

数据结构

DATA STRUCTURE

陶文兵

wenbingtao@hust.edu.cn



群号: 1047920004

教学计划

一、课堂教学（36学时）

- ❖ 第一章 绪论
- ❖ 第二章 线性表
- ❖ 第三章 堆栈和队列
- ❖ 第四章 字符串
- ❖ 第五章 数组和矩阵
- ❖ 第六章 树
- ❖ 第七章 图
- ❖ 第八章 排序
- ❖ 第九章 查找

二、上机实验（12学时）

- ❖ 时间：第4-12周（共3次）
- ❖ 地点：逸夫科技楼12楼机房

三、考核方式

- ❖ 平时成绩（平时作业+上机，30%）
- ❖ 期末考试（70%）

四、作业

- 每次课堂结束后均有程序编写任务
- 除了程序以外，需另外附一篇**调试程序报告**，谈谈自己在调试程序中的体会，所遇到的问题，如何解决的。
- 作业由每个班的**学习委员**收齐，压缩后发到我的 `wenbingtao@hust.edu.cn` 邮箱。每个同学的作业包含两个文件，一个是**程序文件*.c**，一个是**文档文件*.docx**，文件的命名规则如下，比如一个同学的学号是M2341，姓名是张三，那么两个文件的名称分别是：

M2341_张三.c

M2341_张三.docx

再一次提醒：每个同学的两个文件不要单独压缩，不然每个人的压缩文件我都解压一下会很浪费时间，只有学习委员收齐作业后统一压缩成一个文件再发给我。

参考书目

- ❖ 严蔚敏，吴伟民. **数据结构(C语言版)** . 北京：清华大学出版社, 2007.
- ❖ 许卓群等. **数据结构与算法**. 北京：高等教育出版社, 2004.
- ❖ Donald E. Knuth. **计算机程序设计艺术**(第3版). 苏运霖译. 北京：国防工业出版社, 2007.
(**第一卷 基本算法；第二卷半数值算法；第三卷 排序与查找**)

C语言回顾

- ❖ C语言的成份
- ❖ C语言语句
- ❖ C语言变量
- ❖ C语言的数据类型
- ❖ 函数
- ❖ 程序结构
- ❖ 结构和联合
- ❖ 指针

C语言成份

❖ C语言的成份：

– 语句

– 定义

- 变量定义
- 函数定义

– 申明

- 变量申明
- 数据类型申明
- 函数原型申明
- 编译指示申明

– 注释

C语言语句

❖ C语言的语句用“;”标记。分为以下几种：

- 赋值语句/表达式语句
 - 函数调用(子程序)
 - 流程控制语句
 - 空语句
- 复合语句
(用一对{ }括起来)

C语言变量的定义与申明

◆ 变量的定义

存储类型 数据类型 变量名 = 初始值;

```
int x;
```

```
int x = 3;
```

```
static int x = 3;
```

- 变量的作用域：变量只在定义它的(函数)范围内起作用
- 在所有函数体之外定义的变量称全局变量

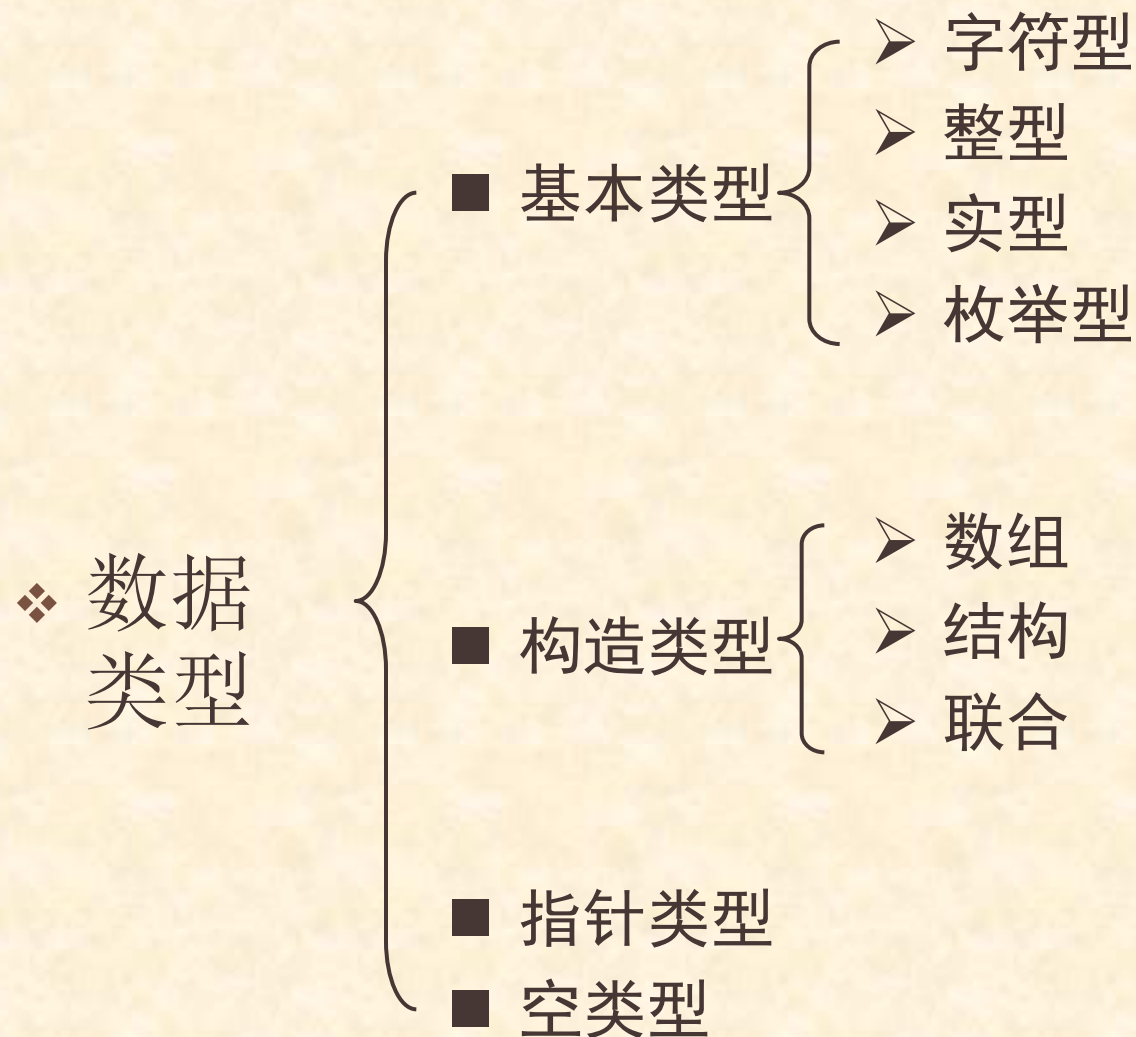
◆ 变量的申明

```
extern int Conter;
```

```
void strcpy(const char *Tar, char *Src);
```

- 变量的申明通常是给编译器以指示，并不实际分配内存。因此通常放在头文件中。

C语言的数据类型



C语言基本数据类型

- ❖ C语言的基本数据类型包含：字符型、整型、实型、枚举型；
- ❖ 不同的C编译器可能对一些类型的在内存中的字节数规定不同，可以用sizeof()检测各种类型所需的存储字节数。

| C语言基本数据类型 | 字节数 |
|---|-----|
| unsigned/signed char | 1 |
| unsigned/signed short (int) | 2 |
| unsigned/signed int | 4 |
| unsigned/signed long (int) unsigned/signed float | 4 |
| enum | 4 |
| unsigned/signed int64 unsigned/signed double | 8 |

| Windows API类型 | 存储 字节数 |
|------------------|-----------|
| BYTE | 1 |
| WORD | 2 |
| DWORD | 4 |

数组

❖ 数组的定义

存储类型 数据类型 数组名[元素个数]= {初始值表};

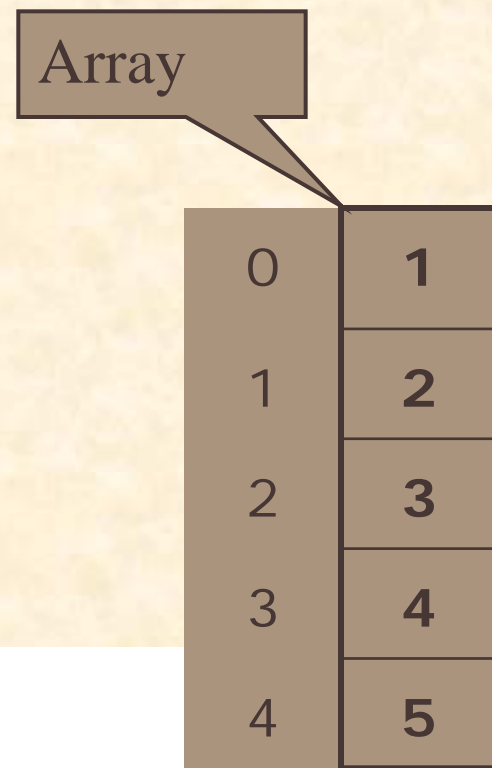
- 数组的初值
- 数组在内存中的存储：顺序存储
- 数组的引用：以元素下标引用
 - 下标总是从0开始，到元素个数-1

❖ 数组引用的实质：地址+偏移

- 数组等同于一个指针

❖ 数组的申明

```
static int Array[3];
```



字符数组与字符串

◆ 字符数组

- 定义：跟上面的数组一样，数据类型为char
- 初值：跟上面一样，但增加了一种

◆ 字符串：字符数组的最后一种形式，自动加0的字符数组

```
char *str = "Hello";
```

函数

❖ 函数定义

存储类型 返回类型 函数名(形参列表) {函数体语句}

返回类型: **void**型相当于子程序

形式参数: 传值型、传址型。注意申明形式

函数调用: 非**void**返回值可赋给变量;可直接作语句用。

```
int max(x, y)
    int x, y;
{
}

```

```
int max( int x, int y)
{
}

```

```
void show_message( char *Msg)
{
}

```

```
static int WinApi( int &x, int &y)
{
}

```

```
char *strcpy( const char *Tar, char *Src)
{
}

```

函数

❖ 函数申明

extern 返回类型 函数名(形参[类型]列表);

注意：有；符号。形参可只给类型。

通常函数申明放在头文件中。有时也叫函数原型。

```
extern int max( int, int);
```


程序结构

❖ 顺序结构

- 按照语句的顺序执行

❖ 流程控制

- 选择结构

- if – else
- switch

- 循环结构

- for
- while { }
- do { } while
- do { } until

- 跳转 goto

□ 1966年，Bohra和Jacopini证明了
可以只用三种结构表达以上各种结构



■ 顺序结构

■ 选择结构 if – else

■ 循环结构 for

结构

❖ 结构定义

```
struct 结构名 {  
    结构成员类型    结构成员名;  
    .....  
};
```

❖ 结构使用

```
struct 结构名 变量名;
```

```
struct StudentInfo aStudent;  
struct StudentInfo Students[35];  
struct StudentInfo *lpStuentsList;  
int x = aStudent.StID;  
int y = Students[10].StID;
```

联合

❖ 联合定义

```
union联合名 {  
    联合成员类型      联合成员名;  
    .....  
};
```

```
struct Resident {  
    int    CertID;  
    char   Name[16];  
};  
  
union Person {  
    struct Resident    Res;  
    struct Student     St;  
    Char               InfoData[20];  
};
```

