数据结构与算法

张铭　赵海燕　王腾蛟

http://db.pku.edu.cn/mzhang/DS/

北京大学信息科学与技术学院

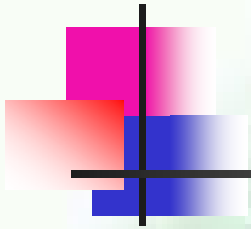
“数据结构与算法”教学小组

©版权所有，转载或翻印必究

教学目的

- 掌握常用的基本数据结构的**ADT**及其应用
- 学会合理地组织数据，有效地表示数据，有效地处理数据
- 基本掌握算法的设计分析技术
- 提高程序设计的质量

©版权所有，转载或翻印必究



- 数据结构与算法（信息学院）
  - http://db.pku.edu.cn/mzhang/DS/
- 数据结构实习（计算机专业强化）
  - http://www.db.pku.edu.cn/mzhang/DS/shixi/index.htm
- 课程答疑
  - http://db.cs.pku.edu.cn/mzhang/ds/bbs/index.asp

©版权所有，转载或翻印必究

主课教学要求

- 平时(考勤+作业)20 %
- 上机(+报告)15 %
- 期中20 %
- 期末40 %
- 态度5%

©版权所有，转载或翻印必究

考 勤

- 可以申请自学，必须写出书面申请
  - 自学的同学可以不来听课
  - 同样交作业、上机题、考试
- 不要迟到早退、旷课
  - 有事提前请假

©版权所有，转载或翻印必究

诚 信

- 端正学习态度、调动学习兴趣
  - 提倡讨论，但严禁抄袭
    - 可以讨论思路，请同学看算法的逻辑问题和效率问题
    - 但要亲自动手实现
  - 发现抄袭，则抄袭者和被抄袭者本次作业或上机题计双倍倒扣分，即得 **- 20**分。以后的作业题会得到重点检查。严重的期评将给予不及格处理
- 数据结构教学计划和要求

©版权所有，转载或翻印必究

按时提交作业，严禁抄袭

- 所有书面作业和上机作业都必须在指定的期限内完成并提交
- 一般周三交书面作业。除非不可抗拒的客观原因，请严格按提交时间完成书面作业和上机作业。例如，一个满分为10分的作业题，记分标准为：
  - （1）准时提交，满分可达10分（个别加分）；
  - （2）延迟3天之内提交，满分可达7分；
  - （3）延迟7天之内提交，满分可达3分；
  - （4）7天之后提交或不交，得分 - 5分。
  - （5）抄袭得 – 20分。

©版权所有，转载或翻印必究

书面作业提交要求

- 1) 写学号、名字
- 2) 每次作业，都在作业本或电子稿的word文档中写上“我保证没有抄袭他人作业”的诚实保证。否则，计零分或根据抄袭情况倒扣分。
- 3) 写算法分析、注释
- 4) 算法中直接使用的函数、过程先写ADT，并说明函数功能、入口参数、出口参数
- 5) 注意算法格式(层次嵌套、不同功能块之间留空)

©版权所有，转载或翻印必究

- ftp://db.pku.edu.cn
  - 计算机、智能微电子、电子
  - 用户名: s1、s2、s3
  - 密码: 课堂给出

©版权所有，转载或翻印必究

上机题提交要求

上机作业提交时打一个**zip**包，

学号+姓名+作业次数，如”**00308096**张宁**1.zip**”

包中含有：

- 1. readme.txt文件，把你的程序运行环境、编译运行步骤、程序功能等等简单说明一下。
- 2. 附加了诚实代码保证和足够注释的源程序以及相关的项目和资源文件。例如，VC++中的.dsw，.dsp文件，rc目录中的图像资源文件；Jbuilder中的.jpr或.jpx文件，特殊的Java包等等。
- **3.** 上机实习总结报告

©版权所有，转载或翻印必究

上机题编程风格要求

- 1. 诚实代码保证
- 2. 内部文档要求
- 3. 过程代码要求
- 4. 面向对象的代码要求

©版权所有，转载或翻印必究

教材

- 许卓群、杨冬青、唐世渭、张铭，《数据结构与算法》，高等教育出版社，2004年 7月。
- 张铭、赵海燕、王腾蛟，《数据结构与算法——学习指导与习题解析》，高等教育出版社，2005年 9月。

©版权所有，转载或翻印必究

参考书目

- 张铭、刘晓丹译，《数据结构与算法分析》(C++第二版、Java版)，电子工业出版社，2002年。（用1998年的第一版也可以，也可以直接用英文原版）
- 许卓群，《数据结构》，高等教育出版社，1988。
- 严蔚敏，《数据结构题集》，清华大学出版社

©版权所有，转载或翻印必究

教学参考书

- **Donald E. Knuth, *The Art of Computer Programming,* Addison Wesley. Vol. 1， Vol 3.** 国防工业出版社影印。（苏运霖译）
- **Thomas H.Cormen, Charles E.Leiserson, Ronald L. Rivest, Clifford Stein, *Inroduction to Algorithms, MIT Press*.** 高等教育出版社影印。
- **William Ford，《Data Structure with C++》，** 清华大学出版社

©版权所有，转载或翻印必究

教学参考书(续)

- 殷人昆，《数据结构——用面向对象方法与C++描述》，清华大学出版社，1999。
- 张乃孝，裘宗燕，《数据结构——C++与面向对象的途径》，高等教育出版社，2001。
- 张乃孝，《数据结构基础》，北京大学出版社。
- 严蔚敏，《数据结构》 第二版(Pascal和语言版都可以)，清华大学出版社。

