绪论

什么是数据结构、怎样学习数据结构，是数据结构作为一门学科的两个重要问题。

数据结构，顾名思义，就是**数据在计算机中的存储结构**。一般地，我们创造新的数据结构的原则一般在于：传统的数据结构不适应计算机在进行大数、大量、多维度计算中日新月异的要求。当然，有一些数据结构的提出是为了适应某个领域的特殊需要，如为了算法竞赛而提出了主席树。人们常说，计算机学科就是数据结构加算法的学科。即：计算机学科的一切内容都建构在数据结构和算法的基础上。由此可见学好数据结构的重要性。

那么该怎样学习数据结构呢？

首先，就是要搞懂各种数据结构建立的过程和目的。例如，谈到哈夫曼树，我们就应该知道，这是一个为了解决"怎样构造最有效的二进制编码"问题而提出的树，它实现了任意非等概率数据的最小期望编码。

其次，就是要提高自己的编程基础。数据结构必然是和程序设计紧紧地挂钩的，各种各样的数据结构必然是要通过程序定义的。因此，用好$C/C++$就显得极其重要（我向来反对使用$Python$学习数据结构）。

最后，就是要不怕困难、敢于斗争。数据结构最大的难点不在于内容难以理解，而在于内容多、广、杂。因此，我们需要拿出"靡不有初，鲜克有终"的精神，敢于学习、善于学习，尽可能全面而深入地理解问题。

作为绪论的结语，我想说的是，由于近来以$ChatGPT$为代表的大模型风起云涌，由人工智能生成标准而正确的代码是未来的趋势。因此，计算机科学专业的学生应当更加努力地提高自己，让自己成为能够指引人工智能的人，而不是被人工智能役使的人。

第一章 算法及其复杂度

1.1 算法的定义

算法，顾名思义，就是计算的方法。此计算方法不同于计算机中的“计算方法”学科，那个学科探索的是计算机在做计算时怎样提高解精度的问题，而我们常说的“算法”可以看作一种广义的映射。当带着数据信息的数据结构通过算法之后，就会被映射到对应的解。一般地，计算机学科的“算法”有这样的广为接受的定义：

定义1.1.1（算法） 一个有穷的指令集，当这些指令为解决某一特定任务规定了一个运算序列时，称作一个**算法**。

由于计算机学科是一个偏实践的学科，其中的定义大都不要求背诵。所以上边这个定义看看就得了。接下来我们看一看算法有什么性质：

1.1.2 算法的性质：算法具有**确切性**、**有穷性**、**有效性**。

确切性，说的是算法的每一步指令不能有歧义；**有穷性**，说的是算法应在执行有穷步后结束；**有效性**，说的是每一个步骤都可用计算机指令实现。算法有三大性质，正如集合有三大性质（确定性、互异性、无序性）。

有了算法的定义，我们不难联想：当我们要解决某个特定问题时，我们可能会设计出不同而正确的多种算法。这就引出了两个问题：一是如何设计出算法？二是如何评价这些算法的好坏？这两个问题我们在下面两节里重点地聊一聊。

1.2 算法的设计

算法的设计，可不是我三言两语就能说清楚的，毕竟这是计算机专业看家的一门重要学科。但是为了引入数据结构，我在这里还是不得不提一下这方面的内容，希望可以起到一个抛砖引玉的作用。以下的说法假定读者对 $C/C++$程序设计有一定的基础。

1.2.1 暴力算法：毫不使用技巧，而是直接靠暴力枚举或暴力搜索解决问题的算法。

这一算法无论是在理解上还是代码书写上都是简单的。我们以排序问题为例：

排序问题：***有n个数，请把这些数按照由小到大的顺序排列。***

这个问题显然可以使用暴力搜索求解，例如，我们可以找到序列中最小的数置于顶端，再找到次小的数置于第二位，以此类推。我们称这样的暴力搜索算法叫**插入排序**，其正确性是显然的。接下来，我们再介绍一种经典的暴力算法：依次对相邻两个元素的值进行两两比较，若发现逆序则交换，使值较大的元素逐渐从前移向后部。你会发现，这样一来，值较小的元素就像气泡浮出水面一样“浮”到了序列的顶端。因此这一算法有一个形象的名字——**冒泡排序**。其正确性可以使用**归纳法**证明。读者可以自行证明并完成代码实现。

