离散数学

（第 3 版）

智能科学与技术学院 2024 级

目录

目录

11.1 递推方程的定义及实例

定义 1.1.1

设序列 $a_0, a_1, \cdots, a_n, \cdots$, 简记为 $\{a_n\}$, 一个把 a_n 与某些个 $a_i, i < n$, 联系起来的等式称作关于序列 $\{a_n\}$ 的 *递推方程*.

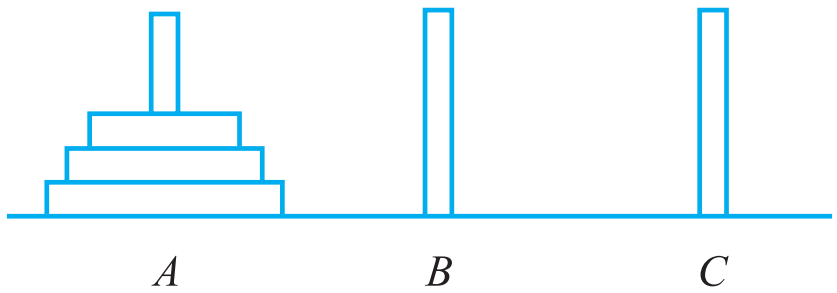
11.1 递推方程的定义及实例

定义 1.1.1

设序列 $a_0, a_1, \cdots, a_n, \cdots$, 简记为 $\{a_n\}$, 一个把 a_n 与某些个 $a_i, i < n$, 联系起来的等式称作关于序列 $\{a_n\}$ 的 *递推方程*.

例 1.1.1

Hanoi 塔问题. 下图中有 A、B、C 三个柱子, 在 A 柱上放着 n 个圆盘(下图中 $n = 3$), 其中小圆盘放在大圆盘的上边. 从 A 柱将这些圆盘移到 C 柱上去. 如果把一个圆盘从一个柱子移到另一个柱子称作一次移动, 在移动和放置时允许使用 B 柱, 但不允许大圆盘放到小圆盘的上面. 问: 把所有的圆盘从 A 移到 C 总计需要多少次移动?



下面的递归算法中，Hanoi(X, Y, m) 表示从 X 柱到 Y 柱用 Hanoi 算法 移动 m 个盘子的过程，move(X, Y) 表示从 X 柱移动 1 个盘子到 Y 柱的过程.

算法 Hanoi (A, C, n)

```
1 if n = 1 then move(A, C) ;
2 else
3     Hanoi(A, B, n − 1)
4     move(A, C)
5     Hanoi(B, C, n − 1)
```

下面的递归算法中，Hanoi(X, Y, m) 表示从 X 柱到 Y 柱用 Hanoi 算法移动 m 个盘子的过程，move(X, Y) 表示从 X 柱移动 1 个盘子到 Y 柱的过程.

算法 Hanoi (A, C, n)

```
1 if n = 1 then move(A, C) ;
2 else
3     Hanoi(A, B, n − 1)
4     move(A, C)
5     Hanoi(B, C, n − 1)
```

设使用 Hanoi 算法移动 n 个盘子的总次数为 $T(n)$. 步骤 3 使用 Hanoi 算法递归地将 $n-1$ 个盘子从 A 柱移到 B 柱，移动次数为 $T(n-1)$；步骤 4 利用 1 次移动将最下面的大盘子从 A 柱移到 C 柱；步骤 5 还是用 Hanoi 算法将 B 柱上的 $n-1$ 个盘子移到 C 柱，移动次数为 $T(n-1)$. 因此，得到递推方程 $T(n) = 2T(n-1) + 1$. 这个方程的初值是 $T(1) = 1$. 后面将证明这个方程的解是 $T(n) = 2^n - 1$.

下面的递归算法中，Hanoi(X, Y, m) 表示从 X 柱到 Y 柱用 Hanoi 算法移动 m 个盘子的过程，move(X, Y) 表示从 X 柱移动 1 个盘子到 Y 柱的过程.

算法 Hanoi (A, C, n)

```
1 if n = 1 then move(A, C) ;
2 else
3     Hanoi(A, B, n − 1)
4     move(A, C)
5     Hanoi(B, C, n − 1)
```

设使用 Hanoi 算法移动 n 个盘子的总次数为 $T(n)$. 步骤 3 使用 Hanoi 算法递归地将 $n-1$ 个盘子从 A 柱移到 B 柱，移动次数为 $T(n-1)$；步骤 4 利用 1 次移动将最下面的大盘子从 A 柱移到 C 柱；步骤 5 还是用 Hanoi 算法将 B 柱上的 $n-1$ 个盘子移到 C 柱，移动次数为 $T(n-1)$. 因此，得到递推方程 $T(n) = 2T(n-1) + 1$. 这个方程的初值是 $T(1) = 1$. 后面将证明这个方程的解是 $T(n) = 2^n - 1$.

这个问题就是著名的 Hanoi 塔问题，据说古代的僧侣按照这种方法移动 64 个金盘子，他们认为当 64 个金盘子全部移完以后，世界的末日就到了.

下面计算移动时间. 如果每秒钟移动 1 次,那么 64 个盘子需要 $2^{64}-1=18\ 446\ 744\ 073\ 709\ 551\ 615$ 秒,大约是 5 000 亿年. 对于 Hanoi 塔问题,盘子的个数 n 代表问题规模,$T(n)$ 代表求解规模为 n 的问题所做的基本运算次数,它代表了这种算法的效率. Hanoi 算法的 $T(n)$ 是 n 的指数函数. 不难看到,指数函数的值随着自变量 n 的增加呈爆炸性增长.

下面计算移动时间. 如果每秒钟移动 1 次,那么 64 个盘子需要 $2^{64}-1=18\ 446\ 744\ 073\ 709\ 551\ 615$ 秒,大约是 5 000 亿年. 对于 Hanoi 塔问题,盘子的个数 n 代表问题规模,$T(n)$ 代表求解规模为 n 的问题所做的基本运算次数,它代表了这种算法的效率. Hanoi 算法的 $T(n)$ 是 n 的指数函数. 不难看到,指数函数的值随着自变量 n 的增加呈爆炸性增长.

例 1.1.2

一个著名的数列称作 *Fibonacci 数列* ,这个数列的项是

$$1,1,2,3,5,8,13,21,\cdots$$

当 $n\geqslant 2$ 时,它的第 n 项恰好等于第 $n-1$ 项与第 $n-2$ 项之和,即

$$f_n=f_{n-1}+f_{n-2},\quad f_0=1,\quad f_1=1.$$

下面计算移动时间. 如果每秒钟移动 1 次,那么 64 个盘子需要 $2^{64}-1=18\ 446\ 744\ 073\ 709\ 551\ 615$ 秒,大约是 5 000 亿年. 对于 Hanoi 塔问题,盘子的个数 n 代表问题规模,$T(n)$ 代表求解规模为 n 的问题所做的基本运算次数,它代表了这种算法的效率. Hanoi 算法的 $T(n)$ 是 n 的指数函数. 不难看到,指数函数的值随着自变量 n 的增加呈爆炸性增长.

例 1.1.2

一个著名的数列称作 *Fibonacci 数列* ,这个数列的项是

$$1,1,2,3,5,8,13,21,\cdots$$

当 $n\geqslant 2$ 时,它的第 n 项恰好等于第 $n-1$ 项与第 $n-2$ 项之和,即

$$f_n=f_{n-1}+f_{n-2},\quad f_0=1,\quad f_1=1.$$

这个方程是关于 *Fibonacci* 数列的递推方程,在第 *11.2* 节例*1.2.1*中我们将证明该方程的解是

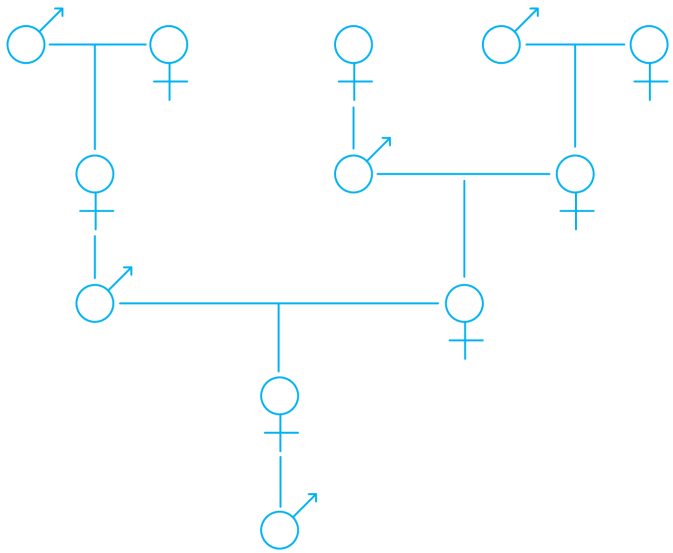
$$f_n=\frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^{n+1}-\frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^{n+1}.$$

Fibonacci 数列

Fibonacci 数列出现在许多实际问题中. 一个有趣的例子就是蜜蜂家族的构成. 一个蜂群通常由蜂王、工蜂、雄蜂构成：只有 1 只雌蜂是蜂王；大多数雌蜂都是工蜂，负责采蜜、抚育幼蜂等，但不产卵；有少量只参与繁殖后代的雄蜂. 所有雄蜂都由未受精的卵发育而成，只有母亲没有父亲；所有雌蜂则由受精卵发育而成，有父亲和母亲. 蜂群中新生的雌蜂大部分将发育成工蜂，只有被王浆喂养的少数雌蜂能够发育成蜂王. 这些新蜂王成熟之后将从拥挤的蜂巢分离，寻找新巢，建立新的蜂群.

Fibonacci 数列

Fibonacci 数列出现在许多实际问题中. 一个有趣的例子就是蜜蜂家族的构成. 一个蜂群通常由蜂王、工蜂、雄蜂构成：只有 1 只雌蜂是蜂王；大多数雌蜂都是工蜂，负责采蜜、抚育幼蜂等，但不产卵；有少量只参与繁殖后代的雄蜂. 所有雄蜂都由未受精的卵发育而成，只有母亲没有父亲；所有雌蜂则由受精卵发育而成，有父亲和母亲. 蜂群中新生的雌蜂大部分将发育成工蜂，只有被王浆喂养的少数雌蜂能够发育成蜂王. 这些新蜂王成熟之后将从拥挤的蜂巢分离，寻找新巢，建立新的蜂群.



这里以 ♀、♂ 符号分别代表雌蜂和雄蜂，那么可以用左图描述一只雄蜂的祖先树. 问：在这只雄蜂的家族中总共有多少个祖先? 他有 1 个母亲，2 个祖父母，3 个曾祖父母，5 个曾曾祖父母，$\cdots$，所有这些恰好构成 Fibonacci 数列 $\{f_n\}$. 这个结果的证明略去.

Fibonacci 数列(续)

考虑一个串的计数问题. 有 n 位长的二进制串, 如果要求其中没有两个连续的 0, 求这样的串的个数 C_n. 考虑满足上述条件的 n 位二进制串. 如果它的最后一位是 1, 那么它的前 $n-1$ 位也构成满足上述要求的串, 这样的串有 C_{n-1} 个; 如果它的最后一位是 0, 那么它的第 $n-1$ 位只能是 1, 剩下的 $n-2$ 位构成满足上述要求的串, 这样的串有 C_{n-2} 个.

Fibonacci 数列(续)

考虑一个串的计数问题. 有 n 位长的二进制串,如果要求其中没有两个连续的 0,求这样的串的个数 C_n. 考虑满足上述条件的 n 位二进制串. 如果它的最后一位是 1,那么它的前 $n-1$ 位也构成满足上述要求的串,这样的串有 C_{n-1} 个;如果它的最后一位是 0,那么它的第 $n-1$ 位只能是 1,剩下的 $n-2$ 位构成满足上述要求的串,这样的串有 C_{n-2} 个.

根据加法法则有 $C_n = C_{n-1} + C_{n-2}$,初值 $C_0 = 1$(空串), $C_1 = 2$(串 0 和 1).

Fibonacci 数列(续)

考虑一个串的计数问题. 有 n 位长的二进制串,如果要求其中没有两个连续的 0,求这样的串的个数 C_n. 考虑满足上述条件的 n 位二进制串. 如果它的最后一位是 1,那么它的前 $n-1$ 位也构成满足上述要求的串,这样的串有 C_{n-1} 个;如果它的最后一位是 0,那么它的第 $n-1$ 位只能是 1,剩下的 $n-2$ 位构成满足上述要求的串,这样的串有 C_{n-2} 个.

根据加法法则有 $C_n = C_{n-1} + C_{n-2}$,初值 $C_0 = 1$(空串), $C_1 = 2$(串 0 和 1).

不难看出 $C_i = f_{i+1}$. $\{C_n\}$ 和 $\{f_n\}$ 是满足同一个递推式的许多序列中的两个,这些序列的项之间的依赖关系一样,由于初值不同,项不相等. 但是,这些项的表达式具有共同的形式,只是个别参数的取值有所不同. 这个特点将成为求解这类递推方程的基础.

Fibonacci 数列(续)

考虑一个串的计数问题. 有 n 位长的二进制串,如果要求其中没有两个连续的 0,求这样的串的个数 C_n. 考虑满足上述条件的 n 位二进制串. 如果它的最后一位是 1,那么它的前 $n-1$ 位也构成满足上述要求的串,这样的串有 C_{n-1} 个;如果它的最后一位是 0,那么它的第 $n-1$ 位只能是 1,剩下的 $n-2$ 位构成满足上述要求的串,这样的串有 C_{n-2} 个.

根据加法法则有 $C_n = C_{n-1} + C_{n-2}$,初值 $C_0 = 1$(空串), $C_1 = 2$(串 0 和 1). 不难看出 $C_i = f_{i+1}$. $\{C_n\}$ 和 $\{f_n\}$ 是满足同一个递推式的许多序列中的两个,这些序列的项之间的依赖关系一样,由于初值不同,项不相等. 但是,这些项的表达式具有共同的形式,只是个别参数的取值有所不同. 这个特点将成为求解这类递推方程的基础.

上述递推方程可以看成 Fibonacci 数的递归定义,它用前面的项给出第 n 项 f_n 的表达式,但没有显式给出 f_n 的值. 从这个表达式看不出随 n 的增长这个值究竟有多大. 当通过解递推方程了解 f_n 是指数函数时,可以清楚地知道:任何枚举上述 n 位二进制串的算法对于比较大的 n 都是没办法工作的.

11.2 递推方程的公式解法

常系数线性递推方程是一类常用的递推方程，前面 Hanoi 塔和 Fibonacci 数列的递推方程都是常系数线性的递推方程，可以使用公式法求解.

11.2 递推方程的公式解法

常系数线性递推方程是一类常用的递推方程，前面 Hanoi 塔和 Fibonacci 数列的递推方程都是常系数线性的递推方程，可以使用公式法求解.

定义 1.2.1

设递推方程满足

$$\begin{cases} H(n) - a_1 H(n-1) - a_2 H(n-2) - \cdots - a_k H(n-k) = f(n), \\ H(0) = b_0, H(1) = b_1, H(2) = b_2, \cdots, H(k-1) = b_{k-1}, \end{cases} \tag{1.2.1}$$

其中 $a_1, a_2, \cdots, a_k$ 为常数，$a_k \neq 0$，这个方程称作 k 阶常系数线性递推方程.
$b_0, b_1, \cdots, b_{k-1}$ 为 k 个初值. 当 $f(n) = 0$ 时称这个递推方程为齐次方程.

11.2 递推方程的公式解法

常系数线性递推方程是一类常用的递推方程，前面 Hanoi 塔和 Fibonacci 数列的递推方程都是常系数线性的递推方程，可以使用公式法求解.

定义 1.2.1

设递推方程满足

$$\begin{cases} H(n) - a_1 H(n-1) - a_2 H(n-2) - \cdots - a_k H(n-k) = f(n), \\ H(0) = b_0, H(1) = b_1, H(2) = b_2, \cdots, H(k-1) = b_{k-1}, \end{cases} \tag{1.2.1}$$

其中 $a_1, a_2, \cdots, a_k$ 为常数，$a_k \neq 0$，这个方程称作 k 阶常系数线性递推方程. $b_0, b_1, \cdots, b_{k-1}$ 为 k 个初值. 当 $f(n) = 0$ 时称这个递推方程为齐次方程.

上述关于 Hanoi 塔的递推方程不是齐次的，而关于 Fibonacci 数列的递推方程是齐次的.

为了说明常系数线性齐次递推方程的解的结构，下面引入特征根的概念.

定义 1.2.2

设给定常系数线性齐次递推方程如下.

$$\begin{cases} H(n) - a_1 H(n-1) - a_2 H(n-2) - \cdots - a_k H(n-k) = 0, \\ H(0) = b_0, H(1) = b_1, H(2) = b_2, \cdots, H(k-1) = b_{k-1}, \end{cases} \tag{1.2.2}$$

方程 $x^k - a_1 x^{k-1} - \cdots - a_k = 0$ 称作该递推方程的特征方程,特征方程的根称作递推方程的特征根.

定义 1.2.2

设给定常系数线性齐次递推方程如下.

$$\begin{cases} H(n) - a_1 H(n-1) - a_2 H(n-2) - \cdots - a_k H(n-k) = 0, \\ H(0) = b_0, H(1) = b_1, H(2) = b_2, \cdots, H(k-1) = b_{k-1}, \end{cases} \tag{1.2.2}$$

方程 $x^k - a_1 x^{k-1} - \cdots - a_k = 0$ 称作该递推方程的 特征方程 ,特征方程的根称作递推方程的 特征根 .

定理 1.2.1

设 q 是非零复数,则 q^n 是递推方程 (1.2.2) 的解当且仅当 q 是它的特征根.

定义 1.2.2

设给定常系数线性齐次递推方程如下.

$$\begin{cases} H(n) - a_1 H(n-1) - a_2 H(n-2) - \cdots - a_k H(n-k) = 0, \\ H(0) = b_0, H(1) = b_1, H(2) = b_2, \cdots, H(k-1) = b_{k-1}, \end{cases} \tag{1.2.2}$$

方程 $x^k - a_1 x^{k-1} - \cdots - a_k = 0$ 称作该递推方程的 特征方程 ,特征方程的根称作递推方程的 特征根 .

定理 1.2.1

设 q 是非零复数,则 q^n 是递推方程 (1.2.2) 的解当且仅当 q 是它的特征根.

证明.

由定义知, q^n 是递推方程 (1.2.2)的解当且仅当

$q^n - a_1 q^{n-1} - a_2 q^{n-2} - \cdots - a_k q^{n-k} = 0$, 即

$q^{n-k}(q^k - a_1 q^{k-1} - a_2 q^{k-2} - \cdots - a_k) = 0.$

因为 q 是非零复数,所以上式等价于

$q^k - a_1 q^{k-1} - q_2 q^{k-2} - \cdots - a_k = 0$, 即 q 是递推方程 (1.2.2)的特征根. □

定理 1.2.2

设 $h_1(n)$ 和 $h_2(n)$ 是递推方程 (1.2.2) 的解，c_1, c_2 为任意常数，则 $c_1 h_1(n) + c_2 h_2(n)$ 也是这个递推方程的解.

定理 1.2.2

设 $h_1(n)$ 和 $h_2(n)$ 是递推方程 (1.2.2) 的解，c_1, c_2 为任意常数，则 $c_1 h_1(n) + c_2 h_2(n)$ 也是这个递推方程的解.

证明.

将 $c_1 h_1(n) + c_2 h_2(n)$ 代入该递推方程进行验证. □

定理 1.2.2

设 $h_1(n)$ 和 $h_2(n)$ 是递推方程 (1.2.2) 的解，c_1, c_2 为任意常数，则 $c_1 h_1(n) + c_2 h_2(n)$ 也是这个递推方程的解.

证明.

将 $c_1 h_1(n) + c_2 h_2(n)$ 代入该递推方程进行验证. □

根据定理 1.2.1 和定理 1.2.2，对 k 进行归纳，不难得到以下推论.

推论 1.2.1

若 $q_1, q_2, \cdots, q_k$ 是递推方程 (1.2.2) 的特征根，则 $c_1 q_1^n + c_2 q_2^n + \cdots + c_k q_k^n$ 是该递推方程的解，其中 $c_1, c_2, \cdots, c_k$ 是任意常数.

以上推论说明 $c_1 q_1^n + c_2 q_2^n + \cdots + c_k q_k^n$ 是递推方程的解.

通解

除了这种形式的解以外，是否存在其他形式的解？为此，先定义通解.

定义 1.2.3

若对递推方程 (1.2.2) 由不同的初值确定的每个解 $h(n)$ 都存在一组常数 $c_1', c_2', \cdots, c_k'$，使得

$$h(n) = c_1' q_1^n + c_2' q_2^n + \cdots + c_k' q_k^n$$

成立，则称 $c_1' q_1^n + c_2' q_2^n + \cdots + c_k' q_k^n$ 为该递推方程的 通解 .

通解

除了这种形式的解以外，是否存在其他形式的解？为此，先定义通解.

定义 1.2.3

若对递推方程 (1.2.2) 由不同的初值确定的每个解 $h(n)$ 都存在一组常数 $c_1', c_2', \cdots, c_k'$，使得

$$h(n) = c_1' q_1^n + c_2' q_2^n + \cdots + c_k' q_k^n$$

成立，则称 $c_1' q_1^n + c_2' q_2^n + \cdots + c_k' q_k^n$ 为该递推方程的 通解 .

下面的定理说明，当 k 个特征根彼此不等时，上述的解就是递推方程 (1.2.2) 的通解.

定理 1.2.3

设 $q_1, q_2, \cdots, q_k$ 是递推方程 (1.2.2)的 k 个互不相等的特征根，则 $H(n) = c_1 q_1^n + c_2 q_2^n + \cdots + c_k q_k^n$ 是该递推方程的通解.

证明.

