

数据结构与算法

Data Structures and Algorithms

张正

本章主要内容

1.1	数据结构研究对象
1.2	数据结构发展概况
1.3	抽象数据类型(ADT)*
1.4	数据结构与程序设计
1.5	算法描述与算法分析*

1.1 数据结构研究对象

- ◆ **计算机科学**：信息在计算机内使用数据来表示，研究信息表示和信息处理。

什么是数据？

- ◆ **数据**：是用以描述客观事物的数值、字符，以及一切可以输入到计算机中并由计算机程序加以处理的符号的集合；
数据的基本单位称为**数据元素**；
数据的最小单位称为**数据项**。
- ◆ **数据特征**：数值、文本、多媒体、超媒体、实时、海量
- ◆ **数据对象**：性质相同数据元素的集合
- ◆ **数据类型**？高级语言中变量的取值范围和允许的操作



◆ **结构**：关系，组成整体的各部分的搭配和安排

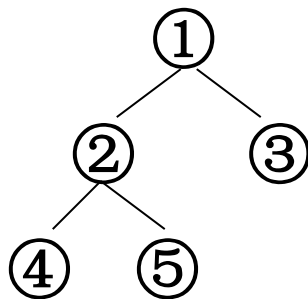
◆ **数据结构**：数据元素彼此之间抽象的相互关系，解决数据的存储和组织的方式，不涉及数据元素的具体内容。描述现实世界事物的数学模型及其操作在计算机中的表示和实现

◆ **数据结构分类**：

线性表： $(a_1, a_2, a_3, \dots, a_{i-1}, a_i, a_{i+1}, \dots, a_{n-1}, a_n) \rightarrow$ **线性结构**

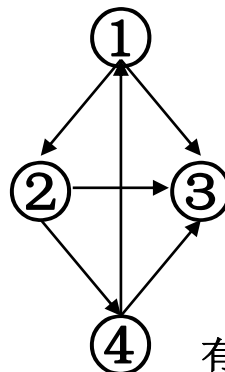
线性表的一般概念、字符串、数组，广义表等

树：层次结构
二叉树，树等




二叉树

图：网状结构
有向图、无向图等



有向图

非线性结构



数据结构

- 数学模型及其操作在计算机中的表示和实现;
- 高级语言中变量的取值范围和允许的操作的集合;



数据类型

- 一个数学模型和在该模型上定义的一组操作的集合;