聪明的你肯定会思考起来，是否存在其他的排序方法呢？答案是肯定的。例如，我们可以将待排序元素分成大小大致相同的两个子集合，分别对两个子集合进行排序，最终将排好序的子集合合并成为所要求的排好序的集合。那么这就涉及到三个过程：分（Divide）、治（Conquer）、合（Combine），这种排序算法叫**归并排序**。另外，我们把使用了分、治、合思想的算法统称为**分治算法**。其正确性依然可以使用**归纳法**证明。

1.2.2 分治算法：把规模较大的大问题分为数个规模较小的子问题，然后通过求解子问题一步步求出大问题的解的算法。

容易发现的是，当我们使用分治算法的时候，我们可能需要把 $n \times k$ 规模的问题压缩为 n 个规模为 k 的问题。为了方便理解，我们暂且机械地把求解该问题的算法定义为函数 $f(x)$ ，其中自变量 x 代表问题的规模。那么当我们想要求解 $f(n \times k)$ 的时候，我们就要调用 n 次 $f(k)$。像这种直接或间接地调用自身的算法称为**递归算法**。如果你是计算机专业的学生，相信你

一定写过使用递归求**整数阶乘**和使用递归求解$Hanoi$ **塔问题**的算法。下面是一个使用递归求**第** n **个** $Fibonacci$ **数**的示例代码：

```
int F(int n)
{
    if(n == 0)
        return 0;
    else if(n == 1)
        return 1;
    else
        return F(n - 1) + F(n - 2);
}
```

我们再来思考这样的问题：分治算法对于子问题彼此独立的情况具有良好的求解效果。但是当子问题存在重叠部分的时候，相同子问题将被重复计算多次，浪费计算资源。例如上面的递归求解斐波那契数列的函数就存在这样的问题。对于这种问题的解决方案，相信每位读者都有自己的理解。我们在本章的习题中再做讨论。

1.3 算法复杂度

在前面的学习中，我们看到：对于同一个问题，可能有多个算法能够求出正确的解。那么如何来评价这些算法的优劣程度呢？定性的研究显然是不恰当的。我们应当定义一个或多个**定量**的标准来比较同一问题的不同算法的效率。为此，我们思考以下问题：

对于一个问题，有两个不同的算法A和B。当给定的数据规模为 n 时，算法A需要占用系统内规模为 $\log_2 n$ 的存储空间，而算法B需要占用系统内规模为 n 的存储空间。那么显然算法A**在内存占用上**是优于算法B的。

对于一个问题，有两个不同的算法A和B。当给定的数据规模为 n 时，算法A需要计算 n 次，而算法B需要计算 n^2 次。那么显然算法A**在计算次数上**是优于算法B的。

为了定量描述上述内容，我们引入算法复杂度的概念。

定义1.3.1（空间复杂度） 算法中定义的所有量所占的存储空间之和为算法的**空间复杂度**。

定义1.3.2（时间复杂度） 算法中的所有语句运行次数之和为算法的**时间复杂度**。

时间复杂度一般为数据规模 n 的函数 $T(n)$。我们知道，在数学中，我们会使用**渐进表达式** $T(n)=O(f(n))$ 来表示 $T(n)$ 的上界为 $f(n)$, 即：$n\to\infty,\ \exists k\in\mathbb{N}_+,\ s.t. T(n)\le kf(n)$。因此，当我们能够求出数据规模为 n 的算法具有上界 $f(n)$ 时，称**算法的时间复杂度上界为** $O(f(n))$。例如，冒泡排序算法的算法复杂度上界为 $O(n^2)$ 。一般地，当我们提到算法复杂度的时候，说的都是时间复杂度。

