

第1章 算法概述

学习要点:

- 理解算法的概念。
- 理解什么是程序，程序与算法的区别和内在联系。
- 掌握算法的计算复杂性概念。
- 掌握算法渐近复杂性的数学表述。

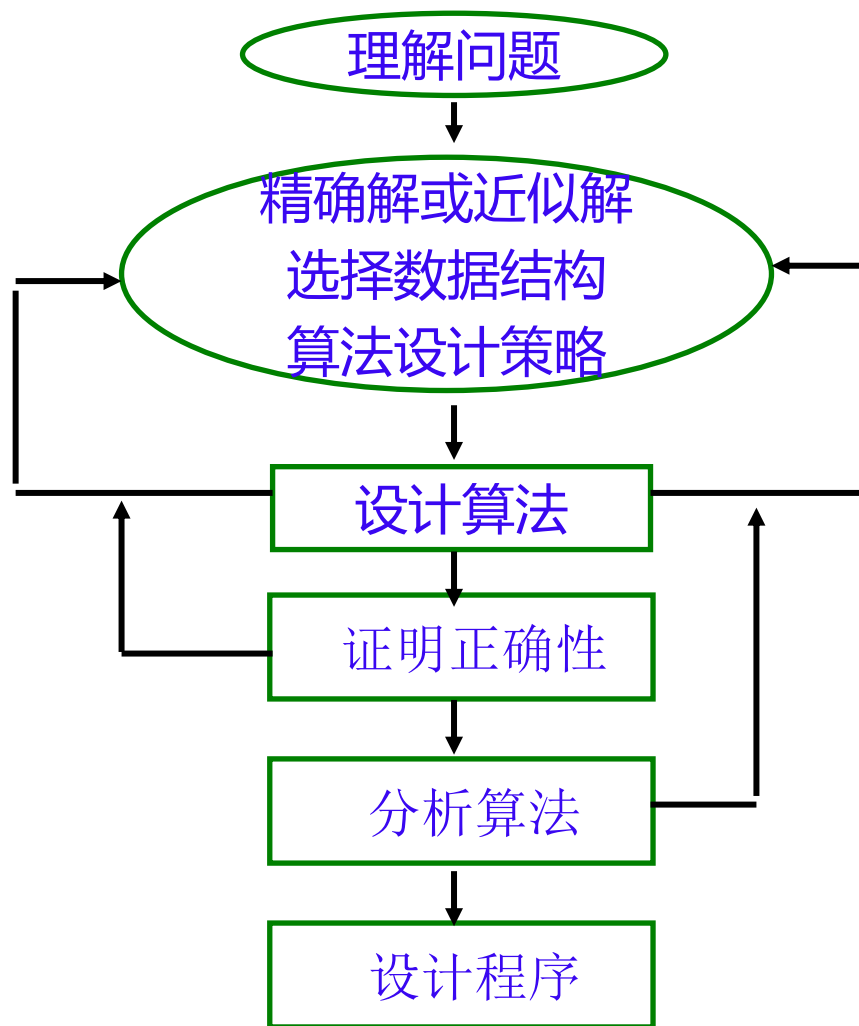
算法(Algorithm)

- 算法是指解决问题的一种方法或一个过程。
- 算法是若干指令的有穷序列，满足性质：
 - (1)输入：有外部提供的量作为算法的输入。
 - (2)输出：算法产生至少一个量作为输出。
 - (3)确定性：组成算法的每条指令是清晰，无歧义的。
 - (4)有限性：算法中每条指令的执行次数是有限的，执行每条指令的时间也是有限的。

程序(Program)

- 程序是算法用某种程序设计语言的具体实现。
- 程序可以不满足算法的性质(4)。
- 例如操作系统，是一个在无限循环中执行的程序，因而不是一个算法。
- 操作系统的各种任务可看成是单独的问题，每一个问题由操作系统中的一个子程序通过特定的算法来实现。该子程序得到输出结果后便终止。

问题求解(Problem Solving)



算法复杂性分析

- 算法复杂性 = 算法所需要的计算机资源
- 算法的时间复杂性 $T(n)$;
- 算法的空间复杂性 $S(n)$ 。

其中 n 是问题的规模（输入大小）。

算法的时间复杂性

(1) 最坏情况下的时间复杂性

$$T_{\max}(n) = \max\{ T(I) \mid \text{size}(I)=n \}$$

(2) 最好情况下的时间复杂性

$$T_{\min}(n) = \min\{ T(I) \mid \text{size}(I)=n \}$$

(3) 平均情况下的时间复杂性

$$T_{\text{avg}}(n) = \sum_{\text{size}(I)=n} p(I)T(I)$$

其中 I 是问题的规模为 n 的实例， $p(I)$ 是实例 I 出现的概率。

算法渐近复杂性

$T(n) \rightarrow \infty$, as $n \rightarrow \infty$; $(T(n) - t(n)) / T(n) \rightarrow 0$, as $n \rightarrow \infty$;

- $t(n)$ 是 $T(n)$ 的渐近性态，为算法的渐近复杂性。
- 在数学上， $t(n)$ 是 $T(n)$ 的渐近表达式，是 $T(n)$ 略去低阶项留下的主项。它比 $T(n)$ 简单。