根据前面的推论知道 $H(n)$ 是解，下面证明这个解是通解. 设 $h(n)$ 是递推方程 (1.2.2) 的任意一个解，其中 $h(0), h(1), \cdots, h(k-1)$ 由初值 $b_0, b_1, \cdots, b_{k-1}$ 唯一确定. 将初值代入得到以下线性方程组.

$$\begin{cases} c_1 + c_2 + \cdots + c_k = b_0, \\ c_1 q_1 + c_2 q_2 + \cdots + c_k q_k = b_1, \\ \quad \cdots \\ c_1 q_1^{k-1} + c_2 q_2^{k-1} + \cdots + c_k q_k^{k-1} = b_{k-1}. \end{cases}$$

证明.

根据前面的推论知道 $H(n)$ 是解，下面证明这个解是通解. 设 $h(n)$ 是递推方程 (1.2.2) 的任意一个解，其中 $h(0), h(1), \cdots, h(k-1)$ 由初值 $b_0, b_1, \cdots, b_{k-1}$ 唯一确定. 将初值代入得到以下线性方程组.

$$\begin{cases} c_1 + c_2 + \cdots + c_k = b_0, \\ c_1 q_1 + c_2 q_2 + \cdots + c_k q_k = b_1, \\ \quad \cdots \\ c_1 q_1^{k-1} + c_2 q_2^{k-1} + \cdots + c_k q_k^{k-1} = b_{k-1}. \end{cases}$$

由于上述方程组的系数行列式是范德蒙德行列式 $\prod_{1 \leqslant i < j \leqslant k}(q_i - q_j)$，当 $q_i \neq q_j$ 时，这个行列式不等于 0，因此线性方程组有唯一解. 如果这个方程组的唯一解为 $c_1', c_2', \cdots, c_k'$，那么说明 $h(n) = c_1' q_1^n + c_2' q_2^n + \cdots + c_k' q_k^n$，从而证明了 $H(n)$ 是递推方程的通解. □

例 1.2.1

求解 *Fibonacci* 数列的递推方程.

例 1.2.1

求解 *Fibonacci* 数列的递推方程.

解

递推方程是 $f_n = f_{n-1} + f_{n-2}$, 初值是 $f_0 = 1, f_1 = 1$. 它的特征方程是 $x^2 - x - 1 = 0$, 求解得到特征根为 $\frac{1+\sqrt{5}}{2}, \frac{1-\sqrt{5}}{2}$. 因此, 递推方程的通解为

$$f_n = c_1 \left(\frac{1+\sqrt{5}}{2}\right)^n + c_2 \left(\frac{1-\sqrt{5}}{2}\right)^n,$$

其中 c_1, c_2 为待定常数. 代入初值 $f_0 = 1, f_1 = 1$, 得

$$\begin{cases} c_1 + c_2 = 1, \\ c_1 \cdot \dfrac{1+\sqrt{5}}{2} + c_2 \cdot \dfrac{1-\sqrt{5}}{2} = 1. \end{cases}$$

解得待定常数 $c_1 = \frac{1}{\sqrt{5}} \cdot \frac{1+\sqrt{5}}{2}$, $c_2 = -\frac{1}{\sqrt{5}} \cdot \frac{1-\sqrt{5}}{2}$. 从而得到递推方程的解为

$$f_n = \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2}\right)^{n+1} - \frac{1}{\sqrt{5}} \left(\frac{1-\sqrt{5}}{2}\right)^{n+1}.$$

$$\begin{cases} H(n) - a_1 H(n-1) - a_2 H(n-2) - \cdots - a_k H(n-k) = 0, \\ H(0) = b_0, H(1) = b_1, H(2) = b_2, \cdots, H(k-1) = b_{k-1}. \end{cases}$$

上面递推方程 (1.2.2) 的特征根中如果存在重根，当把对应这些特征根的项 q_i^n 进行线性组合时，那些对应于同一个重根的项就归并成一项. 于是，当把这个通解代入初值时，所得到的线性方程组中方程的个数将比未知数的个数多. 这样的方程组可能无解.

解决的方法是必须使用线性无关的解来构造通解. 定理 1.2.4 给出通解的表达式，这里不再给出证明.

$$\begin{cases} H(n) - a_1 H(n-1) - a_2 H(n-2) - \cdots - a_k H(n-k) = 0, \\ H(0) = b_0, H(1) = b_1, H(2) = b_2, \cdots, H(k-1) = b_{k-1}. \end{cases}$$

上面递推方程 (1.2.2) 的特征根中如果存在重根，当把对应这些特征根的项 q_i^n 进行线性组合时，那些对应于同一个重根的项就归并成一项. 于是，当把这个通解代入初值时，所得到的线性方程组中方程的个数将比未知数的个数多. 这样的方程组可能无解.

解决的方法是必须使用线性无关的解来构造通解. 定理 1.2.4 给出通解的表达式，这里不再给出证明.

定理 1.2.4

设 $q_1, q_2, \cdots, q_t$ 是递推方程 (1.2.2) 的互不相等的特征根，且 q_i 的重数为 m_i，其中 $i = 1, 2, \cdots, t$. 令

$$H_i(n) = (c_{i1} + c_{i2} n + \cdots + c_{im_i} n^{m_i - 1}) q_i^n,$$

那么该递推方程的通解是

$$H(n) = \sum_{i=1}^{t} H_i(n).$$

例 1.2.2

求解以下递推方程

$$\begin{cases} H(n) - 3H(n-1) + 4H(n-3) = 0, \\ H(0) = 1, H(1) = 0, H(2) = 0. \end{cases}$$

例 1.2.2

求解以下递推方程

$$\begin{cases} H(n) - 3H(n-1) + 4H(n-3) = 0, \\ H(0) = 1, H(1) = 0, H(2) = 0. \end{cases}$$

解

特征方程为 $x^3 - 3x^2 + 4 = 0$，特征根为 $-1, 2, 2$，通解为

$$H(n) = (c_1 + c_2 n)2^n + c_3(-1)^n,$$

其中待定常数满足以下方程组

$$\begin{cases} c_1 + c_3 = 1, \\ 2c_1 + 2c_2 - c_3 = 0, \\ 4c_1 + 8c_2 + c_3 = 0. \end{cases}$$

解得 $c_1 = \frac{5}{9}, c_2 = -\frac{1}{3}, c_3 = \frac{4}{9}$，原方程的解为

$$H(n) = \frac{5}{9}2^n - \frac{1}{3}n2^n + \frac{4}{9}(-1)^n.$$

常系数线性非齐次递推方程

常系数线性非齐次递推方程的标准型是

$$H(n) - a_1 H(n-1) - \cdots - a_k H(n-k) = f(n), \tag{1.2.3}$$

其中 $n \geqslant k, a_k \neq 0, f(n) \neq 0.$

为了求解上述方程，必须了解通解的结构.

常系数线性非齐次递推方程

常系数线性非齐次递推方程的标准型是

$$H(n) - a_1 H(n-1) - \cdots - a_k H(n-k) = f(n), \tag{1.2.3}$$

其中 $n \geqslant k, a_k \neq 0, f(n) \neq 0$.

为了求解上述方程，必须了解通解的结构.

定理 1.2.5

设 $\bar{H}(n)$ 是对应的齐次方程 (1.2.2) 的通解，$H^*(n)$ 是一个特解，则

$$H(n) = \bar{H}(n) + H^*(n)$$

是递推方程 (1.2.3) 的通解.

常系数线性非齐次递推方程

常系数线性非齐次递推方程的标准型是

$$H(n) - a_1 H(n-1) - \cdots - a_k H(n-k) = f(n), \tag{1.2.3}$$

其中 $n \geqslant k, a_k \neq 0, f(n) \neq 0$.

为了求解上述方程，必须了解通解的结构.

定理 1.2.5

设 $\bar{H}(n)$ 是对应的齐次方程 (1.2.2) 的通解，$H^*(n)$ 是一个特解，则

$$H(n) = \bar{H}(n) + H^*(n)$$

是递推方程 (1.2.3) 的通解.

定理 1.2.5 说明递推方程 (1.2.3) 的通解结构是对应的齐次方程的通解加上一个特解，而特解的形式依赖于 $f(n)$. 求解的关键是确定一个特解，可以先根据 $f(n)$ 写出特解的函数形式，然后用待定系数法确定其中的系数.

定理1.2.5(续)

证明.

将 $H(n)$ 代入就可以验证它是递推方程 (1.2.3) 的解. 下面证明它是通解.

设 $h(n)$ 是递推方程 (1.2.3) 的一个解,只需证明 $h(n)$ 可以表示为对应齐次方程的一个解与特解 $H^*(n)$ 之和. 因为 $h(n)$ 与 $H^*(n)$ 都是递推方程 (1.2.3) 的解,因此

$$h(n) - a_1 h(n-1) - \cdots - a_k h(n-k) = f(n),$$
$$H^*(n) - a_1 H^*(n-1) - \cdots - a_k H^*(n-k) = f(n).$$

将以上两个式子相减得

$$[h(n) - H^*(n)] - a_1[h(n-1) - H^*(n-1)] - \cdots - a_k[h(n-k) - H^*(n-k)] = 0.$$

这说明 $h(n) - H^*(n)$ 是对应齐次方程的一个解,换句话说,$h(n)$ 是对应齐次方程的一个解与特解 $H^*(n)$ 之和. □

下面针对递推方程 (1.2.3)中 $f(n)$ 的某些特殊形式进行讨论.

第一种情况:$f(n)$ 为 n 的 t 次多项式,那么特解一般也为 n 的 t 次多项式. 但是如果递推方程的特征根是 1,就必须提高所设定特解的多项式次数. 因为当把这个特解代入原方程时,最高次项和常数项都会消去,于是化简后等式左边多项式的次数将低于右边函数 $f(n)$ 的次数.

下面针对递推方程 (1.2.3)中 $f(n)$ 的某些特殊形式进行讨论.

第一种情况:$f(n)$ 为 n 的 t 次多项式,那么特解一般也为 n 的 t 次多项式. 但是如果递推方程的特征根是 1,就必须提高所设定特解的多项式次数. 因为当把这个特解代入原方程时,最高次项和常数项都会消去,于是化简后等式左边多项式的次数将低于右边函数 $f(n)$ 的次数.

例 1.2.3

估计下面的顺序插入排序算法 *Insertsort*$(\boldsymbol{A}, \boldsymbol{n})$ 在最坏情况下的时间复杂度 $W(n)$.

```
1 for j ← 2 to n do
2     x ← A[j]
3       i ← j − 1
4       while i > 0 and A[i] > x
        do
5         A[i + 1] ← A[i]
6           i ← i − 1
7       A[i + 1] ← x
```

算法行 4–7 将 $A[j]$ 插入 $A[1..j-1]$. 对 n 个数的数组 A 进行排序,算法对 A 的第 1 项不做任何工作. 接着只需 1 次比较就可以把第 2 项插到恰当的位置. 然后算法将第 3 项插入由排好序的前 2 项构成的子数组中,这至多需要 2 次比较. 以此类推,如果前 $j-1$ 项已经排好,插入第 j 项至多需要 $j-1$ 次比较. 因此,对于 n 个数的数组,在

例1.2.3(续)

最坏情况下算法所做的比较次数 $W(n)$ 满足以下递推方程:

$$\begin{cases} W(n) = W(n-1) + n - 1, \\ W(1) = 0. \end{cases}$$

整理成标准型

$$\begin{cases} W(n) - W(n-1) = n - 1, \\ W(1) = 0. \end{cases}$$

这里函数 $f(n)$ 是 n 的一次多项式,但是特征根是 1. 若把特解设为 $W^*(n) = P_1 n + P_2$,将它代入递推方程得

$$P_1 n + P_2 - [P_1(n-1) + P_2] = n - 1.$$

化简得 $P_1 = n - 1$,左边是 n 的 0 次多项式,右边是 n 的 1 次多项式. 没有常数 P_1 能够使它成立.

例1.2.3(续)

最坏情况下算法所做的比较次数 $W(n)$ 满足以下递推方程:

$$\begin{cases} W(n) = W(n-1) + n - 1, \\ W(1) = 0. \end{cases}$$

整理成标准型

$$\begin{cases} W(n) - W(n-1) = n - 1, \\ W(1) = 0. \end{cases}$$

这里函数 $f(n)$ 是 n 的一次多项式,但是特征根是 1. 若把特解设为 $W^*(n) = P_1 n + P_2$,将它代入递推方程得

$$P_1 n + P_2 - [P_1(n-1) + P_2] = n - 1.$$

化简得 $P_1 = n - 1$,左边是 n 的 0 次多项式,右边是 n 的 1 次多项式. 没有常数 P_1 能够使它成立.

根据上面的分析,将特解设为 $W^*(n) = P_1 n^2 + P_2 n$,代入递推方程得

$$(P_1 n^2 + P_2 n) - [P_1(n-1)^2 + P_2(n-1)] = n - 1.$$

化简得 $2P_1 n - P_1 + P_2 = n - 1$, 解得 $P_1 = 1/2, P_2 = -1/2$.

于是通解为 $W(n) = c \cdot 1^n + n(n-1)/2 = c + n(n-1)/2$.

代入初值 $W(1) = 0$,得 $c = 0$,最终得到 $W(n) = n(n-1)/2$. 这说明插入排序在最坏情况下是 $O(n^2)$ 的算法.

例 1.2.4

Hanoi 塔问题的递推方程是

$$H(n) = 2H(n-1) + 1,$$

例 1.2.4

Hanoi 塔问题的递推方程是

$$H(n) = 2H(n-1) + 1,$$

这是一个含有 *0* 次多项式函数的非齐次的递推方程. 设特解为 $H^*(n) = P$, 代入原方程得

$$P = 2P + 1.$$

因此 $P = -1$. 从而得到递推方程的通解是

$$H(n) = c2^n - 1.$$

代入初值 $H(1) = 1$, 得 $c = 1$, 解为 $H(n) = 2^n - 1$. □

第二种情况：$f(n)$ 为指数函数 $A\beta^n$，这里的 A 代表某个常数. 若 β 不是特征根，则特解为 $P\beta^n$，其中 P 为待定系数；若 β 是 $m(\geqslant 1)$ 重特征根，则特解为 $Pn^m\beta^n$.

第二种情况：$f(n)$ 为指数函数 $A\beta^n$，这里的 A 代表某个常数. 若 β 不是特征根，则特解为 $P\beta^n$，其中 P 为待定系数；若 β 是 $m(\geqslant 1)$ 重特征根，则特解为 $Pn^m\beta^n$.

例 1.2.5

求解下述递推方程

$$\begin{cases} a_n - 4a_{n-1} + 4a_{n-2} = 2^n, \\ a_0 = 1, a_1 = 5. \end{cases}$$

第二种情况：$f(n)$ 为指数函数 $A\beta^n$，这里的 A 代表某个常数. 若 β 不是特征根，则特解为 $P\beta^n$，其中 P 为待定系数；若 β 是 $m(\geqslant 1)$ 重特征根，则特解为 $Pn^m\beta^n$.

例 1.2.5

求解下述递推方程

$$\begin{cases} a_n - 4a_{n-1} + 4a_{n-2} = 2^n, \\ a_0 = 1, a_1 = 5. \end{cases}$$

解

因为 2 是对应齐次方程的二重特征根，因此特解 $a_n^* = Pn^2 2^n$，代入递推方程得 $Pn^2 2^n - 4P(n-1)^2 2^{n-1} + 4P(n-2)^2 2^{n-2} = 2^n$, 解得 $P = 1/2$.

因此原递推方程的通解是 $a_n = c_1 2^n + c_2 n 2^n + n^2 2^{n-1}$. 代入初值得

$$\begin{cases} c_1 = 1, \\ 2c_1 + 2c_2 + 1 = 5. \end{cases}$$

解得 $c_1 = c_2 = 1$，从而得到递推方程的解 $a_n = 2^n + n2^n + n^2 2^{n-1}$. □

11.3 递推方程的其他解法

换元法 的基本思想就是将原来关于某个变元的递推方程通过函数变换转变成关于其他变元的常系数线性递推方程，然后使用公式法求解. 在得到解以后，再利用相反的变换将解转变成关于原来变元的函数.

11.3 递推方程的其他解法

换元法 的基本思想就是将原来关于某个变元的递推方程通过函数变换转变成关于其他变元的常系数线性递推方程，然后使用公式法求解. 在得到解以后，再利用相反的变换将解转变成关于原来变元的函数.

例 1.3.1

二分归并排序算法 .

这个算法的主要思想是：将被排序的数组划分成相等的两个子数组，然后递归使用同样的算法分别对两个子数组排序，最后将两个排好序的子数组归并成一个数组.

归并的过程如下：假设两个子数组是 A 和 B，它们的元素都按照从小到大的顺序排列. 将 A 与 B 归并后的数组记作 C. 设定两个指针 p_1, p_2，初始分别指向 A 和 B 的最小元素. 归并时只需比较 p_1 和 p_2 指向的元素，哪个元素小，就把它从原来的数组移到 C，原来指向它的指针向后移动一个位置. 如果 A 或 B 中有一个数组(如 A)的元素已经被全部移走，那么比较过程结束，剩下的工作就是将 B 中剩下的元素顺序移到 C 的后面.

例1.3.1(续)

算法描述如下.

算法 Mergesort(A, p, r) //对数组 A 的下标 p 到 r 之间的数排序

```
1 if p < r then
2     q ← ⌊(p + r)/2⌋    //q 为 p 到 r 的中点, ⌊x⌋ 是不超过 x 的最大整数
3     Mergesort(A, p, q)
4     Mergesort(A, q + 1, r)
5     Merge(A, p, q, r)    //把排好序的数组 A[p..q] 与 A[q + 1..r] 归并.
```

如果 $n = 2^k$, 以比较作为基本运算, 试给出最坏情况下归并排序算法的时间复杂度函数.

例1.3.1(续)

算法描述如下.

算法 Mergesort(A, p, r) //对数组 A 的下标 p 到 r 之间的数排序

```
1 if p < r then
2     q ← ⌊(p + r)/2⌋    //q 为 p 到 r 的中点, ⌊x⌋ 是不超过 x 的最大整数
3     Mergesort(A, p, q)
4     Mergesort(A, q + 1, r)
5     Merge(A, p, q, r)    //把排好序的数组 A[p..q] 与 A[q + 1..r] 归并.
```

如果 $n = 2^k$, 以比较作为基本运算, 试给出最坏情况下归并排序算法的时间复杂度函数.

解

设 $W(n)$ 表示归并排序算法在最坏情况下所做的比较次数, 根据上面的分析, 分别排序 A 和 B 的总工作量为 $2W(n/2)$. 在归并 A 和 B 时, 每比较 1 次, 就可以移走 1 个元素. 因为 A 和 B 总共有 n 个元素, 至多需要 $n-1$ 次比较, 就可以完成归并工作. 因此, 对 n 个数进行二分归并排序在最坏情况下的比较次数

例1.3.1解(续)

满足如下递推方程:

$$\begin{cases} W(n) = 2\,W(n/2) + n - 1, n = 2^k, \\ W(1) = 0. \end{cases}$$

将 $n = 2^k$ 代入,该递推方程可以转换成关于变元 k 的常系数线性递推方程,即

$$\begin{cases} H(k) = 2H(k-1) + 2^k - 1, \\ H(0) = 0. \end{cases}$$

例1.3.1解(续)

满足如下递推方程:

$$\begin{cases} W(n) = 2\,W(n/2) + n - 1, n = 2^k, \\ W(1) = 0. \end{cases}$$

将 $n = 2^k$ 代入,该递推方程可以转换成关于变元 k 的常系数线性递推方程,即

$$\begin{cases} H(k) = 2H(k-1) + 2^k - 1, \\ H(0) = 0. \end{cases}$$

该方程是非齐次的,其函数部分是 $2^k - 1$,为指数函数 2^k 与多项式函数 -1 之和,因此特解也是指数函数与多项式函数的组合形式. 由于 2 是特征根,令

$$H^*(k) = P_1 k 2^k + P_2,$$

将这个特解代入原方程,解得 $P_1 = P_2 = 1$,

例1.3.1解(续)

满足如下递推方程:

$$\begin{cases} W(n) = 2W(n/2) + n - 1, n = 2^k, \\ W(1) = 0. \end{cases}$$

将 $n = 2^k$ 代入,该递推方程可以转换成关于变元 k 的常系数线性递推方程,即

$$\begin{cases} H(k) = 2H(k-1) + 2^k - 1, \\ H(0) = 0. \end{cases}$$

该方程是非齐次的,其函数部分是 $2^k - 1$,为指数函数 2^k 与多项式函数 -1 之和,因此特解也是指数函数与多项式函数的组合形式. 由于 2 是特征根,令

$$H^*(k) = P_1 k 2^k + P_2,$$

将这个特解代入原方程,解得 $P_1 = P_2 = 1$,

从而得到

$$H^*(k) = k2^k + 1.$$

根据特解得到通解

$$H(k) = c2^k + k2^k + 1.$$

代入初值,得 $c = -1$,因此得到原方程的解

$$H(k) = -2^k + k2^k + 1.$$

将 $k = \log n$ 代入得

$$W(n) = n \log n - n + 1,$$

这正好验证了

$W(n) = O(n \log n)$.

下面考虑迭代归纳法. 所谓迭代,就是从原始递推方程开始,利用方程所表达的数列中后项对前项的依赖关系,把表达式中的后项用相等的前项的表达式代入,直到表达式中没有函数项为止. 这时等式右边可能是一系列迭代后的项之和,然后,将右边的项求和并将结果进行化简. 为了保证结果的正确性,往往需要代入原递推方程进行验证.

下面考虑迭代归纳法. 所谓迭代,就是从原始递推方程开始,利用方程所表达的数列中后项对前项的依赖关系,把表达式中的后项用相等的前项的表达式代入,直到表达式中没有函数项为止. 这时等式右边可能是一系列迭代后的项之和,然后,将右边的项求和并将结果进行化简. 为了保证结果的正确性,往往需要代入原递推方程进行验证.

下面用迭代归纳法求解归并排序的递推方程:

$$\begin{cases} W(n) = 2\,W(n/2) + n - 1,\ n = 2^k, \\ W(1) = 0. \end{cases}$$

下面考虑迭代归纳法. 所谓迭代,就是从原始递推方程开始,利用方程所表达的数列中后项对前项的依赖关系,把表达式中的后项用相等的前项的表达式代入,直到表达式中没有函数项为止. 这时等式右边可能是一系列迭代后的项之和,然后,将右边的项求和并将结果进行化简. 为了保证结果的正确性,往往需要代入原递推方程进行验证.