第一章　概论

- 1.1　为什么要学习数据结构
- 1.2　什么是数据结构
- 1.3　抽象数据类型
- 1.4　算法的特性及分类
- 1.5　算法的效率度量
- 1.6　数据结构的选择和评价

©版权所有，转载或翻印必究

1.1 为什么要学习数据结构

- 计算机软件与理论学科的专业基础课程
- 后续专业课程学习的必要知识与技能准备
  - 编译技术要使用栈、散列表及语法树
  - 操作系统中用队列、存储管理表及目录树
  - 数据库系统运用线性表、多链表、及索引树
  - **etc.**
- 增强求解复杂问题的能力

©版权所有，转载或翻印必究

1.2 什么是数据结构

- 数据的逻辑结构
- 数据的存储结构
- 数据的运算

©版权所有，转载或翻印必究

1.2.1 数据的逻辑结构

- 反映了我们对数据含义的解释
- 数据的逻辑结构可以用一组数据（表示为结点集合K），以及这些数据之间的一组二元关系（关系集合R）来表示：**(K，R)**
  - **K**是由有限个结点组成的集合，每一个结点都代表一个数据或一组有明确结构的数据
  - 而关系集**R**是定义在集合**K**上的一组关系，其中每个关系**(relation) r**（**r∈R**）都是**K×K**上的二元关系，用它描述结点数据之间的逻辑关系

©版权所有，转载或翻印必究

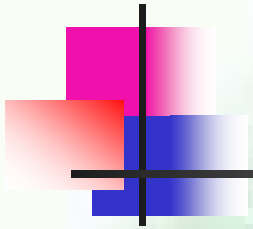
数据的逻辑结构（示例）

- 家族人员
  - 把每个成员个体的属性描述作为数据结点，而全部人员组成结点集**K**
  - 家族的各类亲属关系就是一组关系**R,** 其中如母系血缘关系**r**、远亲关系**r***、和非血缘的亲情关系**r'**等等，每一个关系要给出具体人员的关系元组
  - 例如：母子关系（王爱莲，张选）

    兄弟关系（张选，张立）

    妯娌关系 （王爱莲，李美英）

©版权所有，转载或翻印必究



结点的类型

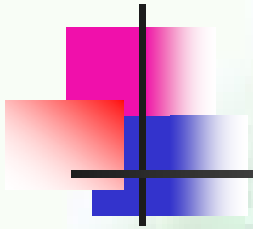
- 基本数据类型
  - 整数类型(integer)：该类型规定了所能表示的整数范围，在计算机中一般使用1个字节到4个字节来存储整数
  - 实数类型(real)：计算机的浮点数据类型所能表示的数值范围和精度是有限的。 机器一般使用4个字节到8个字节来存储浮点数
  - 布尔类型(boolean)：取值为真(true)和假(false)，在C++语言中一般使用整数0表示false，用非0表示真

©版权所有，转载或翻印必究

结点的类型

- 基本数据类型
  - 字符类型(char)：用单个字节（8bit，最高位bit为0）表示ASCII字符集中的字符。
    - 汉字符号需要使用2个字节（每个字节的最高位bit为1）的编码，单个字节对于汉字是没有独立含义的。
    - Unicode，GB，Big5，HZ

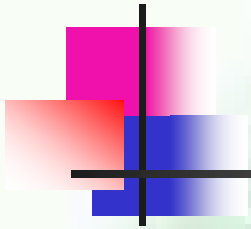
©版权所有，转载或翻印必究



结点的类型

- 基本数据类型
  - 指针类型(pointer)：用于表示机器内存地址，指针表示指向某一内存单元的地址。
    - 由于机器的指令系统一般采用32 bit或64bit的地址长度，所以指针类型也相应地用4个字节或8个字节来表示一个指针。

©版权所有，转载或翻印必究



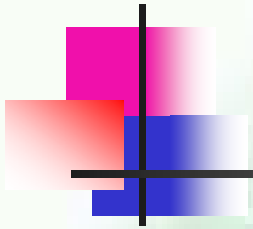
- 指针值的存储和指针值的运算方式，在形式上与正整数相似。
- 但指针的运算一般仅限于两个指针地址的比较，加减，或对一个指针增减一个整数量等

©版权所有，转载或翻印必究

结点的类型

- 复合数据类型
  - 复合类型是由基本数据类型组合而成的数据结构类型。例如：在程序语言中常用的数组类型，结构（记录）类型等皆属复合数据类型
  - 复合数据类型本身，又可以参与定义结构更为复杂的结点类型。
  - 结点的类型不限于基本数据类型，可以根据应用的需要来灵活定义

©版权所有，转载或翻印必究



结构的分类

- 讨论逻辑结构（**K,R**）的分类，一般把讨论重点放在关系集**R**上。用**R**的性质来刻划数据结构的特点，并对数据结构进行分类
  - 线性结构（**linear structure**）
  - 树型结构（**tree structure**）
  - 图结构（**graph structure**）

