数据结构

Data Structure

主讲人：刘坤良

群名称：数据结构学习群

群　号：689547548

教材

《数据结构(C语言版)》严蔚敏等编著，清华大学出版社

参考书

【1】《数据结构题集》(C语言版）严蔚敏等编著，清华大学出版社

【2】殷人昆等，数据结构（用面向对象方法与C++描述），清华大学出版社， 1999年7月。

【3】殷人昆等，数据结构习题解析，清华大学出版社， 2002年4月。

【4】李春保，数据结构习题与解析（C语言篇），清华大学出版社， 2001年1月。

【5】丁宝康等，数据结构自学考试指导，清华大学出版社，2001年5月。

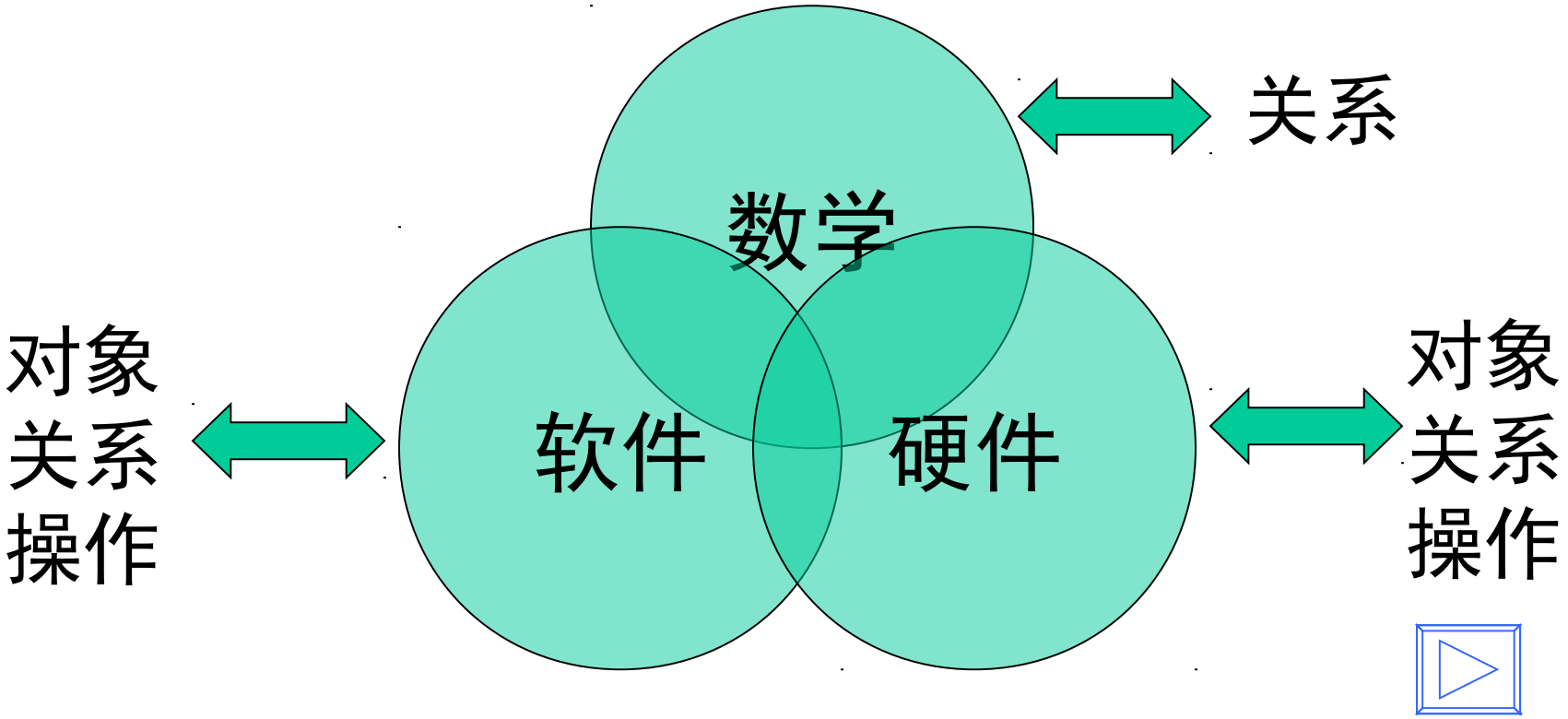
考 核

- 期末成绩＝平时成绩（**40%**）＋期末试卷成绩（**60%**）

- 平时成绩＝出勤＋作业＋实验

数据结构课程的地位

是介于数学、计算机硬件和计算机软件三者之间的一门计算机专业核心课程



第一章 绪论

- 什么是数据结构
- 基本概念和术语
- 抽象数据类型
- 算法分析
- 性能分析与度量

数学模型（Mathematical Model）

数学模型就是为了某种目的，用字母、数字及其它数学符号建立起来的等式或不等式以及图表、图象、框图等描述客观事物的特征及其内在联系的数学结构表达式。它将现实问题归结为相应的数学问题，并在此基础上利用数学的概念、方法和理论进行深入的分析和研究 。