下面用迭代归纳法求解归并排序的递推方程:

$$\begin{cases} W(n) = 2\,W(n/2) + n - 1, \ n = 2^k, \\ W(1) = 0. \end{cases}$$

求解过程如下.

$$\begin{aligned} W(n) &= 2\,W(2^{k-1}) + 2^k - 1 = 2[2\,W(2^{k-2}) + 2^{k-1} - 1] + 2^k - 1 \\ &= 2^2\,W(2^{k-2}) + 2^k - 2 + 2^k - 1 \\ &= 2^2[2\,W(2^{k-3}) + 2^{k-2} - 1] + 2^k - 2 + 2^k - 1 \\ &= 2^3\,W(2^{k-3}) + 2^k - 2^2 + 2^k - 2 + 2^k - 1 \\ &= \cdots \\ &= 2^k\,W(1) + k2^k - (2^{k-1} + 2^{k-2} + \cdots + 2 + 1) \\ &= k2^k - 2^k + 1 = n\log n - n + 1. \end{aligned}$$

对结果进行验证. 使用数学归纳法. 把 $n = 1$ 代入上述公式得

$$W(1) = 1 \log 1 - 1 + 1 = 0,$$

符合初始条件.

对结果进行验证. 使用数学归纳法. 把 $n = 1$ 代入上述公式得

$$W(1) = 1 \log 1 - 1 + 1 = 0,$$

符合初始条件.

假设对于任何小于 n 的正整数 t, $W(t)$ 都是正确的, 将结果代入原递推方程的右边得

$$\begin{aligned} 2\,W(n/2) + n - 1 &= 2(2^{k-1} \log 2^{k-1} - 2^{k-1} + 1) + 2^k - 1 \\ &= 2^k(k-1) - 2^k + 2 + 2^k - 1 \\ &= k2^k - 2^k + 1 \\ &= n \log n - n + 1 \\ &= W(n), \end{aligned}$$

这说明得到的解满足原来的递推方程.

迭代方法一般适用于一阶递推方程，对于某些二阶以上的递推方程，需要先化简.

迭代方法一般适用于一阶递推方程，对于某些二阶以上的递推方程，需要先化简.

例 1.3.2

在例 ??中，我们使用包含排斥原理求解了错位排列的计数问题，下面用递推方程来求解这个问题.

迭代方法一般适用于一阶递推方程，对于某些二阶以上的递推方程，需要先化简.

例 1.3.2

在例 ??中，我们使用包含排斥原理求解了错位排列的计数问题，下面用递推方程来求解这个问题.

解

将 n 位的错位排列按照它的第一位是 $2, 3, \cdots, n$ 分成 $n-1$ 个组，不难看出每个组的排列个数一样多. 考虑其中的一组，不妨设它的第 1 位是 2. 那么它的第 2 位可能是 1，也可能不是 1. 如果第 2 位是 1，那么剩下的 $n-2$ 位是 $3, 4, \cdots, n$ 的错位排列，有 D_{n-2} 个；如果第 2 位不是 1，那么从第 2 位到第 n 位构成 $1, 3, 4, \cdots, n$ 的错位排列，有 D_{n-1} 个. 根据这个分析得到下述递推方程和初值.

$$\begin{cases} D_n = (n-1)(D_{n-1} + D_{n-2}), \\ D_1 = 0, D_2 = 1. \end{cases}$$

这个方程不是常系数线性的，不能使用公式法. 由于它是二阶的，如果直接迭代，所得到的项太多，求和比较困难. 这里先使用差消的方法把它转换为一阶方程.

例1.3.2解(续)

因为

$$\begin{aligned} D_n - nD_{n-1} &= -[D_{n-1} - (n-1)D_{n-2}] = \cdots \\ &= (-1)^{n-2}(D_2 - 2D_1) = (-1)^{n-2}, \end{aligned}$$

从而得到一阶递推方程

$$\begin{cases} D_n = nD_{n-1} + (-1)^n, \\ D_1 = 0. \end{cases}$$

例1.3.2解(续)

因为

$$\begin{aligned} D_n - nD_{n-1} &= -[D_{n-1} - (n-1)D_{n-2}] = \cdots \\ &= (-1)^{n-2}(D_2 - 2D_1) = (-1)^{n-2}, \end{aligned}$$

从而得到一阶递推方程

$$\begin{cases} D_n = nD_{n-1} + (-1)^n, \\ D_1 = 0. \end{cases}$$

不断迭代得

$$\begin{aligned} D_n &= n(n-1)D_{n-2} + n(-1)^{n-1} + (-1)^n \\ &= n(n-1)(n-2)D_{n-3} + n(n-1)(-1)^{n-2} + n(-1)^{n-1} + (-1)^n \\ &= \cdots \\ &= n(n-1)\cdots 2D_1 + n(n-1)\cdots 3(-1)^2 + \cdots + n(-1)^{n-1} + (-1)^n \\ &= n!\left[1 - \frac{1}{1!} + \frac{1}{2!} - \cdots + (-1)^n \frac{1}{n!}\right]. \end{aligned}$$

这个结果与第 1 章使用包含排斥原理得到的结果完全一样.

使用 差消法 也可以将某些高阶递推方程化简为一阶递推方程. 下面的例子是关于快速排序算法平均情况下复杂度 $T(n)$ 的递推方程.

使用 差消法 也可以将某些高阶递推方程化简为一阶递推方程. 下面的例子是关于快速排序算法平均情况下复杂度 $T(n)$ 的递推方程.

快速排序算法的基本思想是:设要排序的数组是 $A[p..r]$,不妨假设 A 中的元素彼此不等. 以 A 的首元素 $A[p]$ 作为标准,对 A 进行划分,使得所有小于 $A[p]$ 的元素构成子数组 A_1,所有大于 $A[p]$ 的元素构成子数组 A_2. 需要说明的是,这个划分过程 Partition 并不对子数组 A_1 和 A_2 进行排序,只是把原来规模为 n 的问题转变成两个小规模的子问题. 然后算法分别递归地对 A_1 和 A_2 进行排序. 这个递归调用过程一直进行下去,直到子问题的数组只含有 1 个元素为止.

使用 差消法 也可以将某些高阶递推方程化简为一阶递推方程. 下面的例子是关于快速排序算法平均情况下复杂度 $T(n)$ 的递推方程.

快速排序算法的基本思想是:设要排序的数组是 $A[p..r]$,不妨假设 A 中的元素彼此不等. 以 A 的首元素 $A[p]$ 作为标准,对 A 进行划分,使得所有小于 $A[p]$ 的元素构成子数组 A_1,所有大于 $A[p]$ 的元素构成子数组 A_2. 需要说明的是,这个划分过程 Partition 并不对子数组 A_1 和 A_2 进行排序,只是把原来规模为 n 的问题转变成两个小规模的子问题. 然后算法分别递归地对 A_1 和 A_2 进行排序. 这个递归调用过程一直进行下去,直到子问题的数组只含有 1 个元素为止.

下面给出 快速排序算法 的伪码描述.

算法 Quicksort(A, p, r) // p 和 r 分别表示数组 A 的首元素和末元素的下标

```
1 if p < r then
2     q ← Partition(A, p, r)   //划分数组 A 为 A[p..q − 1] 和 A[q + 1, ..r]
3     A[p] ↔ A[q]
4     Quicksort(A, p, q − 1)
5     Quicksort(A, q + 1, r)
```

其中的划分过程 Partition 描述如下.

算法 Partition(A, p, r)

```
1 x ← A[p]   //选首元素作为划分标准 x
2 i ← p − 1
3 j ← r + 1
4 while true do
5     repeat j ← j − 1 until A[j] < x   //A[j] 是从后向前找到的第一个比 x 小的元素;
6     repeat i ← i + 1 until A[x] > x   //A[i] 是从前向后找到的第一个比 x 大的元素;
7     if i < j then
8         A[i] ↔ A[j]   //A[i] 与 A[j] 交换,回到行 4 继续搜索 A[i] 到 A[j] 之间的范围
9     else return j   //结束 while 循环;
```

考虑下面的划分实例，设数组 $A[1..13]$ 初始是

$\underline{27}$ 99 0 8 13 64 86 16 7 10 88 25 90

A 的首元素是 27，它就是划分数组的标准. 算法 Quicksort 先调用 Partition 过程，Partition 过程的步骤 5 从后向前寻找第一个比 27 小的数，第一次找到 25；然后步骤 6 从前向后找第一个比 27 大的数，就是 99. 这时 $i = 2, A[2] = 99$, $j = 12, A[12] = 25$, 过程进行到步骤 8，99 和 25 交换. 交换后的数组变成

$\underline{27}$ 25 0 8 13 64 86 16 7 10 88 99 90

考虑下面的划分实例，设数组 $A[1..13]$ 初始是

$$\underline{27} \quad 99 \quad 0 \quad 8 \quad 13 \quad 64 \quad 86 \quad 16 \quad 7 \quad 10 \quad 88 \quad 25 \quad 90$$

A 的首元素是 27，它就是划分数组的标准. 算法 Quicksort 先调用 Partition 过程，Partition 过程的步骤 5 从后向前寻找第一个比 27 小的数，第一次找到 25；然后步骤 6 从前向后找第一个比 27 大的数，就是 99. 这时 $i = 2, A[2] = 99$, $j = 12, A[12] = 25$, 过程进行到步骤 8，99 和 25 交换. 交换后的数组变成

$$\underline{27} \quad 25 \quad 0 \quad 8 \quad 13 \quad 64 \quad 86 \quad 16 \quad 7 \quad 10 \quad 88 \quad 99 \quad 90$$

接着，过程 Partition 回到步骤 4，从当前交换的位置继续向中间搜索，寻找下一对交换的位置. 不难看到，下一次需要交换的是 64 和 10，交换后的数组是

$$\underline{27} \quad 25 \quad 0 \quad 8 \quad 13 \quad 10 \quad 86 \quad 16 \quad 7 \quad 64 \quad 88 \quad 99 \quad 90$$

考虑下面的划分实例，设数组 $A[1..13]$ 初始是

$$\underline{27} \quad 99 \quad 0 \quad 8 \quad 13 \quad 64 \quad 86 \quad 16 \quad 7 \quad 10 \quad 88 \quad 25 \quad 90$$

A 的首元素是 27，它就是划分数组的标准. 算法 Quicksort 先调用 Partition 过程，Partition 过程的步骤 5 从后向前寻找第一个比 27 小的数，第一次找到 25；然后步骤 6 从前向后找第一个比 27 大的数，就是 99. 这时 $i = 2, A[2] = 99,$ $j = 12, A[12] = 25,$ 过程进行到步骤 8，99 和 25 交换. 交换后的数组变成

$$\underline{27} \quad 25 \quad 0 \quad 8 \quad 13 \quad 64 \quad 86 \quad 16 \quad 7 \quad 10 \quad 88 \quad 99 \quad 90$$

接着，过程 Partition 回到步骤 4，从当前交换的位置继续向中间搜索，寻找下一对交换的位置. 不难看到，下一次需要交换的是 64 和 10，交换后的数组是

$$\underline{27} \quad 25 \quad 0 \quad 8 \quad 13 \quad 10 \quad 86 \quad 16 \quad 7 \quad 64 \quad 88 \quad 99 \quad 90$$

继续这个过程，交换 86 和 7，得到

$$\underline{27} \quad 25 \quad 0 \quad 8 \quad 13 \quad 10 \quad 7 \quad 16 \quad 86 \quad 64 \quad 88 \quad 99 \quad 90$$

考虑下面的划分实例，设数组 $A[1..13]$ 初始是

$\underline{27}$ 99 0 8 13 64 86 16 7 10 88 25 90

A 的首元素是 27，它就是划分数组的标准. 算法 Quicksort 先调用 Partition 过程，Partition 过程的步骤 5 从后向前寻找第一个比 27 小的数，第一次找到 25；然后步骤 6 从前向后找第一个比 27 大的数，就是 99. 这时 $i = 2, A[2] = 99$, $j = 12, A[12] = 25$, 过程进行到步骤 8，99 和 25 交换. 交换后的数组变成

$\underline{27}$ 25 0 8 13 64 86 16 7 10 88 99 90

接着，过程 Partition 回到步骤 4，从当前交换的位置继续向中间搜索，寻找下一对交换的位置. 不难看到，下一次需要交换的是 64 和 10，交换后的数组是

$\underline{27}$ 25 0 8 13 10 86 16 7 64 88 99 90

继续这个过程，交换 86 和 7，得到

$\underline{27}$ 25 0 8 13 10 7 16 86 64 88 99 90

下面的搜索将导致 i 大于 j，步骤 7 的条件不再满足，过程 Partition 结束，返回 $q = 8$. 算法 Quicksort 继续进行第 3 行，把 27 和 16 交换，得到

16 25 0 8 13 10 7 $\underline{27}$ 86 64 88 99 90

考虑下面的划分实例，设数组 $A[1..13]$ 初始是

$$\underline{27} \quad 99 \quad 0 \quad 8 \quad 13 \quad 64 \quad 86 \quad 16 \quad 7 \quad 10 \quad 88 \quad 25 \quad 90$$

A 的首元素是 27，它就是划分数组的标准. 算法 Quicksort 先调用 Partition 过程，Partition 过程的步骤 5 从后向前寻找第一个比 27 小的数，第一次找到 25；然后步骤 6 从前向后找第一个比 27 大的数，就是 99. 这时 $i = 2, A[2] = 99$, $j = 12, A[12] = 25$, 过程进行到步骤 8，99 和 25 交换. 交换后的数组变成

$$\underline{27} \quad 25 \quad 0 \quad 8 \quad 13 \quad 64 \quad 86 \quad 16 \quad 7 \quad 10 \quad 88 \quad 99 \quad 90$$

接着，过程 Partition 回到步骤 4，从当前交换的位置继续向中间搜索，寻找下一对交换的位置. 不难看到，下一次需要交换的是 64 和 10，交换后的数组是

$$\underline{27} \quad 25 \quad 0 \quad 8 \quad 13 \quad 10 \quad 86 \quad 16 \quad 7 \quad 64 \quad 88 \quad 99 \quad 90$$

继续这个过程，交换 86 和 7，得到

$$\underline{27} \quad 25 \quad 0 \quad 8 \quad 13 \quad 10 \quad 7 \quad 16 \quad 86 \quad 64 \quad 88 \quad 99 \quad 90$$

下面的搜索将导致 i 大于 j，步骤 7 的条件不再满足，过程 Partition 结束，返回 $q = 8$. 算法 Quicksort 继续进行第 3 行，把 27 和 16 交换，得到

$$16 \quad 25 \quad 0 \quad 8 \quad 13 \quad 10 \quad 7 \quad \underline{27} \quad 86 \quad 64 \quad 88 \quad 99 \quad 90$$

以 27 为界，原来数组被划分成两个子数组：$[16, 25, 0, 8, 13, 10, 7]$ 与 $[86, 64, 88, 99, 90]$. 算法 Quicksort 步骤 4、步骤 5 接着递归地对这两个子数组继续排序.

考虑算法 Quicksort 在平均情况下的时间复杂度. 如果首元素处在第 k 个位置, 那么划分后的两个子问题的规模分别为 $k-1$ 和 $n-k$, 其中 $k=1,2,\cdots,n$. 假定这 n 种情况出现的可能性相等, 为计算平均时间复杂度, 只需要对这 n 种情况分别计算各自的比较次数, 然后取平均值即可.

考虑算法 Quicksort 在平均情况下的时间复杂度. 如果首元素处在第 k 个位置, 那么划分后的两个子问题的规模分别为 $k-1$ 和 $n-k$, 其中 $k=1,2,\cdots,n$. 假定这 n 种情况出现的可能性相等, 为计算平均时间复杂度, 只需要对这 n 种情况分别计算各自的比较次数, 然后取平均值即可.

设 Quicksort 算法对于规模为 n 的输入的平均比较次数是 $T(n)$. 若首元素处在第 k 个位置, 划分后的子问题规模分别为 $k-1$ 和 $n-k$, 平均比较次数分别是 $T(k-1)$ 和 $T(n-k)$, 而划分过程 Partition 需要的比较次数是 $O(n)$, 因此总的比较次数为 $T(k-1)+T(n-k)+O(n)$. 对 $k=1,2,\cdots,n$ 求和, 去掉 $T(0)=0$ 的项, 就得到 $2\sum_{k=1}^{n-1}T(k)+O(n^2)$.

考虑算法 Quicksort 在平均情况下的时间复杂度. 如果首元素处在第 k 个位置, 那么划分后的两个子问题的规模分别为 $k-1$ 和 $n-k$, 其中 $k=1,2,\cdots,n$. 假定这 n 种情况出现的可能性相等, 为计算平均时间复杂度, 只需要对这 n 种情况分别计算各自的比较次数, 然后取平均值即可.

设 Quicksort 算法对于规模为 n 的输入的平均比较次数是 $T(n)$. 若首元素处在第 k 个位置, 划分后的子问题规模分别为 $k-1$ 和 $n-k$, 平均比较次数分别是 $T(k-1)$ 和 $T(n-k)$, 而划分过程 Partition 需要的比较次数是 $O(n)$, 因此总的比较次数为 $T(k-1)+T(n-k)+O(n)$. 对 $k=1,2,\cdots,n$ 求和, 去掉 $T(0)=0$ 的项, 就得到 $2\sum_{k=1}^{n-1}T(k)+O(n^2)$.

把上式除以 n 就得到平均比较次数 $T(n)$, 因而得到下述递推方程.

$$\begin{cases} T(n)=\dfrac{2}{n}\displaystyle\sum_{i=1}^{n-1}T(i)+O(n),\ n\geqslant 2, \\ T(1)=0. \end{cases}$$

考虑算法 Quicksort 在平均情况下的时间复杂度. 如果首元素处在第 k 个位置, 那么划分后的两个子问题的规模分别为 $k-1$ 和 $n-k$, 其中 $k=1,2,\cdots,n$. 假定这 n 种情况出现的可能性相等, 为计算平均时间复杂度, 只需要对这 n 种情况分别计算各自的比较次数, 然后取平均值即可.

设 Quicksort 算法对于规模为 n 的输入的平均比较次数是 $T(n)$. 若首元素处在第 k 个位置, 划分后的子问题规模分别为 $k-1$ 和 $n-k$, 平均比较次数分别是 $T(k-1)$ 和 $T(n-k)$, 而划分过程 Partition 需要的比较次数是 $O(n)$, 因此总的比较次数为 $T(k-1)+T(n-k)+O(n)$. 对 $k=1,2,\cdots,n$ 求和, 去掉 $T(0)=0$ 的项, 就得到 $2\sum_{k=1}^{n-1}T(k)+O(n^2)$.

把上式除以 n 就得到平均比较次数 $T(n)$, 因而得到下述递推方程.

$$\begin{cases} T(n)=\dfrac{2}{n}\displaystyle\sum_{i=1}^{n-1}T(i)+O(n),\ n\geqslant 2, \\ T(1)=0. \end{cases}$$

从这个方程不难看出, $T(n)$ 依赖于 $T(n-1),T(n-2),\cdots,T(1)$ 等所有的项, 这种递推方程也称作 全部历史递推方程. 求解过程请见例 1.3.3.

例 1.3.3

求解关于快速排序算法平均复杂度 $T(n)$ 的递推方程.

例 1.3.3

求解关于快速排序算法平均复杂度 $T(n)$ 的递推方程.

解

先使用差消法将原方程化简成一阶的方程. 由原方程得到

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

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

其中 c 为某个常数. 将两个方程相减得

$$nT(n) - (n-1)T(n-1) = 2T(n-1) + O(n).$$

化简得

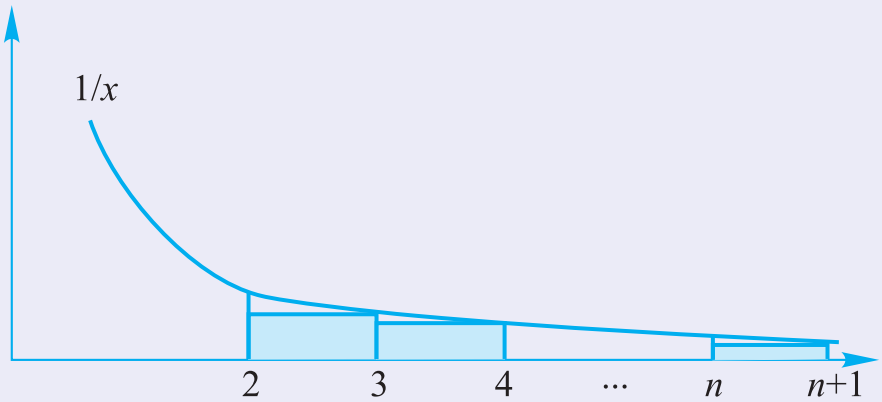
$$nT(n) = (n+1)T(n-1) + O(n).$$

变形并迭代得到

例1.3.3解(续)

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

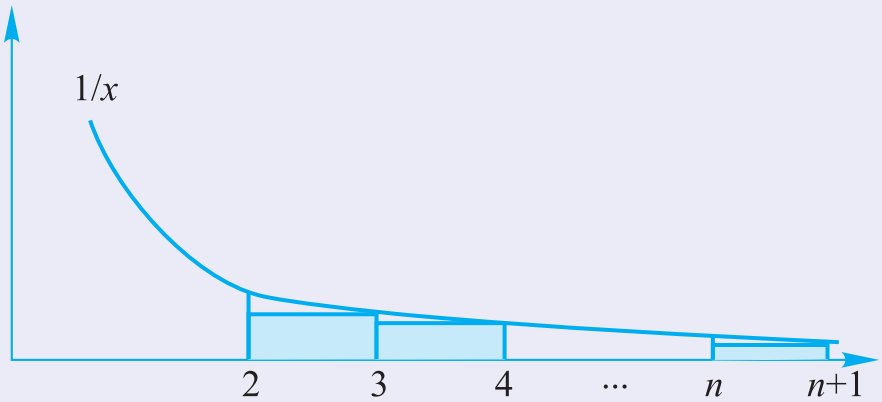
上面公式中的 c 是某个常数,随后的求和使用了积分作为近似结果,请见下图.



