数据结构研究

1. 数据结构研究的内容

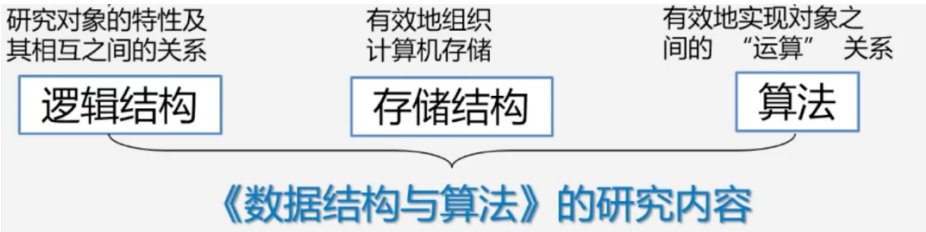
- 通常，用计算机解决问题的步骤
  a) 具体问题抽象为数学模型

     其实质为:

     - 分析问题
     - 提取操作对象
     - 找出操作对象之间的关系
     - 用数学语言描述，即**数据结构**

  b) 设计算法

  c) 编程，调试，运行
- 描述非数值计算问题的数学模型不寿数学方釋，而是诸如表/树和图之类的具有逻辑关系的数据。
- 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家以及它们之间的关系和操作的学科。



基本概念和术语

1. 数据 Data

- 是能输入计算机且能被计算机处理的各种符号的集合。
  a) 信息的载体
  b) 是对客观事物符号化的表示
  c) 能够被计算机识别、存储和加工
- 包括:
  a) 数值型的数据: 整数、实数等
  b) 非数值型的数据: 文字、图像、图形、声音

2. 数据元素 Data Element 和数据项

- **数据元素**是数据的**基本单位**，在计算机程序中通常作为一个整体进行考虑和处理。也简称为元素，或称为记录、结点或顶点。

- **数据项**是构成数据元素的不可分割的**最小单位**。

3. 数据对象 Data Object

- 是**性质相同的数据元素**的集合，是数据的一个子集。
- 例如

  整数数据对象是集合$N = \{0, \pm1, \pm2, \dots\}$

  字母字符数据对象是集合$C = \{'A', 'B', 'C', \dots\}$

  学籍表也可以看作一个数据对象

4. 数据结构 Data Structure

- 数据元素不是孤立存在的，它们之间存在着某种关系，**数据元素相互之间的关系称为结构**。
- 指相互之间存在一种或多种特定关系的数据元素集合。或者说，**数据结构是带结构的数据元素的集合**。
- 包括以下三个方面的内容:
  a) 数据元素之间的逻辑关系，也称为**逻辑结构**。
  b) 数据元素及其关系在计算机内存中的表示（又称为映像），称为数据的**物理结构**或数据的**存储结构**。
  c) 数据的**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。
- **数据结构的两个层次:**
  - **逻辑结构:**

    描述数据元素之间的逻辑关系

    与数据的存储无关，独立于计算机

    是从具体问题抽象出来的数学模型
  - **物理结构（存储结构）:**

    数据元素及其关系在计算机存储器中的结构（存储方式）

    是数据结构在计算机中的表示
  - 逻辑结构与存储结构的**关系：**

    存储结构是逻辑关系的映象与元素本身的映象。

    逻辑结构是数据结构的抽象，存储结构是数据结构的实现

4.1 逻辑结构

- **逻辑结构的种类**

  划分方式一:

  a) 线性结构

  有且仅有一个开始和一个终端结点，并且所有结点都最多只有一个直接前趋和一个直接后继。

  例如: 线性表、栈、队列、串

  b) 非线性结构

  一个结点可能有多个直接前趋和直接后继

  例如: 树、图

  划分方式二，四类基本逻辑结构:

  a) 集合结构 :结构中的数据元素之间除了同属于一个集合的关系外，无任何其它关系。

  b) 线性结构：结构中的数据元素之间存在着一对一的线性关系。

  c) 树形结构：结构中的数据元素之间存在着一对多的层次关系。

d) 图状结构或网状结构：结构中的数据元素之间存在着多对多的任意关系。

4.2 存储结构

- **四种基本的存储结构**

a) 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。

· C 语言中用数组来实现顺序存储结构。

· 例如: bat，cat，eat，...

