数据结构：概论

Data Structure

主讲教师：杨晓波

E-mail:　248133074@qq.com

第1章 数据结构

- 1.1 数据结构的原则
  - 课程目的
  - 什么是数据结构
  - 数据结构关注什么程序设计目标
  - 为什么学习数据结构
  - 如何衡量数据结构的效率
  - 常见数据关系的类型
- 1.2 抽象数据类型和数据结构
- 1.3 问题、算法和程序

数据结构的课程目的

- 大多数计算机程序的主要目标是**存储信息和尽快地检索信息**。因此，研究数据结构和算法就成了计算机科学的核心问题。
- 课程目的：怎样组织信息，以便支持高效的数据处理。
  - 介绍常用的数据结构
  - 引入并加强“权衡”(tradeoff)的概念，每一个数据结构都有其相关的代价和效益的权衡
  - 评估一个数据结构或算法的有效性。

数据结构的定义

- **数据结构:**
- （教材：）一类数据的表示及其相关的操作。
  - 我的理解和思考
    - 数据结构体现了“物以类聚”的思想
    - 通过寻找数据的共性，进行抽象和封装，实现软件的可重用，提高了编程效率（类似于建筑中的预制件，不用从头修房子）

- 按照逻辑关系组织起来的一批数据，按一定的存储方法把它存储在计算机中，并在这些数据上定义了一个运算的集合。
  - 逻辑数据结构（ADT）
  - 物理数据结构

数据结构所关注的程序设计目标

程序设计目标

1．设计一种容易理解、编码和调试的算法

2．设计一种能有效利用计算机资源的算法

目标1主要涉及到的是软件工程原理；

本课程主要讲的是与目标2有关的问题

学习数据结构的必要性

- **有利于软件重用和提高代码质量**
  - **实际应用中许多数据有共同的属性和操作，通过学习，做到举一反三，恰当地选择数据结构并重用相关代码**
  - **封装好的数据结构往往经过多次实测，已验明正确性，可通过使用它们提高代码质量**
- **数据结构中往往封装了一组数据项的组织或者结构和所有必需的运算（如：查找、打印或排序，或者更改数据项的值）。选择不同的数据结构可能会产生很大的差异。**
  - 同样一个程序，选择一种数据结构可能运行几秒就完成，选择另一种数据结构时可能要运行几天，如：数据库表设计时是否设计主键对数据记录多的表的查询效率影响极大。

效率度量

- 算法分析：
  - 估算一种算法或者一个计算机程序的效率的方法
  - 算法分析可以度量一个问题的内在复杂程度

资源限制

- 资源限制包括空间和时间。
- 算法的代价(cost)是指这种算法消耗的资源量。
- 选择数据结构的步骤：
  - 分析问题，以确定任何算法均会遇到的资源限制。
  - 确定必须支持的基本操作，并度量每种操作所受的资源限制。
  - 选择最接近这些开销的数据结构。

〖Example最大子序列和〗 Given (possibly negative) integers $A_1, A_2, \ldots, A_N$, find the maximum value of $\sum_{k=i}^{j} A_k$.

Max sum is 0 if all the integers are negative.

Algorithm 1

```
int  MaxSubsequenceSum ( const int A[ ],  int  N )
{
          int  ThisSum,  MaxSum,  i,  j,  k;
/* 1*/    MaxSum = 0;   /* initialize the maximum sum */
/* 2*/    for( i = 0; i < N; i++ )  /* start from A[ i ] */
/* 3*/        for( j = i; j < N; j++ ) {   /* end at A[ j ] */
/* 4*/            ThisSum = 0;
/* 5*/            for( k = i; k <= j; k++ )
/* 6*/                ThisSum += A[ k ];  /* sum from A[ i ] to A[ j ] */
/* 7*/            if ( ThisSum > MaxSum )
/* 8*/                MaxSum = ThisSum;  /* update max sum */
            }  /* end for-j and for-i */
/* 9*/    return  MaxSum;
}
```

$$T(N) = O(N^3)$$

Algorithm 2

```
int  MaxSubsequenceSum ( const int A[ ],  int  N )
{
          int  ThisSum,  MaxSum,  i,  j;
/* 1*/    MaxSum = 0;   /* initialize the maximum sum */
/* 2*/    for( i = 0; i < N; i++ )  {   /* start from A[ i ] */
/* 3*/         ThisSum = 0;
/* 4*/         for( j = i; j < N; j++ ) {   /* end at A[ j ] */
/* 5*/              ThisSum += A[ j ];  /* sum from A[ i ] to A[ j ] */
/* 6*/              if ( ThisSum > MaxSum )
/* 7*/                   MaxSum = ThisSum;  /* update max sum */
              }  /* end for-j */
          }  /* end for-i */
/* 8*/    return  MaxSum;
}
```