例如： 数值计算的程序设计问题

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

（球面坐标系）

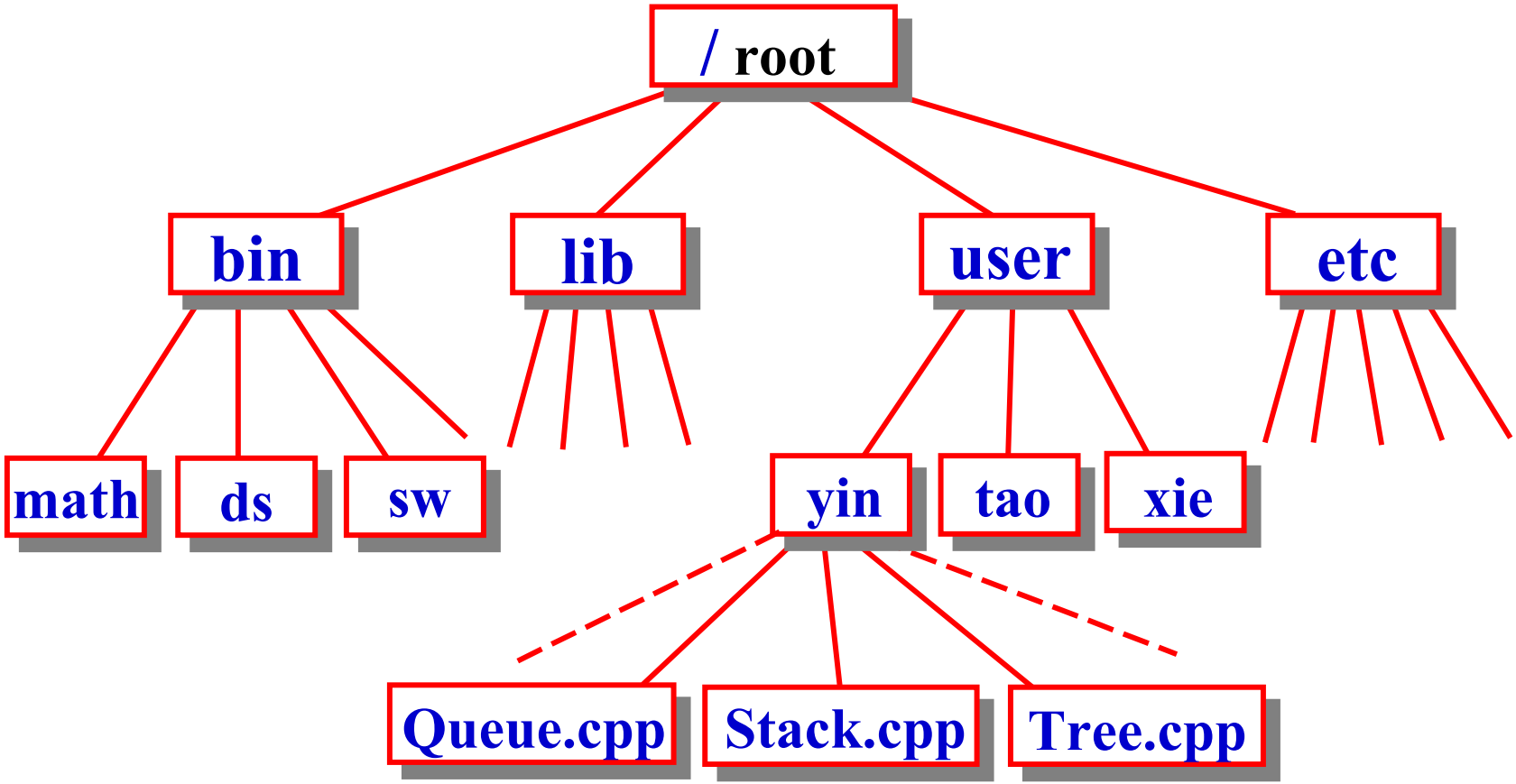
学生成绩表格

	学 号	姓 名	数据结构	系统结构	数学
1	20001	刘扬	89	69	67
2	20002	李平	70	83	89
3	20003	王方	86	81	78
4	20004	张策	69	69	78
5	20005	董立	79	89	68
6	20006	谢平	80	88	79
7	20007	高月	81	81	80
8	20008	刘平	89	85	87
9	20009	好园	86	80	84

