算法设计与分析

Algorithm Design and Analysis

第一章 绪论

1.1 算法的基本特征

算法是一系列解决问题的过程，能够对给定的输入在有限的时间内获得所要求的输出。

算法应具备的五个基本特征为：

①有穷性：必须在执行有穷步后终止

②确定性：算法的每一步均有明确的意义

③输入: 0或多个

④输出： 1个或多个

⑤可行性：每一步的执行时间都是有限的

1.2 算法研究的意义

例1: 排序10^6个数据的数组

即：输入的规模$n=10^6$

	运算速度	算法效率	排序时间
计算机A	10^9次/秒	$2n^2$	$2\times(10^6)^2/10^9=2000$秒
计算机B	10^7次/秒	$50n\lg n$	$5\times10^6\times\lg 10^6/10^7\approx100$秒

例2：五个算法的运行时间分别为：$T_1(n)=33n$，$T_2(n)=46n\lg n$, $T_3(n)=13n^2$，$T_4(n)=3.4n^3$,$T_5(n)=2^n$，其中n为输入的规 模，计算机处理速度为1,000,000次/秒。

n	$T_1(n)$	$T_2(n)$	$T_3(n)$	$T_4(n)$	$T_5(n)$
10	0.00033	0.0015	0.0013	0.0034	0.001
100	0.0033	0.03	0.13	3.4	4×10^{16}年
1000	0.033	0.45	13	94小时	
10000	0.33	6.1	22分	39天	

1.3 算法的描述形式(Pseudocode)

例3：插入排序算法

```
Insert-sort(A)
for  j←2 to length[A]
do  key←A[j]
    ▸A[j]插入到已排好序的子数组A[1..j-1]中
      i←j-1
      while i>0 and A[i]>key
      do    A[i+1]←A[i]
                i←i-1
      A[i+1]←key
```

伪语言说明(P11)

1.4 算法分析

包括时间复杂度和空间复杂度的分析

传统分析方法：$\sum$(指令的一次执行时间×一条指令的执行次数)

其中：C_i表示每条指令一次执行的时间

t_j表示第j次循环while语句执行的次数

指令执行时间	执行次数
C_1	n
C_2	$n-1$
C_3	$n-1$
C_4	$\sum_{j=2}^{n} t_j$
C_5	$\sum_{j=2}^{n} (t_j - 1)$
C_6	$\sum_{j=2}^{n} (t_j - 1)$
C_7	$n-1$

$$T(n) = C_1 n + C_2(n-1) + C_3(n-1) + C_4 \sum_{j=2}^{n} t_j$$

$$+ C_5 \sum_{j=2}^{n}(t_j - 1) + C_6 \sum_{j=2}^{n}(t_j - 1) + C_7(n-1)$$

最好情况：输入数据状态为升序，此时A[i]≤key，所以t_j=1

$$T(n) = C_1n+C_2(n-1)+C_3(n-1)+C_4(n-1)+C_7(n-1)$$
$$= (C_1+C_2+C_3+C_4+C_7)n-(C_2+C_3+C_4+C_7)$$
$$= An+B$$

最坏情况：输入数据状态为倒序，此时A[i]>key，所以t_j=j

$$T(n) = C_1n+C_2(n-1)+C_3(n-1)+C_4$$
$$= \tfrac{1}{2}(C_4+C_5+C_6)n^2+(C_1+C_2+C_3+\tfrac{1}{2}C_4-\tfrac{1}{2}C_5$$
$$-\tfrac{1}{2}C_6+C_7)n-(C_2+C_3+C_4+C_7)$$
$$= An^2+Bn+C$$

平均情况：

是指在不同的输入条件下算法平均的运行时间。

直观分析：插入A[j]时需比较j/2次，即$t_j=j/2$

所以T(n)应和n^2相关。

对于三种情况下算法的时间复杂度，通常会更多地关注最坏情况下的时间复杂度，原因为:

①是算法运行时间的上限，也是算法改进的一个判断标准

②在很多时间问题中，由于缺少足够的信息，因此算法往往运行在最坏请况

③算法的平均情况下的时间往往与最坏情况下的时间复杂度相同

1.5 增长速率(order of growth)

一个算法的时间是由其增长速率所决定的，所以对于算法时间复杂度的研究主要侧重的是算法的渐进性能，即当问题的输入规模n很大时算法的运行效率。对于一个多项式时间的算法，如：$T(n) = An^2+Bn+C$，当n很大时，$T(n)$的增长速率是由最高项n^2所决定。