$$T(N) = O(N^2)$$

时间效率

- 时间效率
  - 数据集：100万个数
  - 计算机每秒运行10亿条指令
  - 算法1：

    $(10^6)^3/10^9 = 10^9$秒≈11574天≈31.7年
  - 算法2：

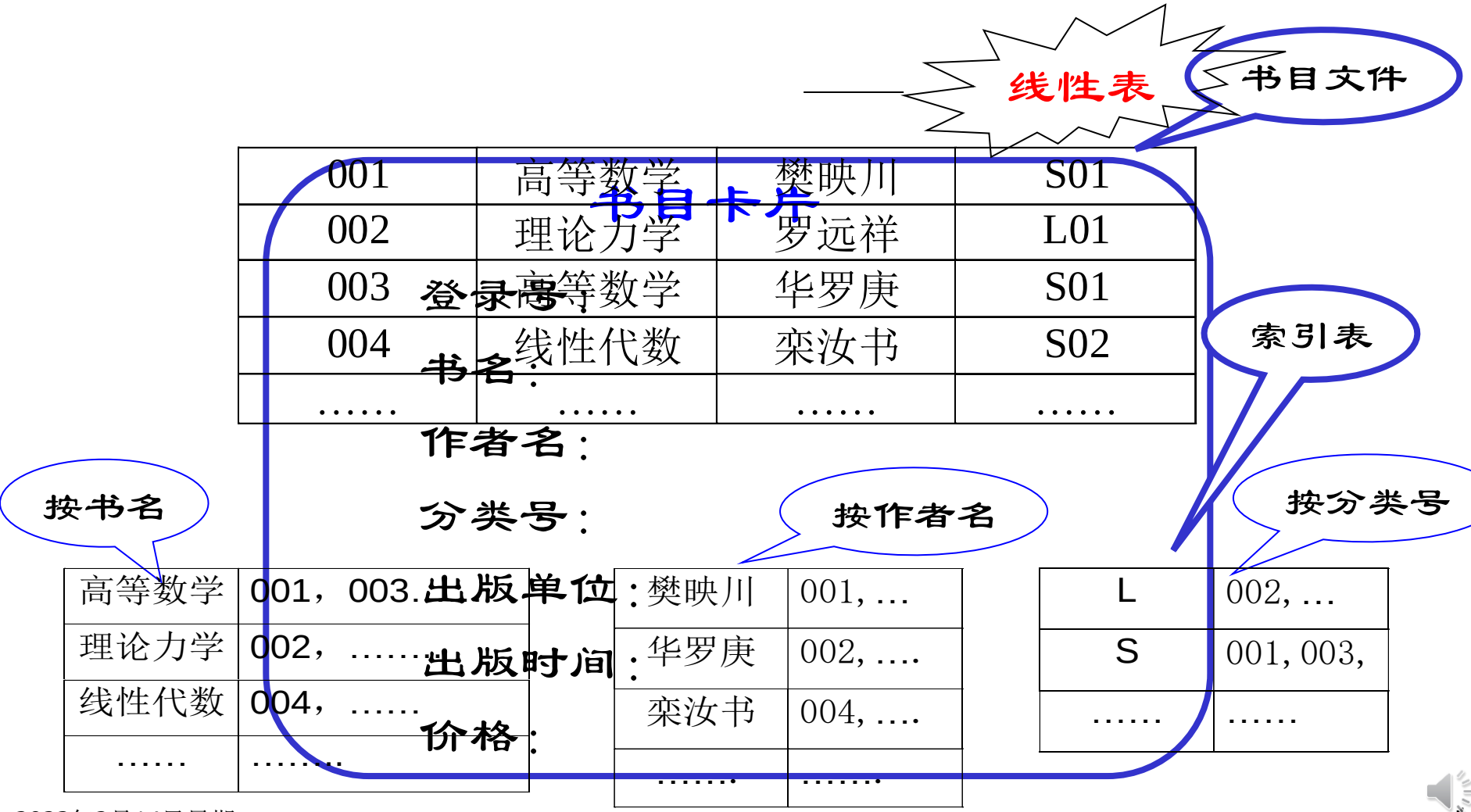
