数据结构

北京邮电大学 信息安全中心
武 斌 杨 榆

一个实际问题

- 电话号码查询问题
  - 用C语言编写程序，查询某个城市或单位的私人电话号码。
  - 要求对任意给出的一个姓名，若该人有电话号码，则迅速找到其电话号码；否则指出该人没有电话号码。

一个实际问题—方法一

- 构造一张电话号码登记表

  表中每个结点存放两个数据项： 姓名和电话号码

- 进行顺序查询

- **查找效率低**

序号	姓名	电话号码
1	张三	13812345678
2	李四	13987654321
3	王五	15900000001
…	…	…
200	赵六	18688888888

一个实际问题—方法二

- 高效的查找算法：取决于表的结构及存储方式。
  - 表按姓氏排列
  - 另外构造一张姓氏索引表
  - 查找过程：

    先在索引表中查对姓氏，然后根据索引表中的地址到电话号码登记表中核查姓名。
  - 计算查找概率，将高概率的姓氏排在索引表前面

一个实际问题—方法二

姓氏	地址
张	xxx
李	xxx
…	
王	xxx

姓名	电话号码
张三	1xxx
张一元	1xxx
:	:
:	:
张无忌	1xxx
李四	1xxx
:	:
李寻欢	1xxx
王五	1xxx
:	:
王重阳	1xxx

数据结构的主要内容

数据的各种逻辑结构

↓

逻辑结构与物理结构及之间的相应关系

↓

为每种数据结构定义相应的各种操作

↓

设计相应的算法

↓

分析算法的效率

课程安排

●课程内容

- 第一章 绪论
- 第二章 线性表
- 第三章 栈和队列
- 第四章 串
- 第五章 数组和广义表
- 第六章 树和二叉树
- 第七章 图
- 第九章 查找
- 第十章 内部排序

课程安排

●课时安排

本课程共48学时，每周一次课，每次课3学时。

●授课方式

课堂：讲授、演示、讨论、学生参与

课下：作业、答疑

●考核方式

平时成绩、期中作业成绩、期末考试成绩

学习方法

●迭代式学习

- 研究显示，认知是一个逐步深入的过程。因此，一种合理的方法是迭代式学习，或是螺旋上升式学习。
- 这意味着，学习新知识时，知识的系统掌握程度和知识点理解深度不必一步到位。
- 先达到力所能及的程度，留着问题开始后续学习。并不是所有未解问题都会妨碍后续的学习。遇到无法绕开的问题时，再折返回到前面的章节。此时，有了对相关知识的进一步掌握，起点已经不同，难度已然下降。

学习方法

●迭代式学习

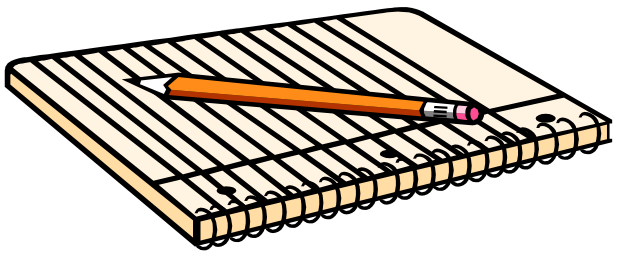
- 换位思考很重要。
- 只站在接收的角度，去理解，去记忆，貌似省力。例如，看懂算法省了设计、编写或描述算法的步骤。
- 换位为设计角度，尝试写算法解决问题，实际是“磨刀不误砍柴工”。
- 在设计和实现的过程中，会遇到问题，这既有助于理解相关算法，又有助于“欣赏”算法的美妙，能真正吸收思考和问题解决方式的精华。

本次课程学习目标

学习完本次课程，您应该能够：

- 理解数据结构
- 知道数据结构的基本概念和术语
- 了解抽象数据类型的表示与实现
- 理解算法和算法分析
- 掌握计算语句频度和估算算法时间复杂度的方法

本章课程内容（第一章 绪论）

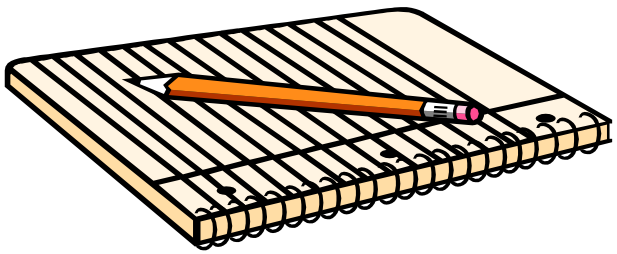


- 1.1 什么是数据结构
- 1.2 基本概念和术语
- 1.3 抽象数据类型的表示与实现
- 1.4 算法和算法分析

第一章 绪 论

- 计算机是一门研究用计算机进行信息表示和处理的科学。这里面涉及到两个问题：
  - 信息的表示
  - 信息的处理
- **信息的表示和组成，直接关系到处理信息的程序的效率**。随着计算机的普及，信息量的增加，信息范围的拓宽，使许多系统程序和应用程序的规模很大，结构又相当复杂。
- 因此，为了编写出一个“好”的程序，必须分析待处理的对象的特征及各对象之间存在的关系，这就是数据结构这门课所要研究的问题。

什么是数据结构



- 1.1 什么是数据结构
- 1.2 基本概念和术语
- 1.3 抽象数据类型的表示与实现
- 1.4 算法和算法分析

什么是数据结构

- 众所周知，计算机的程序是对信息进行加工处理。在大多数情况下，这些信息并不是没有组织，信息（数据）之间往往具有重要的结构关系，这就是数据结构的内容。那么，什么是数据结构呢？
- PASCAL之父Niklaus Wirth在1976年出版了一本书，书名为**《算法+数据结构 = 程序设计》**，并于1984年获得了图灵奖。它正说明了数据结构在程序设计中的作用。程序设计的实质即为计算机处理问题编制一组“指令集"，首先需要解决两个问题：即算法和数据结构。**算法即处理问题的策略，而数据结构即为问题的数学模型。**
- 很多数值计算问题的数学模型通常可用一组线性或非线性的代数方程组或微分方程组来描述，而大量**非数值计算问题的数学模型**正是本门课程要讨论的数据结构。