类似地，我们还可以定义时间复杂度的下界。即：对于函数 $f(n)$, 若 $n\to\infty,\ \exists k\in\mathbb{N}_+,\ s.t. T(n)\ge kf(n)$, 记作$T(n)=\Omega(f(n))$，称**算法的时间复杂度下界为** $\Omega(f(n))$。例如，冒泡排序算法的算法复杂度下界为 $\Omega(n^2)$ 。

对于像冒泡排序这样的算法，它具有一个相等的上界和下界，都是 $f(n)=n^2$，这时我们就可以称**算法的时间复杂度为**$\Theta(f(n))$。即：存在这样的函数 $f(n)$, 使得：

$$n\to\infty,\ \exists k_1,k_2\in\mathbb{N}_+,\ s.t.\ k_1f(n)\le T(n)\le k_2f(n).$$

在实际的问题中，我们往往要处理规模很大的数据。因此，考察时间复杂度的下界对于算法优劣程度意义不大。一般地，我们只通过考察复杂度的上界来考察算法的优劣程度。不难理解的是，对于一个算法，其上界函数 $f(n)$ 的无穷大量阶数越高，其算法的劣势越明显。例如， $O(n)$ 的算法就要比 $O(n^2)$ 的算法要好。

而对于空间复杂度的情况，有以下结论：假定现在有一个解决问题 K 的算法，在这一算法中，我们需要建立 k 个规模为 n 的数组。那么该算法的空间复杂度就是 $O(kn)$，当 k 为常数时，我们可以认为其复杂度为 $O(n)$ 。类似地，如果我们需要建立 k 个规模为 n^2 的二维数组，空间复杂度就是 $O(kn^2)$，当 k 为常数时，我们可以认为其复杂度为 $O(n^2)$。

接下来我们讨论一些算法复杂度的表示问题。

1.3.3 复杂度的性质：根据数学中上界的定义，算法复杂度具有以下性质：

性质1：$O(kf(n)) = O(f(n)),\ k \in \mathbb{N}_+,$

性质2：$O(f(n)) + O(g(n)) = O(f(n) + g(n)) = max\{f(n), g(n)\}.$

性质1是显然成立的。下面证明性质2：设函数 $T(n)$ 的两个上界函数分别为 $f(n)$、$g(n)$，即：

$$n \to \infty,\ \exists k_1 \in \mathbb{N}_+,\ s.t. T(n) \leq k_1 f(n)$$
$$n \to \infty,\ \exists k_2 \in \mathbb{N}_+,\ s.t. T(n) \leq k_2 g(n)$$

则有：$T(n) = O(f(n) + g(n))$。不妨假定 $f(n) = O(g(n))$，即 $n \to \infty,\ \exists k \in \mathbb{N}_+,\ s.t.\ f(n) \leq kg(n)$。

所以有：$T(n) \leq k_1 f(n) \leq (k_1 + k)g(n)$，即$T(n) = O(g(n))$。原命题得证。

1.3.4 算法复杂度分析：我们可以发现的是，在算法设计的过程中，我们其实就能大致估计出算法的复杂度。因此，当我们在设计算法的时候，完全可以以此为工具，判断我们的算法是否有更好的改进措施。

常见的算法复杂度有以下几种：$O(n!)$，$O(x^n)(x > 0)$，$O(n^\alpha \log n)(\alpha > 0)$，$O(n^\alpha)(\alpha > 0)$，$O(\log n)$，$O(1)$。值得注意的是，这里的"log"一般指的是以 2 为底的对数。

接下来，我们来分析一下分治算法的复杂度。根据算法复杂度的定义，我们想要计算分治算法的复杂度，就需要“数”分治算法做了多少次运算。

以分治法递归求解 $Fibonacci$ 数列第 n 项的算法为例：我们可以画出这样一棵**递归树**：

```
                    f(n)
                 /        \
          f(n - 1)       f(n - 2)
         /      \          /      \
   f(n - 2)  f(n - 3)  f(n - 3)  f(n - 4)
                    ......
```