    $(10^6)^2/10^9 = 10^3$秒≈16.7分钟

应用中的典型数据关系

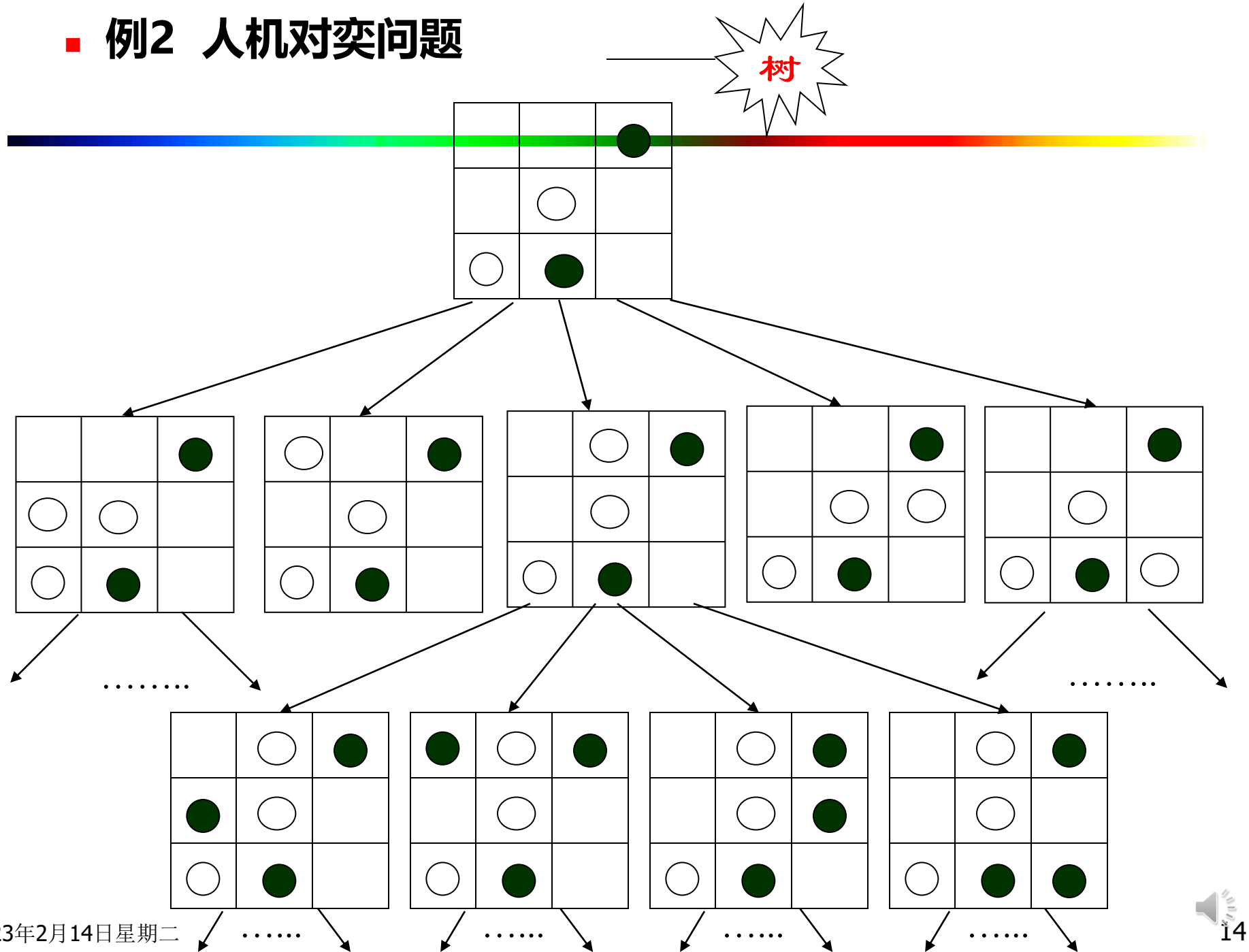
- 应用中，数据之间可能存在以下典型关系，如：
  - 一对一，线性关系
  - 一对多，层次关系
  - 多对多，网状关系

