算法分析

陈卫东

chenwd@scnu.edu.cn

华南师范大学计算机学院

1. 算法分析的基本方法

- 1.1. 算法及其特性
- 1.2. 衡量一个算法性能的好坏的标准
- 1.3. 算法的时间和空间复杂度
- 1.4. 算法分析(Algorithm Analysis)
  - 1.4.1. 分析算法时间复杂度的基本步骤
  - 1.4.2. 算法时间复杂度的有关概念
  - 1.4.3. 分析、求解算法复杂度的方法
- 1.5. 最优算法(Optimal Algorithm)

1.1. 算法及其特性

- 算法（Algorithm）：

  算法就是一组有穷的规则，它们规定了解决某一特定类型问题的一系列运算。

算法的描述形式：

(1) 程序（形式化描述）
(2) 自然语言描述（非形式化描述）
(3) 伪语言描述（非形式化描述）

1.1. 算法及其特性

- 算法有五个特性：
  - 确定性
  - 能行性
  - 有穷性
  - 输入
  - 输出

注： 有穷性是算法与程序的主要区别

[例] 选择排序算法

(1) 选择排序算法的自然语言描述形式

对数组***A***[1…*n*]中元素进行非降次序的排序思路如下：

首先找到最小元素，将其存放到***A***[1]中，然后再找剩下的*n*-1个元素中的最小元素，将其存放在***A***[2]中，重复这个过程直到找到第二大元素，将其存放在***A***[*n*-1]中。

(2) 选择排序算法的伪语言描述形式

输入：n 个元素的数组 $A[1\cdots n]$。

输出：按非降序排列的数组 $A[1\cdots n]$。

1. **for** $i \leftarrow 1$ **to** $n-1$
2. $\quad k \leftarrow i$
3. $\quad$ **for** $j \leftarrow i+1$ **to** n $\quad$ {查找第 i 小的元素}
4. $\quad\quad$ **if** $A[j] < A[k]$ **then** $k \leftarrow j$
5. $\quad$ **end for**
6. $\quad$ **if** $k \neq i$ **then** 交换 $A[i]$与 $A[k]$
7. **end for**

1.2. 衡量算法性能的标准

- 衡量算法性能通常有如下标准：
  - 正确性
  - 易读性
  - 健壮性
  - 算法的时间和空间性能：高效率和低存储空间

注： 1. 我们主要讨论算法的时间和空间性能，并以此作为衡量算法性能的重要标准，而且主要侧重于时间方面。

2. 算法分析是对算法运行中所耗费的时间和空间的定量化分析。

1.3.算法的时间和空间复杂度

- 时间复杂度（Time Complexity）
  - 算法的时间复杂度：在算法运行期间所花费的时间。
  - 通常用渐近形式表示

  比如，$T(n)= O(n^2)$、 $\Omega(n^2)$ 或 $\Theta(n^2)$

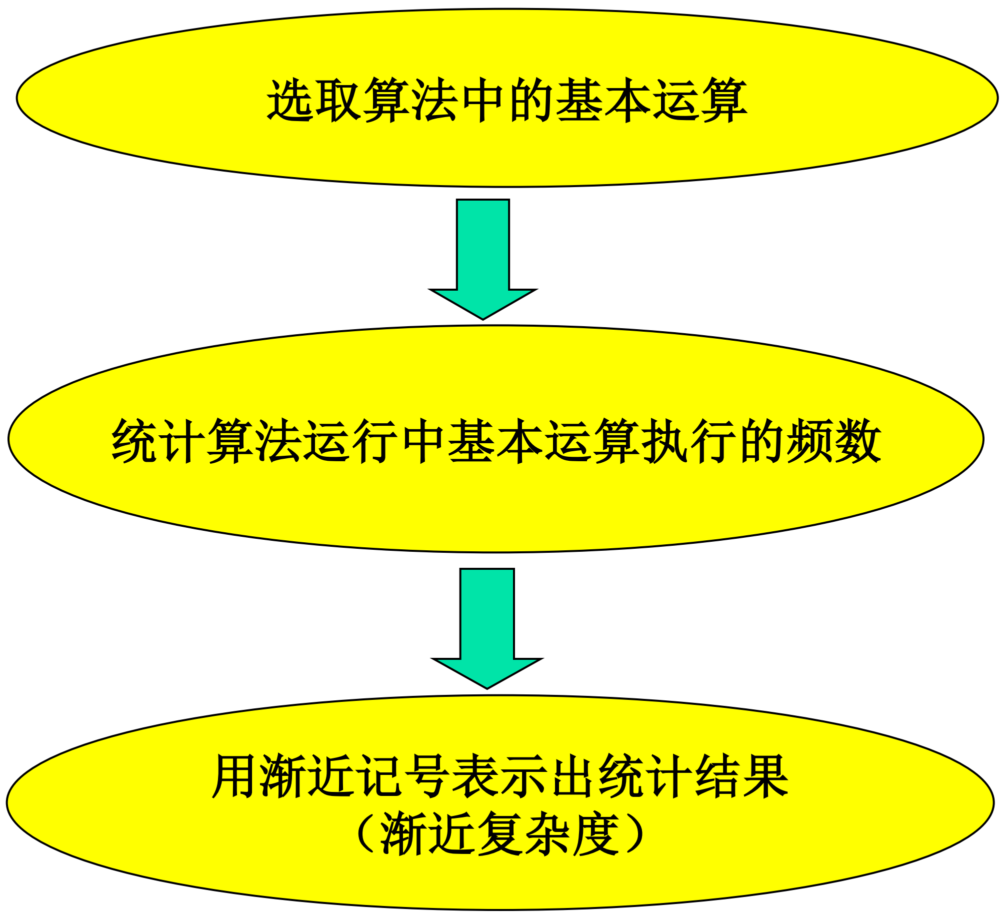
- 空间复杂度（Space Complexity）
  - 算法的空间复杂度：算法运行期间所需要的内存空间。一般是指，除开容纳输入数据之外的附加空间（Auxiliary Space/Work Space）。
  - 通常用渐近形式表示

