Analysis of Algorithm Complexity: Time and Space III

Instructor: Shizhe Zhou

Course Code:00125401

所谓算法分析

一个算法执行时间，从理论上是不能算出来的，必须通过依据该算法编制的程序上机运行测试才能知道。有两种方法:事后统计的方法和事前分析估算的方法

和算法执行时间相关的因素：

1．算法选用的策略

2．问题的规模（处理的数据量）

3．编写程序的语言

4．编译程序产生机器代码的质量

5．计算机执行指令的速度

算法复杂度问题：

- 一般是指问题随规模的增长算法所需消耗的运算时间和内存空间的增长趋势。
- 因此不考虑计算机本身硬件的特质，一般也忽略算法所消耗的与问题规模无关的固定量的计**算与存储空间。**

算法复杂度的考察方法

☜

- 考察一个算法的复杂度，一般考察的是当问题复杂度n的增加时，运算所需时间、空间代价f(n)的上下界。（***Asymptotic upper or lower bound***）
- **进一步而言、又分为最好情况、平均情况、最**坏情况三种情况。通常最坏情况往往是我们最关注的。

时间复杂度

算法的执行时间如何计算？

$$\sum_{i=1}^{n} 原操作\, i\, 的执行次数 \times 原操作\, i\, 的执行时间$$

操作（简单操作）:如赋值操作、转向操作、比较操作等等.

既然执行一种原操作所需的时间与算法无关，那么我们只讨论影响运行时间的另一个因素——原操作被*执行的次数*。

显然，在一个算法中，执行简单操作的次数越少，则运行时间也越少.

所以: 算法中包含简单操作的次数的多少叫做时间复杂度.

为便于计算，对这一时间复杂度大多采用一种近似的形式来描述，即采用基本语句执行次数的数量级来表示时间复杂度。

数量级是这样定义的：

如果变量**n**的函数**f(n)**和**g(n)**满足：

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=k(k\neq 0)$$

则称**f(n)**和**g(n)**是同一数量级的

设： $f(n)=4n^2+2n+1$ $\quad g(n)=n^2$

p28

符号O(ɒmɨkrɒn)

定义：存在常数c和N， 当$n \geq N$时， 有$g(n) \leq cf(n)$

则称函数$g(n)$相对$f(n)$是 $O(f(n))$

渐近时间复杂度

1. 从上限上进行约束(软上限，符号o为硬上限).
2. 符号O中没有常数(带有常数也没有意义).

表 3.1 在不同假定（$n = 1000$）下的运行时间（s）

运行时间	时间 1 1000 步/秒	时间 2 2000 步/秒	时间 3 4000 步/秒	时间 4 8000 步/秒
lb$n$	0.010	0.005	0.003	0.001
n	1	0.5	0.25	0.125
n lb$n$	10	5	2.5	1.25
$n^{1.5}$	32	16	8	4
n^2	1 000	500	250	125
n^3	1 000 000	500 000	250 000	125 000
1.1^n	10^{39}	10^{39}	10^{38}	10^{38}

Log(1000,2)= 9.965784284662087

定理3.1

对于所有常数 $c>0$ 和 $a>1$，以及所有单调递增函数 $f(n)$，有

$$(f(n))^c = O(a^{f(n)})。$$

换句话说，一个指数函数要比一个多项式函数增长得快。□

这条规则可用于许多函数的比较。例如，如果我们用 n 来替换定理 3.1 中的 $f(n)$，则对于所有常数 $c>0$ 和 $a>1$，有

$$n^c = O(a^n)。\qquad (3.1)$$

另一个例子，可以用 $\log_a n$ 来替换 $f(n)$。对于所有常数 $c>0$ 和 $a>1$，有

$$(\log_a n)^c = O(a^{\log_a n}) = O(n)。\qquad (3.2)$$

p30

符号 Ω

定义：存在常数c和N，当$n \ge N$时，有$g(n) \ge cf(n)$

则有$g(n) = \Omega(f(n))$.

1. 从下限上进行约束.

符号 Θ

如果一个特定函数 $f(n)$ 同时满足 $f(n)=O(g(n))$ 和 $f(n)=\Omega(g(n))$，则称 $f(n)=\Theta(g(n))$。例如，$5n\,\mathrm{lb}\,n-10=\Theta(n\log n)$。（在表达式 $\Theta(n\log n)$中对数的底数可被忽略，这是因为不同的底数仅仅是为对数带来常数因子的改变。）用来证明 O 部分和 Ω 部分的常数项不需要一致。

对应关系：$O"\le"$，$\Omega"\ge"$，$\Theta"="$

$o"<"$，$\omega">"$

符号 o, ω

如果有 $\lim\limits_{n\to\infty}\dfrac{f(n)}{g(n)}=0$, 可称 $f(n)=o(g(n))$,

反过来，若 $g(n)=o(f(n))$, 则称 $f(n)=\omega(g(n))$.

