第一章 算法及基础知识

- ✓目录
  - 算法的基本概念
  - 算法设计的一般过程
  - 算法分析
  - 递归、基本数据结构、常用数学公式

第一章 算法及基础知识

✓教学目标

- 充分理解并掌握算法的相关概念
- 理解算法设计的一般过程
- 掌握对算法复杂性进行分析的方法
- 掌握使用面向对象程序设计语言C++进行算法描述的方法
- 掌握递归的概念及运用要点
- 熟悉基本数据结构和数学公式的概念及使用方法

学习算法的重要性

- 算法与日常生活息息相关
- 算法是程序设计的根基
- 学习算法能够提高分析问题的能力
- 算法是推动计算机行业发展的关键
- 研究算法是件快乐的事情

➢ 思考：怎样学习算法？

算法的定义、特性及描述方式

- 算法的定义
  - 对于计算机科学来说，算法指的是对特定问题求解步骤的一种描述，是若干条指令的有穷序列。
- 算法的特性
  - 输入、输出、确定性、有限性、可行性
- 描述方式
  - 本书采用了面向对象程序设计语言C++
- 思考：算法与程序的区别？

算法设计的一般过程

- 充分理解要解决的问题
- 数学模型拟制
- 算法详细设计
- 算法描述
- 算法思路的正确性验证
- 算法分析
- 算法的计算机实现和测试
- 文档资料的编制

算法分析

- 算法复杂性 = 算法运行时所需要的计算机资源的量
  - 时间复杂性、空间复杂性
- 影响时间复杂性的因素（板书：查找为例）
  - 问题规模n、输入序列I、算法本身A
- 影响空间复杂性的因素
  - 算法本身、输入输出数据、辅助变量

三种情况下的复杂性（结合查找操作）

- 最好情况T_{min}（N)
  - 1次
- 最坏情况T_{max}（N)
  - N次
- 平均情况T_{avg}（N)
  - (N+1)/2

算法渐近复杂性态

- 设算法的运行时间为T(n)，如果存在T*(n)，使得

$$\lim_{n\to\infty}\frac{T(n)-T^{*}(n)}{T(n)}=0$$

就称T*(n)为算法的渐进性态或渐进时间复杂性。

举例说明（见教案）

渐进复杂性态的引入

假设算法A的运行时间表达式$T_1(n)$为：

$$T_1(n)=30n^4+20n^3+40n^2+46n+100$$

算法B的运行时间表达式$T_2(n)$为：

$$T_2(n)=1000n^3+50n^2+78n+10$$

思考：（1）算法A效率高还是算法B效率高？

（2）为什么引入算法的渐进复杂性？

简化

渐近意义下的记号（O、 Ω、Θ）

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**（*举例*）

- $O(g(n)) = \{ f(n) \mid$ 存在正常数c和自然数n_0使得对所有$n \geq n_0$有：$0 \leq f(n) \leq cg(n) \}$

（2）渐近下界记号**Ω**（*举例*）

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

有用的规则

- O(f)+O(g)=O(max(f，g))；
- O(f)+O(g)=O(f+g)；
- O(f)O(g)=O(fg)；
- 如果g(n)=O(f(n))，则O(f)+O(g)=O(f)；
- O(Cf(n))=O(f(n))，其中C是一个正的常数；
- f=O(f)。

（3）渐近精确界记号Θ（举例）

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

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

算法的运行时间T(n)建立的依据

- 非递归算法
  - （a）选择某种能够用来衡量算法运行时间的依据；
  - （b）依照该依据求出运行时间T(n)的表达式；
  - （c）采用渐进符号表示T(n)；
  - （d）获得算法的渐进时间复杂性，进行进一步的比较和分析。
  - 举[例1-4]、[例1-5]进行说明

递归

- 子程序（或函数）直接调用自己或通过一系列调用语句间接调用自己，称为递归。直接或间接调用自身的算法称为递归算法。
- 采用递归算法来求解问题的一般步骤：
  - 分析问题，寻找递归关系
  - 找出停止条件
  - 构建函数体

n的阶乘

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

停止条件

递归关系

停止条件与递归关系是递归函数的两个要素，递归函数只有具备了这两个要素，才能在有限次计算后得出结果。

排列问题

- 问题描述
  - n个元素，它们的编号为1,2,…,n，排列问题的目的是生成这n个元素的全排列。
- 算法设计思路
  - 将规模为n的排列问题转化为规模为n-1的排列问题。
  - 将规模为n-1的排列问题转化为规模为n-2的排列问题
  - 将问题规模一级一级降至1，1个元素的排列是它本身，此时到达递推的停止条件。数组中的元素即为1个排列，然后进行回归依次得到其它的排列。

排列问题

- 算法描述
  - 使用递归技术来解决全排列问题。
  - 假定排列算法perm(A,k,n)的功能是：生成数组A后面k个元素的排列。当k=1时，只有一个元素，已构成一个排列。当1<kn，可由算法Perm(A,k-1,n)生成数组A后面k-1个元素的排列，为完成数组A后面k个元素的排列，需要逐一将数组第n-k个元素与数组中第n-k~n-1个元素互换，每互换一次，就执行一次perm(A,k-1,n)。

递归算法复杂性分析

- 递归算法的时间复杂性分析
  - 决定采用哪个（或哪些）参数作为输入规模的度量；
  - 找出对算法的运行时间贡献最大的语句作为基本语句；
  - 检查一下，对于相同规模的不同输入，基本语句的执行次数是否不同。如果不同，则需要从最好、最差及平均三种情况进行讨论；
  - 对于选定的基本语句的执行次数建立一个递推关系式，并确定停止条件；
  - 通过计算该递推关系式得到算法的渐进时间复杂性。
- 递归算法的空间复杂性分析
  - 递归深度

以排列问题为例分析递归算法的复杂性

- 当k=1时，已构成一个排列，第一个for循环需要执行n次操作将排列输出；当k=n时，第二个for循环的循环体，对perm(A,k-1,n)执行n次调用。因此，排序算法perm对应的递归定义式为：

$$T(n)=\begin{cases} O(1) & n=1 \\ nT(n-1) & n>1 \end{cases}$$

- 采用后向代入法计算可得到通项公式：

T(n) =nT(n-1)

- =.....
- =n(n-1)(n-2) .....2T(1)
- =n!

所以，全排列算法perm的时间复杂性为O(n!)。

- 回顾常用的数据结构

算法渐近复杂性分析中常用数学公式

（1）对数公式

（2）组合公式

（3）求和公式

（4）向下取整和向上取整公式

（1）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 $a = b^{\log_b a}$

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

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

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

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

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

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

取整函数的若干性质

- $x-1 < \lfloor x \rfloor \le x \le \lceil x \rceil < x+1$；
- $\lfloor n/2 \rfloor + \lceil n/2 \rceil = n$;
- 对于$n \ge 0$，$a,b>0$，有：
- $\lceil \lceil n/a \rceil /b \rceil = \lceil n/ab \rceil$;
- $\lfloor \lfloor n/a \rfloor /b \rfloor = \lfloor n/ab \rfloor$;
- $\lceil a/b \rceil \le (a+(b-1))/b$;
- $\lfloor a/b \rfloor \ge (a-(b-1))/b$;
- $f(x)=\lfloor x \rfloor$, $g(x)=\lceil x \rceil$为单调递增函数。