其存储顺序就是按照元素顺序进行的

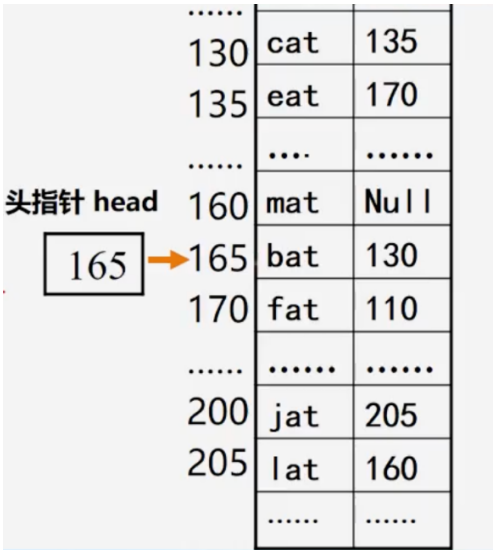
b) 链接存储结构:

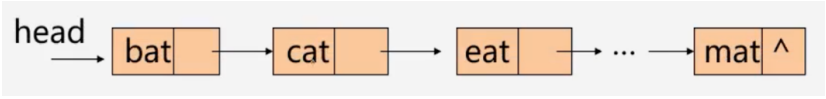
· 用一组任意的存储单元存储数据元素，数据元素之间的逻辑关系用指针来表示。

· C 语言中用指针来实现链式存储结构。

· 例如: bat，cat，eat，...

在存储一个元素本身时，也会同时存储下一个元素的地址（如下图所示）





c) 索引存储结构

· 在存储结点信息的同时，还建立附加的索引表。

· 索引表中的每一项成为一个索引项

· 索引项的一般形式是:（关键字，地址）

· 关键字是能唯一标识一个结点的那些数据项

· 若每个结点在索引表中都有一个索引项，则该索引表称之为**稠密索引**(Dense Index)。若一组结点在索引表中只对应一个索引项，则该索引表称之为**稀疏索引**(Sparse Index)

d) 散列存储结构

- 根据结点的关键字直接计算出该结点的存储地址。

key	32	75	70	63	48	94	25	36	18
key%11	10	9	4	8	4	6	3	3	7

得到的散列表如下

0	1	2	3	4	5	6	7	8	9	10
18			25	70	48	94	36	63	75	32

5. 数据类型和抽象数据类型

5.1 引言

- 在使用高级程序设计语言编写程序时，必须对程序中出现的每个变量、常量或表达式，明确说明它们所属的数据类型。

  例如: C 语言中

  - 提供 int，char，float，double 等基本数据类型
  - 数组、结构、共用体，枚举 等构造数据类型
  - 还有指针、空(void)类型
  - 用户也可用 typedef 自己定义数据类型
- 一些最基本数据结构可以用数据类型来实现，如数组、字符串等。
- 而另一些常用的数据结构，如栈、队列、树、图等，不能直接用数据类型来表示。
- 高级语言中的数据类型明显地或隐含地规定了在程序执行期间变量和表达的所有可能的取值范围，以及在这些数值范围上所允许进行的操作。

  例如，C 语言中定义变量为 int 类型，就表示是[-min, max]范围的整数，在这个整数集上可以进行+、-、*、\、%等操作。
- **数据类型的作用:**

  a) 约束变量或常量的取值范围

  b) 约束变量或常量的操作

5.2 概念

- 数据类型 Data Type

  定义：数据类型是一组性质相同的值的集合以及定义于这个值集合上的一组操作的总称。

  数据类型 = 值的集合 + 值集合上的一组操作

- 抽象数据类型 Abstract Data Type

  定义：指一个数学模型以及定义在此数学模型上的一组操作。

  - 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 = 数据的逻辑结构 + 抽象运算（运算的功能描述）

5.3 抽象数据类型

- **形式定义**

  抽象数据类型可用(D, S, P) 三元组表示。

  其中：D 是数据对象

  S 是 D 上的关系集

  P 是对 D 的基本操作集

- **形式定义格式**

```
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代码描述

基本操作的定义格式为:

a) 基本操作名 <参数表>

赋值参数：只为操作提供输入值。

引用参数：以&打头，除可提供输入值外，

还将返回操作结果。

b) 初始条件 <初始条件描述>

描述操作执行之前数据结构和参数应满足的条件，

若不满足，则操作失败，并返回相应出错信息。

若初始条件为空，则省略之。

c) 操作结果 <操作结果描述>

说明操作正常完成之后，数据结构的变化状况和应返回的结果。

```
ADT 抽象数据类型名{
Data
    数据对象的定义
    数据元素之间逻辑关系的定义
Operation
    操作 1
        初始条件
        操作结果描述
    操作 2
        ……
    操作 n
        ……
} ADT 抽象数据类型名
```

- 举例

