数据结构研究的主要问题

数据结构是一门研究非数值计算的程序设计中计算机的操作对象以及它们之间的关系和操作的学科

基本术语和概念

几个概念

数据（**Data**）：所有能输入到计算机中并能被计算机程序处理的符号的集合
数据元素（**Data Element**）：

- 数据的基本单位，即数据集合中的一个个体
- 也称为元素，或称为记录、结点或顶点
- 一个数据元素可由若干个数据项组成
- 数据项是数据的最小单位
  数据对象（**Data Object**）：
- 是具有相同性质的数据元素的集合，是数据的一个子集
- 例如整数数据对象是集合$N = \{0, \pm1, \pm2\}$
- 与数据元素的区别
  - 数据元素——组成数据的基本单位，与数据的关系：是集合的个体
  - 数据对象——性质相同的数据元素的集合，与数据的关系：是集合的子集
    数据结构（**Data Structure**）：
- 是相互之间存在一种或多种特定关系的数据元素的集合
- 数据元素之间的关系称为结构
- 从集合论的观点加以描述：数据结构是一个二元组$DataStructure = (D, S)$
  - D：数据元素的有限集
  - S：是D上关系的有限集

几种机构

逻辑结构：数据元素之间的逻辑关系

- 描述数据元素之间的逻辑关系
- 是从具体问题抽象出来的数学模型
  存储结构：数据元素及其关系在计算机内存中的表示（又称为映像）
- 是数据的逻辑结构在计算机存储器中的表示
- 包括数据元素的表示和关系的表示
  运算和实现：即对数据元素可以施加的操作以及这些操作在相应的存储结构上的实现

FxMarkBrown's Blog

逻辑结构与存储结构的关系

- 逻辑结构是数据结构的抽象（算法设计），存储结构是数据结构的实现（算法实现）
- 综合起来建立了数据元素之间的结构关系

逻辑结构的种类

一种分类方式：

- 线性结构
  - 线性表
    - 一般线性表
      - 特殊线性表：栈、队列、字符串
      - 线性表的推广：数组、广义表
- 非线性结构
  - 树结构：树、二叉树
  - 图结构：有向图、无向图
  - 集合结构

  或可以分类为：
- 集合：数据元素间除“同属于一个集合”外，无其他关系
- 线性结构：数据元素之间存在一个对一个的关系，如线性表、栈、队列
- 树形结构：一个对多个关系
- 图状结构：多个对多个关系

存储结构的种类

- 顺序存储结构：用一组连续的存储单元依次存储数据元素，数据元素之间的逻辑关系由元素的存储位置来表示
- 链式存储结构：利用结点中的指针来表示数据元素之间的关系

数据类型和抽象数据类型

数据类型：一组性质相同的值的集合以及定义在这个值集合上的一组操作的统称
抽象数据类型（**Abstract Data Type, ADT**）：是指一个数学模型以及定义在此模型上的一组操作

- 由用户定义，从问题抽象出数据模型（逻辑结构）
- 还包括定义在数据模型上的一组抽象运算（相关操作）
- 不考虑计算机内的具体存储结构与运算的具体实现算法

抽象数据类型可以用三元组表示（D,S,P）：

- D：数据对象

FxMarkBrown's Blog

- S：D上的关系集
- P：对D的基本操作集

抽象数据类型的定义格式：

抽象数据类型的定义格式：

```
ADT 抽象数据类型名{
        数据对象：<数据对象的定义>
        数据关系：<数据关系的定义>
        基本操作：<基本操作的定义>
}ADT 抽象数据类型名
```

