

第1章 基础知识

#数据结构与算法

概述

本章介绍数据结构的核心理念，包括数据存储方式、算法定义、时间/空间复杂度分析。建议学习时间分配：数据结构占2/3，算法占1/3。

重点关注：数据结构 (Data Structure)、算法 (Algorithm)、时间复杂度 (Temporal Complexity) 和空间复杂度 (Spatial Complexity)。

数据结构与算法

第1章 基础知识

知识要点

- (1) 理解数据结构的基本术语
- (2) 掌握抽象数据类型与数据结构的关系
- (3) 能够描述数据结构的逻辑结构
- (4) 了解算法增长率的概念；
- (5) 掌握渐近分析的分析方法；
- (6) 能够对程序、算法或问题的上下限作出估算；
- (7) 掌握时间复杂度与空间复杂度的分析方法；

联系方式: 13909271400
邮箱: 22068319@qq.com

西安交通大学计算机科学与技术系

1.1 数据结构的基本概念 ❤️

分析问题的三个方面

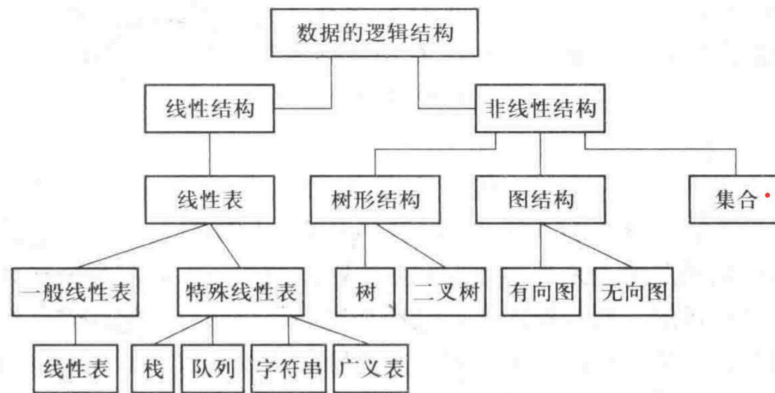
在设计数据结构时，需要从以下角度分析问题：

- **数据的存储方式 (Storage)**：如何在内存中组织数据。
- **数据的操作实现 (Operation)**：如何对数据进行插入、删除、查找等操作。

- **数据的逻辑结构 (Logic)**：数据之间的逻辑关系（如线性或**树状**）。

数据结构 是研究数据及其之间关系的一门学科。它不仅关注数据本身，还关注数据的组织和操作效率。

数据的逻辑结构按关系分为线性结构（数据元素之间的关系是线性组织的）和非线性结构（数据元素之间的关系是非线性组织的）。线性结构包括线性表、栈和队列等。非线性结构包括树、二叉树、图以及集合等。数据的逻辑结构可用一个层次图描述，如图 1-1 所示。



数据存储方式

数据结构有多种存储方式，每种方式适用于不同场景。以下是常见类型：

顺序存储

- 优点：元素连续存储，便于随机访问（ $O(1)$ 时间）。
- 缺点：无前驱/后继灵活性、固定大小（易导致“Out of Memory”）、插入/删除需移动元素、顺序查找效率低（ $O(n)$ ）。
- 适用：数据量固定、频繁随机访问。具体应用可参考[邻接矩阵](#)。

链式存储

- 优点：动态大小，支持插入/删除（ $O(1)$ 如果有指针）。
- 缺点：顺序查找效率低（ $O(n)$ ）、可能产生内存碎片。
- 适用：数据量不确定、频繁修改。具体应用可参考[单链表](#)和[邻接表](#)。

索引存储

- 原理：为关键字段建立索引表，按规律顺序存放数据。
- 优点：快速定位（ $O(\log n)$ 或更好）。
- 适用：数据库查询场景。

哈希存储

- 原理：设计[哈希函数](#)，将数据映射到存储位置（通过计算得到）。
- 优点：平均查找/插入/删除时间为 $O(1)$ 。

- 缺点：可能冲突，需要处理（如链地址法）。
适用：键值对快速查找。

跳跃式存储及特殊序列

- 原理：结合顺序和链式，支持跳跃访问，提高查找效率。

1.2 问题、算法和程序

问题

定义：计算机需要完成的任务，例如“排序一个数组”或“查找最大值”。

特点：有输入、处理过程和输出。

算法 (Algorithm)

定义：为求解特定问题而设计的、计算机执行的**有限序列**指令。

算法的特性：

1. **有穷性**：步骤有限，执行时间有限。
2. **确定性**：每步指令无歧义。
3. **适用范围**：针对特定问题。
4. **可读性和可行性**：易懂且能在计算机上实现。
5. **健壮性**：处理异常输入时可靠。
6. **输入/输出**：有零个或多个输入，有一个或多个输出。

程序 (Program)

算法的具体实现形式，使用编程语言编写。

1.3 算法的评价

高级语言程序在计算机上的运行时间取决于：

1. **算法策略**：核心效率来源（如分治和暴力）。
2. **问题的规模**：输入大小 n 越大，效率差异越明显。
3. **计算机语言级别**：高级语言（如 Python）实现效率低于低级语言（如 C）。
4. **编译和运行环境**：硬件、编译器优化等。

评价重点：时间复杂度和空间复杂度（详见下节）。

1.4 时间复杂度

分析阶段：分为**算法分析**（计算基本操作次数）和**渐近分析**（忽略常数，关注 $n \rightarrow \infty$ 的行为）。

严格分析规则（参考教材 P10）：

- 基本操作：算法的核心步骤（如比较、赋值）。
- 最坏/平均/最好情况：通常关注最坏情况。
- 大 O 表示法： $T(n) = O(f(n))$ ，表示上界（例如 $O(n^2)$ 表示不超过 cn^2 次操作）。

常见复杂度：

- $O(1)$ ：常量时间。
- $O(\log n)$ ：对数时间（如二分查找）。
- $O(n)$ ：线性时间。
- $O(n \log n)$ ：线性对数（如快速排序）。
- $O(n^2)$ ：平方时间（如冒泡排序）。

1.5 空间复杂度

定义：算法运行时占用的存储空间，包括：

- **固定空间：**程序本身和常量空间（不随 n 变化）。
 - **附加空间：**临时变量、递归栈等（随 n 变化）。
- $S(n) = C + S_p(n)$ ，其中 C 是固定部分， $S_p(n)$ 是变量部分。

评价标准：关注附加空间，通常希望 $O(1)$ （原地算法）。

示例：将整数数组的奇数放到前面（参考 [第2章 线性表](#)）。

- 要求：时间 $O(n)$ ，空间 $O(1)$ （不使用额外数组）。
- 思路：双指针从两端向中间移动，交换奇偶位置。

```
void partitionOddEven(int a[], int n) {
    int left = 0, right = n - 1;
    while (left < right) {
        // 从左找第一个偶数
        while (left < right && a[left] % 2 != 0) left++;
        // 从右找第一个奇数
        while (left < right && a[right] % 2 == 0) right--;
        if (left < right) {
            // 交换
            int temp = a[left];
            a[left] = a[right];
            a[right] = temp;
        }
    }
}
```

```
}  
}  
}
```

分析：

- 时间复杂度： $O(n)$ （比较和交换最多 n 次）。
- 空间复杂度： $O(1)$ （仅用一个临时变量 `temp`；循环下标不算附加空间）。

描述程序的方式：

1. 自然语言描述（伪代码）。
2. 形式语言（数学符号）。
3. 类语言（类似编程语言）。
4. 数据流程图（图形化表示）。

下一章 第2章 线性表