什么是数据结构

- 例如 **求 n 个整数中的最大值**

这似乎不成问题，但如果这些整数的值有可能达到10^{12}，那么对32位的计算机来说，就存在一个如何表示的问题。

- 例如 **图书馆的计算机书目检索系统**

书目文件+检索表

线性数据结构

- 例如 **人机对弈**

棋盘状态 格局

树形数据结构

什么是数据结构

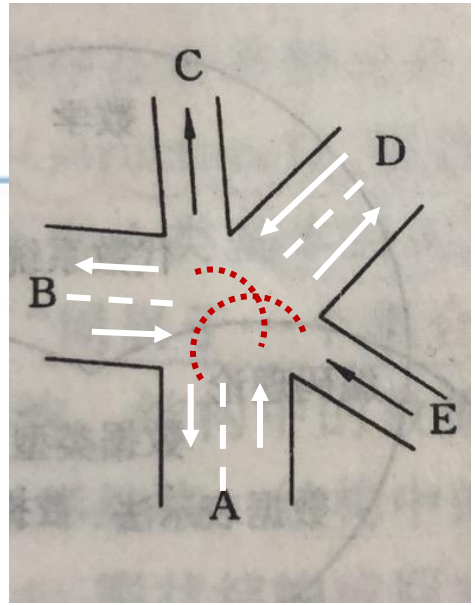
- 例如 交叉路口的红绿灯管理

如今十字路口横竖两个方向都有三个红绿灯，分别控制左拐、直行和右拐，那么如何控制这些红绿灯既使交通不堵塞，又使流量最大呢？若要编制程序解决问题，首先要解决一个如何表示的问题。

- 例如 煤气管道的铺设问题

需为城市的各小区之间铺设煤气管道，对 n 个小区只需铺设 n-1 条管线，由于地理环境不同等因素使各条管线所需投资不同，如何使投资成本最低？这是一个讨论图的生成树的问题。

什么是数据结构



●例如 交叉路口的红绿灯管理

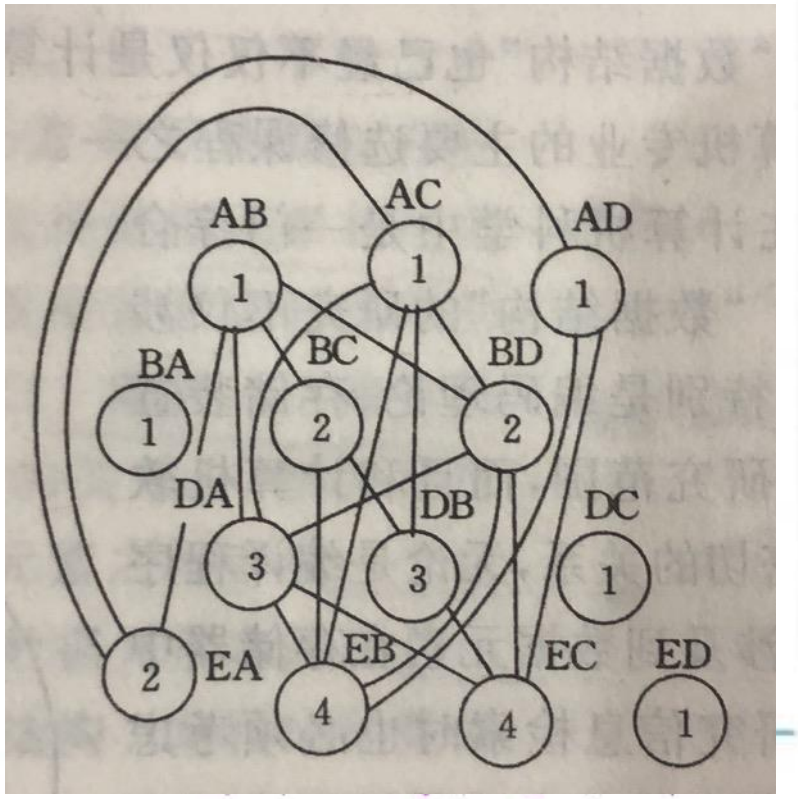
如今十字路口横竖两个方向都有三个红绿灯，分别控制左拐、直行和右拐，那么如何控制这些红绿灯既使交通不堵塞，又使流量最大呢？若要编制程序解决问题，首先要解决一个如何表示的问题。

针对具体问题的模型为：

1.把路口之间的每条通路表示为一个顶点。

2.对于不能同时同行的通路，用一条边连接其对应顶点。

3.问题转化为“颜色问题”：使用最少的颜色染完所有顶点，并且相邻（有边连接）的顶点不同色。



什么是数据结构

- 以上所举例子中的数学模型正是数据结构要讨论的问题。

因此，简单地说，数据结构是一门讨论“**描述现实世界实体的数学模型（非数值计算）及其上的操作在计算机中如何表示和实现**”的学科。

学习数据结构的意义

- 意义
  - 算法和数据结构是计算机学科的两大支柱

    早期计算机科学：研究算法的科学

    近期计算机科学：研究数据的科学
  - 数据结构是程序设计的基础

    程序=算法+数据结构
  - 数据结构是计算机专业的综合性专业基础课程

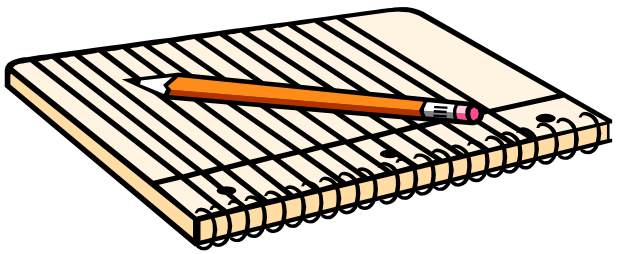
学习数据结构的要求

- **要求**
  - 掌握各类基本数据结构类型及其相应的存储结构
  - 提高阅读和编写算法的能力
  - 针对给定问题，能够选择相应的数据结构，设计和分析算法