例1.3.3解(续)

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

上面公式中的 c 是某个常数,随后的求和使用了积分作为近似结果,请见下图.



根据积分有

$$\begin{aligned}&\frac{1}{n+1} + \frac{1}{n} + \cdots + \frac{1}{3} \\ &\leqslant \int_2^{n+1} \frac{1}{x}\,\mathrm{d}x \\ &= \ln x\Big|_2^{n+2} \\ &= \ln(n+1) - \ln 2 \\ &= O(\log n),\end{aligned}$$

因此得到原递推方程的解

$$T(n) = O(n \log n).$$

用递归树的模型可以说明迭代的思想. 下面以二分归并排序算法的递推方程

$$\begin{cases} W(n) = 2\,W(n/2) + n - 1, \ n = 2^k, \\ W(1) = 0, \end{cases}$$

为例来构造递归树. 递归树是一棵带权的二叉树, 每个结点都有权. 初始的递归树只有一个结点, 它的权标记为 $W(n)$. 然后不断进行迭代, 直到树中不再含有权为函数的结点为止.

用递归树的模型可以说明迭代的思想. 下面以二分归并排序算法的递推方程

$$\begin{cases} W(n) = 2\,W(n/2) + n - 1, \ n = 2^k, \\ W(1) = 0, \end{cases}$$

为例来构造递归树. 递归树是一棵带权的二叉树,每个结点都有权. 初始的递归树只有一个结点,它的权标记为 $W(n)$. 然后不断进行迭代,直到树中不再含有权为函数的结点为止.

迭代规则就是把递归树中权为函数的结点,如 $W(n), W(n/2), W(n/4)$ 等,用和这个函数相等的递推方程右部的子树来代替. 这种子树只有 2 层,树根标记为方程右部除了函数之外的剩余表达式,每一片树叶则代表方程右部的一个函数项. 例如,第一步迭代,树中唯一的结点(第 0 层)$W(n)$ 可以用根是 $n-1$、2 片树叶都是 $W(n/2)$ 的子树来代替. 代替以后递归树由 1 层变成了 2 层. 第二步迭代,应该用根为 $n/2-1$、2 片树叶都是 $W(n/4)$ 的子树来代替树中权为 $W(n/2)$ 的叶结点(第 1 层),代替后递归树就变成了 3 层. 照这样进行下去,每迭代一次,递归树就增加一层,直到树叶都变成初值 1 为止. 整个迭代过程与递归树的生成过程完全对应起来,正如图 1.3.1 所示. 不难看出,在整个迭代过程中,递归树中全部结点的权之和不变,总是等于函数 $W(n)$.

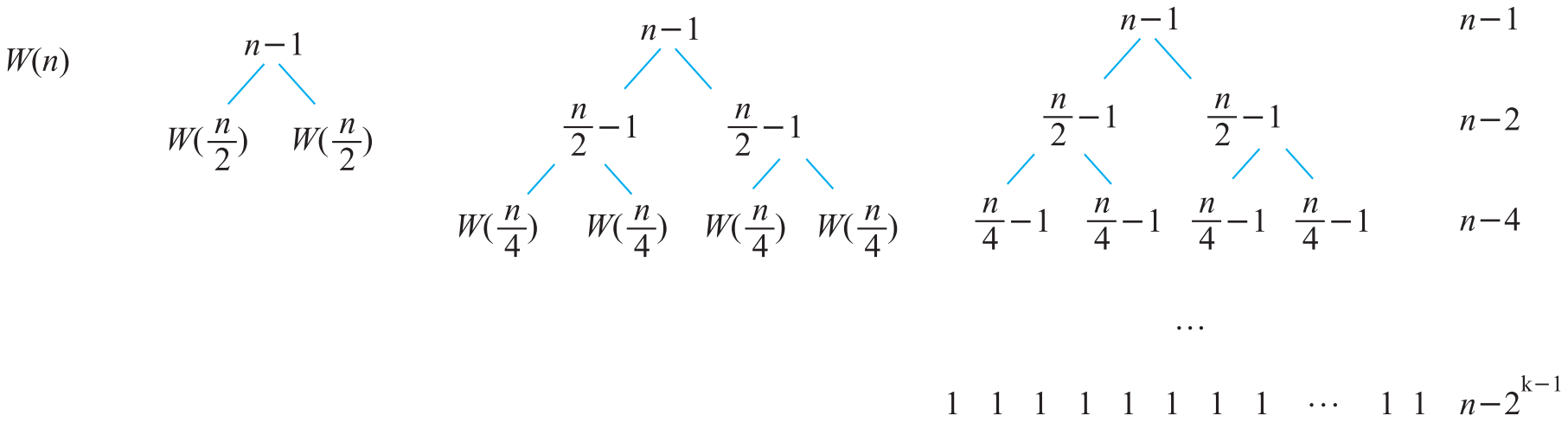

W(n)
n−1
W(n/2) W(n/2)
n−1
n/2−1 n/2−1
W(n/4) W(n/4) W(n/4) W(n/4)
n−1
n/2−1 n/2−1
n/4−1 n/4−1 n/4−1 n/4−1
n−1
n−2
n−4
…
1 1 1 1 1 1 1 1 … 1 1
n−2^(k−1)

图 1.3.1

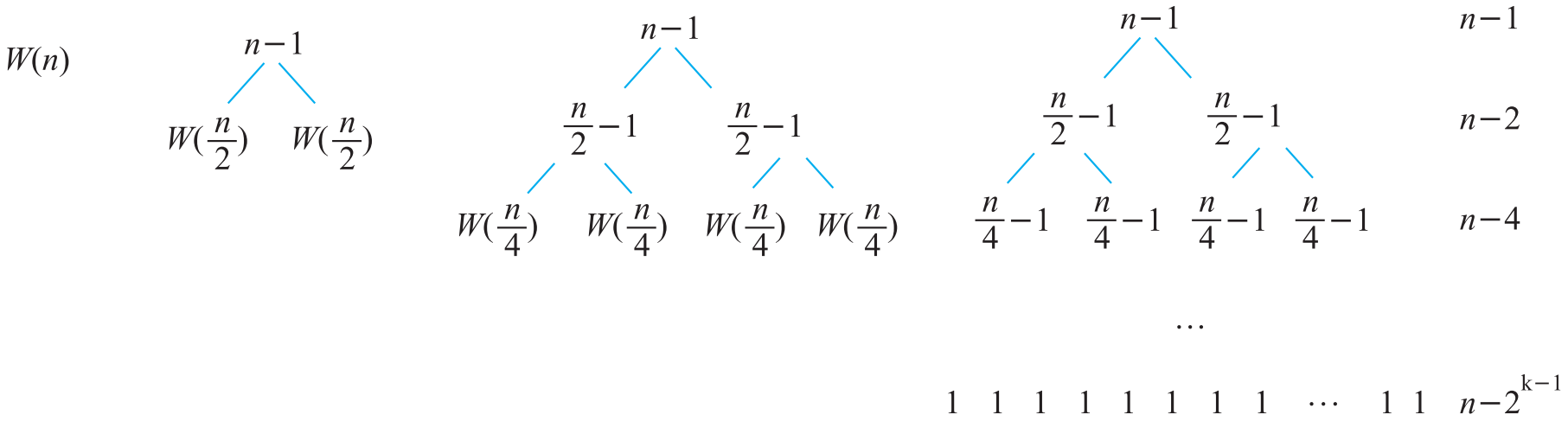


图 1.3.1

为了计算最终的递归树中所有结点的权之和，可以采用分层计算的方法. 递归树有 k 层，各层结点的值之和分别为

$$n-1, \quad n-2, \quad n-4, \quad \cdots, \quad n-2^{k-1}.$$

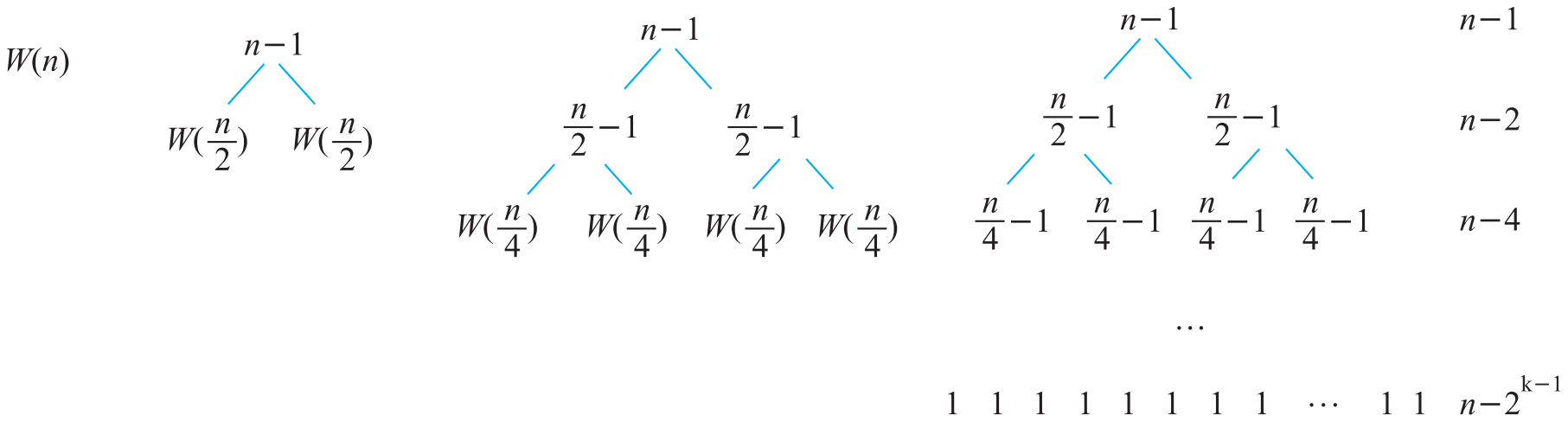


图 1.3.1

为了计算最终的递归树中所有结点的权之和，可以采用分层计算的方法. 递归树有 k 层，各层结点的值之和分别为

$$n-1, \quad n-2, \quad n-4, \quad \cdots, \quad n-2^{k-1}.$$

因此总和为

$$\begin{aligned} nk-(1+2+\cdots+2^{k-1}) &= nk-(2^k-1) \\ &= n\log n - n + 1. \end{aligned}$$

这个结果与前面用换元法计算的结果完全一致.

递归算法是一种常用的算法，它的特点就是在算法中要递归调用自己. 递归算法的分析中经常用到递推方程.

递归算法是一种常用的算法，它的特点就是在算法中要递归调用自己. 递归算法的分析中经常用到递推方程.

分治策略是算法设计中的一种重要技术，它的主要思想是将原问题分解成规模更小的子问题，分别求解每个子问题，然后将子问题的解进行综合，从而得到原问题的解. 设 a, b 为正整数，n 为问题的输入规模，n/b 为子问题的输入规模，a 为子问题个数，$d(n)$ 为将原问题分解成子问题以及将子问题的解综合得到原问题解的代价.

递归算法是一种常用的算法，它的特点就是在算法中要递归调用自己. 递归算法的分析中经常用到递推方程.

分治策略是算法设计中的一种重要技术，它的主要思想是将原问题分解成规模更小的子问题，分别求解每个子问题，然后将子问题的解进行综合，从而得到原问题的解. 设 a, b 为正整数，n 为问题的输入规模，n/b 为子问题的输入规模，a 为子问题个数，$d(n)$ 为将原问题分解成子问题以及将子问题的解综合得到原问题解的代价.

例如，对 n 个正整数进行二分归并排序，那么 $b = 2, a = 2, d(n) = n - 1$. 一般情况下有

$$\begin{cases} T(n) = aT(n/b) + d(n), \ n = b^k, \\ T(1) = 1. \end{cases} \tag{1.3.1}$$

经过迭代得到

$$\begin{aligned} T(n) &= a^2 T(n/b^2) + ad(n/b) + d(n) \\ &= \cdots \\ &= a^k T(n/b^k) + a^{k-1} d(n/b^{k-1}) + a^{k-2} d(n/b^{k-2}) + \cdots + ad(n/b) + d(n) \\ &= a^k + \sum_{i=0}^{k-1} a^i d(n/b^i), \end{aligned}$$

其中

$$a^k = a^{\log_b^n} = n^{\log_b^a}.$$

经过迭代得到

$$\begin{aligned} T(n) &= a^2 T(n/b^2) + ad(n/b) + d(n) \\ &= \cdots \\ &= a^k T(n/b^k) + a^{k-1} d(n/b^{k-1}) + a^{k-2} d(n/b^{k-2}) + \cdots + ad(n/b) + d(n) \\ &= a^k + \sum_{i=0}^{k-1} a^i d(n/b^i), \end{aligned}$$

其中

$$a^k = a^{\log_b^n} = n^{\log_b^a}.$$

当 $d(n) = c$ 时，代入上式得到

$$T(n) = \begin{cases} a^k + c\frac{a^k-1}{a-1} = O(a^k) = O(n^{\log_b a}), & a \neq 1, \\ a^k + kc = O(\log n), & a = 1. \end{cases} \tag{1.3.2}$$

当 $d(n) = cn$ 时，代入上式得到

$$
\begin{aligned}
T(n) &= a^k + \sum_{i=0}^{k-1} a^i \frac{cn}{b^i} \\
&= a^k + cn \sum_{i=0}^{k-1} \left(\frac{a}{b}\right)^i \\
&= \begin{cases}
n^{\log_b a} + cn\frac{(a/b)^k - 1}{a/b-1} = O(n), & a < b, \\
n + cnk = O(n \log n), & a = b, \\
a^k + cn\frac{(a/b)^k - 1}{a/b-1} = a^k + c\frac{a^k - b^k}{a/b-1} = O(n^{\log_b a}), & a > b.
\end{cases}
\end{aligned}
\tag{1.3.3}
$$

这些结果可以直接用于求解递推方程.

当 $d(n) = cn$ 时，代入上式得到

$$
\begin{aligned}
T(n) &= a^k + \sum_{i=0}^{k-1} a^i \frac{cn}{b^i} \\
&= a^k + cn \sum_{i=0}^{k-1} \left(\frac{a}{b}\right)^i \\
&= \begin{cases}
n^{\log_b a} + cn\frac{(a/b)^k - 1}{a/b-1} = O(n), & a < b, \\
n + cnk = O(n \log n), & a = b, \\
a^k + cn\frac{(a/b)^k - 1}{a/b-1} = a^k + c\frac{a^k - b^k}{a/b-1} = O(n^{\log_b a}), & a > b.
\end{cases}
\end{aligned} \tag{1.3.3}
$$

这些结果可以直接用于求解递推方程.

例如，二分归并排序的递推方程是

$$
\begin{cases}
W(n) = 2\,W(n/2) + n - 1, n = 2^k, \\
W(1) = 0,
\end{cases}
$$

其中 $a = 2, b = 2, d(n) = O(n)$，根据上面的结果有 $W(n) = O(n \log n)$.

这些公式在递归算法的分析中经常会用到. 下面给出一个实际应用的例子.

例 1.3.4

设 n 为正整数且恰好是 2 的幂. 下述算法 *Power* 是计算 a^n 的算法.

算法 ***Power***(a, n)

```
1 if n = 1 then return a;
2 else x ← Power(a, n/2);
3 return x * x    //x 与 x 相乘
```

针对这个算法考虑下面的问题.

这些公式在递归算法的分析中经常会用到. 下面给出一个实际应用的例子.

例 1.3.4

设 n 为正整数且恰好是 2 的幂. 下述算法 *Power* 是计算 a^n 的算法.

算法 ***Power***(a, n)

---

1 **if** $n = 1$ **then return** a;
2 **else** $x \leftarrow$ *Power*$(a, n/2)$;
3 **return** $x * x$ $\quad //x$ 与 x 相乘

针对这个算法考虑下面的问题.

1. 设 a 为实数, 如果以两个数的相乘做基本运算, 估计算法 Power 最坏情况下的时间复杂度.
2. 在 Fibonacci 数列 $1, 1, 2, 3, 5, 8$ 等的前面加上一个 0, 得到数列 $\{F_n\}$, 证明
$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n.$$
3. 对于 $n = 2^k$, k 为正整数, 如何利用上述公式和算法 Power 计算 F_n? 把这个算法与直接利用递推公式计算 F_n 的算法进行比较, 哪个效率更高? 为什么?

解

(1) 令 $T(n)$ 表示 Power 算法最坏情况下的计算复杂度，则 $T(n)$ 满足

$$\begin{cases} T(n) = T(n/2) + 1, \\ T(1) = 0. \end{cases}$$

根据公式 (1.3.2) 得到 $T(n) = O(\log n)$.

解

(1) 令 $T(n)$ 表示 Power 算法最坏情况下的计算复杂度，则 $T(n)$ 满足

$$\begin{cases} T(n) = T(n/2) + 1, \\ T(1) = 0. \end{cases}$$

根据公式 (1.3.2) 得到 $T(n) = O(\log n)$.

(2) 对 n 归纳. $n = 1$ 时显然成立. 假设 n 时也成立，则

$$\begin{aligned} \begin{pmatrix} F_{n+2} & F_{n+1} \\ F_{n+1} & F_n \end{pmatrix} &= \begin{pmatrix} F_{n+1} + F_n & F_{n+1} \\ F_n + F_{n-1} & F_n \end{pmatrix} \\ &= \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n+1}. \end{aligned}$$

解

(1) 令 $T(n)$ 表示 Power 算法最坏情况下的计算复杂度,则 $T(n)$ 满足

$$\begin{cases} T(n) = T(n/2) + 1, \\ T(1) = 0. \end{cases}$$

根据公式 (1.3.2) 得到 $T(n) = O(\log n)$.

(2) 对 n 归纳. $n = 1$ 时显然成立. 假设 n 时也成立,则

$$\begin{aligned} \begin{pmatrix} F_{n+2} & F_{n+1} \\ F_{n+1} & F_n \end{pmatrix} &= \begin{pmatrix} F_{n+1} + F_n & F_{n+1} \\ F_n + F_{n-1} & F_n \end{pmatrix} \\ &= \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n+1}. \end{aligned}$$

(3) 根据

$$\begin{pmatrix} F_n & F_{n-1} \\ F_{n-1} & F_{n-2} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-1},$$

只要利用 Power 算法计算出上述 2 阶矩阵的 $n-1$ 次幂,就得到了 F_n. 两个 2 阶矩阵相乘需要 8 次乘法,使用 Power 算法完成整个计算需要 $O(\log n)$ 次乘法. 而按照递归定义直接从初值计算 F_n 需要 $O(n)$ 次加法. 对于比较大的 n,显然 $O(\log n)$ 比起 $O(n)$ 的值要小很多,因此使用 Power 算法效率更高.

11.4 生成函数及其应用

定义 1.4.1

设 r 为实数，n 为整数，引入形式符号

$$\binom{r}{n}=\begin{cases}0, & n<0,\\ 1, & n=0,\\ \dfrac{r(r-1)\cdots(r-n+1)}{n!}, & n>0,\end{cases}$$

称作牛顿二项式系数.

11.4 生成函数及其应用

定义 1.4.1

设 r 为实数，n 为整数，引入形式符号

$$\binom{r}{n}=\begin{cases}0, & n<0,\\ 1, & n=0,\\ \frac{r(r-1)\cdots(r-n+1)}{n!}, & n>0,\end{cases}$$

称作牛顿二项式系数.

例如

$$\binom{4}{3}=\frac{4\cdot 3\cdot 2}{3!}=4,\ \binom{-2}{5}=\frac{(-2)(-3)(-4)(-5)(-6)}{5!}=-6,$$

$$\binom{1/2}{4}=\frac{\frac{1}{2}\left(\frac{1}{2}-1\right)\left(\frac{1}{2}-2\right)\left(\frac{1}{2}-3\right)}{4!}=\frac{1(-1)(-3)(-5)}{2^4 4!}=-\frac{5}{128}.$$

表面上看，这个符号与二项式系数的符号一样，但是在这里它只是一个形式符号，不具有任何组合意义. 当 r 为自然数时，牛顿二项式系数就成为普通的二项式系数，这时才与集合的组合计数联系到一起.

与二项式定理对应，有下面的牛顿二项式定理，它恰好表示了某些函数的幂级数.

定理 1.4.1

设 α 为实数，则对一切实数 $x, y, |x/y| < 1$，有

$$(x+y)^{\alpha} = \sum_{n=0}^{+\infty} \binom{\alpha}{n} x^n y^{\alpha-n}, \text{ 其中 } \binom{\alpha}{n} = \frac{\alpha(\alpha-1)\cdots(\alpha-n+1)}{n!}.$$

与二项式定理对应，有下面的牛顿二项式定理，它恰好表示了某些函数的幂级数.

定理 1.4.1

设 α 为实数，则对一切实数 $x, y, |x/y| < 1$，有

$$(x+y)^{\alpha} = \sum_{n=0}^{+\infty} \binom{\alpha}{n} x^n y^{\alpha-n}, \text{ 其中 } \binom{\alpha}{n} = \frac{\alpha(\alpha-1)\cdots(\alpha-n+1)}{n!}.$$

这个定理的证明可以在一般的数学分析书中找到，这里不再赘述.

与二项式定理对应，有下面的牛顿二项式定理，它恰好表示了某些函数的幂级数.

定理 1.4.1

设 α 为实数，则对一切实数 $x, y, |x/y| < 1$，有

$$(x+y)^\alpha = \sum_{n=0}^{+\infty} \binom{\alpha}{n} x^n y^{\alpha-n}, \text{ 其中 } \binom{\alpha}{n} = \frac{\alpha(\alpha-1)\cdots(\alpha-n+1)}{n!}.$$

这个定理的证明可以在一般的数学分析书中找到，这里不再赘述.