选课单

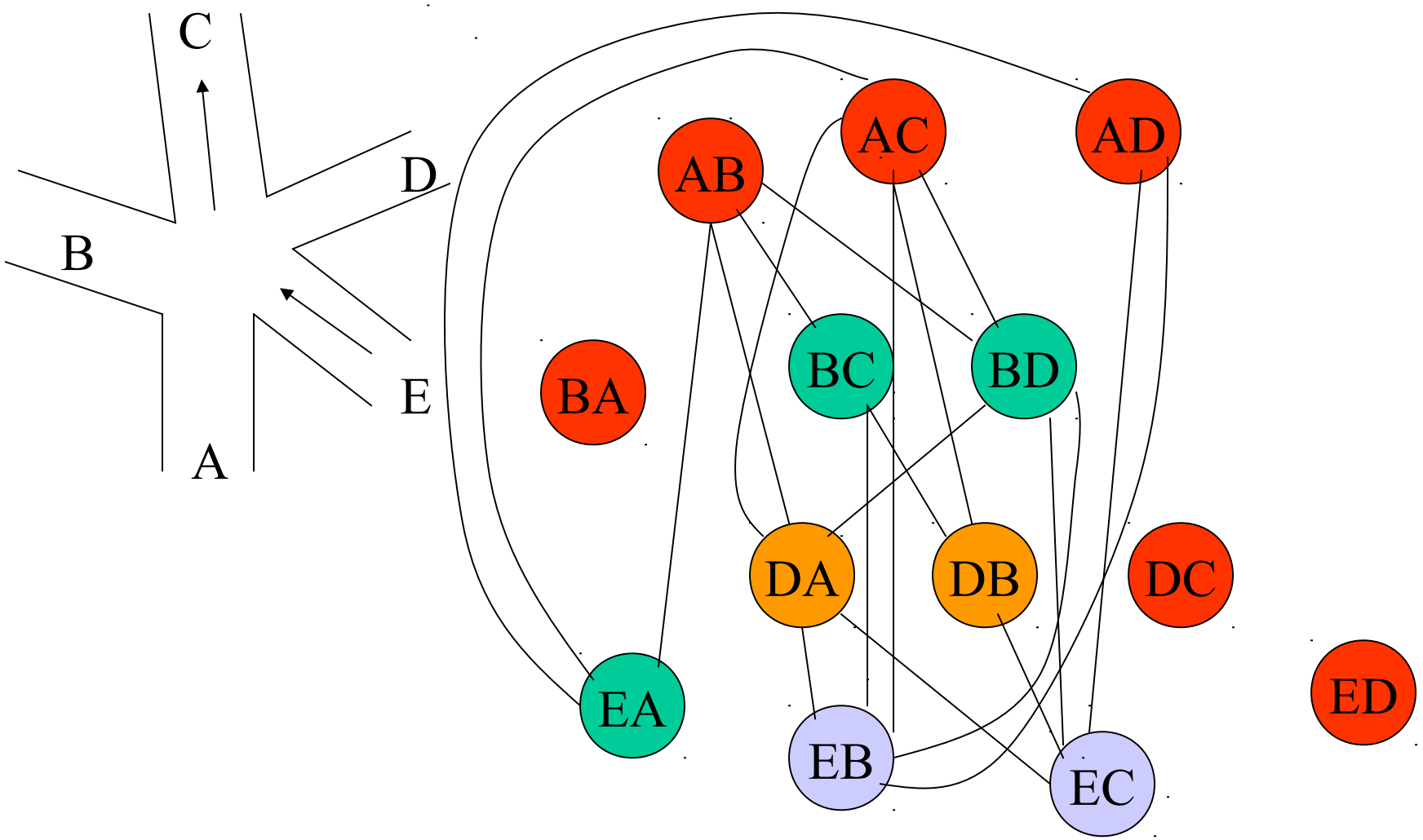
学号	课程号	时间	成绩
20001	DS2000 SX2000	2001,2 2000,9	78 87
20002	ART2000 DS2000	2002,2 2001,2	68 90
20003	SX2000 DS2000	2000,9 2001,2	87 78
20004	SX2000 ART2000	2000,9 2002,2	89 76

Windows 文件系统结构图



五叉路口交通管理

圆圈表示一条通路，两个圆圈之间的**连线**表示这两个圆圈表示的**两条通路不能同时通行**，**设置交通灯的问题等价为对图的顶点染色的问题**，要求对图上的每个顶点染一种颜色，并且要求有线相连的两个顶点不能有相同的颜色，而总的颜色种类应尽可能的少。



例如：当黄色灯亮时只有 D ⟶A 和 D⟶ B 两条路可以通行

五叉路口交通管理示意图 (图的存储结构)

