



数据结构与算法设计

北京理工大学
高春晓





上课时间及安排

◆ 学时安排

🔑 课堂教学：72学时

🔑 上机实验：8学时

◆ 主讲教师：高春晓

🔑 Gao_chunxiao@bit.edu.cn

🔑 办公室：中心教学楼 1045室

🔑 13810667249





上课时间及安排

- ◆ 乐学课程:
- ◆ 2023-2024-I-数据结构与算法设计
 - 🔗 <https://lexue.bit.edu.cn/course/view.php?id=14014>
- ◆ 请务必注意作业关闭时间! 按时完成作业!



教材和参考资料

◆ 教材

- 📖 [1] [殷]殷人昆. 数据结构（C语言描述）（第二版）[M].北京:清华大学出版社, 2017.04.
- 📖 [2] [严]严蔚敏, 吴伟民编. 数据结构（C语言版）[M].北京:清华大学出版社, 2012.
- 📖 [3] [王]王晓东编著, 计算机算法设计与分析[M]. 北京: 电子工业出版社, 2012.02
- 📖 [4] [S] Sipser著, 唐常杰等译. 计算理论导引[M]. 北京: 机械工业出版社, 2017.11



教材和参考资料

◆ 参考书

- 📖 [1] [C]Cormen等著, 殷建平等译. 算法导论[M], 北京: 机械工业出版社, 2013.01
- 📖 [2] [M]Manber著, 黄林鹏等译. 算法引论-一种创造性方法[M], 北京: 电子工业出版社, 2010.01
- 📖 [3] [L]Lewis等著, 张利昂等译. 计算理论基础[M], 北京: 清华大学出版社, 2006.07
- 📖 [4] [W]Mark Allen Weiss. Data Structures and Algorithm Analysis in C (Second Edition)[M]. 北京: 机械工业出版社, 2010.

◆ 习题参考

- 📖 数据结构题集, 严蔚敏, 吴伟民. 清华大学出版社
- 📖 数据结构习题解析 第二版. 殷人昆. 清华大学出版社





课程内容

- ◆ 概述算法复杂度
- ◆ 线性表、栈和队列、数组和广义表
- ◆ 树、图
- ◆ 查找表
- ◆ 排序与分治
- ◆ 动态规划算法
- ◆ 有限自动机、图灵机
- ◆ 可判定性及可归约性
- ◆ 时间复杂性



课程教学目标

- ◆ 建立数据结构的基本概念，掌握基本数据操作。
- ◆ 掌握基本的算法策略，学习求解问题的方法。
- ◆ 理解可计算理论
- ◆ 提高编程能力，能够独立开发完成**300-500行**的C/C++语言程序。
- ◆ 能够完成一般问题的算法设计，为后续课程打好基础。





第一章 绪论



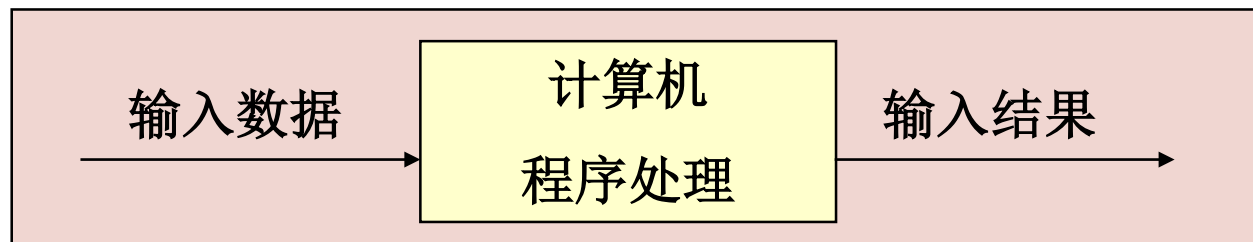


List of Contents

- ◆ 1.1 数据结构的定义
- ◆ 1.2 数据结构的基本概念
- ◆ 1.3 算法和算法的量度



1.1 数据结构的定义



计算机处理



用计算机解决问题的过程（程序设计）

- ◆ 程序设计:为计算机处理问题编制一组指令集
- ◆ 算法:处理问题的策略
- ◆ 数据结构:数据的组织与操作





1.1 数据结构的定义

◆ Niklaus Wirth:

‣ Algorithm + Data Structures = Programs

- Data Structures =
 - Data Set + Relations + **Operations**

- 数据集 + 关系 + **操作**



Very Important!



数据结构研究的主要内容

- ◆ 计算机处理的问题
- ◆ 计算机处理数据的种类和能力
 - ┑ 数字 (整数, 实数)
 - ┑ 字符、文字、图形、图象、声音
- ◆ 计算机应用
 - ┑ 数值领域→数值计算问题
 - ┑ 非数值领域→非数值计算问题



数值计算问题举例

◆ 数值计算的程序设计问题

🔧 例1: 物体从100米高的塔顶落到地面的时间——二次方程

🔧 例2: 结构静力分析计算——线性代数方程组

◆ 建模:

🔧 涉及对象: 高度 h , 时间 t , 重力加速度 g ($g=9.8$)

🔧 对象

◆ 设计求

◆ 编程:

```
main ( )  
{  
    float t, h , g ; g=9.8;  
    scanf ("%f", &h);  
    t = aqrt(2*h/g);  
    printf ("The falling time is %f\n", t);  
}
```



非数值计算的程序设计问题1

◆ 例1：求一组整数(假设5个)中的最大值

◆ 建模：

‖ 涉及对象： 5个整数

‖ 对象之间的关系： 大小关系

◆ 设计求解问题的方法：

‖ 基本操作是“比较两个数的大小”

‖ 首先将第一个数记为当前最大值，然后依次比较其余 $n-1$ 个整数，如果该某个整数大于当前最大值，就更新当前最大值。

◆ 编程：



非数值计算的程序设计问题1

◆ 例1：求一组整数(5个)中的最大值

```
main ( )
{
    int d[5], i, max;
    for( i=0; i<5; i++)    scanf ("%d", &d[i]);
    max = d[0];
    for( i=1; i<5; i++)
        if ( max < d[i]) max = d[i];
    printf ("The max number is %f\n", max);
}
```

非数值计算的程序设计问题2

- ◆ 例2 已知研究生的选课情况，试设计安排课程的考试日程的程序。要求在尽可能短的时间内完成考试。

A	B	C	D	E	F
算法分析	形式语言	计算机图形学	模式识别	网络技术	人工智能

	杨润生	石磊	魏庆涛	马耀先	齐砚生
选修1	A	C	C	D	B
选修2	B	D	E	F	F
选修3	E		F	A	



非数值计算问题举例2: 建模

◆ 建立模型

🔧 涉及对象: 课程考试

🔧 约束关系: 同一学生选修的课程不能安排在同一时间考试

🔧 模型: 图——表达课程之间的约束关系

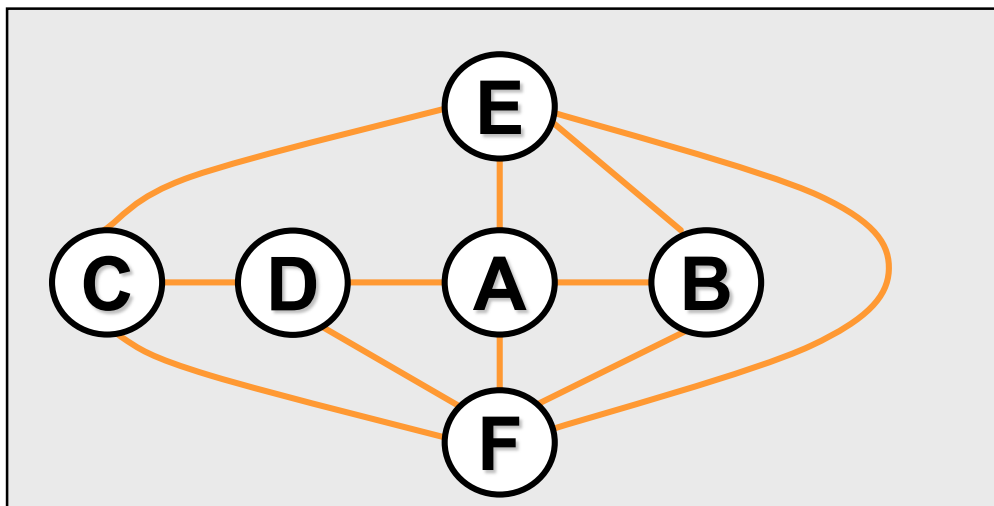
A	B	C	D	E	F
算法分析	形式语言	计算机图形学	模式识别	网络技术	人工智能

	杨润生	石磊	魏庆涛	马耀先	齐砚生
选修1	A	C	C	D	B
选修2	B	D	E	F	F
选修3	E		F	A	



非数值计算问题举例2: 建模

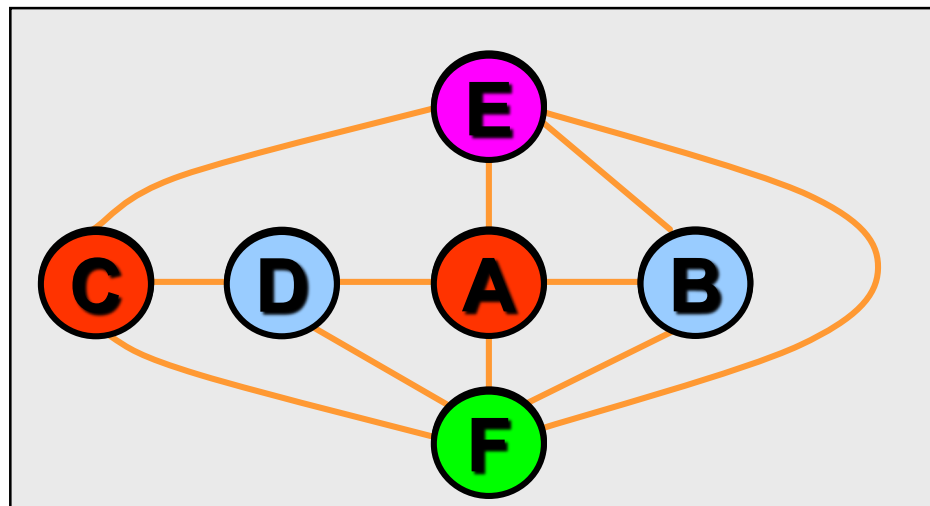
◆ 图：顶点：表示课程；边：同一学生选修的课程用边连接



	杨润生	石 磊	魏庆涛	马耀先	齐砚生
选修1	A	C	C	D	B
选修2	B	D	E	F	F
选修3	E		F	A	

非数值计算问题举例2: 求解（着色法）

- ◆ 每种颜色代表一个考试时间，用尽量少的颜色为顶点着色；
- ◆ 着色原则：相邻顶点着不同颜色；不相邻顶点着相同颜色；
- ◆ 着相同颜色的顶点（课程）安排在同一时间考试；



不冲突课程组

{ A, C }	{ B, D }	{ E }	{ F }
----------	----------	-------	-------

考试日程:

第1天: A, C

第2天: B, D

第3天: E

第4天: F

非数值计算问题举例2: 求解 (着色法)

◆ 求解考试日程的流程

- 1) $i=1$; $V = \{ \text{图中所有顶点的集合} \}$ (i 表示第 i 天)
- 2) 若 V 非空 DO
 - 2-1) 置 **NEW** 为空集合;
 - 2-2) 在 V 中取一点, 找出所有与之不相邻, 且彼此也不相邻的顶点;
 - 2-3) 将这些顶点加入 **NEW**, 从 V 中去掉这些顶点;
(第 i 天考试课程为 **NEW** 中顶点所对应的课程)
 - 2-4) 输出 **NEW** 中顶点所对应的课程;
 - 2-5) $i = i + 1$;
- 3) 若 V 空, 结束

