信息安全数学基础

韩　琦

计算机科学与技术学院

Overview

Detailed overview

概述

本章介绍计算复杂性，计算复杂性在计算机领域是一个非常重要的概念，表示了程序运行的代价，在信息安全领域中，计算复杂性又被赋予了与安全有关的内涵，如加密算法的安全性是以某些困难问题为基础的，这些问题的困难性也就是指问题的计算复杂性。

所谓"计算复杂度"，就是用计算机求解问题的难易程度。其度量标准：一是计算所需的步数或指令条数(这叫时间复杂度)，二是计算所需的存储单元数量(这叫空间复杂度)。一个算法的时间复杂度和空间复杂度通常与该算法的输入长度n紧密相关，他们都是输入长度n的函数，分别记为$T(n)$和$S(n)$。通常，当涉及算法的复杂度时，主要分析算法的时间复杂度。

概述

本章介绍计算复杂性，计算复杂性在计算机领域是一个非常重要的概念，表示了程序运行的代价，在信息安全领域中，计算复杂性又被赋予了与安全有关的内涵，如加密算法的安全性是以某些困难问题为基础的，这些问题的困难性也就是指问题的计算复杂性。

所谓"计算复杂度"，就是用计算机求解问题的难易程度。其度量标准：一是计算所需的步数或指令条数(这叫时间复杂度)，二是计算所需的存储单元数量(这叫空间复杂度)。一个算法的时间复杂度和空间复杂度通常与该算法的输入长度n紧密相关，他们都是输入长度n的函数，分别记为$T(n)$和$S(n)$。**通常，当涉及算法的复杂度时，主要分析算法的时间复杂度。**

Detailed overview

算法的概念

给定一个计算问题P，问题P的复杂度是由求解P的算法的时间复杂度所确定的。

所谓**算法**，是指用计算机求解一类问题的方法。

一个问题是**算法可解的**或**可解的**是指：能够编制计算机程序，只要让该程序运行足够长的时间和使用足够多的存储空间，那么该程序对任何输入能够产生正确的回答。

图灵证明了这样一个事实：有无穷多个问题是不可解的。

算法的概念

给定一个计算问题P，问题P的复杂度是由求解P的算法的时间复杂度所确定的。

所谓**算法**，是指用计算机求解一类问题的方法。

一个问题是**算法可解的**或**可解的**是指：能够编制计算机程序，只要让该程序运行足够长的时间和使用足够多的存储空间，那么该程序对任何输入能够产生正确的回答。

图灵证明了这样一个事实：有无穷多个问题是不可解的。

算法的概念

给定一个计算问题P，问题P的复杂度是由求解P的算法的时间复杂度所确定的。

所谓**算法**，是指用计算机求解一类问题的方法。

一个问题是**算法可解的**或**可解的**是指：能够编制计算机程序，只要让该程序运行足够长的时间和使用足够多的存储空间，那么该程序对任何输入能够产生正确的回答。

图灵证明了这样一个事实：有无穷多个问题是不可解的。

算法的概念

给定一个计算问题P，问题P的复杂度是由求解P的算法的时间复杂度所确定的。

所谓**算法**，是指用计算机求解一类问题的方法。

一个问题是**算法可解的**或**可解的**是指：能够编制计算机程序，只要让该程序运行足够长的时间和使用足够多的存储空间，那么该程序对任何输入能够产生正确的回答。

图灵证明了这样一个事实：有无穷多个问题是不可解的。

举例

例

一个推销商从所在的城市出发，到其他$(n-1)$个城市推销产品，然后返回到所在的城市，假定任意两个城市之间都有一条线路。问题是如何选择一条周游路线使得推销商经过每个城市一次且仅一次且使他所有的路径最短？

如果用枚举法，则有本质上不同的周游线路$\frac{1}{2}(n-1)!$条，求一条周游线路的长度需要$n-1$次加法运算，所以这个算法共需要$\frac{1}{2}(n-1)\cdot(n-1)!$次加法运算。当$n=50$时，共需要$\frac{1}{2}49\cdot 49!\approx 1.5\times 10^{64}$次加法运算。假如一台计算机每秒执行$10^8$次加法运算，那么该算法大约需要$10^{49}$年！

可见，对于一个问题，不仅需要一个求解该问题的算法，还需要一个“好”算法。

举例

例

一个推销商从所在的城市出发，到其他$(n-1)$个城市推销产品，然后返回到所在的城市，假定任意两个城市之间都有一条线路。问题是如何选择一条周游路线使得推销商经过每个城市一次且仅一次且使他所有的路径最短？

