数据结构与算法

Data Structrues and Algorithms

张岩

海量数据计算研究中心

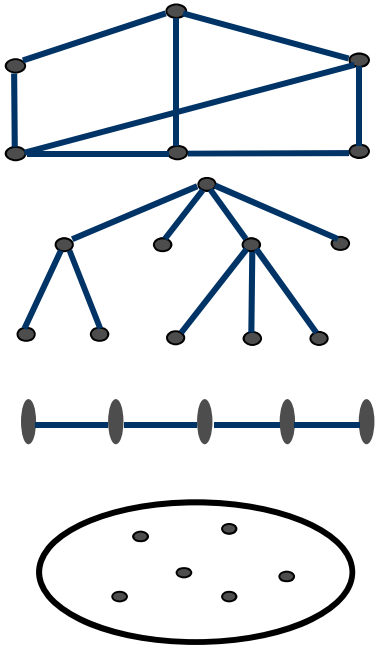
哈工大计算机科学与技术学院

课程说明

- 课程编号：**CS32131**
- 授课学时：**40**（**7**至**16**周，**4**学时/周）
- 实验学时：**16**（第**10/12/14/16**周，**3**学时/周，第**17**周，**4**学时/周）
- 课程分类：专业（技术）基础
- 答疑地点：科创大厦**K1421**，每周**1**次
- 课程资源：
  - 课件、实验和作业：**QQ**群：**894375334**
  - 实验和作业上传和下载：**10.160.3.21:8080**（ **Google**或**Firefox** ）
- 考核形式：
  - 期末笔试**70%+**平时和作业成绩**10%+**实验成绩**20%**
- 主讲教师：张岩
  - 联系方式：电话**13845139350**
  - **email: zhangy@hit.edu.cn**

课程组织与结构

- 通俗地讲，
  - 数据结构：数据及其之间的相互关系，是对数据的抽象。
  - 算法，是求解特定问题的方法，是将输入转为输出的一系列计算步骤。
- 课程的内容组织
  - 四类基本的数据结构
    - 图型结构：多对多的任意关系；
    - 树型结构：一对多的层次关系；
    - 线性结构：一对一的线性关系；
    - 集合结构：元素同属一个集合；



  - 查找技术：线性结构、树型结构和散列结构上查找；
  - 排序方法：内部排序和外部排序。

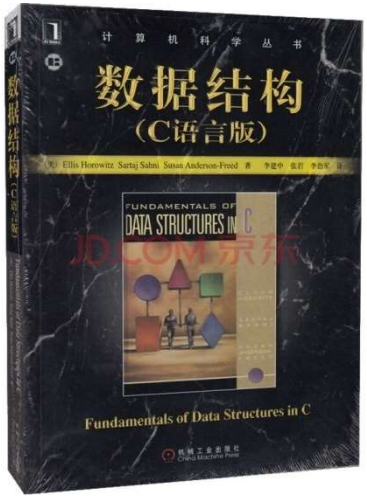
课程目标

- 象精通小学乘法口诀一样，
  - 彻底精通数据结构
  - 不能只停留在“逻辑”层面上
- 让“数据结构与算法”
  - 不再是你程序设计的障碍!
  - 不再是你学习其他课程的障碍!
  - 不再是你考研的障碍!
  - 不再是你找工作的障碍!
- 在编程中升华你的计算思维，爱上计算机专业！