相关资料

- 《算法（第4版）》algs4.cs.princeton.edu
- **《数据结构与算法分析**（Data Structures and Algorithm Analysis in C）》
- 《算法导论》https://visualgo.net/en*visualising data structures and algorithms through animation*
- Mooc：https://www.ichunqiu.com/mooc/course
- 文泉学堂电子书资料：数据结构算法解析（第2版），高一凡，清华大学出版社，https://lib-nuanxin.wqxuetang.com/#/Book/2130197
- 文泉学堂电子书资料：数据结构及应用算法教程（修订版）， 严蔚敏、陈文博， 清华大学出版社，https://lib-nuanxin.wqxuetang.com/#/Book/2484

基本概念和术语



- 1.1 什么是数据结构
- 1.2 基本概念和术语
- 1.3 抽象数据类型的表示与实现
- 1.4 算法和算法分析

基本概念和术语

- **数据(Data):**是对信息的一种符号表示。在计算机科学中是指所有能输入到计算机中并被计算机程序处理的**符号的总称**。
- **数据元素(Data Element):**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。
- **数据项(Data Item):** 一个数据元素可由若干个数据项组成。数据项是数据的**不可分割**的最小单位。
- **关键字：** 指的是能识别一个或多个数据元素的数据项。若能起唯一识别作用，则称之为 "主" 关键字，否则称之为 "次" 关键字。
- **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：整数数据对象是集合{0,±1, ±2, ±3,……}
- **数据结构(Data Structure)：** 是相互之间存在一种或多种特定关系的（带结构的）数据元素的集合。

例（数据、数据元素、数据项）

姓名	学号	出生日期	性别	成绩
赵一	001	1985.1.1	男	98
钱二	002	1986.2.1	女	58
孙	0	198		78
李四	004	1988.4.1	女	66
周五	005	1989.5.1	男	80
吴六	006	1990.6.1	女	72

数据元素

数据项　数据项　数据项　数据项　数据项

可以进一步分为“年、月、日”三项，所以出生日期可以成为“**组合项**”

不可分割的数据项为"原子项"。

例（数据对象）

数据对象

姓名	学号	班级	性别	成绩
赵一	001	01	男	98
钱二	002	01	女	58
孙三	003	01	男	78
李四	004	02	女	66
周五	005	02	男	80
吴六	006	02	女	72

数据对象(Data Object)：是性质相同的数据元素的集合

数据结构

若在特性相同的数据元素集合中的数据元素之间存在一种或多种特定的关系，则称该数据元素的集合为“**数据结构**”。

数据结构(Data Structure)： 带**结构**的数据元素的集合。
结构 (Structure)： 数据元素相互之间的关系。

例一： 994089985（十进制）
3B409C01（十六进制）
59.64.156.1（IP地址）
a1 a2 a3 a4
在a1、a2、a3、a4之间存在“次序”关系
<a1,a2>、 <a2,a3>、 <a3,a4>

如果颠倒次序

a2 a3 a1 a4≠ a1 a2 a3 a4
64.156.59.1 59.64.156.1

例二：二维数组{a1,a2,a3,a4,a5,a6}

a1	a2	a3
a4	a5	a6

行次序关系：

row = {<a1,a2>, <a2,a3>, <a4,a5>, <a5,a6>}

列次序关系：

col = {<a1,a4>, <a2,a5>, <a3,a6>}

如果颠倒次序则结果不同

a1	a6	a3
a5	a2	a4

≠

a1	a2	a3
a4	a5	a6

数据结构

- 例如，描述1行6列的矩阵的数据结构的定义为：它是一个含6个数据元素{a1,a2,a3,a4,a5,a6}的集合，且集合上只存在一个次序关系，即：

  {<a1,a2>，<a2,a3>，<a3,a4>，<a4,a5>，<a5,a6>}。

- 例如 假设以三个4位的十进制数表示一个含12位十进制数的“长整数”，则可用如下描述的数学模型表示：

  它是一个含三个数据元素{a1,a2,a3}的集合，且在集合上存在下列次序关系：

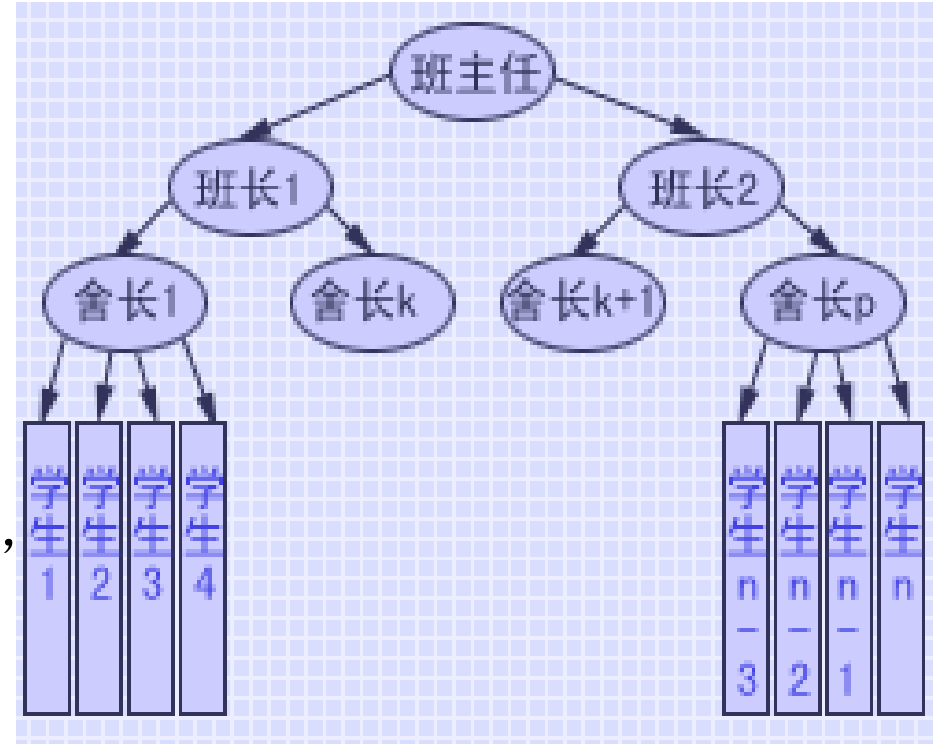
  {<a1,a2>，<a2,a3>}

  例如，长整数 "321465879345" 可用 a1=3214，a2=4658 和 a3=9345 的集合表示，且三者之间的次序关系必须是，a1 表示最高4位，a3 表示最低的4位，a2 则是中间4位。