数据之间的关系举例

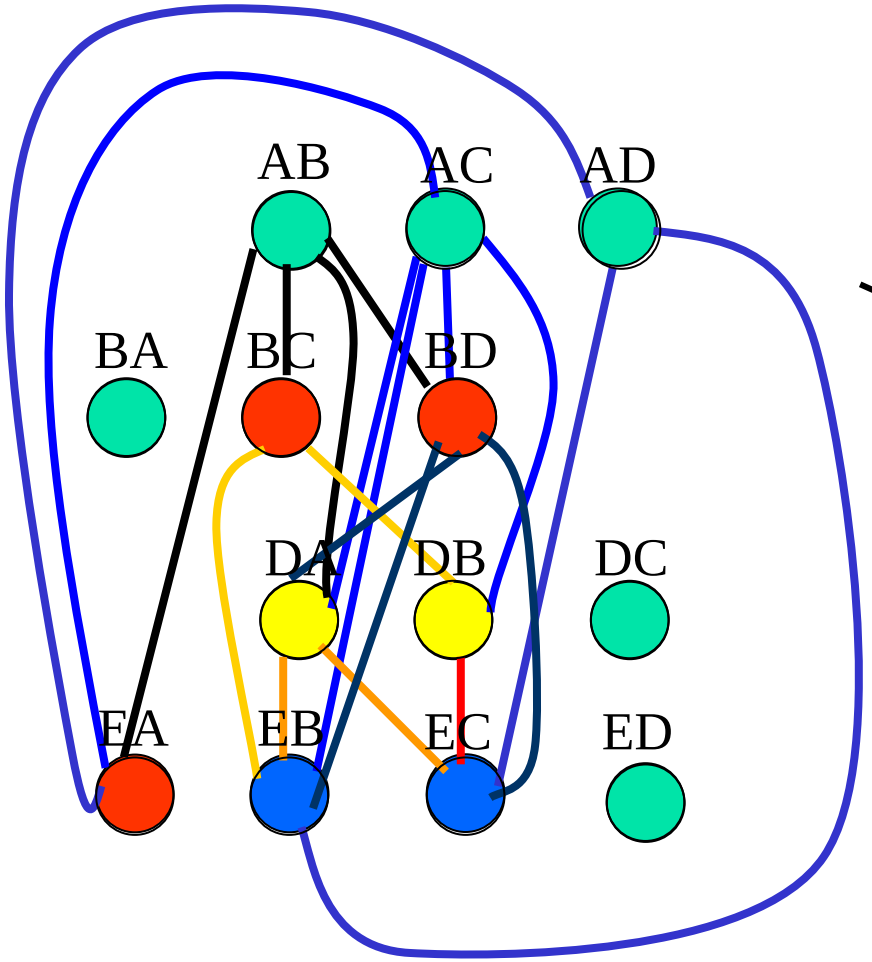
例1 书目自动检索系统

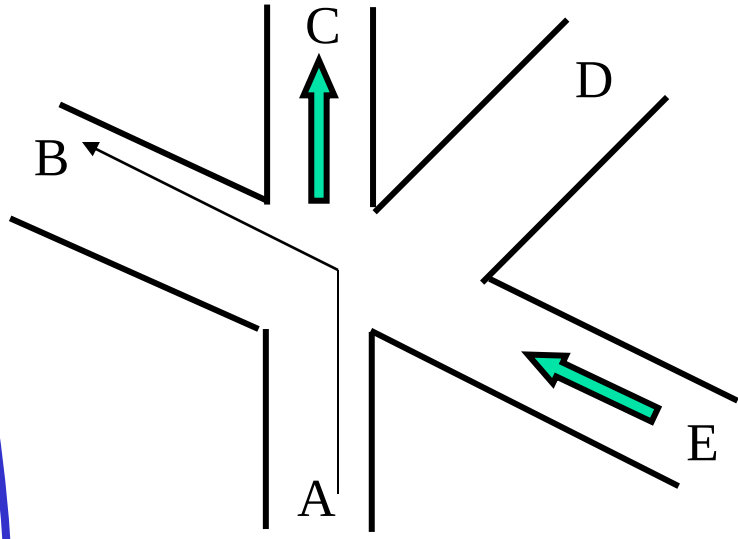


例2 人机对奕问题



例3 多叉路口交通灯管理问题





第1章 数据结构

- 1.1 数据结构的原则
- 1.2 抽象数据类型和数据结构
  - 数据的逻辑结构和数据的存储结构
- 1.3 问题、算法和程序

1.1.1 数据的逻辑结构

- **定义:由某一数据对象及该对象中所有数据成员之间的关系组成。记为:**

  Data_Structure = (K, R)

  其中，K是某一数据对象，R是该对象中所有数据成员之间的关系的有限集合。
- **理解：数据应使用什么样的数据结构，取决于数据成员之间的逻辑关系**
- **常见逻辑关系有: 线性(线性表，栈，队列，向量，字符串，多维数组，广义表)、树、图、文件（本质上是线性结构，而其索引则往往用树型)**