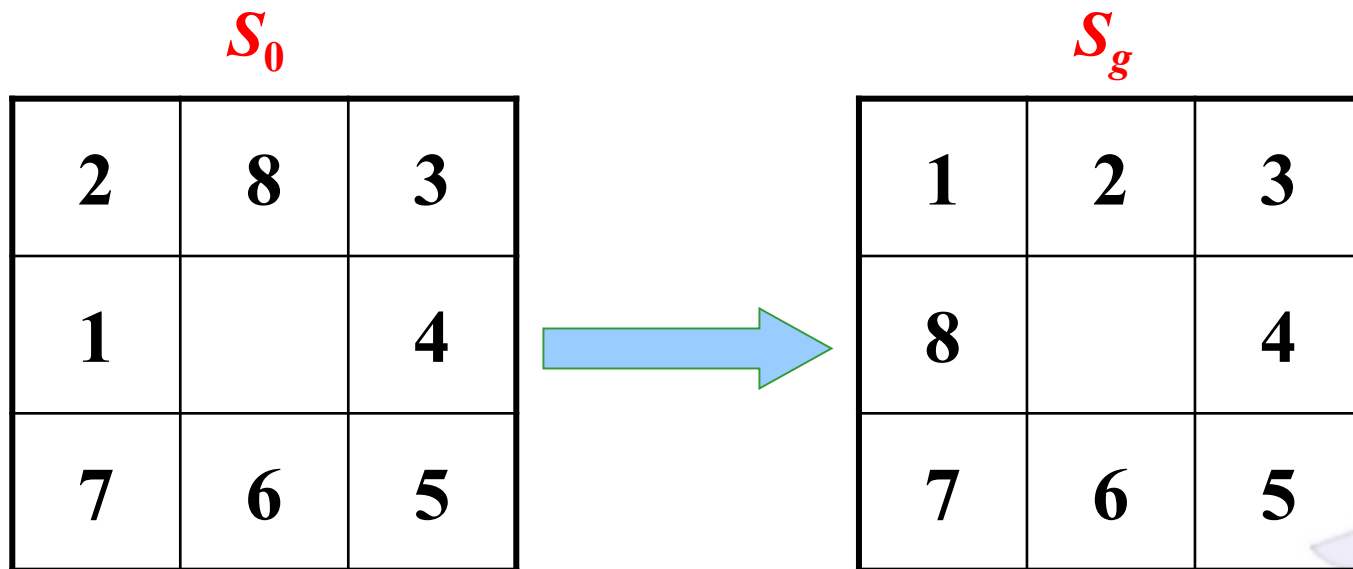
非数值计算问题举例3

◆ 例3：八数码问题

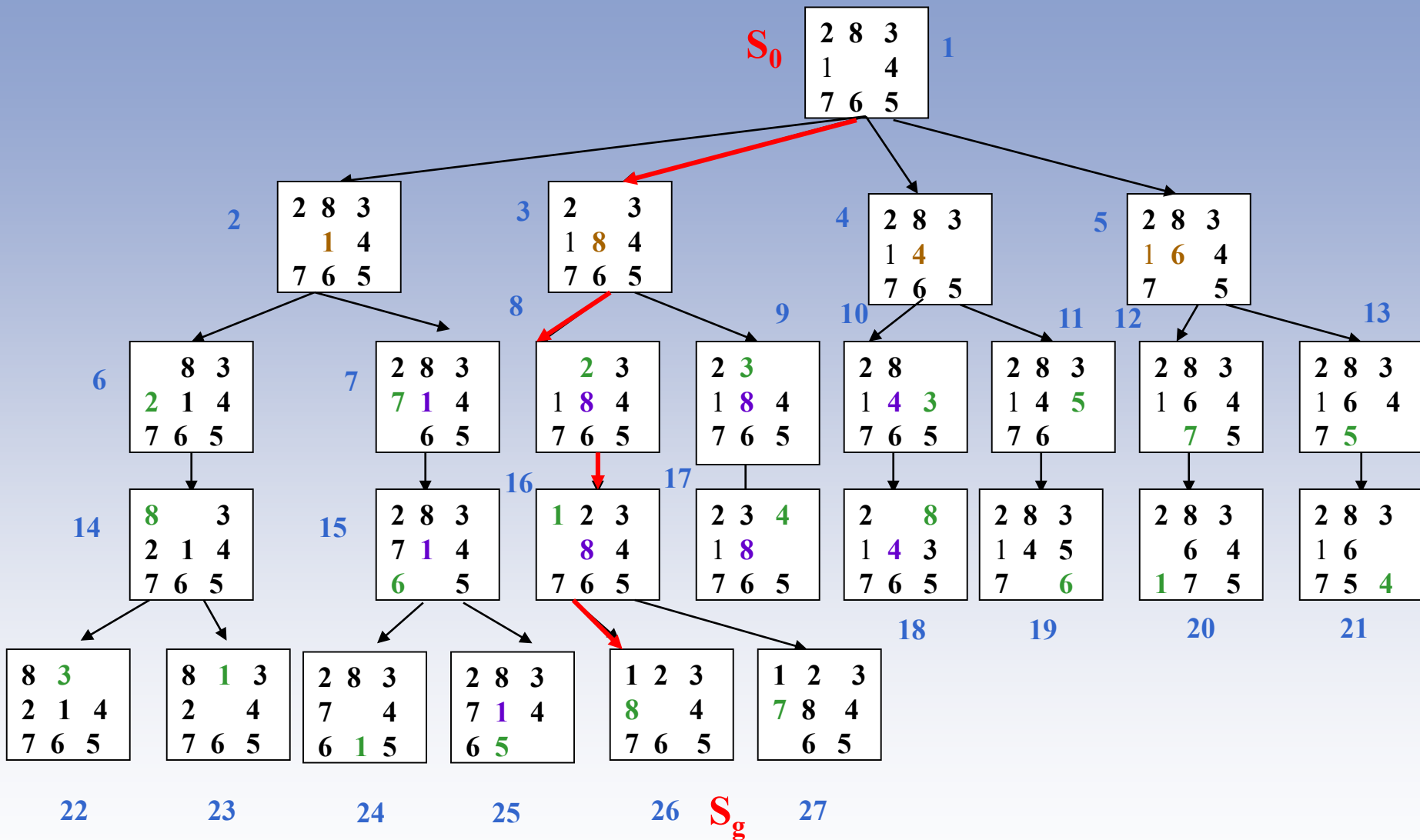
🔧 **涉及对象：** 棋盘→棋盘的格局

🔧 **对象关系：** 移动方格操作

🔧 **基本关系：** 空格上移，空格下移，空格左移，空格右移



八数码问题



Route: $1(S_0) \rightarrow 3 \rightarrow 8 \rightarrow 16 \rightarrow 26(S_g)$

数据结构研究的主要内容

◆ 概括地说：

- 🔧 数据结构是一门研究“程序设计问题中计算机操作对象以及它们之间的关系和操作”的学科。

◆ 具体地说：

- 🔧 数据结构主要研究数据之间有哪些结构关系，如何表示，如何存储，如何处理。





1.2 基本概念和术语

- ◆ 1.2.1 数据与数据结构
- ◆ 1.2.2 数据类型
- ◆ 1.2.3 抽象数据类型



1.2.1 数据与数据结构

◆ 数据（data）：

- 所有能被输入到或产生在计算机中，且能被计算机处理的符号的集合。
- 是对计算机处理的对象的一个统称，被看作为信息的载体。

◆ 数据元素（data element）：

- 是数据结构中讨论的基本单位
- 例如：描述一个学生的数据元素如下表

◆ 数据项（data item）：

- 数据不可分割的最小标识单位。
- 一个数据元素可由若干数据项组成。

学号	姓名	出生日期	入学日期	班级	专业
----	----	------	------	----	----

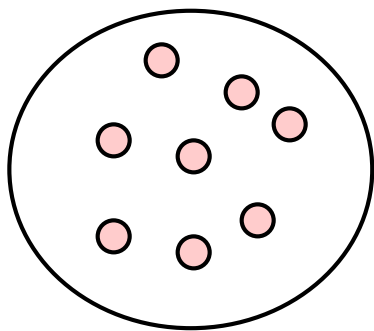
数据在系统开发中的视图

- ◆ 数据在系统开发中的讨论范畴分为
数据结构 + 数据内容 + 数据流
- ◆ 数据结构指某一数据元素集合中数据元素之间的关系。
- ◆ 数据内容指这些数据元素的具体涵义和内容。
- ◆ 数据流指这些数据元素在系统处理过程中是如何传递和变换的。
- ◆ 因此，讨论数据结构时，主要不是讨论数据元素的内容和如何传递。

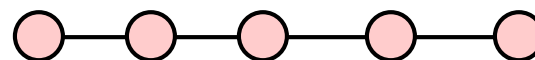


1.2.1 数据与数据结构

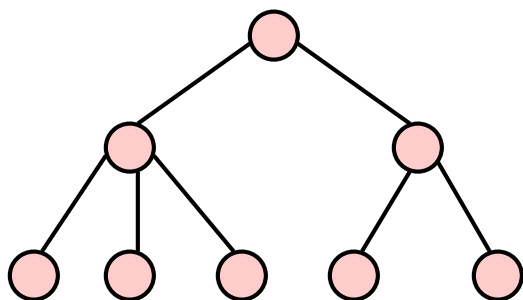
◆ **数据结构**：带有**关系**和**运算**的数据集合



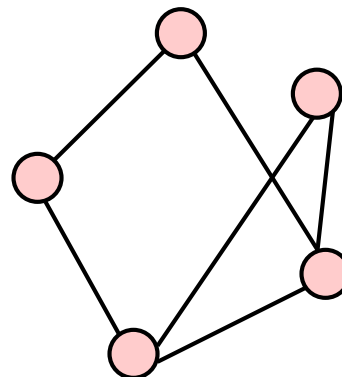
a. 集合关系



b. 线性关系



c. 树型关系



d. 图型关系

数据结构

- ◆ **数据结构**：带有**关系**和**运算**的数据集合
- ◆ 数据之间结构关系：是具体关系的抽象。例1：

学生间学号顺序关系是一种**线性结构**关系

线性结构关系是对学生间学号顺序关系的一种抽象表示

学号	姓名	出生日期	入学日期	班级	专业
cs001	张扬	02121995	01092014	cs20141	计算机
cs002	李明	07111995	01092014	cs20141	计算机
cs003	杨华	01051995	01092014	cs20142	计算机
cs004	贾茹	15041995	01092014	cs20142	计算机



