数据结构与算法 笔记

数据结构与算法（DSA）被誉为计算机类专业中最难的一门学位课，部分学校的挂科率甚至超过50%。但是其实我们并没有必要望而生畏，它并不比高中语文更难。

什么是DSA呢？它是一门研究非数值计算的程序设计问题中的操作对象，以及它们之间的关系和操作等相关问题的学科。我们学习的目标就是研究出一个程序的**最优算法**。

概论和入门

数据结构

传统上，我们把数据结构分为**逻辑结构**和**物理结构**。

- 逻辑结构：是指对象中数据元素之间的相互关系，也是我们今后最需要关注和讨论的问题。
  - 集合结构：若干元素同属于一个集合。
  - 线性结构：元素之间最多存在**一对一**的关系。
  - 树形结构：元素之间最多存在**一对多**的关系。
  - 图形结构：元素之间存在**多对多**的关系。
- 物理结构：是指数据在逻辑结构在计算机中的存储形式。
  - 顺序存储：是把数据元素存放在地址连续的存储单元里，其数据间的逻辑关系和物理关系是一致的。如**数组**。
  - 链式存储：是把数据元素存放在任意的存储单元里，这组存储单元可以是连续的，也可以是不连续的。如**链表**。

算法

算法是是解决特定问题求解步骤的描述，在计算机中表现为指令的有限序列，并且每条指令表示一个或多个操作。

算法具有五个基本特征：**输入、输出、有穷性、确定性和可行性**。

- 输入：算法具有零个或多个输入。
- 输出：算法至少有一个或多个输出。
- 有穷性：指算法在执行有限的步骤之后，自动结束而不会出现无限循环，并且在每一个步骤在可接受的时间内完成。
- 确定性：
  - 算法的每一个步骤都具有确定的含义，不会出现二义性。
  - 算法在一定条件下，只有一条执行路径，相同的输入只能有唯一的输出结果。
  - 算法的每个步骤都应该被精确定义而无歧义。
- 可行性：算法的每一步都必须是可行的，即每一步都能够通过执行有限次数完成。

算法设计还具备一些额外的可选要求。

- 正确性：算法的正确性是指算法至少应该具有输入、输出和加工处理无歧义性、能正确反应问题的需求、能够得到问题的正确答案。
  正确性的四个层次：

    - 算法程序没有错误。
    - 算法程序对于合法输入能够产生满足要求的输出。
    - 算法程序对于非法输入能够产生满足规格的说明。
    - 算法程序对于故意刁难的测试输入都有满足要求的输出结果。
- 可读性：算法设计另一目的是便于阅读、理解和交流。
- 健壮性：当输入数据不合法时，算法也能做出相关处理，而不是产生异常、崩溃或莫名其妙的结果。
- 经济性：消耗的时间短、占用的硬件资源少。

时间复杂度

> 函数的渐近增长：对于任意的`f(n) = an ^ x + bn ^ y + ... + pn + q`，低次项和常数的部分往往可以忽略，我们更需要关注主项/最高项的阶数。

在进行算法分析的时候，**语句总的执行次数T(n)是关于问题规模n的函数**，进而分析T(n)随n的变化情况并确定T(n)的数量级。算法的时间复杂度，也就是算法的时间量度，记作**T(n) = O(f(n))**。它表示随问题规模的增大，算法执行时间的增长率和F(n)的增长率相同，称作算法的渐近时间复杂度，简称为时间复杂度，其中f(n)是问题规模n的**某个函数**（n一般只与循环有关）。

> 我们常把顺序线性执行的一整段代码视为1次运算。

一般情况下，随着输入规模n的增大，T(n)增长最慢的算法为最优算法。

推导出函数表达式的O(n)的步骤：

1. 将常数项替换为1。
2. 去除**所有非最高次项。**
3. 将最高项的系数替换为1。

举例：请推导出g(n) = 3n ^ 3 + n ^ 2 + 4的大O阶。

```
O(n ^ 3)
```

如果T(n) = O(1)，则称其含有常数阶；如果T(n) = O(n)，则称其含有线性阶；如果T(n) = O(n ^ 2)，则称其含有平方阶；如果T(n) = O(logn)，则称其含有对数阶（底数也认为是常数，故抹去，无默认值）。此外，还有立方阶、指数阶、nlogn阶……

