第三章　　基础：算法和整数

§3.1　　算法 (Algorithm)

1. 定义：算法是一个有穷的指令序列。每条指令必须有清楚的含义，并且在有穷长的时间用有穷的动作能完成。一个算法无论接受任何输入都必须在有穷步内停止。
2. 例子：

例 1：设计一个算法在有穷长的整数序列中找最大的那个整数。

*用类 Pascal 语言来描述。

算法 1：

```
PROCEDURE max(a_1, a_2, ⋯, a_n: integers)
max := a_1;
FOR i := 2 TO n DO
   IF max < a_i THEN max := a_i;
{max is the largest element}
```

*算法一般具有以下性质：

（1）输入：一个算法具有取自某一个指定集合的输入数据；

（2）输出：一个算法产生取自某一个集合的输出数据，这些输出数据是问题的解；

（3）确定性：算法的每一步都必须精确地定义；

（4）正确性：算法应对每一组输入数据产生正确的输出数据；

（5）有穷性：输入数据后，算法应在有穷步后产生所期望的输出；

（6）有效性：算法应能确切地在有穷时间内执行完每一步；

（7）一般性：算法应该能对某种形式的所有问题求解，而不是只对某一组特殊的数据求解。

*例如：求有穷序列中的最大元问题。

3. 查找算法：

例 2：线性查找算法。

问题：在某个有穷长的序列中，找其中是否有某个元素 x.

算法 2：

PROCEDURE linear search(x: integer; $a_1, a_2, \cdots, a_n$: distinct integers)

i := 1;

WHILE (i $\leq n$ and $x \neq a_i$) DO

  i := i+1;

IF i $\leq$ n THEN location := i

ELSE location := 0;

{location is the subscript of the term that equals x, or is 0 if x is not found}

例 3：二分查找：假设有穷序列已经从小到大排好序，找其中是否有某个元素 x。

例如：在以下有序序列中找元素 x=19.

1, 2, 3, 5, 6, 7, 8, 10, 12, 13, 15, 16, 18, 19, 20, 22

将该序列分成两半(该序列原有 16 项,分成两半,各有 8 项)

1, 2, 3, 5, 6, 7, 8, 10；　　12, 13, 15, 16, 18, 19, 20, 22；

X=19 与前一半的最后一项比较，如果 x 小于或等于该项，则在前一半中查找；如果 x 大于该项，则在后一半中查找。现在 x=19>10,故在后一半中查找。

再将后一半分成两半（各含 4 项元素）：

12, 13, 15, 16；　　18, 19, 20, 22；

X=19 与前一半的最后一项比较，如果 x 小于或等于该项，则在前一半中查找；如果 x 大于该项，则在后一半中查找。现在 x=19>16,故在后一半中查找。

再将后一半分成两半（各含 2 项元素）：

18, 19;　　20, 22;

再用以上同样的比较方法，现在 x=19≤19, 故在前一半中查找。再将前一半分成两半（各含 1 项元素）。

18；　　19；

再用以上同样的比较方法,现在 x=19>18, 故在后一半中查找。这时，后一半只剩一个元素 19 了，x=19 与该项相同，因此查到该元素。

算法 3：

PROCEDURE binary search (x: integer; $a_1, a_2, \cdots, a_n$: increasing integers)

```
i := 1;    {i is left endpoint of search interval}
j := n;    {j is right endpoint of search interval}
WHILE i < j DO
BEGIN
    m := ⌊(i + j)/2⌋;
    IF x > a_m THEN i := m+1
    ELSE j := m;
END;
IF x = a_i THEN location := i
ELSE location := 0;
```

{location is the subscript of the term that equals x, or 0 if x is not found}.

3. 贪心算法

*最优化问题：有些问题是寻找一个问题的最优解，即该解使得问题的某个参数最大或最小。

*贪心算法：在求解最优化问题时，每一步都取最合意的选择，这样的算法称为贪心算法。

例 6：假设现在要找 n 分钱的零钱，共有 4 种硬币，25 分，10 分，5 分和 1 分，问怎样找钱，才能使所用的硬币数最少？

例如：找 67 分钱。先用最大的硬币：25 分，$67-25=42$分，再用最大的硬币，$42-25=17$分；这时，用第二大的硬币，$17-10=7$分；再用第三大的硬币，$7-5=2$分；最后用两

次最小的硬币，$2-1=1$，和 $1-1=0$分。这样找的硬币数恰好是最小的。

算法 6：（贪心找零钱算法）

PROCEDURE change($c_1, c_2, \cdots, c_r$: values of denominations of coins, where $c_1 > c_2 > \cdots > c_r$; n: a positive integer);

