算法设计与分析

1. 算法概述

1.0 学习要点

- 理解算法的概念。
- 掌握算法的计算复杂性概念。
- 掌握算法渐近复杂性的数学表述。
- 掌握复杂性的阶及表示

1.1 什么是算法

- **计算问题**

  输入和**正确输出**之间的二元关系

- **问题的描述**
  - 描述输入和正确输出
  - 输入通常是一个参数
  - 通常不列举所有输入对应的正确输出
  - 通常提供输出所要满足的谓词（性质）

- **问题实例**

  参数的一组赋值可得到问题的一个实例

1.1 什么是算法

- **算法**

  有限条指令的序列

  这个指令序列确定了解决某个问题的一系列运算或操作

- **算法 A 解决问题 P**

  把问题 P 的任何实例作为算法 A 的输入：

  - 每步计算是确定性的
  - A 能够在有限步停机
  - 输出该实例的正确的解

1.1 什么是算法

- **算法的五个特性**
  - 输入： 算法开始执行执行之前指定初始值（有零个或多个输入
  - 输出：产生与输入相关的量（至少有一个）。
  - 确定性：每一条规则都是明确、无二义的。
  - 有限性：求解问题的运算序列，必须在有限的计算步后停止。
  - 可行性：每一计算步都是基本的、可实现的。

什么是基本计算?

1.1 什么是算法

- 算法 VS 程序
  - 程序是算法用某种程序设计语言的具体体现
  - 程序=算法 + 数据结构
  - 程序可以不满足算法的有限性要求。

    例如：操作系统，是一个在无限循环中执行的程序，因而不是一个算法。
  - 算法是理论计算机科学的一个分支，而程序设计是软件开发的一个必要步骤属于工程领域

1.2 如何分析和描述算法的效率

- 算法时间复杂度: 针对指定基本运算，算法所做运算次数
- 基本运算: 比较, 加法, 乘法, 置指针, 交换…
- 输入规模：输入串编码长度
  通常用下述参数度量：数组元素多少，调度问题的任务个数，图的顶点数与边数等.
- 算法基本运算次数可表为输入规模的函数
- 给定问题和基本运算就决定了一个算法类

例：输入规模

- 排序：数组中元素个数 n
- 检索：被检索数组的元素个数 n
- 整数乘法：两个整数的位数 m, n
- 矩阵相乘：矩阵的行列数 i, j, k
- 图的遍历：图的顶点数 n, 边数 m

  …

基本运算

- 排序: 元素之间的比较
- 检索: 被检索元素x与数组元素的比较
- 整数乘法: 每位数字相乘(位乘) 1 次
  m位和n位整数相乘要做mn次位乘
- 矩阵相乘: 每对元素乘1 次
  $i \times j$矩阵与$j \times k$ 矩阵相乘要做$i\,jk$ 次乘法
- 图的遍历: 置指针

…

三种典型的复杂性

- 时间复杂性细化--3种典型的复杂性：
  最坏、最好、平均复杂性
- 通常使用T(I)表示输入规模为I时的问题的算法复杂性

三种典型的复杂性

（1）**最坏情况**下的时间复杂性

$T_{max}(n) = \max\{ T(\mathrm{I}) \mid \mathrm{size}(\mathrm{I})=n \}$

（2）**最好情况**下的时间复杂性

$T_{min}(n) = \min\{ T(\mathrm{I}) \mid \mathrm{size}(\mathrm{I})=n \}$

（3）**平均情况**下的时间复杂性

$$T_{avg}(n) = \sum_{size(I)=n} p(I)T(I)$$

其中I是问题的规模为n的实例，p(I)是实例I出现的概率。

算法渐近复杂性

- 假定当 $n\to\infty$时, $T(n)\to\infty$;
- 取$t(n)$，使$(T(n) - t(n))/ T(n)\to 0$，当 $n\to\infty$;
- $t(n)$是$T(n)$的渐近性态，为算法的渐近复杂性。
- 在数学上， $t(n)$是$T(n)$的渐近表达式，是$T(n)$略去低阶项留下的主项。它比$T(n)$ 简单。

几个复杂性参照函数

C

Log n

$\log^2 n$

n

n log n

n^2

n^3

2^n

n!