教材和主要参考书

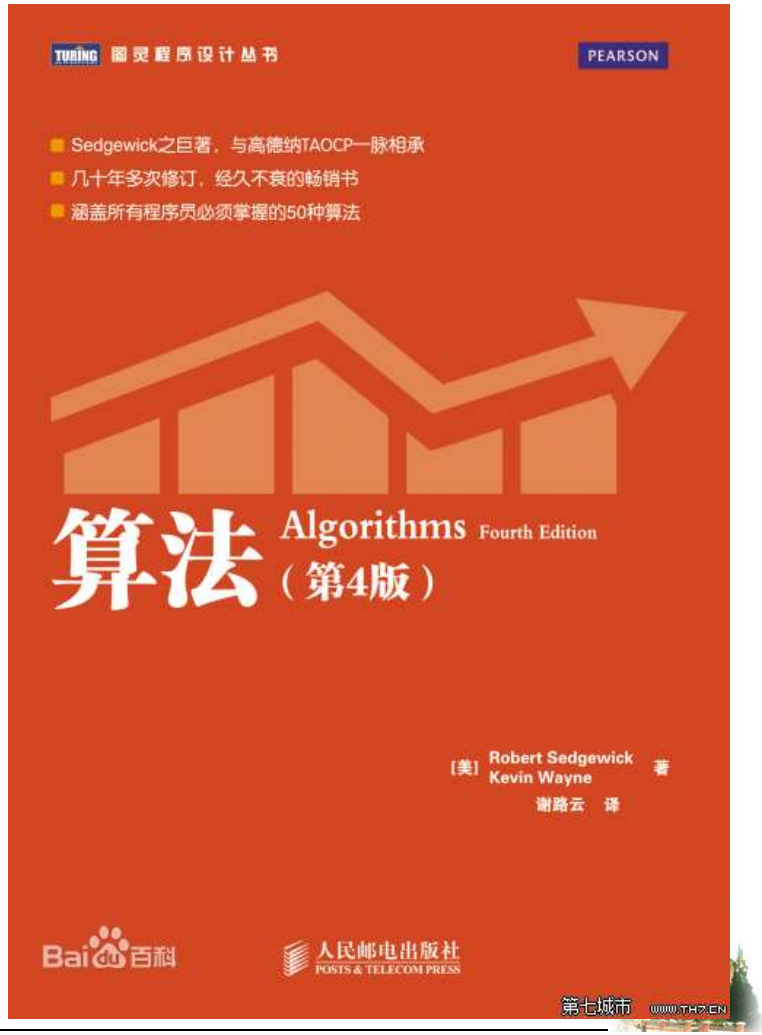
主要参考书

- 数据结构与算法(第4版). 廖明宏, 郭福顺 ,张岩,李秀坤, 高等教出版社, 2007.
- 数据结构(C语言版).严蔚敏,吴伟民, 清华大学出版社,2012.
- Fundamentals of Data Structures in C. Ellis Horawitz, Sartaj Sahni, Susan Anderson-Freed著,李建中,张岩,李治军译. 机械工业出版社.2006.
- Data Structures, Algorithms,and Applications in C++, Second Edition. Sartaj Sahni著,王立柱,刘志红译,机械工业出版社.2015.