数据结构

- 以上所举数据结构例子中的关系都是“线性关系”，数据元素之间还可能存在非线性的关系。
- 例如，某校一个年级有两个班，由一个班主任带班，每个班按所住宿舍分组，他们之间的关系可如下描述：

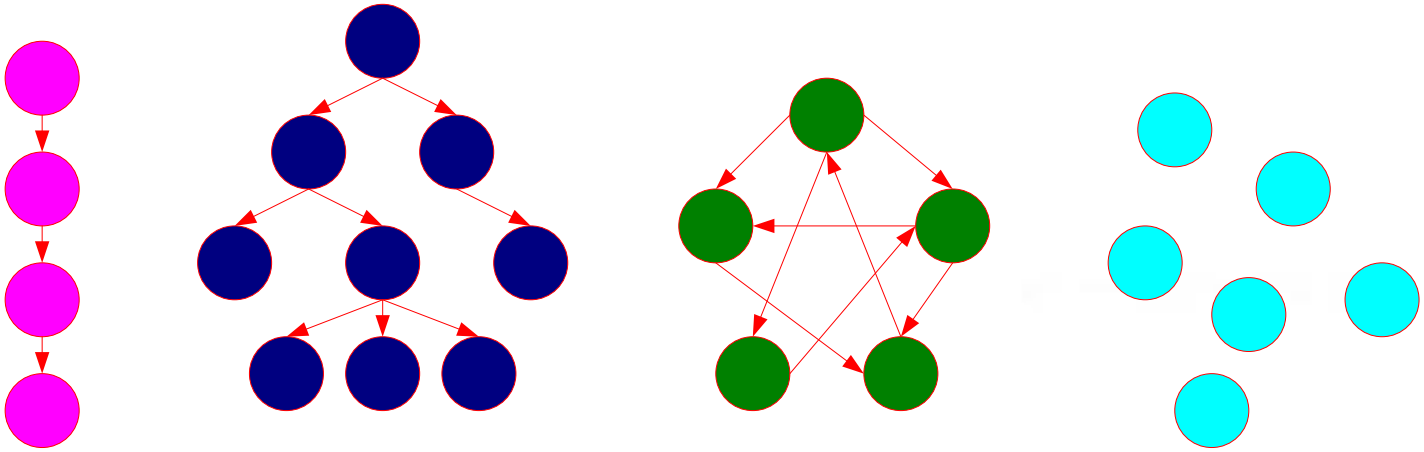
{ <班主任，班长1>，<班主任，班长2>，<班长1，舍长1>，……，<班长2，舍长p>,<舍长1，学生1>，<舍长1，学生2>，……，<舍长p，学生n> }。



数据结构的逻辑结构

数据之间的相互关系称为**逻辑结构**。通常分为四类基本结构：

- 一、**集合结构** 结构中的数据元素除了同属于一种类型外，别无其它关系。
- 二、**线性结构** 结构中的数据元素之间存在一对一的关系。
- 三、**树形结构** 结构中的数据元素之间存在一对多的关系。
- 四、**图状结构或网状结构** 结构中的数据元素之间存在多对多的关系。



线性结构　　树形结构　　网状结构　　集合结构

数据结构的形式定义

数据结构的形式定义为：

数据结构是一个二元组：Data_Structures=(D,S)

其中：D是**数据元素的有限集**，S是D上**关系的有限集**。

例　复数的数据结构定义如下：

Complex=(C，R)

其中：C是含两个实数的集合｛C1，C2｝，分别表示复数的实部和虚部。R={〈C1，C2〉}是定义在集合上的一种关系。

逻辑结构与物理结构

- 上述数据结构中的“关系”描述的是数据元素之间的逻辑关系，即“**逻辑结构**”。
- 数据结构在计算机中的表示称为数据的**物理结构**，又称为**存储结构**。
  - 存储结构是逻辑结构在存储器中的**映象**
  - 包括**数据元素的表示**和**关系的表示**

