蘇州大學
SOOCHOW UNIVERSITY

Chap8 Advanced Counting

Jin-Hui Wu

2026-04-16

大纲

- 递推关系的应用与求解 (8.1-8.2)
- 分治算法和递推关系
- 生成函数
- 容斥原理

回顾：递推关系

- 递推关系 (recurrence relation)
  - 将a_n用$a_0, \ldots, a_{n-1}$表示的等式
  - $a_n = a_{n-1} + 2$

回顾：递推关系

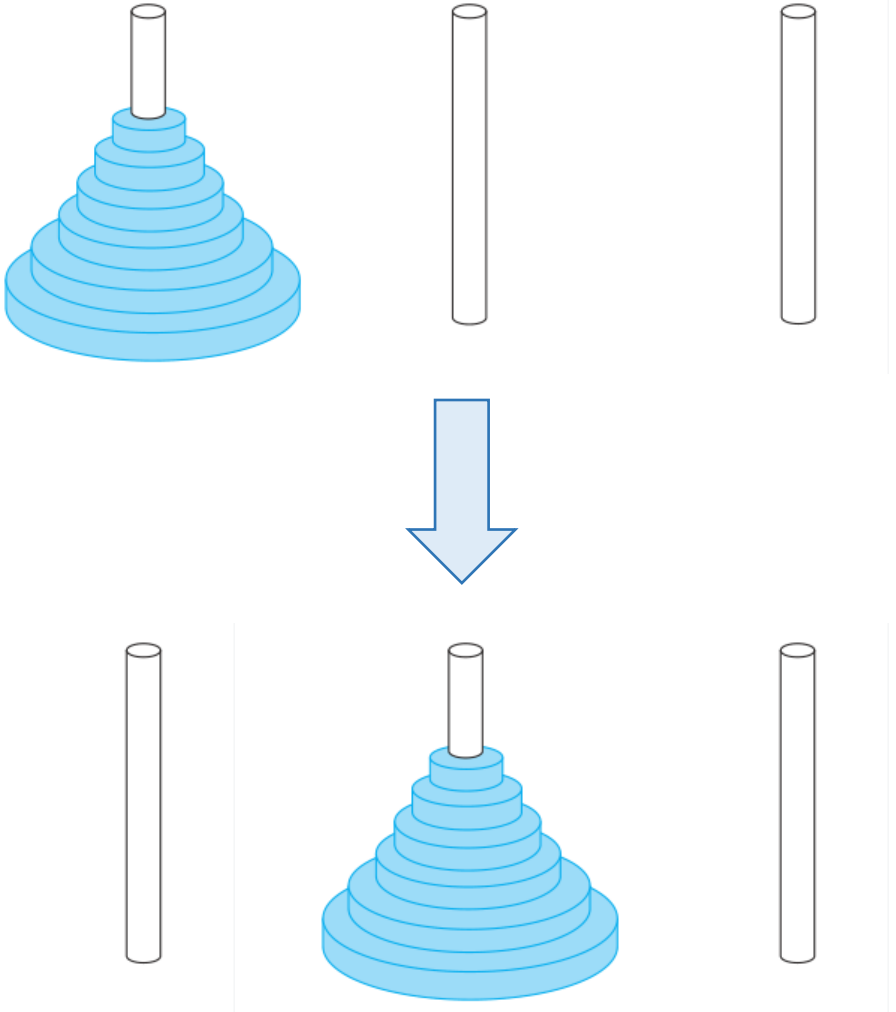
- 递推关系 (recurrence relation)
  - 将a_n用$a_0, \ldots, a_{n-1}$表示的等式
  - $a_n = a_{n-1} + 2$
- 递推关系的解 (solution)
  - 若一个序列满足递推关系，则称其为一个解
  - $a_n = 2n + c$

回顾：递推关系

- 递推关系 (recurrence relation)
  - 将a_n用$a_0, \ldots, a_{n-1}$表示的等式
  - $a_n = a_{n-1} + 2$
- 递推关系的解 (solution)
  - 若一个序列满足递推关系，则称其为一个解
  - $a_n = 2n + c$
- 初始条件 (initial condition)
  - 为递推关系生效前的所有项指定取值
  - $a_0 = 1 \rightarrow a_n = 2n + 1$