渐近分析的记号 $\Theta, O, \Omega, o, \omega$

在下面的讨论中，对所有 n ， $f(n) \geq 0$ ， $g(n) \geq 0$ 。

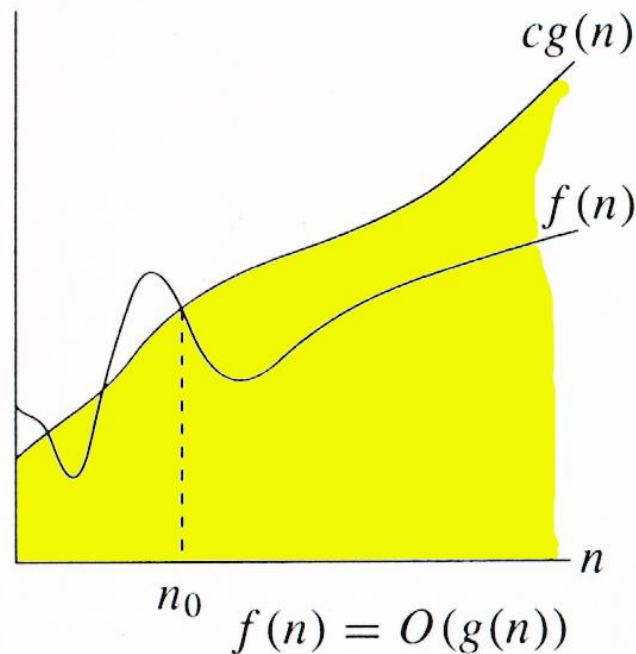
(1) 渐近上界记号 O

$O(g(n)) = \{ f(n) \mid \text{存在正常数 } c \text{ 和 } n_0 \text{ 使得对所有 } n \geq n_0 \text{ 有:}$

$$0 \leq f(n) \leq cg(n) \}$$

$$2n + 3 = O(n^2).$$

$$3n^3 = O(n^4)$$

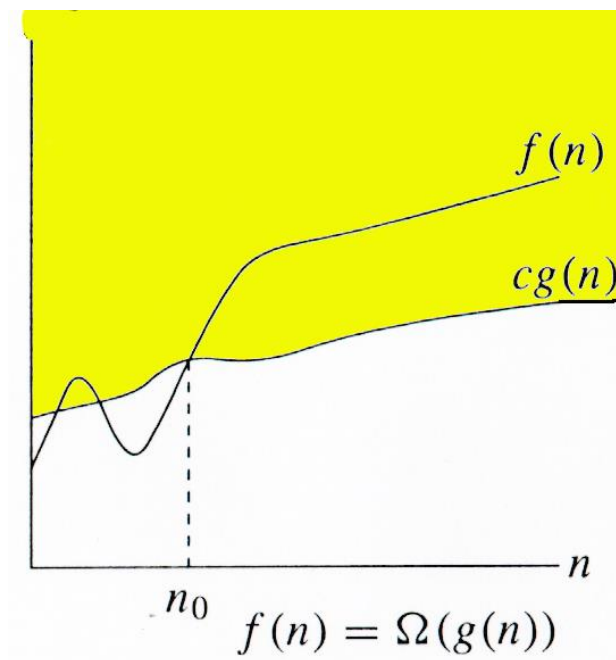


- 用来作比较的函数 $g(n)$ 应该尽量接近所考虑的函数 $f(n)$. $2n+3=O(n^2)$ 松散的界限; $2n+3=O(n)$ 较好的界限。
- $f(n)=O(g(n))$ 不能写成 $g(n)=O(f(n))$, 因为两者并不等价。实际上, 这里的等号并不是通常相等的含义。按照定义, 用集合符号更准确些。
- $O(g(n))=\{f(n)|f(n)\text{满足: 存在正的常数}c\text{和}n_0, \text{使得当}n\geq n_0\text{时}f(n)\leq cg(n)\}$ 所以, 人们常常把 $f(n)=O(g(n))$ 读作: “ $f(n)$ 是 $g(n)$ 的一个大O成员”。

(2) 渐近下界记号 Ω

$\Omega(g(n)) = \{ f(n) \mid \text{存在正常数 } c \text{ 和 } n_0 \text{ 使得对所有 } n \geq n_0 \text{ 有: } 0 \leq cg(n) \leq f(n) \}$

