$$

为保证正确性，需要代入递推方程验证。

当 $n=1$ 时，$T(1)=1\times(1-1)/2=0$ 成立。假设 $n=k$ 时，$T(k)=k(k-1)/2$ 成立，则当 $n=k+1$ 时，有 $T(k+1)=T(k)+k=k(k-1)/2+k=k(k+1)/2$，故公式成立。

(2) 差消迭代：迭代一般用于一阶递推方程，高阶方程需要使用差消法化简为一阶方程求解。

平均情况下，快速排序的时间复杂度为

$$T(n)=\frac{2}{n}\sum_{I=1}^{n-1}T(i)+n-1,\quad T(1)=0$$

则有

$$nT(n)=2\sum_{i=1}^{n-1}T(i)+n^2-n,(n-1)T(n-1)=2\sum_{i=1}^{n-2}T(i)+(n-1)^2-(n-1)$$

两边加 $2T(n-1)$，得

$$(n+1)T(n-1)=2\sum_{i=1}^{n-2}T(i)+2T(n-1)+(n-1)^2-(n-1)$$
$$=2\sum_{i=1}^{n-1}T(i)+(n-1)^2-(n-1)$$
$$nT(n)=2\sum_{i=1}^{n-1}T(i)+n^2-n$$
$$=(n+1)T(n-1)-(n-1)^2+(n-1)+n^2-n$$
$$=(n+1)T(n-1)+2n-2$$

等式两边均除以 $n(n+1)$，得

$$\frac{T(n)}{n+1}=\frac{T(n-1)}{n}+\frac{2n-2}{n(n+1)}=\cdots=2\left[\frac{1}{n+1}+\frac{1}{n}+\cdots+\frac{1}{3}\right]+\frac{T(1)}{2}-1-O\left(\frac{1}{n}\right)$$
$$=\theta(\log n)$$

所以，$T(n)=\Theta(n\log n)$。

(3) 换元迭代：将 n 的递推式换成其他变元 k 的递推式，对 k 直接迭代，最后将解变换为 n 的函数。

最好情况下，快速排序的时间复杂度为 $T(n)=2T(n/2)+n-1,T(1)=0$。

假设 $n=2^k$，则 $k=\log n$，代入上式有 $T(2^k)=2T(2^{k-1})+2^k-1,T(2^{k-1})=2T(2^{k-2})+2^{k-1}-1$。

$$T(2^k)=2T(2^{k-1})+2^k-1$$
$$=2[2T(2^{k-2})+2^{k-1}-1]+2^k-1$$
$$=2^2T(2^{k-2})+2(2^k)-2-1$$
$$=2^3T(2^{k-3})+3(2^k)-2^2-2-1$$
$$\cdots$$

$$=2^k T(2^0)+k(2^k)-2^{k-1}-\cdots-2^2-2-1$$
$$=k(2^k)-2^k$$

所以 $T(n)=n\log n-n=\Theta(n\log n)$。

2. 递归树

最好情况下，快速排序的时间复杂度为 $T(n)=2T(n/2)+O(n)$，$T(1)=0$。其中，$O(n)$为长度为 n 的数组划分为 2 个 $n/2$ 个元素的数组的时间复杂度。

使用递归树表示，如图 5-3 所示，节点是划分的子问题。右边是分解和合并的代价＝每层子问题数×(分解和合并该层一个子问题的代价)。

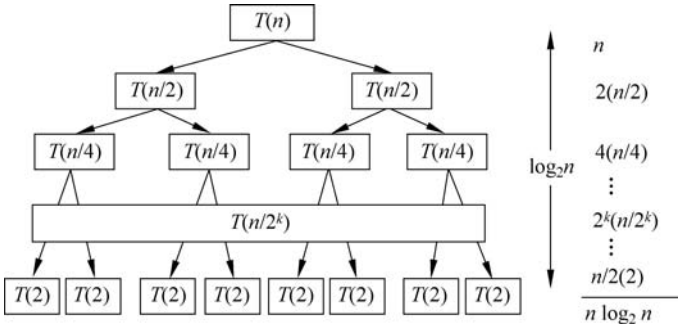

图 5-3　快速排序的递归树表示

根据公式 $T(n)=2T(n/2)+O(n)$，有 $T(n/2)=2T(n/4)+O(n/2)$，$T(n/4)=2T(n/8)+O(n/4)$，…，$T(2)=2T(1)+O(2)$。如图 5-3 所示，$T(2)$层一共有 $n/2$ 组，每组划分的时间复杂度为 2，因此该层划分的时间为 $2(n/2)=n$。$T(n/2)$层一共有 2 组，每组划分的时间复杂度为 $n/2$，因此该层划分的时间同样为 $2(n/2)=n$。$T(n)$层一共有 1 组，划分的时间复杂度为 n，因此该层划分的时间同样为 n。设 $n=2^x$，则 $x=\log n$。递归树一共有 x 层，每次划分的时间为 n，因此总的时间复杂度为 $nx=n\log n$。