ADT

1.1 数据结构研究对象

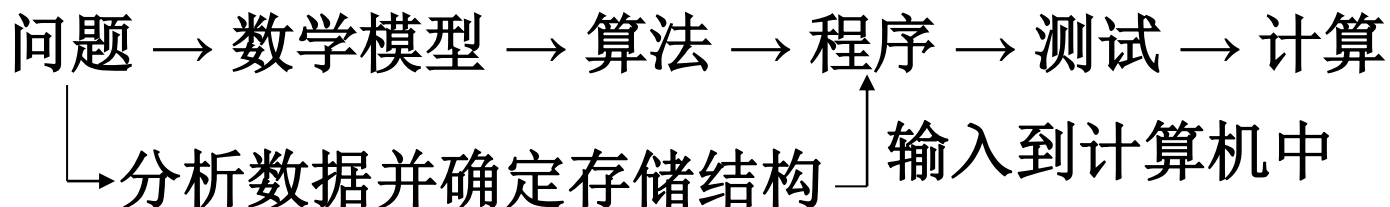
1、数据结构研究方法

逻辑结构：对操作对象的一种数学描述，或从操作对象中抽象出的数学模型，用以描述数据元素之间的逻辑关系，与存储方式无关。

物理结构：数据结构在计算机中的表示或映象，逻辑结构的存储方式。

又称存储结构，分为顺序映象和非顺序映象；
或称顺序存储结构和链式存储结构；

计算机解题过程：



关于存储结构

静态存储（顺序、数组） VS 动态存储（指针、链式）

1、线性表

数组(静态) VS 线性链表（动态），包括栈和队列

2、树

双亲表示法（静态） VS 孩子表示法（动态）

VS 孩子兄弟表示法（动态）

二叉树：

满或完全二叉树（静态） VS 二叉链表（动态）

3、图

邻接矩阵（静态） VS 邻接表（动态），包括有向图和无向图

静态存储 与 动态存储 的比较

2、数据结构研究的范畴 程序设计、算法、数据结构

程序设计：为计算机处理问题编制一组指令集

算 法：怎么处理的策略

数据结构：问题的数学模型

数值计算的程序设计问题：

问题	算法	数学模型
求解一元二次方程	?	求根公式
一组整数排序	“冒泡”方法	数组
结构静力分析计算	?	线性代数方程组
全球天气预报	?	环流模式方程

非数值计算的程序设计问题：

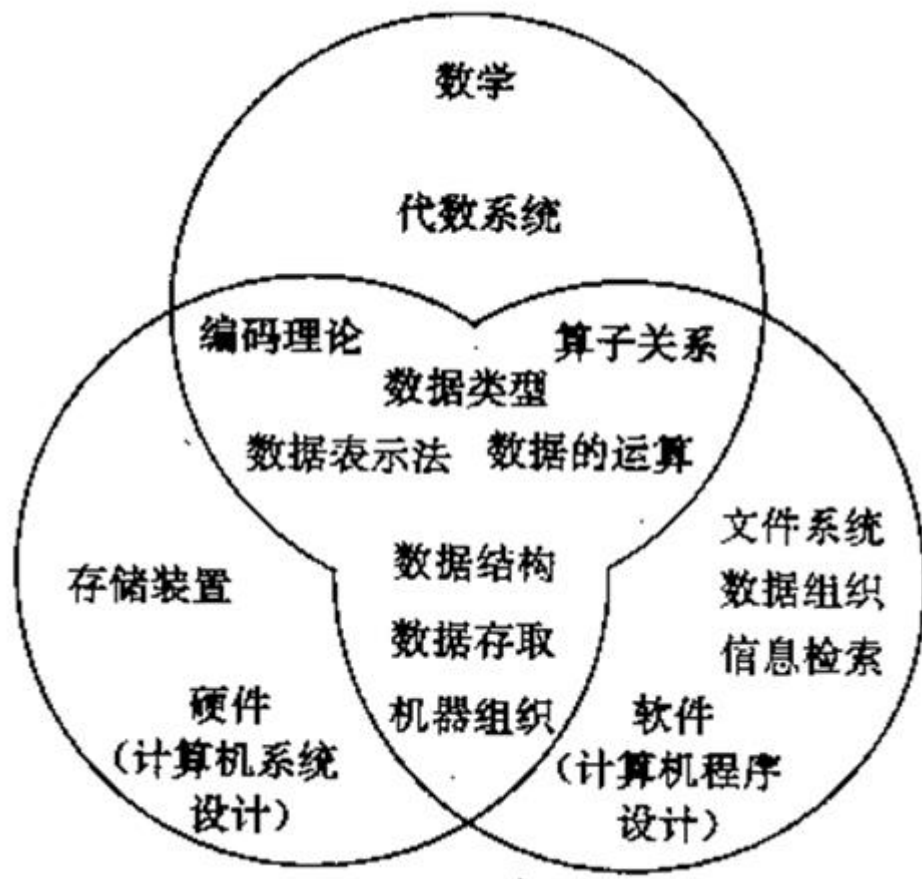
问题	算法	数学模型
求一组整数的最大值	比较两个数的大小	?
计算机对弈	对弈的规则和策略	棋子、棋盘的表示
足协的数据库管理	什么项目？ 如何管理？ 接口？ 条例？ 规则？	表格，数据库

1.2 数据结构发展概况

➤ 数据结构侧重解决非数值问题

- ✓ **FORTRAN,ALGOL**等高级语言是以程序为中心
侧重于解决数值问题;
 - ✓ **LISP,SONBOL**表或串处理语言是以数据为中心
侧重于解决非数值问题;
 - ✓ 从**PASCAL**语言开始逐渐形成二者结合;
- 1968年以前，数据结构的内容大多包含在形如表、树、图论、集合代数论、格、关系等方面。1968年开始，“数据结构”逐渐开始成为独立的一门课程;
- 作为一门专业基础课，“数据结构”是计算机专业的核心课程之一，是其他专业课的基础。是数学、硬件和软件三者的交叉，很受重视。

知识结构



《数据结构》所处的地位

1.3 抽象数据型 Abstract Data Types (ADT)