数据元素的映象方法

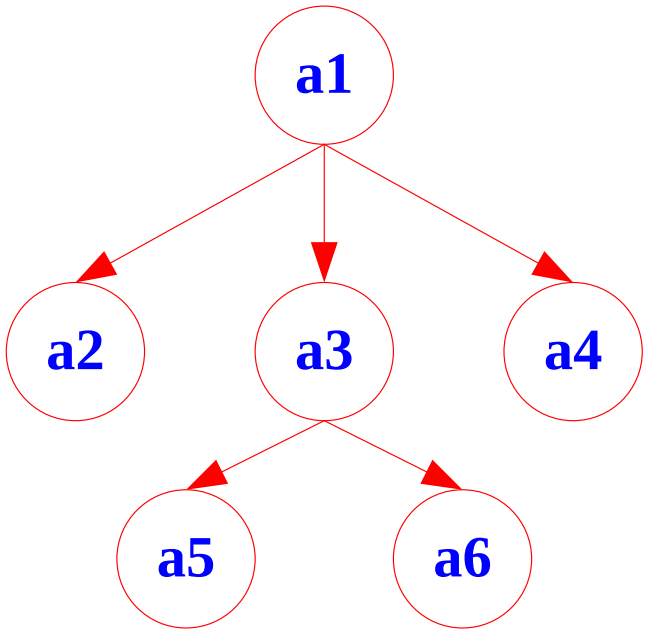
- 用二进制位(bit)的位串表示数据元素

$(321)_{10} = (141)_{16} = (501)_8 = (101000001)_2$

$A = (65)_{10} = (41)_{16} = (101)_8 = (001000001)_2$

关系的映象方法

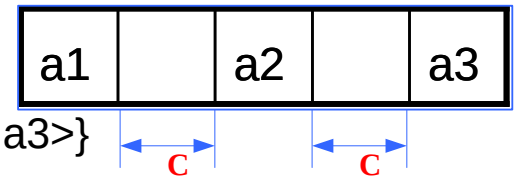
- 有序对表示：<X,Y>



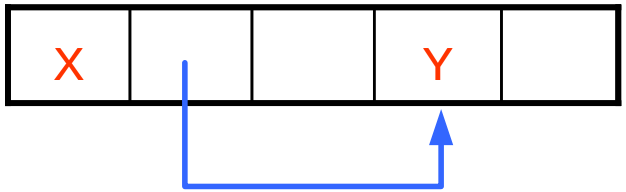
<a1,a2> <a1,a3> <a1,a4> <a3,a5> <a3,a6>

关系的映象方法

- **关系的映象方法:**表示<X,Y>的方法
- **顺序映象：**以存储位置的相邻表示后继关系
  - Y的存储位置和X的存储位置之间差一个常量C
  - C是一个隐含值，存储结构中只含数据元素本身的信息
  - 例D={a1,a2,a3}



  - R={<a1,a2>,<a2,a3>}
  - 由此得到的数据存储结构为"**顺序存储结构**"。
- **链式映象：**以附加信息（指针）表示后继关系
  - 需要用一个和X在一起的附加信息指示Y的存储位置
  - 例<X,Y>



  - 由此得到的数据存储结构为"**链式存储结构**"。

存储结构

- 存储结构的描述方法随编程环境的不同而不同，当用高级程序设计语言编程时，通常可用高级编程语言中提供的数据类型描述。
  - 例如，当以"顺序存储结构"表示前述定义的长整数时，可将它定义为：

```
typedef int Long_int[3];
```

  - 例如，定义“日期”为：

```
typedef struct {
   int y;        // 年号 Year
   int m;        // 月号 Month
   int d;        // 日号 Day
} DateType;     // 日期类型
```

  - 例如，定义"学生"为：

```
typedef struct {
   char id[8];        // 学号
   char name[16];     // 姓名
   char sex;          // 性别
   DateType bdate;    // 出生日期
} Student;            // 学生类型
```

数据结构的操作

- 对每一个数据结构而言，必定存在与它密切相关的一组操作。若操作的种类和数目不同，即使逻辑结构相同，数据结构能起的作用也不同。
- **不同的数据结构其操作集不同，但下列操作必不可缺：**
  - **(1) 结构的生成；**
  - **(2) 结构的销毁；**
  - **(3) 在结构中查找满足规定条件的数据元素；**
  - **(4) 在结构中插入新的数据元素；**
  - **(5) 删除结构中已经存在的数据元素；**
  - **(6) 遍历。**

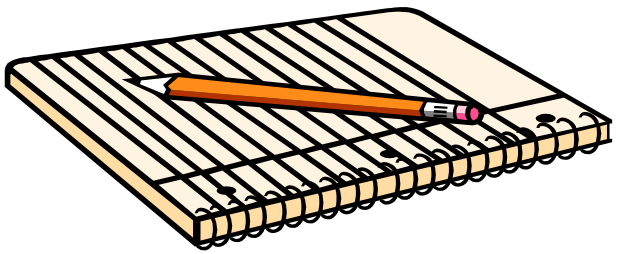
数据类型

- 高级语言编写的程序中，需对变量、常量或表达式明确数据类型。例如，C语言中的基本数据类型有：整型、字符型、实型（包括单精度型和双精度型）及枚举型。
- **数据类型**是一个“值”的集合和定义在此集合上的“一组操作”的总称。
  - 所有高级语言中都有“整型”数据类型，它们的实现方法可以各自不同，但对程序员而言，它们是“相同”的。
  - 换句话说，各种语言中实现的是同一个“整数类型”，而这个“整数类”的定义仅对“整数的数学特性”有明确规定。可称这个“整数类型”为“抽象数据类型”。
  - 程序中对变量或常量说明其所属类型的作用是，对它们加上两个约束条件：**一是可取值的范围，二是可进行的操作**。

数据类型

- 例1、 在FORTRAN语言中，变量的数据类型有整型、实型、和复数型
- 例2、在C语言中
  - 数据类型：基本类型和构造类型
  - 基本类型：整型、浮点型、字符型
  - 构造类型：数组、结构、联合、指针、枚举型、自定义

抽象数据类型的表示与实现



- 1.1 什么是数据结构
- 1.2 基本概念和术语
- 1.3 抽象数据类型的表示与实现
- 1.4 算法和算法分析

抽象数据类型

- **抽象数据类型（Abstract Data Type 简称 ADT）**是指一个数学模型以及定义在此数学模型上的一组操作。
  - 例如，矩阵的抽象数据类型定义为，矩阵是一个由 m*n 个数排成 m 行 n 列的表，它可以进行初等变换、相加、相乘、求逆、……等运算。
- 抽象数据类型有两个重要特性：
  - **数据抽象**

    用ADT描述程序处理的实体时，强调的是其本质的特征、其所能完成的功能以及它和外部用户的接口（即外界使用它的方法）。
  - **数据封装**

    将实体的外部特性和其内部实现细节分离，并且对外部用户隐藏其内部实现细节。

抽象的含义

- 描述数据类型的方法不依赖于具体实现
  - 与存放数据的机器无关
  - 与数据存储的物理结构无关
  - 与实现操作的算法和编程语言无关

只描述数据对象集和相关操作集“是什么”，而不涉及“如何做到”的问题

抽象数据类型

- 抽象数据类型的形式描述为：

  ADT = (D，S，P)

