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(: integers)

max := ;

FOR i := 2 TO n DO

IF max < THEN max := ;

{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; distinct integers)

i := 1;

WHILE (i DO

i := i+1;

IF i 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=1919, 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; 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 := ;

IF x > i := m+1

ELSE j := m;

END;

IF x = THEN location := i

ELSE location := 0;

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

1. 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分，再用最大的硬币，42分；这时，用第二大的硬币，分；再用第三大的硬币，分；最后用两次最小的硬币，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( values of denominations of coins, where n: a positive integer);

FOR i := 1 TO r DO

WHILE n DO

BEGIN

add a coin with value to the change;

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)

1. 大O记号(Big-O notation)
2. 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。

\*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

。

1. 例子：

例1：证明：

解：当x > 1时，有因而有，当x > 1时

。

故取C=4, k=1作为证据，有f(x)是O()。

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))表示所有是O(g(x))的函数的集合。

例2：证明：7。

解：取k=7，C=1，当x>7时，有，故。

例3：证明：不是O(n)。

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

二．大O记号的一些重要结论

\*对任意多项式f，f是，其中是f的最高次项。

定理1：设，其中都是实数。那么f(x)是。

证明：如果x>1，有

其中，.

故f(x)是。

例5：如何估计前n个正整数的和的大O函数。

解：因为,

取C=1，k=1，当时，有

，故是。

例6：给出n阶乘函数f(n)=n!的大O函数。

解：因为f(n)=n!=,

取k=1，C=1，有f(n)=n!=。

另外，,

故取C=1，k=1，有。

例7：已知对任意正整数n，有，即。利用这个不等式证明log n =。

解：取k=1，C=1，有当n > k = 1时，有，故

.

故 log n = O(n)。

\*对于任意底数b的log函数，因为有

取k=1，，有当n > k = 1时，，

故。

1. 一些常见的不同增长率的函数

1, log n , n , n log n , n!

(见书P211, 图3)

三．函数组合的增长率

1. 函数的和

定理2：假设是，是，那么

是。

证明：由定义知，存在正的常数，使得，当,时,有, .

取，当时x>k，有

其中，, g(x)=max.

推论1：假设和都是O(g(x))，那么也是O(g(x))。

2. 函数的积

定理3：假设是，是，那么

是。

证明：由定理知,存在和,使得当x>，x>时，

有。取

当x>k时，有

.

其中。

注意:当和时。

例8：求以下函数的大O函数。

解：因为log (n!)=O(n log n), 3n=O(n)，故由定理3，有

3n log n!. 而由定理2知, 。再由定理3, 有。再由定理2， 有

。

例9：给出以下函数的大O函数。

解：因为时)，故所以. 又因为x+1=O(x)，故

(x+1)log(=O(x log x). 而3)。由定理2，有

))=O( 这因为当x>1时，有

x log x .

四．大和大记号

1. 大记号（Big- 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,是说，存在正的常数C和k，使得，当x>k时，有。

例10：函数f(x)=是的，其中g(x)=。

由定义，存在C=1，k=1，当x > k = 1时，有

.故f(x)=).

2. 大记号（Big- 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，是说，f(x)是O(g(x))且f(x)是。

例11：（由例5）已证前n个正整数的和是。问它是否具有阶？

解：设f(n)=1+2+已知f(n)=.取k=2,当n>2时，有

1+2+

其中C=1/4，故。又由已知f(n)=，有f(n)=。

定理4：设，其中是实数且。那么f(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。
5. 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。

1. 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。

1. 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。

1. 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max比较一次，然后判定是否最后一个元素又作一次比较，故作了2次比较。共扫描个元素（从。最后又作了一次比较，退出循环，故一共作了次比较。故比较次数是。

例2：估计线性查找算法的时间复杂度。

解：每查一个元素，比较一次，要判定是否表末尾，又要比较一次，故每查一个元素，要比较2次。如果, 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. 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 而f(n)是指数函数或阶乘函数或更大，那么，这个算法称为（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, 并找出其中最大的偶数i，若其中没有偶数，则i为0。
2. 找一个最小的整数n，使得.

(1) f(x)=

(2) f(x)=

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)=

4.分析二分查找算法的时间复杂度。