自定义数据类型

- ❖ 使用typedef定义自定义的数据类型

typedef 原数据类型

用户数据类型

- ❖ 使用：等同于系统预定义的数据类型
- ❖ 自定义数据类型与宏定义不同，宏定义只是展开，因此数据类型不变，而自定义类型则不同，是一种新的数据类型。很多编译器在编译时进行类型检查，以确定是否违例

| | |
|-----------|-----------------|
| BYTE | aByte, *lpByte; |
| Student | aStudent; |
| LpStudent | lpStudentList; |

地址指针

- ❖ 程序在内存中才能得以执行，数据在内存中才能被直接访问。内存单元是以地址标记的。
- ❖ 定义：
 - 数据类型 * 变量名;
 - `Int *MyInt;`
 - `Struct Student *lpStudentsList;`
- ❖ 使用：
 - `*MyInt = 30;`
 - `lpStudentsList->StID = 1234;`
- ❖ 注意：指针是一种类型，指针变量也占用内存。

函数指针



- ◆ 每一个函数被编译成代码后，都会装入内存以执行，它们在内存中都会有一个入口地址。可以用一个变量表示这个入口地址，称为指向这个函数的指针。
- ◆ 在编写程序时，程序还没编译，如何知道函数指针呢？
 - 首先申明一个与该函数匹配的函数指针类型；
 - 定义一个该类型的变量；
 - 把一个函数的指针赋给这个变量。
- ◆ 使用

```
int MaxVal = MaxFunPtr(3, 4);
```

使用指针时的注意事项

- ❖ 可以定义指针类型的变量，但使用该指针变量前，**必须初始化**，否则不知道该指针变量指向何处，造成结果混乱，甚至造成访问违例(Exception)。
- ❖ 空指针用NULL表示。指针变量的值为空(NULL)，实际上是0。
- ❖ 指针变量的值存放的是**地址**，可以进行+,-,++,--运算。
- ❖ 指针分为短指针(32位)和长指针(64位)。
- ❖ 指针变量可以作为函数的参数进行传递。Windows API大量使用回调函数，就是将函数作为参数的。
- ❖ 一个指针变量可指向另一个指针变量的地址，即**指针的指针**
- ❖ void指针类型：首先表明是一个指针，但具体**未指明是那一种类型的指针**。

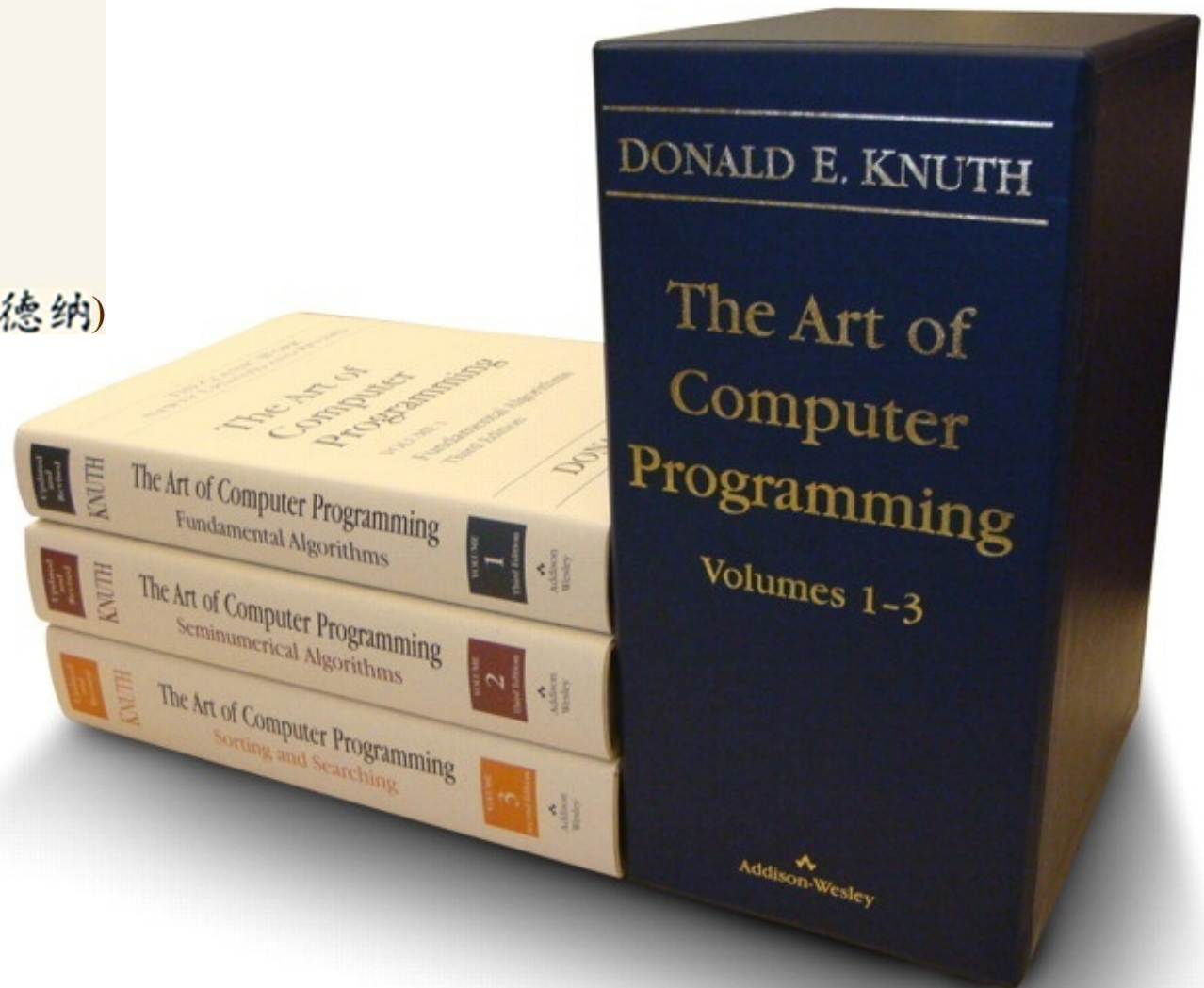
含有指针成员的结构

- ❖ 既然指针是一种数据类型，当然在定义结构时可以使用指针类型的成员，包括函数指针。
 - 在学习C语言时，曾在讲结构和指针之后，讲了如何用指针处理链表 
 - 在C++语言中，是如何实现类的 

第一章 绪论



Donald E. Knuth (高德纳)



1974年，因**克努斯**(Donald E. Knuth)在算法分析和编程语言设计方面的突出贡献，荣获美国计算机协会图灵奖（计算机界的诺贝尔奖），是历史上最年轻的获奖者。《计算机程序设计艺术》一书与牛顿的《自然哲学的数学原理》等书一起，被评为“世界历史上最伟大的十种科学著作”之一。

第一章 绪论

- 1.1 为什么学习数据结构
- 1.2 数据结构的发展简史
- 1.3 数据结构的基本概念
- 1.4 算法的基本概念
- 1.5 算法的正确性证明
- 1.6 算法分析基础

1.1 为什么学习数据结构

1.1.1 学习的必要性

- ❖ 是计算机专业的一门重要专业基础课
- ❖ 是计算机程序设计、编译原理、操作系统、数据库、计算机网络等很多专业课的重要基础

例：

- ❖ **有向无环图**是描述工程或系统进行过程的有效工具。一是用于考查工程能否顺利进行，二是用于估算工程完成所需的最短时间，这就是**拓扑排序和关键路径**问题。
- ❖ 1956年，美国杜邦公司提出**关键路径**法，并于1957年首先用于1000万美元化工厂建设，工期比原计划缩短了4个月。杜邦公司在采用关键路径法的一年中，节省了100万美元。

- ❖ 最短路径算法在物流配送问题中的应用
- ❖ 树结构在数据挖掘中的应用
- ❖ 散列技术在数据加密中的应用
- ❖ 查找技术在数据库中的应用
- ❖ 倒排文件、查找算法在搜索引擎中的应用

倒排文件是大规模信息检索系统常用的一种索引组织技术。在Web检索环境下,搜索引擎索引数据量和用户量都不断增加,从而使得优化倒排文件的组织、改进检索算法,不断提高系统检索效率成为一个被关注的研究课题。

1.1.2 学习目标

- ❖ 深入理解“**数据结构**”和“**算法**”等重要概念
- ❖ 熟练掌握如何有效**组织**、**存储**和**操作数据**
- ❖ 熟练掌握编写计算机程序求解具体问题的基本步骤

1.2 数据结构的发展简史

- ❖ 20世纪40年代：计算机只能进行数值计算，处理简单、小规模的数据，**没有数据结构的概念**；
- ❖ 20世纪50年代末：解决非数值计算问题，处理日益复杂的数据，出现了数组、记录、字符串等新型数据类型（**简单的线性数据结构**）；
- ❖ 20世纪60年代末：数据结构发展成为一门**系统和相对独立的课程**，人们对数据结构有了**全面和深入的认识**；
- ❖ 20世纪70年代后期：随着数据库技术的成功应用，数据结构补充了**文件组织、存储管理**等方面的内容。
- ❖ 20世纪80年代：随着面向对象技术的兴起，数据结构增加了**抽象数据类型**概念，以刻画类、对象等新型数据类型。

数据结构发展中的里程碑

- ❖ 60年代末、70年代初，著名计算机科学家克努斯（D. E. Knuth）陆续出版了包括《基本算法》、《半数值算法》和《排序和查找》等三卷《计算机程序设计技巧》（The Art of Computer Programming），使人们首次对数据结构有了全面、深刻的认识；
- ❖ 1972年，著名计算机科学家霍尔（C. A. R. Hoare）在其论文“数据结构札记”中，澄清了关于数据结构术语和概念等方面的杂乱局面，深刻论述了算法与数据结构密不可分的关系；
- ❖ 1976年，著名计算机科学家沃思（N. Wirth）出版了名为《算法+数据结构=程序》的专著，形象地描述了数据结构、算法与程序之间的关系，旗帜鲜明的提出数据结构和算法对程序设计的重要性。

1.3 数据结构的基本概念

- ❖ 数据
- ❖ 结构
- ❖ 操作
- ❖ 数据结构

1.3.1 数据

❖ 数据的含义

计算机程序处理的对象统称为**数据 (Data)**。

简单数据：整数、实数、字符串

复杂数据：图像、音频、视频、计算机程序、Web

❖ 数据的组成

数据由多个**数据元素**（数据成分、结点、顶点）组成；

数据元素由描述其属性的多个**数据项**（域、字段）组成。

❖ 例子

关系数据库程序处理的数据是一个**二维表格**。

数据类型

- ❖ 数据类型：Data Type — 高级语言中指数据的取值范围及其上可进行的操作的总称。

C语言提供int, char, float, double等基本数据类型；
数组、结构体、共用体、枚举等构造数据类型；
还有指针、空(void)类型等。
用户也可用typedef 定义新的数据类型。



这此类型可以分为原子类型和结构类型。
原子类型是不可分的数据类型。
结构类型是可分为原子类型的数据类型。

那些是原子类型，那些是结构类型？

抽象数据类型

- ❖ 抽象数据类型Abstract Data Type — 是指一个**数学模型**以及定义在该模型上的一组**操作**。抽象数据类型的定义取决于它的一组**逻辑特性**，而与其在计算机内部的表示和实现无关。
- ❖ 一个含抽象数据类型的程序模块通常包含**定义**、**表示**和**实现**三个部分。