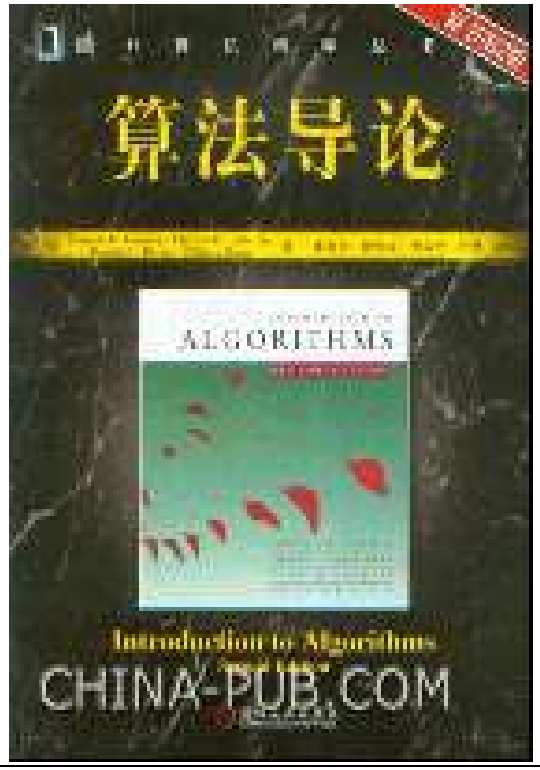
好书力荐

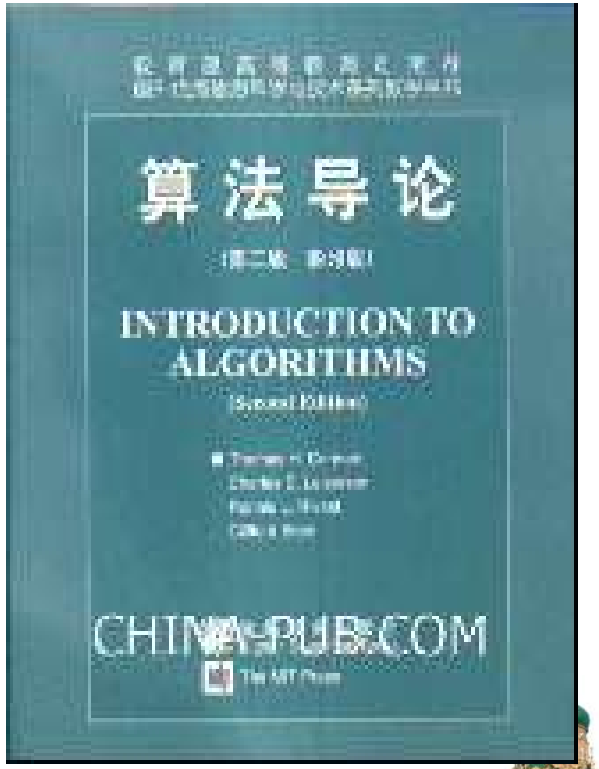
- 重要的参考书
  - **Algorithms Fourth Edition，算法(第4版)**
  - **Robert Sedgewick, Kevin Wayne**
  - **Sedgewick之巨著，与高德纳TAOCP一脉相承，经久不衰的畅销书**
  - 全面介绍了关于算法和数据结构的必备知识
  - 涵盖程序员应知应会的**50**个算法，提供实际代码
  - 与实际应用相结合；用合适的数学模型精确地讨论算法的性能



好书力荐

- 重要的参考书
  - **Introduction to Algorithms（算法导论）**
  - **Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein**
  - “五个一”：
    - 每一章，介绍
    - 一个算法；
    - 一种设计技术；
    - 一个应用领域；
    - 一个相关话题。





第1章 绪论

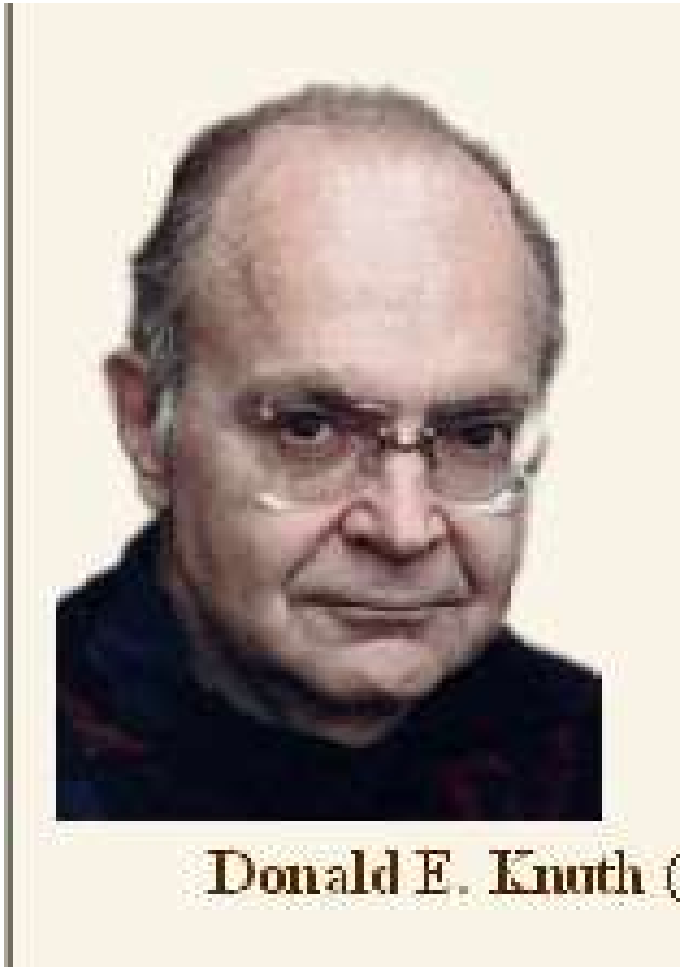
学习目标

- 了解数据结构的基本概念、研究对象以及数据结构课程的发展历史，对数据结构与算法课程有一个宏观的认识
- 掌握贯穿全书的重要概念-----抽象数据型，包括其概念的定义和实现方法，初步掌握数据抽象技术和方法
- 了解算法、算法复杂性，掌握算法性能的评价方法
- 了解解决问题的一般过程和算法的逐步求精方法，掌握问题求解的基本过程和方法