  比如，$S(n)= O(n^2)$、$\Omega(n^2)$或 $\Theta(n^2)$

1.4. 算法分析(Algorithm Analysis)

- 算法分析 ——对于算法的时间和空间复杂度进行定量分析。
- 算法分析涉及到如下内容：
  - 分析算法时间复杂度的基本步骤
  - 算法时间复杂度的有关概念
  - 分析、求解算法复杂度的基本方法

1.4.1. 分析时间复杂度的基本步骤



[例] 选择排序算法

输入：n 个元素的数组 $A[1\cdots n]$。

输出：按非降序排列的数组 $A[1\cdots n]$。

```
for i ← 1 to n − 1
    k ← i
    for j ← i + 1 to n        {查找第 i 小的元素}
        if A[j] < A[k] then k ← j
    end for
    if k ≠ i then  交换 A[i]与 A[k]
end for
```

算法分析：

- 算法中运算（操作）包括：

  循环变量 ***i, j*** 的比较、赋值等运算；

  元素的比较、元素的交换(赋值)运算等。

  ——选取元素的比较运算作为基本运算。

- ——算法执行期间基本运算总的执行次数为：

  $(n-1)+(n-2)+\ldots+1=n(n-1)/2$

- ——用渐近记号表示基本运算总的执行次数：

  $n(n-1)/2=O(n^2)$

1.4.1. 分析时间复杂度的基本步骤

一、选取算法中的基本运算(basic operation)

对算法的分析必须脱离具体的计算机结构和程序设计语言。因此，比较两个算法的好坏，看其中所需的运算时间的长短是由算法所需的运算次数决定的。任何一个算法都可能有几种运算，因此，我们抓住其中影响算法运行时间最大的运算作为基本运算。

- 在表A中寻找某元素X的问题中，把X和表A中元素的比较作为基本运算；
- 在两实数矩阵相乘的问题中，把两实数相乘作为基本运算。

1.4.1. 分析时间复杂度的基本步骤

二、表示出在算法运行中基本运算执行的总频数

同一个问题对不同的输入，基本运算的次数亦可能不同。因此，我们引进问题大小（即规模，size）的概念。例如，在一个姓名表中寻找给定的Z的问题，问题的大小可用表中姓名的数目表示。对于两个实数矩阵相乘的问题，其大小可用矩阵的阶来表示。而对于遍历一棵二叉树的问题，其大小是用树中结点数来表示等等。这样，一个算法的基本运算的次数就可用问题的大小n的函数$f(n)$来表示。

1.4.1. 分析时间复杂度的基本步骤

三、渐近时间复杂度（asymptotic time complexity）

在实际中精确地求一个算法的基本运算次数$f(n)$等于多少，往往不容易，甚至有时根本不可能，有些即使求出结果很长，很繁琐，不易比较，需要简化。这时候我们可以不精确地估计$f(n)$。此外，我们分析算法的时间目的主要在于，能区分不同算法的优劣，在n很小时候，差别不大，随着n的逐渐增大，差别越来越大，是个极限行为。基于上述原因，引进下面渐近表示的方法。

复杂度渐近表示可以将简洁地表示出复杂度的数量级别。

渐近表示的记号—O

- O-记号

设$f(n)$和$g(n)$均是从自然数集到非负实数集上的函数。如果存在常数$c>0$和一个自然数n_0，使得对于任意的$n \geq n_0$，均有

$$f(n) \leq cg(n),$$

则 $f(n)= O(g(n))$

  - 含义：阶至多为$g(n)$的函数；上限
  - 读法：O$(g(n))$读作“大Oh $g(n)$”
  - 显然，若有$\lim_{n\to\infty} f(n)/g(n)<\infty$,则有$f(n)=O(g(n))$。

渐近表示的记号—O

- 【例】$f(n)=10n^2+20n$

因为, $\frac{f(n)}{n^2}=\frac{10n^2+20n}{n^2}\to 10(n\to\infty)$

所以, $f(n)= O(n^2)$。

当然, $f(n)= O(n^3)$, $f(n)= O(n^4)$…也都是成立的。

应该用哪一个更好呢? 为什么?

渐近表示的记号—Ω

- **Ω-记号**

设$f(n)$和$g(n)$均是从自然数集到非负实数集上的函数。如果存在常数$c>0$和一个自然数n_0，使得对于任意的$n \geq n_0$，均有

$$f(n) \geq cg(n),$$

则 $f(n)=\Omega(g(n))$

- 含义：阶至少为$g(n)$的函数；下限
- 读法：$\Omega(g(n))$读作"omega $g(n)$"
- 显然，若有$\lim_{n\to\infty} f(n)/g(n) \neq 0$,则有$f(n)=\Omega(g(n))$。

渐近表示的记号— Ω

- [例] $f(n)=10n^2+20n$

  显然，n^2, n, $\log n$等都是下界，最大下界是n^2。

  所以，$f(n)=\Omega(n^2)$。

渐近表示的记号—Θ

- **Θ-记号**

设$f(n)$和$g(n)$均是从自然数集到非负实数集上的函数。如果存在一个自然数n_0和两个正常数c_1，c_2，使得对于任意的$n \geq n_0$，均有

$$c_1 g(n) \leq f(n) \leq c_2\, g(n),$$

则　$f(n)= \Theta(g(n))$

- 含义：阶恰好为$g(n)$的函数；
- 读法：$\Theta(g(n))$读作"theta g(n)"
- 显然，若有$\lim_{n\to\infty} f(n)/g(n)=c>0$,则有$f(n)= \Theta(g(n))$。