抽象：从众多事物中舍弃个别的、非本质的属性，抽出共同的、本质性的特征。

是形成概念的重要手段，其目的是使复杂度降低。

* 软件系统由数据结构、操作过程和控制机能三者组成，软件设计是对三者的抽象过程，即数据抽象、过程抽象和控制抽象。

【定义】 抽象数据型是一个数学模型和在该模型上定义的操作的集合。

ADT特点：

- ①降低了软件设计的复杂性；
- ②提高了程序的可读性和可维护性；
- ③程序的正确性容易保证。

数学模型

线性表 $LIST = (D, R)$

$D = \{ a_i \mid a_i \in \text{ElementSet}, i = 1, 2, \dots, n, n \geq 0 \}$

$R = \{ H \}$

$H = \{ \langle a_{i-1}, a_i \rangle \mid a_{i-1}, a_i \in D, i = 2, \dots, n \}$

5	8	6	3	7	...
---	---	---	---	---	-----

操作： 设L的型为LIST线性表实例，e 的型为ElementType的元素实例，i为位置变量。所有操作描述为：

① ListInsert(&L,i,e);

② ListDelete(&L,i,&e);

③ LocateElem(L,e,compare());

④ GetElem(L,i,&e);

⑤ PriorElem(L,cur_e,&pre_e);

⑥ NextElem(L,cur_e,&next_e);

⑦ ClearList(&L);

⑧ ListFirst(L); ListEnd(L).

		e				...
--	--	---	--	--	--	-----

5	8	6	3	27	...
---	---	---	---	----	-----



5	8	6	3	7	...
---	---	---	---	---	-----

ADT 抽象数据类型名 {

 数据对象：（数据对象的定义或描述）

 数据关系：（数据关系的定义或描述）

 基本操作：（基本操作的定义或描述）

} ADT 抽象数据类型名；

数据结构：

$DS = (D, R)$

基本操作（参数表） {

 初始条件：（初始条件描述）

 操作结果：（操作结果描述）

}

ADT描述：

$DS_ADT = (D, R, P)$

$= (DS, P)$

几点说明：

- (1) 数据对象+数据关系：刻画数据结构的数学模型；
- (2) 基本操作：结构上的常用的基本操作，用来设计解决问题的算法，设计算法时可以直接调用，脱离存储结构；
- (3) 基本操作往往都是通过函数来实现的，在存储结构确定后，要一一实现所有的基本操作；
- (4) 基本操作是非标准的（库函数），由用户自己定义。

ADT List {
 数据对象: $D = \{ a_i \mid a_i \in \text{ElemSet}, i = 1, 2, \dots, n, n \geq 0 \}$
 数据关系: $R1 = \{ \langle a_{i-1}, a_i \rangle \mid a_{i-1}, a_i \in D, i = 2, \dots, n \}$
 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 已存在。

操作结果:销毁线性表 L。

■ ClearList(&L)

初始条件:线性表 L 已存在。

操作结果:将 L 重置为空表。

■ ListEmpty(L)

初始条件:线性表 L 已存在。

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 已存在。

操作结果:返回 L 中数据元素个数。

■ GetElem(L, i, &e)

初始条件:线性表 L 已存在, $1 \leq i \leq \text{ListLength}(L)$ 。

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 已存在,compare()是数据元素判定函数。

操作结果:返回 L 中第 1 个与 e 满足关系 compare()的数据元素的位置。若这样的数据元素不存在,则返回值为 0。

■ PriorElem(L, cur_e, &pre_e)

初始条件:线性表 L 已存在。

操作结果:若 cur_e 是 L 的数据元素,且不是第一个,则用 pre_e 返回它的前驱,否则操作失败,pre_e 无定义。

■ NextElem(L, cur_e, &next_e)

初始条件:线性表 L 已存在。

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 返回它的后继,否则操作失败,next_e 无定义。

■ ListInsert(&L, i, e)

初始条件:线性表 L 已存在, $1 \leq i \leq \text{ListLength}(L) + 1$ 。

操作结果:在 L 中第 i 个位置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 已存在且非空, $1 \leq i \leq \text{ListLength}(L)$ 。

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 已存在。

操作结果:依次对 L 的每个数据元素调用函数 visit()。一旦 visit()失败,则操作失败。

} ADT List

- 分别求前驱元素和后继元素?
- 不如给出位置, 求前驱或后继元素在表中的位置来得方便;
- 有了元素位置可直接得到元素值。

PriorList(L,P,&q)
 NextList(L,p,&q)

ADT Triplet {

数据对象: $D = \{e_1, e_2, e_3 \mid e_1, e_2, e_3 \in \text{ElemSet} \text{ (定义了关系运算的某个集合)}\}$

数据关系: $R_1 = \{ \langle e_1, e_2 \rangle, \langle e_2, e_3 \rangle \}$

基本操作:

InitTriplet(&T, v1, v2, v3)

操作结果:构造了三元组 T, 元素 e_1, e_2 和 e_3 分别被赋以参数 v1, v2 和 v3 的值。

DestroyTriplet(&T)

操作结果:三元组 T 被销毁。

Get(T, i, &e)

初始条件:三元组 T 已存在, $1 \leq i \leq 3$ 。

操作结果:用 e 返回 T 的第 i 元的值。

Put(&T, i, e)

初始条件:三元组 T 已存在, $1 \leq i \leq 3$ 。

操作结果:改变 T 的第 i 元的值为 e。

IsAscending(T)

初始条件:三元组 T 已存在。

操作结果:如果 T 的 3 个元素按升序排列,则返回 1, 否则返回 0。

IsDescending(T)

初始条件:三元组 T 已存在。

操作结果:如果 T 的 3 个元素按降序排列,则返回 1, 否则返回 0。

Max(T, &e)

初始条件:三元组 T 已存在。

操作结果:用 e 返回 T 的 3 个元素中的最大值。

Min(T, &e)

初始条件:三元组 T 已存在。

操作结果:用 e 返回 T 的 3 个元素中的最小值。

}ADT Triplet

```
// - - - - - 基本操作的实现 - - - - -  
Status InitTriplet (Triplet &T, ElemType v1, ElemType v2, ElemType v3) {  
    // 构造三元组 T,依次置 T 的 3 个元素的初值为 v1,v2 和 v3。  
    T = (ElemType *) malloc (3 * sizeof(ElemType));    // 分配 3 个元素的存储空间  
    if (!T) exit(OVERFLOW);    // 分配存储空间失败  
    T[0] = v1;    T[1] = v2;    T[2] = v3;  
    return OK;  
} // InitTriplet  
Status DestroyTriplet (Triplet &T) {  
    // 销毁三元组 T。  
    free(T);    T = NULL;  
    return OK;  
} // DestroyTriplet  
Status Get (Triplet T, int i, ElemType &e) {  
    // 1≤i≤3,用 e 返回 T 的第 i 元的值。  
    if (i<1 || i>3) return ERROR;  
    e = T[i-1];  
    return OK;  
} // Get  
Status Put (Triplet &T, int i, ElemType e) {  
    // 1≤i≤3,置 T 的第 i 元的值为 e。  
    if (i<1 || i>3) return ERROR;  
    T[i-1] = e;  
    return OK;  
} // Put
```


【例1-1】 定义任意线性表类型为LIST，其中元素类型为Elementtype，设有线性表L，函数Purge用以删除线性表L中所有重复出现的元素。

```
Void Purge ( LIST &L )  
{ Position p, q ;  
  p = ListFirst(L) ;  
  while ( p != ListEnd(L) )  
  { NextList(L, p, &q)  
    while (q != ListEnd(L) )  
    { GetElem( L,p, &e1); GetElem( L,q, &e2);  
      if ( compare(e1,e2) )  
        ListDelete(&L,q,&e1);  
      else  
        NextList(L,q,&q) ;  
    }  
    NextList(L,p,&p) ;  
  }  
}
```

【例1-2】 假设利用两个线性表LA和LB分别表示两个集合，设计算法求一个新的集合 $A = A \cup B$ 。

```
Void Union( LIST &A; LIST B)
{ Position p ;
  ElementType e ;
  p = ListFirst( B ) ;
  while ( p != ListEnd ( B )
    { GetElem(B, p, &e ) ;
      if ( Equal(LocateElem( A,e,compare() ) , ListEnd( A ) ) )
        ListInsert( &A, ListEnd( A ), e ) ;
      NextList ( B,p,&p ) ;
    }
}
```

聚集

数组, 链表(结构体、记录), 文件

抽象数据型的规格描述

- △完整性: 反映所定义的抽象数据型的全部特征;
- △统一性: 前后协调, 不自相矛盾;
- △通用性: 适用于尽量广泛的对象;
- △不依赖性: 不依赖于程序设计语言, 可以用任意的语言来描述;

规格描述的两个方面: 语法 和 语义

- △语法: 给出操作的名称、I/O参数的数目和类型;
- △语义: 由一组等式组成, 定义各种操作的功能及相互间的关系;

例如: 抽象数据型——栈(Stack)