汉诺塔

- Tower of Hanoi
  - 每次移动一个盘
  - 大盘不能在小盘上
  - 求移动次数



字符串计数

- 令a_n为不含2个连续0的n位比特串的个数，找出其递推关系和初始条件

拓展：递推关系的求解

- 令a_n为不含2个连续0的n位比特串的个数
  - 递推关系：$a_n = a_{n-1} + a_{n-2}$
  - 初始条件：$a_0 = 1$，$a_1 = 2$

- Theorem 1, P.542

Let c_1 and c_2 be real numbers. Suppose that $r^2 - c_1 r - c_2 = 0$ has two distinct roots r_1 and r_2. Then the sequence $\{a_n\}$ is a solution of the recurrence relation $a_n = c_1 a_{n-1} + c_2 a_{n-2}$ if and only if $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$ for $n = 0, 1, 2, \ldots$, where α_1 and α_2 are constants.

- 常系数线性齐次递推关系 (8.2.2)
  - 重根，无实根，k阶依赖

字符串计数

- 令a_n为包含偶数个0的十进制数字串，求其递推关系和初始条件

拓展：递推关系的求解

- 令a_n为包含偶数个0的十进制数字串
  - 递推关系：$a_n = 8a_{n-1} + 10^{n-1}$
  - 初始条件：$a_0 = 1$

- Theorem 5, P.547

If $\{a_n^{(p)}\}$ is a particular solution of the nonhomogeneous linear recurrence relation with constant coefficients

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k} + F(n),$$

then every solution is of the form $\{a_n^{(p)} + a_n^{(h)}\}$, where $\{a_n^{(h)}\}$ is a solution of the associated homogeneous recurrence relation

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k}.$$

字符串计数

- 令C_n是对$n+1$个数的乘积$x_0 \cdot x_1 \cdot \ldots \cdot x_n$加括号来规定乘法的次序的方式数，求$C_n$的递推关系和初始条件
  - C_n被称作Catalan数

拓展：递推关系的求解

- 令C_n是对$n+1$个数的乘积$x_0 \cdot x_1 \cdot \ldots \cdot x_n$加括号来规定乘法的次序的方式数
  - 递推关系：$C_n = \sum_{j=0}^{n-1} C_j C_{n-j-1}$
  - 初始条件：$C_0 = 1$
  - 证明：$C_n = \frac{1}{n+1}\binom{2n}{n}$
    - 广义二项式定理：$(x+y)^\alpha = \sum_{k=0}^{\infty}\binom{\alpha}{k} x^{\alpha-k}y^k$
    - $\binom{\alpha}{k} = \frac{\alpha(\alpha-1)\cdots(\alpha-k+1)}{k!}$

大纲

- 递推关系的应用与求解
- 分治算法和递推关系 (8.3)
- 生成函数
- 容斥原理

分治算法

- 分治算法 (divide-and-conquer algorithm)
  - 将原问题拆分为若干个相同且规模更小的问题
  - 小规模问题的解可用于得到原问题解

分治算法

- 分治算法 (divide-and-conquer algorithm)
  - 将原问题拆分为若干个相同且规模更小的问题
  - 小规模问题的解可用于得到原问题解

- 分治递推关系
  - 将规模为n的问题拆分为a个规模为n/b的子问题
  - 由子问题得到原问题解需$g(n)$的计算复杂度
  - 规模为n的问题的计算复杂度为

$$f(n) = af\left(\frac{n}{b}\right) + g(n)$$

分支算法：二分搜索

❑ 二分搜索

procedure binary search(x: integer, $a_1, a_2, \ldots, a_n$: increasing integers)
$i := 1$ {i is the left endpoint of interval}
$j := n$ {j is right endpoint of interval}
while $i < j$
    $m := \lfloor (i + j)/2 \rfloor$
    if $x > a_m$ then $i := m + 1$
    else $j := m$
if $x = a_i$ **then** *location* $:= i$
else *location* $:= 0$
return *location*{location is the subscript i of the term a_i equal to x, or 0 if x is not found}