- 综上，描述这类非数值计算问题的数学模型不是数学方程，而是表、树和图之类的数据结构。

- **什么是数据结构?**

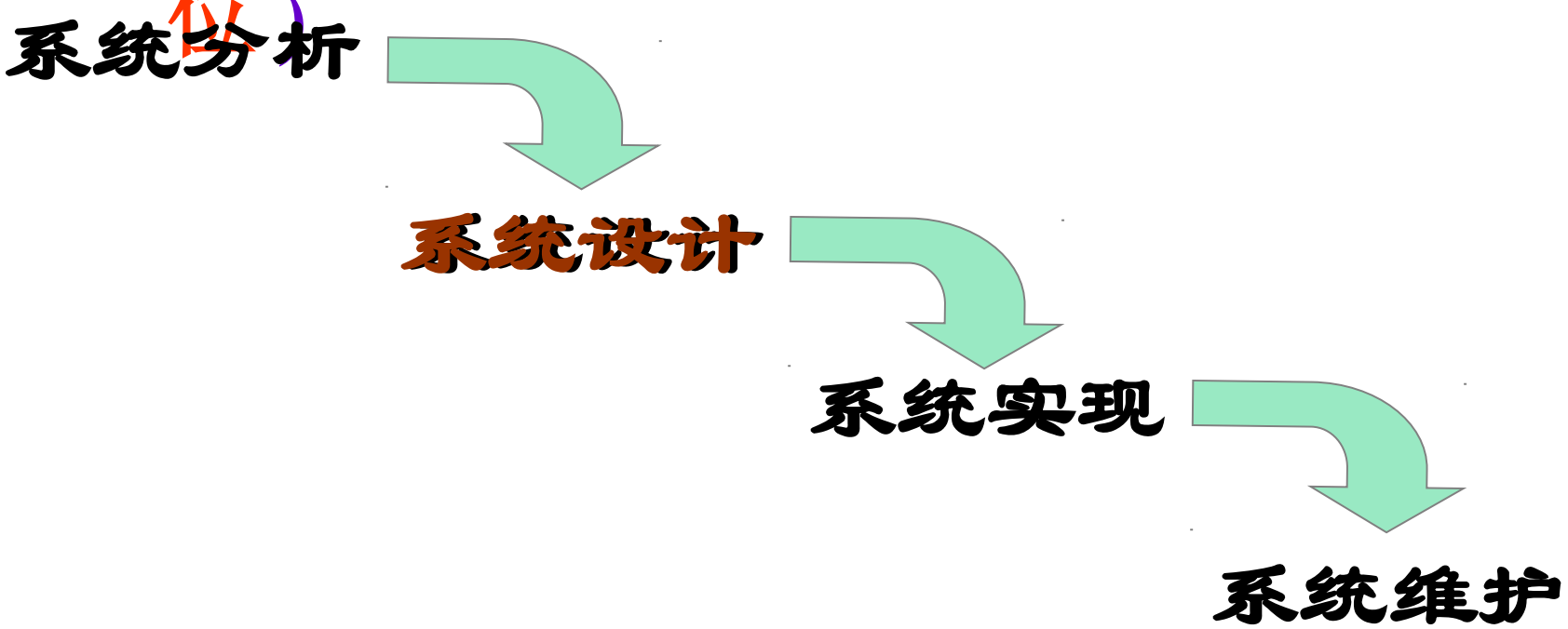
- 从广义上讲，数据结构描述现实世界实体的数学模型及其上的操作在计算机中的表示和实现。

- 数据结构是研究非数值计算的程序设计问题中计算机的操作对象以及它们之间的关系和操作等的学科。

为什么学习数据结构?

同样的数据对象，用不同的数据结构来表示，运算效率可能有明显的差异。

（数据结构在软件开发中的地位）



Niklaus Wirth(尼古拉斯·沃斯)

Algorithm + Data Structures = Programs

这个公式对计算机科学的影响程度足以类似物理学中爱因斯坦的“ E=MC^2”—— 一个公式展示出了程序的本质。

程序设计：为计算机处理问题编制一组指令集

处理问题的策略

算法：问题的数学模型（处理的信息怎么表示）

基本概念和术语

数据 (Data)

- 是信息的载体，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。
  - 数值性数据
  - 非数值性数据