$$3n^3 = \Omega(n^2)$$

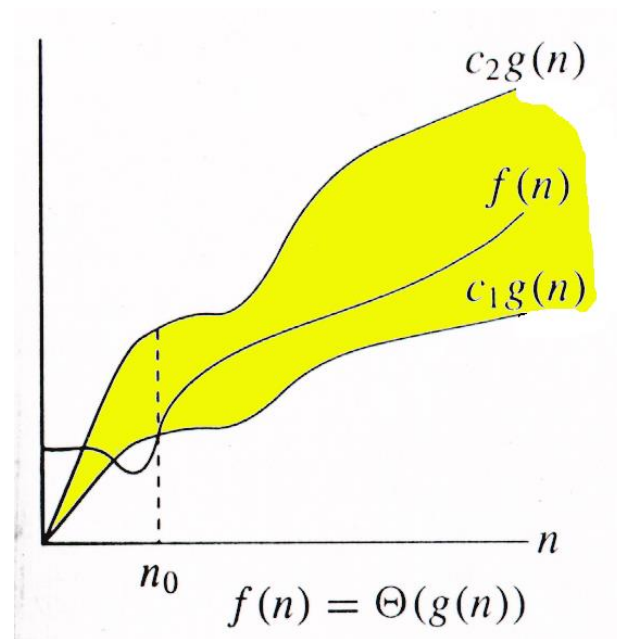


(3) 紧渐近界记号 Θ

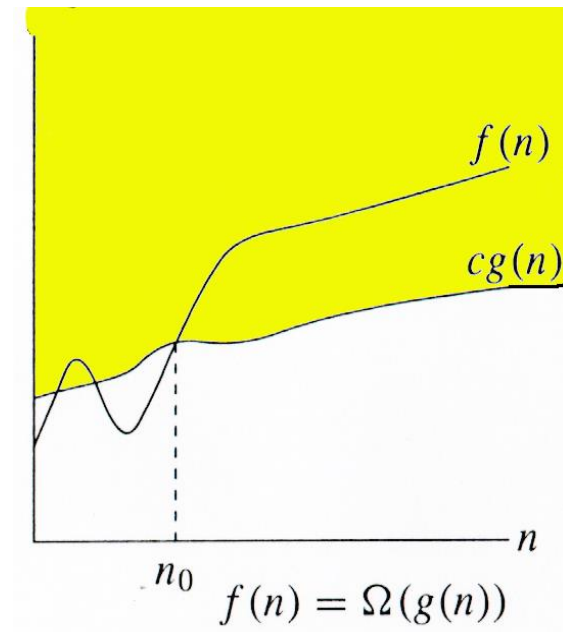
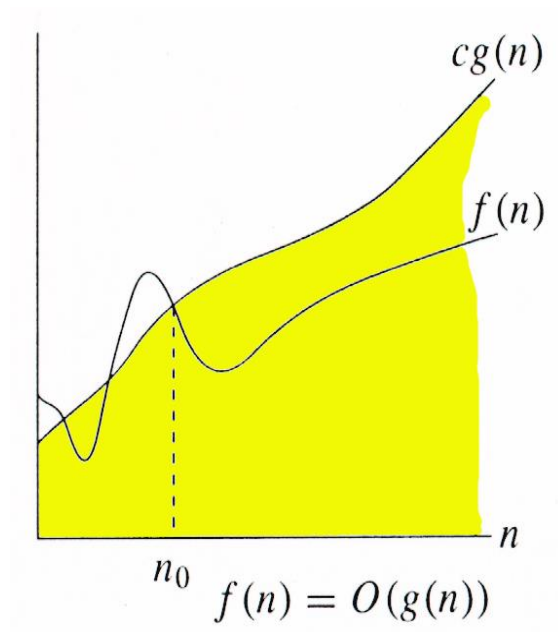
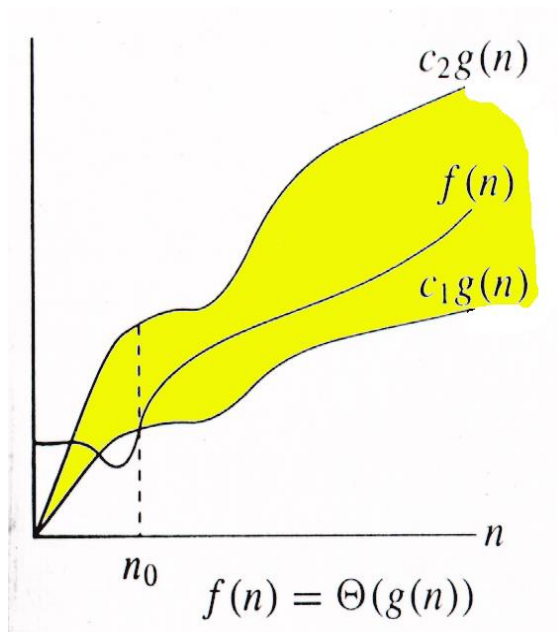
$\Theta(g(n)) = \{ f(n) \mid \text{存在正常数 } c_1, c_2 \text{ 和 } n_0 \text{ 使得对所有 } n \geq n_0 \text{ 有}$
 $: c_1 g(n) \leq f(n) \leq c_2 g(n) \}$

定理1: $\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$

$$10n^2 - 3n = \Theta(n^2)$$



Θ , O , Ω 之间的关系



Big-O

略去低阶项和常数系数项留下的主项

略去低阶项

- $4n + 5 \Rightarrow 4n$
- $0.5 n \log n - 2n + 7 \Rightarrow 0.5 n \log n$

略去常数系数项

- $4n \Rightarrow n$
- $0.5 n \log n \Rightarrow n \log n$
- $\log n^2 = 2 \log n \Rightarrow \log n$
- $\log_3 n = (\log_3 2) \log n \Rightarrow \log n$

$$2n^2 + 4n = O(n^2) \quad \checkmark$$

$$O(n^2) = 2n^2 + 4n \quad \times$$

Big-O 实例

$$n^2 + 100n = O(n^2)$$

$$(n^2 + 100n) \leq 2n^2 \quad \text{for } n \geq 10$$

$$n^2 + 100n = \Omega(n^2)$$

$$(n^2 + 100n) \geq 1n^2 \quad \text{for } n \geq 0$$

$$n^2 + 100n = \theta(n^2)$$

$$n \log n = O(n^2)$$

$$n \log n = \theta(n \log n)$$

$$n \log n = \Omega(n)$$

- ♦ 插入排序在最坏的情况下需要 $\theta(n^2)$ ，所以排序是 $O(n^2)$
- ♦ 任意的排序算法都需要查看每个元素，所以排序是 $\Omega(n)$.
- ♦ 实际上，合并排序在最坏的情况下是 $\theta(n \log n)$