分支算法：二分搜索

❑ 二分搜索

procedure binary search(x: integer, $a_1, a_2, \ldots, a_n$: increasing integers)
$i := 1$ {i is the left endpoint of interval}
$j := n$ {j is right endpoint of interval}
while $i < j$
  $m := \lfloor (i + j)/2 \rfloor$
  if $x > a_m$ then $i := m + 1$
  else $j := m$
if $x = a_i$ **then** *location* $:= i$
else *location* $:= 0$
return *location*{location is the subscript i of the term a_i equal to x, or 0 if x is not found}

1个规模减半的子问题

分支算法：二分搜索

❑ 二分搜索

procedure binary search(x: integer, $a_1, a_2, \ldots, a_n$: increasing integers)
$i := 1$ {i is the left endpoint of interval}
$j := n$ {j is right endpoint of interval}
while $i < j$
  $m := \lfloor (i + j)/2 \rfloor$
  if $x > a_m$ then $i := m + 1$
  else $j := m$
if $x = a_i$ **then** *location* $:= i$
else *location* $:= 0$
return *location*{location is the subscript i of the term a_i equal to x, or 0 if x is not found}

1个规模减半的子问题

2次比较操作

分支算法：二分搜索

❑ 二分搜索

procedure binary search(x: integer, $a_1, a_2, \ldots, a_n$: increasing integers)
$i := 1$ {i is the left endpoint of interval}
$j := n$ {j is right endpoint of interval}
while $i < j$
  $m := \lfloor (i + j)/2 \rfloor$
  if $x > a_m$ then $i := m + 1$
  else $j := m$
if $x = a_i$ **then** *location* $:= i$
else *location* $:= 0$
return *location*{location is the subscript i of the term a_i equal to x, or 0 if x is not found}

1个规模减半的子问题

2次比较操作

❑ 比较次数满足：$f(n) = f\left(\frac{n}{2}\right) + 2$

分治算法：归并排序

❑ 归并排序

procedure *mergesort*($L = a_1, a_2,\ldots,a_n$)
if $n > 1$ **then**
  $m := \lfloor n/2 \rfloor$
  $L_1 := a_1, a_2,\ldots,a_m$
  $L_2 := a_{m+1}, a_{m+2},\ldots,a_n$
  L := *merge*(*mergesort*(L_1), *mergesort*(L_2))
{L is now sorted into elements in increasing order}

procedure *merge*(L_1, L_2 :sorted lists)
L := empty list
while L_1 and L_2 are both nonempty
  remove smaller of first elements of L_1 and L_2 from its list;
  put at the left end of L
  if this removal makes one list empty
    then remove all elements from the other list and append them to L
return L {L is the merged list with the elements in increasing order}

分治算法：归并排序

❑ 归并排序

procedure *mergesort*($L = a_1, a_2,\ldots,a_n$)

if $n > 1$ **then**

$m := \lfloor n/2 \rfloor$

$L_1 := a_1, a_2,\ldots,a_m$

$L_2 := a_{m+1}, a_{m+2},\ldots,a_n$

$L := merge(mergesort(L_1), mergesort(L_2))$

{L is now sorted into elements in increasing order}

2个规模减半的子问题

procedure *merge*(L_1, L_2 :sorted lists)

L := empty list

while L_1 and L_2 are both nonempty

remove smaller of first elements of L_1 and L_2 from its list;

put at the left end of L

if this removal makes one list empty

then remove all elements from the other list and append them to L

return L {L is the merged list with the elements in increasing order}

分治算法：归并排序

❑ 归并排序

procedure *mergesort*($L = a_1, a_2,\ldots,a_n$)

if $n > 1$ **then**

$m := \lfloor n/2 \rfloor$

$L_1 := a_1, a_2,\ldots,a_m$

$L_2 := a_{m+1}, a_{m+2},\ldots,a_n$

$L :=$ *merge*(*mergesort*(L_1), *mergesort*(L_2))

{L is now sorted into elements in increasing order}

2个规模减半的子问题

procedure *merge*(L_1, L_2 :sorted lists)