- 查询学生信息
- 插入学生信息
- 修改学生信息
- 删除学生信息

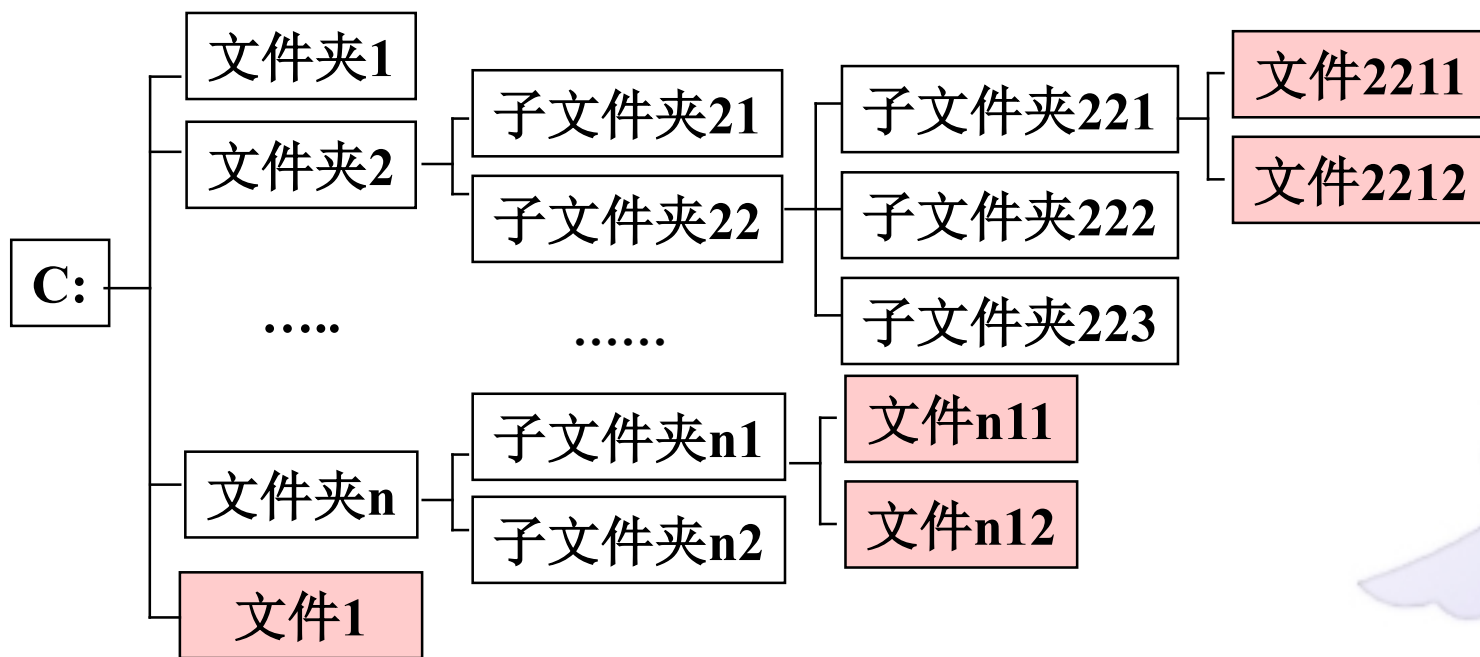
学号	姓名	出生日期	入学日期	班级	专业
cs001	张扬	02121995	01092014	cs20141	计算机
cs002	李明	07111995	01092014	cs20141	计算机
cs003	杨华	01051995	01092014	cs20142	计算机
cs004	贾茹	15041995	01092014	cs20142	计算机

数据结构

◆ 例2：计算机文件系统

文件夹或文件间的关系是一种**树型结构**关系

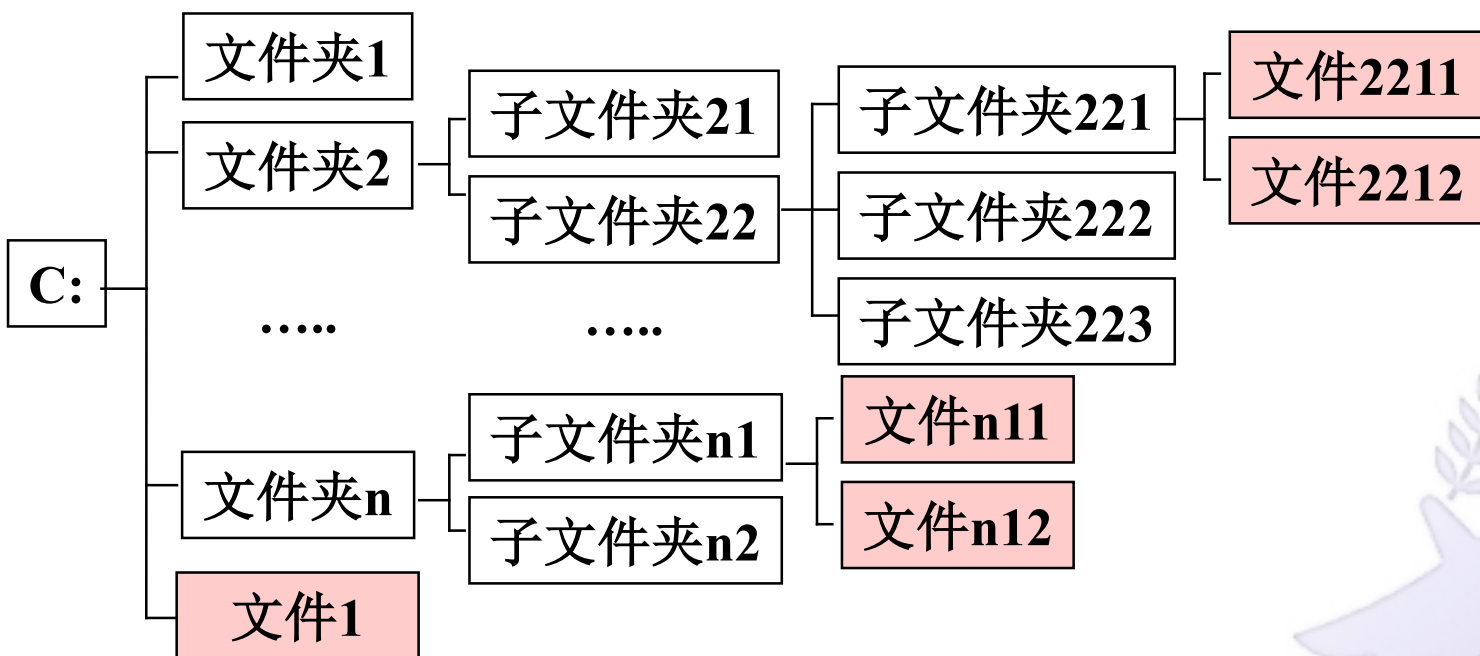
树型结构关系是对文件系统关系的一种抽象表示



数据结构

◆ 例2：定义计算机文件系统中的操作

- 🔑 查找、新建、删除文件夹
- 🔑 查找、新建、删除文件





数据结构

◆ 定义地铁图中的操作:

- 🚪 查询地铁站
- 🚪 规划乘车路线
- 🚪 增加地铁站
- 🚪 关闭地铁站...

可见，不同的“**关系**”构成不同的“**数据结构**”。

这些关系称为**数据的逻辑结构**。

数据结构是相互之间存在着某种逻辑关系的

数据元素集合及定义在该集合上的**操作集合**

数据结构的表示

◆ (1) 图示表示

图示表示是由顶点和边构成的图，其中顶点表示数据，边表示数据之间的结构关系。

◆ (2) 三元组表示：

◆ 数据结构是一个三元组：

$$\text{Data_Structures} = (D, R, M)$$

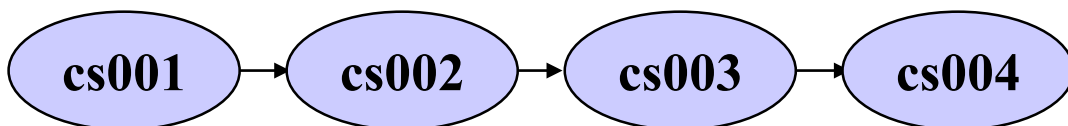
‖ D 是数据元素的有限集合，

‖ R 是 D 上关系的有限集合。

‖ M 是作用在该集合所有数据成员之上的方法（或操作）

数据结构的表示

◆ 线性关系的表示

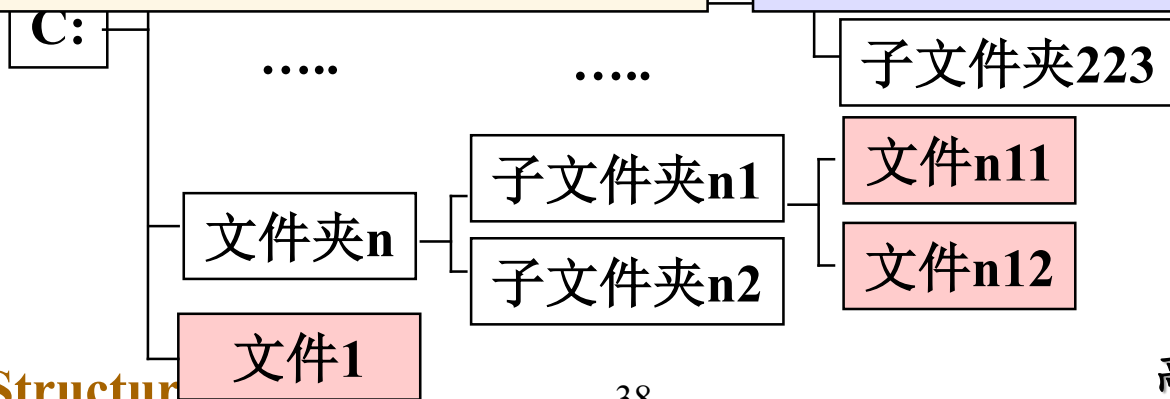
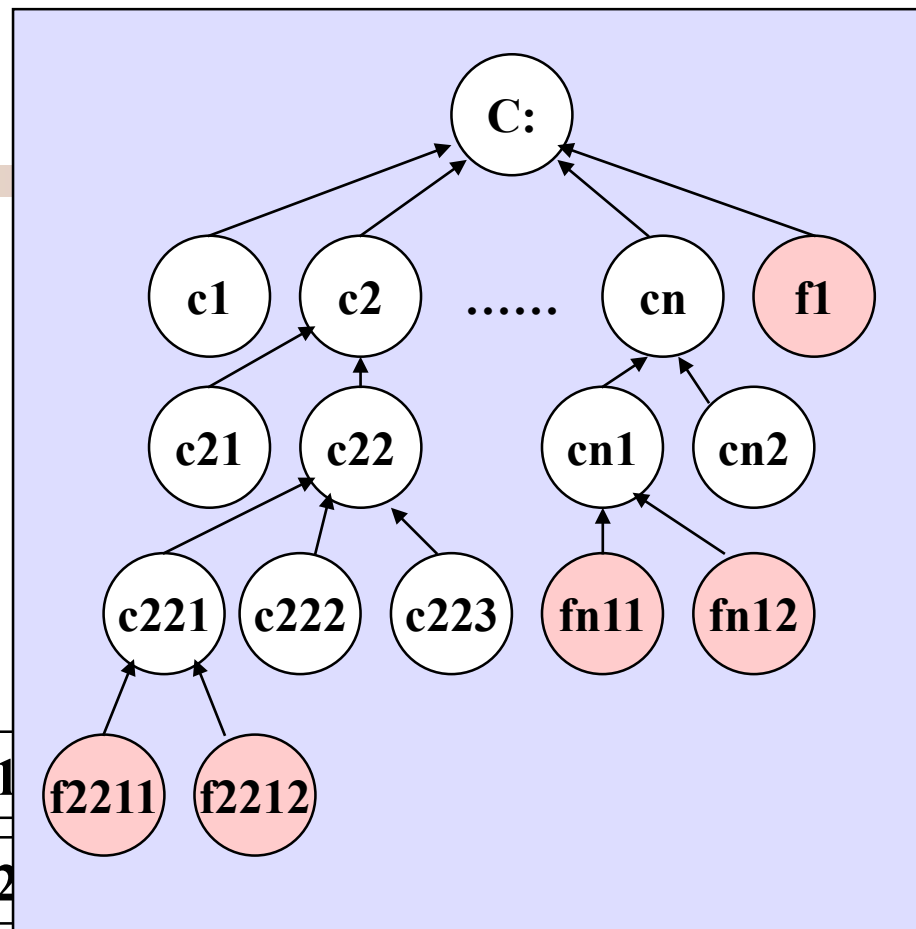
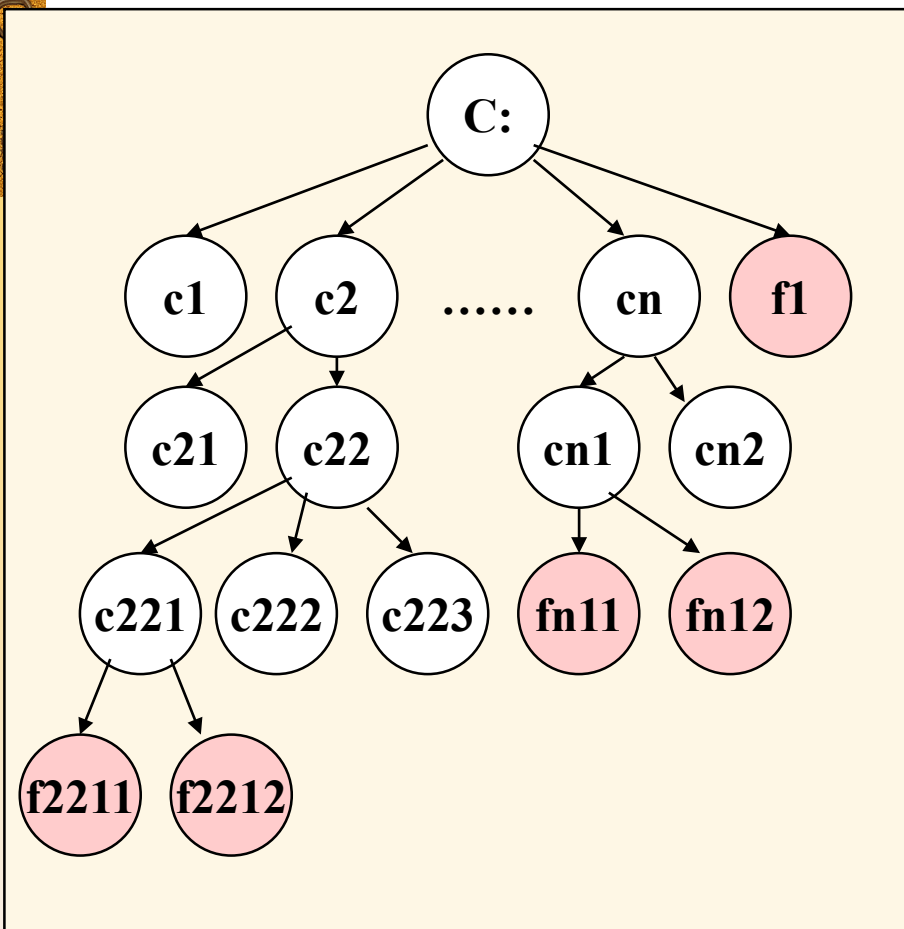