(4) 非紧上界记号 o

$o(g(n)) = \{ f(n) \mid \text{对于任何正常数 } c > 0, \text{ 存在正数和 } n_0 > 0$
使得对所有 $n \geq n_0$ 有: $0 \leq f(n) < cg(n) \}$

等价于 $f(n) / g(n) \rightarrow 0$, as $n \rightarrow \infty$ 。

(5) 非紧下界记号 ω

$\omega(g(n)) = \{ f(n) \mid \text{对于任何正常数 } c > 0, \text{ 存在正数和 } n_0 > 0$
使得对所有 $n \geq n_0$ 有: $0 \leq cg(n) < f(n) \}$

等价于 $f(n) / g(n) \rightarrow \infty$, as $n \rightarrow \infty$ 。

$$f(n) \in \omega(g(n)) \Leftrightarrow g(n) \in o(f(n))$$

渐近分析记号在等式和不等式中的意义

$f(n) = \Theta(g(n))$ 的确切意义是： $f(n) \in \Theta(g(n))$ 。

一般情况下，等式和不等式中的渐近记号 $\Theta(g(n))$ 表示 $\Theta(g(n))$ 中的某个函数。

例如： $2n^2 + 3n + 1 = 2n^2 + \Theta(n)$ 表示

$2n^2 + 3n + 1 = 2n^2 + f(n)$ ，其中 $f(n)$ 是 $\Theta(n)$ 中某个函数。

等式和不等式中渐近记号 O, o, Ω 和 ω 的意义是类似的。

渐近分析中函数比较

$$f(n) = O(g(n)) \approx a \leq b;$$

$$f(n) = \Omega(g(n)) \approx a \geq b;$$

$$f(n) = \Theta(g(n)) \approx a = b;$$

$$f(n) = o(g(n)) \approx a < b;$$

$$f(n) = \omega(g(n)) \approx a > b.$$

渐近分析记号的若干性质

(1) 传递性:

$$f(n) = \Theta(g(n)), \quad g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n));$$

$$f(n) = O(g(n)), \quad g(n) = O(h(n)) \Rightarrow f(n) = O(h(n));$$

$$f(n) = \Omega(g(n)), \quad g(n) = \Omega(h(n)) \Rightarrow f(n) = \Omega(h(n));$$

$$f(n) = o(g(n)), \quad g(n) = o(h(n)) \Rightarrow f(n) = o(h(n));$$

$$f(n) = \omega(g(n)), \quad g(n) = \omega(h(n)) \Rightarrow f(n) = \omega(h(n));$$

(2) 反身性:

$$f(n) = \Theta(f(n));$$

$$f(n) = O(f(n));$$

$$f(n) = \Omega(f(n)).$$

(3) 对称性:

$$f(n) = \Theta(g(n)) \Leftrightarrow g(n) = \Theta(f(n)).$$

(4) 互对称性:

$$f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n));$$

$$f(n) = o(g(n)) \Leftrightarrow g(n) = \omega(f(n));$$

(5) 算术运算:

$$O(f(n)) + O(g(n)) = O(\max\{f(n), g(n)\}) ;$$

$$O(f(n)) + O(g(n)) = O(f(n) + g(n)) ;$$

$$O(f(n)) * O(g(n)) = O(f(n) * g(n)) ;$$

$$O(cf(n)) = O(f(n)) ;$$

$$g(n) = O(f(n)) \Rightarrow O(f(n)) + O(g(n)) = O(f(n)) .$$

规则 $O(f(n)) + O(g(n)) = O(\max\{f(n), g(n)\})$ 的证明:

对于任意 $f_1(n) \in O(f(n))$ ，存在正常数 c_1 和自然数 n_1 ，使得对所有 $n \geq n_1$ ，有 $f_1(n) \leq c_1 f(n)$ 。

类似地，对于任意 $g_1(n) \in O(g(n))$ ，存在正常数 c_2 和自然数 n_2 ，使得对所有 $n \geq n_2$ ，有 $g_1(n) \leq c_2 g(n)$ 。

令 $c_3 = \max\{c_1, c_2\}$ ， $n_3 = \max\{n_1, n_2\}$ ， $h(n) = \max\{f(n), g(n)\}$ 。