FOR i := 1 TO r DO

  WHILE n $\geq c_i$ DO

  BEGIN

    add a coin with value c_i to the change;

    $n := n - c_i$;

  END;

注意：贪心算法并不总能够求出最优解，贪心算法是否可用，必须证明其正确性才可用。

§3.2 函数的增长率(The growth of functions)

一． 大 O 记号(Big-O notation)

1. 定义：设 f 和 g 是从整数集合或实数集合到实数集合的函数，我们记 f(x)是 O(g(x))是说：存在正的常数 C 和 k，使得当 x>k，有$|f(x)| \leq C|g(x)|$。

*常数 C 和 k 称为 f(x)是 O(g(x))的证据(witnesses).

*注意：如果存在 f(x)是 O(g(x))的证据 C 和 k，那么这个证据有无穷多个。任取 C’ > C, k’ > k，当 x > k’ > k 时，有$|f(x)| \leq$

$C|g(x)| < C'|g(x)|$。

2. 例子：

例 1：证明：$f(x) = x^2 + 2x + 1$ 是 $O(x^2)$。

解：当 $x > 1$ 时，有$x < x^2, 1 < x^2$。因而有，当 $x > 1$ 时

$0 \le x^2 + 2x + 1 \le x^2 + 2x^2 + x^2 = 4x^2$。

故取 C=4, k=1 作为证据，有 f(x)是 $O(x^2)$。

注意：f(x)是 O(g(x))也可以表示为 f(x)=O(g(x))或 f(x)∈ $O(g(x))$。

O(g(x))表示所有是 O(g(x))的函数的集合。

例 2：证明：$7x^2$是 $O(x^3)$。

解：取 k=7，C=1，当 x>7 时，有$|7x^2| < C|x^3|$，故$7x^2 = O(x^3)$。

例 3：证明：n^2不是 O(n)。

解：要证n^2不是 O(n)，就是要证不存在常数 C>0，k>0，使得只要 n>k，就有$n^2 \le Cn$。给定常数 C>0，要使$n^2 \le Cn$，就有$n \le C$。显然，对任意常数 C>0，只要取 k>C，当 n>k>C 时，总有 n$\le C$不成立。故$n^2 \le Cn$不成立。

二．大 O 记号的一些重要结论

*对任意多项式 f，f 是$O(x^n)$，其中x^n是 f 的最高次项。

定理 1：设$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$，其中$a_0, a_1, \cdots, a_n$都是实数。那么 f(x)是$O(x^n)$。

证明：如果 x>1，有

$|f(x)| = |a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0|$

$\le |a_n| x^n + |a_{n-1}| x^{n-1} + \cdots + |a_1| x + |a_0|$

$= x^n(|a_n| + \frac{|a_{n-1}|}{x} + \cdots + \frac{|a_1|}{x^{n-1}} + \frac{|a_0|}{x^n})$

$\leq x^n(|a_n| + |a_{n-1}| + \cdots + |a_0|)$

$\leq Cx^n$

其中，$C = |a_n| + |a_{n-1}| + \cdots + |a_0|$, $x > k = 1$.

故 f(x)是$O(x^n)$。

例 5：如何估计前 n 个正整数的和的大 O 函数。

解：因为$1 + 2 + \cdots + n \leq n + n + \cdots + n = n^2$,

取 C=1，k=1，当$n \geq k = 1$时，有

$1 + 2 + \cdots + n \leq Cn^2$，故$1 + 2 + \cdots + n$是$O(n^2)$。

例 6：给出 n 阶乘函数 f(n)=n!的大 O 函数。

解：因为 f(n)=n!=$1 \cdot 2 \cdot 3 \cdots \cdot n \leq n \cdot n \cdot \cdots \cdot n = n^n$,

取 k=1，C=1，有 f(n)=n!=$O(n^n)$。

另外，$\log n! \leq \log n^n = n \log n$,

故取 C=1，k=1，有$\log n! = O(n \log n)$。

例 7：已知对任意正整数 n，有$n < 2^n$，即$n = O(2^n)$。利用这个不等式证明 log n =$O(n)$。

解：取 k=1，C=1，有当 n > k = 1 时，有$n < 2^n$，故

$$\log n < Cn = n \ .$$

故 log n = O(n)。

*对于任意底数 b 的 log 函数，因为有

$$\log_b n = \frac{\log n}{\log b} < \frac{n}{\log b}$$

取 k=1，$C=\frac{1}{\log b}$，有当 n > k = 1 时，$\log_b n < Cn$，

故$\log_b n = O(n)$。

3. 一些常见的不同增长率的函数

1, log n , n , n log n , $n^2, 2^n$, n!