抽象数据类型

- 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。

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

个人图书数据

| 个人 | | | | | | | |
|----|--------|--------|----------------------|-----|------|-------|--|
| | 登录号 | 书号 | 书名 | 作者 | 出版社 | 价格 | |
| ▶ | 000001 | TP2233 | Windows NT4.0中文版教程 | 赵健雅 | 电子工业 | 28.00 | |
| | 000002 | TP1844 | Authorware 5.1速成 | 孙 强 | 人民邮电 | 40.00 | |
| | 000003 | TP1684 | Lotus Notes网络办公平台 | 赵丽萍 | 清华大学 | 16.00 | |
| | 000004 | TP2143 | Access 2000入门与提高 | 张 堪 | 清华大学 | 22.00 | |
| | 000005 | TP1110 | PowerBuilder 6.5实用教程 | 樊金生 | 科技大学 | 29.00 | |
| | 000006 | TP1397 | Delphi数据库编程技术 | 刘前进 | 人民邮电 | 43.00 | |
| | 000007 | TP2711 | 精通MS SQL Server 7.0 | 罗会涛 | 电子工业 | 35.00 | |
| | 000008 | TP3239 | Visual C++实用教程 | 郑阿奇 | 电子工业 | 30.00 | |
| | 000009 | TP1787 | 电子商务万事通 | 赵乃真 | 人民邮电 | 26.00 | |
| | 000010 | TP42 | 数据结构 | 江 涛 | 中央电大 | 18.80 | |

数据 = 表格

数据元素 = 记录(表中的一行)

数据项 = 属性(表中的一列) 关键字

1.3.2 结构

- ❖ 数据的逻辑结构
- ❖ 数据的存储结构

数据结构的表示

- ❖ 逻辑结构：用数学模型描述数据元素之间的逻辑关系
- ❖ 物理(存储)结构：在计算机内数据元素的存储映像
 - 顺序存储结构—借助元素在存储器中的相对位置(顺序映像) 来表示数据元素间的逻辑关系
 - 链式存储结构—借助指示元素存储地址的指针(非顺序映像) 来表示数据元素间的逻辑关系
- ❖ 数据存储结构的表示：并非直接用机器内存表示。
 - 高级语言—虚拟机—虚拟存储器
- ❖ 数据的逻辑结构与存储结构密切相关
 - 算法设计 → 逻辑结构，应该与语言无关的
 - 算法实现 → 存储结构，与语言相关

1.3.2 结构 --- 数据的逻辑结构

❖ 什么是数据的逻辑结构？

数据由多个数据元素组成，数据的逻辑结构讨论的是数据元素之间的逻辑关系。

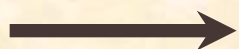
❖ 如何刻画数据元素间的“逻辑关系”？

引入“前驱”与“后继”的概念。

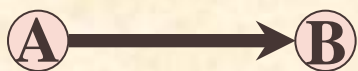
(1) 直观表示（非严格定义）：



用结点(顶点)表示数据元素；



用箭头表示数据元素间的前驱/后继关系；



数据元素A是B的前驱，数据元素B是A的后继。

1.3.2 结构 --- 数据的逻辑结构

(2) 前驱与后继的数学定义

设二元组 $L=(N, r)$ 表示数据的逻辑结构，其中 $N(L)$ 是数据元素(结点)的有限集合， r 是 N 上的一个二元关系。

- 若 $a, b \in N$ ，且关系 $\langle a, b \rangle \in r$ ，则称 a 是 b 的**前趋结点**，称 b 是 a 的**后继结点**，称 a 和 b 是**相邻结点**；
- 如果不存在 $a \in N$ ，使 $\langle a, b \rangle \in r$ ，则称 b 为**始结点**；
- 如果不存在 $b \in N$ ，使 $\langle a, b \rangle \in r$ ，则称 a 为**终结点**；
- 既非始结点又非终结点的结点被称为**内结点**。

1.3.2 结构 --- 数据的逻辑结构

❖ 逻辑结构的分类

根据关系 r ，可将数据的逻辑结构分为：

线性结构（线性表， r 称为线性关系）

非线性结构

层次结构（树结构， r 称为父子关系）

网络结构（图结构， r 称为相邻关系）

集合（ r 为空关系）

线性结构：结构中有且仅有一个始结点和一个终结点，始结点只有一个后继结点，终结点只有一个前趋结点，每个内结点有且仅有一个前趋结点和一个后继结点。

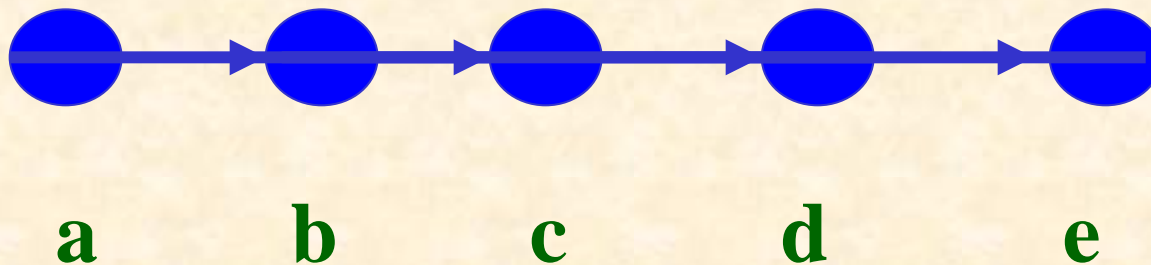
非线性结构：结构中的结点可能有多个前趋结点和多个后继结点。

例1 线性结构

$L=(N,r1),$

$N=\{a, b, c, d, e\}$

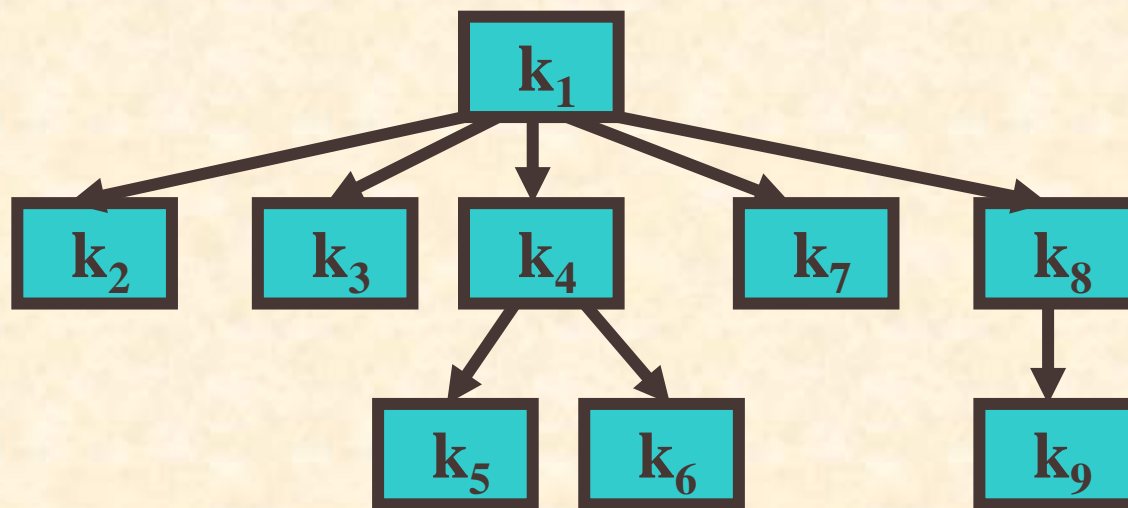
$r1=\{<a,b>, <b,c>, <c,d>, <d,e>\}$ (线性关系)



例2 层次结构(树形结构)

$L=(N,r_2)$, $N=\{k_1, k_2, \dots, k_9\}$

$r_2=\{< k_1,k_2 >, < k_1,k_3 >, < k_1,k_4 >, < k_1,k_7 >, < k_1,k_8 >, < k_4,k_5 >, < k_4,k_6 >, < k_8,k_9 >\}$ (父子关系)

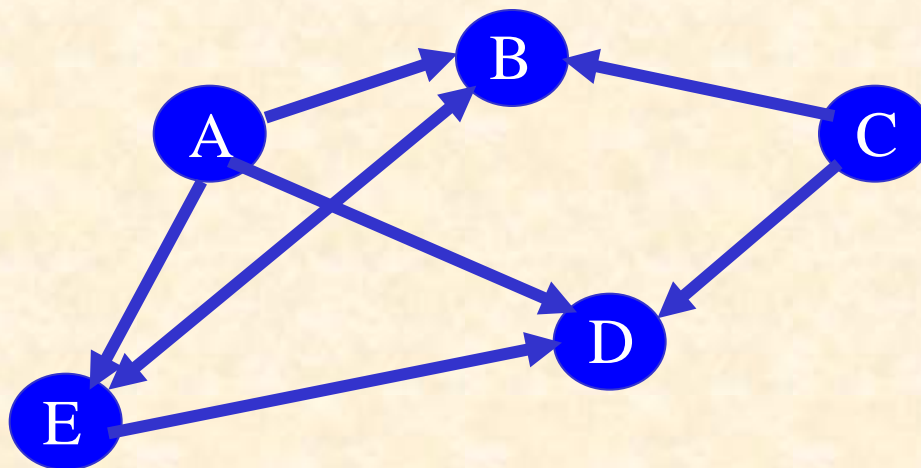


层次结构：根节点没有前驱；除根节点外，每个节点有且仅有一个前驱，但可以有多个后继。

例3 网状结构(图结构)

$L=(N,r3)$, $N=\{A, B, C, D, E\}$

$r3=\{< A,B >, < A,E >, < A,D >, < B,E >, < C,B >, < C,D >, < E,B >, < E,D >\}$ (相邻关系)

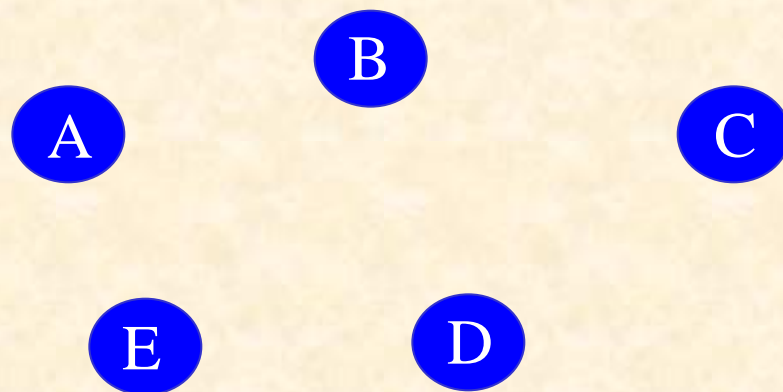


图结构：每个节点可以有任意多个前驱和后继。

例4 集合结构

$L=(N,r_4)$, $N=\{A, B, C, D, E\}$

$r_4=\{\}$ (空关系)



集合： 每个节点都没有前驱和后继。

数据结构的例 - 之1

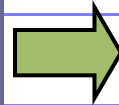
□ 书目自动检索系统

线性表

书目文件

书目卡片

登 录 号:
分 类 号:
书 名:
作 者:
出版单位:
出版时间:



| | | | |
|-------|-------|-------|-------|
| 001 | 高等数学 | 樊映川 | S01 |
| 002 | 理论力学 | 罗远祥 | L01 |
| 003 | 高等数学 | 华罗庚 | S01 |
| 004 | 线性代数 | 栾汝书 | S02 |
| | | | |