```
ADT Circle {
  数据对象：D={r,x,y| r,x,y 均为实数}
  数据关系：R={< r,x,y >| r是半径,<x,y>是
圆心坐标 }
  基本操作:
   Circle(&C,r,x,y)
              操作结果：构造一个圆。
   double Area(C)
              初始条件：圆已存在。
              操作结果：计算面积。
   double Circumference (C)
              初始条件：圆已存在。
              操作结果： 计算周长。
          ......
} ADT Circle
```

6. 总结

算法与算法分析

1. 算法的描述

- 自然语言: 中文，英文
- 流程图: 传统流程图，NS 流程图
- 伪代码: 类语言: 类 C 语言
- 程序代码: C 语言程序，JAVA 语言程序…

2. 算法与程序

- 算法是解决问题的一种方法或一个过程，考虑如何将输入转换成输出，一个问题可以有多种算法。
- 程序是用某种程序设计语言对算法的具体实现。

3. 算法的特性

- **有穷性:** 一个算法必须总是在执行有穷步之后结束，且每一步都在有穷时间内完成。
- **确定性:** 算法中的每一条指令必须有确切的含义，没有二义性，在任何条件下，只有唯一的一条执行路径，即对于相同的输入只能得到相同的输出。
- **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操作执行有限次来实现。
- **输入:** 一个算法有**零个**或多个输入。
- **输出:** 一个算法有**一个**或多个输出。

4. 算法设计要求

- **正确性 Correctness**

  程序对于**精心选择的、典型、苛刻且带有刁难性**的几组输入数据能够得出满足要求的结果。

- **可读性 Readability**

  算法主要是为了人的阅读和交流，其次才是为计算机执行，因此算法应该易于人的理解。

  另一方面，晦涩难读的算法易于隐藏较多错误而难以调试。

- **健壮性 Robustness**

  指当输入非法数据时，算法恰当的做出反应或进行相应处理，而不是产生莫名其妙的输出结果。

  处理出错的方法，不应是中断程序的执行，而应是返回一个表示错误或错误性质的值，以便在更高的抽象层次上进行处理。

- **高效性 Efficiency**

  要求花费尽量少的时间和尽量低的存储需求

5. 算法评估/算法设计衡量

- 一个好的算法首先要具备正确性，然后是健壮性，可读性，在几个方面都满足的情况下，主要考虑算法的效率，通过**算法的效率**高低来评判不同算法的优劣程度。
- 算法效率以下两个方面来考虑:

  a) **时间效率**: 指的是算法所耗费的时间。

  b) **空间效率**: 指的是算法执行过程中所耗费的存储空间。

  时间效率和空间效率有时候是矛盾的。

5.1 算法时间效率的度量

- 算法时间效率可以用依据该算法编制的程序在计算机上执行所消耗的时间来度量。
- 两种度量方法

  a) 事后统计

  将算法实现，测算其时间和空间开销。

  b) 事前分析

  对算法所消耗资源的一种估算方法。

5.2 事前分析方法

- 一个算法的运行时间是指一个算法在计算机上运行所耗费的时间大致可以等于计算机执行一种简单的操作（如赋值、比较、移动等）所需的时间与算法中进行的简单操作次数乘积。

$$\textbf{算法运行时间} = \textbf{一个简单操作所需的时间} \times \textbf{简单操作次数}$$

- 也即算法中每条语句的执行时间之和。

$$\textbf{算法运行时间} = \sum \textbf{每条语句的执行次数} \times \textbf{该语句执行一次所需的时间}$$

每条语句的执行次数又称为**语句频度**。

每条语句执行一次所需的时间，一般是随机器而异的。取决于机器的指令性能、速度以及编译的代码质量。是由机器本身软硬件环境决定的，它与算法无关。

所以，可假设执行每条语句所需的时间均为**单位时间**。此时对算法的运行时间的讨论就可转化为**讨论该算法中所有语句的执行次数**，即**频度之和**了。

$$\textbf{算法运行时间} = \sum \textbf{每条语句的执行次数}$$

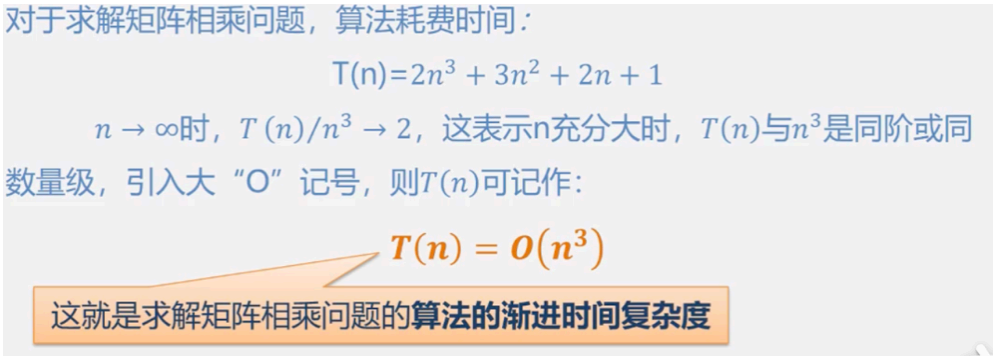
5.3 算法时间复杂度的渐进表示法

- 为了便于比较不同算法的时间效率，仅比较它们的**数量级**

例如：两个不同的算法，时间消耗分别是：

$$T_1(n) = 10n^2 \quad \text{与} \quad T_2(n) = 5n^3$$

- 若有某个辅助函数$f(n)$，使得当n趋近于无穷大时，$T(n)/f(n)$的**极限值**为**不等于零的常数**，则称$f(n)$是$T(n)$的同数量级函数。记作$T(n)=O(f(n))$，称$O(f(n)$为**算法的渐进时间复杂度**(O是数量级的符号)，简称**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执行的次数，它是问题规模n的某个函数，用$T(n)$表示。
- **总结:**

  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$f(n)$，算法时间量度记作:

  $$Tn = O(f(n))$$

  它表示随着n的增大，算法执行的时间的增长率和$f(n)$的增长率相同，称**渐近时间复杂度**。

  其中，**基本语句**的判定:

  a) 算法中重复执行次数和算法的执行时间成正比的语句

  b) 对算法运行时间的贡献最大

  c) **执行次数最多，即语句频率最大**

5.4 算法时间复杂度

- 有的情况下，算法中基本操作重复执行的次数还随问题的输入数据集不同而不同

【例】顺序查找，在数组a[i]中查找值等于e的元素，返回其所在位置。