$L :=$ empty list

while L_1 and L_2 are both nonempty

remove smaller of first elements of L_1 and L_2 from its list;

put at the left end of L

至多n次比较操作

if this removal makes one list empty

then remove all elements from the other list and append them to L

return L {L is the merged list with the elements in increasing order}

分治算法：归并排序

❑ 归并排序

procedure *mergesort*($L = a_1, a_2,\ldots,a_n$)
if $n > 1$ **then**
  $m := \lfloor n/2 \rfloor$
  $L_1 := a_1, a_2,\ldots,a_m$
  $L_2 := a_{m+1}, a_{m+2},\ldots,a_n$
  $L := merge(mergesort(L_1), mergesort(L_2))$
{L is now sorted into elements in increasing order}

2个规模减半的子问题

procedure *merge*(L_1, L_2 :sorted lists)
L := empty list
while L_1 and L_2 are both nonempty
  remove smaller of first elements of L_1 and L_2 from its list;
  put at the left end of L
  if this removal makes one list empty
    then remove all elements from the other list and append them to L
return L {L is the merged list with the elements in increasing order}

至多n次比较操作

❑ 比较次数满足：$f(n) = 2f\left(\frac{n}{2}\right) + n$

拓展：主定理

- 主定理 (master theorem)
  - f为增函数且满足

$$f(n) = af\left(\frac{n}{b}\right) + cn^d,$$

其中$a \geq 1, b \geq 2$为整数，$c > 0, d \geq 0$为实数，则

  - $a < b^d$时，$f(n) = O(n^d)$
  - $a = b^d$时，$f(n) = O(n^d \log n)$
  - $a > b^d$时，$f(n) = O(n^{\log_b a})$

拓展：主定理的应用

- 计算$f(n)$的上界
  - 二分搜索的比较次数满足：$f(n) = f\left(\frac{n}{2}\right) + 2$

- f为增函数且满足

$$f(n) = af\left(\frac{n}{b}\right) + cn^d,$$

其中$a \geq 1, b \geq 2$为整数，$c > 0, d \geq 0$为实数，则

- $a < b^d$时，$f(n) = O(n^d)$
- $a = b^d$时，$f(n) = O(n^d \log n)$
- $a > b^d$时，$f(n) = O(n^{\log_b a})$

拓展：主定理的应用

- 计算$f(n)$的上界
  - 归并排序比较次数满足：$f(n) = 2f\left(\frac{n}{2}\right) + n$

- f为增函数且满足

$$f(n) = af\left(\frac{n}{b}\right) + cn^d,$$

其中$a \geq 1, b \geq 2$为整数，$c > 0, d \geq 0$为实数，则

- $a < b^d$时，$f(n) = O(n^d)$
- $a = b^d$时，$f(n) = O(n^d \log n)$
- $a > b^d$时，$f(n) = O(n^{\log_b a})$

最大子数组

- 最大子数组 (maximum subarray) 问题
  - 在整数数组 nums 中，找到元素和最大的非空连续子数组，并返回最大和

  - 输入：nums = [-2, 1, -3, 4, -1, 2, 1, -5, 4]
  - 输出：6

最大子数组

- 暴力算法
  - 思路：枚举所有子数组，求和，记录最大值

```python
def maxSubArray(nums):
    n = len(nums)
    max_sum = float('-inf')
    for i in range(n):
        for j in range(i, n):
            current_sum = sum(nums[i:j+1])
            max_sum = max(max_sum, current_sum)
    return max_sum
```

最大子数组

- 暴力算法
  - 思路：枚举所有子数组，求和，记录最大值

```python
def maxSubArray(nums):
    n = len(nums)
    max_sum = float('-inf')
    for i in range(n):
        for j in range(i, n):
            current_sum = sum(nums[i:j+1])
            max_sum = max(max_sum, current_sum)
    return max_sum
```

i：$0 \to n-1$
j：$i \to n-1$
求和：$j-i$ 次

  - 求和次数：$O(n^3)$

最大子数组

- 分治算法
  - 思路：讨论最大子数组的位置
    - 在左半部分
    - 在右半部分
    - 横跨两部分
  - 例
    - [-2, 1, -3, 4, -1, 2, 1, -5, 4]