按书名

| | |
|-------|---------|
| 高等数学 | 001,003 |
| 理论力学 | 002 |
| 线性代数 | 004 |
| | |

按作者

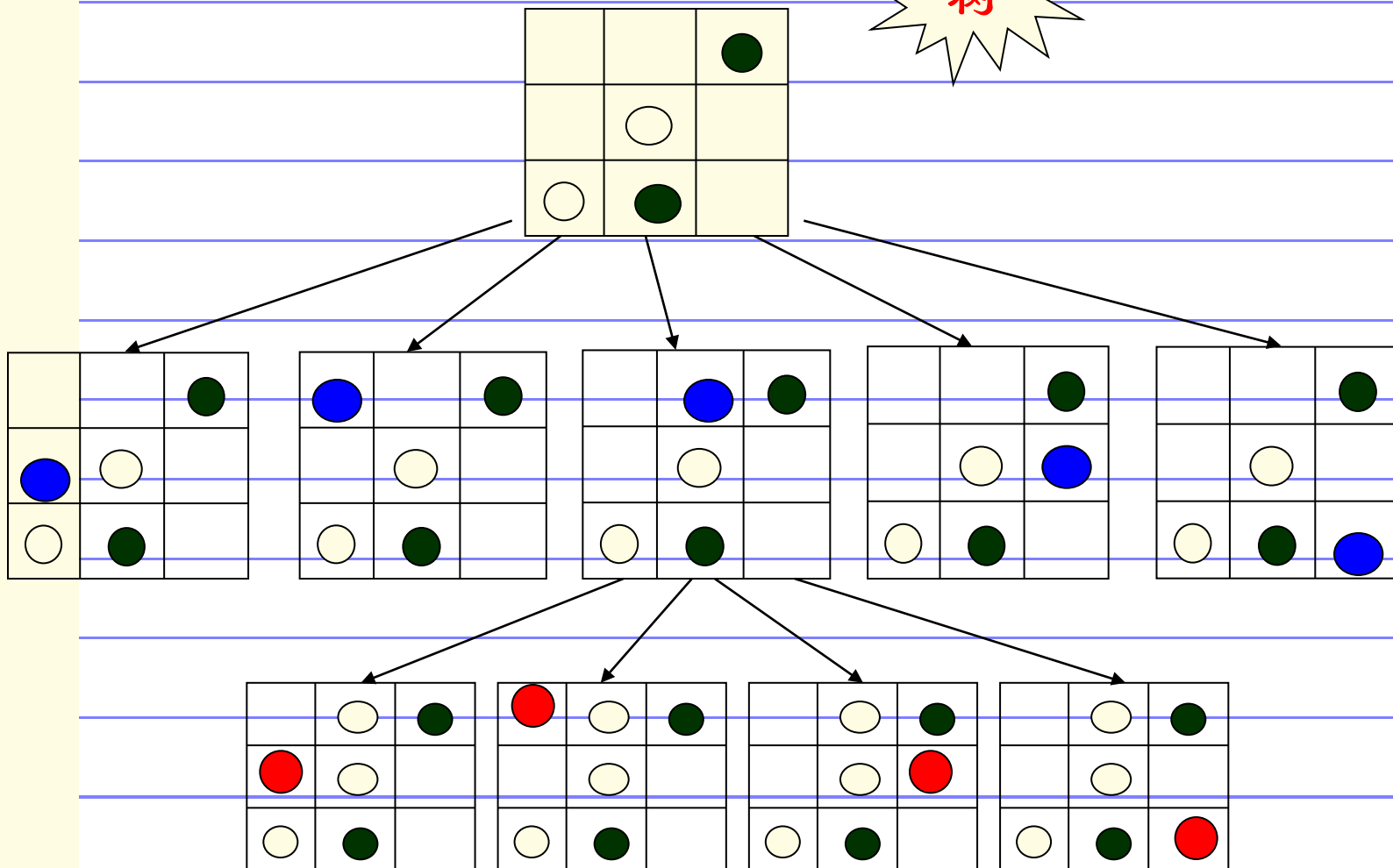
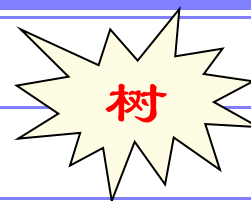
| | |
|-------|-------|
| 樊映川 | 001 |
| 罗远祥 | 002 |
| 华罗庚 | 003 |
| 栾汝书 | 004 |
| | |

按分类号

| | | |
|-----|-------|---------|
| S | S01 | 001,003 |
| | S02 | 004 |
| L | L01 | 002 |
| ... | | |

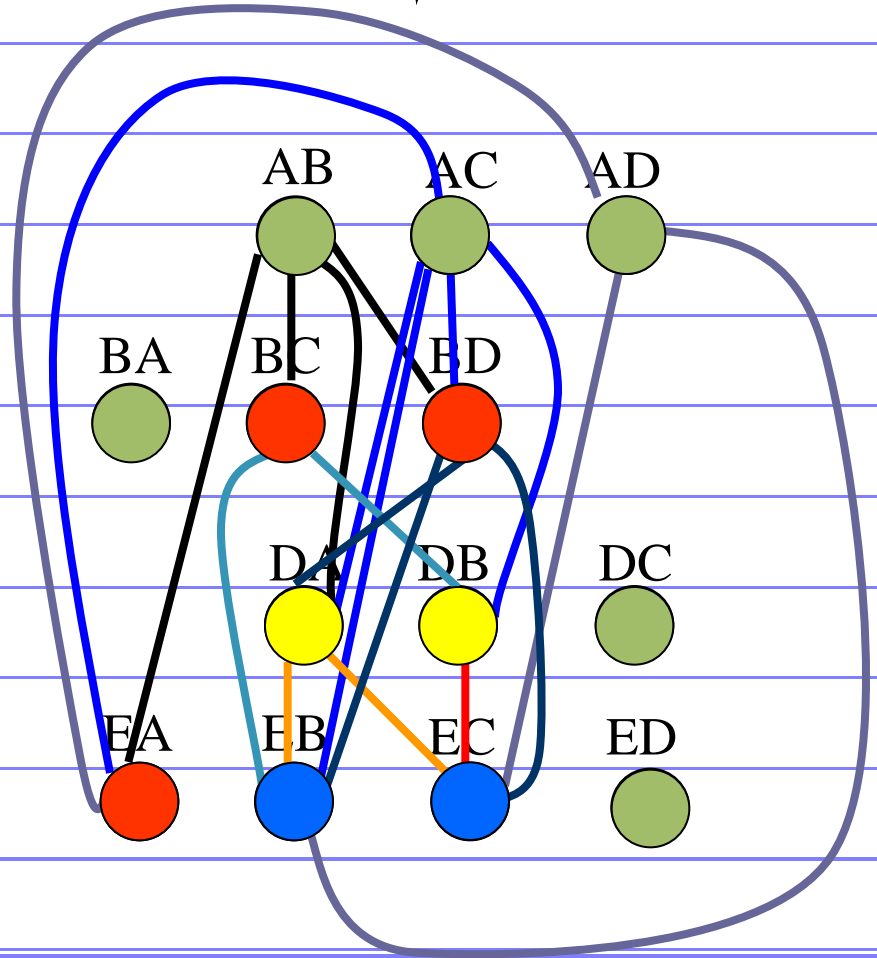
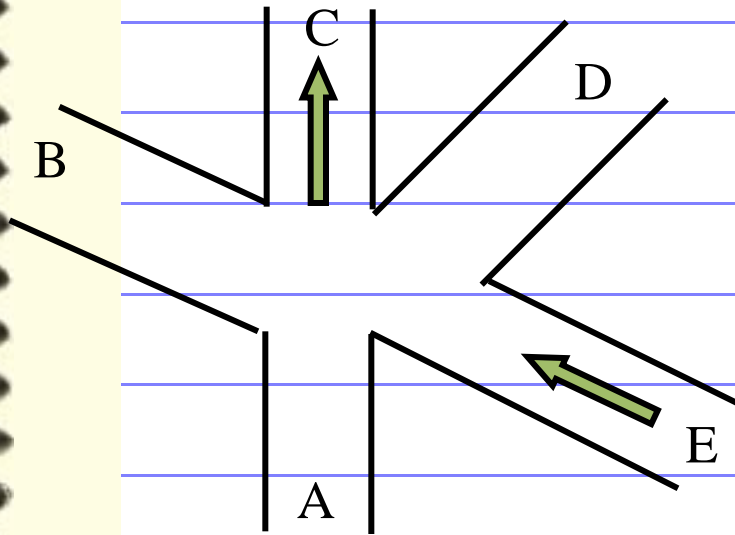
数据结构的例 - 之2

□ 例2 人机对奕问题



数据结构的例 - 之3

□ 多叉路口交通灯管理问题



1.3.2 结构 --- 数据的存储结构

❖ 什么是数据的存储结构

数据及其**逻辑关系**在计算机中的存储表示称为**数据的存储结构**。

设计存储结构需要考虑两个问题：

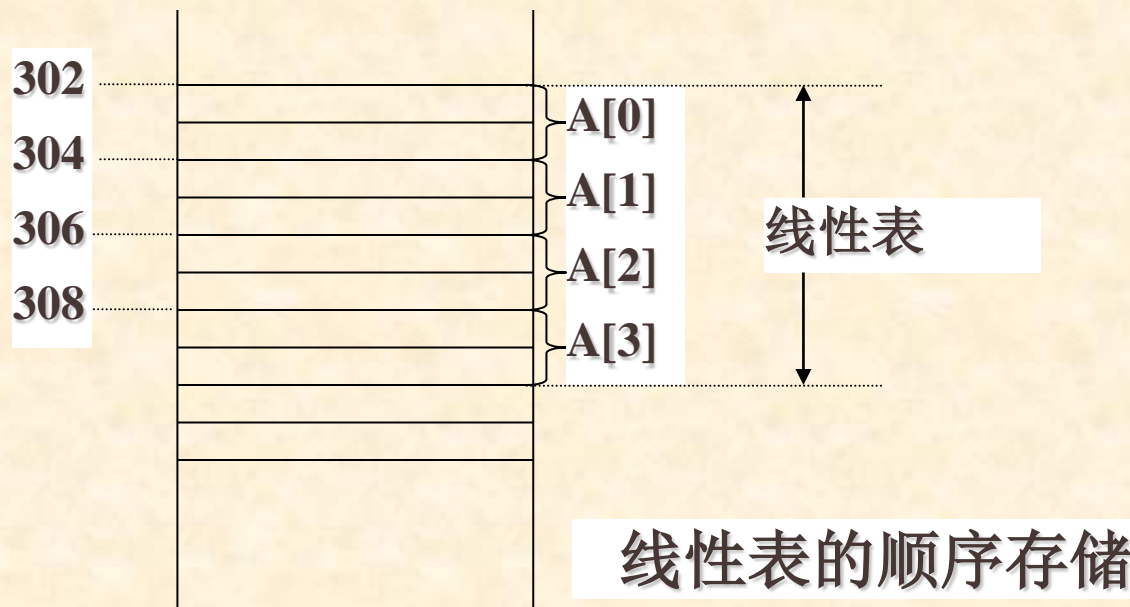
- 如何存储数据元素 N ($N \rightarrow M$ 的映射, M 表示存储空间)
- 如何存储逻辑关系 r ($r \rightarrow M$ 的映射)