渐近表示的记号——Θ

$$\frac{f(n)}{g(n)} \to a(n \to \infty)(a \geq 0, a \neq \infty),\text{则}f(n) = O(g(n))$$

$$\frac{f(n)}{g(n)} \to a(n \to \infty)(0 < a \leq \infty),\text{则}f(n) = \Omega(g(n))$$

$$\frac{f(n)}{g(n)} \to a(n \to \infty)(0 < a < \infty),\text{则}f(n) = \Theta(g(n))$$

渐近表示— Examples

- [例1]　设$f(n)=10n^2+20n$。则有
  - $f(n)=O(n^2)$
  - $f(n)=\Omega(n^2)$
  - $f(n)=\Theta(n^2)$
- [例2]　设$f(n)=a_k n^k+a_{k-1}n^{k-1}+\ldots+a_1 n+a_0$,$(a_k>0)$。

$$\frac{f(n)}{n^k} \to a_k (n \to \infty)$$

则有

  - $f(n)=O(n^k)$
  - $f(n)=\Omega(n^k)$；
  - $f(n)=\Theta(n^k)$

由此可见，渐近表示可以简洁地表示出复杂度的数量级别。

1.4.2. 算法时间复杂度的有关概念

对于算法的时间复杂度，通常从分平均、最坏、最好几种情形来衡量，尤其是前两种。

- 算法的平均复杂性
- 算法的最坏复杂性
- 算法的最好复杂性

1.4.2. 算法时间复杂度的有关概念

- **[例] 检索问题的顺序查找算法**

- ***A*=** $\boxed{a_1 \ \ a_2 \ \ a_3 \ldots\ldots \ \ a_n}$ ，*元素x是否在数组A中？*

- 以元素的比较作为基本操作。考虑成功检索的情况：
  - 最好情况下的时间复杂度： $1=\Theta(1)$
  - 最坏情况下的时间复杂度： $n=\Theta(n)$
  - 等概率下，平均情况时间复杂度： $(1+2+\ldots+n)/n=(n+1)/2=\Theta(n)$

- **[例] 直接插入排序算法**

- ***A*=** $\boxed{a_1\ \ a_2\ \ a_3 \ldots\ldots\ \ a_n}$ ，*对A进行非降序排序。*

$$\boxed{a_1\ \ a_2\ \ a_3 \ldots\ \ a_{i-1}\ \ a_i\ \ a_{i+1}\ \ldots\ \ a_n}$$

往前找位置插入

- 以元素的比较作为基本操作。
  - 最好情况时间复杂度： $n-1=\Theta(n)$
  - 最坏情况时间复杂度： $1+2+\ldots+n-1=n(n-1)/2=\Theta(n^2)$
  - 等概率下，平均情况时间复杂度：$(1+2+\ldots+n-1)/2=n(n-1)/4=\Theta(n^2)$

1.4.3. 分析、求解算法复杂度的方法

- 分析算法复杂度的方法
  - 根据循环来统计基本操作的次数
  - 利用递归关系来求基本操作的次数
  - 用平摊的办法来统计基本操作的次数（Amortized Analysis）

1.4.3. 分析、求解算法复杂度的方法

[例] 根据循环来统计基本操作的次数

Input: n=2^k（k为某个正整数）.
Output: count（step 4的执行次数）

1.count←0
2.**while** n ≥1
3.   **for** j←1 **to** n
4.   count ←count+1
5.   **end for**
6.   n←n/2
7.**end while**
8.**return** count

[算法分析]

- 基本操作：Step 4
- 基本操作的次数：

$$2^k+2^{k-1}+ \ldots + 2+1=2^{k+1}-1=2n-1$$
$$= \Theta(n)$$

即， $\Sigma_{0\le j\le k}\, n/2^j =2n-1= \Theta(n)$

1.4.3. 分析、求解算法复杂度的方法

[例] **利用递归关系来求基本操作的次数**

求Fibonacci数列的第n项。该数列的定义为：

$F_0= F_1=1,\ F_i= F_{i-1} + F_{i-2}，\ i \geq 2$。

由此定义自然地导出如下递归算法。

```
int  F(int n)
{ //输入非负整数n
   if (n==0||n==1) return 1;
   return F(n-1)+F(n-2);
}
```

[算法分析]