数据元素（Data Element）

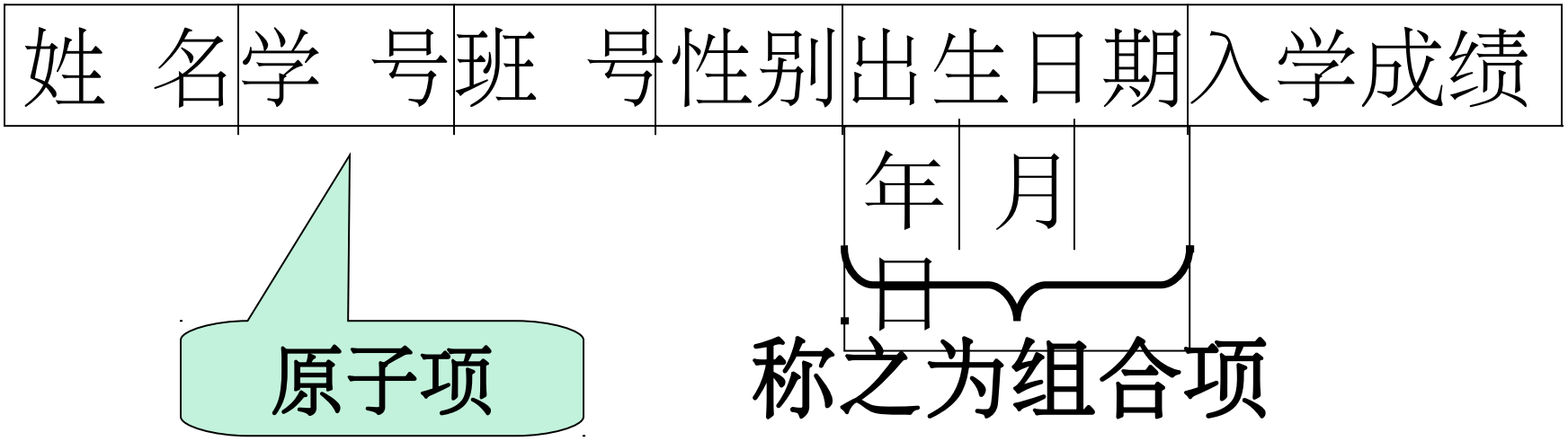
- 数据的基本单位。在计算机程序中常作为一个整体进行考虑和处理。
- 数据元素又称为元素、结点、记录
- 有时一个数据元素可以由若干数据项(Data Item) 组成。数据项是具有独立含义的最小标识单位。

数据元素有时由若干款项构成。

例如：描述一个学生的数据元素

其中每个款项称为一个“数据项”

它是数据结构中讨论的最小单位



数据对象 (data object)

- 具有相同性质的数据元素的集合。
  - 整数数据对象

$$N = \{ 0, \pm1, \pm2, \ldots \}$$

  - 字母字符数据对象

$$C = \{ \text{‘A’}, \text{‘B’}, \text{‘C’}, \ldots \text{‘F’} \}$$

数据结构（Data Structure）

存在一种或多种特定的关系数据元素的集合。记为：

Data_Structure = {D, S}

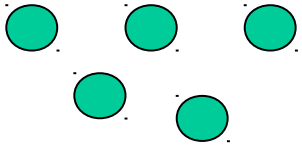
其中，D 是某一数据对象，S 是该对象中所有数据成员之间的关系的有限集合。

带**结构**的数据元素的集合

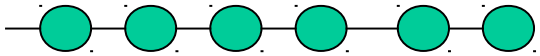
指的是数据元素之间存在的关系。

四个基本结构

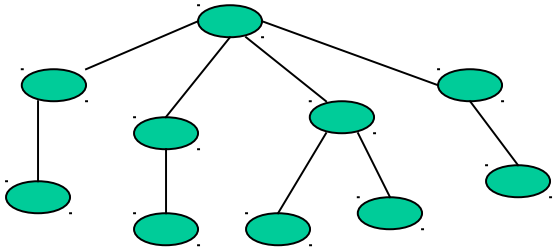
- 集合



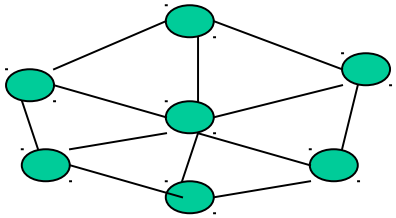
- 线性结构



- 树形结构



- 网状结构



数据结构包括“逻辑结构” 和“物理结构”两个方面 (层次):

数据的逻辑结构

- 从逻辑关系上描述数据，与数据的存储无关;
- 从具体问题抽象出来的数据模型;
- 与数据元素本身的形式、内容无关;