最大子数组

- 分治算法

```python
def maxSubArray(nums, left, right):
    if left == right:
        return nums[left]

    mid = (left + right) // 2
    left_max = maxSubArray(nums, left, mid)
    right_max = maxSubArray(nums, mid + 1, right)

    left_sum = float('-inf')
    s = 0
    for i in range(mid, left - 1, -1):
        s += nums[i]
        left_sum = max(left_sum, s)
    right_sum = float('-inf')
    s = 0
    for i in range(mid + 1, right + 1):
        s += nums[i]
        right_sum = max(right_sum, s)
    cross_max = left_sum + right_sum

    return max(left_max, right_max, cross_max)
```

最大子数组

- 分治算法

```python
def maxSubArray(nums, left, right):
    if left == right:
        return nums[left]

    mid = (left + right) // 2
    left_max = maxSubArray(nums, left, mid)
    right_max = maxSubArray(nums, mid + 1, right)

    left_sum = float('-inf')
    s = 0
    for i in range(mid, left - 1, -1):
        s += nums[i]
        left_sum = max(left_sum, s)
    right_sum = float('-inf')
    s = 0
    for i in range(mid + 1, right + 1):
        s += nums[i]
        right_sum = max(right_sum, s)
    cross_max = left_sum + right_sum

    return max(left_max, right_max, cross_max)
```

2个规模减半的子问题

最大子数组

- 分治算法
  - 求和次数满足
  - $f(n) = 2f\left(\frac{n}{2}\right) + n$
  - 求和次数：
  - $O(n \log n)$

```python
def maxSubArray(nums, left, right):
    if left == right:
        return nums[left]

    mid = (left + right) // 2
    left_max = maxSubArray(nums, left, mid)
    right_max = maxSubArray(nums, mid + 1, right)

    left_sum = float('-inf')
    s = 0
    for i in range(mid, left - 1, -1):
        s += nums[i]
        left_sum = max(left_sum, s)
    right_sum = float('-inf')
    s = 0
    for i in range(mid + 1, right + 1):
        s += nums[i]
        right_sum = max(right_sum, s)
    cross_max = left_sum + right_sum

    return max(left_max, right_max, cross_max)
```

2个规模减半的子问题

$\Theta(n)$次求和操作

最大子数组

- 动态规划 (dynamic programming)
  - 思路：记录以i位为结尾的最大子数组和$dp[i]$
    - $dp[i] = nums[i]$
    - $dp[i] = dp[i-1] + nums[i]$
  - 例
    - [-2, 1, -3, 4, -1, 2, 1, -5, 4]

最大子数组

- 动态规划 (dynamic programming)
  - 思路：记录以i位为结尾的最大子数组和$dp[i]$
    - $dp[i] = nums[i]$
    - $dp[i] = dp[i-1] + nums[i]$

```python
def maxSubArray(nums):
    current_max = global_max = nums[0]
    for i in range(1, len(nums)):
        current_max = max(nums[i], current_max + nums[i])
        global_max = max(global_max, current_max)
    return global_max
```

最大子数组

- 动态规划 (dynamic programming)
  - 思路：记录以i位为结尾的最大子数组和$dp[i]$
    - $dp[i] = nums[i]$
    - $dp[i] = dp[i-1] + nums[i]$

```
1 def maxSubArray(nums):
2     current_max = global_max = nums[0]
3     for i in range(1, len(nums)):
4         current_max = max(nums[i], current_max + nums[i])
5         global_max = max(global_max, current_max)
6     return global_max
```

i：$0 \to n-1$
求和：1次

  - 求和次数：$O(n)$

大纲

- 递推关系的应用与求解
- 分治算法和递推关系
- 生成函数 (8.4)
- 容斥原理

生成函数

- 生成函数 (generating function)
  - $\{a_k\}_{k=0}^{\infty}$是实数序列
  - 普通生成函数

$$G(x) = \sum_{k=0}^{\infty} a_k x^k$$

    - 将序列作为多项式系数
  - 常用的还有指数生成函数

拓展：生成函数与计数

- 求$e_1 + e_2 + e_3 = 7$的解的个数，其中：
  - $1 \leq e_1 \leq 6$
  - e_2为非负偶数
  - e_3为非负的三的倍数