□ **定理 3.3**

对于所有常数 $c>0$ 和 $a>1$，以及所有单调递增函数 $f(n)$，有 $(f(n))^c=o(a^{f(n)})$。换句话说，一个指数式函数要比一个多项式函数增长得快。

- 时间复杂度：

    往往被**最大的循环**决定.

- 空间复杂度：

    运行所需的临时存储空间(峰值)，一般不将输入输出所需空间计算在内.

类似于时间复杂度，我们也讨论最坏的情况，即内存消耗的峰值.

计算下面求累加和程序段的时间复杂性

（1） sum=0； （1次）

（2） for(i=1;i<=n;i++) （n次 ）

（3） for(j=1;j<=n;j++) （n^2次 ）

（4） sum++； （n^2次 ）

解：$T(n)=2n^2+n+1=O(n^2)$

常见C++循环的时间复杂度

[1]　X = X + 1 ;　　**O(1)**

[2]　for (i = 1; i < n; i + +)
**　　　X++;**　　**O(n)**

[3]　for (i = 1; i <= n; i + +)
**　　for(j = 1; j <= i; j + +)**
**　　　X + +;**　　**O(n^2)**

$$\frac{n^2+3n}{2}$$

分类讨论

- 求和关系
- 递推关系
  - **The Substitution method** $\boxed{T(n) = 4T(n/2) + n}$
    - Special case for Fibonacci-type recurrence

      $\boxed{\text{T(n) = aT(n-1)+bT(n-2), T(1)=s, T(2)=t;}}$
  - **The Recursion-tree method** 为subsitution method提供猜测
  - **The Master method** $\boxed{T(n) = \sum_{i=1}^{k} a_i T(n/b_i) + f(n)}$
  - The General method for all-history-related type

    $\boxed{T(n) = a\,T(n/b) + f(n)}$

求和关系

- 例子3.2:

$$F(n)=\sum_{i=1}^{n}2^{i}=1+2+4+\cdots+2^{n}$$

利用相邻项因子差2，做2F(n)−F(n)=$2^{n+1}-1$

- 例子3.3:

$$G(n)=\sum_{i=1}^{n}i2^{i}=1\cdot 2^{1}+2\cdot 2^{2}+3\cdot 2^{3}+\cdots+n\cdot 2^{n}$$

- 例子3.4:

$$G(n)=\sum_{i=1}^{n}i2^{n-i}=1\cdot 2^{n-1}+2\cdot 2^{n-2}+3\cdot 2^{n-3}+\cdots+n\cdot 2^{0}$$

Fibonacci Type – A Special Case

Fibonacci Type Recurrence

- T(n) = aT(n-1)+bT(n-2), T(1)=s, T(2)=t;
- 利用固定的推导过程: P35~P36

Fibonacci型递推关系的通项等式:

$$F(n)=F(n-1)+F(n-2),\quad F(1)=1,\quad F(2)=1。$$

$$c2^n = c2^{n-1} + c2^{n-2}$$

$$a^n = a^{n-1} + a^{n-2}$$

待定系数法求取初始值

对F(1)=1和F(2)=1使用待定系数法, 可求出a_1 , a_2和c_1 , c_2

特征等式适用于这一类递推关系:

$F(n)=b_1F(n-1)+b_2F(n-2)+...+b_kF(n-k)$

However,需求解高次方程!

替换法在证明时的'误区'

Example: $T(n) = 4T(n/2) + n$

We shall prove that $T(n) = O(n^2)$.

Assume that $T(k) \le ck^2$ for $k < n$:

$$
\begin{aligned}
T(n) &= 4T(n/2) + n \\
&\le 4cn^2 + n \\
&= O(n)
\end{aligned}
$$

Wrong! We must prove the I.H.

$$
\begin{aligned}
&= cn^2 - (-n) \quad [\text{ desired} - \text{residual }] \\
&\le cn^2
\end{aligned}
$$

for ***no*** choice of $c > 0$. Lose!

不可以在证明中直接用O表达式来替换T(n)！

正确的证明过程

IDEA: Strengthen the inductive hypothesis.

- ***Subtract*** a low-order term.

Inductive hypothesis: $T(k) \le c_1 k^2 - c_2 k$ for $k < n$.

$$
\begin{aligned}
T(n) &= 4T(n/2) + n \\
&\le 4(c_1(n/2)^2 - c_2(n/2)) + n \\
&= c_1 n^2 - 2c_2 n + n \\
&= c_1 n^2 - c_2 n - (c_2 n - n) \\
&\le c_1 n^2 - c_2 n \quad \text{if } c_2 > 1.
\end{aligned}
$$

Pick c_1 big enough to handle the initial conditions.