- 基本操作：“+”
- 记F(n) 中基本操作的次数为T(n)，则T(n)满足如下递归方程：

$$T(n)=T(n-1)+T(n-2)+1,\ n>1;$$

$$T(0)=T(1)=0。$$

解此递归方程即可得T(n)。

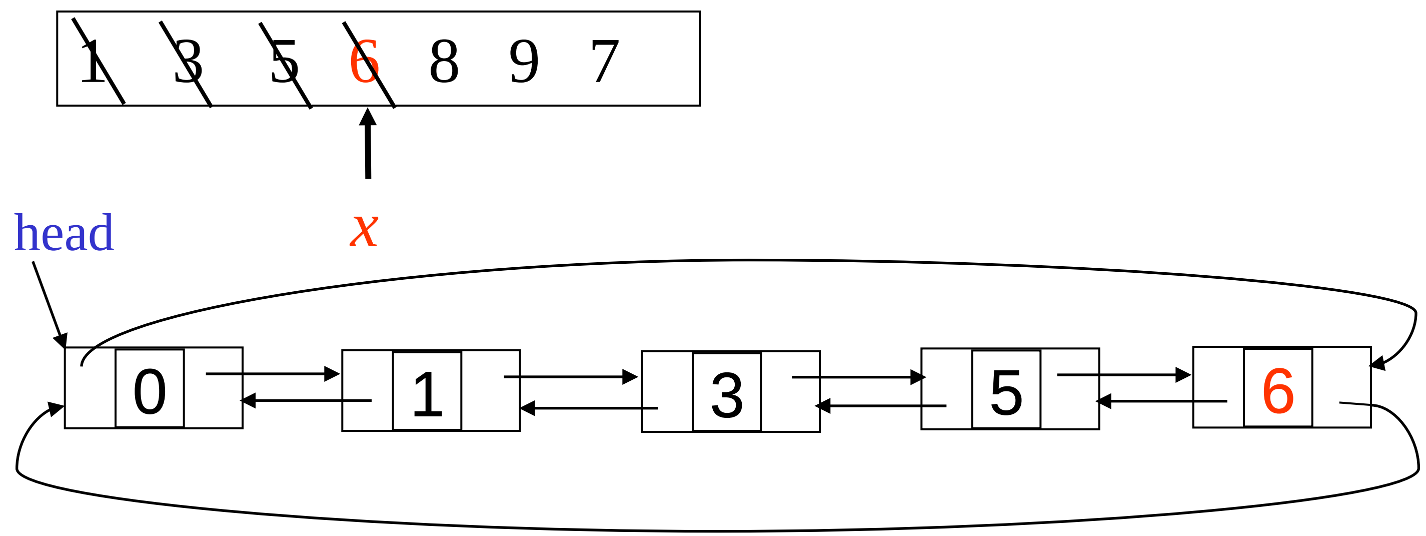
平摊分析方法
- 会计方法
- 聚合方法
- 势能方法

[例] 用平摊方法来统计基本操作的次数（Amortized Analysis）

数组$A[1：n]$初始化为n个正整数的集合

双链表**List**初始化为仅含一个元素0

- **A:**
- **List:**



此时需要删除6前面所有的奇数节点

```
for (j=1;j<=n;j++)
    x=A(j);
    将 x 插入到表List尾
    if x 是偶数 then
        while 表List中x的前面元素为奇数
            在表List中删除x的前面的那个元素
        end while
    end if
end for
```

[算法分析]

- 基本操作：插入，删除
- 基本操作的次数：

$$\begin{cases} \text{插入：} n\text{次} \\ \text{删除：} \le (n-1)(n-1)\text{次} \end{cases} \Longrightarrow T(n)=O(n^2)$$

若采用平摊分析方法，则基本操作的次数为：

$$\begin{cases} \text{插入：} n \text{ 次} \\ \text{删除：} \le (n-1) \text{ 次} \end{cases}$$

即有， $n \le T(n) \le 2n-1$

因此，$T(n)=\Theta(n)$

分析算法复杂度的基本步骤

基本技术：

- 根据循环统计基本操作次数
- 用递归关系统计基本操作次数
- 用平摊方法统计基本操作次数

选取基本操作

统计基本操作频数

计算并简化统计结果

基本概念：

- 平均时间复杂度, 最坏时间复杂度
  最好时间复杂度
- 渐近复杂度：O, Ω, Θ

基本技术：

- 常用求和公式
- 定积分近似求和
- 递归方程求解

1.4.3. 分析、求解算法复杂度的方法

- 数学基础
  - 典型的求和公式
  - 积分近似求和
  - 递归关系

典型的求和公式

$\Sigma_{0\le i\le n} f(i)$

- $\Sigma_{1\le i\le n}\, i=n(n+1)/2=\Theta(n^2)$
- $\Sigma_{1\le i\le n}\, (a+bi)=na+bn(n+1)/2=\Theta(n^2)$
- $\Sigma_{1\le i\le n}\, i^2=n(n+1)(2n+1)/6=\Theta(n^3)$
- $\Sigma_{1\le i\le n}\, i^k=\Theta(n^{k+1})$
- $\Sigma_{0\le i\le n}\, a^i=(1-a^{n+1})/(1-a)=\Theta(a^n)$ $(a\ne 1)$
- $\Sigma_{0\le i\le n}\, ia^i=\Theta(na^n)$ $(a\ne 1)$

$\sum_{0\le i\le n} ia^i = \Theta(na^n)(a\neq 1)$的推导过程如下：

考虑$\sum_{i=0}^{n} x^i = \frac{x^{n+1}-1}{x-1}(x\neq 1)$，记 $g(x)=\frac{x^{n+1}-1}{x-1}(x\neq 1)$