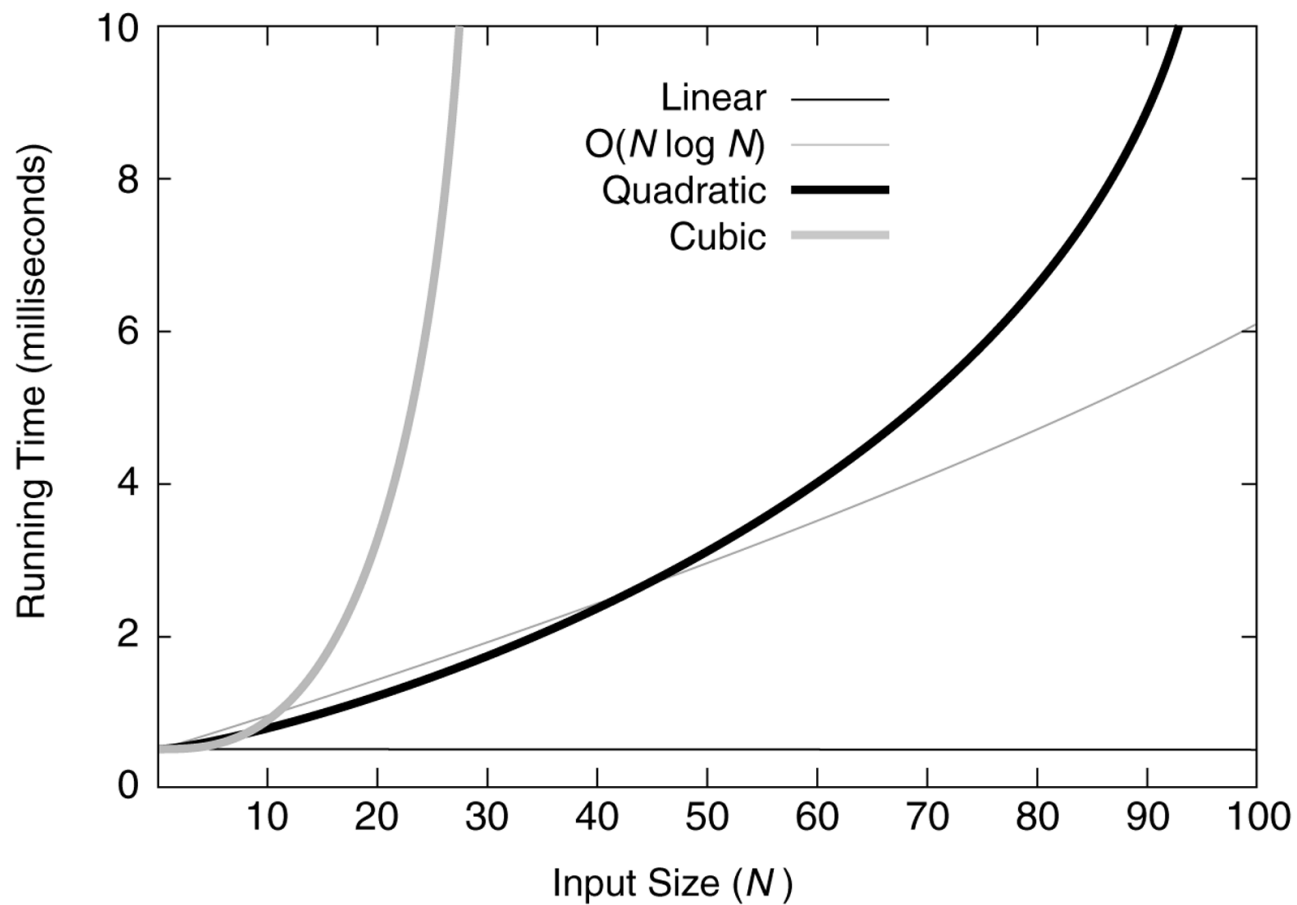
则对所有的 $n \geq n_3$ ，有

$$\begin{aligned} f_1(n) + g_1(n) &\leq c_1 f(n) + c_2 g(n) \\ &\leq c_3 f(n) + c_3 g(n) = c_3 (f(n) + g(n)) \\ &\leq c_3 2 \max\{f(n), g(n)\} \\ &= 2c_3 h(n) = O(\max\{f(n), g(n)\}) . \end{aligned}$$