  其中：D 是数据对象，

  S 是 D 上的关系集，

  P 是 D 的基本操作集。

抽象数据类型

- **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

- 其中，数据对象和数据关系的定义用伪码描述，基本操作的定义格式为

  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〉

  - 基本操作有两种参数：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。
  - "初始条件"描述了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。
  - "操作结果"说明了操作正常完成之后，数据结构的变化状况和应返回的结果。
  - 若初始条件为空，则可省略之。

例（抽象数据类型“复数”）

- ADT Complex {

  数据对象：D = {e1,e2 | e1,e2 ∈ RealSet }

  数据关系：R1 = {<e1,e2> | e1是复数的实部，e2是复数的虚部 }

  基本操作：

  InitComplex(&Z, v1, v2)

  操作结果：构造复数Z，实部和虚部分别赋以参数v1和v2的值。

  DestroyComplex(&Z)

  初始条件：复数已存在。

  操作结果：复数Z被销毁。

  GetReal(Z, &realPart)

  初始条件：复数已存在。

  操作结果：用 realPart 返回复数Z的实部值。

  GetImag(Z, &ImagPart)

  初始条件：复数已存在。

  操作结果：用 ImagPart 返回复数Z的虚部值。

  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。

} ADT Complex

抽象数据类型的表示与实现

- 抽象数据类型可以通过**固有数据类型**表示和实现

  利用 已经存在的数据类型 表示 **新的结构**

  利用 已经实现的操作 组合 **新的操作**

- 伪码
- 类C语言

C语言实现的"复数"类型

- // 存储结构的定义

```
typedef struct {
    float realpart;
    float imagpart;
} complex;
```

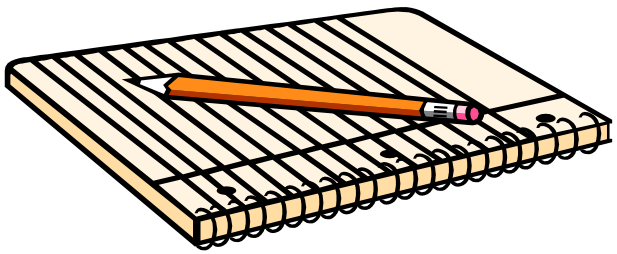
- // 基本操作的函数原型说明

```
void Assign( complex &Z, float realval, float imagval );    // 构造复数 Z，实部和虚部
void DestroyComplex( complex &Z)                            // 销毁复数 Z
float GetReal( complex Z );                                 // 返回复数 Z 的实部值
float Getimag( complex Z );                                 // 返回复数 Z 的虚部值
void add( complex z1, complex z2, complex &sum );           // sum为 z1，z2 的和
        ……
```

- // 基本操作的实现

```
void add( complex z1, complex z2, complex &sum )            // sum为 z1，z2 的和
{
    sum.realpart = z1.realpart + z2.realpart;
    sum.imagpart = z1.imagpart + z2.imagpart;
}
```

算法和算法分析



- 1.1 什么是数据结构
- 1.2 基本概念和术语
- 1.3 抽象数据类型的表示与实现
- 1.4 算法和算法分析

算法和算法分析

- **算法(algorithm)：** 是对特定问题求解步骤的一种描述，算法是指令的有限序列，其中每一条指令表示一个或多个操作。
- 算法具有以下五个特性：
  - **(1) 有穷性**
  - **(2) 确定性**
  - **(3) 可行性**
  - **(4) 输入**
  - **(5) 输出**

有穷性

- **有穷性:**一个算法必须总是在执行有穷步骤之后结束，且每一步都在有穷时间内完成。
- 这里有两重意思，即算法中的操作步骤为有限个，且每个步骤都能在有限时间（合理的时间）内完成。

确定性

- **确定性：**对于每种情况下所应执行的操作，在算法中都有确切的规定，使算法的执行者或阅读者都能明确其含义及如何执行。并且在任何条件下，算法都只有一条执行路径。
- 确定性表现在对算法中每一步的描述都没有二义性，只要输入相同，初始状态相同，则无论执行多少遍，所得结果都应该相同。

可行性

- **可行性：** 一个算法是可行的。即算法描述的操作都是可以通过已经实现的基本运算执行有限次来实现的。
- 可行性指的是，序列中的每个操作都是可以简单完成的，其本身不存在算法问题，例如，“求x和y的最大公因子"就不够基本。

输入

- **输入：**一个算法有零个或多个输入，这些输入取自于某个特定的对象集合。
- **输入**作为算法加工对象的量值，通常体现为算法中的一组变量。但有些算法的字面上可以没有输入，实际上已被嵌入算法之中。
- 输入值即为算法的操作对象，但操作的对象也可以由算法自身生成，如"求100以内的素数"，操作对象是自然数列，可以由变量逐个增1生成。

输出

- **输出：**一个算法有一个或多个输出，这些输出是同输入有着某些特定关系的量。

算法设计的要求

- 评价一个好的算法有以下几个标准:
  - (1) 正确性(Correctness)
  - (2) 可读性(Readability)
  - (3) 健壮性(Robustness)
  - (4) 效率与存储量需求

正确性

- **正确性(Correctness)** 算法应满足具体问题的需求，要以特定的规格说明方式给出。
- 算法是正确的，除了应该满足算法说明中写明的“功能”之外，应对各组典型的带有苛刻条件的输入数据得出正确的结果。
- “正确”理解的四个层次
  - 程序不含语法错误
  - 程序对于几组输入数据能够得出满足规格说明要求的结果
  - 程序对于精心选择的典型、苛刻而带有刁难性的几组输入数据能够得出满足规格说明要求的结果
  - 程序对于一切合法的输入数据都能产生满足规格说明要求的结果
- 通常以第三层意义的正确性作为衡量一个程序是否合格的标准

可读性

- **可读性：**算法应该好读。算法首先应考虑人的阅读和交流，其次是计算机的执行，因此，应有利于阅读者对程序的理解。
  - 可读性在当今大型软件需要多人合作完成的环境下是非常重要的
  - 晦涩难读的程序代码容易隐藏错误而且难以调试。