则$\sum_{i=1}^{n} ix^{i-1} = g'(x)(x\neq 1)$

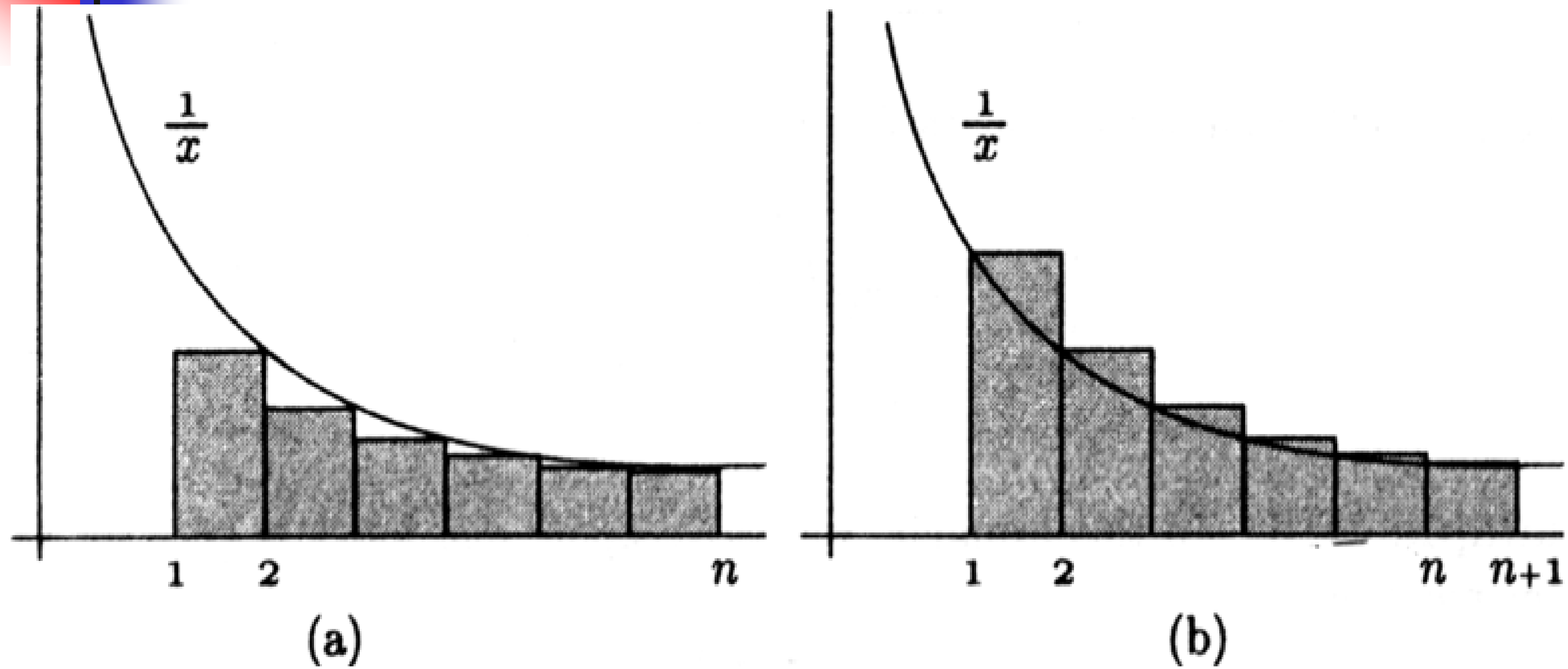
将$x=a$代入则有

$$\sum_{i=1}^{n} ia^{i-1} = g'(a)$$

$$\sum_{i=1}^{n} ia^{i} = ag'(a) = \Theta(na^n)$$

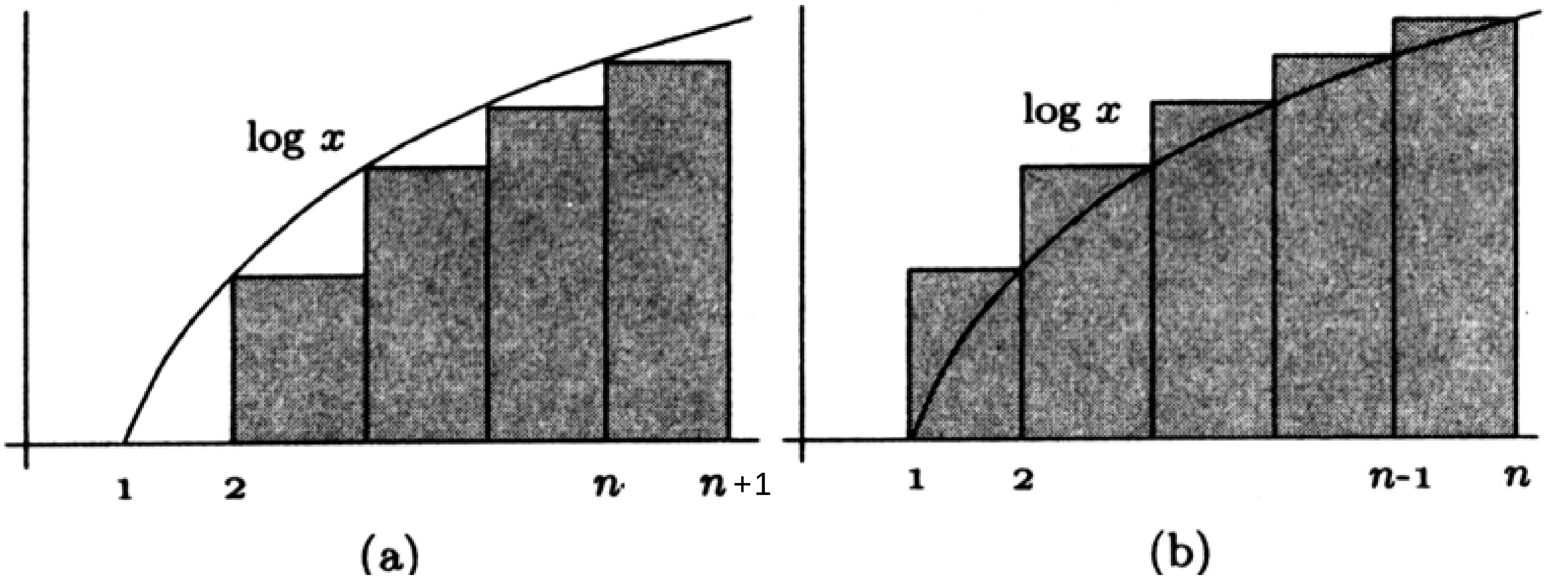
同理可求$\sum_{i=1}^{n} i^2a^i, \sum_{i=1}^{n} i^3a^i$

$$\sum_{i=1}^{n} \frac{1}{i} = ?$$



$$\int_{1}^{n+1} \frac{1}{x} dx \leq \sum_{i=1}^{n} \frac{1}{i} \leq 1 + \int_{1}^{n} \frac{1}{x} dx$$

$$\sum_{i=1}^{n} \log i = ?$$



$$\int_{1}^{n} \log x dx \le \sum_{i=1}^{n} \log i \le \int_{1}^{n+1} \log x dx$$