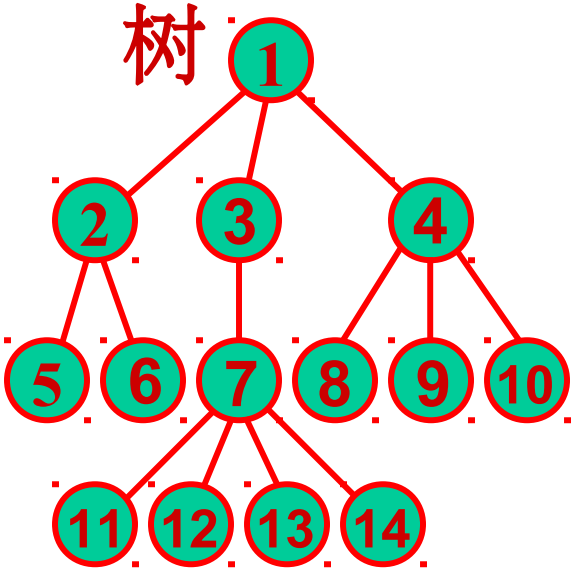
数据的逻辑结构分类

- **线性结构**
  - **线性表**
- **非线性结构**
  - **树**
  - **图（或网络）**

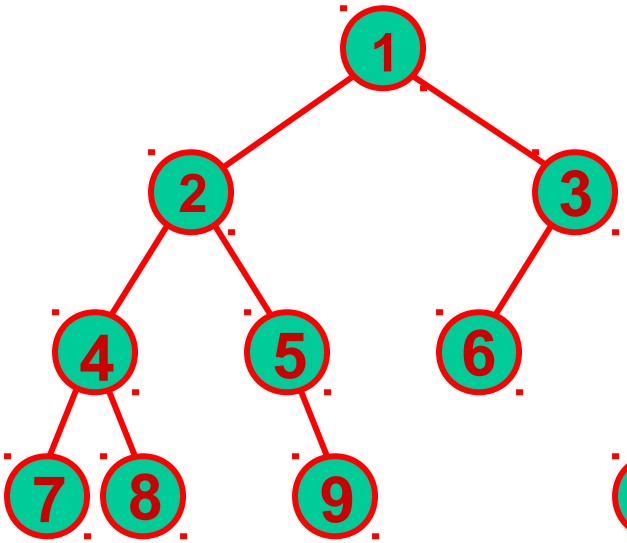
线性结构

树形结构

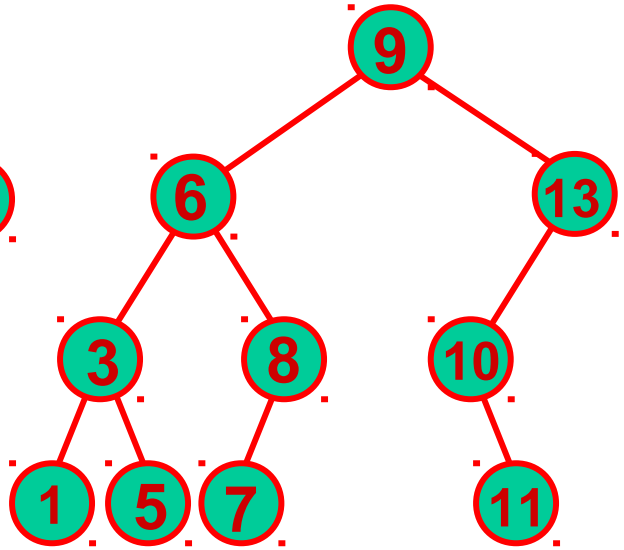
树



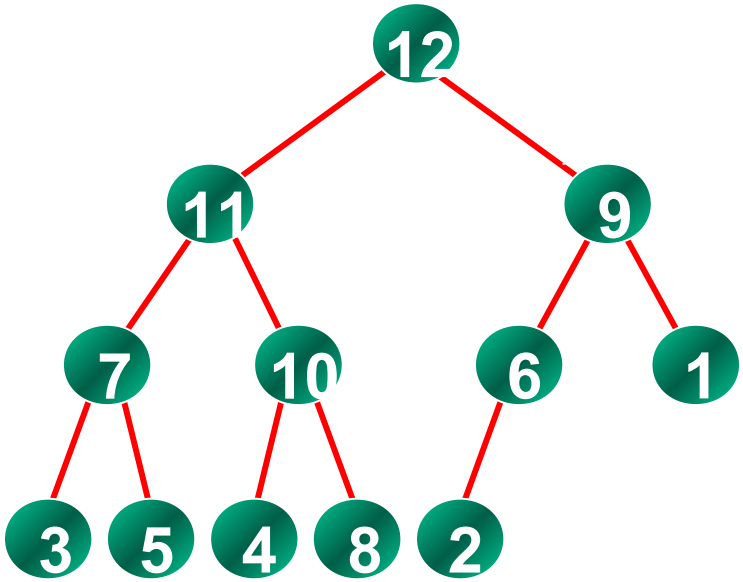
二叉树



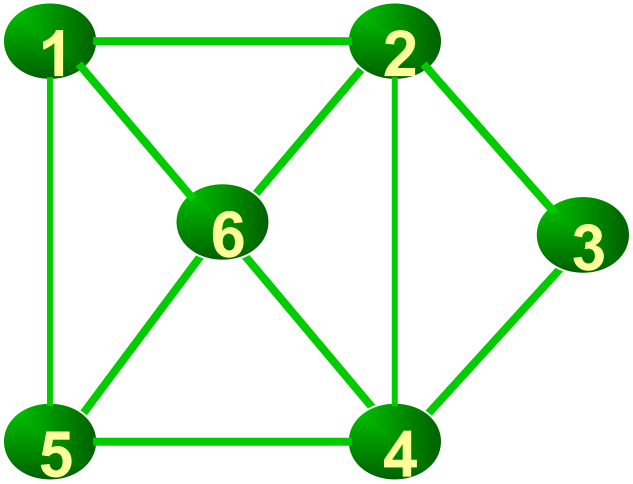
二叉排序



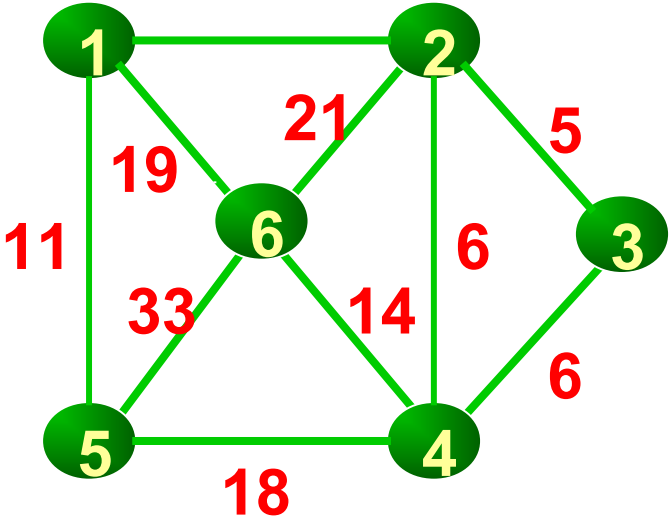
堆结构



图结构



网络结构



数据的存储结构（物理结构）

—— 逻辑结构在计算机中的映象

“数据元素”的映象 ？

“关系”的映象 ？

- 数据的存储结构依赖于计算机语言。
  - 顺序存储表示
  - 链接存储表示
  - 索引存储表示
  - 散列存储表示

抽象数据类型（Abstract Data Type）

- 数据类型

定义：一个值的集合和定义在这个值集上的一组操作的总称。

- C 语言中的基本数据类型

int　char　float　double　void

整型　字符型　浮点型　双精度型　无值

抽象数据类型

- 是指一个数学模型以及定义在此数学模型上的一组操作。
- 数据结构 + 定义在此数据结构上的一组操作 ＝ 抽象数据类型
- 例如：矩阵 ＋ (求转置、加、乘、求逆、求特征值)

构成一个矩阵的抽象数据类型

抽象数据类型的描述

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

其中，数据对象、数据关系用伪码描述；基本操作定义格式为

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。若初始条件为空，则省略之。