©版权所有，转载或翻印必究

线性结构

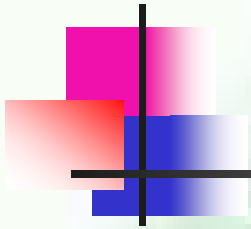
- 这种结构在程序设计中应用最多。
- 它的关系r 是一种线性关系，或称为‘前后关系’，有时也称为‘大小关系’。关系r是有向的，且满足全序性和单索性等约束条件
  - 全序性是指，线性结构的全部结点两两皆可以比较前后（关系r）
  - 单索性是指，每一个结点x都存在唯一的一个直接后继结点y。如果其他结点z在y之前，则这个z也一定在x之前，不会在x，y之间

©版权所有，转载或翻印必究

树型结构

- 树型结构简称树结构，或称为层次结构。其关系r称为层次关系，或称‘父子关系’、‘上下级关系’等
- 每一个结点可以有多于一个的‘直接下级’，但是它只能有唯一的‘直接上级’。树型结构的最高层次的结点称为根（root）结点。只有它没有父结点
- 从数学上看，树型结构和图结构的基本区别就是“每个结点是否仅仅从属一个直接上级”。而线性结构和树型结构的基本区别是“每个结点是否仅仅有一个直接后继”。
- 树型结构存在着很多变种，如二叉树结构，堆结构等，它们都有着各自独特的有效应用

©版权所有，转载或翻印必究



图结构

- 图结构有时称为结点互联的网络结构，因特网的网页链接关系就是一个非常复杂的图结构
- 对于图结构的关系r没有加任何约束。这样也就无法像线性结构及树结构那样，利用关系r的约束来设计图结构的存储结构
- 在日常应用中图结构往往只是层次结构的一种扩展——允许结点具有多个‘直接上级结点’，关系r表现为树型结构约束的放松

©版权所有，转载或翻印必究

结点和结构

- 对于数据结构（K,R），结点数据类型不限于基本数据类型，可以根据应用需要来灵活设计结点的数据类型
- 可以认为数据结构的设计是一层一层地进行的
  - 先明确数据结点，及其主要关系r
  - 在分析关系r的同时，也要分析其数据结点的数据类型
  - 如果数据结点的逻辑结构比较复杂，那么把它作为下一个层次，再分析下一层次的逻辑结构
  - 这是一种自顶向下的分析设计方法

©版权所有，转载或翻印必究

1.2.2 数据的存储结构

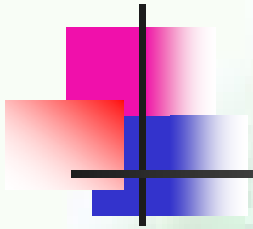
- 计算机的主存储器的特性
  - 其存储空间提供了一种具有非负整数地址编码的，相邻单元的集合，其基本的存储单元是字节
  - 计算机的指令具有按地址随机访问存储空间内任意单元的能力，访问不同地址所需的访问时间基本相同

©版权所有，转载或翻印必究

1.2.2 数据的存储结构

- 用数学上的映射来表示，数据的存储结构是建立一种映射，对于数据逻辑结构（K，r），其中r∈R
  - 对它的结点集合K建立一个从K到存储器M的单元的映射：K→M，对于每一个结点j∈K都对应一个唯一的连续存储区域c∈M。
  - 每一个关系元组（j_1，j_2）∈r（其中j_1，j_2∈K是结点），亦即j_1，j_2的逻辑后继关系应映射为存储单元的地址顺序关系（或指针的地址指向关系）。
- 四种基本存储映射方法：顺序、链接、索引、散列

©版权所有，转载或翻印必究

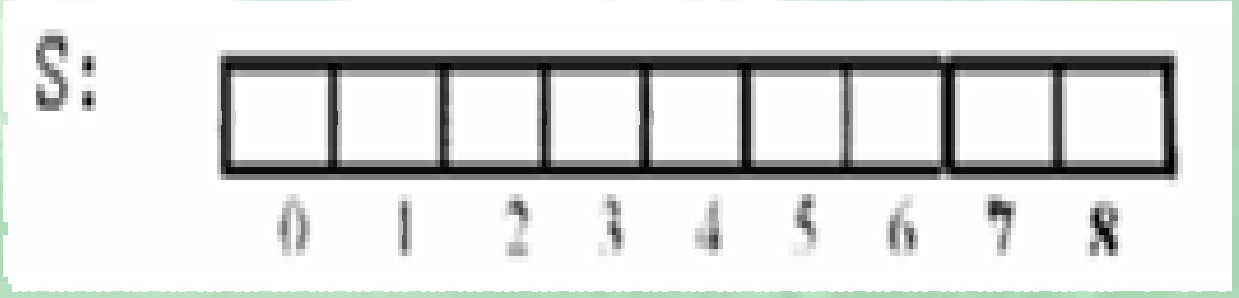


顺序（sequential）的方法

- 用一块无空隙的存储区域存储数据称为顺序存储
- 顺序存储把一组结点存储在按地址相邻的顺序存储单元里，结点间的逻辑后继关系用存储单元的自然顺序关系来表达
- 顺序存储法为使用整数编码来访问数据结点提供了便利

©版权所有，转载或翻印必究

顺序（sequential）的方法



©版权所有，转载或翻印必究

顺序（sequential）的方法

- 顺序存储结构称为紧凑存储结构，其紧凑性是指它的存储空间除了存储有用数据外，没有用于存储其他附加的信息
- 紧凑性可以用‘存储密度’来度量：它是一个存储结构所存储的‘有用数据’和该结构（包括附加信息）整个存储空间大小之比。
- 有时为了‘用空间换取时间’，在存储结构中存储一些附加信息还是很必要的。譬如用于提高算法的执行速度，或者让算法实现更为简便等

