Lecture 1/2

陈纳川

2024 年 3 月 1 日

目录

1 算法分析的基本概念

强烈建议大家自行阅读教材的第一部分导读和第一章对算法的介绍。

第二章第三章的具体例子如果阅读困难可以先不管，之后会细讲。

1.1 渐近复杂度

对于一个有输入或有参数的程序，我们可以使用一些变量来描述它的参数规模，设它为 n ，通常是一个正整数，也可能是一组正整数或正实数。变量 n 可以是参数的个数、最大值、范数等等。这视具体问题的意义而定。

将程序分解为若干 **原子步骤**（例如四则运算、赋值运算等），那么一个程序需要执行的原子步骤的步数 T 是参数 n 的函数，函数 $T(n)$ 在 $n \to \infty$ 处的一个同阶无穷大表示就称为该程序的 **渐近时间复杂度**。

类似地，将该程序需要申请的空间大小也写成 n 的函数 $M(n)$ ，其同阶无穷大就称为该程序的 **渐近空间复杂度**。其中，同阶无穷大使用记号 $\Theta()$ 。例如 $\Theta(n^2), \Theta(n \log n), \Theta(2^n)$ 等。

渐近复杂度是衡量一个算法好坏的重要指标。

注. 由于 $\log_a n$ 和 $\log_b n$ 是同阶无穷大，因此在描述渐近复杂度时底数可以省略。在离散数学里，如无指明底数，可以按底数为 2 计算。

1.2 平均复杂度和上下界

对于同样的参数规模，有的算法对不同的参数可能有不同的表现，这时就不能以单个渐近函数描述复杂度。因此我们引入渐近上界 $O()$，渐近下界 $\Omega()$ 来描述。另外可以使用其平均复杂度，仍然使用记号 $\Theta()$ 。

单从概念上理解可能没有实感，今后对于每种算法我们都会讨论其渐近时间空间复杂度，大家可以在每一个算法的复杂度分析过程中慢慢熟悉其中的方法和小 trick。其中一些复杂的分析方法会在专题章节提及。

1.3 算法的正确性证明

每一个具有输出的算法其实就是一个构造。实现了一个算法可以被视作完成了一个构造性证明。这是在说每一句代码都可以翻译成一句证明。其中最具有计算机特色的内容就是循环和递归函数。也许你没有反应过来它们如何转化成数学构造。我们很快可以看到这俩在数学证明上其实是一回事。书上使用了插入排序的例子，可能略抽象，我们来举另外一个例子：

算法 1. 在长度为 n 的数组 $A[1..n]$ 中找到第一个等于 x 的位置，若找不到输出 -1。

```
1 for i = 1 to n
2     if A[i] == x
3         return i
4 return -1
```

证明. 这个循环算法的正确性由类似于数学归纳法的形式保证。

对于每个 i （即每次进入循环），我们作归纳假设：$P(i): A[1], A[2], \cdots, A[i-1]$ 均不为 x 。

$P(1)$: 空命题无需验证，自然成立。

$P(i) \Rightarrow P(i+1)$: 若 $A[i] == x$ ，则由于前面全都不为 x ，所以找到了第一个等于 x 的位置，返回 i。否则 $A[i] \neq x$ ，不执行额外操作。这使得 $P(i+1)$ 成立，允许进入下一个循环。

因此，程序结束时，$P(n+1)$ 成立，即，找不到这样的位置，返回 -1。 □

我们观察，for 循环内部仅有两行。只有在每次进入循环内部之前保证归纳假设 $P(i)$ 成立，才能保证循环内部语句的合理性。命题 $P(i)$ 被称作 **循环不变量**。这是因为将命题视作取值为 0/1 的一个逻辑变量，其值在每次进入循环时保持为 1 。这个名称可能对于不熟悉逻辑代数的数学系同学来说令人困惑。

那么，循环是一个有始有终的数学归纳法，具有分支结构的递归程序也可以是一个有始有终的归纳法。这就是老师上课时提及的归纳证明。

我们一般会认为，打满补丁的程序、结构冗余混乱的程序、开销大的程序是不好的，就像我们认为一些数学证明是非本质的、冗余的、粗暴的。算法的渐进复杂度、算法的证明结构等等因素共同构成了一种抽象的算法的审美（而不是代码的审美！）。比较有代表性的就是：渐进复杂度的降低（$O(n^2) \to O(n \log n)$）比起时间开销除以或减去常数的降低（$10n^2 \to 2n^2$）是更令人心情愉悦的，很多时候这也是评判一项优化是否本质的标准。当然，具体的算法分析并不是这样简单地完成的，这就需要同学们感受算法世界中的这些精巧的结构了。

有兴趣可以上网了解一下老师的课件里提到的书籍 The Art of Computer Programming 。不过就没必要买来细看了，毕竟这作为一本编程书已经很老了。

2 关于作业和期末考试

2.1 作业

每节课会布置少量书面题，通常是一些有关算法理解的小题，基本上在课后花上十几分钟就能完成。

在题库网站搭建好后，会在上面布置一些程序设计题目。这些题目不同于编程语言基础的堆砌式工程代码，以考察算法理解为主。关于具体的事项之后会说明。这是主要的作业形式，顺利的话每周平均大约两小时可以解决。但以过往的经验来看，一些同学可能会在调试程序上花费一些时间，建议留出一晚上的时间以防万一。