- “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。

抽象数据类型的表示和实现

- 抽象数据类型可以通过固有数据类型 (高级编程语言中已实现的数据类型) 来实现。
  - 抽象数据类型　　类　class
  - 数据对象　　数据成员
  - 基本操作　　成员函数 (方法)

1.4 算法和算法分析

一、算法

二、算法设计的原则

三、算法效率的衡量方法和准则

四、算法的存储空间需求

算法分析

- 定义：为了解决某类问题而规定的一个有限长的操作序列 。
- 特性：
  - 有穷性　算法在执行有穷步后能结束
  - 确定性　每步定义都是确切、无歧义
  - 可行性　每一条运算应足够基本
  - 输入　　有 0 个或多个输入
  - 输出　　有一个或多个输出

算法设计

- 例子： 选择排序
- 问题：递增排序
- 解决方案：逐个选择最小数据
- 算法框架：

```
for ( int i = 0; i < n-1; i++ ) {    //n-1 趟
    从 a[i] 检查到 a[n-1];
    若最小整数在 a[k]，交换 a[i] 与 a[k];
}
```

- 细化： Select Sort

```
void selectSort ( int a[ ], int n )
{
    // 对 n 个整数 a[0],a[1],…,a[n-1] 按递增顺序排序
     for ( int i = 0; i < n−1; i++ )
    {
    int k = i;
         for ( int j = i+1; j < n; j++ )
        if ( a[j] < a[k] ) k = j;
          // 从 a[i] 查到 a[n-1], 找最小整数，在 a[k]
   int temp = a[i];  a[i] = a[k];   a[k] = temp;
    }
}
```

二、算法设计的原则

设计算法时，通常应考虑达到以下目标：

1．正确性 [p14]