❖ 根据不同存储策略的组合，常见的存储结构包括：

- 顺序存储
- 链接存储
- 索引存储
- 散列存储

1. 顺序存储结构

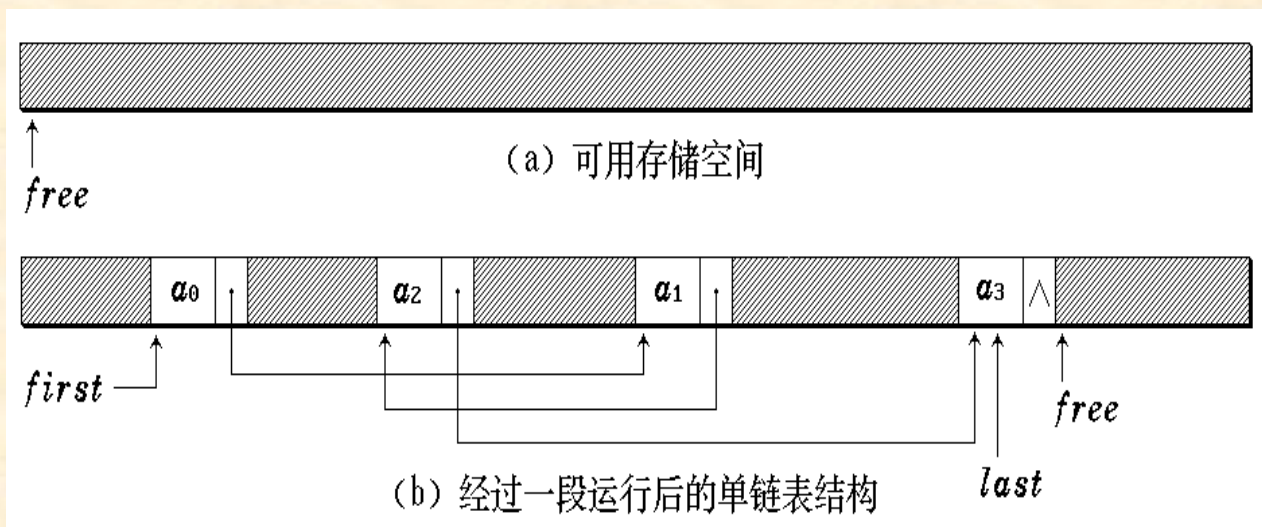
- ❖ **方法:** 数据元素连续地存放在存储空间中，使逻辑上相邻的数据元素存放在物理上相邻的存储单元中。元素的逻辑关系(前驱/后继关系)用物理地址的相邻关系表示。



- ❖ **优点:** (基于下标的) 随机检索高效;
- ❖ **缺点:** 插入、删除等维护操作低效;
空间利用率低, 会出现“存储碎片”的情况。

2. 链接存储结构

- ❖ **方法：**数据元素任意地存放在存储单元中，逻辑上相邻地结点不必物理上相邻。数据元素间的逻辑关系采用**指针**的方式表示。



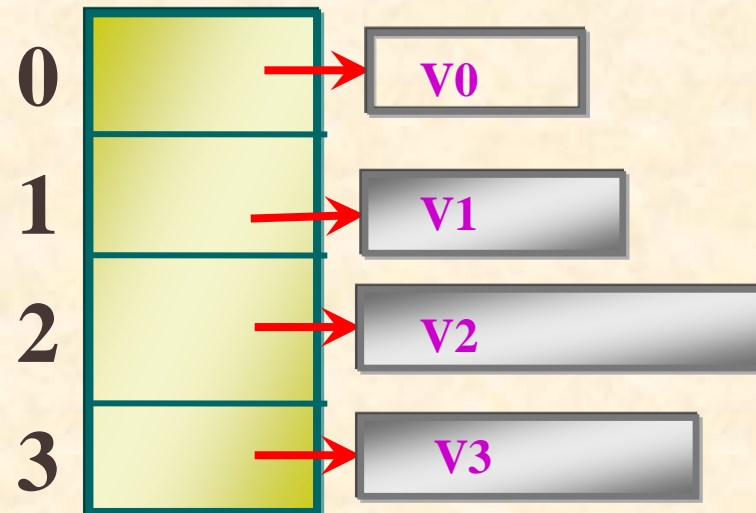
线性表的链接存储

- ❖ **优点：**插入、删除等维护操作高效；
空间利用率高，可有效处理“存储碎片”的情况。
- ❖ **缺点：**（基于下标或关键字）检索操作低效。

3、索引存储结构

方法：数据元素任意地存放在存储单元中，逻辑上相邻地结点不必物理上相邻。采用**顺序存储**的方法存储一个**索引表**，表中的每*i*项存放指向存储区域中第*i*个数据元素的指针。