(见书 P211，图 3)

三．函数组合的增长率

1. 函数的和

定理 2：假设$f_1(x)$是$O(g_1(x))$，$f_2(x)$是$O(g_2(x))$，那么

$(f_1+f_2)(x)$是$O(\max(|g_1(x)|,|g_2(x)|))$。

证明：由定义知，存在正的常数C_1, C_2, k_1, k_2，使得，当 $x>k_1, x>k_2$时,有$|f_1(x)| \le C_1|g_1(x)|$，$|f_2(x)| \le C_2|g_2(x)|$.

取$k=\max(k_1,k_2)$，当时 x>k，有

$|(f_1+f_2)(x)| = |f_1(x)+f_2(x)|$

$\le |f_1(x)| + |f_2(x)|$

$\le C_1|g_1(x)| + C_2|g_2(x)|$

$\le C_1|g(x)| + C_2|g(x)|$

$= (C_1+C_2)|g(x)|$

$= C|g(x)|$

其中，$C = C_1 + C_2$, g(x)=max($|g_1(x)|, |g_2(x)|$).

推论 1：假设$f_1(x)$和$f_2(x)$都是 O(g(x))，那么$(f_1+f_2)(x)$也是 O(g(x))。

2. 函数的积

定理 3：假设$f_1(x)$是$O(g_1(x))$，$f_2(x)$是$O(g_2(x))$，那么$(f_1f_2)(x)$是$O(|g_1(x)||g_2(x)|)$。

证明：由定理知,存在k_1, C_1和k_2, C_2,使得当 x>k_1，x>k_2时，有$|f_1(x)| \le C_1|g_1(x)|, |f_2(x)| \le C_2|g_2(x)|$。取$k = \max(k_1, k_2)$当 x>k 时，有

$$|(f_1f_2)(x)| = |f_1(x)||f_2(x)|$$

$$\le C_1|g_1(x)| \cdot C_2|g_2(x)|$$

$$= C_1C_2|g_1(x)||g_2(x)|$$

$$= C|g_1(x)||g_2(x)| \ .$$

其中$C = C_1C_2$。

注 意 ：当 $g_1(x) \ge 0$ 和 $g_2(x) \ge 0$ 时 ,$O(|g_1(x)||g_2(x)|) = O(g_1(x)g_2(x))$。

例 8：求以下函数$f(n) = 3n\log(n!) + (n^2 + 3)\log n$的大 O 函数。

解：因为 log (n!)=O(n log n), 3n=O(n)，故由定理 3，有 3n log n!$= O(n^2 \log n)$. 而由定理 2 知，$n^2 + 3 = O(n^2)$。再由定理 3，有$(n^2 + 3)\log n = O(n^2 \log n)$。再由定理 2，有

$$f(n) = 3n\log(n!) + (n^2 + 3)\log n = O(n^2 \log n)$$ 。

例 9：给出以下函数$f(x) = (x + 1)\log(x^2 + 1) + 3x^2$的大 O 函数。

解：因为$x^2 + 1 < 2x^2 (x > 1$时$)$，故$\log(x^2 + 1) \le \log(2x^2) = \log 2 + \log x^2 = \log 2 + 2\log x \le 3\log x \ (x > 2)$. 所 以

$\log(x^2+1)=O(\log x)$. 又因为 x+1=O(x)，故

(x+1)log(x^2+1)=O(x log x). 而 $3x^2=O(x^2)$。由定理 2，有

$f(x)=O(\max(x\log x, x^2))$=O($x^2$), 这因为当 x>1 时，有

x log x $<x^2$.

四．大Ω和大Θ记号

1. 大Ω记号（Big-Ω notation）

定义：设 f 和 g 是从整数集合或实数集合到实数集合的函数。我们说 f(x)是$\Omega(g(x))$,是说，存在正的常数 C 和 k，使得，当 x>k 时，有$|f(x)|\geq C|g(x)|$。

例 10：函数 f(x)=$8x^3+5x^2+7$是$\Omega(g(x))$的，其中 g(x)=x^3。

由定义，存在 C=1，k=1，当 x > k = 1 时，有

$|f(x)|=8x^3+5x^2+7\geq 8x^3\geq Cx^3=x^3$.故 f(x)=$\Omega(x^3)$.