线性结构

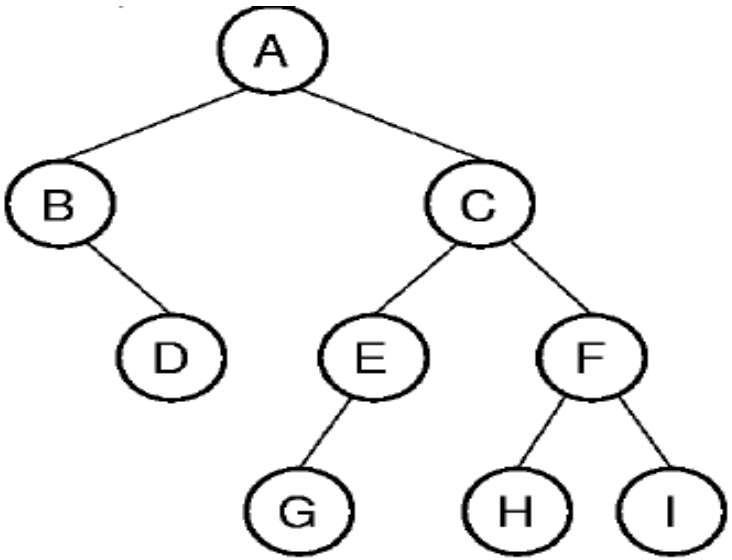
- **线性结构：各结点最多只有一个直接前驱和一个直接后继，有且仅有第一个结点无直接前驱，有且仅有最后一个结点无直接后继。**
- **线性表的逻辑结构B=(K, R)，其中**

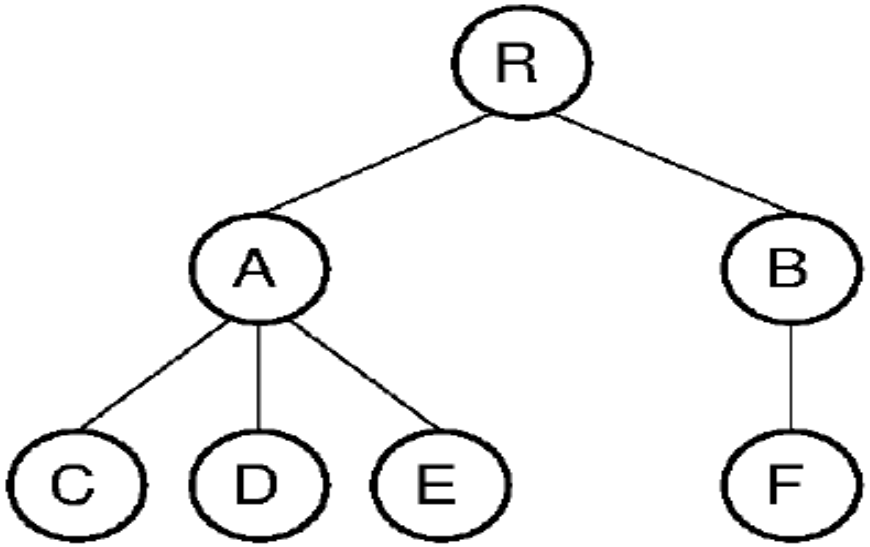
  $K = \{k_0, k_1, \ldots, k_{n-1}\}$

  $R = \{r\}$，$r = \{\langle k_{i-1}, k_i \rangle \mid k_i \in K, 1 < i < n\}$
- **图示法（注意与链表的图示区分开来）**

树形结构

树形结构： B=(K, R), R={r}, r满足下列条件：有且仅有一个称为**根**的结点没有前驱；其它结点有且仅有一个前驱；对于非根结点都存在一条从根到该结点的路径。