优点：是顺序存储和链接存储的结合体，具有两者的优点，对**基于下标**的随机检索、插入、删除等操作均相对高效。



4、散列存储结构

方法：数据元素任意地存放在存储单元中，逻辑上相邻地结点不必物理上相邻。采用“**散列表**”方法存储一个索引表，表中的每项存放相应数据元素的首地址。

优点：对**基于关键字**的随机检索、插入、删除等操作均相对高效；

5、组合存储结构

将以上4种基本存储结构灵活地组合使用，实现复杂逻辑结构的存储。
如图结构的可用顺序结构+链接结构实现。

要根据问题的特点有针对性的选择合适的存储结构。

1.3.3 操作

❖ 对数据的基本操作

查找：在数据中查找具有指定关键字值的数据元素(记录)；

插入：向数据中插入新的数据元素；

删除：从数据中删除指定的数据元素；

修改：修改指定数据元素某些数据项的值；

排序：对数据中所有数据元素按关键字值升序或降序排列；

遍历：以某种方式访问数据中所有数据元素。

❖ 例子

数据：电话簿

操作：增加联系人信息(**插入**)、更改某人的办公电话(**查找+修改**)、按姓氏笔画将联系人**排序**、打印所有人的电话信息(**遍历**)。

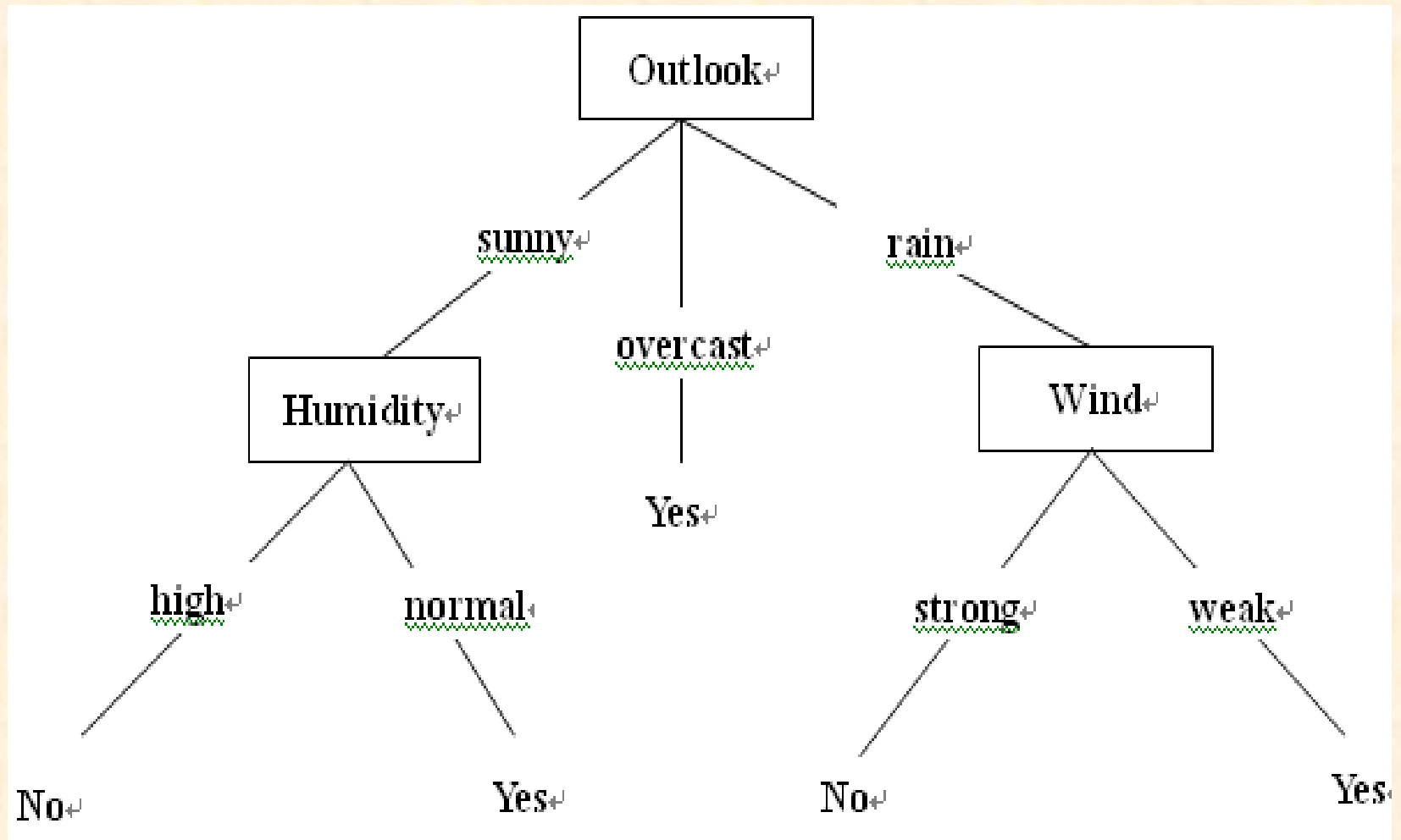
1.3.3 操作

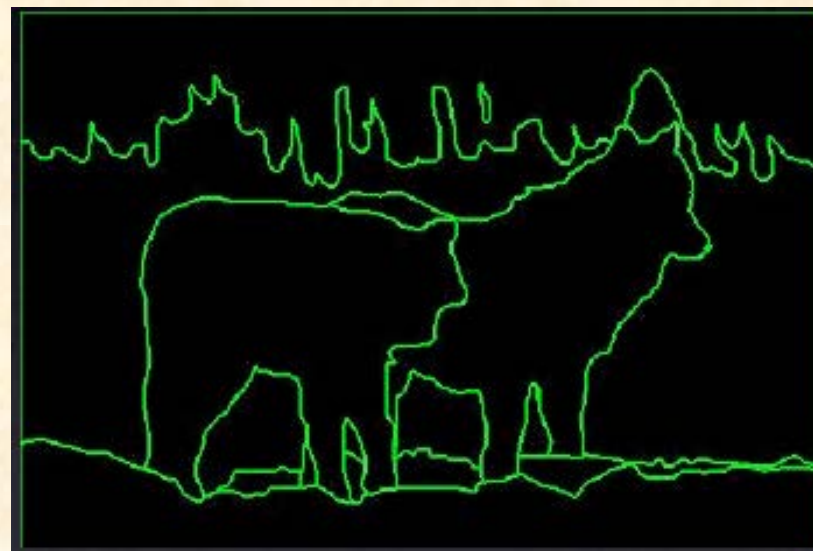
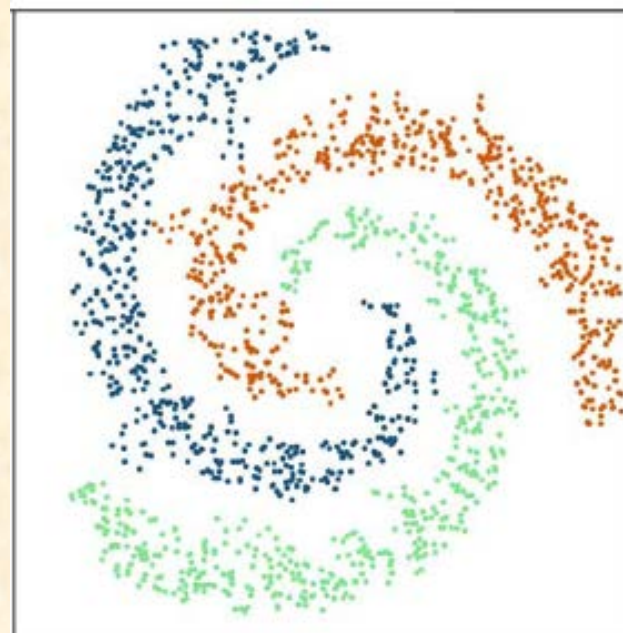
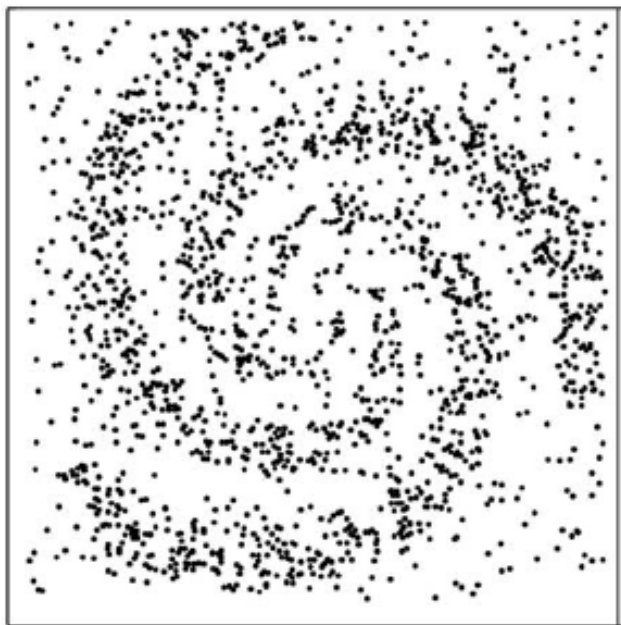
❖ 高级数据操作

➤ 发现数据背后隐藏什么规律和模式

| Day | Outlook | Temperature | Humidity | Wind | PlayTennis |
|-----|----------|-------------|----------|--------|------------|
| 1 | sunny | hot | high | weak | no |
| 2 | sunny | hot | high | strong | no |
| 3 | overcast | hot | high | weak | yes |
| 4 | rain | mild | high | weak | yes |
| 5 | rain | cool | normal | weak | yes |
| 6 | rain | cool | normal | strong | no |
| 7 | overcast | cool | normal | strong | yes |
| 8 | sunny | mild | high | weak | no |
| 9 | sunny | cool | normal | weak | yes |
| 10 | rain | mild | normal | weak | yes |
| 11 | sunny | mild | normal | strong | yes |
| 12 | overcast | mild | high | strong | yes |
| 13 | overcast | hot | normal | weak | yes |
| 14 | rain | mild | high | strong | no |

发现如下决策规则：

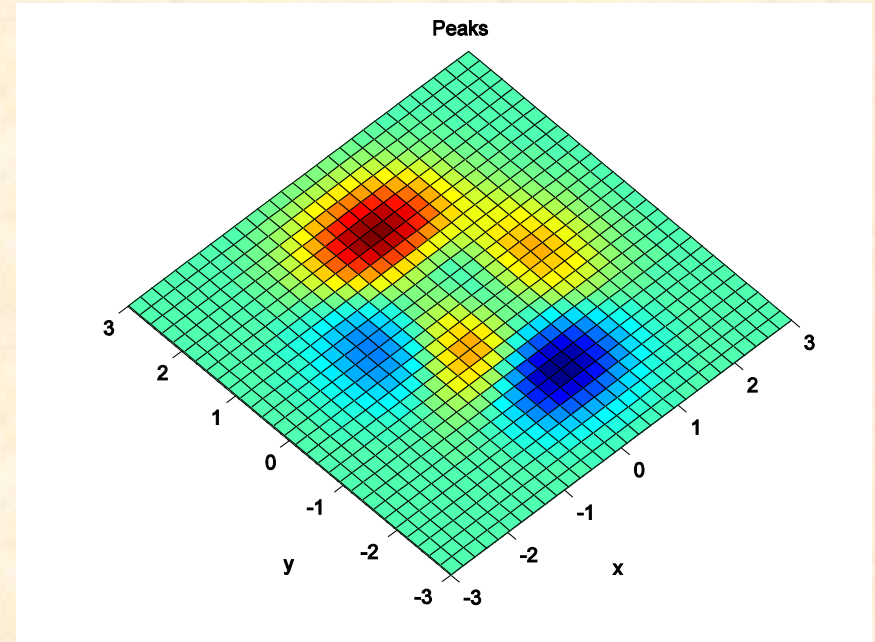
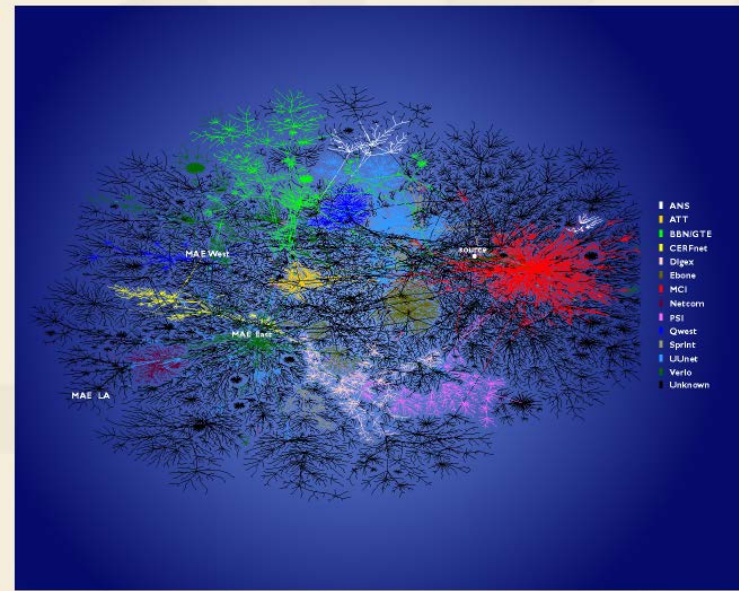




数据是如何产生的？

Complex Network Example: WWW

(K. C. Claffy)



www.sciencemag.org SCIENCE VOL 286 15 OCTOBER 1999

Emergence of Scaling in Random Networks

Albert-László Barabási* and Réka Albert

Systems as diverse as genetic networks or the World Wide Web are best described as networks with complex topology. A common property of many large networks is that the vertex connectivities follow a scale-free power-law distribution. This feature was found to be a consequence of two generic mechanisms: (i) networks expand continuously by the addition of new vertices, and (ii) new vertices attach preferentially to sites that are already well connected. A model based on these two scale-free distributions, which is governed by robust self-organization of the individual systems.

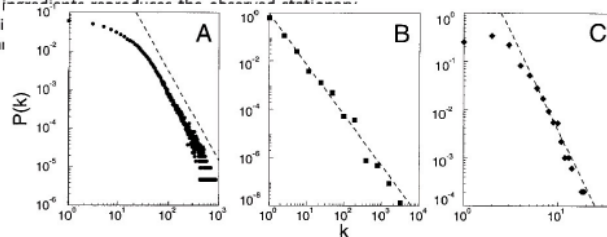
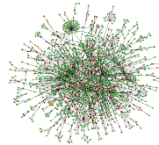
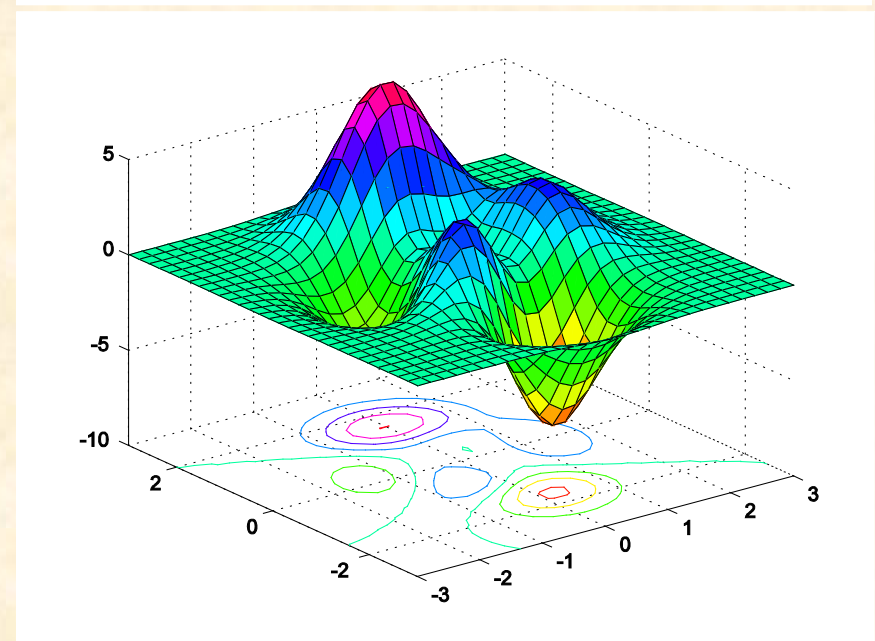


Fig. 1. The distribution function of connectivities for various large networks. (A) Actor collaboration graph with $N = 212,250$ vertices and average connectivity $\langle k \rangle = 28.78$. (B) WWW, $N = 325,729$, $\langle k \rangle = 5.46$ (6). (C) Power grid data, $N = 4941$, $\langle k \rangle = 2.67$. The dashed lines have slopes (A) $\gamma_{\text{actor}} = 2.3$, (B) $\gamma_{\text{www}} = 2.1$ and (C) $\gamma_{\text{power}} = 4$.