递归树法求解过程使用部分近似，如 $n=2^x$，为保证正确性，需要代入递推方程进行验证。

当 $n=1$ 时，$T(1)=\log 1=0$ 成立。假设 $n=k$ 时，$T(k)=k\log_2 k$ 成立，则当 $n=2k$ 时，有 $T(2k)=2T(k)+2k=2k\log_2 k+2k=2k(\log_2 k+1)=2k\log 2k$，故公式成立。

3. 归纳法

最好情况下，快速排序的时间复杂度为 $T(n)=2T(n/2)+O(n)$，$T(1)=0$。使用归纳法证明 $T(n)=n\ \log_2 n$。

证明：

当 $n=1$ 时，$T(1)=1\times\log_2 1=0$ 成立。假定 $n=k$ 时，$T(k)=k\log_2 k$ 成立，则当 $n=2k$ 时，有 $T(2k)=2T(k)+2k=2k\log_2(k)+2k=2k[\log_2(2k)-1]+2k=2k\log_2(2k)$，故公式成立。

4. 主定理法

设 $a\geqslant 1$，$b>1$ 为常数，$f(n)$为函数，$T(n)$为非负整数，且 $T(n)=aT(n/b)+f(n)$，则有

$$T(n)=\begin{cases}\Theta(n^{\log_b a}) & \exists\varepsilon,\text{使 } f(n)=O(n^{\log_b a-\varepsilon}) \\ \Theta(n^{\log_b a}\log n) & f(n)=\Theta(n^{\log_b a}) \\ \Theta(f(n)) & \exists\varepsilon,\text{使 } f(n)=\Omega(n^{\log_b a+\varepsilon}),\text{且 } \exists \text{ 常数 } c \text{ 和 } \forall \text{ 充分大的 } n,\text{使 } af(n/b)\leqslant cf(n)\end{cases}\left.\begin{array}{l}\\ \\ \\ \end{array}\right\}\begin{array}{l}\varepsilon>0\\ c<1\end{array}$$

例 9：$T(n)=9T(n/3)+n$。

$n^{\log_b a}=n^{\log_3 9}=n^2$，$f(n)=n=n^{2-\varepsilon}=O(n^{\log_b a-\varepsilon})$。依据主定理，有 $T(n)=\Theta(n^2)$。

例 10：$T(n)=T(2n/3)+1$。

$n^{\log_b a}=n^{\log_{3/2}1}=n^0=1$，$f(n)=1=\theta(n^{\log_b a})$。依据主定理，有 $T(n)=\Theta(\log n)$。

例 11：$T(n)=3T(n/4)+n\log n$。

$n^{\log_b a}=n^{\log_4 3}=n^{0.793}$，$f(n)=n\log n$，$f(n)=\Omega(n^{\log_b a+\varepsilon})$。

又 $af(n/b)=3f(n/4)=(3n\log(n/4))/4=(3n(\log n-2))/4<cn\log n$，$3/4\leqslant c<1$。依据主定理，$T(n)=\Theta(n\log n)$。

例 12：$T(n)=2T(n/2)+n\log n$。

$n^{\log_b a}=n^{\log_2 2}=n$，$f(n)=n\log n$，不存在 ε，使 $f(n)=\Omega(n^{\log_b a+\varepsilon})$。

又不存在 $c<1$，使 $2f(n/2)=2(n/2)\log(n/2)=n(\log n-1)<cn\log n$，因此不能使用主定理。

使用迭代求解，有

$$\begin{aligned}T(n)&=2T(n/2)+n\log n\\&=2[2T(n/4)+(n/2)(\log n/2)]+n\log n\\&=2^2T(n/4)+n(\log n-1)+n\log n\\&\cdots\\&=2^kT(n/2^k)+n(\log n-k)+\cdots+n(\log n-1)+n\log n\end{aligned}$$