【定义】 栈是一个后进先出 (LIFO) 的线性表, 所有插入、删除操作是在表的一端 (栈顶) 进行。

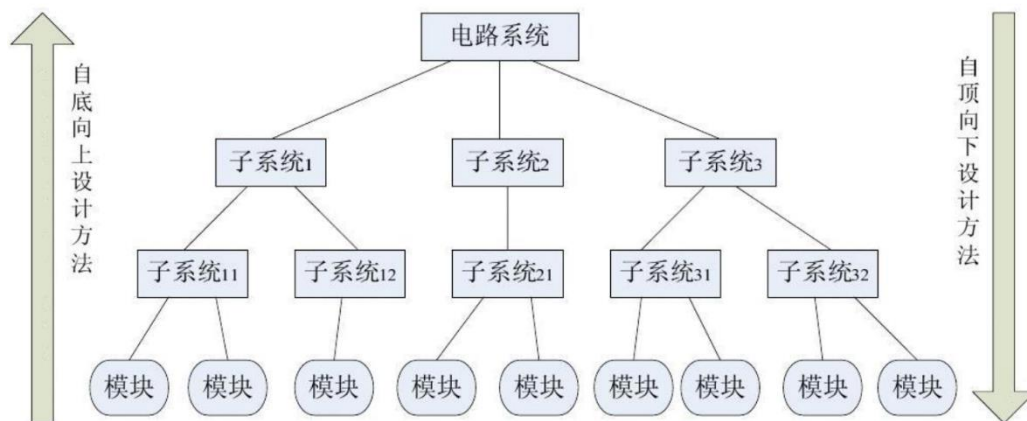
抽象数据型的实现

抽象描述：（高级语言）编写的程序

- 三条原则：
- ①符合**规格**描述的定义；
 - ②有尽可能好的**通用**性；
 - ③尽可能**独立**于程序的其他部分。

多层次抽象技术

自底向上
自顶向下
二者结合
由简单到复杂



(1) 可以用 (D) 定义一个完整的数据结构。

A. 数据元素 B. 数据对象 C. 数据关系 D. 抽象数据类型

(2) 以下数据结构中, (A) 是非线性数据结构。

A. 树 B. 字符串 C. 队列 D. 栈

(3) 以下数据逻辑结构的是 (C) 。

A. 顺序表 B. 哈希表 C. 有序表 D. 单链表

(4) 链式存储设计时, 节点内的存储单元地址 (A) 。

A. 一定连续 B. 一定不连续 C. 不一定连续 D. 部分连续, 部分不连续

(5) 以下关于数据结构的说法, 正确的是 (A) 。

A. 数据的逻辑结构独立于其存储结构

B. 数据的存储结构独立于逻辑结构

C. 数据的逻辑结构唯一决定了其存储结构

D. 数据结构仅由其逻辑结构和存储结构决定

1.4 数据结构与程序设计

Algorithms + Data Structures = Programs

算法 + 数据结构 = 程序设计

尼古拉斯·沃斯 (Niklaus Wirth, 1934年2月15日) ;

生于瑞士温特图尔, 瑞士计算机科学家;

苏黎世工学院获得学士学位, 美国加州大学伯克利分校获得博士学位;

代表性著作有:

- 《系统程序设计导论》 (《Systematic Programming: An Introduction》, Prentice-Hall, 1973;
- 《算法+数据结构=程序》 (《Algorithms + Data Structures=Programs》, Prentice-Hall, 1976) ;
- 《算法和数据结构》 (《Algorithms and Data Structures》, Prentice-Hall, 1986) ;
- 《Modula-2程序设计》 (《Programming in Modula-2》, Springer, 1988, 第4版) 。
- 《PASCAL用户手册和报告: ISO PASCAL标准》 (《PASCAL User Manual and Report: ISO PASCAL Standard》, Springer, 1991) ;
- 《Oberon计划: 操作系统和编译器的设计》 (《Project Oberon: the Design of an Operating System and Compiler》, ACM Pr., 1992) ;
- 《Oberon程序设计: 超越Pascal和Modula》 (《Programming in Oberon: Steps beyond Pascal and Modula》, ACM Pr., 1992) ;
- 《数字电路设计教材》 (《Digital Circuit Design for Computer Science Students: An Introductory Textbook》, Springer, 1995) 。

获奖:

1984年: 获图灵奖 (ACM) ;

1983年: Emanuel Piore奖 (IEEE) ;

1992年: 加州大学伯克利分校命名沃斯为“杰出校友”。

1987年: 获计算机科学教育杰出贡献奖 (ACM) ;

1988年: 计算机先驱奖 (IEEE) ;