> 我们一般考虑的O(f(n))都是指最恶情况。快速排序除外。

常见的时间复杂度从短耗时到长耗时排序为：
O(1) < O(logn) < O(n) < O(nlogn) < o(n ^ 2) < O(n ^3) < O(2 ^ n) < O(n!) < O(n ^ n)

空间复杂度

> 现在市场上所说的“复杂度”一般指时间复杂度，这也是算法的潮流。

在程序设计的时候，我们必须衡量时间或者空间的重要性。对于不吝啬空间的运行设备来说，我们完全可以写一套低效的算法，甚至完全靠穷举，这样节省的是程序员的时间和精力，并且也降低了运算压力，但大大增加了存储开销。**这里重点批评国内的一众互联网大厂，985程序员有能耐就写出来这种垃圾给人用是吧？天天写垃圾只为了等工资到账，格局属实是没打开，整个人就卡在那个位置不上不下，算是只能度过一个相对失败的人生。** 对于吝啬空间的运行设备，如工控设备、单片机等，我们就可以使用结构复杂但高效的算法，因为它们不需要太高的运算速度，而是需要面临各种挑战（穷举法很容易漏情况）。具体怎么调节两者的关系，需要看使用的环境需要。

算法的空间复杂度通过计算计算所需的存储空间实现，计算公式记作：S(n) = O(f(n))，其中n为问题的规模，f(n)为语句关于n所占存储空间的函数。

数据结构

抽象数据类型

> 抽象：是指抽取出事物具有的普遍性的本质。它要求抽出问题的特征而忽略非本质的细节，是对具体事物的一个概括。抽象是一种思考问题的方式，它隐藏了繁杂的细节。

> 数据类型：是指一组性质相同的值的集合及定义在此集合上的一些操作的总称。如整型、浮点型、字符型等。

在C中，按照取值的不同，数据类型可以分为两种类型：

- 原子类型：不可再分解的基本类型，例如整型、浮点型、字符型等。
- 结构类型：由若干个类型组合而成，是可以再分解的，例如数组、结构体等。

我们对已有的数据类型进行抽象，就有了抽象数据类型（Abstract Data Type）：是指一个数学模型及定义在该模型上的一组操作。其定义仅取决于它的一组逻辑特性，而与其在计算机内部如何表示和实现无关。也就是说，我们只抽象数据类型的数学特性。

线性表

线性表（List）是由零个或多个数据元素组成的有限序列。
你需要注意：

- 线性表是一个序列，元素之间存在“先来后到”的关系。
- 若元素个数为0，则称之为**空表**。
- 若元素存在多个，则第一个元素无**前驱（直接前驱元素）**，最后一个元素无**后继（直接后继元素）**，其他元素**有且只有**一个前驱和后继。
- 线性表强调是有限的，是因为计算机只能处理有限的元素。

线性表有两种物理存储结构：顺序和链式。

顺序存储（动态数组）的结构伪代码：

```
#define MAXSIZE 20 // 线性表最大存储容量

typedef int ElemType;
typedef struct {
    ElemType data[MAXSIZE]; // 存储空间起始位置
    int length; // 线性表当前长度
} ArrayList;
```

- 优点：
  - 无需为表示表中元素之间的逻辑关系而增加额外的存储空间。
  - 可以快速地获取表中任意位置的元素。
- 缺点：
  - 插入和删除需要移动大量元素（最恶情况是O(n)）。
  - 当线性表长度变化较大时，难以确定存储空间的容量。
  - 难以充分利用内存空间，碎片化太严重。

> 在链表中，我们把存储数据元素信息的域称为**数据域**，把存储直接后继位置的域称为**指针域**。指针域中存储的信息称为**指针或链**。这两部分信息组成的数据元素称为**结点（Node）或存储映像**。链表有一个**头指针**变量，它存放一个地址，该地址指向**头结点**，最后一个结点的指针指向NULL。

链式存储（链表）的结构伪代码：

```c
typedef struct Node {
    ElemType data; // 数据域
    struct Node *next; // 指针域
} node;
Node *linkedList;
```

栈和队列

在写了，在写了（新建文本文档