健壮性

- **健壮性：**算法应具有容错处理。当输入非法数据时，算法应对其作出反应，而不是产生莫名其妙的输出结果。
- 算法的健壮性指的是，算法应对非法输入的数据做出恰当反映或进行相应处理，一般情况下，应向调用它的函数返回一个表示错误或错误性质的值。

效率与存储量需求

- **效率与存储量需求：**效率指的是算法执行的时间；存储量需求指算法执行过程中所需要的最大存储空间。一般，这两者与问题的规模有关。

估算算法的时间效率

- 和算法执行时间相关的因素有：
  - （1）算法所用“策略”；
  - （2）算法所解问题的“规模”；
  - （3）编程所用“语言”；
  - （4）“编译”的质量；
  - （5）执行算法的计算机的“速度”。
- 显然，后三条受着计算机硬件和软件的制约，既然是"估算"，仅需考虑前两条。

算法效率的衡量方法

- 通常有两种衡量算法效率的方法：
  - **事后统计法**：收集此算法的执行时间和实际占用空间的统计资料
  - **事前分析估算法**：求出该算法的一个时间界限函数
- **事后统计法的缺点**
  - 必须在计算机上实地运行程序，容易由其它因素掩盖算法本质；
  - 容易陷入盲目境地；例如，当程序执行很长时间仍未结束时，不易判别是程序错了还是确实需要那么长的时间。
- **事前分析估算法的优点**
  - 可以预先比较各种算法，以便均衡利弊而从中选优。

估算算法的时间效率

- 一个**特定算法**“运行工作量”的大小，只依赖于问题的规模（通常用整数量n表示），或者说，它是**问题规模**的函数。

时间复杂度

- 一个算法的“运行工作量”通常是随问题规模的增长而增长，因此比较不同算法的优劣主要应该以其“增长的趋势”为准则。
- 假如，随着问题规模 n 的增长，**算法执行时间的增长率和f(n)的增长率相同**，则可记作：

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

称T(n)为算法的**(渐近)时间复杂度**。

时间复杂度

- **简化原则**：**避免将常数项或低阶项**放入大O。
- 例如：应写$T(n) = O(n^2)$，避免写为$T(n) = O(2n^2)$或$T(n) = O(n^2 + n)$。
- 我们能够通过$\lim_{n\to\infty}(f(n)/g(n))$来确定两个函数$f(n)$和$g(n)$的相对增长率，必要时可以使用洛必达法则（L'Hospital rule）。我们目前会接触到的，该极限可能的值有：
- 极限是0：这意味着$g(n)$的增长速度更快，即$f(n) = O\big(g(n)\big)$。
- 极限是$c \neq 0$：这意味着两者增长速率一样，即$f(n) = O\big(g(n)\big)$且$g(n) = O\big(f(n)\big)$。
- 极限是∞：这意味着$f(n)$的增长速度更快，即$g(n) = O\big(f(n)\big)$。

时间复杂度

- **简化原则**：**避免将常数项或低阶项**放入大O。
- 例如：应写$T(n) = O(n^2)$，避免写为$T(n) = O(2n^2)$或$T(n) = O(n^2 + n)$。
- 用相对增长率分析$n^2, 2n^2$和$n^2 + 1$
- $\lim_{n\to\infty}(n^2/2n^2) = \lim_{n\to\infty}(1/2) = 1/2$，所以$T(n) = O(n^2)$；
- $\lim_{n\to\infty}((n^2 + 1)/n^2) = \lim_{n\to\infty}(1 + 1/n^2) = 1$，所以$T(n) = O(n^2)$；
- 洛必达法则（L'Hospital rule）：
- 若$\lim_{n\to\infty} f(n) = \infty$，且$\lim_{n\to\infty} g(n) = \infty$，那么
- $\lim_{n\to\infty}(f(n)/g(n)) = \lim_{n\to\infty}(f'(n)/g'(n))$
- $f'(n)$和$g'(n)$分别是$f(n)$和$g(n)$的导数。

估算算法的时间复杂度

- 任何一个算法都是由一个“**控制结构**”和若干“**原操作**”组成的（**算法＝控制结构＋原操作**），因此一个算法的执行时间可以看成是所有原操作的执行时间之和

$$\sum(\text{原操作}(i)\text{的执行次数}\times\text{原操作}(i)\text{的执行时间})$$

- 算法的**执行时间**与所有原操作的**执行次数**之和**成正比**。
- 从算法中选取一种对于所研究的问题来说是**基本操作**的原操作，以该基本操作在算法中**重复执行的次数**作为算法时间复杂度的**依据**。
- 这种衡量效率的办法所得出的不是时间量，而是一种**增长趋势的量度**。它与软硬件环境无关，只暴露算法本身执行效率的优劣。

原操作

- “原操作”指的是固有数据类型的操作，显然每个原操作的执行时间和算法无关，相对于问题的规模是常量。
- 由于算法的时间复杂度只是算法执行时间增长率的量度，因此，只需要考虑在算法中"起主要作用"的原操作即可，称这种原操作为"基本操作"，它的重复执行次数和算法的执行时间成正比。

例（时间复杂度）

- 两个 n×n 的矩阵相乘。其中矩阵的"阶" n 为问题的规模。

```
void Mult_matrix( int c[ ][ ], int a[ ][ ], int b[ ][ ]， int n)
  { // a、b 和 c 均为 n 阶方阵，且 c 是 a 和 b 的乘积
    for (i=1; i<=n; ++i)
      for (j=1; j<=n; ++j) {
        c[i,j] = 0;
        for (k=1; k<=n; ++k)
          c[i,j] += a[i,k]*b[k,j];
      }
  }// Mult_matrix
```

- 算法中的控制结构是三重循环，每一重循环的次数是 n 。原操作有赋值，加法和乘法，显然，在三重循环之内的**“乘法”是基本操作**。
- **算法的时间复杂度为O (n^3)** 。

例（时间复杂度）

- 对 n 个整数的序列进行**选择排序**。其中序列的"长度" n 为问题的规模。

```
void select_sort(int a[], int n)
  {// 将 a 中整数序列重新排列成自小至大有序的整数序列。
    for ( i = 0; i< n-1; ++i ) {
      j = i;
      for ( k = i+1; k < n; ++k )
        if (a[k] < a[j] ) j = k;
      if ( j != i ) { w = a[j]; a[j] = a[i]; a[i] = w;}
  } // select_sort
```