如果用枚举法，则有本质上不同的周游线路$\frac{1}{2}(n-1)!$条，求一条周游线路的长度需要$n-1$次加法运算，所以这个算法共需要$\frac{1}{2}(n-1)\cdot(n-1)!$次加法运算。当$n=50$时，共需要$\frac{1}{2}49\cdot 49!\approx 1.5\times 10^{64}$次加法运算。假如一台计算机每秒执行$10^8$次加法运算，那么该算法大约需要$10^{49}$年！

可见，对于一个问题，不仅需要一个求解该问题的算法，还需要一个“好”算法。

举例

例

一个推销商从所在的城市出发，到其他$(n-1)$个城市推销产品，然后返回到所在的城市，假定任意两个城市之间都有一条线路。问题是如何选择一条周游路线使得推销商经过每个城市一次且仅一次且使他所有的路径最短？

如果用枚举法，则有本质上不同的周游线路$\frac{1}{2}(n-1)!$条，求一条周游线路的长度需要$n-1$次加法运算，所以这个算法共需要$\frac{1}{2}(n-1)\cdot(n-1)!$次加法运算。当$n=50$时，共需要$\frac{1}{2}49\cdot 49!\approx 1.5\times 10^{64}$次加法运算。假如一台计算机每秒执行$10^8$次加法运算，那么该算法大约需要$10^{49}$年！

可见，对于一个问题，不仅需要一个求解该问题的算法，还需要一个“好”算法。

时间复杂度

通常算法的时间复杂度除了与问题的输入规模n相关外，还与具体的输入有关。所以如果用T_A表示算法A的时间复杂度，那么T_A是问题输入规模和具体输入的函数，即：

$$T_A = T(n, I)$$

其中n表示所求解问题的输入规模，I表示算法的一个规模为n的输入。

通常只考虑3种情况下的计算复杂度，即最好情形、最坏情形和平均情形下的时间复杂度。

时间复杂度

通常算法的时间复杂度除了与问题的输入规模n相关外，还与具体的输入有关。所以如果用T_A表示算法A的时间复杂度，那么T_A是问题输入规模和具体输入的函数，即：

$$T_A = T(n, I)$$

其中n表示所求解问题的输入规模，I表示算法的一个规模为n的输入。

通常只考虑3种情况下的计算复杂度，即**最好情形、最坏情形和平均情形下的时间复杂度**。

时间复杂度

定义

设D_n为所考虑问题的所有规模为n的输入的集合，对任$I \in D_n$，令$t(I)$表示当输入为I时算法执行的初等运算的次数，那么算法在最好情形下的时间复杂度定义为：

$$T_{min}(n) = \min\{t(I)|I \in D_n\}$$

定义

设D_n为所考虑问题的所有规模为n的输入的集合，对任$I \in D_n$，令$t(I)$表示当输入为I时算法执行的初等运算的次数，那么算法在最坏情形下的时间复杂度定义为：

$$T_{min}(n) = \max\{t(I)|I \in D_n\}$$

时间复杂度

定义

设D_n为所考虑问题的所有规模为n的输入的集合，对任$I \in D_n$，令$t(I)$表示当输入为I时算法执行的初等运算的次数，那么算法在最好情形下的时间复杂度定义为：

$$T_{min}(n) = \min\{t(I)|I \in D_n\}$$

定义

设D_n为所考虑问题的所有规模为n的输入的集合，对任$I \in D_n$，令$t(I)$表示当输入为I时算法执行的初等运算的次数，那么算法在最坏情形下的时间复杂度定义为：

$$T_{min}(n) = \max\{t(I)|I \in D_n\}$$

时间复杂度

定义

设D_n为所考虑问题的所有规模为n的输入的集合，对任$I \in D_n$，令$t(I)$表示当输入为I时算法执行的初等运算的次数，$P(I)$表示I作为输入出现的概率，那么算法在平均情形下的时间复杂度定义为：

$$T_{min}(n) = \sum_{I \in D_n} P(I)t(I)$$

尽管定义了3种特殊情况的时间复杂度，想要精确获得算法的时间复杂度也是非常困难的，通常的做法是估计算法时间复杂度的上界或下界。为此，给出下面的几个定义。

时间复杂度

定义

设D_n为所考虑问题的所有规模为n的输入的集合，对任$I \in D_n$，令$t(I)$表示当输入为I时算法执行的初等运算的次数，$P(I)$表示I作为输入出现的概率，那么算法在平均情形下的时间复杂度定义为：

