数据与算法知识与方法

T$^{\mathrm{T}}$T

2024 年 9 月 28 日

目录

0 绪论：数据、数学模型、算法

0.1 数据及其结构

定义 0.1.1. 数据的基本单元称为**数据元素**。

数据元素并不是孤立存在的，而是存在密切的联系。

数据的逻辑结构：

- **集合结构**：数据元素之间没有任何关系
- **线性结构**：数据元素之间存在一对一的关系，如线性表、栈、队列、串
- **树形结构**：数据元素之间存在一对多的关系，如树
- **图形结构**：数据元素之间存在多对多的关系，如图

数据的存储结构：

- **顺序存储结构**：将数据元素存放在地址连续的存储单元里
- **链式存储结构**：将数据元素存放在任意的存储单元里，通过指针相连

0.2 数学模型

定义 0.2.1. **数学模型**是对于现实世界的某一特定对象，根据其内在规律，为特定目的而得到的一个抽象的、简化的数学结构。

常用的数学模型的类型：

- **线性方程组**：超定线性方程组的线性拟合、欠定线性方程组的线性规划
- **非线性方程组**：非线性方程组的求解
- **微分方程**：常微分方程的数值解法、偏微分方程的数值解法
- **概率模型**：概率预测、概率统计
- **统计模型**：统计预测、统计分析
- **离散模型**：线性结构、树结构、图结构
- **优化模型**：线性规划、整数规划、非线性规划

0.3 算法

定义 0.3.1. **算法**是用以解决某一问题的有限长度的指令序列。

算法的基本特点：

- **有穷性**：算法必须在执行有穷步之后能够结束，且每一步都可在有穷时间内完成
- **确定性**：算法的每一步必须有确切的含义，算法的实际执行结果是确定的、且精确地符合要求或期望
- **可行性**：算法中描述的操作都可以通过已经实现的基本操作运算的有限次执行来实现
- **输入**：算法必须有零个或多个输入
- **输出**：算法必须有一个或多个输出

好的算法应该具有以下特点：

- **正确性**：算法应该能够解决问题
  - 不含语法错误

  - 对一般的输入数据能够产生正确的输出结果
  - 对精心选择的苛刻数据也能产生正确的输出结果
  - 对于所有的输入数据都能产生正确的输出结果
- **可读性**：算法应该容易理解
- **健壮性**：算法应该能够处理各种异常情况
- **高效性**：算法应该能够在合理的时间和空间开销内解决问题

一个特定算法的运行工作量的大小，与问题规模的大小有关，这种关系称为算法的复杂度。

定义 0.3.2. 渐进时间复杂度

算法的**（渐进）时间复杂度**是指算法的运行时间与问题规模之间的关系。

例 0.3.1. 现有程序如下：

```
for(int i = 0; i < n; i++)
    for(int j = i; j < n; j++)
        if(a[i] > a[j])
            swap(a[i], a[j]);
```

其中，基本操作 swap 函数的时间复杂度为 $O(1)$，其执行次数为 $\sum\limits_{i=0}^{n-1}\sum\limits_{j=i}^{n-1}1=\sum\limits_{i=0}^{n-1}(n-i)=\sum\limits_{i=0}^{n-1}i=\dfrac{n(n-1)}{2}$，则该程序的时间复杂度为 $O\left(\dfrac{n(n-1)}{2}\right)=O(n^2)$。

例 0.3.2. 现有程序如下：

```
for(int i = 1; i < n; i++)
    for(int j = 0; j < i; j *= 2)
        a[i] += a[j];
```

其内层循环的时间复杂度为 $O(\log_2 i)$，则该程序的时间复杂度为

$$\begin{aligned}\sum_{i=2}^{n-1}O(\log_2 i)=O\left(\sum_{i=2}^{n-1}\log_2 i\right)=O\left(\log_2\left(\prod_{i=2}^{n-1}i\right)\right)\\=O\left(\log_2(n-1)!\right)=O\left(\log_2 n!\right)=O(n\log_2 n)\end{aligned}$$

定义 0.3.3. 渐进空间复杂度

算法的**（渐进）空间复杂度**是指算法的空间开销与问题规模之间的关系。

1 数据结构

1.1 抽象数据类型

定义 1.1.1. 数据类型

数据类型是元素的集合 D、集合中元素的关系 R 和定义在此集合上的对这些元素的操作的集合 C 的总称。

数据类型是一种封装好的数据结构，把用户无须了解的实现细节封装起来，只提供刻画外在特性的接口。对于高级语言，除开语言本身定义好的**原子类型**外，用户还可以自定义**结构类型**，即**抽象数据类型（ADT）**。

例 1.1.1. 复数的抽象数据类型定义如下：

```
ADT Complex {
    数据对象: D = {e1, e2 | e1, e2 ∈ ℝ}
    数据关系: R = {⟨e1, e2⟩ | e1 = 𝔢(D), e2 = 𝔪(D)}
    基本操作:
        InitComplex(&Z, v1, v2)
            操作结果: 构造复数 Z，其实部和虚部分别被赋以参数 v1 和 v2 的值。
        DestroyComplex(&Z)
            初始条件: 复数 Z 存在;
            操作结果: 复数 Z 被销毁。
        Add(z1, z2, &sum)
            初始条件: z1, z2 是复数。
            操作结果: 用 sum 返回两个复数 z1, z2 的和值。
        Sub(z1, z2, &sub)
            初始条件: z1, z2 是复数。
            操作结果: 用 sub 返回两个复数 z1, z2 的差值。
} ADT Complex
```