视情况而定，可能会介绍一些书上的算法的实际应用作为拓展作业，不会有工程项目类的大作业。

2.2 C++ 编程

传统上来说，我们把一个程序视作若干指令的集合，其中可以通过自定义函数来模块化封装一些指令。这称为 **面向过程编程**。随着现代的代码库中的函数越来越丰富，人们不得不以自定义类 (class) 的方式规定函数的适用对象，以解决函数太多带来的函数名冲突问

题。这称为 **面向对象编程**。C++ 和 Python 都属于面向对象的编程语言。C 语言中没有关于类的方法，是面向过程的编程语言。

算法在实际应用时，因为需要考虑到代码的可移植性和可读性，绝大多数内容都以面向对象的方式封装完成。虽然在算法课程中我们基本上只需要对每个单独的算法单独实现一份代码，但我们还是鼓励同学们使用 C++ 的 class 方法将每个算法都封装成一个类。

很多同学不熟悉关于地址、指针的一些内容。如果是对语言规范不熟悉，我们实际上几乎都有利用数组实现的替代方法，这在之后用到时会具体介绍。但另一方面，其中的申请空间、编排空间的思想是一定要熟练掌握的。

关于 C++ 的使用说明可以看 runoob-cpp 。面向对象的内容也在这里面。此外该网站还有大量其他编程语言的 tutorial ，用到其他语言时都可以在这里查阅。

2.3 OJ

这门课程的编程作业在在线评测系统上完成自动批改。你编写的程序应当对于给定的所有可能输入参数保证正确性。通常来说，程序的输入和输出都是预先由题目规定好意义和格式的若干数据，程序不应该输出额外的字符。

同时，题目会对程序申请的空间和运行的时间作出限制。通常来说，在 256MB/1000ms 的条件下，使用 4 字节或 8 字节的数据类型，你的数组大小一般至多在 10^7 这个量级。同时，假设你的算法时间复杂度上界为 $O(f(n))$ ，通常它的运行时间可以用 $C \cdot f(n)$ 表示，一般情况 $C < 50$ ，此时将 n 代入程序的时间复杂度渐近式 $f(n)$ ，它应当在 10^8 这个量级。

程序评测会使用大量的而全面的测试数据，对每个测试点的输入检验程序的输出。你需要对所有情况的输入数据保证输出正确无误、不超出时间限制和空间限制、不发生爆栈等问题。当然首先应该不发生编译错误。

题面一般会提供 2 到 3 个样例数据，以便在读题时具体理解题目的逻辑以及对程序做基本的测试。很多时候程序通过了少量的数据不代表程序就是正确的，这就像证明题中的伪证。如果你的程序提交后评测不通过，这个时候需要检查程序是否有不显眼的错误。可以自己另构造一组数据来检验，也可以通过在程序中输出中间结果来检查运算逻辑是否正确（提交题目时要记得删掉！）。调试程序是非常痛苦的，因此在写程序之前就尽量要理清自己的逻辑。

2.4 考试

同学们很关心期末考试的具体题型。这里统一回复一下。

作为数院的课程，这门课以考察算法的理解和程序设计思维为主。这门课的考试一般会有三种题型，下面的样题来自往年期末考试。

题 5. (15 分)
现有一个整数序列 $\{a_i\}, i = 1, \ldots, n$，各元素符号未知，但至少有一个正数。请回答以下问题：
(a)(5 分) 请用分治的思想，设计时间复杂度为 $O(n \log n)$ 的算法，求出该数列中和最大的非空连续子列的元素值的和。
(b)(5 分) 请设计一个非递归的，时间复杂度为 $O(n)$ 的算法，来求出该数列中和最大的非空连续子列的元素值的和。（提示：$\{a_1, \ldots, a_j\}$ 的和最大子列，要么与 $\{a_1, \ldots, a_{j-1}\}$ 的和最大子列相同，要么包含 a_j）。
(c)(5 分) 请设计一个非递归的，时间复杂度为 $O(n)$ 的算法，来求出该数列的积最大的非空连续子列的元素值的积。

图 1: 代码较短、以考察算法思想为主的的程序设计题

二. (15 分) 堆是一种常用的数据结构，对于二叉堆，请回答以下问题：
(a) (5 分) 证明: 含 n 个元素的堆的高度为 $\lfloor \lg n \rfloor$。
(b) (5 分) 证明：对于任一包含 n 个元素的堆中，至多有 $\lceil n/2^{h+1} \rceil$ 个高度为 h 的结点。
(c) (5 分) 给定一个无序数组，往往采用自底向上的方式利用 MAX-HEAPIFY 过程建立最大堆。请给出数组 $A = [1, -3, 12, 7, 9, 1, 6, 18, 5]$ 建立最大堆的过程，并结合该例子证明：对于一个 n 维数组，建立最大堆的时间复杂度为 $O(n)$。

图 2: 算法和数据结构的性质证明题

题 7.(10 分)
(a)(5 分) 请设计高效算法求解二分图的最大匹配问题，描述算法思想并给出相应的示意图。
(b)(5 分) 简述求解图的最大流问题的 Push-Relabel 算法思想，从以下四个方面回答：算法核心思想，两个基本操作，算法终止条件。

图 3: 大算法的描述思路题

今年这门课合并了前置课程《数据结构与数据库》增加到了 80 课时，教材新版也增加了一些内容。老师打算根据数院的实际情况对课程大纲进行一些增删，使得授课内容更加和近年来的实际应用接轨。课程大纲和作业内容也还在讨论，欢迎各位大手子在群里积极发言提意见。

考试难度不会为难大家，基本只要理解完整算法就能拿完分。可能有一两个算法设计小题一时想不出也不用担心，放张图在这里你们自己体会