- 算法中的控制结构是两重循环，所以**基本操作是在内层循环中的"比较"**，它的重复执行次数是

$$\sum_{i=0}^{n-2}(n-i-1)=\frac{n(n-1)}{2}=\frac{n^2-n}{2}$$

对时间复杂度而言，只需要取最高项，并忽略常数系数。

- **算法的时间复杂度为O (n^2) 。**

例（时间复杂度）

- 对 n 个整数的序列进行起泡排序。其中序列的"长度" n 为问题的规模。

```
void bubble_sort(int a[], int n)
  { // 将 a 中整数序列重新排列成自小至大有序的整数序列。
    for (i=n-1, change=TRUE; i>1 && change; --i) {
      change = FALSE;
      for (j=0; j<i; ++j)
        if (a[j] > a[j+1])
          { w = a[j]; a[j]= a[j+1]; a[j+1]= w; change = TRUE }
    }   } // bubble_sort
```

- 起泡排序有两个结束条件，**i=1或"一趟起泡" 中没有进行过一次交换操作**，后者说明该序列已经有序。因此起泡排序的算法执行时间和序列中整数的初始排列状态有关，它在初始序列本已从小到大有序时达最小值，而在初始序列从大到小逆序时达最大值，在这种情况下，通常**以最坏的情况下的时间复杂度为准**。

- **算法的时间复杂度为O (n^2)** 。

语句的"频度"

- 从这三个例子可见，算法时间复杂度取决于最深循环内包含基本操作的语句的重复执行次数，称语句重复执行的次数为**语句的"频度"**。

语句的"频度"

- 例如 {++x;s=0;}
  - 将x自增看成是基本操作，则语句频度为1，即时间复杂度为O(1)
  - 如果将s=0也看成是基本操作，则语句频度为2，其时间复杂度仍为O(1)，即常量阶。
- 例如 for(i=1;i<=n;++i)

  {++x;s+=x;}
  - 语句频度为：2n　其时间复杂度为：O(n)
  - 即时间复杂度为线性阶。

时间复杂度

- 以下六种计算算法时间的多项式是最常用的。其关系为：

$$O(1)<O(\log n)<O(n)<O(n\log n)<O(n^2)<O(n^3)$$

- 指数时间的关系为：

$$O(2^n)<O(n!)<O(n^n)$$

- 当n取得很大时，指数时间算法和多项式时间算法在所需时间上非常悬殊。因此，只要有人能将现有指数时间算法中的任何一个算法化简为多项式时间算法，那就取得了一个伟大的成就。

空间复杂度

- 类似于算法的时间复杂度，通常以算法的**空间复杂度**作为算法所需存储空间的量度。定义算法空间复杂度为

$$S(n)=O(g(n))$$

- 表示随着问题规模n的增大，算法运行所需辅助存储量的增长率与g(n)的增长率相同。

算法的存储空间需求

- 算法的存储量指的是算法执行过程中所需的**最大存储空间**。算法执行期间所需要的存储量应该包括以下三部分：
  - **(1)输入数据所占空间；**
  - **(2)程序本身所占空间；**
  - **(3)辅助变量所占空间。**

算法的存储空间需求

- 程序代码本身所占空间对不同算法通常不会有数量级之差别，因此在比较算法时可以不加考虑。
- 算法的输入数据量和问题规模有关，若输入数据所占空间只取决于问题本身，和算法无关，则在比较算法时也可以不加考虑，由此**只需要分析除输入和程序之外的额外空间**。
- 若额外空间相对于输入数据量来说是常数，则称此算法为**原地工作**。
- 与算法时间复杂度的考虑类似，若算法所需存储量依赖于特定的输入，则通常按**最坏情况**考虑。

本章小结

- 本章是为以后各章讨论的内容作基本知识的准备，介绍**数据结构**和**算法**等基本概念。
- **数据**是计算机操作对象的总称，它是**计算机处理的符号**的集合，集合中的个体为一个**数据元素**。数据元素可以是不可分割的原子，也可以由若干**数据项**合成，因此在数据结构中讨论的基本单位是数据元素，而最小单位是数据项。
- **数据结构**是由若干特性相同的数据元素构成的集合，且在集合上存在一种或多种关系。由关系不同可将数据结构分为四类：**线性结构、树形结构、图状结构和集合结构**。数据的**存储结构**是数据逻辑结构在计算机中的映象，由关系的两种映象方法可得到两类存储结构：一类是**顺序存储结构**；另一类是**链式存储结构**。
- 数据结构的**操作**是和数据结构本身密不可分的，两者作为一个整体可用抽象数据类型进行描述。**抽象数据类型**是一个数学模型以及定义在该模型上的一组操作，因此它和高级程序设计语言中的数据类型具有相同含义，而抽象数据类型的范畴更广。抽象数据类型的三大要素为**数据对象、数据关系**和**基本操作**，同时**数据抽象**和**数据封装**是抽象数据类型的两个重要特性。

本章小结（续）

- 算法是进行程序设计的另一不可缺少的要素。算法是对问题求解的一种描述，是为解决一个或一类问题给出的一种确定规则的描述。一个完整的算法应该具有下列五个要素：有穷性、确定性、可行性、有输入和有输出。一个正确的算法应对苛刻且带有刁难性的输入数据也能得出正确的结果，并且对不正确的输入也能作出正确的反映。
- 算法的时间复杂度是比较不同算法效率的一种准则，算法时间复杂度的估算基于算法中基本操作的重复执行次数，或处于最深层循环内的语句的频度。
- 算法空间复杂度可作为算法所需存储量的一种量度，它主要取决于算法的输入量和辅助变量所占空间，若算法的输入仅取决于问题本身而和算法无关，则算法空间复杂度的估算只需考察算法中所用辅助变量所占空间，若算法的空间复杂度为常量级，则称该算法为原地工作的算法。