设 $n=2^k$，则 $T(n)=n(k+1)\log n-(k(k+1))/2=n(\log n+1)\log n-(\log n(\log n+1))/2=O(n\log^2 n)$。

主定理推导过程如下。

$T(n)=aT(n/b)+f(n)$ 则 $T(n/b)=aT(n/b^2)+f(n/b)$，$T(n/b^2)=aT(n/b^3)+f(n/b^2)$，故有

$$T(n)=aT\left(\frac{n}{b}\right)+f(n)=a^2T\left(\frac{n}{b^2}\right)+af\left(\frac{n}{b}\right)+f(n)$$

$$=a^3T\left(\frac{n}{b^3}\right)+a^2f\left(\frac{n}{b^2}\right)+af\left(\frac{n}{b}\right)+f(n)$$

设 $n=b^x$，则 $x=\log_b n$，又 $a^x=a^{\log_b n}=n^{\log_b a}$，因此有

$$T(n)=aT\left(\frac{n}{b}\right)+f(n)=\cdots=a^xT\left(\frac{n}{b^x}\right)+a^{x-1}f\left(\frac{n}{b^{x-1}}\right)+\cdots+af\left(\frac{n}{b}\right)+f(n)$$

$$=a^xT\left(\frac{n}{b^x}\right)+\sum_{j=0}^{x-1}a^jf\left(\frac{n}{b^j}\right)$$

$$=n^{\log_b a}+\sum_{j=0}^{\log_b n-1}a^j f\left(\frac{n}{b^j}\right)$$

(1) 若 $f(n)=O(n^{\log_b a-\varepsilon})$,则

$$f\left(\frac{n}{b^j}\right)=O\left(\left(\frac{n}{b^j}\right)^{\log_b a-\varepsilon}\right)$$

$$\sum_{j=0}^{\log_b n-1}a^j f\left(\frac{n}{b^j}\right)=O\left(\sum_{j=0}^{\log_b n-1}a^j\left(\frac{n}{b^j}\right)^{\log_b a-\varepsilon}\right)=O\left(n^{\log_b a-\varepsilon}\sum_{j=0}^{\log_b n-1}\left(\frac{ab^{\varepsilon}}{b^{\log_b a}}\right)^j\right)$$

$$=O\left(n^{\log_b a-\varepsilon}\sum_{j=0}^{\log_b n-1}(b^{\varepsilon})^j\right)=O\left(n^{\log_b a-\varepsilon}\left(\frac{b^{\varepsilon\log_b n}-1}{b^{\varepsilon}-1}\right)\right)=O\left(n^{\log_b a-\varepsilon}\left(\frac{n^{\varepsilon}-1}{b^{\varepsilon}-1}\right)\right)$$

$$=O(n^{\log_b a-\varepsilon}(n^{\varepsilon}))=O(n^{\log_b a})$$

因此,$\exists\varepsilon>0$,若 $f(n)=O(n^{\log_b a-\varepsilon})$,则 $T(n)=n^{\log_b a}+\sum_{j=0}^{\log_b n-1}a^j f(n/b^j)=\Theta(n^{\log_b a})$。

(2) 若 $f(n)=\Theta(n^{\log_b a})$,则

$$f\left(\frac{n}{b^j}\right)=\Theta\left(\left(\frac{n}{b^j}\right)^{\log_b a}\right)$$

$$\sum_{j=0}^{\log_b n-1}a^j f\left(\frac{n}{b^j}\right)=\Theta\left(\sum_{j=0}^{\log_b n-1}a^j\left(\frac{n}{b^j}\right)^{\log_b a}\right)=\Theta\left(n^{\log_b a}\sum_{j=0}^{\log_b n-1}\left(\frac{a}{b^{\log_b a}}\right)^j\right)$$

$$=\Theta\left(n^{\log_b a}\sum_{j=0}^{\log_b n-1}(1)^j\right)=\Theta(n^{\log_b a}(\log_b n))$$

因此,若 $f(n)=\Theta(n^{\log_b a-\varepsilon})$,则 $T(n)=n^{\log_b a}+\sum_{j=0}^{\log_b n-1}a^j f(n/b^j)=\Theta(n^{\log_b a}\log n)$。