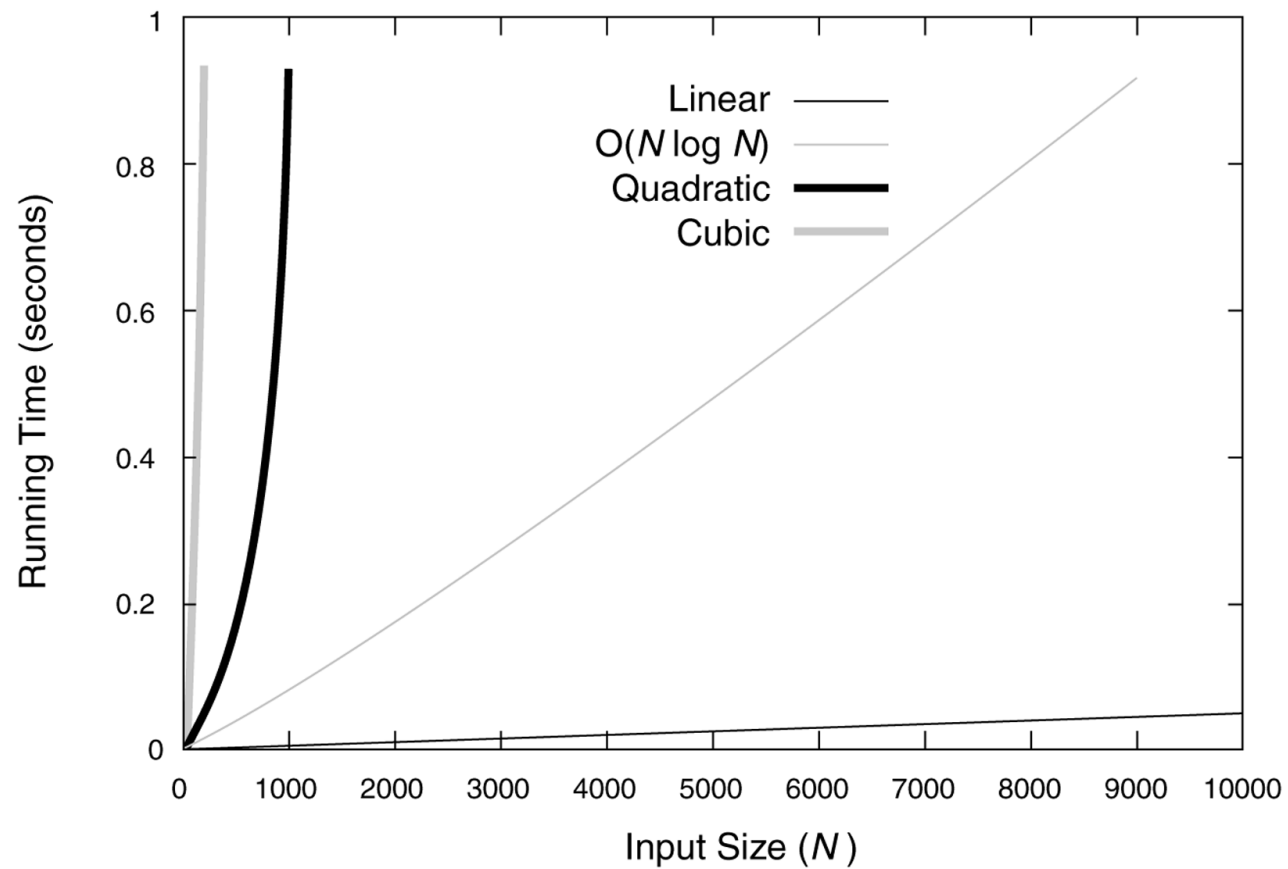
算法分析中常见的复杂性函数

FUNCTION	NAME
c	Constant
$\log N$	Logarithmic
$\log^2 N$	Log-squared
N	Linear
$N \log N$	$N \log N$
N^2	Quadratic
N^3	Cubic
2^N	Exponential

小规模数据



中等规模数据



用C++描述算法

CATEGORY	EXAMPLES	ASSOCIATIVITY
Operations on References	. []	Left to right
Unary	++ -- ! - (type)	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift (bitwise)	<< >>	Left to right
Relational	< <= > >= instanceof	Left to right
Equality	== !=	Left to right
Boolean (or bitwise) AND	&	Left to right
Boolean (or bitwise) XOR	^	Left to right
Boolean (or bitwise) OR		Left to right
Logical AND	&&	Left to right
Logical OR		Left to right
Conditional	?:	Right to left
Assignment	= *= /= %= += -=	Right to left

分析代码

C++ 操作

顺序语句

条件语句

循环

函数调用

递归函数

常数时间

语句时间和

较大分支+条件测试

迭代和

函数体代价

求解递归方程

嵌套循环

```
for i = 1 to n do  
  for j = 1 to n do  
    sum = sum + 1
```

$$\sum_{i=1}^n \sum_{j=1}^n 1 = \sum_{i=1}^n n = n^2$$

```
for i = 1 to n do
  for j = i to n do
    sum = sum + 1
```

$$\sum_{i=1}^n \sum_{j=i}^n 1 = \sum_{i=1}^n (n - i + 1) = \sum_{i=1}^n (n + 1) - \sum_{i=1}^n i =$$

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

递归 (Recursion)

- 递归过程一般可以通过解递归方程进行分析

- 基本形式:

$$T(n) =$$

base case: some constant

recursive case: $T(\text{subproblems}) + T(\text{combine})$

- 结果依赖于

- 子问题的个数
- 子问题的规模
- 子问题的解如何合并形成整个问题的解

二分查找

BinarySearch (A, x)

在有序数组A中查询x

7	12	30	35	75	83	87	90	97	99
---	----	----	----	----	----	----	----	----	----

子问题规模是原来的一半

方程:

$$T(1) \leq b$$

$$T(n) \leq T(n/2) + c \quad \text{for } n > 1$$

二分查找

方程:

$$T(1) \leq b$$

$$T(n) \leq T(n/2) + c \quad \text{for } n > 1$$

求解:

$$T(n) \leq T(n/2) + c$$

$$\leq T(n/4) + c + c$$

$$\leq T(n/8) + c + c + c$$

$$\leq T(n/2^k) + kc$$

$$\leq T(1) + c \log n, \quad k = \log n$$

$$\leq b + c \log n = O(\log n)$$

算法分析方法

例：顺序搜索算法

```
template<class Type>
int seqSearch(Type *a, int n, Type k)
{
    for(int i=0;i<n;i++)
        if (a[i]==k) return i;
    return -1;
}
```