$D = \{ \text{cs001}, \text{cs002}, \text{cs003}, \text{cs004} \}$

$S = \{ R \}$

$R = \{ \langle \text{cs001}, \text{cs002} \rangle, \langle \text{cs002}, \text{cs003} \rangle, \langle \text{cs003}, \text{cs004} \rangle \}$

学号	姓名	出生日期	入学日期	班级	专业
cs001	张扬	02121995	01092014	cs20141	计算机
cs002	李明	07111995	01092014	cs20141	计算机
cs003	杨华	01051995	01092014	cs20142	计算机
cs004	贾茹	15041995	01092014	cs20142	计算机



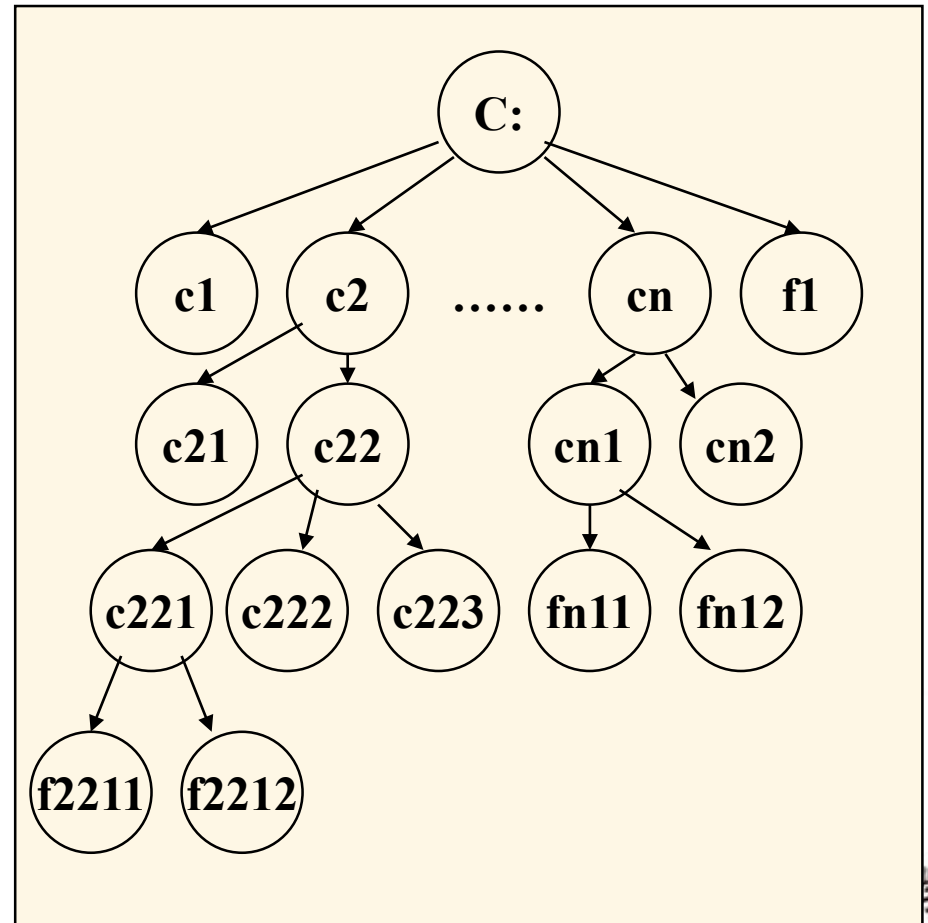


◆ 树型关系的表示

$D = \{ C, c1, c2, \dots, cn, f1, c21, c22, cn1, cn2, c2221, c222, c222, c223, fn11, fn12, f2211, f2212 \}$

$S = \{ R \}$

$R = \{ \langle C, c1 \rangle, \langle C, c2 \rangle, \dots, \langle C, cn \rangle, \langle C, f1 \rangle, \dots \}$

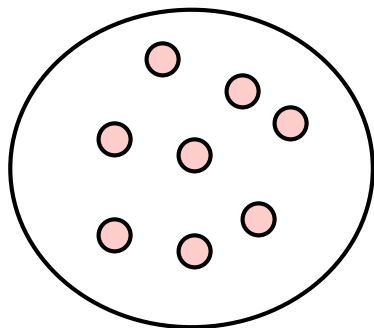


数据的逻辑结构

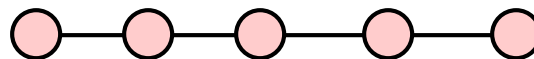
◆ 数据的逻辑结构

—— 描述数据间的逻辑关系

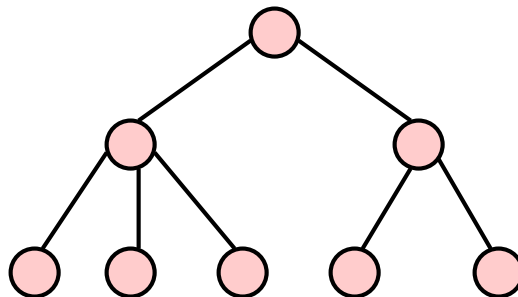
◆ 数据的逻辑结构可归结为以下四类:



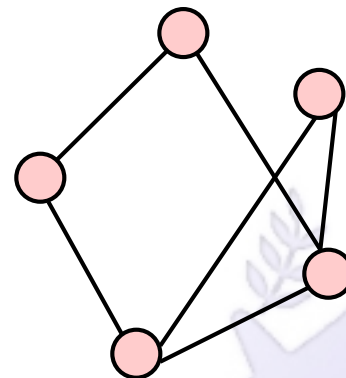
a. 集合关系



b. 线性关系



c. 树型关系



d. 图型关系



小结：数据的逻辑结构

- ◆ **数据的逻辑结构**抛弃了数据元素及其关系的实现细节，只反映了系统的需求（从用户使用层面上）而不考虑如何实现。
 - ┑ 数据的逻辑结构从逻辑关系上描述数据，**与数据存储无关**
 - ┑ 数据的逻辑结构**与数据元素的内容**无关。
 - ┑ 数据的逻辑结构与它所包含的**数据元素个数**无关。
 - ┑ 数据的逻辑结构**与数据元素所处位置**无关。





数据的存储结构

- ◆ 数据的存储结构
—— 逻辑结构在存储器中的映象
- ◆ 存储结构包含两个方面：
 - ┆ “数据元素”的映象
 - ┆ “关系”的映象



数据的存储结构—数据元素的映象

◆ 用二进制位(bit)的位串表示数据元素

¶ $(321)_{10} = (101000001)_2$

¶ $A = (001000001)_2$

¶ 图像

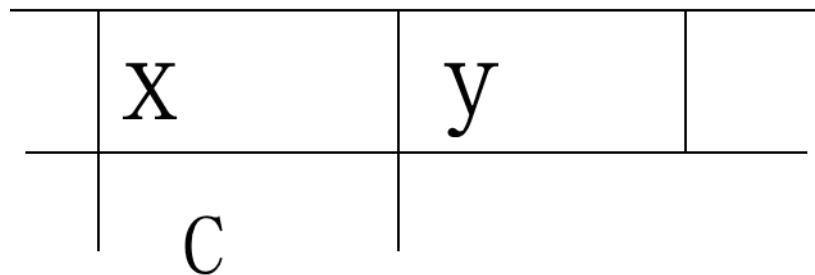
¶ 图形

¶ 声音



数据的存储结构—关系的映象

- ◆ 即如何表示两个元素之间的关系 $\langle x, y \rangle$
- ◆ 方法一：顺序映象（顺序存储结构）
 - ‖ 用数据元素在存储器中的相对位置来表示数据元素之间的逻辑关系
- ◆ 例如：令 y 的存储位置和 x 的存储位置之间差一个常量 C