拓展：生成函数与递推关系

- 令a_n为包含偶数个0的十进制数字串
  - 递推关系：$a_n = 8a_{n-1} + 10^{n-1}$
  - 初始条件：$a_0 = 1$

- 用生成函数求解a_n通项公式

拓展：生成函数与组合恒等式

- 用生成函数证明

$$\sum_{k=0}^{n} \binom{n}{k}^2 = \binom{2n}{n}$$

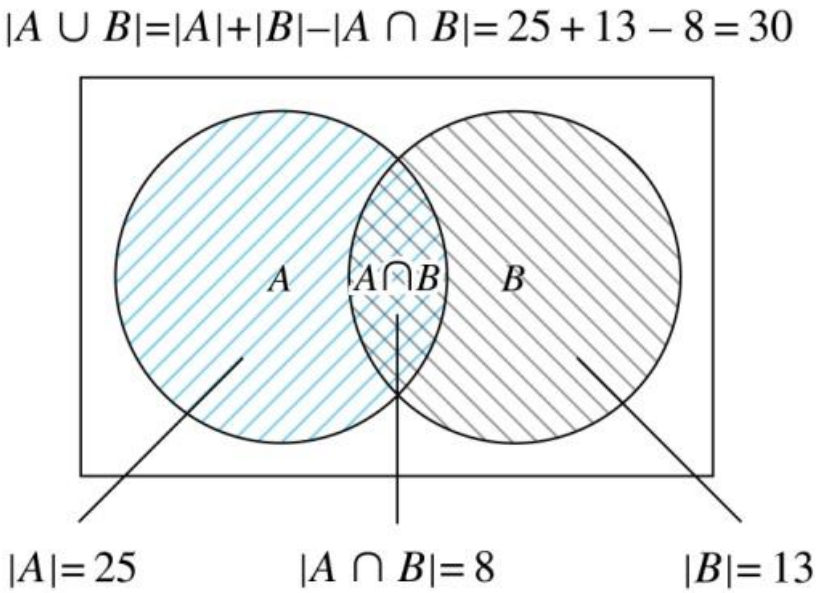
大纲

- 递推关系的应用与求解
- 分治算法和递推关系
- 生成函数
- 容斥原理 (8.5-8.6)

容斥原理

- 容斥原理 (Principle of Inclusion-Exclusion)
  - 两个集合

$$|A \cup B| = |A| + |B| - |A \cap B|$$



容斥原理

- 容斥原理 (Principle of Inclusion-Exclusion)
  - 三个集合

$$
\begin{aligned}
|A \cup B \cup C| = {} & |A| + |B| + |C| \\
& - |A \cap B| - |A \cap C| - |B \cap C| \\
& + |A \cap B \cap C|
\end{aligned}
$$

容斥原理

- 容斥原理 (Principle of Inclusion-Exclusion)
  - n个集合

$$|A_1 \cup A_2 \cup \cdots \cup A_n| = \sum_{1 \le i \le n} |A_i| - \sum_{1 \le i \le j \le n} |A_i \cap A_j| + \sum_{1 \le i \le j \le k \le n} |A_i \cap A_j \cap A_k| - \ldots + (-1)^{n+1} |A_1 \cap A_2 \cap \ldots \cap A_n|$$

例：满射个数

- 6元集到3元集的满射有____个

例：满射个数

- 6元集到3元集的满射有____个

- m元集到n元集的满射个数为

$$n^m - C(n,1)(n-1)^m + C(n,2)(n-2)^m - \cdots + (-1)^{n-1}C(n,n-1)\cdot 1^m$$

例：错排

- 错排 (derangements)
  - 没有一个物体在其初始位置上的排列
  - 例
    - 21453 是 12345 的一个错排
    - 21543 不是 12345 的错排

例：错排

- 错排 (derangements)
  - 没有一个物体在其初始位置上的排列
  - n个元素的错排数为

$$D_n = n!\left[1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \cdots + (-1)^n \frac{1}{n!}\right]$$

例：错排

- 4名学生同时参加英语和德语面试，要求每门科目只能同时面试1人，则共有_____种不同的面试次序

总结

- 递推关系的应用
  - 汉诺塔、字符串计数等，会列递推关系
- 分治算法
  - 二分搜索、归并排序、最大子数组和等
  - 主定理
- 生成函数
  - 计数、求解递推关系、证明组合恒等式
- 容斥原理
  - 3个集合的容斥原理
  - 满射个数、错排数等