```
for (i=0;i< n;i++)
  if (a[i]==e) return i+1;  //找到，则返回是第几个元素
return 0;
```

- **最好情况：1次**
- **最坏情况：n**
- **平均时间复杂度为:O(n)**

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。

- 一般总是考虑在**最坏情况下的时间复杂度**，以保证算法的运行时间不会比它更长。

- 对于复杂的算法，可以将其分成几个容易估算的部分，然后利用加法法则和乘法法则，计算算法的时间复杂度:

  a) 加法规则

  $$T(n) = T_1(n) + T_2(n) = O\big(f(n)\big) + O\big(g(n)\big) = O(\max(f(n), g(n)))$$

  b) 乘法规则

  $$T(n) = T_1(n) \times T_2(n) = O\big(f(n)\big) \times O\big(g(n)\big) = O(f(n) \times g(n))$$

- 时间复杂度$T(n)$按数量级递增顺序为:

复杂度低 → 复杂度高

常数阶	对数阶	线性阶	线性对数阶	平方阶	立方阶	…	K次方阶	指数阶
O(1)	$O(\log_2 n)$	O(n)	$O(n \log_2 n)$	$O(n^2)$	$O(n^3)$		$O(n^k)$	$O(2^n)$

5.5. 渐进空间复杂度

- 空间复杂度: **算法所需存储空间的度量**

  记作: $S(n) = O(f(n))$

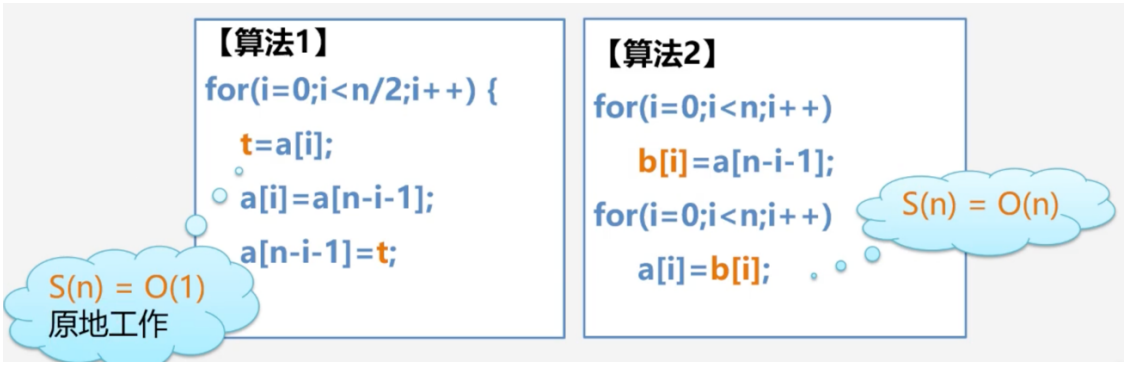
  其中n为问题的规模（或大小）

- 算法要占据的空间

  a) 算法本身要占据的空间，输入/输出，指令，常数，变量等

  b) 算法要使用的辅助空间

- 例如: 将一维数组a中的n个数逆序存放到原数组中。



第一个算法只使用了t一个变量，即空间复杂度为 1