1.6 算法正确性分析

循环不变式(loop Invariant)方法:先确定算法的循环不变式，然后检查此循环不变式在算法执行期间是否满足以下三个条件：

①初始状态(Initialization):循环前循环不变式为真

②保持状态(Maintance):如果本次循环前循环不变式为真，则本次循环后依然为真

③终止状态(Termination):退出循环时，循环不变式为真

例:插入排序算法的正确性分析:

循环不变式为：子数组A[1..j-1]始终为排好序的状态

① 初始状态(Initialization):

∵ 循环开始前j=2

∴ 进入循环前子数组A[1..j-1]只有一个元素，此时子数组为排好序的状态

② 保持状态(Maintance):

∵ while循环前i=j-1，若子数组A[1..i]=A[1..j-1]为真，循环开始时，插入排序的过程是依次后移A[j-1]、A[j-2]、…并把新的关键字插入到合适的位置，此时子数组A[1..j-1]扩充为A[1..j]且为排好序的状态

∴ j++后循环不变式A[1..j-1]为真

③ 终止状态(Termination):退出循环时，

∵ j=n+1,此时子数组A[1..j-1]=A[1..n]包含所有元素且为排好序的状态

∴ 循环不变式为真

第二章 渐进符号(Asympototic notation)

2.1 Θ符号(渐近符号)

def: $\Theta(g(n)) = \{ f(n)$:存在大于0的常数C_1、C_2和n_0，使得对于所有的$n \geq n_0$，都有$0 \leq C_1 g(n) \leq f(n) \leq C_2 g(n)\}$

记为：$f(n) = \Theta(g(n))$。

注意点:

① 常数C_1、$C_2>0$

② 只需要存在某个C_1、$C_2>0$即可，不要求对任意的C_1、C_2。

例1： $\mathbf{f(n)=\frac{1}{2}n(n-1)}$**，证** $\mathbf{f(n)=\Theta(n^2)}$

proof：令 $\mathbf{g(n)=n^2}$**，就是要找到两个正常数** $\mathbf{C_1}$**和** $\mathbf{C_2}$**，满足**

$$\mathbf{C_1 g(n)\le f(n)\le C_2 g(n)}$$

例2： $f(n)=\frac{1}{2}n^2+n$，证 $f(n)=\Theta(n^2)$

proof： $C_1 n^2\le f(n)\le C_2 n^2$

2.2　O符号（渐近上界）

def: $O(g(n))=\{f(n)$:存在大于0的常数C、n_0，使得对于所有的$n \geq n_0$时，都有$0 \leq f(n) \leq Cg(n)\}$

记为：$f(n)=O(g(n))$

例3：证明$f(n)=an+b=O(n)$，其中$a>0$。

Proof: 即要证$f(n)=an+b \leq Cn$

例4：证明$f(n)=2n^2+n=O(n^3)$

proof: 即要找到一个正常数C，使得$f(n) \leq Cn^3$

特别提示，大O既可表示紧上界也可表示松上界。

2.3 Ω符号(渐进下界)

def: $\Omega(g(n)=\{f(n)$:存在正常数C和n_0，使得对于所有的$n \geq n_0$，都有$0 \leq Cg(n) \leq f(n)\}$

记为：$f(n) = \Omega(g(n))$

例5：证明$f(n)=3n^2 = \Omega(n)$

proof: 即要证 $Cn \leq 3n^2$

Θ,O,Ω符号之间存在关系如下：

定理3.1（P29）

对于任意的函数f(n)和g(n)，当且仅当$f(n)=O(g(n))$且$f(n)=\Omega(g(n))$时，$f(n)=\Theta(g(n))$。

2.4 利用极限比较函数间的增长速度

对于函数f(n),g(n),如果

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\begin{cases}0 & \Rightarrow f(n)<g(n) \quad \Rightarrow f(n)=O(g(n)) \\ C & \Rightarrow f(n)=g(n) \quad \Rightarrow f(n)=\Theta(g(n)) \\ \infty & \Rightarrow f(n)>g(n) \quad \Rightarrow f(n)=\Omega(g(n))\end{cases}$$

例6: $f(n)=\frac{1}{2}n(n-1),\ g(n)=n^2$

例7: $f(n)=\lg n,\ g(n)=\sqrt{n}$