当 $\alpha = m$ 时，这个定理就变成二项式定理(定理 ??)；如果 $\alpha = -m$，其中 m 为正整数，那么

$$\begin{aligned}
\binom{\alpha}{n} &= \binom{-m}{n} \\
&= \frac{(-m)(-m-1)\cdots(-m-n+1)}{n!} \\
&= \frac{(-1)^n m(m+1)\cdots(m+n-1)}{n!} \\
&= (-1)^n \binom{m+n-1}{n}.
\end{aligned}$$

这时令 $x=z, y=1$, 牛顿二项式定理就变成

$$(1+z)^{-m}=\frac{1}{(1+z)^m}=\sum_{n=0}^{+\infty}(-1)^n\binom{m+n-1}{n}z^n,\ |z|<1.$$

这时令 $x=z, y=1$，牛顿二项式定理就变成

$$(1+z)^{-m}=\frac{1}{(1+z)^m}=\sum_{n=0}^{+\infty}(-1)^n\binom{m+n-1}{n}z^n,\ |z|<1.$$

在上面式子中用 $-z$ 代替 z，就得到

$$(1-z)^{-m}=\frac{1}{(1-z)^m}=\sum_{n=0}^{+\infty}\binom{m+n-1}{n}z^n,\ |z|<1.$$

这时令 $x=z, y=1$，牛顿二项式定理就变成

$$(1+z)^{-m}=\frac{1}{(1+z)^m}=\sum_{n=0}^{+\infty}(-1)^n\binom{m+n-1}{n}z^n,\ |z|<1.$$

在上面式子中用 $-z$ 代替 z，就得到

$$(1-z)^{-m}=\frac{1}{(1-z)^m}=\sum_{n=0}^{+\infty}\binom{m+n-1}{n}z^n,\ |z|<1.$$

特别当 $m=1$ 或 2 时有 $\frac{1}{1-x}=1+x+x^2+\cdots$，$\frac{1}{(1-x)^2}=\sum_{n=0}^{+\infty}(n+1)x^n$.

这时令 $x=z, y=1$，牛顿二项式定理就变成

$$(1+z)^{-m}=\frac{1}{(1+z)^m}=\sum_{n=0}^{+\infty}(-1)^n\binom{m+n-1}{n}z^n,\ |z|<1.$$

在上面式子中用 $-z$ 代替 z，就得到

$$(1-z)^{-m}=\frac{1}{(1-z)^m}=\sum_{n=0}^{+\infty}\binom{m+n-1}{n}z^n,\ |z|<1.$$

特别当 $m=1$ 或 2 时有 $\frac{1}{1-x}=1+x+x^2+\cdots$，$\frac{1}{(1-x)^2}=\sum_{n=0}^{+\infty}(n+1)x^n$.
当 $\alpha=1/2$ 时，牛顿二项式定理就变成

$$\begin{aligned}(1+x)^{\frac{1}{2}}&=\sum_{n=0}^{+\infty}\binom{\frac{1}{2}}{n}x^n=1+\sum_{n=1}^{+\infty}\frac{\frac{1}{2}\left(\frac{1}{2}-1\right)\cdots\left(\frac{1}{2}-n+1\right)}{n!}x^n\\&=1+\sum_{n=1}^{+\infty}\frac{(-1)^{n-1}1\cdot3\cdot5\cdot\cdots\cdot(2n-3)}{2^n n!}x^n\\&=1+\sum_{n=1}^{+\infty}\frac{(-1)^{n-1}(2n-2)!}{2^n n!\cdot 2^{n-1}(n-1)!}x^n=1+\sum_{n=1}^{+\infty}\frac{(-1)^{n-1}}{2^{2n-1}n}\binom{2n-2}{n-1}x^n.\end{aligned}$$

生成函数

定义 1.4.2

给定序列 $\{a_n\}$, 构造形式幂级数

$$G(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n + \cdots ,$$

称 $G(x)$ 为序列 $\{a_n\}$ 的 生成函数 .

生成函数

定义 1.4.2

给定序列 $\{a_n\}$，构造形式幂级数

$$G(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n + \cdots,$$

称 $G(x)$ 为序列 $\{a_n\}$ 的 生成函数.

例如，$\{C(m,n)\}$ 的生成函数为 $(1+x)^m$.
给定正整数 k，$\{k^n\}$ 的生成函数为

$$G(x) = 1 + kx + k^2x^2 + k^3x^3 + \cdots = \frac{1}{1-kx}.$$

生成函数与序列是一一对应的，我们经常使用生成函数作为工具来求解序列的通项公式，而这些项恰好代表了某个组合计数问题的解.

生成函数

定义 1.4.2

给定序列 $\{a_n\}$，构造形式幂级数

$$G(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n + \cdots,$$

称 $G(x)$ 为序列 $\{a_n\}$ 的 生成函数.

例如，$\{C(m,n)\}$ 的生成函数为 $(1+x)^m$.
给定正整数 k，$\{k^n\}$ 的生成函数为

$$G(x) = 1 + kx + k^2x^2 + k^3x^3 + \cdots = \frac{1}{1-kx}.$$

生成函数与序列是一一对应的，我们经常使用生成函数作为工具来求解序列的通项公式，而这些项恰好代表了某个组合计数问题的解.
给定序列 $\{a_n\}$ 或关于 a_n 的递推方程，如何求它的生成函数 $G(x)$ 呢？反之，给定生成函数 $G(x)$，如何求对应序列的通项公式 a_n 呢？下面给出两个例子.

例 1.4.1

求序列 $\{a_n\}$ 的生成函数.

1. $a_n = 7 \cdot 3^n$.
2. $a_n = (-1)^n(n+1)$.

例 1.4.1

求序列 $\{a_n\}$ 的生成函数.

1. $a_n = 7 \cdot 3^n$.
2. $a_n = (-1)^n(n+1)$.

解

(1) 根据生成函数定义, 有

$$G(x) = 7\sum_{n=0}^{+\infty} 3^n x^n = 7\sum_{n=0}^{+\infty}(3x)^n$$
$$= \frac{7}{1-3x}.$$

例 1.4.1

求序列 $\{a_n\}$ 的生成函数.

1. $a_n = 7 \cdot 3^n$.
2. $a_n = (-1)^n(n+1)$.

解

(1) 根据生成函数定义, 有

$$G(x) = 7\sum_{n=0}^{+\infty} 3^n x^n = 7\sum_{n=0}^{+\infty}(3x)^n = \frac{7}{1-3x}.$$

(2) 令 $G(x) = \sum_{n=0}^{+\infty}(-1)^n(n+1)x^n$, 对 $G(x)$ 积分得

$$\int_0^x G(x)\,\mathrm{d}x = \int_0^x \sum_{n=0}^{+\infty}(-1)^n(n+1)x^n\,\mathrm{d}x$$
$$= \sum_{n=0}^{+\infty}(-1)^n \int_0^x (n+1)x^n\,\mathrm{d}x$$
$$= \sum_{n=0}^{+\infty}(-1)^n x^{n+1}$$
$$= x\sum_{n=0}^{+\infty}(-x)^n = \frac{x}{1+x}.$$

对两边求导得到

$$G(x) = \left(\frac{x}{1+x}\right)' = \frac{1}{1+x} - \frac{x}{(1+x)^2}$$
$$= \frac{1+x-x}{(1+x)^2} = \frac{1}{(1+x)^2}.$$

给定序列 $\{a_n\}$ 的生成函数，求 a_n. 基本方法就是利用部分分式的待定系数法将原来的函数化成基本生成函数的表达式之和，然后利用这些基本生成函数的展开式求出 a_n.

给定序列 $\{a_n\}$ 的生成函数，求 a_n. 基本方法就是利用部分分式的待定系数法将原来的函数化成基本生成函数的表达式之和，然后利用这些基本生成函数的展开式求出 a_n.

例 1.4.2

已知 $\{a_n\}$ 的生成函数为 $G(x) = \frac{2+3x-6x^2}{1-2x}$，求 a_n.

给定序列 $\{a_n\}$ 的生成函数，求 a_n. 基本方法就是利用部分分式的待定系数法将原来的函数化成基本生成函数的表达式之和，然后利用这些基本生成函数的展开式求出 a_n.

例 1.4.2

已知 $\{a_n\}$ 的生成函数为 $G(x)=\frac{2+3x-6x^2}{1-2x}$，求 a_n.

解

$$\begin{aligned} G(x) &= \frac{2+3x-6x^2}{1-2x} \\ &= \frac{2}{1-2x}+3x \\ &= 2\sum_{n=0}^{+\infty}(2x)^n+3x \\ &= \sum_{n=0}^{+\infty}2^{n+1}x^n+3x. \end{aligned}$$

因此 $a_n=\begin{cases} 2^{n+1}, & n\neq 1, \\ 2^2+3=7, & n=1. \end{cases}$

卡塔兰数

可以用生成函数求解递推方程，特别是某些不适合使用公式法和迭代归纳法的方程，下面关于卡塔兰数 h_n 的递推方程就是一个例子.

例 1.4.3

求解递推方程

$$\begin{cases} h_n = \sum_{k=1}^{n-1} h_k h_{n-k}, \ n \geqslant 2, \\ h_1 = 1. \end{cases}$$

卡塔兰数

可以用生成函数求解递推方程，特别是某些不适合使用公式法和迭代归纳法的方程，下面关于卡塔兰数 h_n 的递推方程就是一个例子.

例 1.4.3

求解递推方程

$$\begin{cases} h_n = \sum_{k=1}^{n-1} h_k h_{n-k}, \ n \geqslant 2, \\ h_1 = 1. \end{cases}$$

解

设 $\{h_n\}$ 的生成函数为 $H(x) = \sum_{n=1}^{+\infty} h_n x^n$，两边平方得

$$H^2(x) = \sum_{k=1}^{+\infty} h_k x^k \cdot \sum_{l=1}^{+\infty} h_l x^l = \sum_{k=1}^{+\infty} \sum_{l=1}^{+\infty} h_k h_l x^{k+l}$$

$$= \sum_{n=2}^{+\infty} x^n \sum_{k=1}^{n-1} h_k h_{n-k} = \sum_{n=2}^{+\infty} h_n x^n = H(x) - h_1 x = H(x) - x.$$

这是一个关于 $H(x)$ 的一元二次方程，利用求根公式得到

$$H_1(x)=\frac{1+(1-4x)^{\frac{1}{2}}}{2},\ H_2(x)=\frac{1-(1-4x)^{\frac{1}{2}}}{2}.$$

由于 $H(0)=0$，因此取 $H(x)=H_2(x)$. 将 $H(x)$ 展开得

$$\begin{aligned}
H(x)&=\frac{1-(1-4x)^{\frac{1}{2}}}{2}\\
&=\frac{1}{2}-\frac{1}{2}(1-4x)^{\frac{1}{2}}\\
&=\frac{1}{2}-\frac{1}{2}\left[1+\sum_{n=1}^{+\infty}\frac{(-1)^{n-1}}{n2^{2n-1}}\binom{2n-2}{n-1}(-4x)^n\right]\\
&=\sum_{n=1}^{+\infty}\frac{(-1)^n}{n2^{2n}}\binom{2n-2}{n-1}(-1)^n2^{2n}x^n\\
&=\sum_{n=1}^{+\infty}\frac{1}{n}\binom{2n-2}{n-1}x^n,
\end{aligned}$$

因此 $h_n=\frac{1}{n}\binom{2n-2}{n-1}$.

栈输出计数

回顾例 ?? 关于栈输出结果的计数实例，通过使用非降路径的模型，得到 n 个元素的栈的不同输出的个数是 $\frac{1}{n+1}\binom{2n}{n}$，这个数恰好是第 $n+1$ 个卡塔兰数.
下面使用生成函数的方法求解这个问题.

栈输出计数

回顾例 ?? 关于栈输出结果的计数实例，通过使用非降路径的模型，得到 n 个元素的栈的不同输出的个数是 $\frac{1}{n+1}\binom{2n}{n}$，这个数恰好是第 $n+1$ 个卡塔兰数.

下面使用生成函数的方法求解这个问题.

考虑字符序列 $1, 2, \cdots, n$. 当某个字符 X 进栈时，在 X 前面记录一个左括号“(”；当 X 出栈时在 X 后面记录一个右括号“)”. 在这两个括号之间，除 X 之外的其他字符就是在 X 之后进栈并且在 X 之前出栈的字符.

例如，$(1(2(3))(4))$ 表示的过程是：

1 进栈，2 进栈，3 进栈，3 出栈，2 出栈，4 进栈，4 出栈，1 出栈.

栈输出计数

回顾例 ?? 关于栈输出结果的计数实例，通过使用非降路径的模型，得到 n 个元素的栈的不同输出的个数是 $\frac{1}{n+1}\binom{2n}{n}$，这个数恰好是第 $n+1$ 个卡塔兰数.

下面使用生成函数的方法求解这个问题.

考虑字符序列 $1, 2, \cdots, n$. 当某个字符 X 进栈时，在 X 前面记录一个左括号“(”；当 X 出栈时在 X 后面记录一个右括号“)”. 在这两个括号之间，除 X 之外的其他字符就是在 X 之后进栈并且在 X 之前出栈的字符.

例如，$(1(2(3))(4))$ 表示的过程是：

$$1 \text{ 进栈}, 2 \text{ 进栈}, 3 \text{ 进栈}, 3 \text{ 出栈}, 2 \text{ 出栈}, 4 \text{ 进栈}, 4 \text{ 出栈}, 1 \text{ 出栈}.$$

按照上述对应规则，栈的任何一种输出都对应了 n 个字符的进栈、出栈的一种操作序列，而这个操作序列又对应了 n 对括号的合理配对的方法数. 显然，在 n 次进栈、n 次出栈的操作序列中，从开始到中间的任何位置，进栈次数不可能少于出栈次数. 这就意味着在括号配对的序列中，从左边算起到序列的任何位置，左括号的数目都不少于右括号的数目.

设 n 对括号的配对方法数是 $T(n)$，考虑与最左边的左括号配对的右括号的位置，在这对括号中间有 k 对其他括号，这 k 对括号有 $T(k)$ 种配对方法；而在这对括号的后面有 $n-1-k$ 对括号，其配对方法数是 $T(n-1-k)$. 因此，对于给定的 k，构成输出序列的方法数是 $T(k)T(n-1-k)$. 由于 k 可能的取值是 $0,1,2,\cdots,n-1$. 根据加法法则，可以得到递推方程

$$\begin{cases} T(n)=\sum\limits_{k=0}^{n-1}T(k)T(n-1-k), \\ T(0)=1. \end{cases}$$

设 n 对括号的配对方法数是 $T(n)$，考虑与最左边的左括号配对的右括号的位置，在这对括号中间有 k 对其他括号，这 k 对括号有 $T(k)$ 种配对方法；而在这对括号的后面有 $n-1-k$ 对括号，其配对方法数是 $T(n-1-k)$. 因此，对于给定的 k，构成输出序列的方法数是 $T(k)T(n-1-k)$. 由于 k 可能的取值是 $0,1,2,\cdots,n-1$. 根据加法法则，可以得到递推方程

$$\begin{cases} T(n)=\sum\limits_{k=0}^{n-1}T(k)T(n-1-k), \\ T(0)=1. \end{cases}$$

设序列 $\{T(n)\}$ 的生成函数是 $T(x)$，那么有 $T(x)=\sum_{n=0}^{+\infty}T(n)x^n$，从而得到

$$\begin{aligned} T^2(x) &= \left[\sum_{k=0}^{+\infty}T(k)x^k\right]\left[\sum_{l=0}^{+\infty}T(l)x^l\right] \\ &= \sum_{n=1}^{+\infty}x^{n-1}\left[\sum_{k=0}^{n-1}T(k)T(n-1-k)\right] \\ &= \sum_{n=1}^{+\infty}T(n)x^{n-1}=\frac{T(x)-1}{x}. \end{aligned}$$

求解关于 $T(x)$ 的一元二次方程，得到 $2xT(x) = 1 \pm \sqrt{1-4x}$. 由于 $x \to 0$ 时，上式的左边趋于 0，因此取根为 $T(x) = \frac{1-\sqrt{1-4x}}{2x}$，展开成幂级数得

$$T(x) = \sum_{n=0}^{+\infty} \frac{1}{n+1} \binom{2n}{n} x^n.$$

因此，不同输出的个数就是右边展开式中 x^n 的系数，即 $\frac{1}{n+1}\binom{2n}{n}$.

求解关于 $T(x)$ 的一元二次方程，得到 $2xT(x) = 1 \pm \sqrt{1-4x}$. 由于 $x \to 0$ 时，上式的左边趋于 0，因此取根为 $T(x) = \frac{1-\sqrt{1-4x}}{2x}$，展开成幂级数得

$$T(x) = \sum_{n=0}^{+\infty} \frac{1}{n+1} \binom{2n}{n} x^n.$$

因此，不同输出的个数就是右边展开式中 x^n 的系数，即 $\frac{1}{n+1}\binom{2n}{n}$.

注 1.4.1

通过这个例子可以看到，可以使用生成函数来求解关于 $\{a_n\}$ 的递推方程. 主要步骤是：

1. 先设定序列 $\{a_n\}$ 的生成函数 $G(x)$.
2. 利用递推方程的依赖关系导出关于生成函数 $G(x)$ 的方程(可以是一次方程、二次方程、二元一次方程组、微分方程等不同的形式).
3. 通过求解方程得到 $G(x)$ 的函数表达式.
4. 将 $G(x)$ 展开成幂级数，其中 x^n 项的系数就是 a_n.

用生成函数计算多重集的 r 组合数

设 $S=\{n_1\cdot a_1, n_2\cdot a_2, \cdots, n_k\cdot a_k\}$ 是多重集，S 的一个 r 组合就是一个子多重集 $\{x_1\cdot a_1, x_2\cdot a_2, \cdots, x_k\cdot a_k\}$，其中 x_i 表示在这个 r 组合中含有元素 a_i 的个数. 因此 $0\leqslant x_i\leqslant n_i, i=1,2,\cdots,k$，并且所有的 x_i 之和应该等于 r. 于是得到不定方程

$$x_1+x_2+\cdots+x_k=r,\ 0\leqslant x_i\leqslant n_i, i=1,2,\cdots,k.$$

这就建立了 S 的 r 组合与上述不定方程的解之间的一一对应关系.

用生成函数计算多重集的 r 组合数

设 $S=\{n_1\cdot a_1, n_2\cdot a_2, \cdots, n_k\cdot a_k\}$ 是多重集，S 的一个 r 组合就是一个子多重集 $\{x_1\cdot a_1, x_2\cdot a_2, \cdots, x_k\cdot a_k\}$，其中 x_i 表示在这个 r 组合中含有元素 a_i 的个数. 因此 $0\leqslant x_i\leqslant n_i, i=1,2,\cdots,k$，并且所有的 x_i 之和应该等于 r. 于是得到不定方程

$$x_1+x_2+\cdots+x_k=r,\ 0\leqslant x_i\leqslant n_i, i=1,2,\cdots,k.$$

这就建立了 S 的 r 组合与上述不定方程的解之间的一一对应关系.

考虑函数

$$G(y)=(1+y+\cdots+y^{n_1})(1+y+\cdots+y^{n_2})\cdots(1+y+\cdots+y^{n_k}),$$

右边展开式中的每个项，恰好由 k 个因式 y^{x_i} 相乘构成，因此具有下述形式：$y^{x_1+x_2+\cdots+x_k}$. 展开式中 y^r 的系数，恰好是上述不定方程的解的个数，也就是多重集 S 的 r 组合数.

例 1.4.4

求 $S = \{3 \cdot a, 4 \cdot b, 5 \cdot c\}$ 的 *10* 组合数 N.

例 1.4.4

求 $S = \{3 \cdot a, 4 \cdot b, 5 \cdot c\}$ 的 *10* 组合数 N.

解

生成函数

$$\begin{aligned} G(y) &= (1 + y + y^2 + y^3)(1 + y + y^2 + y^3 + y^4)(1 + y + y^2 + y^3 + y^4 + y^5) \\ &= (1 + 2y + 3y^2 + 4y^3 + 4y^4 + 3y^5 + 2y^6 + y^7)(1 + y + y^2 + y^3 + y^4 + y^5) \\ &= 1 + \cdots + 3y^{10} + 2y^{10} + y^{10} + \cdots , \end{aligned}$$

其中 y^{10} 的系数是 6, 因此 $N = 6$. □

用生成函数计算不定方程解的个数

考虑不定方程

$$x_1 + x_2 + \cdots + x_k = r,\ x_i \in \mathbb{N}.$$

根据定理 ??,解的个数是 $C(k+r-1, r)$,下面通过生成函数的方法求解这个问题. 类似于上面的分析,生成函数为

用生成函数计算不定方程解的个数

考虑不定方程

$$x_1 + x_2 + \cdots + x_k = r,\ x_i \in \mathbb{N}.$$

根据定理 ??,解的个数是 $C(k+r-1,r)$,下面通过生成函数的方法求解这个问题. 类似于上面的分析,生成函数为

$$\begin{aligned} G(y) &= (1 + y + \cdots)^k = \frac{1}{(1-y)^k} \\ &= \sum_{r=0}^{+\infty} \frac{(-k)(-k-1)\cdots(-k-r+1)}{r!}(-y)^r \\ &= \sum_{r=0}^{+\infty} \frac{(-1)^r k(k+1)\cdots(k+r-1)}{r!}(-1)^r y^r \\ &= \sum_{r=0}^{+\infty} \binom{k+r-1}{r} y^r, \end{aligned}$$

其中 y^r 的系数是 $N = C(k+r-1,r)$.

用生成函数计算不定方程解的个数(续)

进一步,考虑对变量取值存在限制情况下的不定方程

$$x_1 + x_2 + \cdots + x_k = r,\ l_i \leqslant x_i \leqslant t_i.$$

这时没有一般的公式,生成函数是

$$G(y) = (y^{l_1} + y^{l_1+1} + \cdots + y^{t_1})(y^{l_2} + y^{l_2+1} + \cdots + y^{t_2}) \cdots (y^{l_k} + y^{l_k+1} + \cdots + y^{t_k}),$$

$G(y)$ 的展开式中 y^r 的系数就是不定方程的解的个数.