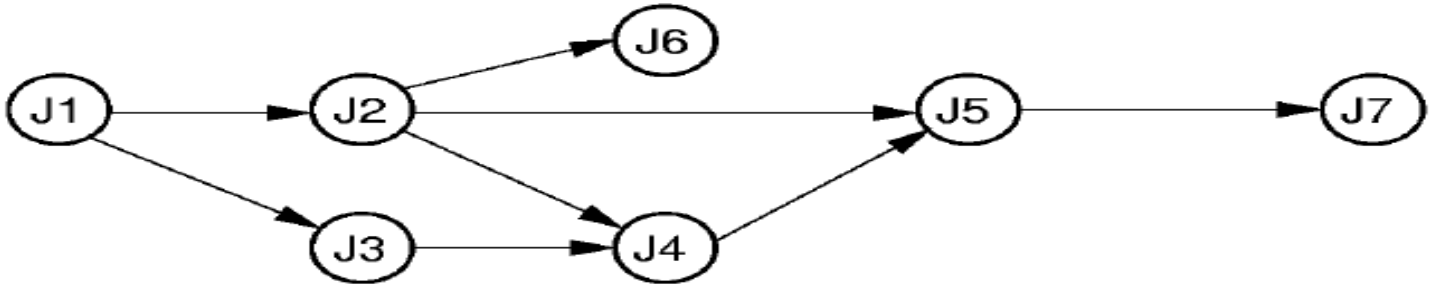




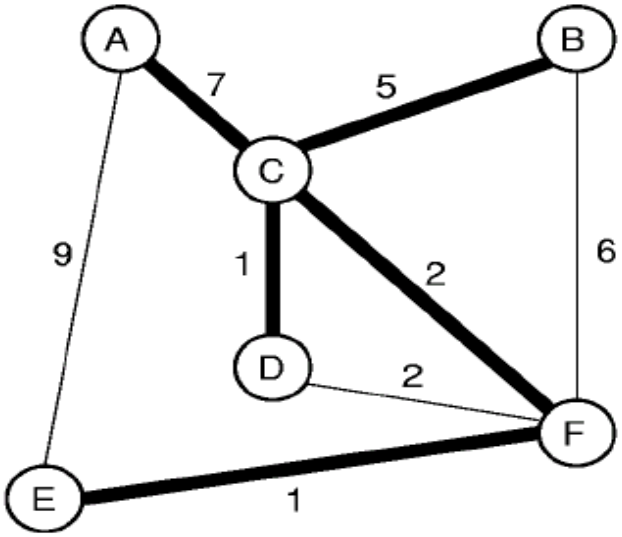
图的数据关系特点及图的分类

图： B= （K, R），R={r}, K中结点相对于r前驱和后继个数都没有限制。

有向图



无向图



左图逻辑结构B=(K, R)，其中K={A,B,C,D,E,F}，R={r}，
r={（A, C), (A,E), (B,C)，(E,F)
(B,F)，(C,D)，(C,F), (F,D)｝

数据结构的定义（慕课）

In computer science,

a data structure is a way of storing data in a computer so that it can be **used efficiently**.

Often a carefully chosen data structure will allow **the most efficient algorithm** to be used.

数据结构研究的问题

数据

- 信息的载体，是描述客观事物的数、字符、以及所有能输入到计算机中，被计算机程序识别和处理的符号的集合
- 包括：**数值性数据**和**非数值性数据**

- 数值问题 — 对以数学方式表示的问题求数值解。例如，代数方程计算、矩阵计算、线性方程组求解、数值积分、微分方程求解等；

- 非数值问题 — 求非数值解。例如，排序查找、模式匹配等。

数据结构研究的问题

- 非数值数据之间的结构关系，以及如何表示，如何存储，如何处理

数据结构的内容

- 数据的逻辑结构
  - 描述数据元素之间的逻辑关系
  - 集合、线性、树型、网状（图）
- 数据的物理（存储）结构
  - 数据逻辑结构（数据和结构关系）在计算机中的存储
- 数据结构的操作
  - 针对数据结构的基本操作（处理）
  - 初始化，插入，删除，查找…

数据结构与算法的关系

- 数据结构包含了实现基本操作的算法
  - 如：初始化，插入，删除，查找……
- 一些算法是基于数据结构解决问题的
  - 如：基于图的最短路径问题算法

1.1.2 数据的存储结构

定义：若B=（K，R）是一个数据结构，有一个映射S：K→M，对于每一个k∈K，都有唯一的z∈M（存储区域），使得S（k）=z，同时这个映射S应具有明显地或隐含地体现关系R的能力，则称对逻辑结构进行了存储。

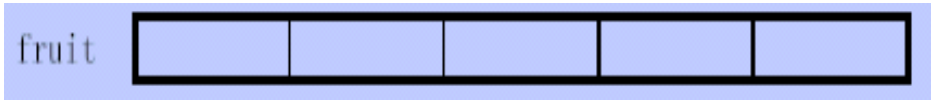
- 通俗地说，就是如何在存储中实现数据结构。

存储结构的类型

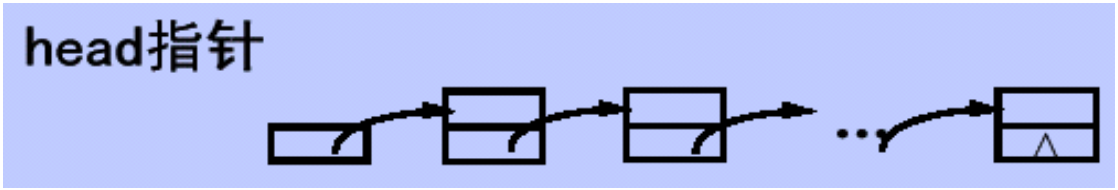
- 顺序存储
- 链接存储
- 索引存储
- 散列存储

存储结构

- 顺序方法：借助元素在存储器中的**相对位置**来表示数据元素间的逻辑关系



- 链接方式：借助指示元素存储地址的**指针**表示数据元素间的逻辑关系



- 索引方法：
- 散列方法：

存储密度(≤ 1) =数据本身存储量/整个结构占用存储量

链表： $\rho = n \times E/n(P+E) = E/(P+E)$

n表示线性表中当前元素的数目

存储结构的特点

- 顺序：逻辑上相邻，物理位置也相邻；
- 链式：借助指针表示数据元素之间的逻辑关系；
- 索引：高效查询，有序表的查询；
- 散列：高效查询，O（1），有冲突问题。