(3) 若 $f(n)=(n^{\log_b a+\varepsilon})$,$c<1$。则有

$$af\left(\frac{n}{b}\right)\leqslant cf(n),\quad f\left(\frac{n}{b}\right)\leqslant\left(\frac{c}{a}\right)f(n),\quad f\left(\frac{n}{b^j}\right)\leqslant\left(\frac{c}{a}\right)^j f(n)$$

$$\sum_{j=0}^{\log_b n-1}a^j f\left(\frac{n}{b^j}\right)\leqslant\sum_{j=0}^{\log_b n-1}a^j\left(\frac{c}{a}\right)^j f(n)+O(1)\leqslant\sum_{j=0}^{\log_b n-1}(c)^j f(n)+O(1)$$

$$=\frac{f(n)}{1-c}+O(1)=\Theta(f(n))$$

因此,$\exists\varepsilon>0$,$c<1$,若 $f(n)=\Omega(n^{\log_b a+\varepsilon})$,则 $T(n)=n^{\log_b a}+\sum_{j=0}^{\log_b n-1}a^j f(n/b^j)=\Theta(f(n))$。

本节思考题

1. 设算法 A 的时间复杂度递推方程为 $T(n)=7T(n/2)+n^2$,算法 B 的时间复杂度递推方程为 $T(n)=\lambda T(n/4)+n^2$,确定最大的正整数 λ,使算法 B 的阶不高于 A 的阶。

2. 设原问题的规模为 n,从下面算法中选择复杂度最低的算法,并简要说明原因。

算法 A:将原问题划分为规模减半的 5 个子问题,递归求解每个子问题,然后在线性时

间内合并子问题的解并得到原问题的解。

算法 B：先递归求解两个规模为 $n-1$ 的子问题，然后在常量时间内将子问题的解合并得到原问题的解。

算法 C：将原问题划分为规模为 $n/3$ 的 9 个子问题，递归求解每个子问题，然后在 $O(n^3)$时间内将子问题的解合并得到原问题的解。

本章习题

1. 编程实现划分问题算法(POJ 3122、POJ 1664)。
2. 编程实现递推算法(POJ 1942)。
3. 编程实现递归算法(POJ 2083)。
4. 编程实现约瑟夫问题算法(POJ 2244、POJ 3517)。
5. 编程实现快速排序算法(POJ 2388、POJ 2263、POJ 1974)。
6. 编程实现线性方程组算法(POJ 2345)。
7. 编程实现牛顿迭代解方程算法(POJ 3111)。
8. 编程实现石子游戏算法(POJ 1067、POJ 3688)。
9. 简述递推方程的求解方法。
10. 简述递推与递归、循环的关系。
11. 求解下列递推方程。

(1) $T(n)=\begin{cases}T(n-1)+n^2\\T(1)=1\end{cases}$。

(2) $T(n)=\begin{cases}8T(n/2)+n^2\\T(1)=1\end{cases}$。

(3) $T(n)=\begin{cases}T(n/2)+T(n/4)+n\\T(1)=1\end{cases}$。

(4) $T(n)=\begin{cases}T(n/2)+\log 3^n\\T(1)=1\end{cases}$。

(5) $T(n)=\begin{cases}2T(n/2)+n^2\log n\\T(1)=1\end{cases}$。

(6) $T(n)=\begin{cases}T(n-1)+1/n\\T(1)=1\end{cases}$。

12. 一群猴子编号是 1,2,3,…,m，这群猴子按照 1～m 的顺序围坐一圈，从第 1 开始数，每数到第 n 个，该猴子就要离开此圈，这样依次下来，直到圈中只剩下最后一只猴子，则该猴子为大王。大王的编号是多少？

13. 集合划分问题：给定正整数 $n(1\leqslant n\leqslant 20)$，计算出 n 个元素的集合$\{1,2,\cdots,n\}$可

以化为多少个不同的非空子集。

14. 集合划分问题：给定正整数 n 和 m，计算出 n 个元素的集合$\{1,2,\cdots,n\}$可以划分为多少个不同的由 m 个非空子集组成的集合。

15. 给定 n 个物品的重量，给定常数 k，编写算法，选择部分物品，使其重量之和正好等于 k。

16. 请给出汉诺塔问题的递推公式和时间复杂度、汉诺塔问题递归与非递归算法。

17. 冯·诺依曼邻居问题：从 1×1 的方格开始，每次在上次图形周围增加一圈方格，第 1 次只有 1 个方格，第 2 次增加 4 个方格，第 3 次增加 8 个方格，……，第 n 次生成多少方格？请给出递推关系和时间复杂度。