积分近似求和

- 如果连续函数$f(n)$是单调递减的，则有

$$\int_{m}^{n+1} f(x)dx \le \sum_{m\le j\le n} f(j) \le \int_{m-1}^{n} f(x)dx$$

- 如果连续函数$f(n)$是单调递增的，则有

$$\int_{m-1}^{n} f(x)dx \le \sum_{m\le j\le n} f(j) \le \int_{m}^{n+1} f(x)dx$$

[例1] $\Sigma_{1 \le j \le n} 1/j = \Theta(\log n)$

$$\text{解：} \int_1^{n+1} \frac{1}{x} dx \le \sum_{j=1}^{n} \frac{1}{j} \le 1 + \int_1^{n} \frac{1}{x} dx$$

$$\ln x \Big|_1^{n+1} \le \sum_{j=1}^{n} \frac{1}{j} \le 1 + \ln x \Big|_1^{n}$$

$$\text{则} \ln(n+1) \le \sum_{j=1}^{n} \frac{1}{j} \le 1 + \ln n$$

$$\text{故} \sum_{j=1}^{n} \frac{1}{j} = \Theta(\log n)$$

[例2] $\Sigma_{1\le j\le n}\log j=\Theta(n\log n)$

$$\text{解：}\int_{1}^{n}\log x dx\le\sum_{j=1}^{n}\log j\le\int_{1}^{n+1}\log x dx$$

$$\text{则} n\log n-n\log e+\log e\le\sum_{j=1}^{n}\log j$$

$$\le(n+1)\log(n+1)-(n+1)\log e+\log e$$

$$\text{故}\sum_{j=1}^{n}\log j=\Theta(n\log n)$$

递归关系

(I) 线性齐次递归方程

$$T(n)=a_1T(n-1)+ a_2T(n-2)+\ldots+a_kT(n-k), \quad (n \ge k)$$

(II) 线性非齐次递归方程

$$T(n)=a_1T(n-1)+ a_2T(n-2)+\ldots+a_kT(n-k)+f(n), (n \ge k)$$

(III) 分而治之型递归方程

$$T(n)=a_1T(n/c_1)+a_2T(n/c_2)+\ldots+a_pT(n/c_p)+f(n), \quad (n > n_0)$$

递归关系

- 常用方法
  - 差分方程法
  - 展开法
  - 换元法
  - 数学归纳法

递归关系

- **(I) 线性齐次递归方程**

  $T(n)=a_1T(n-1)+a_2T(n-2)+\ldots+a_kT(n-k)$， $n \geq k$

  初始条件（略）。

  - **主要解法：　差分方程法**

- **[例1]**

  $f(n)=3f(n-1)+4f(n-2)$, $n>1$;

  初始条件：$f(0)=1$，$f(1)=4$

观察： $t^n(t\neq 0)$是递归方程的解（忽略初始条件）

$\Leftrightarrow \quad t^n=3t^{n-1}+4t^{n-2}$

$\Leftrightarrow \quad t^2=3t+4$ (特征方程)

t^n是递归方程的解(忽略初始条件) $\Leftrightarrow$ t是对应特征方程的根

解答：

Step 1

特征方程为：$t^2 - 3t - 4=0$。解得 $t_1=4,\ t_2= -1$。

Step 2

于是，$f(n)=c_1\cdot 4^n + c_2\cdot(-1)^n$（$n\geq 0$），其中 c_1,c_2待定。

Step 3

由$f(0)=1,\ f(1)=4$有：$\begin{cases} c_1 + c_2 = 1 \\ 4c_1 - c_2 = 4 \end{cases}$ 由此得$c_1=1,\ c_2 =0$。

Step 4

所以，$f(n)=4^n$（$n\geq 0$）

[例2] $f(n)=4f(n-1)-4f(n-2)$, $n>1$;

初始条件：$f(0)=6, f(1)=8$。

- **观察：** 如果t是特征方程$at^2+bt+c=0$ 的重根，

则 $b^2-4ac=0$， 即$b^2=4ac$，

且 $t=-b/2a$。

因此， $ant^n+b(n-1)t^{n-1}+c(n-2)t^{n-2}$

$=nt^{n-2}(at^2+bt+c)-t^{n-2}(bt+2c)$

$=0-t^{n-2}[b\times(-b/2a)+2c]=0$

若 t是特征方程$at^2+bt+c=0$ 的重根， 则t^n, nt^n是原递归方程$af(n)+bf(n-1)+cf(n-2)=0$的解(忽略初始条件)