©版权所有，转载或翻印必究

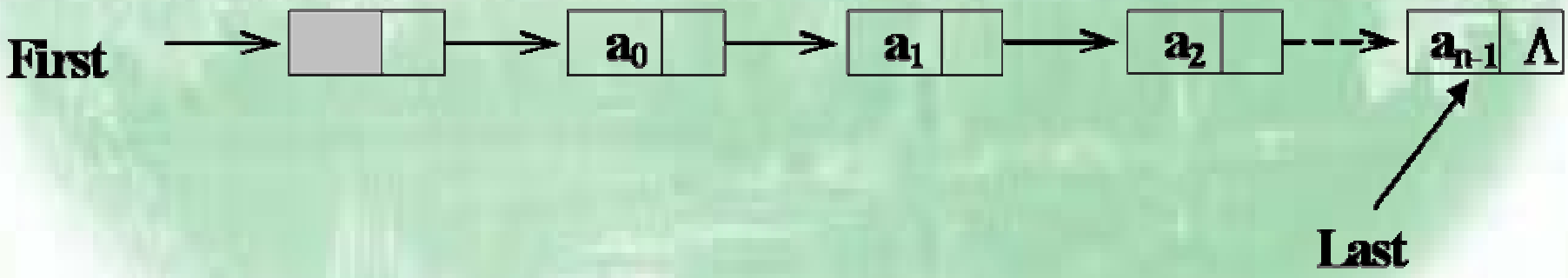
链接（linked）的方法

- 利用指针，在结点的存储结构中附加指针字段称为链接法。两个结点的逻辑后继关系可以用指针的指向来表达
- 任意的逻辑关系r，也可以使用这种指针地址来表达。一般的做法是将数据结点分为两部分：
  - 一部分存放结点本身的数据，称为数据字段
  - 另一部分存放指针，称指针字段，链接到某个后继结点，指向它的存储单元的开始地址。多个相关结点的依次链接就会形成链索

©版权所有，转载或翻印必究

链接（linked）的方法

单链表



©版权所有，转载或翻印必究

链接（linked）的方法

- 对于经常增删结点的复杂数据结构，顺序存储往往会遇到困难，链接方法结合new动态存储为这些复杂问题提供了解决方法
- 它也有缺陷：
  - 为了访问结点集K中某个结点，必须用该结点的存储指针
  - 当不知道结点指针时，为了在结点集K中寻找某个符合条件的结点，就要沿着链接结点的链索，一个个结点比较搜索。所需花费的时间是较大的

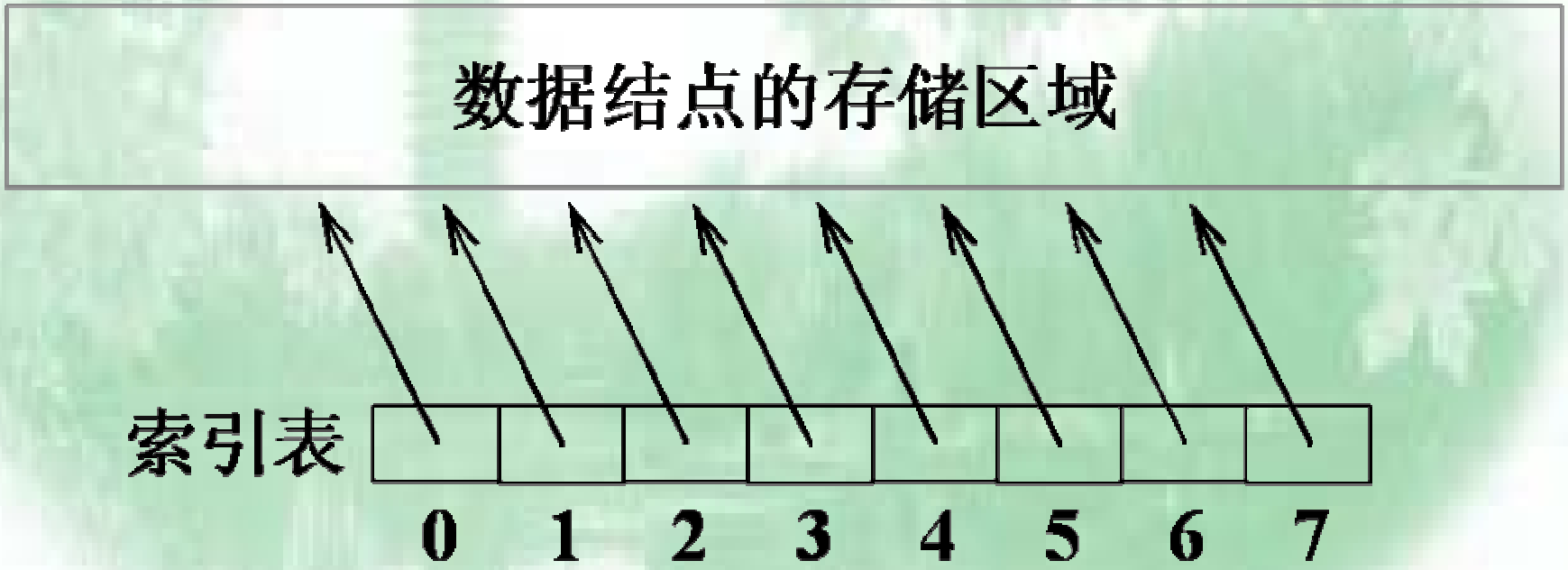
©版权所有，转载或翻印必究

索引（indexing）的方法

- 索引法是顺序存储法的一种推广，它也使用整数编码来访问数据结点位置
- 索引方法是要建造一个由整数域Z映射到存储地址域D的函数 Y：Z→D，把结点的整数索引值 z∈Z映射到结点的存储地址 d∈D 。它称为索引函数，一般而言它并不象数组那样，是简单的线性函数。
- 当数据结点长度不等的情况下，索引函数就无法用线性表达式给出