本章主要内容

- 数据结构的兴起和发展
- 基本概念与研究对象
- 抽象数据类型
- 算法及算法分析
- 逐步求精的程序设计方法
- 本章小结

数据结构的创始人—Donald. E. Knuth



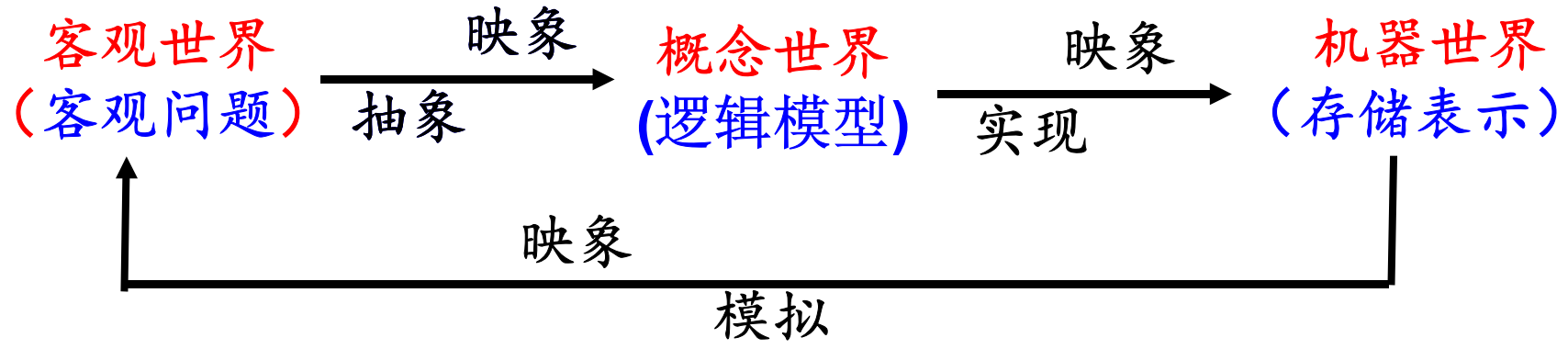
1938年出生，25岁毕业于加州理工学院数学系，博士毕业后留校任教，28岁任副教授。30岁时，加盟斯坦福大学计算机系，任教授。从31岁起，开始出版他的历史性经典巨著：

The Art of Computer Programming

他计划共写7卷，然而出版三卷之后，已震惊世界，使他获得计算机科学界的最高荣誉图灵奖，此时，他年仅36岁。

1.1 数据结构的兴起和发展

- 客观世界与计算机世界的关系
  - 计算机科学是研究信息表示和信息处理的科学
  - 信息在计算机内是用数据表示的，数据是信息的载体
  - 用计算机解决实际问题的实质可以用下图表示：



客观世界与计算机的关系

1.1 数据结构的兴起和发展(Cont.)

- 程序设计的实质是什么？
  - 数据表示：将数据存储在计算机中
  - 数据处理：处理数据，求解问题
  - 数据结构问题起源于程序设计

- 数据结构随着程序设计的发展而发展
  - 1. 无结构阶段：在简单数据上作复杂运算
  - 2. 结构化阶段：数据结构＋算法＝程序
  - 3. 面向对象阶段：（对象＋行为）＝程序
- 以数据为中心的程序设计方法学推动了数据结构的发展
- 数据结构的发展并未终结……

1.2 研究对象与基本概念 (Cont.)

数据结构的基本概念

- 数据：一切能输入到计算机中并能被计算机程序识别和处理的符号集合
  - 数值数据：整数、实数等
  - 非数值数据：图形、图象、声音、文字等
- 数据元素：数据的基本单位，在计算机程序中通常作为一个整体进行考虑和处理
- 数据项：构成数据元素的不可分割的最小单位
- 数据对象：具有相同性质的数据元素的集合
- 结点：数据元素在计算机内的位串表示
- 域（字段）：数据元素中数据项在计算机内的表示
- 信息表：是数据对象在计算机内的表示

1.2 研究对象与基本概念

- 例1 表结构—学籍管理问题
  - 完成什么功能？各表项之间是什么关系？

学生学籍表

学号	姓名	性别	出生日期	政治面貌
170300101	王晓东	男	1999/7/17	团员
170300102	李明远	男	2000/1/25	党员
170300103	张蔷薇	女	1999/9/25	团员