- **解答：**

特征方程为：$t^2-4t+4=0$。解得 $t_1=t_2=2$。

于是， $f(n)=c_1\cdot 2^n+c_2\cdot n\cdot 2^n\ (n\geq 0)$，其中$c_1$，$c_2$待定。

由$f(0)=6$，$f(1)=8$有：$c_1=6$，$2c_1+2c_2=8$。

由此得 $c_1=6, c_2=-2$。

所以，

$$\boldsymbol{f(n)=3\cdot 2^{n+1}-n\cdot 2^{n+1}\ (n\geq 0)}$$

■ **思考：**

如何解递归方程：$af(n)+bf(n-1)+cf(n-2)+df(n-3)=0$ ？

■ **解法：**

考虑特征方程 $at^3+bt^2+ct+d=0$。

1）如有三个不同的特征根t_1, t_2, t_3，则递归方程的通解为：

$$f(n)=c_1t_1^n+c_2t_2^n+c_3t_3^n$$

2）如特征根为$t_1=t_2, t_3$，则递归方程的通解为：

$$f(n)=c_1t_1^n+c_2nt_1^n+c_3t_3^n$$

3）如特征根为$t_1=t_2=t_3$，则递归方程的通解为：

$$f(n)=c_1t_1^n+c_2nt_1^n+c_3n^2t_1^n$$

递归关系

- **(II) 线性非齐次递归方程**

  $T(n)=a_1T(n-1)+a_2T(n-2)+\ldots+a_kT(n-k)+f(n),\ (n\ge k)$

  初始条件（略）。

  - **主要解法：　差分方程法**

t^n是递归方程齐次部分的解
(忽略初始条件)
$\Leftrightarrow$ t是对应特征方程的根

- **[例3]**

$T(n)=T(n-1)+T(n-2)+b$, $n>1$,

初始条件： $T(1)=T(0)=a$ ，其中a，b均为非负常数。

- **解答：**

Step 1 特征方程为：$t^2-t-1=0$，解得$t_1=(1+5^{1/2})/2$, $t_2=(1-5^{1/2})/2$。

Step 2 显然，$T(n)=-b$是方程的一个特解。

Step 3 于是，$T(n)=c_1\cdot t_1^n+c_2\cdot t_2^n-b$ $(n\geq 0)$，其中c_1，c_2待定。

Step 4 由$T(0)=a$，$T(1)=a$有：$\begin{cases} c_1+c_2-b=a \\ c_1\cdot t_1+c_2\cdot t_2-b=a \end{cases}$ 得$c_1=$？ $c_2=$？

Step 5 所以，$\mathbf{T(n)= c_1\cdot t_1^n + c_2\cdot t_2^n - b}$ （$n\geq 0$）

其渐近表示为：$\mathbf{T(n)= \Theta(((1+5^{1/2})/2)^n)}$

- **说明：**

与上例相关实例有：求Fibonacci数列的第n+1项的递归算法。该数列的定义为：

$F_0 = F_1 = 1, F_i = F_{i-1} + F_{i-2}$， $i \geq 2$。

由此定义自然导出如下递归算法：

```
int  F(int n)
{  if(n= =0||n= =1) return 1;
   else if(n>=2) return F(n –1)+F(n –2);
}
```

分析： 选“+”为基本操作，则算法的时间复杂度T(n)满足：

$T(n)=T(n-1)+T(n-2)+1, T(0)=0$。

由例3可解得T(n)。

[例4] 河内塔问题的递归求解算法如下(C++描述)：

```
void Hanoi(int n,char a, char b,char c)
{
   if(n>0)
   {     Hanoi(n–1,a,c,b);
         cout<<"Move Disk No."
             <<n<<"from  Pile"<<a<<"to"<<c<<endl;
         Hanoi(n–1,b,a,c);
    }
}
```