2. 大Θ记号（Big-Θ notation）

定义：设 f 和 g 是从整数集合或实数集合到实数集合的函数。我们说 f(x)是$\Theta(g(x))$，是说，f(x)是 O(g(x))且 f(x)是$\Omega(g(x))$。

例 11：（由例 5）已证前 n 个正整数的和是$O(n^2)$。问它是否具有阶n^2？

解：设 f(n)=1+2+$\cdots+n$.已知 f(n)=$O(n^2)$.取 k=2,当 n>2 时，有

1+2+$\cdots+n\geq\lceil n/2\rceil+\left(\left\lceil\frac{n}{2}\right\rceil+1\right)+\cdots+n$

$\geq\lceil n/2\rceil+\lceil n/2\rceil+\cdots+\lceil n/2\rceil$

$=(n-\lceil n/2\rceil+1)\lceil n/2\rceil$

$\geq\left(\frac{n}{2}\right)\left(\frac{n}{2}\right)$

$= n^2/4$

$\geq Cn^2$

其中 C=1/4，故$f(n) = \Omega(n^2)$。又由已知 f(n)=$O(n^2)$，有 f(n)=$\Theta(n^2)$。

定理 4：设$f(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$，其中 $a_0, a_1, \cdots, a_n$是实数且$a_n \neq 0$。那么 f(n)=$\Theta(x^n)$。

§3.3 算法的复杂性 （Complexity of algorithm）

一. 程序运行时间的测量

影响程序运行时间的因素：

1. 程序的输入的长度；
2. 编译程序生成目标代码的质量；
3. 计算机指令的性质和速度；
4. 算法的时间复杂性。

二. 评价算法运行时间的标准

运行时间作为输入长度的函数 T(n)，即对于长度为 n 的输入，算法执行运算的步数（操作的次数）T(n).

1. 最坏运行时间：

算法对长度为 n 的任何输入的最长运行时间。

2. 平均运行时间：

即在“平均”输入下，算法的运行时间。通常我们假设给定长度的各种输入概率相同。平均运行时间是在这个假设下，

运行时间的数学期望值。

3. 为什么常用最坏运行时间来估计？

最坏运行时间是算法运行时间的上界，在实际问题中，算法的运行时间常常达到这个上界。平均运行时间难以计算，假设每个输入具有相同的概率有时没意义。平均运行时间常常与最坏运行时间有相同的数量级。

三． 例子

例 1：求 3.1 节算法 1 的时间复杂度。

解：我们用元素的比较次数作为估计时间复杂度的标准。

每扫描一个元素a_i，a_i与max 比较一次，然后判定a_i是否最后一个元素又作一次比较，故作了 2 次比较。共扫描$n-1$个元素（从a_2到a_n）。最后又作了一次比较，退出循环，故一共作了$2(n-1)+1=2n-1$次比较。故比较次数是$\Theta(n)$。

例 2：估计线性查找算法的时间复杂度。

解：每查一个元素，x与 a_i比较一次，要判定a_i是否表末尾，又要比较一次，故每查一个元素a_i，要比较 2 次。如果$x=a_i$，则后面的元素就不用查了，故比较了 2i+1 次，（退出循环后要作一次比较），因为我们要估计算法的最坏运行时间，即是x不在该序列中的情形，因而要查遍所有a_i. 故要进行 2n+2 次比较。所以，最坏运行时间是 O(n).

四．理解算法的时间复杂度

1. 常见的算法时间复杂度

*见书 P226，表 1。

当一个算法的时间复杂度为 O(f(n))，其中 f(n)是一个多项式，那么，这个算法称为“好算法”(good algorithm)或者“有效算法”(efficient algorithm)。如果一个算法的时间复杂度是$\Theta\big(f(n)\big)$，而 f(n)是指数函数或阶乘函数或更大，那么，这个算法称为“坏算法”（bad algorithm)。

*易处理的问题(tractable problems)和难处理的问题(intractable problems)

如果一个问题有最坏情况是多项式时间的求解算法，那么这个问题称为易处理（易解）的；如果一个问题没有最坏情况是多项式时间的求解算法，那么这个问题称为难处理（难解）的。

2. 几种时间复杂度算法运行时间的比较

*见书 P228，表 2.

3. P, NP, NP-完全问题。

作业：

1. 描述一个算法，该算法输入 n 个不同的整数$a_1, a_2, \cdots, a_n$，并找出其中最大的偶数a_i的位置i，若其中没有偶数，则 i 为 0。

2. 找一个最小的整数 n，使得f(x)是 $O(x^n)$.

(1) f(x)=$2x^3 + x^2 \log x$

(2) f(x)=$(x^4+x^2+1)/(x^3+1)$

3. 给出以下函数 f(n)的大 O 函数 g(n)，使得 f(n)=O(g(n))。

f(n)=$(n^3+n^2\log n)(\log n+1)+(17\log n+19)(n^3+2)$

4.分析二分查找算法的时间复杂度。