©版权所有，转载或翻印必究

索引（indexing）的方法



©版权所有，转载或翻印必究

索引（indexing）的方法

- 为了构造任意的索引函数，可以为索引函数提供附加的存储空间，称为索引表S
- 索引表中每一元素是指向数据结点的指针。因为索引表S由等长元素（指针）组成，所以可以进行线性的索引计算：

  始址(元素S[i]) ＝

  始址(元素S[0]) ＋ i •（指针尺寸）
- 通过上述公式，由索引号i可以计算出索引表中的单元S[i]的始址，再通过读出S[i]元素的内容（是指针），访问真正需要访问的数据结点

©版权所有，转载或翻印必究

索引（indexing）的方法

- 索引方法也付出了存储开销，其数据结点要附加用于指针的存储空间。
- 索引方法在程序设计中是一种经常使用的方法，其主要原因是对于非顺序的存储结构来说，使用索引表是快速地由整数索引值找到其对应数据结点的唯一方法

©版权所有，转载或翻印必究

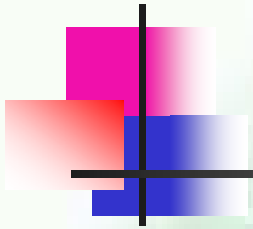
散列（hashing）的方法

- 散列方法是索引方法的一种延伸和扩展
- 利用一种称为散列函数(hash functions)

  进行索引值的计算，然后通过索引表求出结点的指针地址
- 散列函数是将字符串s映射到非负整数z的一类函数h: S → Z，

  对任意的 s∈ S，散列函数 h(s)=z，z∈ Z

©版权所有，转载或翻印必究



散列（hashing）的方法

- 散列函数h(s)除了它取非负整数值外，关键的问题：
  - 恰当地选择散列函数
  - 如何建造散列表
  - 在构建散列表的中间解决‘碰撞’的办法

©版权所有，转载或翻印必究

抽象数据类型 (abstruct data structure)

- 抽象数据类型是描述数据结构的一种理论工具
  - 特点是把数据结构作为独立于应用程序的一种抽象代数结构来描述
  - 因此在很大程度上可以使人们独立于程序的实现细节来理解数据结构的主要性质和约束条件

©版权所有，转载或翻印必究

抽象数据类型
(abstruct data structure)

- 抽象数据类型不同于具体的数据结构，前者所描述的是一种模板以及模板的结构和性质。而模板的类型参数T（元素的数据类型）必须用具体的数据类型所代入，才能成为具体的数据类型
- 抽象数据类型是把数据结构作为独立于应用程序的一种抽象，目的是使人们能够独立于程序的实现细节来理解数据结构的特性

©版权所有，转载或翻印必究

抽象数据类型 (abstruct data structure)

- 抽象数据结构 Λ 由 <取值空间，运算集>

  组成：

  - Λ 的取值空间
  - Λ 的运算集

©版权所有，转载或翻印必究

抽象数据类型
(abstruct data structure)

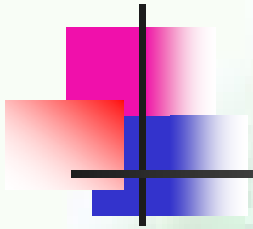
- Λ 的取值空间：给出该数据结构的所有可能取值的集合。为此需要准确地给出该数据结构 Λ 的组成元素的取值类型
- Λ 的取值空间大小取决于它的元素类型**T**。具体的数据结构是Λ 和**T**联合确定的
- 对于取值空间，进一步说明它的大小是否可以动态改变是很重要的

©版权所有，转载或翻印必究

抽象数据类型
(abstruct data structure)

- Λ的运算集：一般使用函数来定义运算，参与运算的对象成为函数的输入参数，运算结果是函数值
- 为适合不同应用领域以及它的元素类型T的不同，同一个结构的含义会有所不同，也就需要适当地添加一些新的运算种类
- 运算集的选择也会受到该数据结构的存储方案的影响

©版权所有，转载或翻印必究



ADT示例：向量vector容器

```
template<class ELEM>
class vector
//向量 类模板vector，模板参数ELEM。当程序使用此vector模
//板时，应该在前面附加 #include <vector>
{
    //1. 向量的取值类型：
    //它的元素类型为ELEM，k个元素的顺序存储成为一个向量
    //vector<double> v(100); 用于声明元素为浮点数的一个
    //向量v，程序中创建一个向量就是用这种格式的语句。其中
    //double替换了模板参数ELEM，向量长度k确定为100；
    //如果创建整型向量，那就用<int>替换模板参数
```