用生成函数计算不定方程解的个数(续)

进一步,考虑对变量取值存在限制情况下的不定方程

$$x_1 + x_2 + \cdots + x_k = r,\ l_i \leqslant x_i \leqslant t_i.$$

这时没有一般的公式,生成函数是

$$G(y) = (y^{l_1} + y^{l_1+1} + \cdots + y^{t_1})(y^{l_2} + y^{l_2+1} + \cdots + y^{t_2}) \cdots (y^{l_k} + y^{l_k+1} + \cdots + y^{t_k}),$$

$G(y)$ 的展开式中 y^r 的系数就是不定方程的解的个数.

对于某些不定方程,变量的系数不全是 1,而是其他正整数作为系数,即

$$p_1x_1 + p_2x_2 + \cdots + p_kx_k = r,\ x_i \in \mathbb{N},\ p_1, p_2, \cdots, p_k \text{ 为正整数.}$$

那么也可以使用生成函数的方法求解,对应的生成函数是

$$G(y) = (1 + y^{p_1} + y^{2p_1} + \cdots)(1 + y^{p_2} + y^{2p_2} + \cdots) \cdots (1 + y^{p_k} + y^{2p_k} + \cdots),$$

$G(y)$ 的展开式中 y^r 的系数就是这个不定方程的解的个数.

例 1.4.5

设 n 为自然数，求平面上由直线 $x+2y=n$ 与两个坐标轴所围成的直角三角形内(包括边上)的整点个数，其中整点表示横、纵坐标都是整数的点.

例 1.4.5

设 n 为自然数，求平面上由直线 $x+2y=n$ 与两个坐标轴所围成的直角三角形内(包括边上)的整点个数，其中整点表示横、纵坐标都是整数的点.

解

对于 $r=0,1,\cdots,n$，直线 $x+2y=r$ 上的整点个数就是不定方程 $x+2y=r$ 的非负整数解的个数 a_r. 设关于 $\{a_r\}$ 的生成函数为 $A(z)$，则

$$\begin{aligned} A(z) &= \frac{1}{(1-z)(1-z^2)} \\ &= \frac{1}{4}\cdot\frac{1}{1+z}+\left(-\frac{z}{4}+\frac{3}{4}\right)\cdot\frac{1}{(1-z)^2} \\ &= \frac{1}{4}\sum_{r=0}^{+\infty}(-1)^r z^r-\frac{z}{4}\sum_{r=0}^{+\infty}(1+r)z^r+\frac{3}{4}\sum_{r=0}^{+\infty}(1+r)z^r. \end{aligned}$$

于是求得

$$a_r=\frac{r}{2}+\frac{3}{4}+\frac{1}{4}(-1)^r.$$

例1.4.5解(续)

对 r 求和就得到三角形中的全部整点个数:

$$
\begin{aligned}
N &= \sum_{r=0}^{n} a_r \\
&= \sum_{r=0}^{n} \left[\frac{r}{2} + \frac{3}{4} + \frac{1}{4}(-1)^r \right] \\
&= \frac{1}{4}(n+1)(n+3) + \frac{1}{8}[1+(-1)^n] \\
&= \begin{cases} \frac{1}{4}(n+2)^2, & n \text{ 为偶数}, \\ \frac{1}{4}(n+1)(n+3), & n \text{ 为奇数}. \end{cases}
\end{aligned}
$$

用生成函数计算正整数拆分的方案数

使用生成函数还可以求解正整数拆分的计数问题. 这也是一个常用的组合计数模型. 所谓正整数拆分, 就是将给定正整数 N 表示成若干个正整数之和.
根据拆分后的组成部分是否允许重复、是否有序, 可以将拆分问题划分成 4 类.
表 1.4.1 给出了正整数 3 的对应于不同分类的拆分方案.

用生成函数计算正整数拆分的方案数

使用生成函数还可以求解 正整数拆分 的计数问题. 这也是一个常用的组合计数模型. 所谓正整数拆分, 就是将给定正整数 N 表示成若干个正整数之和.
根据拆分后的组成部分是否允许重复、是否有序, 可以将拆分问题划分成 4 类.
表 1.4.1 给出了正整数 3 的对应于不同分类的拆分方案.

表 1.4.1

	有序	无序
不重复	$3=3$ $3=1+2$ $3=2+1$	$3=3$ $3=1+2$
重复	$3=3$ $3=1+2$ $3=2+1$ $3=1+1+1$	$3=3$ $3=1+2$ $3=1+1+1$

无序拆分的方案数

下面考虑拆分问题的计数，首先考虑无序拆分.

设 N 是给定正整数，将 N 无序拆分成正整数 $a_1, a_2, \cdots, a_n$，则有等式

$$a_1 x_1 + a_2 x_2 + \cdots + a_n x_n = N.$$

这个问题可以归结为不定方程的解的计数问题.

无序拆分的方案数

下面考虑拆分问题的计数，首先考虑无序拆分.

设 N 是给定正整数，将 N 无序拆分成正整数 $a_1, a_2, \cdots, a_n$，则有等式

$$a_1 x_1 + a_2 x_2 + \cdots + a_n x_n = N.$$

这个问题可以归结为不定方程的解的计数问题.

如果拆分后的部分不允许重复，那么对应的生成函数是

$$G(y) = (1 + y^{a_1})(1 + y^{a_2}) \cdots (1 + y^{a_n}).$$

无序拆分的方案数

下面考虑拆分问题的计数，首先考虑无序拆分.

设 N 是给定正整数，将 N 无序拆分成正整数 $a_1, a_2, \cdots, a_n$，则有等式

$$a_1 x_1 + a_2 x_2 + \cdots + a_n x_n = N.$$

这个问题可以归结为不定方程的解的计数问题.

如果拆分后的部分不允许重复，那么对应的生成函数是

$$G(y) = (1 + y^{a_1})(1 + y^{a_2}) \cdots (1 + y^{a_n}).$$

如果允许重复，对应的生成函数是

$$\begin{aligned} G(y) &= (1 + y^{a_1} + y^{2a_1} + \cdots)(1 + y^{a_2} + y^{2a_2} + \cdots) \cdots (1 + y^{a_n} + y^{2a_n} + \cdots) \\ &= \frac{1}{(1 - y^{a_1})(1 - y^{a_2}) \cdots (1 - y^{a_n})}. \end{aligned}$$

例 1.4.6

证明任何正整数都可以唯一地表示成二进制数.

例 1.4.6

证明任何正整数都可以唯一地表示成二进制数.

证明.

设正整数为 N,不难看出,将 N 拆分成 2 的幂 $2^0, 2^1, 2^2, 2^3, \cdots$,且不允许重复的方案,恰好与 N 表示成一个二进制数的方法对应. 因此,N 的二进制表示法的个数与上述拆分方案数相等. 根据前面的分析,拆分方案数的生成函数是

$$G(y) = (1+y)(1+y^2)(1+y^4)(1+y^8)\cdots$$

展开为

$$\begin{aligned} G(y) &= \frac{1-y^2}{1-y}\frac{1-y^4}{1-y^2}\frac{1-y^8}{1-y^4}\cdots \\ &= \frac{1}{1-y} = \sum_{n=0}^{+\infty} y^n. \end{aligned}$$

例 1.4.6

证明任何正整数都可以唯一地表示成二进制数.

证明.

设正整数为 N,不难看出,将 N 拆分成 2 的幂 $2^0, 2^1, 2^2, 2^3, \cdots$,且不允许重复的方案,恰好与 N 表示成一个二进制数的方法对应. 因此,N 的二进制表示法的个数与上述拆分方案数相等. 根据前面的分析,拆分方案数的生成函数是

$$G(y) = (1+y)(1+y^2)(1+y^4)(1+y^8)\cdots$$

展开为

$$\begin{aligned} G(y) &= \frac{1-y^2}{1-y}\frac{1-y^4}{1-y^2}\frac{1-y^8}{1-y^4}\cdots \\ &= \frac{1}{1-y} = \sum_{n=0}^{+\infty} y^n. \end{aligned}$$

在上述幂级数中,由于每项的系数都是 1,因此对于所有的 n, $a_n = 1$,这就证明了正整数 N 只能表示成唯一的二进制数. □

如果对正整数被拆分后的部分的大小加以限制，那么可以使用生成函数计算拆分的方案数. 若对拆分部分的数目加以限制，则不能直接写出相应的生成函数，但是可以使用组合对应的方法来解决这类问题，见下例.

如果对正整数被拆分后的部分的大小加以限制，那么可以使用生成函数计算拆分的方案数. 若对拆分部分的数目加以限制，则不能直接写出相应的生成函数，但是可以使用组合对应的方法来解决这类问题，见下例.

例 1.4.7

给定 r，求将正整数 N 无序并允许重复地拆分成 k 个部分的方法数，这里 $k \leqslant r$.

如果对正整数被拆分后的部分的大小加以限制，那么可以使用生成函数计算拆分的方案数. 若对拆分部分的数目加以限制，则不能直接写出相应的生成函数，但是可以使用组合对应的方法来解决这类问题，见下例.

例 1.4.7

给定 r，求将正整数 N 无序并允许重复地拆分成 k 个部分的方法数，这里 $k \leqslant r$.

解

考虑任意一个将 N 无序并允许重复地拆分成 $k(\leqslant r)$ 个部分的方案，可以用一个图来表示这个方案. 首先将正整数被拆分后的部分按照从大到小的顺序排列. 例如，对于下述拆分方案 $16 = 6 + 5 + 3 + 2, k \leqslant 4$, 4 个部分的排列顺序是：$6, 5, 3, 2$. 如图 1.4.1(a) 所示，拆分后的每个数从左到右分别用一列点来表示，即第 1 列 6 个点，第 2 列 5 个点，第 3 列 3 个点，第 4 列 2 个点. 这种图称作 Ferrers 图. 由于数是从大到小排列的，因此左边的列上的点数不少于右边的列上的点数.

如果对正整数被拆分后的部分的大小加以限制，那么可以使用生成函数计算拆分的方案数. 若对拆分部分的数目加以限制，则不能直接写出相应的生成函数，但是可以使用组合对应的方法来解决这类问题，见下例.

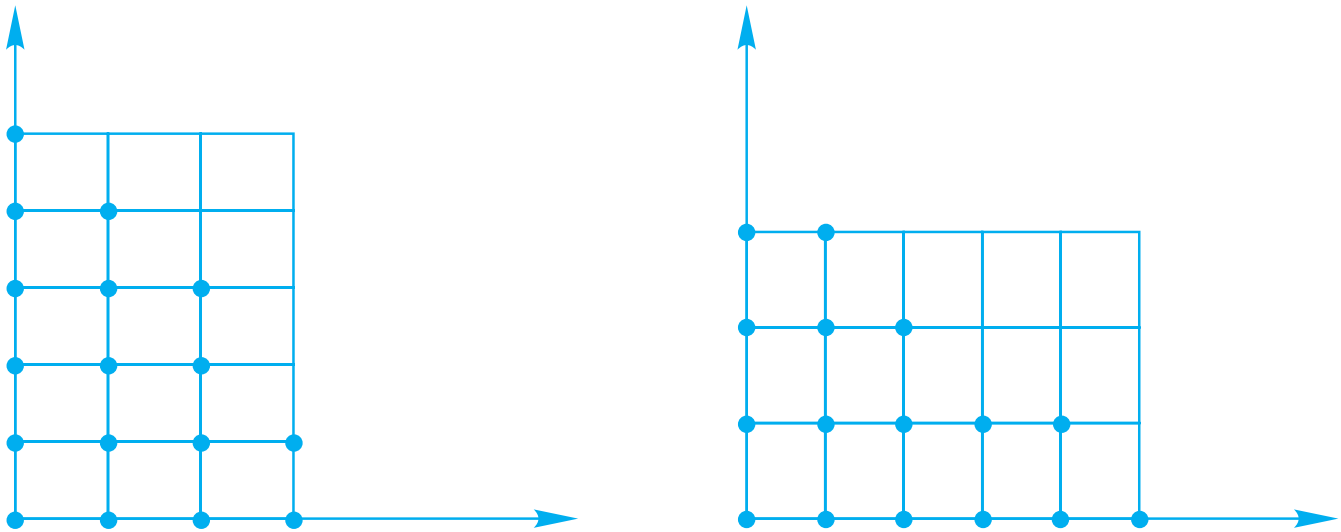
例 1.4.7

给定 r，求将正整数 N 无序并允许重复地拆分成 k 个部分的方法数，这里 $k \leqslant r$.

解

考虑任意一个将 N 无序并允许重复地拆分成 $k(\leqslant r)$ 个部分的方案，可以用一个图来表示这个方案. 首先将正整数被拆分后的部分按照从大到小的顺序排列. 例如，对于下述拆分方案 $16 = 6 + 5 + 3 + 2, k \leqslant 4$, 4 个部分的排列顺序是：$6, 5, 3, 2$. 如图 1.4.1(a) 所示，拆分后的每个数从左到右分别用一列点来表示，即第 1 列 6 个点，第 2 列 5 个点，第 3 列 3 个点，第 4 列 2 个点. 这种图称作 Ferrers 图. 由于数是从大到小排列的，因此左边的列上的点数不少于右边的列上的点数.

将 Ferrers 图看作一个直角坐标系，然后将它围绕 $y = x$ 的直线翻转 $180°$，就得到另一个共轭的 Ferrers 图，如图 1.4.1(b) 所示. 这个图恰好对应了拆分后每个



(a) (b)

图 1.4.1

例1.4.7解(续)

部分都不超过 4 的一种方案,即 $16 = 4 + 4 + 3 + 2 + 2 + 1$.

因此,问题就转变为:求将 N 无序并允许重复地拆分成不超过 r 的数的方案数.

对应的生成函数是

$$G(y) = \frac{1}{(1-y)(1-y^2)\cdots(1-y^r)}.$$

$G(y)$ 的展开式中 y^N 的系数就是所需要的结果. □

例 1.4.8

设将 n 无序拆分成恰好 r 个部分的方案有 $p_r(n)$ 种，证明对于给定的 r 有

$$p_r(n) \sim \frac{n^{r-1}}{r!(r-1)!}.$$

例 1.4.8

设将 n 无序拆分成恰好 r 个部分的方案有 $p_r(n)$ 种，证明对于给定的 r 有

$$p_r(n) \sim \frac{n^{r-1}}{r!(r-1)!}.$$

证明.

考虑 n 的一种无序拆分方案 $n = x_1 + x_2 + \cdots + x_r$，其中 $x_1 \geqslant x_2 \geqslant \cdots \geqslant x_r \geqslant 1$，把 $x_1, x_2, \cdots, x_r$ 做排列就可以得到方程 $x_1 + x_2 + \cdots + x_r = n$ 的一组正整数解. 由于 $x_1, x_2, \cdots, x_r$ 中的数可能存在重复，因此排列数至多是 $r!$，从而得到

$$r!\, p_r(n) \geqslant \binom{n-1}{r-1}.$$

另一方面，对于上述 $x_1, x_2, \cdots, x_r$，令 $y_i = x_i + r - i, i = 1, 2, \cdots, r$，就得到一组对应的 $y_1, y_2, \cdots, y_r$.

例 1.4.8

设将 n 无序拆分成恰好 r 个部分的方案有 $p_r(n)$ 种，证明对于给定的 r 有

$$p_r(n) \sim \frac{n^{r-1}}{r!(r-1)!}.$$

证明.

考虑 n 的一种无序拆分方案 $n = x_1 + x_2 + \cdots + x_r$，其中 $x_1 \geqslant x_2 \geqslant \cdots \geqslant x_r \geqslant 1$，把 $x_1, x_2, \cdots, x_r$ 做排列就可以得到方程 $x_1 + x_2 + \cdots + x_r = n$ 的一组正整数解. 由于 $x_1, x_2, \cdots, x_r$ 中的数可能存在重复，因此排列数至多是 $r!$，从而得到

$$r!\, p_r(n) \geqslant \binom{n-1}{r-1}.$$

另一方面，对于上述 $x_1, x_2, \cdots, x_r$，令 $y_i = x_i + r - i, i = 1, 2, \cdots, r$，就得到一组对应的 $y_1, y_2, \cdots, y_r$.

例如，$10 = 4 + 4 + 2$，这里 $n = 10, r = 3, x_1 = x_2 = 4, x_3 = 2$，那么对应的 $y_1 = 6, y_2 = 5, y_3 = 2$ 且 $y_1 + y_2 + y_3 = 13$.

证明(续)

显然 $y_1 > y_2 > \cdots > y_r$，且 $y_1 + y_2 + \cdots + y_r = n + \frac{r(r-1)}{2}$. 这个方程的正整数解的个数恰好等于上述 $y_1, y_2, \cdots, y_r$ 的全排列数. 因为 $y_1, y_2, \cdots, y_r$ 是彼此不等的，全排列数恰好是 $r!$ 种，大于等于 $x_1, x_2, \cdots, x_r$ 的全排列数 $r!\, p_r(n)$，因此有

$$r!\, p_r(n) \leqslant \binom{n + \frac{r(r-1)}{2} - 1}{r-1}.$$

证明(续)

显然 $y_1 > y_2 > \cdots > y_r$,且 $y_1 + y_2 + \cdots + y_r = n + \frac{r(r-1)}{2}$. 这个方程的正整数解的个数恰好等于上述 $y_1, y_2, \cdots, y_r$ 的全排列数. 因为 $y_1, y_2, \cdots, y_r$ 是彼此不等的,全排列数恰好是 $r!$ 种,大于等于 $x_1, x_2, \cdots, x_r$ 的全排列数 $r!\, p_r(n)$,因此有

$$r!\, p_r(n) \leqslant \binom{n + \frac{r(r-1)}{2} - 1}{r-1}.$$

综合上述两个结果得

$$\frac{(n-1)!}{(n-r)!} \leqslant r!(r-1)!\ p_r(n) \leqslant \frac{\left[n + \frac{r(r-1)}{2} - 1\right]!}{\left[n + \frac{r(r-1)}{2} - r\right]!}.$$

从上述公式不难看出,当 r 给定时,随着 n 的增长,$p_r(n)$ 趋向于 $\frac{n^{r-1}}{r!(r-1)!}$.
这个结果给出了拆分方案数 $p_r(n)$ 随着 n 增长的一个近似估计. 例如,把 n 拆分成 3 个部分,拆分方案数 $p_3(n)$ 是 $O(n^2)$ 的量级;把 n 拆分成 7 个部分,拆分方案数 $p_7(n)$ 相当于 $O(n^6)$ 量级.

有序拆分的计数问题

定理 1.4.2

设 N 是正整数，将 N 允许重复地有序拆分成 r 个部分的方案数为 $C(N-1, r-1)$.

有序拆分的计数问题

定理 1.4.2

设 N 是正整数，将 N 允许重复地有序拆分成 r 个部分的方案数为 $C(N-1, r-1)$.

证明.

设 $N = a_1 + a_2 + \cdots + a_r$ 是满足条件的拆分，令

$$S_i = \sum_{k=1}^{i} a_k, \ i = 1, 2, \cdots, r,$$

那么

$$0 < S_1 < S_2 < \cdots < S_r = N.$$

不难看出，拆分方案与这些 S_i 的选择方法是一一对应的. 下面计数对这些 S_i 有多少种不同的选择方法. 由于 $r-1$ 个 $S_i, i = 1, 2, \cdots, r-1$, 取值于集合 $\{1, 2, \cdots, N-1\}$，所以选择方法数是 $C(N-1, r-1)$. □

根据这个定理，使用加法法则，不难得到下述推论.

推论 1.4.1

对正整数 N 做任意重复的有序拆分，方案数为

$$\sum_{r=1}^{N} \binom{N-1}{r-1} = 2^{N-1}.$$

根据这个定理，使用加法法则，不难得到下述推论.

推论 1.4.1

对正整数 N 做任意重复的有序拆分，方案数为

$$\sum_{r=1}^{N} \binom{N-1}{r-1} = 2^{N-1}.$$

对于不允许重复的有序拆分问题，可以分两步处理. 先将 N 不允许重复进行无序拆分，对应的生成函数是

$$G(x) = (1+x)(1+x^2)\cdots(1+x^n).$$

$G(x)$ 中 x^N 的系数就是无序拆分的方案数. 针对每种无序的拆分方案，计数被拆分部分的全排列数，然后将所有的结果相加，就可以得到所求的拆分方案数.
以上用生成函数解决了多重集的 r 组合数、不定方程解的计数、正整数拆分方案的计数等问题.

11.5 指数生成函数及其应用

本节将进一步引入指数生成函数，并讨论它在有序计数中的应用.

定义 1.5.1

设 $\{a_n\}$ 为序列，称

$$G_{\mathrm{e}}(x)=\sum_{n=0}^{+\infty} a_n \frac{x^n}{n!}$$

为 $\{a_n\}$ 的指数生成函数 .

11.5 指数生成函数及其应用

本节将进一步引入指数生成函数，并讨论它在有序计数中的应用.

定义 1.5.1

设 $\{a_n\}$ 为序列，称

$$G_{\mathrm{e}}(x)=\sum_{n=0}^{+\infty} a_n \frac{x^n}{n!}$$

为 $\{a_n\}$ 的指数生成函数 .

例 1.5.1

设 $b_n=1$，则 $\{b_n\}$ 的指数生成函数为

$$G_{\mathrm{e}}(x)=\sum_{n=0}^{+\infty} \frac{x^n}{n!}=\mathrm{e}^x.$$

11.5 指数生成函数及其应用

本节将进一步引入指数生成函数，并讨论它在有序计数中的应用.

定义 1.5.1

设 $\{a_n\}$ 为序列，称

$$G_{\mathrm{e}}(x)=\sum_{n=0}^{+\infty}a_n\frac{x^n}{n!}$$

为 $\{a_n\}$ 的指数生成函数.

例 1.5.1

设 $b_n=1$，则 $\{b_n\}$ 的指数生成函数为

$$G_{\mathrm{e}}(x)=\sum_{n=0}^{+\infty}\frac{x^n}{n!}=\mathrm{e}^x.$$

例 1.5.2

给定正整数 $m, a_n=P(m,n)$，则 $\{a_n\}$ 的指数生成函数为

$$G_{\mathrm{e}}(x)=\sum_{n=0}^{+\infty}P(m,n)\frac{x^n}{n!}=\sum_{n=0}^{+\infty}\frac{m!}{n!(m-n)!}x^n=\sum_{n=0}^{+\infty}\binom{m}{n}x^n=(1+x)^m.$$

11.5 指数生成函数及其应用

本节将进一步引入指数生成函数，并讨论它在有序计数中的应用.