(1) $T_{\max}(n) = \max\{ T(I) \mid \text{size}(I)=n \} = O(n)$

(2) $T_{\min}(n) = \min\{ T(I) \mid \text{size}(I)=n \} = O(1)$

(3) 在平均情况下，假设：

(a) 搜索成功的概率为 p ($0 \leq p \leq 1$)；

(b) 在数组的每个位置 i ($0 \leq i < n$) 搜索成功的概率相同，均为 p/n 。

$$\begin{aligned} T_{\text{avg}}(n) &= \sum_{\text{size}(I)=n} p(I)T(I) \\ &= \left(1 \cdot \frac{p}{n} + 2 \cdot \frac{p}{n} + 3 \cdot \frac{p}{n} + \cdots + n \cdot \frac{p}{n} \right) + n \cdot (1 - p) \\ &= \frac{p}{n} \sum_{i=1}^n i + n(1 - p) = \frac{p(n+1)}{2} + n(1 - p) \end{aligned}$$

算法分析的基本法则

非递归算法：

(1) **for / while** 循环

循环体内计算时间*循环次数；

(2) 嵌套循环

循环体内计算时间*所有循环次数；

(3) 顺序语句

各语句计算时间相加；

(4) **if-else** 语句

if语句计算时间和**else**语句计算时间的较大者。

```

template<class Type>
void insertion_sort(Type *a, int n)
{
    Type key;                                // cost    times
    for (int i = 1; i < n; i++){              // c1      n
        key=a[i];                             // c2      n-1
        int j=i-1;                            // c3      n-1
        while( j>=0 && a[j]>key ){              // c4      sum of ti
            a[j+1]=a[j];                       // c5      sum of (ti-1)
            j--;                               // c6      sum of (ti-1)
        }
        a[j+1]=key;                           // c7      n-1
    }
}

```

$$T(n) = c_1 n + c_2 (n-1) + c_3 (n-1) + c_4 \sum_{i=1}^{n-1} t_i + c_5 \sum_{i=1}^{n-1} (t_i - 1) + c_6 \sum_{i=1}^{n-1} (t_i - 1) + c_7 (n-1)$$

- 在最好情况下， $t_i=1$, for $1 \leq i < n$;

$$T_{\min}(n) = c_1 n + c_2 (n-1) + c_3 (n-1) + c_4 (n-1) + c_7 (n-1)$$

$$= (c_1 + c_2 + c_3 + c_4 + c_7)n - (c_2 + c_3 + c_4 + c_7) = O(n)$$

- 在最坏情况下， $t_i \leq i$, for $1 \leq i < n$;

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

$$T_{\max}(n) \leq c_1 n + c_2 (n-1) + c_3 (n-1) +$$

$$c_4 \left(\frac{n(n-1)}{2} \right) + c_5 \left(\frac{n(n-2)}{2} + 1 \right) + c_6 \left(\frac{n(n-2)}{2} + 1 \right) + c_7 (n-1)$$

$$= \frac{c_4 + c_5 + c_6}{2} n^2 + \left(c_1 + c_2 + c_3 - \frac{c_4}{2} - c_5 - c_6 + c_7 \right) n - (c_2 + c_3 - c_5 + c_7)$$

$$= O(n^2)$$

对于输入数据 $a[i]=n-i, i=0,1,\dots,n-1$ ，算法insertion_sort 达到其最坏情形。因此，

$$\begin{aligned} T_{\max}(n) &\geq \frac{c_4 + c_5 + c_6}{2} n^2 + \left(c_1 + c_2 + c_3 - \frac{c_4}{2} - c_5 - c_6 + c_7 \right) n - (c_2 + c_3 - c_5 + c_7) \\ &= \Omega(n^2) \end{aligned}$$

由此可见， $T_{\max}(n) = \Theta(n^2)$

最优算法

- 问题的计算时间下界为 $\Omega(f(n))$ ，则计算时间复杂性为 $O(f(n))$ 的算法是最优算法。
- 例如，排序问题的计算时间下界为 $\Omega(n\log n)$ ，计算时间复杂性为 $O(n\log n)$ 的排序算法是最优算法。
- 堆排序算法是最优算法。

NP完全性理论

- **多项式时间：**在计算复杂度理论中，指的是一个问题的计算时间不大于问题规模 n 的多项式倍数。即多项式时间就是指时间复杂度是个多项式。程序运行的时间随着数据规模 n 变化的函数为 $f(n)$ ， $f(n)$ 是个多项式函数，那么就可以说是控制在多项式之内。

NP完全性理论

- **P类问题：**所有可以在多项式时间内求解的判定问题构成P类问题。**判定问题：**判断是否有一种能够解决某一类问题的能行算法的研究课题。
- 时间复杂度如(n^2 , n^4 , $n(\log(n))$)都是P时间的，指数级别的如(2^n , n^n)这些就不是P时间。

- **NP类问题：**所有的非确定性多项式时间可解的判定问题构成NP类问题。(Non-deterministic polynomial)
- 给定一个问题，我们可能不知道如何解，但如果通过连蒙带猜，得到了一个解，对于这个解，我们可以在P时间内验证它正确与否的一类问题，成为NP问题。

算法渐近复杂性分析中常用函数

(1) 单调函数

单调递增: $m \leq n \Rightarrow f(m) \leq f(n)$;

单调递减: $m \leq n \Rightarrow f(m) \geq f(n)$;

严格单调递增: $m < n \Rightarrow f(m) < f(n)$;

严格单调递减: $m < n \Rightarrow f(m) > f(n)$.