©版权所有，转载或翻印必究

ADT示例：向量vector容器

//2. 用名字访问，即使用变量访问向量及其元素的方法

//用整型变量 i 表示向量的下标，用v[i]访问向量的第i个元素，

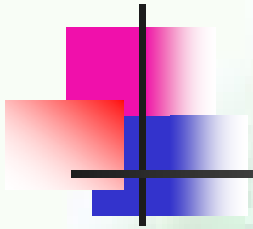
//用 语句v = w；可以将同型的向量w拷贝到向量v

//用 相等算子 v == w 可以比较两个向量的值是否全同

//3. 私有变量和 运算集：

```
private:                    // 张铭不主张涉及到private成员
   ELEM  *V[];              //私有变量，存储向量元素
   int  maxLength;          //向量最大长度
   int  currenSize;         //向量当前长度
```


©版权所有，转载或翻印必究



ADT示例：向量vector容器

```
public:
    // 创建函数有三个：
vector<ELEM>  v;     //创建一个长度为0的向量
vector<ELEM>  v(k); //创建一个长度为k的向量
vector<ELEM>  v(k,value); // 创建一个长度为k的向量
                                  //其元素的初值全部为value
int  size();     // 函数v.size()将返回整数，向量的当前长度
//还可以定义其他类似size的函数，如两个向量的加法、乘法等
}
```


©版权所有，转载或翻印必究

ADT示例：二维矩阵容器matrix

```
template<class ELEM>
class matrix//二维矩阵类模板matrix，模板参数ELEM
//当程序使用此matrix模板时，应该在前面附加 #include
//<vector> 以及#include <matrix>
{
//1. 二维矩阵的取值类型：
//它的k个元素组成行向量，然后用k个行向量组成二维矩阵
//用matrix <double> M(100);声明一个浮点的k×k的矩阵
//M. 程序中创建一个二维矩阵就可以用这样的语句其中double替
//换了模板参数ELEM，矩阵的各维的长度k＝100；
```


©版权所有，转载或翻印必究

ADT示例：二维矩阵容器matrix

```
// 2. 用名字访问，即使用变量访问矩阵及其元素的方法
// 用整型变量 i 表示下标，用M[i]访问矩阵的第I行，
//  用 语句M[i] = M[j]；可以将矩阵的第j行拷贝到第i行
//  3. 私有变量和 运算集：
typedef  vector<ELEM> Mrow;
typedef  vector< Mrow> Mmatrix;
private:
Mmatrix  amatrix;          //私有变量，存储一个二维矩阵
int  maxLength;            //存储各维的尺度
```


©版权所有，转载或翻印必究

ADT示例：二维矩阵容器matrix

```
public:
// 创建函数有三个：
Mmatrix <ELEM>  m;     //创建一个二维矩阵，尺寸待定
Mmatrix <ELEM> m(k); //创建一个k×k的二维矩阵
//创建k×k的二维矩阵
Mmatrix <ELEM> m(k,Mrow(k,value);
//其元素的初值全部为value
int  v.size()                    // 返回函数值为矩阵的当前长度
//可以补充定义其他函数，如两个矩阵的加法、乘法等
}
```


©版权所有，转载或翻印必究

1.4 算法及其特性

- 算法(algorithms)是为了求解问题而给出的指令序列
- 程序是算法的一种实现，计算机按照程序逐步执行算法，实现对问题的求解
- 一个求解问题通常用该问题的输入数据类型和该问题所要求解的结果（算法的输出数据）所应遵循的性质来描述

©版权所有，转载或翻印必究

1.4.1 算法及其特性

- 一定的通用性
  - 对于那些符合输入类型的任意输入数据，都能根据算法进行问题求解，并保证计算结果的正确性
- 算法的有效性
  - 算法是有限条指令组成的指令序列，其中每一条指令都必须是能够被确切执行的，被人或机器所执行。指令的类型应该明确规定，仅限于若干明确无误的指令动作，是一个有限的指令集 。其结果应具有确定的数据类型，是能够预期的

©版权所有，转载或翻印必究

1.4.1 算法及其特性

- 算法的确定性
  - 算法每执行一步之后，关于它的下一步，应该有明确的指示。下一步动作可以是条件判断、分支指令、或顺序执行一条指令、或者指示整个算法的结束等。算法的确定性要保证每一步之后都有关于下一步动作的指令，不能缺乏下一步指令（被锁住）或仅仅含有模糊不清的指令
- 算法的有穷性
  - 算法的执行必须在有限步内结束。换句话说，算法不能含有死循环

©版权所有，转载或翻印必究

1.4.2 计算复杂性和算法的效率

- 根据计算理论(**theory of computation**)，存在着一类问题虽然能够被准确定义，但却不存在能够解决该问题的算法。称为不可解问题
- 计算复杂性理论(**computational complexity theory**)指出，理论上存在一大类难解问题，它们虽然存在着求解算法，但是在算法的计算时间上，都是组合爆炸型的求解算法

©版权所有，转载或翻印必究

1.4.2 计算复杂性和算法的效率

- 所谓组合爆炸型，是指随着问题的规模**n**的增大，算法的时间开销不能约束在**n**的**k**阶多项式数量范围内。（其中**k**是任意不依赖于**n**的常数）
- 根据计算复杂性理论，难解问题的定义就是它的求解算法均无法在多项式时间n^k数量级内解决（其中**k**是任意正整数）。比较常见的难解问题有：图论中的求最优巡游路径问题，判定命题逻辑公式是否为恒真等