定义 1.5.1

设 $\{a_n\}$ 为序列，称

$$G_{\mathrm{e}}(x)=\sum_{n=0}^{+\infty}a_n\frac{x^n}{n!}$$

为 $\{a_n\}$ 的指数生成函数.

例 1.5.1

设 $b_n=1$，则 $\{b_n\}$ 的指数生成函数为

$$G_{\mathrm{e}}(x)=\sum_{n=0}^{+\infty}\frac{x^n}{n!}=\mathrm{e}^x.$$

例 1.5.2

给定正整数 m，$a_n=P(m,n)$，则 $\{a_n\}$ 的指数生成函数为

$$G_{\mathrm{e}}(x)=\sum_{n=0}^{+\infty}P(m,n)\frac{x^n}{n!}=\sum_{n=0}^{+\infty}\frac{m!}{n!(m-n)!}x^n=\sum_{n=0}^{+\infty}\binom{m}{n}x^n=(1+x)^m.$$

不难看出，$(1+x)^m$ 既是集合组合数序列 $\{C(m,n)\}$ 的普通生成函数，也是集合排列数序列 $\{P(m,n)\}$ 的指数生成函数.

定理 1.5.1

设 $S = \{n_1 \cdot a_1, n_2 \cdot a_2, \cdots, n_k \cdot a_k\}$ 为多重集，则 S 的 r 排列数的指数生成函数为

$G_{\mathrm{e}}(x) = f_{n_1}(x) f_{n_2}(x) \cdots f_{n_k}(x)$, 其中

$f_{n_i}(x) = 1 + x + \frac{x^2}{2!} + \cdots + \frac{x^{n_i}}{n_i!}$,

$i = 1, 2, \cdots, k$.

定理 1.5.1

设 $S = \{n_1 \cdot a_1, n_2 \cdot a_2, \cdots, n_k \cdot a_k\}$ 为多重集，则 S 的 r 排列数的指数生成函数为

$G_{\mathrm{e}}(x) = f_{n_1}(x) f_{n_2}(x) \cdots f_{n_k}(x)$, 其中

$f_{n_i}(x) = 1 + x + \frac{x^2}{2!} + \cdots + \frac{x^{n_i}}{n_i!}$,

$i = 1, 2, \cdots, k$.

证明.

考察上述指数生成函数展开式中 x^r 的项，它是由 k 个因式的乘积构成的，并具有下述形式：

$$\frac{x^{m_1}}{m_1!} \frac{x^{m_2}}{m_2!} \cdots \frac{x^{m_k}}{m_k!},$$

其中 $\frac{x^{m_i}}{m_i!}$ 来自 $f_{n_i}(x)$. 注意到 $m_1, m_2, \cdots, m_k$ 满足下述不定方程：

定理 1.5.1

设 $S=\{n_1\cdot a_1, n_2\cdot a_2, \cdots, n_k\cdot a_k\}$ 为多重集，则 S 的 r 排列数的指数生成函数为

$G_{\mathrm{e}}(x)=f_{n_1}(x)f_{n_2}(x)\cdots f_{n_k}(x)$，其中

$f_{n_i}(x)=1+x+\frac{x^2}{2!}+\cdots+\frac{x^{n_i}}{n_i!}$，

$i=1,2,\cdots,k$.

证明.

考察上述指数生成函数展开式中 x^r 的项，它是由 k 个因式的乘积构成的，并具有下述形式：

$$\frac{x^{m_1}}{m_1!}\frac{x^{m_2}}{m_2!}\cdots\frac{x^{m_k}}{m_k!},$$

其中 $\frac{x^{m_i}}{m_i!}$ 来自 $f_{n_i}(x)$. 注意到 $m_1, m_2, \cdots, m_k$ 满足下述不定方程：

$$\begin{aligned} &m_1+m_2+\cdots+m_k=r,\\ &0\leqslant m_i\leqslant n_i,\ i=1,2,\cdots,k, \end{aligned} \tag{1.5.1}$$

即

$$\frac{x^{m_1+m_2+\cdots+m_k}}{m_1!m_2!\cdots m_k!}=\frac{x^r}{r!}\frac{r!}{m_1!m_2!\cdots m_k!}.$$

因此

$$a_r=\sum\frac{r!}{m_1!m_2!\cdots m_k!},$$

其中求和是对满足方程 (1.5.1) 的一切非负整数解来求. 一个非负整数解对应了 S 的一个子多重集 $\{m_1\cdot a_1, m_2\cdot a_2, \cdots, m_k\cdot a_k\}$，即 S 的一个 r 组合，而该组合的全排列数是 $\frac{r!}{m_1!m_2!\cdots m_k!}$，因此 a_r 代表了 S 的所有 r 排列数. □

用指数生成函数求解多重集的排列问题

例 1.5.3

由 $1,2,3,4$ 组成的五位数中，要求 1 出现不超过 2 次，但不能不出现，2 出现不超过 1 次，3 出现至多 3 次，4 出现偶数次. 求这样的五位数个数 N.

用指数生成函数求解多重集的排列问题

例 1.5.3

由 $1,2,3,4$ 组成的五位数中，要求 *1* 出现不超过 *2* 次，但不能不出现，*2* 出现不超过 *1* 次，*3* 出现至多 *3* 次，*4* 出现偶数次. 求这样的五位数个数 N.

解

$$
\begin{aligned}
G_{\mathrm{e}}(x) &= \left(\frac{x}{1!}+\frac{x^2}{2!}\right)(1+x)\left(1+x+\frac{x^2}{2!}+\frac{x^3}{3!}\right)\left(1+\frac{x^2}{2!}+\frac{x^4}{4!}\right) \\
&= x+5\frac{x^2}{2!}+18\frac{x^3}{3!}+64\frac{x^4}{4!}+215\frac{x^5}{5!}+\cdots,
\end{aligned}
$$

故这样的五位数个数 $N=215$. □

例 1.5.4

由 A, B, C, D, E, F 构成长度为 n 的序列，如果要求在排列中 A 与 B 出现的次数之和为偶数，问：这样的序列有多少个？

例 1.5.4

由 A, B, C, D, E, F 构成长度为 n 的序列，如果要求在排列中 A 与 B 出现的次数之和为偶数，问：这样的序列有多少个？

解

设所求序列数为 a_n，在这样的序列中 A 和 B 出现次数的奇偶性一样，指数函数中对应于 A 和 B 的成分或者同时为 $\frac{\mathrm{e}^x+\mathrm{e}^{-x}}{2}$，或者同时为 $\frac{\mathrm{e}^x-\mathrm{e}^{-x}}{2}$. 因此，$\{a_n\}$ 的指数生成函数为

$$\begin{aligned} G_{\mathrm{e}}(x) &= \left(\frac{\mathrm{e}^x+\mathrm{e}^{-x}}{2}\right)^2 \mathrm{e}^{4x} + \left(\frac{\mathrm{e}^x-\mathrm{e}^{-x}}{2}\right)^2 \mathrm{e}^{4x} \\ &= \frac{\mathrm{e}^{2x}+\mathrm{e}^{-2x}}{2}\mathrm{e}^{4x} \\ &= \frac{1}{2}\mathrm{e}^{6x} + \frac{1}{2}\mathrm{e}^{2x}. \end{aligned}$$

将这个函数展开得到 $x^n/n!$ 项的系数是

$$a_n = \frac{6^n+2^n}{2}.$$

11.6 卡塔兰数与斯特林数

定义 1.6.1

给定一个凸 $n+1$ 边形，通过在内部不相交的对角线把它划分成三角形，不同的划分方案数称作 卡塔兰(*Catalan*)数，记作 h_n.

11.6 卡塔兰数与斯特林数

定义 1.6.1

给定一个凸 $n+1$ 边形，通过在内部不相交的对角线把它划分成三角形，不同的划分方案数称作 卡塔兰(*Catalan*)数，记作 h_n.

例如，$h_4 = 5$，说明对一个凸五边形进行三角划分，共有 5 种不同的方案. 图 1.6.1 列出了这 5 种方案.

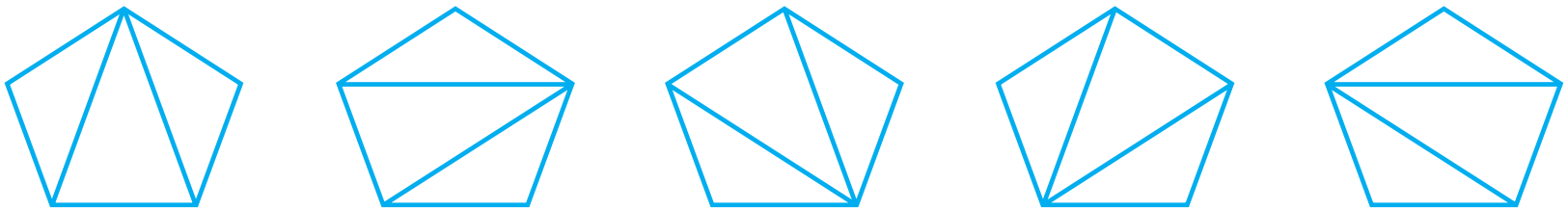


图 1.6.1

为确定 h_n，先要建立关于 h_n 的递推方程. 考虑 $n+1$ 条边的多边形，端点分别被标记为 $A_1, A_2, \cdots, A_{n+1}$. 将 A_1A_{n+1} 的边记为 a，作为三角形的底边. 选择底边以外的顶点 $A_{k+1}, k = 1, 2, \cdots, n-1$，那么底边 a，边 $A_{k+1}A_1$ 和 $A_{n+1}A_{k+1}$ 就构成三角形 T，T 将多边形划分成 R_1 和 R_2 两个部分，分别为 $k+1$ 边形和 $n-k+1$ 边形. 划分结果如图 1.6.2 所示.

为确定 h_n，先要建立关于 h_n 的递推方程. 考虑 $n+1$ 条边的多边形，端点分别被标记为 $A_1, A_2, \cdots, A_{n+1}$. 将 A_1A_{n+1} 的边记为 a，作为三角形的底边. 选择底边以外的顶点 $A_{k+1}, k = 1, 2, \cdots, n-1$，那么底边 a，边 $A_{k+1}A_1$ 和 $A_{n+1}A_{k+1}$ 就构成三角形 T，T 将多边形划分成 R_1 和 R_2 两个部分，分别为 $k+1$ 边形和 $n-k+1$ 边形. 划分结果如图 1.6.2 所示.

$k+1$ 边形 R_1 和 $n-k+1$ 边形 R_2 的三角划分方案数分别为 h_k 和 h_{n-k}，根据乘法法则和加法法则，$\sum_{k=1}^{n-1} h_k h_{n-k}$ 就是 $n+1$ 边形的划分方案总数. 因此得到下述递推方程：

为确定 h_n，先要建立关于 h_n 的递推方程. 考虑 $n+1$ 条边的多边形，端点分别被标记为 $A_1, A_2, \cdots, A_{n+1}$. 将 A_1A_{n+1} 的边记为 a，作为三角形的底边. 选择底边以外的顶点 $A_{k+1}, k = 1, 2, \cdots, n-1$，那么底边 a，边 $A_{k+1}A_1$ 和 $A_{n+1}A_{k+1}$ 就构成三角形 T，T 将多边形划分成 R_1 和 R_2 两个部分，分别为 $k+1$ 边形和 $n-k+1$ 边形. 划分结果如图 1.6.2 所示.

$k+1$ 边形 R_1 和 $n-k+1$ 边形 R_2 的三角划分方案数分别为 h_k 和 h_{n-k}，根据乘法法则和加法法则，$\sum_{k=1}^{n-1} h_k h_{n-k}$ 就是 $n+1$ 边形的划分方案总数. 因此得到下述递推方程：

$$\begin{cases} h_n = \sum_{k=1}^{n-1} h_k h_{n-k}, \ n \geqslant 2, \\ h_1 = 1. \end{cases}$$

例 1.4.3 利用生成函数求解过这个递推方程，它的解是

$$h_n = \frac{1}{n}\binom{2n-2}{n-1}.$$

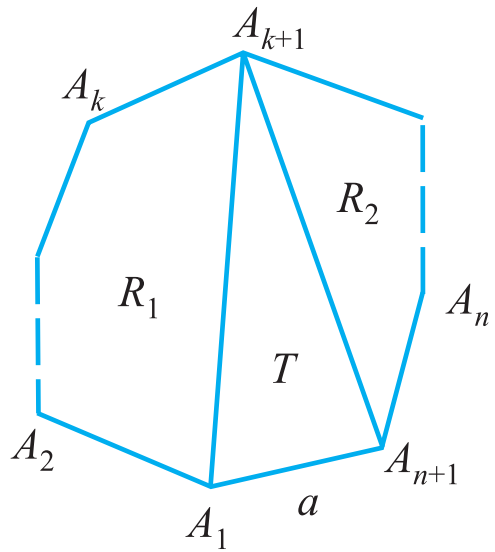


图 1.6.2

卡塔兰数出现在许多组合计数问题中，如前面遇到的从 $(0,0)$ 点到 (n,n) 点除端点外不接触对角线的非降路径计数问题、堆栈输出序列的计数问题等. 矩阵乘法在科学计算中有着广泛的应用，下面考虑一个矩阵链相乘问题.

卡塔兰数出现在许多组合计数问题中，如前面遇到的从 $(0,0)$ 点到 (n,n) 点除端点外不接触对角线的非降路径计数问题、堆栈输出序列的计数问题等. 矩阵乘法在科学计算中有着广泛的应用，下面考虑一个矩阵链相乘问题.

例 1.6.1

设 $\boldsymbol{A}_1, \boldsymbol{A}_2, \cdots, \boldsymbol{A}_n$ 为 n 个矩阵的序列，其中 $\boldsymbol{A}_i$ 为 P_{i-1} 行、P_i 列的矩阵，$i = 1, 2, \cdots, n$. 问题的输入是上述矩阵的行数和列数 $\boldsymbol{P}_0, \boldsymbol{P}_1, \cdots, \boldsymbol{P}_n$ 构成的序列，即向量 $\boldsymbol{P} = \langle P_0, P_1, \cdots, P_n \rangle$. 例如 $\boldsymbol{P} = \langle 10, 100, 5, 50 \rangle$ 表示有 *3* 个矩阵 $\boldsymbol{A}_1, \boldsymbol{A}_2, \boldsymbol{A}_3$，其中 $\boldsymbol{A}_1 : 10 \times 100,\ \boldsymbol{A}_2 : 100 \times 5,\ \boldsymbol{A}_3 : 5 \times 50$.

给定两个矩阵 $\boldsymbol{A}_1$ 和 $\boldsymbol{A}_2$，其输入为 $\boldsymbol{P} = \langle P_0, P_1, P_2 \rangle$. 这意味着 $\boldsymbol{A}_1 = (a_{ij}), i = 1, 2, \cdots, P_0, j = 1, 2, \cdots, P_1, \boldsymbol{A}_2 = (b_{jk}), j = 1, 2, \cdots, P_1, k = 1, 2, \cdots, P_2$. $\boldsymbol{A}_1$ 与 $\boldsymbol{A}_2$ 的乘积 $\boldsymbol{A} = (c_{ik})$ 含有 $P_0 \times P_2$ 个项，对于给定的 i 和 k，项 c_{ik} 的计算依照下面的公式

$$c_{ik} = \sum_{j=1}^{P_1} a_{ij} b_{jk}.$$

这需要做 P_1 次乘法和 $P_1 - 1$ 次加法. 因此计算项 c_{ik} 需要 P_1 次乘法. 于是，计算整个乘积 $\boldsymbol{A}$ 需要做 $P_0 P_1 P_2$ 次乘法.

例1.6.1(续)

考虑矩阵链 $\boldsymbol{A}_1, \boldsymbol{A}_2, \cdots, \boldsymbol{A}_n$，其输入是 $\boldsymbol{P} = \langle P_0, P_1, \cdots, P_n \rangle$. 要计算乘积 $\boldsymbol{A} = \boldsymbol{A}_1 \boldsymbol{A}_2 \cdots \boldsymbol{A}_n$，需要通过 $n-1$ 次两个矩阵的相乘来完成. 如果采用不同的乘法次序，所做的工作量(乘法次数)是不一样的. 例如，对于上面给出的 $\boldsymbol{A}_1, \boldsymbol{A}_2$ 和 $\boldsymbol{A}_3$，采用下面两种乘法顺序所做的工作量分别为

$$(\boldsymbol{A}_1 \boldsymbol{A}_2)\boldsymbol{A}_3 : 10 \times 100 \times 5 + 10 \times 5 \times 50 = 7\ 500,$$
$$\boldsymbol{A}_1(\boldsymbol{A}_2 A_3) : 10 \times 100 \times 50 + 100 \times 5 \times 50 = 75\ 000.$$

显然第 1 种乘法次序比第 2 种乘法次序要好得多，它的工作量仅是第 2 种的 $1/10$. 问题是：给定矩阵链输入 $\boldsymbol{P} = \langle P_0, P_1, \cdots, P_n \rangle$，如何确定一种乘法次序，使得总的工作量达到最小?

例1.6.1(续)

考虑矩阵链 $\boldsymbol{A}_1, \boldsymbol{A}_2, \cdots, \boldsymbol{A}_n$，其输入是 $\boldsymbol{P} = \langle P_0, P_1, \cdots, P_n \rangle$. 要计算乘积 $\boldsymbol{A} = \boldsymbol{A}_1 \boldsymbol{A}_2 \cdots \boldsymbol{A}_n$，需要通过 $n-1$ 次两个矩阵的相乘来完成. 如果采用不同的乘法次序，所做的工作量(乘法次数)是不一样的. 例如，对于上面给出的 $\boldsymbol{A}_1, \boldsymbol{A}_2$ 和 $\boldsymbol{A}_3$，采用下面两种乘法顺序所做的工作量分别为

$$(\boldsymbol{A}_1 \boldsymbol{A}_2)\boldsymbol{A}_3 : 10 \times 100 \times 5 + 10 \times 5 \times 50 = 7\ 500,$$
$$\boldsymbol{A}_1(\boldsymbol{A}_2 A_3) : 10 \times 100 \times 50 + 100 \times 5 \times 50 = 75\ 000.$$

显然第 1 种乘法次序比第 2 种乘法次序要好得多，它的工作量仅是第 2 种的 $1/10$. 问题是：给定矩阵链输入 $\boldsymbol{P} = \langle P_0, P_1, \cdots, P_n \rangle$，如何确定一种乘法次序，使得总的工作量达到最小?

这是一个组合优化问题. 问题的搜索空间是所有可能的乘法次序，每种次序对应一个表示工作量的值. 一种简单的算法就是：按照某种搜索顺序，遍历整个搜索空间，对每个可能的乘法次序计算工作量，最终找到具有最小工作量的乘法次序. 这种蛮力算法在许多简单问题中是非常有效的. 但是对于这个问题它能够工作吗? 需要估计一下搜索空间的大小.

例1.6.1(续)

用 $\boldsymbol{A}_{i..j}$ 表示矩阵链 $\boldsymbol{A}_i\boldsymbol{A}_{i+1}\cdots\boldsymbol{A}_j$. 假设 $n-1$ 次相乘的最后一次发生在两个部分 $\boldsymbol{A}_{1..m}$ 和 $\boldsymbol{A}_{m+1..n}$ 之间,其中 $1 \leqslant m < n$. 那么,n 个矩阵相乘的次序个数 T_n 满足 $T_n = \sum_{m=1}^{n-1} T_m T_{n-m},\ n \geqslant 2,\ T_1 = 1.$

这与卡塔兰数 h_n 的递推方程和初值完全一样. 因此,

$$T_n = \frac{1}{n}\binom{2n-2}{n-1} = \Theta\left(\frac{1}{n}\frac{(2n)!}{n!n!}\right).$$

例1.6.1(续)

用 $\boldsymbol{A}_{i..j}$ 表示矩阵链 $\boldsymbol{A}_i\boldsymbol{A}_{i+1}\cdots\boldsymbol{A}_j$. 假设 $n-1$ 次相乘的最后一次发生在两个部分 $\boldsymbol{A}_{1..m}$ 和 $\boldsymbol{A}_{m+1..n}$ 之间,其中 $1 \leqslant m < n$. 那么,n 个矩阵相乘的次序个数 T_n 满足 $T_n = \sum_{m=1}^{n-1} T_m T_{n-m},\ n \geqslant 2,\ T_1 = 1$.

这与卡塔兰数 h_n 的递推方程和初值完全一样. 因此,

$$T_n = \frac{1}{n}\binom{2n-2}{n-1} = \Theta\left(\frac{1}{n}\frac{(2n)!}{n!n!}\right).$$

利用斯特林公式 $n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1+\Theta\left(\frac{1}{n}\right)\right)$ 得到

$$\begin{aligned} T_n &= \Theta\left(\frac{1}{n}\frac{\sqrt{2\pi 2n}\left(\frac{2n}{\mathrm{e}}\right)^{2n}}{\sqrt{2\pi n}\left(\frac{n}{\mathrm{e}}\right)^n\sqrt{2\pi n}\left(\frac{n}{\mathrm{e}}\right)^n}\right) \\ &= \Theta\left(\frac{1}{n}\frac{n^{\frac{1}{2}}2^{2n}n^{2n}\mathrm{e}^n\mathrm{e}^n}{\mathrm{e}^{2n}n^{\frac{1}{2}}n^n n^{\frac{1}{2}}n^n}\right) = \Theta(2^{2n}/n^{\frac{3}{2}}). \end{aligned}$$

这是指数量级的搜索空间,因此蛮力算法的时间复杂度是指数函数,对于较大的 n 是不能工作的. 而动态规划的算法设计技术可以在 $O(n^3)$ 的时间复杂度下求得这个问题的最优解.