Power law

$$P(k) \sim k^{-\gamma}$$



1.3.4 数据结构

❖ 数据结构概念

按某种逻辑关系将一批数据元素组织起来，按一定的存储方式将它们存储起来，并在这些数据元素上定义一个**操作集合**，就得到了一个特定的数据结构。

❖ 数据结构的基本组成部分

- 数据的逻辑结构
- 数据的存储结构
- 施加于数据上的操作

因此可以说，《数据结构》就是研究如何**组织**、**存储**和**操作数据**的一门课程。

1.4 算法的基本概念

- ❖ 何谓算法？
- ❖ 算法与数据结构的关系
- ❖ 算法的特性
- ❖ 算法的评价准则
- ❖ 算法描述语言

1.4.1 算法

算法（Algorithm）概念最早出现在1747年出版的一本德文《大全数学辞典》中.《Webster词典》指出：算法系指在有限步骤内解一个数学问题的过程，步骤中常包括某一操作的重复。

计算机算法：欲完成解题任务，除选取适当的数据结构外，还要给出解决问题的切实可行的方法和步骤。

在计算机科学中，算法泛指解决问题时所采用的方法或步骤。具体的，算法由有限条指令构成，规定了解决特定问题的一系列操作。

1967年，克努斯指出：“**计算机科学是有关算法的学问**”，并指出“对所有的计算机程序设计来说，算法的概念总是最基本的”。数据结构与算法紧密相关。算法无疑也是本书的重要内容。

1.4.2 算法与数据结构的关系

算法与数据结构密切相关。

一方面算法设计依赖具体的数据结构；

另一方面，数据结构直接影响算法的执行效率。

1976年，沃斯提出：算法 + 数据结构 = 程序

1.4.3. 算法的特性

- (1) 有限性：一条指令都只能执行有限次，整个算法在有限时间内结束
- (2) 确定性：算法中的每条指令都必须清楚，没有二义性
- (3) 输入：具有若干由外界提供的量
- (4) 输出：产生1个或多个结果
- (5) 可行性：每条指令都十分基本，原则上可由人仅用笔和纸在有限的时间内也能完成

注意：算法和计算过程是有区别的，后者不需要满足有限性。

1.4.4 算法的评价准则

- 正确性
- 时间复杂性
- 空间复杂性
- 可读性
- 坚固性(健壮性)

还应该具有灵活性、可重用性和自适应性。

1. 正确性

“正确性”的含义可分为四个层次：

- ① 算法不含语法错误；
- ② 算法对于几组输入数据能够得出满足规格说明要求的结果；
- ③ 算法对于精心选择的典型、苛刻而带有刁难性的几组数据能够得出满足规格说明要求的结果；
- ④ 算法对一切合法的输入数据都能产生满足规格说明要求的结果。

✧ 实际运行时间

✧ 时间复杂性

2. 时间复杂性

度量算法时间效率的标准：

- (1) 能反应算法所采用的方法的时间效率；
- (2) 与算法描述语言及设计风格无关；
- (3) 与算法的许多细节无关；
- (4) 足够精确和具有一般性。

基本运算（关键操作）

算法中起主要作用且费时最多的操作。

时间复杂性

一个算法的时间复杂性是指该算法的基本运算次数。

3. 空间复杂性

算法在运行过程中所占用的存储空间的大小被定义为算法的**空间复杂性**。主要包括：

- 算法本身所占用的存储空间（指令空间）
- 算法的输入、输出数据所占用的存储空间
- 算法运行过程中占用的变量空间和环境栈空间

4. 可读性

可读性好的算法使得证明或测试其正确性比较容易，同时便于阅读、实现、调试和维护。

添加注释有利于提高程序的可读性。

5. 坚固性

对非法输入具有识别和处理能力。

6. 在算法**正确**的前提下**最优算法**定义如下：

某算法A为最优，当且仅当解决同一问题的所有算法集合SA ($A \in SA$)中没有一个算法执行的基本运算次数比算法A更少。

证明A是最优的常用策略：

A的时间复杂性 = 该问题所需时间复杂性的
下确界

1.4.5 算法描述语言

算法可以用自然语言、数学语言或者约定的符号语言来描述。

课堂上主要采用C语言描述算法。

算法描述语言 —— ADL

① ADL 的格式

算法<算法名> (变量 i_1, \dots , 变量 i_m . 变量 j_1, \dots , 变量 j_n)

//单行注释 (或/*...*/多行注释)

步骤名 [步骤1所执行操作的高度概括]

语句序列.

[步骤n所执行操作的高度概括]
语句序列. ■

<算法名>是由字母和数字组成的有限字符串，且串中第一个符号必须是字母。

在变量表中，变量 i_k 为输入变量， $1 \leq k \leq m, m \geq 0$ ；变量 j_k 为输出变量， $1 \leq k \leq n, n \geq 1$ ，输入和输出变量用“.”隔开

算法描述语言

算法的每一步骤都要有名称, 名称由<算法名或算法名缩写><数字><.>组成.

① ADL 的格式

算法<算法名> (变量 i_1, \dots , 变量 i_m . 变量 j_1, \dots , 变量 j_n)

//单行注释 (或/*...*/多行注释)

步骤名1 [<本步骤的概括说明>]

语言序列

步骤号后紧接一方括号, 该方括号内是对该步骤所执行操作的高度概括.

[所执行操作的高度概括]

② 表达式

● 算术运算符 $+$ 、 $-$ 、 $*$ 、 $/$ 、**DIV**、**MOD**、
取商运算, 取余数（模）运算

● 关系运算符 $=$ 、 \neq 、 $<$ 、 $>$ 、 \geq 、 \leq

● 逻辑运算符 **AND**、**OR**、**NOT**

逻辑常量 **true**、**false**

● 集合运算符 \cap 、 \cup 、 $-$ （差）、 \subset 、 \supset

③ 语句

● 每条语句都用 “.” 作为结束符

● 赋值语句 $a \leftarrow b$. $a \leftrightarrow b$.

$a \leftarrow b \leftarrow c$.

● 条件语句

IF <逻辑表达式> **THEN**
 (语句1. 语句m) .

IF <逻辑表达式> **THEN**
 (语句1. 语句m) .
ELSE
 (语句1. 语句n) .

CASE DO
 (<逻辑表达式1>: (语句1. 语句n₁)) .
 ...
 <逻辑表达式m>: (语句1. 语句n_m)).

● 循环语句

WHILE <逻辑表达式> **DO**
(语句1. 语句n) .

FOR<变量> = <算术表达式1> **TO**
<算术表达式2> **STEP** <算术表达式3> **DO**
(语句1. 语句n) .

FOR \forall <变量> \in <集合> **DO**
(语句1. 语句n) .

- **转移语句**

GOTO (<步骤号>).

- **EXIT 语句** 可以提前结束**WHILE**或者**FOR**循环的执行

- **RETURN 语句** 指出算法执行的终点

④其它

- 每个算法都用 “**┆**” 表示算法书写完毕
- 需要注释的语句由 “//” 或者 “/*” 引导
- 输入或输出语句使用 **READ** 和 **PRINT**

ADL 的特点

❖ 书写简便

- 不必考虑过多程序设计语言的细节
- 例: 交换 a 和 b 的值

❖ 易于理解

- 便于不同编程习惯的人交流
- 不必考虑与算法无关的内容(运行环境、编译状态等)

| ADL | C++ |
|-----------------------|---|
| $a \leftrightarrow b$ | <pre>int temp; int a; int b; temp = a; a = b; b = temp;</pre> |

1.5 算法的正确性证明

正确性验证方法：证明 和 测试

- 采用数学或逻辑的方法，严格地证明算法或其关键步骤的正确性；
- 采用软件工程的测试方法，尽可能的保证算法对大部分输入都能产生正确的输出。

Testing (测试)

❖ Black Box Methods (黑盒法)

- 侧重测试程序的功能，不考虑程序是如何实现的，即代码结构
- 设计测试用例，检查是否能得到预想的结果

❖ White Box Methods (白盒法)

- 侧重测试程序的代码结构
- **Statement Coverage** 语句覆盖: 使程序中的每一条语句都至少执行一次。
- **Decision Coverage** 分支覆盖: 使程序中的每一个分支都至少执行一次。

算法正确性证明

算法的正确性是非常重要的。

- 对明显是正确的算法，可通过上机调试验证其正确性。调试用的数据要精心挑选，以保证算法尽可能对所有的数据都是正确的；
- 只要找出一组数据使算法失败，就能否定算法的正确性；
- 算法的正确性一般通过**数学方法**进行推理证明，常用的数学方法除了**直接证明法**外，还有**反证法和数学归纳法**。

正确性证明的常用方法

❖ 反证法

- ①要证明定理 T 是正确的，首先假定 T 是错误的
- ②使用正确的命题和正确的推理规则进行推理，若出现下列条件之一，就会得出矛盾的结果：
 - 与已知条件矛盾
 - 与公理矛盾
 - 与证明过的定理矛盾
 - 自相矛盾
- ③由此推出定理 T 是正确的。

例 证明没有最大的整数

证明：用反证法。

- ①反面假设：假设存在一个最大整数，记为 N 。
- ②由假设推出矛盾：设 $P = N + 1$ ，因为 P 是两个整数的和，所以 P 也是整数，且 $P > N$ 。与 N 是最大整数矛盾。
- ③肯定原来的结论：故没有最大的整数。证毕 ■

正确性证明的常用方法

❖ 数学归纳法

往证定理 T ， n 是 T 中的正参数，执行如下步骤：

- (1) 归纳前提：往证 $n = c$ 时 T 成立；
- (2) 归纳假设：假设 $n < k$ 时 T 成立；
- (3) 归纳推理：在归纳假设的基础上，往证 $n = k$ 时
 T 成立

数学归纳法

设 T 是一个定理，自然数 n 是 T 中参数. 数学归纳法表明，如果下面两个条件为真，则对于参数 n 的任何值， T 都是正确的：

- (1) **基础归纳**： $n = c$ 时， T 成立。
- (2) **归纳步骤**：如果 $n = k - 1$ 时 T 成立，则 $n = k$ 时 T 也成立。

其中， c 是一个较小的常量， $n \geq c$. 通常，证明初始归纳很容易，而证明归纳步骤很难。

例：证明由递归关系式 $T(n) = T(n - 1) + 1$ ， $T(1) = 0$ ，可推出 $T(n) = n - 1$ ，其中， $n > 1$ 。

证明：

基础归纳： $n = 2$ 时， $T(2) = T(1) + 1 = 1$ ，结论成立。

归纳步骤：

假设 $T(n - 1) = n - 2$ 成立（归纳假设）

往证 $T(n) = n - 1$ 成立。

由递归关系式可知：

$$n > 1 \text{ 时，有 } T(n) = T(n - 1) + 1$$

再由归纳假设：

$$T(n) = T(n - 1) + 1 = n - 2 + 1 = n - 1$$

由数学归纳法推出 $T(n) = n - 1$ 成立，证毕 ■

1.6 算法分析基础

- 关键运算与时间复杂性
- 时间复杂性的渐进表示方法
- 算法的时空复杂性分析

1.6.1 关键运算与时间复杂性

➤ 关键运算

算法中起主要作用且费时最多的操作。

例：矩阵运算中的“*”，“+”，

排序算法中的“比较”

顺序存储线性表插入/删除操作的“移动”

➤ 时间复杂性

算法中关键运算的次数。

例 1 FOR i=1 TO n DO

$x \leftarrow x+1.$

$T(n)=n$

例 2 FOR i=1 TO n DO

FOR j=1 TO n DO

$x \leftarrow x+1.$

$T(n)=n^2$

例3 A是一个含有n个不同元素的实数数组，给出求A之最大和最小元素的算法。

算法SM(A,n . max,min)

SM1[初始化]

max←min←A[1].

SM2[比较]

FOR I=2 TO n DO //求最大和最小元素

(IF A[I]> max THEN max←A[I].

IF A[I]< min THEN min←A[I]). ■

$T(n) = 2(n-1)$

➤ 算法的平均、最好和最坏时间复杂

算法时间复杂性的影响因素：

输入规模、输入性质、输入出现的频率

定义1.5 设一个领域问题的输入的规模为 n ， D_n 是该领域问题的所有输入的集合，任一输入 $I \in D_n$ ， $P(I)$ 是 I 出现的频率， $\sum P(I)=1$ ， $T(I)$ 是算法在输入 I 下所执行的基本运算次数。该算法的期望复杂性定义为：

$$E(n) = \sum \{P(I) * T(I)\}$$

该算法的最好时间复杂性为：

$$B(n) = \min \{T(I)\}$$

该算法的最坏时间复杂性为：

$$W(n) = \max \{T(I)\}$$

例 1.8 实数数组R由n个元素组成，给定一个实数K，试确定K是否是R的元素。

算法F(R, n, K. i)

F1 [初始化]

$i \leftarrow 1.$

F2 [比较]

WHILE $i \leq n$ DO

(IF $R[i]=K$ THEN RETURN.