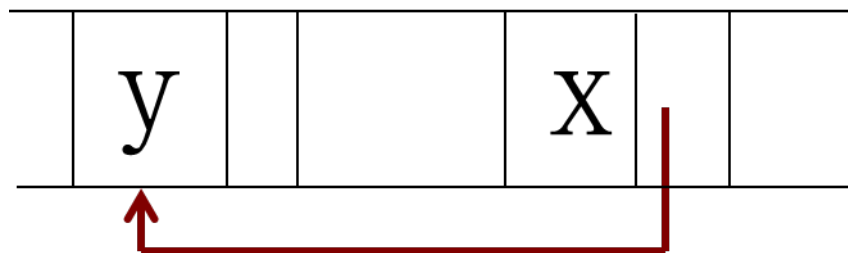


数据的存储结构—关系的映象

◆ 方法二：链式映象（链式存储结构）

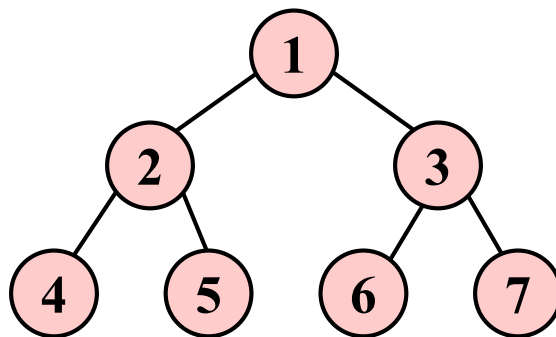
‖ 以附加信息(指针)表示数据关系

‖ 需要用一個和 x 在一起的附加信息指示 y 的存储位置

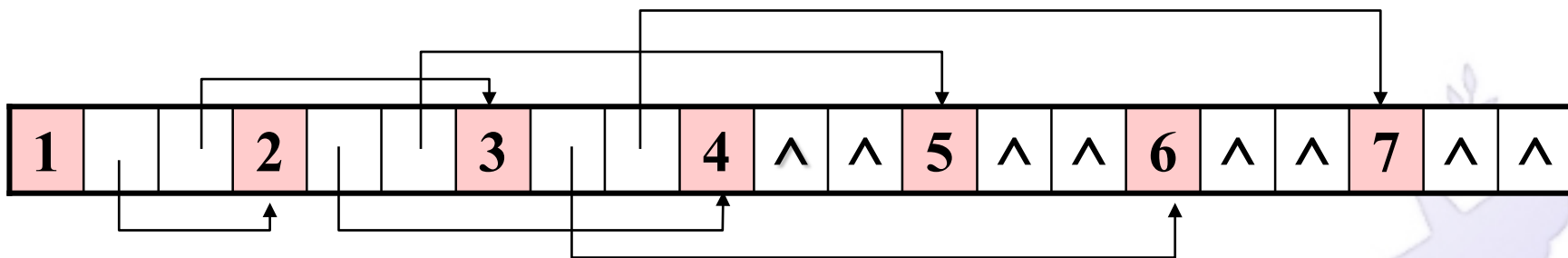


数据的存储结构—关系的映象

◆ 树型结构的链式存储结构



树型结构





数据的存储结构一

- ◆ 在不同的编程环境中，存储结构可有**不同的**描述方法。
- ◆ 当用高级程序设计语言进行编程时，通常可用高级编程语言中提供的**数据类型**描述之。
- ◆ 例如：以三个带有次序关系的整数表示一个长整数时，可利用 C 语言中提供的整数数组类型。
 - ¶ 定义长整数为：
 - ¶ `typedef int Long_int [3]`



1.2.2 数据类型

- ◆ **数据类型**：在一种程序设计语言中，一组性质相同的数值的集合，以及定义于这个数值集合上的一组操作的总称。
- ◆ 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、双精度、逻辑型、字符型
 - ‖ 构造类型：数组、**结构**、联合、**指针**、枚举型、自定义



- ◆ 不同类型的变量，其取值范围不同，所能进行的操作不同
- ◆ 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。
- ◆ 如何处理不同语言中的数据类型呢？



1.2.3 抽象数据类型[严]

◆ 抽象数据类型（Abstract Data Type 简称ADT）

ADT是对数据结构一种更准确的**抽象描述**，它忽略了数据结构的具体实现步骤，将更多的注意力放在数据的**基本操作**上，通过基本操作描述数据的**逻辑关系**。



抽象数据类型的定义格式

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

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〉



抽象数据类型的定义格式

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〉

◆ 参数表

‖ 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，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。

ADT举例

◆ 例如，抽象数据类型“复数”的定义：

ADT Complex {

数据对象: $D = \{e1, e2 \mid e1, e2 \in \text{RealSet} \}$

数据关系: $R1 = \{ \langle e1, e2 \rangle \mid e1 \text{ 是复数的实数部分} \\ \mid e2 \text{ 是复数的虚数部分} \}$

基本操作:

AssignComplex(&Z, v1, v2)

操作结果：构造复数 Z，其实部和虚部分别被赋以参数 v1 和 v2 的值

DestroyComplex(&Z)

操作结果：复数 Z 被销毁。

ADT举例（续）

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

ADT 的特征

◆ 特征一：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）。

◆ 特征二：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。



ADT 的实现

- ◆ ADT需要通过**固有数据类型**(高级编程语言中已实现的数据类型)来实现
- ◆ 例如对上述复数ADT的实现:

//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,其实部和虚部分别被赋以参数

// realval 和 imagval 的值

```
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 );
```

// 以 sum 返回两个复数 z1, z2 的和

ADT 的实现（续）

//// ----基本操作的实现

```
void add( complex z1, complex z2, complex &sum ) {
```

```
// 以 sum 返回两个复数 z1, z2 的和
```

```
    sum.realpart = z1.realpart + z2.realpart;
```

```
    sum.imagpart = z1.imagpart + z2.imagpart;
```

```
}
```



预定义常量和类型

◆ 函数结果状态代码

‖ **#define TRUE 1**

‖ **#define FALSE 0**

‖ **#define OK 1**

‖ **#define ERROR 0**

‖ **#define INFEASIBLE -1**

‖ **#define OVERFLOW -2**

◆ Status 函数返回类型

‖ **typedef int Status;**

◆ 注：课程中使用类C语言语法，请认真看课本1.2节中的定义



1.3 算法和算法设计

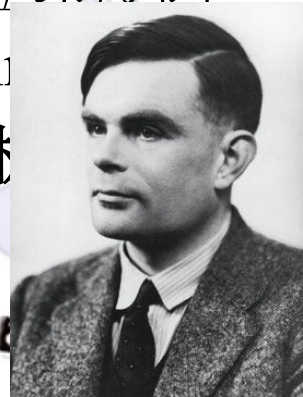
- ◆ 1.3.1 算法及其特征
- ◆ 1.3.2 算法设计的步骤
- ◆ 1.3.3 算法设计的基本方法



1.3.1 算法及其特征



- ◆ algorithm的历史
- ◆ 罗马数字：MMXXVII=2027， MCMLXXXIV=1984
- ◆ 十进制系统，由印度发明于公元600年左右
- ◆ Al-Khwarizmi（阿尔·花刺子模），约780～约850，阿拉伯数学家，代数与算术的整理者，被誉为“代数之父”。他引进了印度数字，发展了算术学。
- ◆ 后经 Fibonacci（1170～1250年）引介到欧洲，逐渐代替了欧洲原有的算板计算及罗马的记数系统。
- ◆ 欧洲人就把 Alkhwarizmi 这个字拉丁化，称用十进位印度阿拉伯数字来进行有规则可寻之计算的算术为Algorithm
- ◆ 图灵：20世纪的英国数学家。图灵机，即有限状态机





什么是算法

◆ 算法导论[C]:

- 🔧 解决可计算问题的一个工具
- 🔧 将输入转换为输出的一个可计算步骤序列

◆ 韦氏大学词典:

- 🔧 求解问题的一个过程, 步骤有限, 通常有重复操作;
广义地说, 是按部就班解决一个问题的过程.

◆ 这些都是算法的直观解释, 包含了严格定义的所有要素



1.3.1 算法

- ◆ **算法**：为了解决某类问题而规定的一个有限长的操作序列。
- ◆ **问题求解**：将问题递归性的分解为一个或者多个小问题，再综合子问题的解决方案。

```
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];
        }
```

找最大数问题

- 输入: 一个实数序列 a_1, \dots, a_n .
- 输出: $\max \{ a_1, \dots, a_n \}$.

序列	3	2	5	7	2	9	1	8
最大	3	3	5	7	7	9	9	9
修改次数	1	0	1	1	0	1	0	0
比较次数	0	1	1	1	1	1	1	1

- 输入样例 $I = \{3, 2, 5, 7, 2, 9, 1, 8\}$
- 约定规模为序列中数的个数: $N = 8$





描述算法的工具

- ◆ **自然语言**：易理解，但不精确，易有二义性
- ◆ 约定的符号描述：
 - ‖ 流程图（直观清晰并且比较精确，但是不容易实现）
 - ‖ **伪代码**（较严谨且简洁，易用程序实现）
- ◆ 计算机高级语言描述：
 - ‖ 最终形式：**C、Pascal、C++、Java...**

真正的程序员

如何测试一个人是真正的程序员？告诉他去蔬果店买一个西瓜，如果看见西红柿就买两个。

要是他买回一个西瓜和两个西红柿就不是真正的程序员。
要是他买回两个西瓜就是真正的程序员。





1.3.1 算法及其特征

◆ 一个算法必须满足以下五个重要特性：

- ¶ 1. 输入（Inputs）
- ¶ 2. 输出（Outputs）
- ¶ 3. 有穷性（finiteness）
- ¶ 4. 确定性（determinative）
- ¶ 5. 可行性（feasibility）



1.3.1 算法及其特征

◆ 1. 有穷性（ finiteness ）：

- ‖ 在执行有穷步骤之后一定能结束
- ‖ 每个步骤都能在有限时间内完成

◆ 2. 确定性（ determinative ）：

- ‖ 算法中每一条指令必须有确切的含义，不存在二义性。
- ‖ 算法只有一个入口和确定性的出口。
- ‖ 并且在任何条件下，算法都只有一条执行路径

◆ 3. 可行性（ feasibility ）：

- ‖ 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。
- ‖ 算法的时间复杂度和空间复杂度适合应用环境。



1.3.2 算法的评价标准

◆ 评价一个好的算法有以下几个标准：

1. **正确性(Correctness)**：算法应满足具体问题的需求。
2. **可读性(Readability)**：算法应该好读。有利于阅读者对程序的理解。
3. **健壮性(Robustness)**：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
4. **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





1.3.2 算法的评价标准

- ◆ 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。



1.3.3 算法设计的基本方法（算法策略）

- ◆ 穷举法[逐一检查所有可能的解，直到找到解]
- ◆ 迭代法[逐步逼近可能的解]
- ◆ 贪心法[分步完成，局部最优得到整体最优]
- ◆ 回溯法[彻底搜索，深度优先试探求得，剪枝]
- ◆ 分支界限法[彻底搜索，广度优先试探求得]
- ◆ 分治法[问题规模缩小，分而治之。如折半查找等]
- ◆ 动态规划法[问题分解（缩小规模），得到各个分解结果，再自底往上求最后结果]
- ◆ ...
- ◆ A*算法
- ◆ 遗传算法、进化计算
- ◆ 蚁群算法、粒子群算法
- ◆ 支持向量机.....





1.4 算法分析和度量

- ◆ 1.4.1 算法效率的衡量方法和准则
- ◆ 1.4.2 算法的存储空间需求





1.4.1 算法效率的衡量方法和准则

◆ 和算法执行时间相关的因素：

- ¶ 1. 计算机执行指令的速度
- ¶ 2. 编译程序产生的机器代码的质量
- ¶ 3. 编写程序的语言
- ¶ 4. 问题的规模
- ¶ 5. 算法选用的策略

◆ 一个特定算法的“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），即它是问题规模的函数。





- ◆ 通常有两种衡量算法效率的方法:
- ◆ 方法一: 事后统计法
 - 🔧 缺点1: 必须执行程序
 - 🔧 缺点2: 其它因素掩盖算法本质
 - 🔧 优点: 效率举证
- ◆ 方法二: 事前分析估算法
 - 🔧 算法分析感兴趣的不是具体的资源占用量, 而是与具体的平台无关、具体的输入实例无关, 且随输入规模增长的值是可预测的。



事前分析估算法

◆ 算法时间复杂度

与问题规模之间的关系，用一定“规模”的数据作为输入时程序运行所需的“基本操作”数来描述时间效率。

- ▶ 数据规模： n ，
- ▶ 运行时间： t 。

说明：完成一个“基本操作”所需的时间应该与具体的被操作的数无关

```
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];
        }
```

找最大数问题

- 输入: 一个实数序列 a_1, \dots, a_n .
- 输出: $\max \{ a_1, \dots, a_n \}$.