第一类斯特林数

定义 1.6.2

考虑多项式 $x(x-1)(x-2)\cdots(x-n+1)$ **的展开式**

$$S_n x^n - S_{n-1} x^{n-1} + S_{n-2} x^{n-2} - \cdots + (-1)^{n-1} S_1 x,$$

将上述展开式中 x^r **的系数的绝对值** S_r **记作** $\begin{bmatrix} n \\ r \end{bmatrix}$**,称作** 第一类斯特林(*Stirling*)数 .

第一类斯特林数

定义 1.6.2

考虑多项式 $x(x-1)(x-2)\cdots(x-n+1)$ 的展开式

$$S_n x^n - S_{n-1}x^{n-1} + S_{n-2}x^{n-2} - \cdots + (-1)^{n-1}S_1 x,$$

将上述展开式中 x^r 的系数的绝对值 S_r 记作 $\begin{bmatrix} n \\ r \end{bmatrix}$,称作 第一类斯特林(*Stirling*)数 .

不难证明第一类斯特林数满足下面的递推方程.

命题 1.6.1

设 n, r 均为正整数,则有

$$\begin{cases} \begin{bmatrix} n \\ r \end{bmatrix} = (n-1)\begin{bmatrix} n-1 \\ r \end{bmatrix} + \begin{bmatrix} n-1 \\ r-1 \end{bmatrix}, \ n > r \geqslant 1, \\ \begin{bmatrix} n \\ 0 \end{bmatrix} = 0, \begin{bmatrix} n \\ 1 \end{bmatrix} = (n-1)!. \end{cases}$$

证明.

将等式

$$x(x-1)\cdots(x-n+2)=\begin{bmatrix} n-1 \\ n-1 \end{bmatrix}x^{n-1}-\begin{bmatrix} n-1 \\ n-2 \end{bmatrix}x^{n-2}+\cdots$$

代入下述等式后得到

$$x(x-1)\cdots(x-n+2)(x-n+1)=\left(\begin{bmatrix} n-1 \\ n-1 \end{bmatrix}x^{n-1}-\begin{bmatrix} n-1 \\ n-2 \end{bmatrix}x^{n-2}+\cdots\right)(x-n+1).$$

由于两边的 x^r 的系数应该相等，所以有

$$\begin{bmatrix} n \\ r \end{bmatrix}=(n-1)\begin{bmatrix} n-1 \\ r \end{bmatrix}+\begin{bmatrix} n-1 \\ r-1 \end{bmatrix}.$$

证明.

将等式

$$x(x-1)\cdots(x-n+2)=\begin{bmatrix}n-1\\n-1\end{bmatrix}x^{n-1}-\begin{bmatrix}n-1\\n-2\end{bmatrix}x^{n-2}+\cdots$$

代入下述等式后得到

$$x(x-1)\cdots(x-n+2)(x-n+1)=\left(\begin{bmatrix}n-1\\n-1\end{bmatrix}x^{n-1}-\begin{bmatrix}n-1\\n-2\end{bmatrix}x^{n-2}+\cdots\right)(x-n+1).$$

由于两边的 x^r 的系数应该相等，所以有

$$\begin{bmatrix}n\\r\end{bmatrix}=(n-1)\begin{bmatrix}n-1\\r\end{bmatrix}+\begin{bmatrix}n-1\\r-1\end{bmatrix}.$$

下面计算两个初值. 根据第一类斯特林数的定义，不难得到 $\begin{bmatrix}n\\0\end{bmatrix}=0$. 为得到展开式中 x 项的系数，除了乘积项中的第一项提供 x 之外，其他各项只能提供常数，即分别贡献 $-1,-2,\cdots,-(n-1)$. 如果不考虑正负号，这些项的乘积是 $(n-1)!$. 因此 $\begin{bmatrix}n\\1\end{bmatrix}=(n-1)!$. □

第一类斯特林数(续)

命题 1.6.2

对任意正整数 n,有

1. $\begin{bmatrix} n \\ n \end{bmatrix} = 1.$
2. $\begin{bmatrix} n \\ n-1 \end{bmatrix} = \binom{n}{2} = \frac{n(n-1)}{2}.$
3. $\sum_{r=1}^{n} \begin{bmatrix} n \\ r \end{bmatrix} = n!.$

第一类斯特林数(续)

命题 1.6.2

对任意正整数 n,有

1. $\begin{bmatrix} n \\ n \end{bmatrix} = 1$.
2. $\begin{bmatrix} n \\ n-1 \end{bmatrix} = \binom{n}{2} = \frac{n(n-1)}{2}$.
3. $\sum_{r=1}^{n} \begin{bmatrix} n \\ r \end{bmatrix} = n!$.

证明.

命题中前两个恒等式的证明比较简单,只需使用第一类斯特林数的定义. 对于第 3 个恒等式,注意到对任意的正整数 r, $1 \leq r \leq n$, 第一类斯特林数 $\begin{bmatrix} n \\ r \end{bmatrix}$ 正好是多项式 $x(x+1)(x+2)\cdots(x+n-1)$ 的展开式中 x^r 的系数. 因此,在该多项式中取 $x = 1$,则知 $\sum_{r=1}^{n} \begin{bmatrix} n \\ r \end{bmatrix} = n!$. □

第二类斯特林数

第一类斯特林数与 n 个不同对象划分的计数相关，在这些划分中 n 个对象被分成 r 个环排列. 与 n 个不同对象的划分相关的另一类计数是第二类斯特林数，通常用放球问题来定义它.

第二类斯特林数

第一类斯特林数与 n 个不同对象划分的计数相关，在这些划分中 n 个对象被分成 r 个环排列. 与 n 个不同对象的划分相关的另一类计数是第二类斯特林数，通常用放球问题来定义它.

定义 1.6.3

n 个不同的球恰好放到 r 个相同的盒子里的方法数称作 *第二类斯特林(Stirling)数*，记作 $\left\{ {n \atop r} \right\}$.

第二类斯特林数

第一类斯特林数与 n 个不同对象划分的计数相关，在这些划分中 n 个对象被分成 r 个环排列. 与 n 个不同对象的划分相关的另一类计数是第二类斯特林数，通常用放球问题来定义它.

定义 1.6.3

n 个不同的球恰好放到 r 个相同的盒子里的方法数称作 *第二类斯特林(Stirling)数*，记作 $\left\{ {n \atop r} \right\}$.

例如，$\left\{ {4 \atop 2} \right\} = 7$，下面给出这 7 种放球方案：

$$a, b, c|d \qquad a, c, d|b \qquad a, b, d|c \qquad b, c, d|a$$
$$a, b|c, d \qquad a, c|b, d \qquad a, d|b, c$$

第二类斯特林数(续)

命题 1.6.3

设 n, r 均为正整数，则有

$$\left\{ {n \atop r} \right\} = r\left\{ {n-1 \atop r} \right\} + \left\{ {n-1 \atop r-1} \right\}, \ \left\{ {n \atop 0} \right\} = 0, \ \left\{ {n \atop 1} \right\} = 1.$$

第二类斯特林数(续)

命题 1.6.3

设 n, r 均为正整数,则有

$$\left\{ {n \atop r} \right\} = r\left\{ {n-1 \atop r} \right\} + \left\{ {n-1 \atop r-1} \right\}, \ \left\{ {n \atop 0} \right\} = 0, \ \left\{ {n \atop 1} \right\} = 1.$$

证明.

考虑将 n 个不同的球恰好放到 r 个盒子. 取球 a_1,将放球的方法进行如下分类. 若 a_1 单独放在一个盒子里,剩下的是对其他 $n-1$ 个球的放置问题,有 $\left\{ {n-1 \atop r-1} \right\}$ 种方法. 若 a_1 与别的球放在同一盒子里,则可以先把 $n-1$ 个球恰好放到 r 个盒子里,有 $\left\{ {n-1 \atop r} \right\}$ 方法,然后把 a_1 插入 r 个盒子,有 r 种方法. 因此,总共 $r\left\{ {n-1 \atop r} \right\}$ 种方法. 根据加法法则得到

$$\left\{ {n \atop r} \right\} = r\left\{ {n-1 \atop r} \right\} + \left\{ {n-1 \atop r-1} \right\}.$$

第二类斯特林数(续)

命题 1.6.3

设 n, r 均为正整数，则有

$$\left\{ {n \atop r} \right\} = r\left\{ {n-1 \atop r} \right\} + \left\{ {n-1 \atop r-1} \right\}, \ \left\{ {n \atop 0} \right\} = 0, \ \left\{ {n \atop 1} \right\} = 1.$$

证明.

考虑将 n 个不同的球恰好放到 r 个盒子. 取球 a_1，将放球的方法进行如下分类. 若 a_1 单独放在一个盒子里，剩下的是对其他 $n-1$ 个球的放置问题，有 $\left\{ {n-1 \atop r-1} \right\}$ 种方法. 若 a_1 与别的球放在同一盒子里，则可以先把 $n-1$ 个球恰好放到 r 个盒子里，有 $\left\{ {n-1 \atop r} \right\}$ 方法，然后把 a_1 插入 r 个盒子，有 r 种方法. 因此，总共 $r\left\{ {n-1 \atop r} \right\}$ 种方法. 根据加法法则得到

$$\left\{ {n \atop r} \right\} = r\left\{ {n-1 \atop r} \right\} + \left\{ {n-1 \atop r-1} \right\}.$$

根据第二类斯特林数的定义，不难得到 $\left\{ {n \atop 0} \right\} = 0, \left\{ {n \atop 1} \right\} = 1.$ □

第二类斯特林数(续)

第二类斯特林数的递推公式也可以采用图形表示. 图 1.6.3 给出了当 $n = 1, 2, 3, 4, 5$ 时所有第二类斯特林数的值.

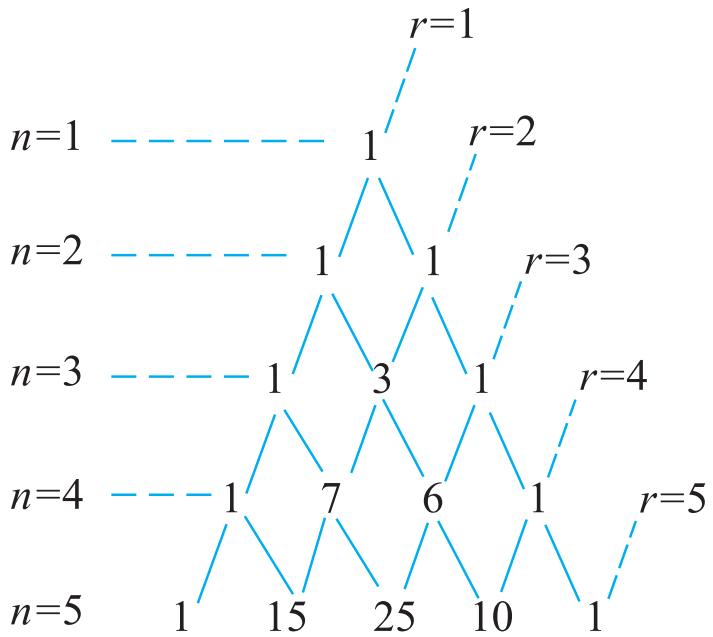


图 1.6.3

命题 1.6.4

第二类斯特林数满足以下恒等式：

1. $\left\{ {n \atop 2} \right\} = 2^{n-1} - 1.$
2. $\left\{ {n \atop n-1} \right\} = \binom{n}{2}.$
3. $\left\{ {n \atop n} \right\} = 1.$
4. $\sum \binom{n}{n_1 n_2 \cdots n_m} = m! \left\{ {n \atop m} \right\}$，其中 $\sum$ 是对 $n_1 + n_2 + \cdots + n_m = n$ 的正整数解求和.
5. $\sum_{k=1}^{m} \binom{m}{k} \left\{ {m \atop k} \right\} k! = m^n.$
6. $\left\{ {n+1 \atop r} \right\} = \sum_{i=0}^{n} \binom{n}{i} \left\{ {i \atop r-1} \right\}.$

命题 1.6.4

第二类斯特林数满足以下恒等式：

1. $\left\{ {n \atop 2} \right\} = 2^{n-1} - 1.$
2. $\left\{ {n \atop n-1} \right\} = \binom{n}{2}.$
3. $\left\{ {n \atop n} \right\} = 1.$
4. $\sum \binom{n}{n_1 n_2 \cdots n_m} = m! \left\{ {n \atop m} \right\}$，其中 $\sum$ 是对 $n_1 + n_2 + \cdots + n_m = n$ 的正整数解求和.
5. $\sum_{k=1}^{m} \binom{m}{k} \left\{ {m \atop k} \right\} k! = m^n.$
6. $\left\{ {n+1 \atop r} \right\} = \sum_{i=0}^{n} \binom{n}{i} \left\{ {i \atop r-1} \right\}.$

证明.

这里选证其中的某些公式，剩下的留给读者思考.

(1) 将 n 个不同的球放到 2 个盒子里. 先选定一个球，如 a_1，把它放在一个盒子里. 然后放剩下的 $n-1$ 个球，每个球有 2 种选择，总计 2^{n-1} 种放法. 但是，这些球全落入 a_1 所在盒子的选法不符合要求，所以要从中减去 1 种选法.

命题1.6.4证明(续)

(4) 使用组合分析的方法证明. 首先证明等式左边计数了 n 个不同的球恰好放到 m 个不同的盒子的方法. 当所有的 n_i 为正整数,且 $n_1 + n_2 + \cdots + n_m = n$ 时,$\binom{n}{n_1 n_2 \cdots n_m}$ 对应了 n 个不同的球恰好放到 m 个不同盒子里,并且使得第一个盒子含有 n_1 个球、第二个盒子含有 n_2 个球、$\cdots$、第 m 个盒子含有 n_m 个球的方法数. 对所有满足上述条件的 $n_1, n_2, \cdots, n_m$,通过对 $\binom{n}{n_1 n_2 \cdots n_m}$ 求和就得到 n 个不同的球恰好放到 m 个不同的盒子的方法数.

再看等式右边. 先把 n 个不同的球放到 m 个相同的盒子,有 $\left\{ {n \atop m} \right\}$ 种方法;然后对盒子进行编号,编号的方式有 $m!$ 种. 因此,$m!\left\{ {n \atop m} \right\}$ 也计数了 n 个不同的球恰好放到 m 个不同的盒子的方法.

命题1.6.4证明(续)

(4) 使用组合分析的方法证明. 首先证明等式左边计数了 n 个不同的球恰好放到 m 个不同的盒子的方法. 当所有的 n_i 为正整数,且 $n_1+n_2+\cdots+n_m=n$ 时,$\binom{n}{n_1 n_2 \cdots n_m}$ 对应了 n 个不同的球恰好放到 m 个不同盒子里,并且使得第一个盒子含有 n_1 个球、第二个盒子含有 n_2 个球、$\cdots$、第 m 个盒子含有 n_m 个球的方法数. 对所有满足上述条件的 $n_1, n_2, \cdots, n_m$,通过对 $\binom{n}{n_1 n_2 \cdots n_m}$ 求和就得到 n 个不同的球恰好放到 m 个不同的盒子的方法数.

再看等式右边. 先把 n 个不同的球放到 m 个相同的盒子,有 $\left\{ {n \atop m} \right\}$ 种方法;然后对盒子进行编号,编号的方式有 $m!$ 种. 因此,$m!\left\{ {n \atop m} \right\}$ 也计数了 n 个不同的球恰好放到 m 个不同的盒子的方法.

(6) 等式左边计数了 $n+1$ 个不同的球恰好放入 r 个相同的盒子的方法. 先选定一个球,如 a_1,把它放在一个盒子里. 将其余 n 个球的放法根据剩下 $r-1$ 个盒子含有的球数 i 进行分类,$i=0,1,\cdots,r-1,r,\cdots,n$. 对于给定的 i,先从 n 个不同的球中选出 i 个球,有 $\binom{n}{i}$ 种选法. 然后将这 i 个球恰好放入 $r-1$ 个相同的盒子,有 $\left\{ {i \atop r-1} \right\}$ 种放法. 这里要注意,当 $i<r-1$ 时,$\left\{ {i \atop r-1} \right\}$ 的值等于 0. 对 i 求和就得到 $n+1$ 个不同的球恰好放入 r 个相同的盒子的方法数.

第二类斯特林数来源于一个重要的组合计数问题——放球问题. 设有 n 个球, m 个盒子, 下表列出了与这个放球问题相关的计数结果.

表 1.6.1

球区别	盒区别	是否空盒	模型	方案计数
有	有	有	选取	m^n
有	有	无	放球子模型	$m!\left\{ {n \atop m} \right\}$
有	无	有		$\sum_{k=1}^{m} \left\{ {n \atop k} \right\}$
有	无	无		$\left\{ {n \atop m} \right\}$
无	有	有	不定方程	$C(n+m-1, n)$
无	有	无		$C(n-1, m-1)$
无	无	有	正整数拆分	$G(x)=\frac{1}{(1-x)(1-x^2)\cdots(1-x^m)}$, x^n 的系数
无	无	无		$G(x)=\frac{x^m}{(1-x)(1-x^2)\cdots(1-x^m)}$, x^n 的系数

例 1.6.2

设 A, B 为集合，$|A| = n, |B| = m$，问：

1. 从 A 到 B 的关系有多少个？
2. A 上关系有多少个？其中等价关系有多少个？
3. 从 A 到 B 的函数有多少个？其中单射函数有多少个？满射函数有多少个？？双射函数有多少个？

例 1.6.2

设 A, B 为集合，$|A| = n, |B| = m$，问：

1. 从 A 到 B 的关系有多少个？
2. A 上关系有多少个？其中等价关系有多少个？
3. 从 A 到 B 的函数有多少个？其中单射函数有多少个？满射函数有多少个？？双射函数有多少个？

解

(1) $|A| = n, |B| = m$，从 A 到 B 的关系是 $A \times B$ 的子集，$|A \times B| = mn$，故从 A 到 B 有 2^{mn} 个二元关系.

(2) A 上的关系有 2^{n^2} 个. 任何 A 上的等价关系都对应 A 的划分. 根据划分块的个数 k, $k = 1, 2, \cdots, n$, 将划分进行分类，具有 k 个划分块的划分相当于将 n 个不同的球恰好放入 k 个相同盒子的放球方案数，故是第二类斯特林数 $\left\{ {n \atop k} \right\}$，对 k 求和就得到所有的划分个数，也就是等价关系的个数. 因此 $\sum_{k=1}^{n} \left\{ {n \atop k} \right\}$ 是 A 上的等价关系个数.

(3) 从 A 到 B 的函数有 m^n 个，而一个单射函数对应于从 m 个元素中选 n 个元素的一种排列，因此单射函数有 $P(m, n) = m(m-1)\cdots(m-n+1)$ 个. 对于满射函数，将 m 个函数值看成 m 个不同的盒子，将 n 个自变量看成 n 个不同的球，将它们恰好放入 m 个不同的盒子，放球的方法数就是满射函数的个数，即 $m!\left\{ {n \atop m} \right\}$. 双射仅当 $m = n$ 时成立，这时 $P(n, n) = n!\left\{ {n \atop n} \right\} = n!$，因此恰好有 $n!$ 个双射函数. □

回顾例 ?? ,根据包含排斥原理,满射函数个数是 $\sum_{r=0}^{m}(-1)^r C(m,r)(m-r)^n$. 可以得到下面的恒等式.

命题 1.6.5

$$\sum_{r=0}^{m}(-1)^r C(m,r)(m-r)^n = m!\left\{ {n \atop m} \right\}.$$

回顾例 ?? ,根据包含排斥原理,满射函数个数是 $\sum_{r=0}^{m}(-1)^r C(m,r)(m-r)^n$. 可以得到下面的恒等式.

命题 1.6.5

$$\sum_{r=0}^{m}(-1)^r C(m,r)(m-r)^n = m!\left\{ n \atop m \right\}.$$

证明.

考虑 $(\mathrm{e}^x-1)^m$ 的两种展开方法. 首先,类似于指数生成函数展开得到:

$$(\mathrm{e}^x-1)^m = \left(x+\frac{x^2}{2}+\cdots\right)^m = \sum_{n=0}^{+\infty} a_n \frac{x^n}{n!},$$

其中 $\frac{x^n}{n!}$ 项的系数是 $a_n = \sum \frac{n!}{n_1!n_2!\cdots n_m!}$,这里的求和是对满足方程 $n_1+n_2+\cdots+n_m=n$ 的一切正整数解来求. 根据命题1.6.4(4) 有

$$a_n = \begin{cases} 0, & n<m, \\ \sum \frac{n!}{n_1!n_2!\cdots n_m!} = \sum \binom{n}{n_1\, n_2 \cdots n_m} = m!\left\{ n \atop m \right\}, & n \geq m. \end{cases}$$

命题1.6.5证明(续)

其次,先按照二项式定理展开,然后再将每个项的 e^{rx} 展开成幂级数,得到:

$$
\begin{aligned}
(\mathrm{e}^x-1)^m &= \binom{m}{0}\mathrm{e}^{mx}-\binom{m}{1}\mathrm{e}^{(m-1)x}+\cdots+(-1)^m\binom{m}{m}\times 1\\
&= \binom{m}{0}\left(1+\frac{m}{1!}x+\frac{m^2}{2!}x^2+\cdots\right)-\\
&\quad \binom{m}{1}\left(1+\frac{m-1}{1!}x+\frac{(m-1)^2}{2!}x^2+\cdots\right)+\cdots+(-1)^m\binom{m}{m}\times 1.
\end{aligned}
$$

上式中 $\frac{x^n}{n!}$ 项的系数是

$$
\begin{aligned}
&\binom{m}{0}m^n-\binom{m}{1}(m-1)^n+\binom{m}{2}(m-2)^2-\cdots+(-1)^{m-1}\binom{m}{m-1}\times 1^n\\
=&\sum_{r=0}^{m-1}(-1)^r\,C(m,r)(m-r)^n+0=\sum_{r=0}^{m}(-1)^r\,C(m,r)(m-r)^n.
\end{aligned}
$$

于是得到 $\sum_{r=0}^{m}(-1)^r\,C(m,r)(m-r)^n=m!\left\{{n \atop m}\right\}$. □