2．可读性

3．健壮性

4．高效率与低存储量需求

1．正确性

首先，算法应当**满足**以特定的“**规格说明**”方式给出的**需求**。

其次，对算法是否“正确”**的**理解可以有以下**四个层次**:

a．程序中不含语法错误;

b．程序对于几组输入数据能够得出满足要求的结果;

c．程序对于精心选择的、典型、苛刻且带有刁难性的几组输入数据能够得出满足要求的结果；

d．程序对于一切合法的输入数据都能得出满足要求的结果；

通常以第 c 层意义的正确性作为衡量一个算法是否合格的标准。

2. 可读性

算法主要是为了人的**阅读与交流**，其次才是为计算机执行。因此算法应该**易于人的理解**；另一方面，晦涩难读的程序易于隐藏较多错误而难以调试；

3．健壮性

当输入的数据**非法**时，算法应当恰当地作出反映或**进行相应处理**，而不是产生莫名奇妙的输出结果。并且，**处理出错的方法**不应是中断程序的执行，而应是**返回**一个**表示错误或错误性质的值**，以便在更高的抽象层次上进行处理。

4．高效率与低存储量需求

通常，效率指的是**算法执行时间**；存储量指的是算法执行过程中**所需的最大存储空间**。两者都与问题的规模 **n** 有关。

三、算法效率的衡量方法和准则

通常有**两种**衡量算法效率的方法：

事后统计法

缺点：1. 必须执行程序

2. 其它因素掩盖算法本质

事前分析估算法

和算法**执行时间**相关的**因素**:

1．**算法选用的策略**

2．**问题的规模**

3．编写程序的语言

4．编译程序产生的机器代码的质量

5．计算机执行指令的速度

3,4,5 与算法设计是无关的

一个特定算法的“运行工作量”的大小（复杂度），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。

算法的复杂度（效率）度量分为时间复杂度度量和空间复杂度度量

算法的时间复杂度是指当问题的规模从 1 增加到 n 时，算法所耗费的时间也由 1 增加到 T(n)，则称算法的时间复杂度为 T(n)，为问题规模 n 的函数。

如何估算

算法的时间复杂度?

时间复杂度度量

- 运行时间 = 算法中每条语句执行时间之和。
- 每条语句执行时间 = 该语句的执行次数（频度） * 语句执行一次所需时间。
- 语句执行一次所需时间取决于机器的指令性能和速度以及编译所产生的代码质量，很难确定。
- 设每条语句执行一次所需时间为单位时间 1，则一个算法的运行时间就是该算法中所有语句的频度（次数）之和。

例一 两个矩阵相乘

```
void mult(int a[], int b[], int& c[] ) {
  // 以二维数组存储矩阵元素，c 为 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];
      } //for
  } //mult
```

原操作

时间复杂度：

$2(n+1)+2n(n+1)+n^2+2n^2(n+1)+2*n^3$

确定一个算法的准确执行频度是困难的，而且没有必要（x=a 和 x=a+b*(c-d)-e/f; 具有相同的执行频度），用精确的执行频度比较两个算法，结果不一定有价值。因此，只要给出算法的执行频度的数量级，从 n 增长过程中分析算法执行次数增长的数量级，即可达

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

当且仅当存在正整数 c 和 n_0，使得 $T(n) \le cf(n)$

认为时间复杂度增长的数量级为 f(n) 。

$$f(n) = 2n^3 + 2n^2 + 2n + 1$$

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** 作为算法运行时间的衡量准则。

例二 选择排序

```
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;   //  选择第 i 个最小元素
      for ( k = i+1;  k < n;  ++k )
        if (a[k] < a[j] )  j = k;
      if ( j != i )  a[j] ←→ a[i]
    }
} // select_sort
```

四、算法的存储空间需求

算法的空间复杂度定义为：

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

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

算法的存储量包括：

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。

若**输入数据**所占空间只取决与问题本身，**和算法无关**，则只需要分析**除输入和程序之外的辅助变量所占额外空间**。

若所需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**。

若所需存储量依赖于特定的输入，则通常按最坏情况考虑。

本章学习要点

1. 熟悉各名词、术语的含义，掌握基本概念。

2. 理解算法五个要素的确切含义。

3. 掌握计算**语句频度**和**估算算法时间复杂度**的方法。

– 下面程序段的时间复杂度为（ ）。

```
for(int i=0; i<m; i++)
  for(int j=0; j<n; j++)
        a[i][j]=i*j;
```

– 设有一个递归算法如下

```
int fact(int  n){//n 大于等于 0
    if(n<=0)  return 1 ;
    else return  n*fact(--n) ;
   }
```

则计算 fact(n) 需要调用该函数的次数为 ( ) 次，不计 fact(n) 。

A． n　　B． n+1　　C． n+2　　D． n-l