$$T_{min}(n) = \sum_{I \in D_n} P(I)t(I)$$

尽管定义了3种特殊情况的时间复杂度，想要精确获得算法的时间复杂度也是非常困难的，通常的做法是估计算法时间复杂度的上界或下界。为此，给出下面的几个定义。

上界

定义

设$f(n)$和$g(n)$是从自然数集N到正实数集上的函数，若存在常数$c > 0$，$n_0 \in N$，使得当$n \geq n_0$时有

$$f(n) \leq cg(n)$$

则称当n充分大时$f(n)$有上界，且$g(n)$是它的一个上界，记为$f(n) = O(g(n))$。

上界的判断

如何判断是否存在$f(n) = O(g(n))$呢？下面的定理给出了一个简单的方法：

定理

设$f(n)$和$g(n)$是从自然数集N到正实数集上的函数，若

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c < \infty$$

那么有$f(n) = O(g(n))$。

下界和下界的判断

定义

设$f(n)$和$g(n)$是从自然数集N到正实数集上的函数，若存在常数$c > 0$，$n_0 \in N$，使得当$n \geq n_0$时有

$$f(n) \geq cg(n)$$

则称当n充分大时$f(n)$有下界，且$g(n)$是它的一个下界，记为$f(n) = \Omega(g(n))$。

定理

设$f(n)$和$g(n)$是从自然数集N到正实数集上的函数，若

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c > 0$$

那么有$f(n) = \Omega(g(n))$。

下界和下界的判断

定义

设$f(n)$和$g(n)$是从自然数集N到正实数集上的函数，若存在常数$c > 0$，$n_0 \in N$，使得当$n \geq n_0$时有

$$f(n) \geq cg(n)$$

则称当n充分大时$f(n)$有下界，且$g(n)$是它的一个下界，记为$f(n) = \Omega(g(n))$。

定理

设$f(n)$和$g(n)$是从自然数集N到正实数集上的函数，若

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c > 0$$

那么有$f(n) = \Omega(g(n))$。

同阶

定义

设$f(n)$和$g(n)$是从自然数集N到正实数集上的函数，若存在常数$c_1, c_2 > 0$，$n_0 \in N$，使得当$n \geq n_0$时有

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

则称$f(n)$和$g(n)$是同阶的，记为$f(n) = \theta(g(n))$。

定理

设$f(n)$和$g(n)$是从自然数集N到正实数集上的函数，若

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = c,\ 0 < c < \infty$$

那么有$f(n) = \theta(g(n))$。

同阶

定义

设$f(n)$和$g(n)$是从自然数集N到正实数集上的函数，若存在常数$c_1, c_2 > 0$，$n_0 \in N$，使得当$n \geq n_0$时有

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

则称$f(n)$和$g(n)$是同阶的，记为$f(n) = \theta(g(n))$。

定理

设$f(n)$和$g(n)$是从自然数集N到正实数集上的函数，若

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = c,\ 0 < c < \infty$$

那么有$f(n) = \theta(g(n))$。

计算复杂度

定义

设$f(n)$和$g(n)$是从自然数集N到正实数集上的函数，若对任意常数$c > 0$，存在$n_0 \in N$，使得当$n \geq n_0$时有

$$f(n) < cg(n)$$

则称$f(n) = O(g(n))$。

定理

设$f(n)$和$g(n)$是从自然数集N到正实数集上的函数，若

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

那么有$f(n) = O(g(n))$。

由定义可知，$f(n) = O(g(n))$当且仅当$f(n) = O(g(n))$，但$g(n) \neq O(f(n))$。

计算复杂度

定义

设$f(n)$和$g(n)$是从自然数集N到正实数集上的函数，若对任意常数$c > 0$，存在$n_0 \in N$，使得当$n \geq n_0$时有

$$f(n) < cg(n)$$

则称$f(n) = O(g(n))$。

定理

设$f(n)$和$g(n)$是从自然数集N到正实数集上的函数，若

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

那么有$f(n) = O(g(n))$。

由定义可知，$f(n) = O(g(n))$当且仅当$f(n) = O(g(n))$，但$g(n) \neq O(f(n))$。

一些表达式的关系

通常用$f(n) \prec g(n)$来表示$f(n) = O(g(n))$。

一般有如下关系：

$$1 \prec \log\log n \prec \log n \prec \sqrt{n} \prec n \prec n\log n \prec n^2 \prec n^3 \prec 2^n \prec n! \prec n^n$$

算法分析实例

例 (线性查找)

输入：数组$A[1...n]$和元素x

输出：若$x = A[j](1 \leq j \leq n)$，则输出$j$；否则输出0。

1. $j = 1$
2. while $(j \leq n)$ and $(A[j] \neq x)$ do
3. $j = j + 1$
4. end while
5. if $j \leq n$ then return j else return 0

算法分析实例

例 (线性查找)

输入：数组$A[1...n]$和元素x

输出：若$x = A[j](1 \leq j \leq n)$，则输出$j$；否则输出$0$。

1. $j = 1$
2. while $(j \leq n)$ and $(A[j] \neq x)$ do
3. $j = j + 1$
4. end while
5. if $j \leq n$ then return j else return 0

谢谢！

hanqi_xf@hit.edu.cn