$i \leftarrow i+1$). ■

n > 1时

最少比较次数： 1: 成功

最大比较次数： n: 成功，失败（最坏复杂性）

设 q ($0 \leq q \leq 1$) 表示 K 在 R 出现中的概率，

| R[1] | R[2] | R[3] | R[4] | R[5] | R[6] | R[7] | R[8] |
|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| 5 | 20 | 12 | 7 | 30 | 40 | 25 | 16 |

q

$K=R[i]$ q/n

$K \neq R[i]$ $1-q$

算法F的期望复杂性为

$$\begin{aligned}
 E(n) &= \sum \{P(I) * T(I)\} \\
 &= \underbrace{(q/n)*1 + (q/n)*2 + \dots + (q/n)*n}_{n \text{项}} + \underbrace{(1-q)*n}_{1 \text{项}} \\
 &= 1/2(n+1)q + (1-q)n
 \end{aligned}$$

如果已知K在R中，即 $q=1$ ，则有

$$E(n) = 1/2(n+1)$$

1.6.2 时间复杂性的渐进表示方法

讨论算法的运行时间随输入规模的变化趋势，即：
随着输入规模的变大，算法的运行时间是如何增长的。

➤ O 表示法

■ O 表示法

设 $f(n)$ 和 $g(n)$ 是正整数集到正实数集上的函数。

称 $g(n)$ 的阶至多为 $f(n)$ ，当且仅当存在一个正常数 C 和 n_0 ，使得对任意的 $n \geq n_0$ ，有 $g(n) \leq C f(n)$ 记为：

$$g(n) = O(f(n))$$

- ❖ n 是算法输入的规模，如数组的维数，图的边数等；
- ❖ 一个算法时间复杂性是 $O(f(n))$ ，称其时间复杂性的阶为 $f(n)$ ；
- ❖ 大写 O 符号定义了函数 g 的一个上限，算法的运行时间至多是 $f(n)$ 的一个常数倍，即：

$g(n)$ 的增长速度至多与 $f(n)$ 的增长速度一样快。

例 1 $T(n)=3n-2$, $f(n)=n$

存在 $C = 3, n_0 = 1$, 当 $n \geq n_0$ 时有

$$3n - 2 \leq 3n$$

于是有: $T(n)=O(n)$

例2 $T(n)=(n+1)^2$, $f(n) = n^2$

存在 $n_0=1, C=4$, 当 $n \geq n_0$ 时有:

$$(n+1)^2 \leq (2n)^2 = 4n^2 ,$$

于是有: $T(n)=O(n^2)$

定理1.1

若 $A(n)=a_m n^m + \dots + a_1 n + a_0$ 是关于 n 的 m 次多项式，
则有：

$$A(n) = O(n^m)$$

| | | | | | | | |
|--------|--------|----------|---------|----------|----------|----------|---------|
| $O(1)$ | $O(n)$ | $O(n^2)$ | \dots | $O(n^3)$ | $O(n^m)$ | $O(2^n)$ | \dots |
| 常数阶 | 线性阶 | | | | 多项式阶 | 指数阶 | |

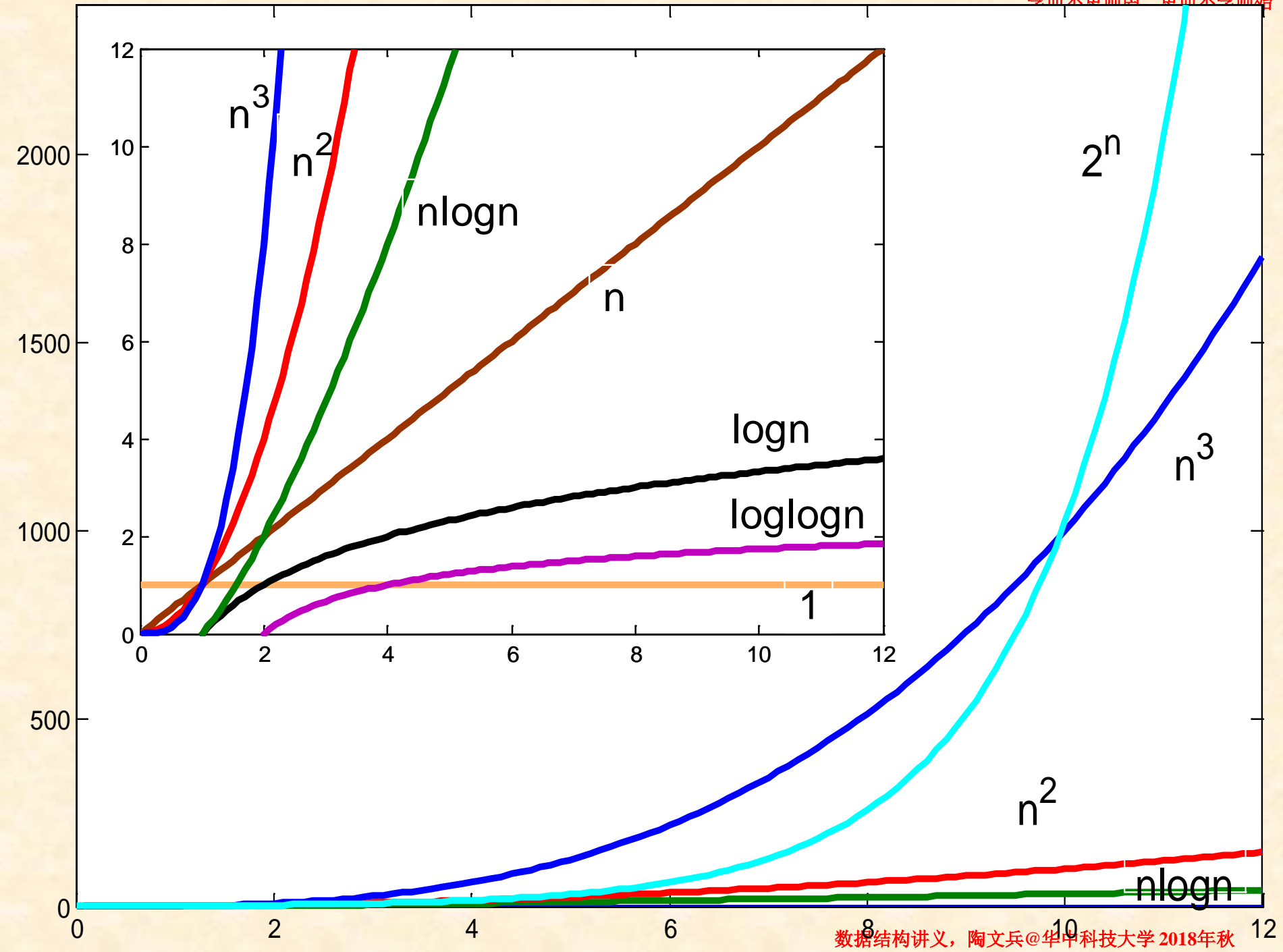
当 n 很大时，有如下关系成立

$$1 < \log \log n < \log n < n < n \log n < n^2 < n^3 < 2^n$$

因此，有

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

算法分成两类：多项式阶算法和指数阶算法。



1.6.3 算法的时空复杂性分析

- 评价一个算法的优劣，综合考虑时间和空间两个因素
- 采用时空积分比较算法优劣，时空积分较小的算法较优。
- 算法在不同的执行时间内，它占用的内存空间量不一定相等。
- 记占用空间量 y 是时间 x 的函数， $y=f(x)$ ，则称积分

$$\int_0^t f(x)dx$$

为该算法的时空积分，其中 t 是该算法的执行时间。

例如，一个算法执行时间为30秒，前10秒算法占用60个字节，第二个10秒算法占用70个字节，最后10秒算法占用80个字节。该算法的时空分布如图2.2所示。



图2.2 时空分布示意图

算法的时空积分为

$$60*10+70*10+80*10=2100 \text{ (字节秒)}$$

小结

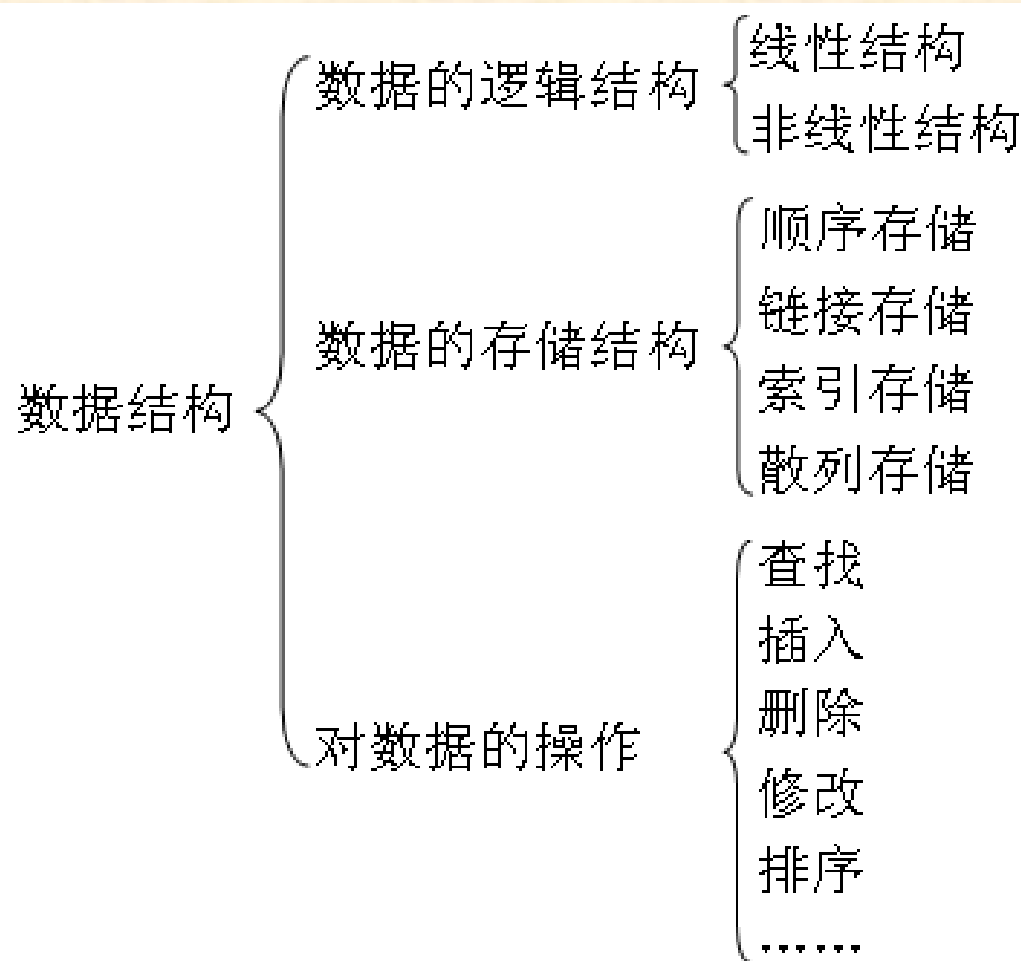


图 1.4 数据结构包含的三方面内容

小 结

- 逻辑结构不同产生不同的数据结构。
- 即使逻辑结构相同，存储结构不同也会产生不同的数据结构。
- 使其逻辑结构和存储结构相同，对数据的操作及其性质不同，也对应着不同的数据结构。