1.2抽象数据类型和数据结构

- **数据抽象与过程抽象，**实现信息隐蔽和局部化以一个严格定义的过程接口的方式，在数据结构上提供一个抽象。数据的实现和处理细节被隐蔽了。
- **抽象数据类型，**定义了一组运算的数学模型。与物理存储结构无关，使软件系统建立在数据之上（面向对象）

抽象数据类型（Abstract Data Type，ADT）

- **数据(data)**是描述客观事物的数字、字符以及其他能输入到计算机中并可被计算机程序处理的符号的集合。
- **数据元素(data element)**是数据的基本单位，即数据集合中的个体。它是计算机的最小可存取单位，在计算机程序中通常被作为一个整体进行考虑和处理。有时一个数据元素可由若干个数据项(data item)组成。数据项是数据的不可分割的最小单位，也是数据集合的最小可命名单位。
- **类型(type)**是一组值的集合。例如，布尔类型由两个值TRUE和FALSE组成；整数也构成一个类型。

ADT

- **数据类型(data type)是指一个类型以及定义在这个类型上的一组操作。例如整数类型是某个区间上的整数集合(区间大小依赖于不同的计算机), 定义在其上的操作为加、减、乘、除和取模等算术运算。**
- **抽象数据类型(abstract data type, 简称ADT)是对数据类型的逻辑形式的规范化描述。一个ADT的定义并不涉及它的实现细节, 这些实现细节对于ADT的用户是隐藏的。隐藏实现细节的过程称为封装(encapsulation)。**

抽象数据类型

抽象（abstract）

抽取反映问题的关键属性，忽略非关键属性

抽取

集合中数据元素之间的（结构）关系的本质

数据类型（data type）

程序设计语言中的概念，是程序设计语言中已经实现的数据结构。例如，C语言中的整型类型

数据类型分为原子类型和结构类型

抽象数据类型

In computer science, an abstract data type (ADT) is a mathematical model for data types, where a data type is defined by its behavior (semantics) from the point of view of a user of the data, specifically in terms of possible values, possible operations on data of this type, and the behavior of these operations.

Liskov, Barbara; Zilles, Stephen (1974). "Programming with abstract data types". Proceedings of the ACM SIGPLAN Symposium on Very High Level Languages. SIGPLAN Notices. 9. pp. 50–59.

Barbara Liskov

Stephen Zilles

抽象数据类型的组成

抽象数据类型的组成要素

1. 数据元素集合（数据对象）
2. 集合中数据元素的（逻辑结构）关系
   - 集合（没关系）、线性、树形和网状（图）
3. 针对数据元素集合的基本操作集
   - 创建一个初始数据结构
   - 往数据结构中增加一个数据元素
   - 往数据结构中删除一个数据元素
   - 在数据结构中查找一个特定元素
   - 获取数据结构本身的一些属性
   - 设置数据结构本身的一些属性

抽象数据类型的特征

- 使用与实现分离
- 实现封装和信息隐蔽
- 最重要的特点：
  - 抽象
    - 只反映集合中数据元素之间的逻辑结构关系
  - 信息隐蔽
    - 只规定数据元素操作的行为和有哪些基本操作，而对于操作的具体实现细节是封装和隐藏的

ADT的例子（1）

- 一个整数线性表的ADT应包含下列操作
  - 把一个新整数插入到线性表的末尾
  - 返回线性表中当前整数的个数
  - 重新初始化线性表
  - 删除线性表中特定位置上的整数
  - ……

线性表的ADT

```
class list {
// List class ADT
public:
list(const int =LIST_SIZE);
~list();
void clear();
void insert(const ELEM&);
void append(const
    ELEM&);
ELEM remove();
void setFirst();
void next();
 void prev();
int length() const;
void setPos(const int);
void setValue(const
    ELEM&);
ELEM currValue() const;
bool isEmpty() const;
bool isInList() const;
bool find(const
    ELEM&); };
```