序列	3	2	5	7	2	9	1	8
最大	3	3	5	7	7	9	9	9
修改次数	1	0	1	1	0	1	0	0
比较次数	0	1	1	1	1	1	1	1

- 约定规模为序列中数的个数: $N = 8$
- 以比较次数计算时间复杂度比较合理
- 比较次数: $T(N) = N-1$





◆ 算法时间复杂度与问题的规模之间的关系

¶ 线性关系: $t_1 = cn$, 若 $n_2 = 2n$

¶ $\rightarrow t_2 = 2t_1;$

¶ 平方关系: $t_1 = n^2$, 若 $n_2 = 2n$

¶ $\rightarrow t_2 = (2n)^2 = 4t_1;$

¶ 立方关系: $t_1 = n^3$, 若 $n_2 = 2n$

¶ $\rightarrow t_2 = (2n)^3 = 8t_1;$

¶ 对数关系: $t_1 = \log_2 n$ 。若 $n_2 = 2n$

¶ $\rightarrow t_2 = \log_2 (2n) = \log_2 n + 1 = t_1 + 1$

¶ 指数关系: $t_1 = 2^n$ 。若 $n_2 = n+1$

¶ $\rightarrow t_2 = 2^{(n+1)} = 2 * 2^n = 2t_1.$



算法的渐进分析 (asymptotic analysis)

◆ 简称算法分析

- 一般这种函数关系都相当复杂，计算时只考虑可以显著影响函数量级的部分，即结果为原函数的一个近似值
- 这是对资源开销的一种不精确估计，提供对于算法资源开销进行评估的简单化模型。
- 算法的渐进分析就是要估计，当数据规模 n 逐步增大时，资源开销 $f(n)$ 的增长趋势

$$f(n) = n^2 + 100n + \log_{10} n + 1000$$



算法渐进分析：大O表式法

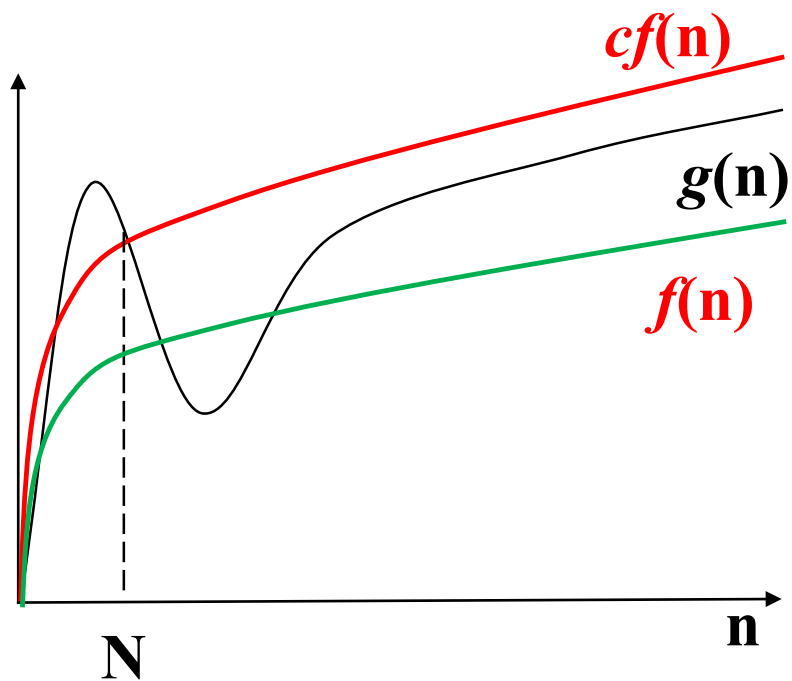
- ◆ 假设 g 和 f 为从自然数到非负实数集的两个函数
- ◆ **定义1**：如果存在正数 c 和 N ，使得对任意的 $n \geq N$ ，都有

$$g(n) \leq cf(n),$$

则称 $g(n)$ 在集合 $O(f(n))$ 中，或简称 $g(n)$ 是 $O(f(n))$ 的。



算法渐进分析：大O表式法

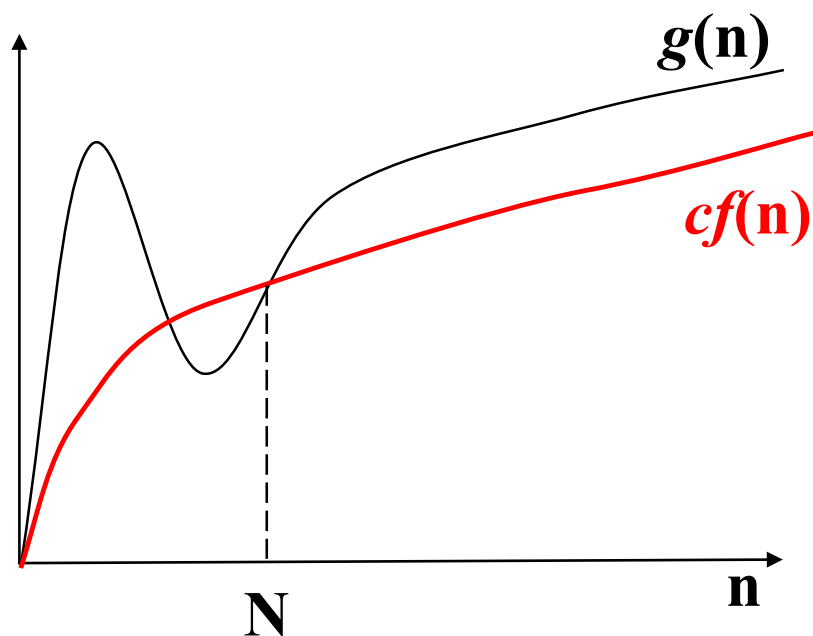


- ◆ 函数 $f(n)$ 是函数 $g(n)$ 取值的上限（upper bound），或说函数 g 的增长最终至多趋同于 f 的增长。
- ◆ 因此，大O表示法提供了一种表达函数增长率上限的方法

$g(n)$ 是 $O(f(n))$ 的， $f(n)$ 是 $g(n)$ 的紧密上限

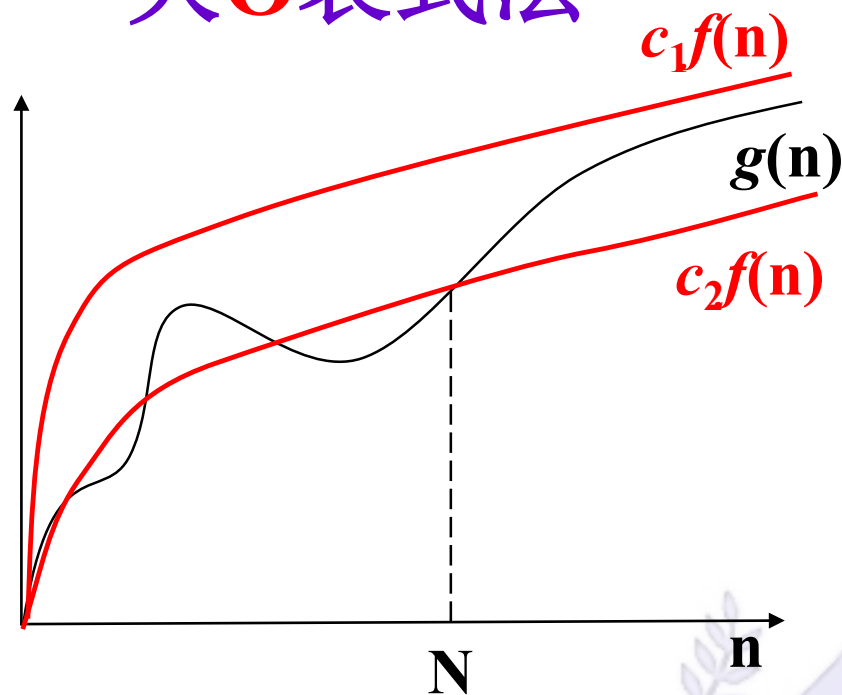


大 Ω 表式法



$g(n)$ 是 $\Omega(f(n))$ 的
 $f(n)$ 是 $g(n)$ 的紧密下限

大 Θ 表式法



$g(n)$ 是 $\Theta(f(n))$ 的
 $f(n)$ 是 $g(n)$ 的紧确界

算法渐进分析：大O表式法

- ◆ 随着问题规模 n 的增长，算法执行时间 $T(n)$ 的增长率和 $f(n)$ 的增长率相同，则可记作

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

称作算法的渐近时间复杂度(Asymptotic time complexity)。

$$\lim_{n \rightarrow \infty} \frac{T(n)}{f(n)} = C$$

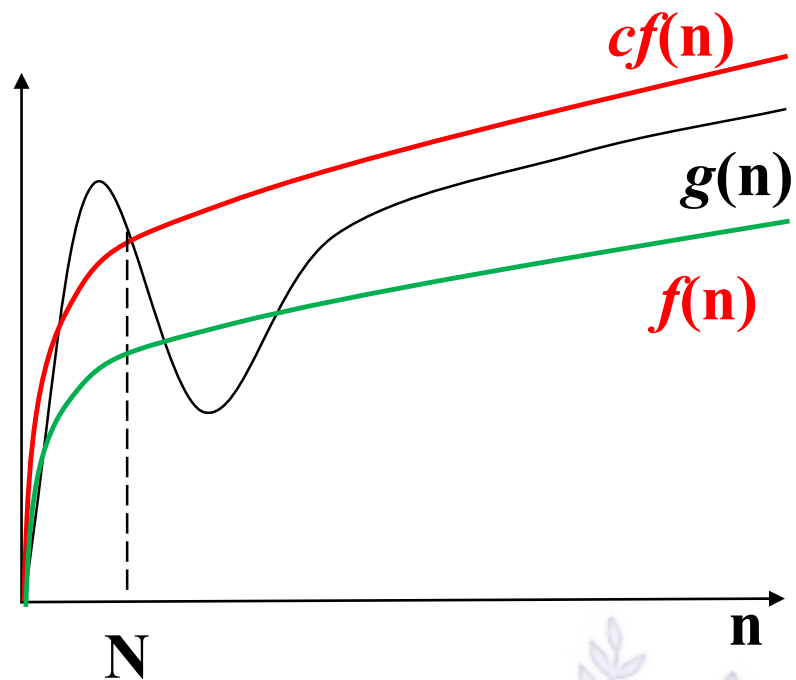
存在一个正常数 C 和 n_0 ，使得

$$T(n) \leq c \cdot f(n) \text{ for all } n \geq n_0.$$

称 $f(n)$ 是 $T(n)$ 的渐进紧密上限 (asymptotic tight upper bound)

如何估计算法的时间复杂度？

- ◆ $t = 0.5n^2 - 3n$
- ◆ 问 $T(n) = O(f(n)) = ?$
- ◆ 解: $f(n) = n^2?$
- ◆ $T(n) \leq cf(n)$
- ◆ $0.5n^2 - 3n \leq cn^2$.
- ◆ $0.5 - 3/n \leq c$
- ◆ 则选择常量 $c \geq 0.5$, $n \geq 1$ 时成立
- ◆ 所以 $T(n) = O(n^2)$