(2) 取整函数

$\lfloor x \rfloor$: 不大于 x 的最大整数;

$\lceil x \rceil$: 不小于 x 的最小整数。

取整函数的若干性质

$$x-1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x+1;$$

$$\lfloor n/2 \rfloor + \lceil n/2 \rceil = n;$$

对于 $n \geq 0$, $a, b > 0$, 有:

$$\lceil \lceil n/a \rceil / b \rceil = \lceil n/ab \rceil;$$

$$\lfloor \lfloor n/a \rfloor / b \rfloor = \lfloor n/ab \rfloor;$$

$$\lceil a/b \rceil \leq (a+(b-1))/b;$$

$$\lfloor a/b \rfloor \geq (a-(b-1))/b;$$

$f(x) = \lfloor x \rfloor$, $g(x) = \lceil x \rceil$ 为单调递增函数。

(3) 多项式函数

$$p(n) = a_0 + a_1 n + a_2 n^2 + \dots + a_d n^d; \quad a_d > 0;$$

$$p(n) = \Theta(n^d);$$

$$f(n) = O(n^k) \Leftrightarrow f(n) \text{ 多项式有界};$$

$$f(n) = O(1) \Leftrightarrow f(n) \leq c;$$

$$k \geq d \Rightarrow p(n) = O(n^k);$$

$$k \leq d \Rightarrow p(n) = \Omega(n^k);$$

$$k > d \Rightarrow p(n) = o(n^k);$$

$$k < d \Rightarrow p(n) = \omega(n^k).$$

(4) 指数函数

对于正整数 m, n 和实数 $a > 0$:

$$a^0 = 1;$$

$$a^1 = a;$$

$$a^{-1} = 1/a;$$

$$(a^m)^n = a^{mn};$$

$$(a^m)^n = (a^n)^m;$$

$$a^m a^n = a^{m+n};$$

$a > 1 \Rightarrow a^n$ 为单调递增函数;

$$a > 1 \Rightarrow \lim_{n \rightarrow \infty} \frac{n^b}{a^n} = 0 \Rightarrow n^b = o(a^n)$$

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots = \sum_{i=0}^{\infty} \frac{x^i}{i!}$$

$$e^x \geq 1+x;$$

$$|x| \leq 1 \Rightarrow 1+x \leq e^x \leq 1+x+x^2 ;$$

$$e^x = 1+x+ \Theta(x^2), \text{ as } x \rightarrow 0;$$

$$\lim_{n \rightarrow \infty} \left(1 + \frac{x}{n}\right)^n = e^x$$

(5) 对数函数

$$\log n = \log_2 n;$$

$$\lg n = \log_{10} n;$$

$$\ln n = \log_e n;$$

$$\log^k n = (\log n)^k;$$

$$\log \log n = \log(\log n);$$

for $a > 0, b > 0, c > 0$

$$a = b^{\log_b a}$$

$$\log_c(ab) = \log_c a + \log_c b$$

$$\log_b a^n = n \log_b a$$

$$\log_b a = \frac{\log_c a}{\log_c b}$$

$$\log_b (1/a) = -\log_b a$$

$$\log_b a = \frac{1}{\log_a b}$$

$$a^{\log_b c} = c^{\log_b a}$$

$$|x| \leq 1 \Rightarrow \ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5} - \dots$$

$$\text{for } x > -1, \quad \frac{x}{1+x} \leq \ln(1+x) \leq x$$

$$\text{for any } a > 0, \quad \lim_{n \rightarrow \infty} \frac{\log^b n}{(2^a)^{\log n}} = \lim_{n \rightarrow \infty} \frac{\log^b n}{n^a} = 0 \Rightarrow \log^b n = o(n^a)$$

(6) 阶层函数

$$n! = \begin{cases} 1 & n = 0 \\ n(n-1)! & n > 0 \end{cases}$$

$$n! = 1 \cdot 2 \cdot 3 \cdots n$$

Stirling's approximation

$$n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right)$$

$$n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n e^{\alpha_n} \quad \frac{1}{12n+1} < \alpha_n < \frac{1}{12n}$$

$$n! = o(n^n)$$

$$n! = \omega(2^n)$$

$$\log(n!) = \Theta(n \log n)$$

主定理 (Master Theorem)

设 $a \geq 1$ 和 $b > 1$ 为常数, 设 $f(n)$ 为一函数, $T(n)$ 由递归式

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

其中 $\frac{n}{b}$ 指 $\left\lfloor \frac{n}{b} \right\rfloor$ 和 $\left\lceil \frac{n}{b} \right\rceil$, 可以证明, 略去上下取整不会对结果造成影响。那么 $T(n)$ 可能有如下的渐进界

(1) 若 $f(n) < n^{\log_b a}$, 且是多项式的小于。即

$$\exists \varepsilon > 0, \text{ 有 } f(n) = O(n^{\log_b a - \varepsilon}), \text{ 则 } T(n) = \Theta(n^{\log_b a})$$

(2) 若 $f(n) = n^{\log_b a}$, 则 $T(n) = \Theta(n^{\log_b a} \log n)$

(3) 若 $f(n) > n^{\log_b a}$, 且是多项式的大于。即

$$\exists \varepsilon > 0, \text{ 有 } f(n) = \Omega(n^{\log_b a + \varepsilon}), \text{ 且对 } \forall c < 1 \text{ 与所有足够大的 } n, \text{ 有 } af\left(\frac{n}{b}\right) \leq cf(n), \text{ 则 } T(n) = \Theta(f(n))$$