解答：

使用输出语句为基本操作。若用T(n)表示该算法的计算时间，则有

$$T(n)=2T(n-1)+1,\ n>1$$

$$T(1)=1。$$

使用展开法、差分法或数学归纳法可得：

$$T(n)=2^n-1,$$

即， $T(n)=\Theta(2^n)$。

递归关系

- **(III) 分而治之型递归方程**

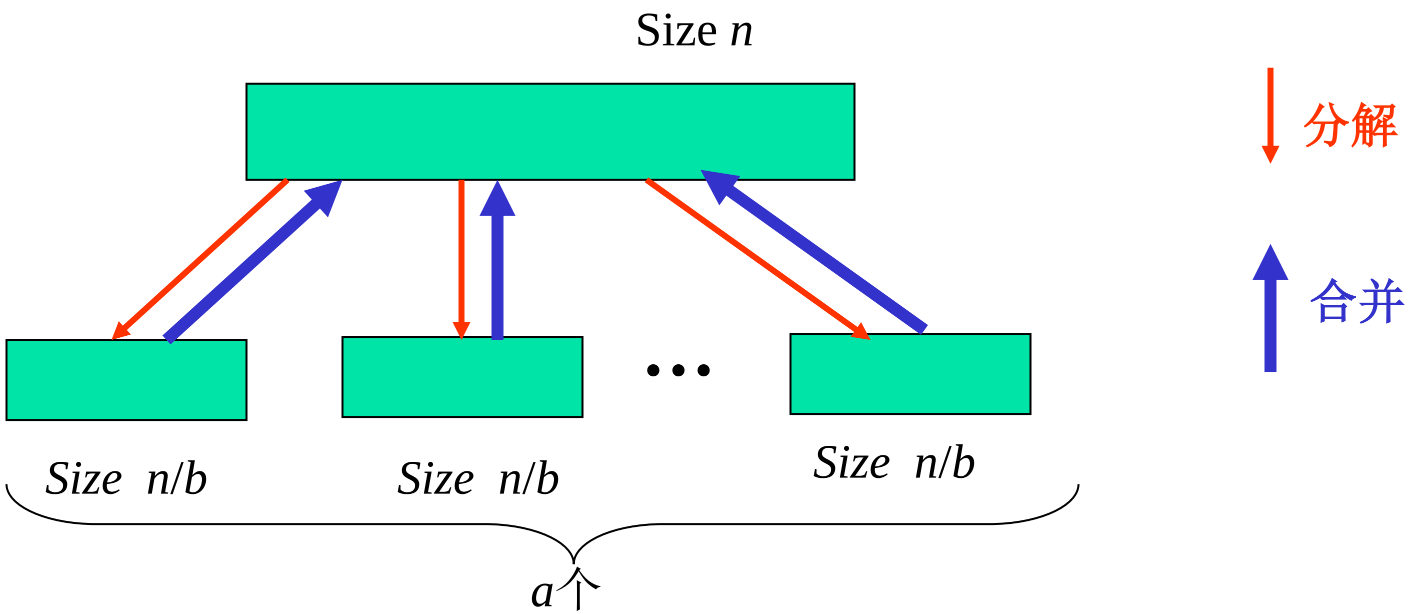
  e.g. $f(n)=af(n/b)+g(n)$, $n>1$，

  初始条件：$f(1)=c$，这里的a,b,c均是正常数。

- **说明：**

  考虑这样的递归过程：它把大小为n的问题分成了a个大小为n/b的子问题去解。设大小为1的问题需要c个单位时间，把a个子问题的解合并成原问题的解需要$g(n)$个单位时间。用$f(n)$表示大小为n的问题的计算时间，则得上述递归方程。

分析



(1) 递归求解子问题的时间为 $f(n/b)$
(2) 分解和合并时间为 $g(n)$
(3) 原问题的求解时间为 $f(n)=a\, f(n/b)+g(n)$

- **[例5]** 解递归方程

  $f(n)=2f(n/2)+bn\log n$,　　$n>1$;

  初始条件：$f(1)=d$ 。其中b和d都是非负实数，$n=2^k$。

解法1（展开法）：

令 $g(n)=bn\log n$，则

- $f(n)=f(2^k)=2f(2^{k-1})+g(2^k)$
- $=2^2f(2^{k-2})+2g(2^{k-1})+g(2^k)$
- ……
- $=2^kf(2^0)+\sum_{i=0}^{k-1}2^i g(2^{k-i})$
- $= 2^k d+\sum_{i=0}^{k-1}2^i\cdot b\cdot 2^{k-i}\cdot\log 2^{k-i}$
- $=n\cdot d+b\cdot 2^k\sum_{i=0}^{k-1}(k-i)$
- $=n\cdot d+b\cdot n\sum_{i=1}^{k}i$
- $=n\cdot d+b\cdot n\cdot k(k+1)/2$
- $=n\cdot d+b\cdot n\cdot\log n\cdot(\log n+1)/2$　（注意：由$n=2^k$ 有$k=\log n$）
- 所以，　$f(n)=\Theta(n\log^2 n)$

解法2（换元法）：

令$T(i)=f(2^i)$，则由 $f(2^i)=2f(2^{i-1})+ b\cdot 2^i \cdot\log 2^i$ 可得递归关系

$$T(i)= 2T(i-1)+ bi\cdot 2^i，\quad T(0)=f(1)=d.$$

- 令$g(i)= bi\cdot 2^i$，则
- $T(k)=2T(k-1)+ g(k)$
- $\quad =2^2 T(k-2)+ 2g(k-1) + g(k)$
- $\quad \cdots\cdots$
- $\quad = 2^k T(0)+ \sum_{i=0}^{k-1}2^i g(k-i)$
- $\quad = 2^k \cdot d+\sum_{i=0}^{k-1}2^i \cdot b \cdot(k-i)\cdot 2^{k-i}$
- $\quad = 2^k \cdot d+ 2^k \cdot b \cdot\sum_{i=0}^{k-1} (k-i)$
- $\quad = 2^k \cdot d+ 2^k \cdot b \cdot\sum_{i=1}^{k} i$
- $\quad = 2^k \cdot d+ 2^k \cdot b \cdot k(k+1)/2$，

所以，$\boldsymbol{f(n)=f(2^k)=T(k)=d\cdot n+b\cdot n\cdot\log n\cdot(\log n+1)/2 = \Theta(n\log^2 n)}$

（注意：由$n=2^k$ 有$k=\log n$）

1.5. 最优算法（optimal algorithm）

- 最优算法

如果能够证明求解问题P的任何算法的时间是$\Omega(f(n))$，那么称求解问题P的时间为 $O(f(n))$的任一算法为问题P的最优算法。

1.5. 最优算法（optimal algorithm ）

- **基于比较的排序问题**

在第12章将证明，任何使用元素的比较给n个元素的数组进行排序的算法的最坏情况运行时间一定是$\Omega(n\log n)$。因此，归并排序、堆排序算法都是最优算法。

小结

- 算法分析的概念
  - 渐近复杂度：O, Ω, Θ
  - 平均时间复杂度，最坏时间复杂度，最好时间复杂度
  - 最优算法

- 分析算法的基本方法
  - 分析算法时间复杂度的步骤
  - 基本运算执行频数的统计方法
  - 数学知识

    求和公式、定积分近似求和、递归方程的求解

小结

分析算法复杂度的基本步骤

基本技术：

- 根据循环统计基本操作次数
- 用递归关系统计基本操作次数
- 用平摊方法统计基本操作次数

基本技术：

- 常用求和公式
- 定积分近似求和
- 递归方程求解

选取基本操作

统计基本操作频数

计算并简化统计结果

基本概念：

- 平均时间复杂度，最坏时间复杂度
  最好时间复杂度
- 渐近复杂度：O, Ω, Θ