大 O 表式法

如何估计算法的时间复杂度？

- ◆ 例1、矩阵乘法
- ◆ **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
- ◆ **频度**：是指该语句重复执行的次数
- ◆ 总次数为： $t = n^2 + n^3$.
- ◆ **n足够大时**， $n^2 < n^3$.
- ◆ 时间复杂度为 **$T(n) = O(n^3)$**

```
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];
    }
} //mult
```


如何估计算法的时间复杂度？

◆ $t = 20 \times \log_2 n + n + 100n \log_2 n + n^2$

◆ 问 $T(n) = O(f(n)) = ?$

n 足够大时,

$$20 \times \log_2 n < n$$

$$< 100 \times n \log_2 n$$

$$< n^2$$

$$\text{所以 } T(n) = O(n^2)$$





如何估计算法的时间复杂度？

- ◆ 算法 = 控制结构 + 原操作（固有数据类型的操作）
- ◆ 算法的执行时间 =
 $\sum \text{原操作(i)的执行次数} \times \text{原操作(i)的执行时间}$
- ◆ 算法的执行时间 与 原操作执行次数之和 成正比



如何估计算法的时间复杂度？

◆ 例 2 {++x;}

- 🔧 基本操作： ++ x，该语句频度为 1，
- 🔧 时间复杂度为 $O(1)$ ，即常量阶

◆ 例 3 : for(i=1; i<=n; ++i) { ++x; s+=x; }

- 🔧 基本操作： ++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。



如何估计算法的时间复杂度？

◆ 例 4 、 `for(i=0; i<n; i++)`
 `for(j=0; j<n; j++)`
 `{++x; s+=x;}`

¶ 语句频度为： $2n^2$

¶ 其时间复杂度为： $O(n^2)$ ，即时间复杂度为平方阶。

◆ 例 5 、 `for(i=0; i<n; i++)`
 `for(j=0; j<=i; j++)`
 `{++x; a[i, j]=x;}`

¶ 语句频度为： $2(1+2+3+\dots+n) = n(n+1)$

¶ 其时间复杂度为： $O(n^2)$ ，即时间复杂度为平方阶。



◆ 例6、`int s=1;`

`while(s<n) s = s*2;`

¶ 语句频度为: $\lfloor \log_2 n \rfloor$

¶ 其时间复杂度为: $O(\log n)$, 即时间复杂度为对数阶。

◆ 例7、`int s=1; int m = 2^n;`

`while(s<m) s++;`

¶ 语句频度为: 2^n .

¶ 其时间复杂度为: $O(2^n)$, 即时间复杂度为指数数阶。



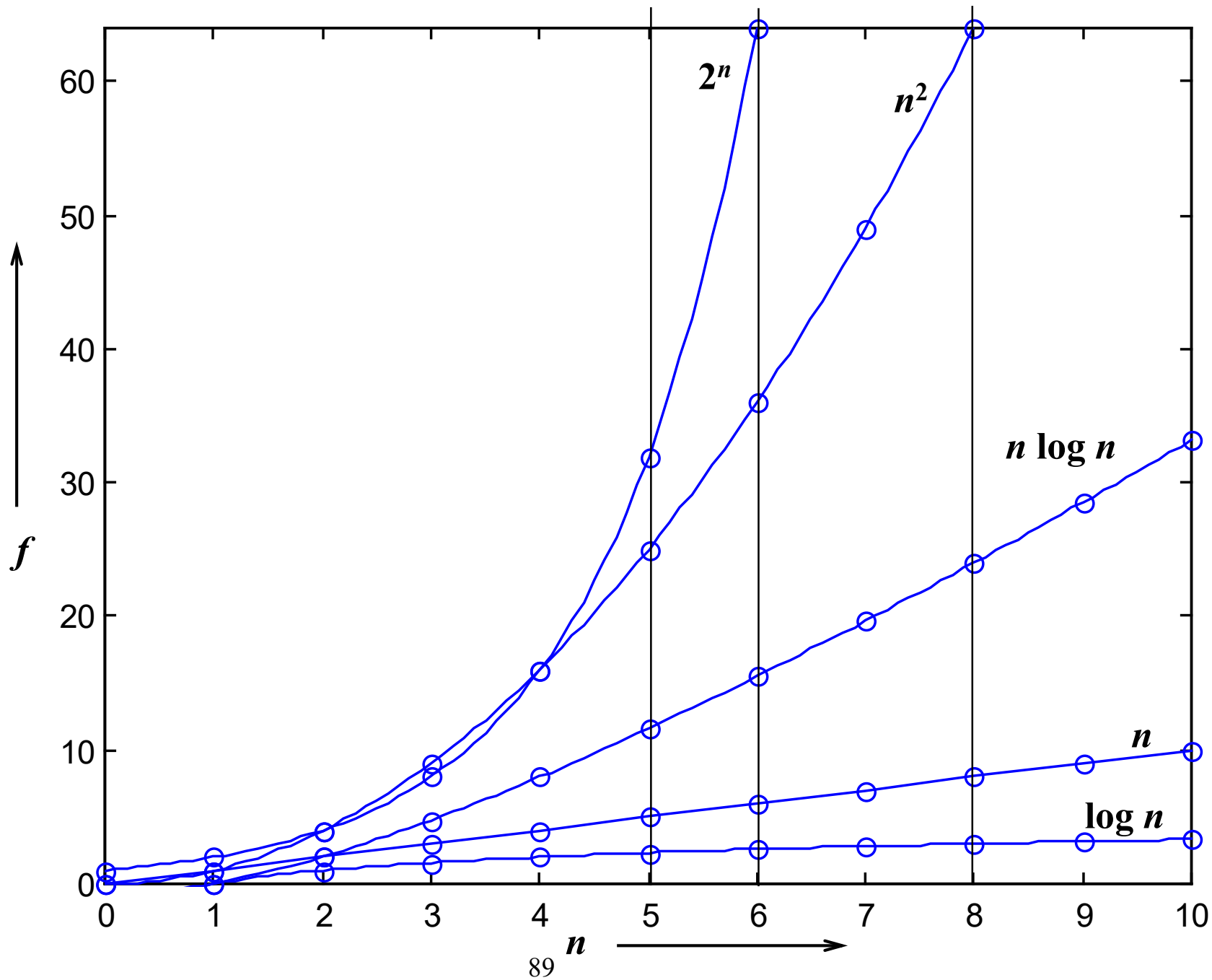


- O(1) < O(log n) < O(n) < O(n log n) < O(n²) < O(n³)**

- ¶ $O(2^n) < O(n!) < O(n^n)$

- ◆ 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。

		Input size n					
Time	Name	1	2	4	8	16	32
1	constant	1	1	1	1	1	1
$\log n$	logarithmic	0	1	2	3	4	5
n	linear	1	2	4	8	16	32
$n \log n$	log linear	0	2	8	24	64	160
n^2	quadratic	1	4	16	64	256	1024
n^3	cubic	1	8	64	512	4096	32768
2^n	exponential	2	4	16	256	65536	4294967296
$n !$	factorial	1	2	24_{88}	40326	2092278988000	26313×10^{33}



递归算法的执行次数

- ◆ 计算正数数组a中前n个元素之和的递归算法

```
float rsum(float a[], int n) { //递归程序
    if (n <= 0) return 0;
    else return rsum(a, n-1) + a[n-1];
}
```



递归算法的执行次数


- ◆ $T(n) = \begin{cases} 1, & n \leq 0 \\ 2 + T(n-1), & n > 0 \end{cases}$
- ◆ $T(n) = 2 + T(n-1)$
- ◆ $= 2 + 2 + T(n-2) = 2 * 2 + T(n-2)$
- ◆ $= 2 * 2 + 2 + T(n-3) = 2 * 3 + T(n-3) = \dots$
- ◆ $= 2 * (n-1) + T(n - (n-1)) = 2 * (n-1) + T(1)$
- ◆ $= 2n + T(0) = 2n + 1$
- ◆ 时间复杂度为 $O(n)$

```
float rsum(float a[], int n) { // 递归程序
    if (n <= 0) return 0;
    else return rsum(a, n-1) + a[n-1];
}
```

递归算法的执行次数

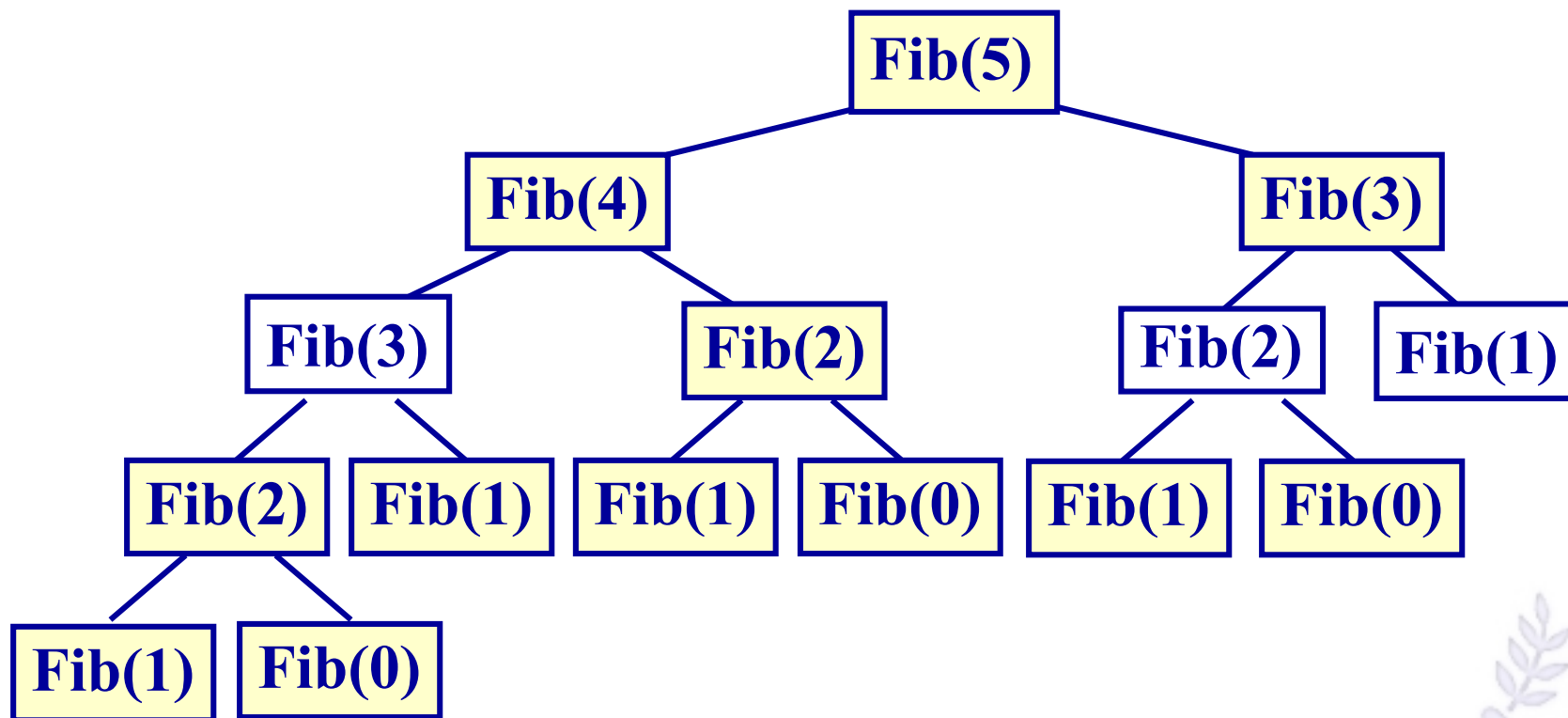
- ◆ 计算斐波那契 (Fibonacci) 数。
- ◆ 斐波那契数列：1, 1, 2, 3, 5, 8, 13, 21.....
 - ¶ $F(0)=F(1)=1$,
 - ¶ $F(n)=F(n-1)+F(n-2) \ (n \geq 2)$

```
long Fib(long n)//递归程序
{
    if (n==0 || n==1) return 1;
    else return Fib(n-1) + Fib(n-2);
}
```



```
if (n==0 || n==1) return 1;  
else return Fib(n-1) + Fib(n-2);
```