不含算法的具体实现

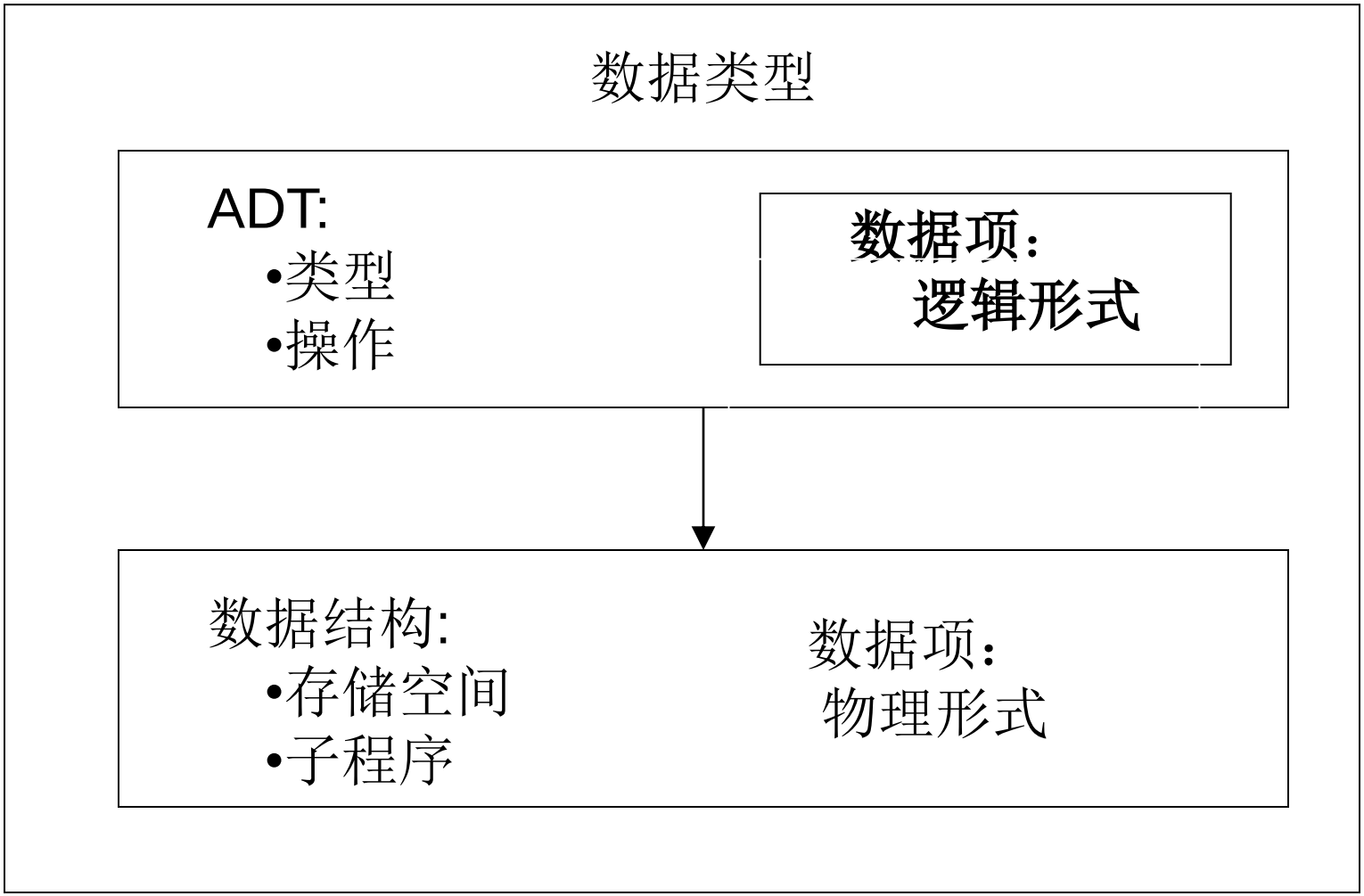
ADT的例子（2）

- 驾驶汽车的主要操作包括控制方向、加速和刹车
  - 几乎所有的汽车都通过转动方向盘控制方向， 通过踩油门加速，通过踩车闸刹车
  - 汽车的这种设计可以看作是具有“控制方向盘”、“加速”和“刹车” 操作的一个ADT
  - 两辆汽车可以用截然不同的方式实现这些操作
    - 但是大多数司机都能够驾驶许多不同的汽车
    - 因为ADT提供了一致的操作方法

逻辑形式和物理形式

- 数据项有逻辑形式（logical form）和物理形式（physical form）两个方面
  - **逻辑形式**：用ADT给出的数据项的定义
  - **物理形式**：数据结构中对数据项的实现
- 实现一个ADT时，是处理相关数据项的物理形式
- 在程序中其他地方使用ADT时，涉及到相关数据项的逻辑形式

数据项、抽象数据类型和数据结构之间的关系



抽象数据类型与数据结构

- 抽象数据类型是数据元素集合的抽象
- 数据结构是数据元素集合的实现
- 数据结构是抽象数据类型的表示（实现）