- 数据对象、数据关系的定义用伪码表示
- 基本操作的定义格式：

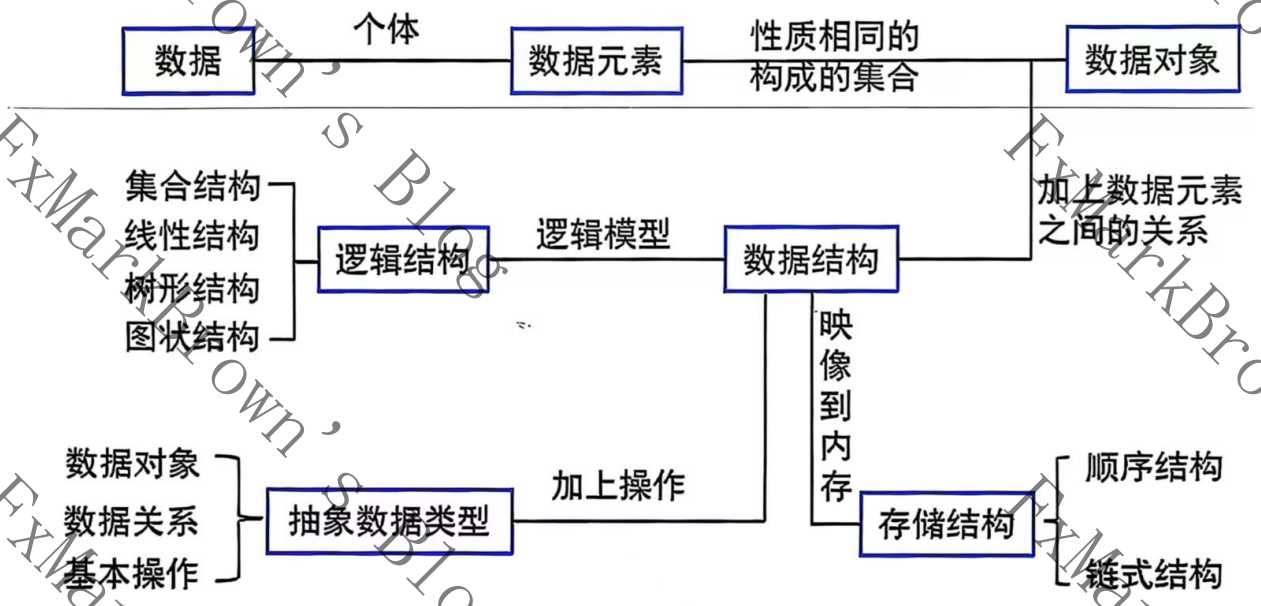
  基本操作名（参数表）

  初始条件：<初始条件描述>

  操作结果：<操作结果描述>

小结

概念小结



FxMarkBrown's Blog

算法和算法分析

算法特性

- 有穷性：必须总是在执行有穷步之后结束，且每一步都可以在有穷时间内完成
- 确定性：每条指令必须有确切的含义，没有二义性
- 可行性：算法是可执行的，算法描述的操作可以通过已经实现的基本运算执行有限次来实现
- 输入：有零个或多个输入
- 输出：有一个或多个输出

算法设计的要求

- 正确性：算法满足问题要求，能正确解决问题
- 可读性：易于理解
- 健壮性：能够处理非法数据、意外情况
- 高效性：对时效性和存储量的需求

算法效率

算法效率主要考虑两个方面：时间效率和空间效率
算法效率可用两种方法度量：事后统计和事前分析

时间复杂度

$$\text{算法运行时间} = \sum \text{每条语句频度} \times \text{该语句执行一次所需的时间}$$

但实际上执行一次所需的时间和硬件有关，所以可以设其为单位时间，则为

$$\text{算法运行时间} = \sum \text{每条语句频度}$$

FxMarkBrown's Blog

例：n*n矩阵相乘的算法

```
1   for (i=1; i<=n; i++)                          n+1
2     for (j=1; j<=n; j++) {                      n(n+1)
3         c[i][j]=0;                              n^2
4         for (k=1; k<=n; k++)                    n^2(n+1)
5             c[i][j]+=a[i][k]*b[k][j];           n^3
6       }
```

- **算法消耗的时间为算法中每条语句的频度之和，则上述算法的时间消耗T(n)为：**

$$T(n) = 2n^3 + 3n^2 + 2n + 1$$

为了便于比较不同算法的时间效率，仅比较它们的数量级，用大O表示法表示为渐进时间复杂度
一般情况下，不必计算所有操作的执行次数，而只考虑算法中基本操作的执行次数，它是问题规模n的某个函数，用$T(n)$表示

时间复杂度定义：算法中基本语句重复执行的次数是问题规模n的某个函数$f(n)$，算法的时间度量为

$$T(n) = O(f(n))$$

例：分析下面程序片段的时间复杂度

```
i=1;                    (1)
while ( i <= n )
        i = i * 2;      (2)
```

关键是找出来执行次数x与n的关系，并表示成n的函数

若循环执行1次： $i=1*2=2$

若循环执行2次： $i=2*2=2^2$

若循环执行3次： $i=2^2*2=2^3$

若循环执行x次： $i=2^x$

设语句(2)执行次数为x次，由循环条件$i<=n$得$2^x<=n$

最好、最坏与平均时间复杂度

- 最坏时间复杂度：指在最坏情况下，算法的时间复杂度
- 平均时间复杂度：指在所有可能输入实例在等概率出现的情况下，算法的期望运行时间
- 最好时间复杂度：指在最好情况下，算法的时间复杂度

空间复杂度

$$S(n) = O(f(n))$$

类似时间复杂度

FxMarkBrown's Blog