这是一棵高为 n 的二叉树。我们知道递归的出口（可理解为树的叶子节点）是 $n=0$ 或 $n=1$，计算这一步的复杂度是$O(1)$ 的；对于非叶节点，其值为两个孩子的值加和，复杂度也是 $O(1)$ 的。所以整个递归算法的复杂度其实就等于整棵递归树中的节点个数，即 $O(2^n-1)$，也就是 $O(2^n)$。

如果你不知道什么是树和二叉树，也不知道什么是叶子节点，可以看第四章的有关内容。另外，有关分治算法的复杂度计算，我们还有以下结论：

设一个分治算法的每一步能够把一个规模为 n 的大问题分割到 k 个规模为 $\frac{n}{m}$ 的子问题，且分割和合并所消费的总复杂度为 $O(n^i)$，则整体分治算法的复杂度可表示为：

$$T(n)=kT(\frac{n}{m})+O(n^i)$$

可解出：

$$T(n)=\begin{cases}\Theta(n^{\log_m k}) & \log_m k>i\\ \Theta(n^i\log n) & \log_m k=i\\ \Theta(n^i) & \log_m k<i\end{cases}$$

上述结论称为求分治算法复杂度的**主方法**($Mensor\ Method$)。其正确性的证明不要求掌握。读者如果感兴趣可以自行证明或寻找材料加以理解。

习题

1. 写出下列函数的渐进表达式：

   （1） $3n^2+10n$;（2）n^2+2^n；（3）$\log n^3$；（4）$10\log 3^n$.

2. 写出下列代码的时间复杂度（以 $O(f(n))$ 的形式）：

```
count = 0;
for(int k = 1; k <= n; k *= 2)
    for(int j = 0; j <= n; j++)
        count ++;
```

```
i = j = k = 1;
for(i = 1; i <= n; i++)
    for(j = 1; j <= i; j++)
        for(k = 1; k <= j; k++)
            x = i + j - k;
```

```
i = s = 0;
while(s < n)
{
    s += i;
    i++;
}
```

3. 请在自己的电脑上编程实现分治法求解 *Fibonacci* 数列第 n 项的算法，谈一谈 n 较大时的运行情况，然后提出一个优化算法的方案，并考察优化后方案与优化前方案的时间复杂度和空间复杂度情况。

4. 请使用递归树或主方法分析归并排序算法的复杂度。

5. 某毕业班有 n 个同学。在学校的最后一天，每人准备了一份独特的礼物。老师收齐 n 份礼物后，将所有礼物分发给各位同学。要求每个同学都不能分到自己准备的礼物。那么老师有多少种不同的分发方法？请写出递归式并使用递归树或主方法分析复杂度。

第二章 线性表、栈与队列

2.1 顺序表

2.2 链表

2.3 栈

2.4 队列

习题

第三章 字符串、多维数组与广义表

3.1 字符串及其模式匹配

3.2 多维数组与特殊矩阵的压缩存储

3.3 广义表及其三大表示

习题

第四章 树与二叉树

4.1 树

4.2 二叉树

4.3 森林

习题

第五章 树与二叉树的应用

5.1 $Huffman$编码

5.2 堆

5.3 二叉查找树

5.4 AVL 树

5.5 并查集

5.6 搜索与剪枝

习题

第六章 图

6.1 图的定义与存储方式

6.2 图的遍历与连通性

6.3 最小生成树

6.4 最短路径问题

6.5 关键路径与拓扑排序

习题

第七章 查找与排序

7.1 静态查找树

7.2 B树和B+树

7.3 红黑树

7.4 散列($Hash$)

7.5 排序算法

习题

部分习题的答案与提示

第一章

1. （1） $O(n^2)$；（2） $O(2^n)$；（3） $O(\log n)$；（4） $O(n)$.

2. （1）$O(n\log n)$；（2）$O(n^3)$；（3）$O(n^{\frac{1}{2}})$.

3. 复杂度为 $O(n\log n)$.

4. 设对于 n 位同学，可能的分发总情况有 $f(n)$ 种。则递归式为：

$$\begin{cases} f(1)=0 \\ f(2)=1 \\ f(n)=(n-1)f(n-2)+f(n-1) \quad (n>2) \end{cases}$$

   复杂度为 $O(2^n)$.