©版权所有，转载或翻印必究

1.5 算法的执行效率及其度量

- 解决同一个问题总是存在着多种算法，而算法设计者在所花费的时间和所使用的空间资源往往要两者之间采取折中，采用某种以空间资源换取时间资源的策略
- 算法设计者可以通过算法分析，判断所提出的算法是否现实，设计出更好的算法

©版权所有，转载或翻印必究

1.5.1算法的渐进分析 (asymptotic analysis)

- 算法的渐进分析，简称算法分析。
  - 算法在计算机上实际执行时，需要消耗时间资源（归结为CPU执行指令的总数），和使用空间资源（归结为所需占用的存储单元数量，字节数）。
- 问题规模n
  - 由于算法分析和它所求解的问题规模直接有关，因此通常将作为一个参照量，求算法的时空开销和n的关系。

©版权所有，转载或翻印必究

1.5.1算法的渐进分析 (asymptotic analysis)

- 算法的渐进分析估计
  - 当数据规模**n**逐步增大时，资源开销**T(n)**的增长趋势
- 从数量级大小的比较来考虑，当**n**增大到一定值以后，资源开销的计算公式中影响最大的就是**n**的幂次最高的项，其他的常数项和低幂次项都是可以忽略的。

©版权所有，转载或翻印必究

1.5.1算法的渐进分析 (asymptotic analysis)

- 算法的渐进分析就是要得到一个大O渐进表达式，简写为：

  $\text{rate}_{n\to\infty}\ T(n) = O(F(n))$

- 其中O是数学分析常用符号‘大O’，而F(n)是自变量为n的某个具体的整函数表达式，例如$F(n)=n^2$

©版权所有，转载或翻印必究

算法的渐进分析示例
矩阵求和（matrix_addition ）

```
void matrix_addition(double **M1, double **M2, int k)
{
// 矩阵M1,M2求和，即两两元素求和，结果存回M1
// k×k是矩阵的规模
 for (int i=0 ; i<k; i++)
    for (int j=0 ; j<k; j++) //对应位置的矩阵元素相加
        M1[i][j] = M1[i][j] +M2[i][j] ;
}
```


©版权所有，转载或翻印必究

算法的渐进分析示例
矩阵求和（matrix_addition ）

- 此问题的规模由输入数据规模决定，主要是浮点矩阵**M1, M2**的规模**n＝2*k²**
- 在渐进分析时，通常我们把浮点运算和数组元素的读写操作的执行时间都看作在同一个数量级内，忽略其执行时间的细微差别

©版权所有，转载或翻印必究

1.5.1算法的渐进分析 (asymptotic analysis)

- 用于渐进分析的常见的**F(n)**还有以下若干种：
  - **F(n) = 1,** 常数函数，不依赖于**n**
  - **F(n) = log n,** 对数函数，它比线性函数**n**增长慢
  - **F(n) = n,** 线性增长，随着问题规模**n**而增长

    例如，**rate n→∞**（**n**个**a**相加）

    = rate n→∞ （n•a）

    = O(n)
  - **F(n) = n^2**，二阶增长

    例如，**rate n→∞** $\Sigma_{i=1..n}\ \Sigma_{j=i..n}$ **(a) = O(n^2)**，这是因为

    $\Sigma_{i=1..n}$（**a •(n－i)**）＝（**a•(n•(n-1)/2)**

©版权所有，转载或翻印必究

1.5.1算法的渐进分析 (asymptotic analysis)

- 用于渐进分析的常见的**F(n)**还有以下若干种：
  - **F(n)= n • log(n)**，其增长率的阶数低于二阶，但高于一阶线性。例如，

    rate n→∞ $\sum_{i=1..log\,n} \sum_{j=i..n} (a) = O(n \bullet \log(n))$
  - **F(n)= a^n** 指数增长，随问题规模**n**而增长极快

©版权所有，转载或翻印必究

1.5.2最坏、最好、和平均情况

- 在具体进行算法增长率估计时，往往会由于算法中的条件分支而遇到困难。
- 由于算法实际执行的操作往往依赖于分支条件的走向，而输入数据的取值又影响这些分支走向，因此很多算法都无法得出独立于输入数据的渐进估计。
- 针对这一情况，提出了最坏情况估计、平均情况估计、和**Theta**（Θ希塔）估计等三种方法

©版权所有，转载或翻印必究

1.5.2示例
求矩阵M中绝对值最大的元素

```
double abs_biggest(double **M, int k)
 {//求矩阵M中绝对值最大的元素，k×k是矩阵的规模
 double  current_big = 0;//临时存储单元，初始化为0
 for (int i=0 ; i<k; i++)
    for (int j=0 ; j<k; j++){//对每一个矩阵元素进行大小比较
        //fabs是浮点数求绝对值函数
        double temp =  fabs(M[i][j]);
        if ( temp > current_big)
            current_big = temp; // current_big存储当前最大者
 return current_big; //函数值返回
}
```


©版权所有，转载或翻印必究

1.5.2示例
求矩阵M中绝对值最大的元素

- 输入参数为浮点矩阵M，它的大小是影响算法时空开销的主要因素。令n＝k×k为这个问题的规模
- 在最坏的情况下，每一次判断(temp) > current_big)都为真，赋值语句执行n = k×k次
- 在最好的情况下（在第一次对current_big赋值后，程序再没有对它作赋值操作），则基本没有赋值时间开销
- 平均而言，在假定浮点矩阵的主元素（绝对值最大）位置随机分布的情况下，可以说，其平均时间开销正比于n/2 = k×k/2