例 1.1.2. 圆柱体的抽象数据类型定义如下：

```
ADT CYLinder {
    数据对象: D = {r, h | r, h ∈ ℝ}
    数据关系: R = {⟨r, h⟩ | r 为圆柱底面半径，h 为圆柱高}
    基本操作:
        InitCyld(r, h)
            操作结果: 构造圆柱体，底面半径 r，圆柱高 h。
        BaseArea(r, &bArea)
            初始条件: 圆柱体存在。
            操作结果: 计算圆柱体底面积，用 bArea 返回。
        SideArea(r, h, &sArea)
            初始条件: 圆柱体存在。
            操作结果: 计算圆柱体侧面积，用 sArea 返回。
        Volume(r, h, &vol)
            初始条件: 圆柱体存在。
            操作结果: 计算圆柱体体积，用 vol 返回。
} ADT CYLinder
```

1.2 线性结构

1.2.1 线性表

定义 1.2.1. 线性表

线性表是一种「有序」结构，即在数据元素的非空有限集合中，

- 存在唯一的一个被称为「**第一个**」的数据元素，无前驱；
- 存在唯一的一个被称为「**最后一个**」的数据元素，无后继；
- 除第一个之外，每个数据元素均只有一个直接**前驱**；
- 除最后一个之外，每个数据元素均只有一个直接**后继**。

对呈现这样结构的数据，可以记为 $L=(a_0, a_1, a_2, \cdots, a_{i-1}, a_i, a_{i+1}, \cdots, a_n)$，并称

- a_i 为**第 i 个**数据元素，i 为数据元素 a_i 的**位序**；
- a_{i-1} 为 a_i 的**直接前驱**；
- a_{i+1} 为 a_i 的**直接后继**；
- $n+1$，即线性表中数据元素的个数，为表的**长度**。

线性表的抽象数据类型可以定义为：

```
ADT List {
    数据对象: D = {a_0, a_1, a_2, ···, a_n | a_i ∈ ElemType, 1 ≤ i ≤ n}
    数据关系: R = {⟨a_i, a_{i+1}⟩ | a_i为第 i 个数据元素, a_{i+1}为第 i+1 个数据元素}
    基本操作:
        InitList(&L)
            操作结果: 构造一个空的线性表 L。
        DestroyList(&L)
            初始条件: 线性表 L 存在。
            操作结果: 销毁线性表 L。
        IsEmpty(L)
            初始条件: 线性表 L 存在。
            操作结果: 若 L 为空表, 则返回 true, 否则返回 false。
        ListLength(L)
            初始条件: 线性表 L 存在。
            操作结果: 返回 L 中数据元素的个数。
        GetElem(L, i, &e)
            初始条件: 线性表 L 存在, 0 <= i < ListLength(L)。
            操作结果: 用 e 返回 L 中第 i 个数据元素的值。
        LocateElem(L, e, compare())
            初始条件: 线性表 L 存在, compare() 是数据元素判定函数。
            操作结果: 返回 L 中第一个与 e 满足 compare() 的数据元素的位序; 若不存在, 则返
                回 -1。
        PriorElem(L, cur_e, &pre_e)
            初始条件: 线性表 L 存在。
            操作结果: 若 cur_e 是 L 的数据元素, 且不是第一个, 则用 pre_e 返回它的直接前
                驱; 否则操作失败。
        NextElem(L, cur_e, &next_e)
            初始条件: 线性表 L 存在。
            操作结果: 若 cur_e 是 L 的数据元素, 且不是最后一个, 则用 next_e 返回它的直接后
                继; 否则操作失败。
        ClearList(&L)
            初始条件: 线性表 L 存在。
```

```
            操作结果：将 L 重置为空表。
        ListInsert(&L, i, e)
            初始条件：线性表 L 存在，0 <= i <= ListLength(L)。
            操作结果：在 L 的第 i 个位置（第 i 个元素之前）插入新的数据元素 e，L 的长度加
                1。
        ListDelete(&L, i, &e)
            初始条件：线性表 L 存在，0 <= i < ListLength(L)。
            操作结果：删除 L 的第 i 个数据元素，并用 e 返回其值，L 的长度减 1。
        ListTraverse(L, visit())
            初始条件：线性表 L 存在，visit() 是对数据元素操作的函数。
            操作结果：依次对 L 的每个数据元素调用 visit()，一旦 visit() 失败，则操作失败。
} ADT List
```

A LambdaOJ 平台介绍

A.1 判题结果

判题有如下结果：

- `ACCEPTED`：结果完全正确。
- `WRONG_ANSWER`：程序正常执行，也没有超时和超内存，但是答案不对。
- `TIME_LIMIT_EXCEEDED`：程序超时，可能是死循环，也可能是算法不够优，或者实现不够优；还没到核对答案那一步，内存情况也不明。
- `MEMORY_LIMIT_EXCEEDED`：程序超过内存限制，可能是算法不够优，或者实现不够优；也没有到核对答案那一步。
- `OUTPUT_LIMIT_EXCEEDED`：输出数据过多，要么程序有 bug，要么是恶意代码。
- `BAD_SYSCALL`：使用了非法的系统调用，要么是恶意代码，大部分情况是 C++ 在 `new` 一块内存的时候，超过了限制失败了，C++ 运行时环境需要一个系统调用关闭信号，然后杀死这个进程。
- `RUN_TIME_ERROR`：运行时错误，可能是由于除零、引用空指针或者数组越界等等。

A.2 注意事项

尽量使用 `stdio.h` 而不使用 `iostream`，前者的输入输出效率更高，时间和空间开销更小。