增长的阶

- 不同的计算过程在消耗计算资源的速率上可能存在着巨大差异，描述这种差异的一种方法是用**增长的阶**的记法，以便我们理解在输入变大时，某一计算过程所需资源的粗略度量情况。

- **令n是一个参数，它能作为问题规模的一种度量，令f(n)是一个计算过程在处理规模为n的问题时所需要的资源量。对某函数g(n)，称f(n)具有复杂性阶Δ(g(n))，记为 f(n)= Δ(g(n))，其中Δ是O、Ω、o、Θ等。** ω
- **含义见下面的几页。**

渐进符号及其意义：O(f)，Ω(f)，θ(f)，o(f)

- 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

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

（2）渐近下界记号Ω

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

（3）**高阶记号o**

$o(g(n)) = \{ f(n)$ | 对于任何正常数$c>0$，存在正数和$n_0>0$使得对所有$n \geq n_0$有：$0 \leq f(n)<cg(n) \}$

等价于 $f(n) / g(n) \to 0$，as $n \to \infty$。

- $f(n)$ 的阶比 $g(n)$的阶低

（4）**同阶记号Θ**

$\Theta(g(n)) = \{ f(n)$ | 存在正常数c_1,c_2和n_0使得对所有$n \geq n_0$有：$c_1g(n) \leq f(n) \leq c_2g(n) \}$

复杂性阶符号的理解举例

- 称f(n)具有Θ(g(n))，记为 f(n)= Θ(g(n))，如果存在与n无关的整数c_1和c_2，使得：

$$c_1 g(n) \le f(n) \le c_2 g(n)$$

对于任何足够大的n都成立。

举例

求n！ 递归算法f(n)

f(n)=1，1，2，6，24，…，nf(n-1)，…

```
f(n){
    if(n==0 || n==1) return 1;
    return n*f(n-1);
}
```

求阶乘的递归算法时间和空间需求增长都是Θ(n)；但对于迭代的算法，时间增长为Θ(n)，空间增长为Θ(1)（常数）。

增长的阶

- 对于线性的过程，问题规模扩大一倍，资源增长一倍；
- 对于指数增长的过程，问题规模加1，资源就增大一个常数倍；
- 对于对数增长的过程，问题规模增大一倍，所需资源增长一个常数

各记号在等式和不等式中的意义

- $f(n)=\Theta(g(n))$的确切意义是：$f(n)\in\Theta(g(n))$。
- 一般情况下，等式和不等式中的渐近记号$\Theta(g(n))$表示$\Theta(g(n))$中的某个函数。
- 例如：$2n^2+3n+1=2n^2+\theta(n)$ 表示
- $2n^2+3n+1=2n^2+f(n)$，其中$f(n)$ 是$\Theta(n)$中某个函数。
- 等式和不等式中渐近记号O,o和Ω的意义是类似的。

各记号的记忆性比较

- $f(n)= O(g(n))$ $\approx$ $f(n)$阶不超过$g(n)$阶
- $f(n)= \Omega(g(n))$ $\approx$ $f(n)$阶以$g(n)$阶为下界
- $f(n)= \Theta(\mathrm{g}(\mathrm{n}))$ $\approx$ $f(n)$与$g(n)$等同
- $f(n)= o(g(n))$ $\approx$ $f(n)$阶小于$g(n)$阶

渐近分析记号的若干性质--传递性

（1）传递性：

$f(n)=\Theta(g(n))$， $g(n)=\Theta(h(n)) \Rightarrow f(n)=\Theta(h(n))$；

$f(n)=O(g(n))$， $g(n)=O(h(n)) \Rightarrow f(n)=O(h(n))$；

$f(n)=\Omega(g(n))$， $g(n)=\Omega(h(n)) \Rightarrow f(n)=\Omega(h(n))$；

$f(n)=o(g(n))$， $g(n)=o(h(n)) \Rightarrow f(n)=o(h(n))$；

渐近分析记号的若干性质

--反身性、对称性、互对称性

（2）反身性：

$f(n)=\Theta(f(n))$；

$f(n)=O(f(n))$；

$f(n)=\Omega(f(n))$.

（3）对称性：

$f(n)=\Theta(g(n)) \Leftrightarrow g(n)=\Theta(f(n))$.