©版权所有，转载或翻印必究

1.5.2最坏、最好、和平均情况

- 对于时间开销，一般不注意算法的‘最好估计’。特别是处理应急事件，计算机系统必须在规定的响应时间内做完紧急事件处理。这时，最坏估计是唯一的选择
- 对于多数算法而言，最坏情况和平均情况估计两者，它们的时间开销的公式虽然不同，但是往往只是常数因子大小的区别，或者常数项的大小区别。因此不会影响渐进分析的增长率函数估计

©版权所有，转载或翻印必究

1.5.3 时间和空间资源开销

- 对于空间开销，也可以实行类似的渐进分析方法
  - 很多算法使用的数据结构是静态的存储结构，即存储空间在算法执行过程中并不发生变化。
    - 对于静态数据结构，空间开销的估计往往是容易的，它们或者与所涉及的问题规模成正比（空间开销为线性增长），或者不随问题的规模而增大，空间开销为常数。

©版权所有，转载或翻印必究

1.5.3 时间和空间资源开销

- 使用动态数据结构算法的存储空间是变化的，在算法运行过程中有时会有数量级地增大或缩小。对于这种情况，空间开销的分析和估计是十分必要的

©版权所有，转载或翻印必究

时空资源的折中原理

- 对于同一个问题求解，一般会存在多种算法。而这些算法在时空开销上的优劣往往表现出‘时空折中’的性质
- ‘时空折中’，是指为了改善一个算法的时间开销，往往可以通过增大空间开销为代价，而设计出一个新算法来
- 有时也可以为了缩小算法的空间开销，而牺牲计算机的运行时间，通过增大时间开销来换取存储空间的节省

©版权所有，转载或翻印必究

1.5.4 大Θ表示法及其分析规则

- 大Θ（希塔）表示法是算法分析的一种渐进估计，简称Θ希塔估计。它是前面讨论的大O表示的扩展
- 对于任意的资源开销函数**T(n)**，以及对**T(n)**的一个渐进估计式**F(n)**，称**F(n)**为**T(n)**的一个渐进Θ希塔估计，当且仅当满足如下数学性质：

  存在正常数C_1,C_2，以及正整数n_0，使得对于任意的正整数$n > n_0$有下列两不等式同时成立：

  $| T(n) | > C_1 \bullet F(n)$ 和 $| T(n) | < C_2 \bullet F(n)$

©版权所有，转载或翻印必究

1.5.4 大Θ表示法及其分析规则

- 对于算法的资源开销，可以给出它的下界估计和上界估计两种估计式
- 如果一个算法存在一个**F(n)**渐进估计式，它可以同时充当它的上界和下界的渐进估计，那么就把这个**F(n)**称作算法开销**T(n)**的希塔估计，或称Θ估计

©版权所有，转载或翻印必究

1.5.4 大Θ表示法示例

假定$T(n)=100 \bullet n^2+5 \bullet n+500$，

令$F(n)= n^2$，这时存在正常数

$C_1=100$，$C_2=105$，以及 $n_0=10$

使得当$n > n_0$时，

$|T(n)| > C1 \bullet F(n)$ 和 $|T(n)| < C2 \bullet F(n)$

同时成立。

上述不等式可以用算术计算验证，由此说明$T(n)$的希塔估计式等于$F(n)= n^2$

©版权所有，转载或翻印必究

1.6 数据结构的选择和评价

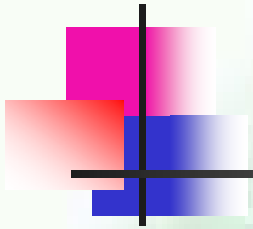
- 仔细分析所要解决的问题，特别是求解问题所涉及的数据类型和数据间逻辑关系
- 数据结构的初步设计往往在算法设计之先
- 注意数据结构的可扩展性。包括考虑当输入数据的规模发生改变时，数据结构是否能够适应 。同时，数据结构应该适应求解问题的演变和扩展
- 数据结构的设计和选择也要比较算法的时空开销的优劣

©版权所有，转载或翻印必究

算法的度量

- 算法设计的两个目标：
  - 易读、易编码和调试（软件工程）
  - 充分利用计算机资源（算法和数据结构）
- 评估方法
  - 实验（运行程序）
  - 渐进分析
  - 关键资源
  - 影响运行时间的因素
  - 往往与问题的输入规模n有关

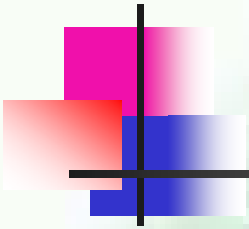
©版权所有，转载或翻印必究



时间/空间权衡

- 数据结构
  - 一定的空间来存储它的每一个数据项
  - 一定的时间来执行单个基本操作
- 代价和效益
  - 空间和时间的限制
  - 程序设计工作量

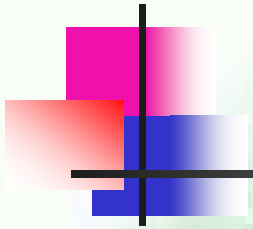
©版权所有，转载或翻印必究



总结

- 数据结构的地位与重要意义
- 数据结构的主要内容
- 抽象数据类型的概念
- 算法及其特点
- 算法的有效性度量
- 数据结构的选择

©版权所有，转载或翻印必究



The End

Thank you!

©版权所有，转载或翻印必究