◆ 递归调用树Fib(5)



◆ 计算Fib(5)，总计算次数为树的节点总数15



递归算法的执行次数

- ◆ 假设二叉树为完全二叉树，其高度为 n ，则：
- ◆ $T(n) = 2^n - 1$
- ◆ 计算斐波那契数递归算法时间复杂度为： $O(2^n)$ 。

```
long Fib (long n) //递归程序
{
    if (n==0 || n==1) return 1;
    else return Fib(n-1) + Fib(n-2);
}
```

大O表示法的加法法则

- ◆ 如果两个并列的程序段的渐近时间复杂度分别为
- ◆ $T_1(n) = O(f(n))$, $T_2(m) = O(g(m))$,
- ◆ 那么将两个程序段连在一起后整个程序的时间复杂度为:
- ◆ $T(n, m) = T_1(n) + T_2(m) = O(\max\{f(n), g(m)\})$



大O表示法的乘法法则

- ◆ 如果嵌套的程序段的渐近时间复杂度分别为
- ◆ $T_1(n) = O(f(n))$, $T_2(m) = O(g(m))$,
- ◆ 那么将两个程序段连在一起后整个程序的时间复杂度为:
- ◆ $T(n, m) = T_1(n) \times T_2(m) = O(f(n) \times g(m))$



如何估计算法的时间复杂度？

- ◆ 有时，算法中基本操作重复执行的次数会随问题的输入数据集不同而不同
- ◆ 最好情况： $n-1$ 次
- ◆ 最坏情况： $1+2+3+\dots+n-1$
 $=n(n-1)/2$
- ◆ 平均时间复杂度为： $O(n^2)$

```
void bubble-sort (int a[], int n)
{ //起泡排序，从小到大排列
    for( i=n-1, change=TURE;
        i>1 && change; -i)
    {
        change=false;
        for( j=0; j<i; ++j)
            if ( a[j]>a[j+1]) {
                a[j]  $\longleftrightarrow$  a[j+1];
                change=TURE;
            }
    }
} //bubble-sort
```

1.3.4 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





1.3.4 算法的存储空间需求

- ◆ 若输入数据所占空间只取决于问题本身，和算法无关，则只需要分析除输入数据和程序之外的**辅助变量所占额外空间**。
- ◆ **即算法的空间复杂度是指算法所需的辅助空间度量。**
- ◆ 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**。



例1：起泡排序

```
Void bubble-sort (int a[], int n)
{ //起泡排序，从小到大排列
    for( i=n-1,change=TURE;
        i>1 && change;- i)
    {
        change=false;
        for( j=0;j<i; ++j)
            if ( a[j]>a[j+1]) {
                a[j]  $\longleftrightarrow$  a[j+1];
                change=TURE}
    }
} //bubble-sort
```

$$S(n) = O(1)$$

例2：求整数 n 的阶乘

方法一：

```
int f(unsigned int n){  
    int result = 1;  
  
    for (int i=2;i<=n;i++) {  
        result *= i;  
    }  
    return result;  
}
```

方法二：

```
int f(unsigned int n){  
    if (n==0 || n==1)  
        return 1;  
    return n*f(n-1);  
}
```





希尔伯特第十问题没有算法

- ◆ Hilbert第十问题：“多项式是否有整数根”有没有算法？
- ◆ 1970's被证明没有算法

●“算法”：对于输入 p ， p 是 k 元多项式

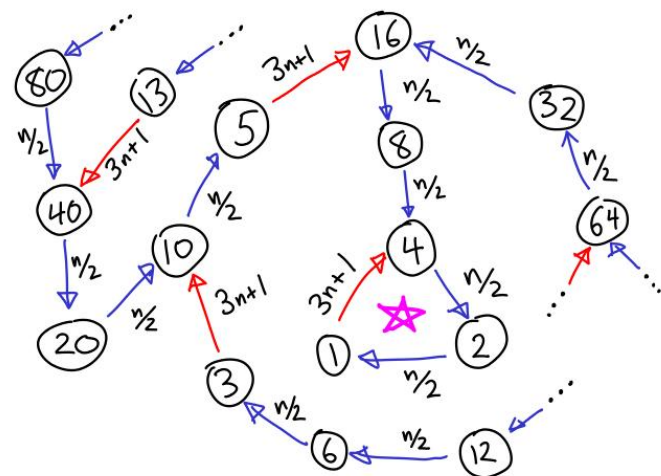
1. 取 k 个整数的向量 x （绝对值和从小到大）。
2. 若 $p(x) = 0$ ，则接受，
3. 否则转1.

$x^2 + y^2 - 3 = 0$ 死循环



3n+1问题目前不知道有没有算法

- ◆ 输入: 一个正整数 n ,
- ◆ 映射: $f(n) = n/2$, 若 n 是偶数;
 $f(n) = 3n+1$, 若 n 是奇数.
- ◆ 迭代: $5 \rightarrow 16 \rightarrow 8 \rightarrow \dots$, 到1则停止
- ◆ 输出: n 可在 f 迭代下是否能停止
- ◆ 直接模拟是正确的算法吗?
- ◆ 27需迭代111步(见右图)
- ◆ $1 \sim 5 \times 10^{18}$ 都能到1.([wiki])





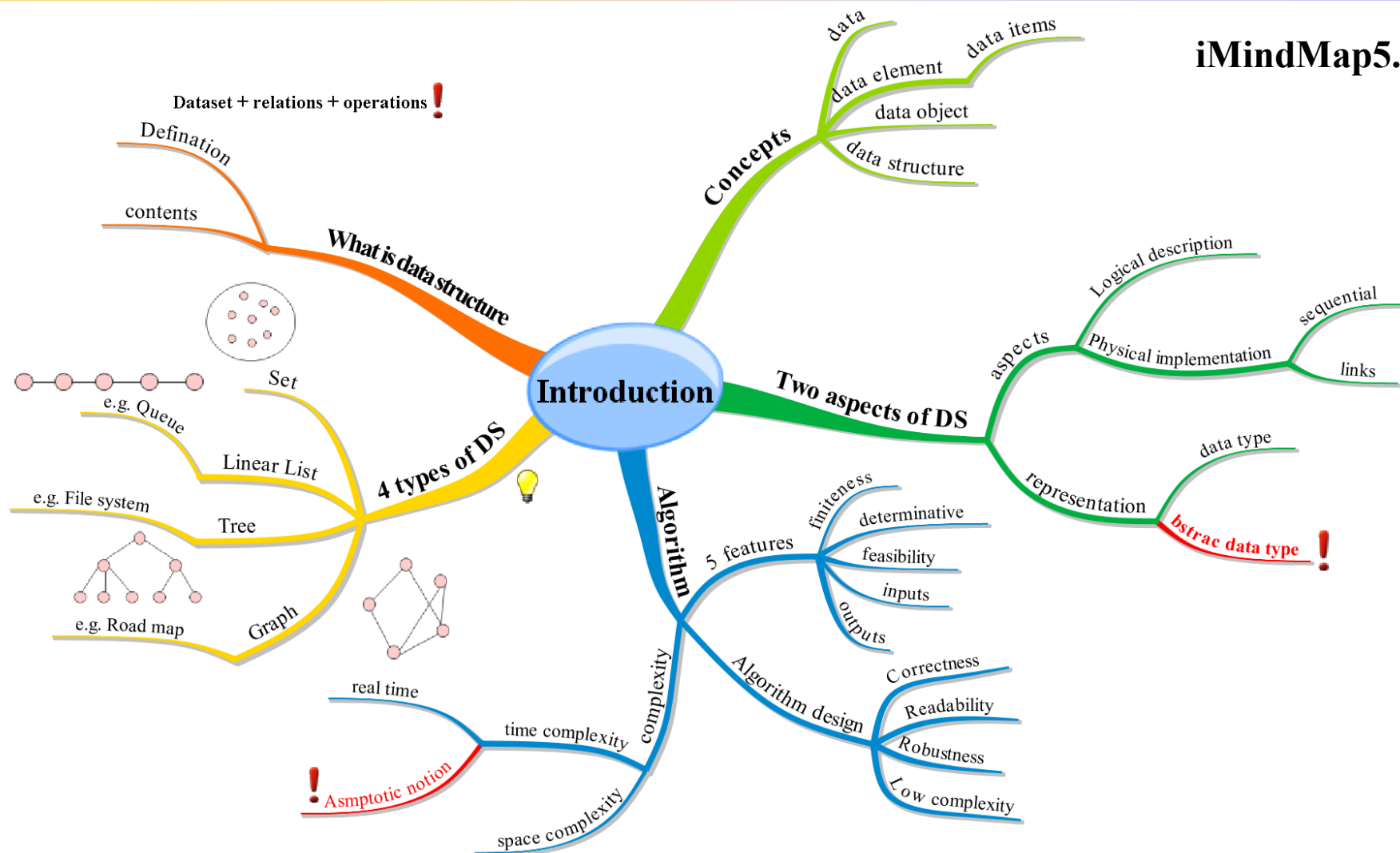
问题分类

- ◆ 什么问题被解决了? (找最大数...) 有算法
- ◆ 什么问题没有被解决? ($3n+1$ 问题) 不知道有没有算法
- ◆ 什么问题没有解决方案? (Hilbert第十问题) 没有算法



Review

iMindMap5.0





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ADT的意义。
- ◆ 3. 理解算法五个要素的确切含义。
- ◆ 4. 掌握计算语句频度和估算算法时间复杂度的方法。
- ◆ 提前复习C语言中的数组、指针、结构、内存申请和释放等内容





对学习过程的建议

- ◆ 上课认真听讲，课后看书复习，多编程多练习。
- ◆ 掌握有效的学习方法：
 - 🔑 学习**算法的基本思想**，掌握如何将现实世界中的问题进行抽象，进行逻辑结构设计，进而**设计**并描述出高效的**算法**。
 - 🔑 通过**编程实现**，将使用伪语言描述的算法变为实际的程序。
- ◆ 提前复习C语言中的数组、指针、结构等内容。



数据结构学习要求

◆ 1 理解各章基本概念

◆ 2 存储结构：

- 1) 掌握基本存储结构：表、栈、队列、二叉树（顺序结构、链式结构（动静态）存储信息、含义及C语言描述）
- 2) 理解复杂存储结构（图、树），理解图和树上的应用：例如最小生成树、最短路径等。

◆ 3 算法

- 1) 掌握基本算法（构造、销毁、插入、删除、遍历、查找、排序等）：主要步骤（或基本思想）、主要操作的实现（C语言描述），并能应用到对应问题中，能推广到相似问题中；
- 2) 复杂算法：掌握方法；
- 3) 会计算基本算法的时间复杂度和空间复杂度。



END of Chapter I