（4）互对称性：

$f(n)=O(g(n)) \Leftrightarrow g(n)=\Omega(f(n))$ ；

渐近分析记号的若干性质
--算术运算

（5）算术运算：

- $O(f(n))+O(g(n)) = O(\max\{f(\mathrm{n}),g(n)\})$；
- $O(f(n))+O(g(n)) = O(f(\mathrm{n})+g(n))$；
- $O(f(n))*O(g(n)) = O(f(\mathrm{n})*g(n))$；
- $O(cf(n)) = O(f(\mathrm{n}))$；C是正常数
- $g(n)= O(f(n)) \Rightarrow O(f(n))+O(g(n)) = O(f(\mathrm{n}))$ 。

规则$O(f(n))+O(g(n)) = O(\max\{f(\mathrm{n}),g(n)\})$ 的证明

对于任意$f_1(n) \in O(f(n))$，存在正常数c_1和自然数n_1，使得对所有$n \geq n_1$，有$f_1(n) \leq c_1 f(n)$。

类似地，对于任意$g_1(n) \in O(g(n))$，存在正常数c_2和自然数n_2，使得对所有$n \geq n_2$，有$g_1(n) \leq c_2 g(n)$。

令$c_3=\max\{c_1, c_2\}$， $n_3 =\max\{n_1, n_2\}$，$h(n)= \max\{f(\mathrm{n}),g(n)\}$。

则对所有的 $n \geq n_3$，有

$$
\begin{aligned}
f_1(n) + g_1(n) &\leq c_1 f(n) + c_2 g(n) \\
&\leq c_3 f(n) + c_3 g(n) = c_3(f(n) + g(n)) \\
&\leq c_3 2 \max\{f(\mathrm{n}),g(n)\} \\
&= 2c_3 h(n) = O(\max\{f(\mathrm{n}),g(n)\}).
\end{aligned}
$$

算法渐近复杂性分析中常用函数

（1）取整函数

$\lfloor x \rfloor$：不大于x的最大整数；

$\lceil x \rceil$：不小于x的最小整数。

取整函数的若干性质

1. $x-1 < \lfloor x \rfloor \le x \le \lceil x \rceil < x+1$；
2. $\lfloor n/2 \rfloor + \lceil n/2 \rceil = n$；
3. 对于$n \ge 0$，$a,b>0$，有：

   $\lceil \lceil n/a \rceil /b \rceil = \lceil n/ab \rceil$；

4. $\lfloor \lfloor n/a \rfloor /b \rfloor = \lfloor n/ab \rfloor$；
5. $f(x)=\lfloor x \rfloor$, $g(x)=\lceil x \rceil$为单调递增函数。

多项式函数

- $p(n) = a_0 + a_1 n + a_2 n^2 + \ldots + a_d n^d$; $a_d > 0$;
- $p(n) = \Theta(n^d)$;
- $f(n) = O(1) \Leftrightarrow f(n) \le c$;
- $k \ge d \Rightarrow p(n) = O(n^k)$;
- $k \le d \Rightarrow p(n) = \Omega(n^k)$;
- $k > d \Rightarrow p(n) = o(n^k)$;

指数函数、对数函数

$a>1 \Rightarrow \quad \lim_{n\to\infty} \frac{n^b}{a^n} = 0 \Rightarrow n^b = o(a^n)$

记号： $\log n = \log_2 n$; $\lg n = \log_{10} n$; $\ln n = \log_e n$;

$|x| \le 1 \Rightarrow \quad \ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5} - \cdots.$

for $x > -1$, $\frac{x}{1+x} \le \ln(1+x) \le x$

对任何 $a > 0$, $\lim_{n\to\infty} \frac{\log^b n}{n^a} = 0 \quad \Rightarrow \log^b n = o(n^a)$

阶乘函数

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

$$n! = 1 \cdot 2 \cdot 3 \cdots n$$

Stirling公式

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right)$$

https://en.wikipedia.org/wiki/Stirling%27s_approximation

算法分析的基本法则

- 非递归算法：

（1）for / while 循环

循环体内计算时间*循环次数；

（2）嵌套循环

循环体内计算时间*所有循环次数；

（3）顺序语句

各语句计算时间相加；

（4）if-else语句

if语句计算时间和else语句计算时间的较大者。

求幂--传统方法

$\mathbf{b^n = b * b^{n-1}}$

$\mathbf{b^0 = 1}$

```
pow(b,n){
  if(n==0) return 1;
  return b*pow(b,n-1);
}
```

时间和空间增长$\Theta(n)$

求幂--二分

$b^n = (b^{n/2})^2$ n是偶数

$b^n = b*b^{n-1}$ n是奇数

```
fast_pow(b,n){
    if(n==0) return 1;
    if(n%2==0) square(fast_pow(b,n/2));
    else b*fast_pow(b,n-1);
}
```

时空增长的阶为 $\Theta(\log n)$