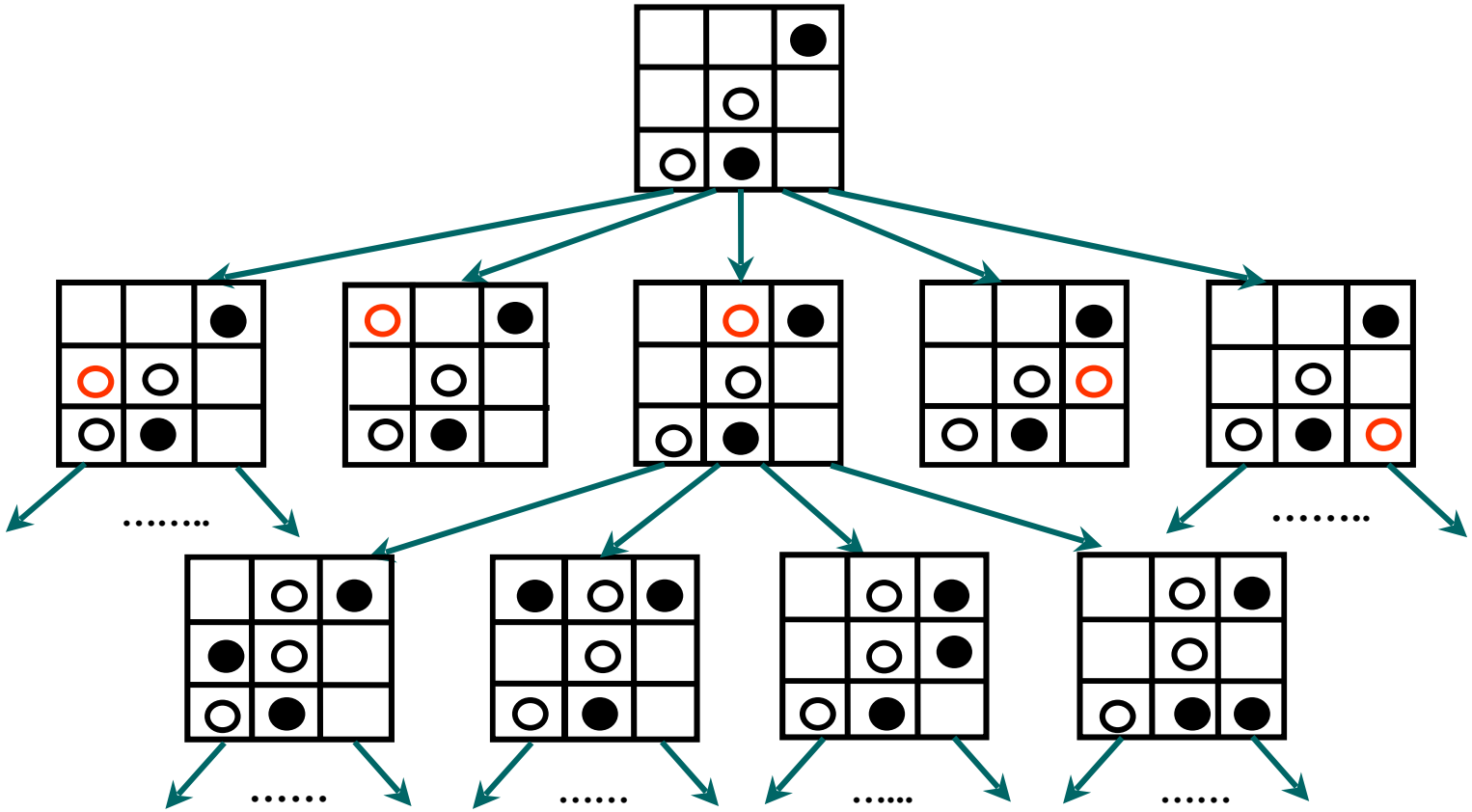
- “课表”：

计算机类第二学年秋季教学进度表（培养方案）

课程编号	课程名称	学分	学时	讲课	实验	课外辅导
MX11003	毛泽东思想和中国特色社会主义理论体系概论	4.0	64	48		16
PE13003	体育	0.5	16	16		
FL12003	大学外语	1.5	36	32		4
MA21016	概率论与数理统计B	3.5	56	56		
PH21013	大学物理实验B	1.0	24	3	21	
CS31112	数理逻辑	2.0	32	32		
CS32122	计算机系统	5.0	80	56		24
CS32131	数据结构与算法	3.5	56	40		16
CS32131E	数据结构与算法	3.5	56	40		16

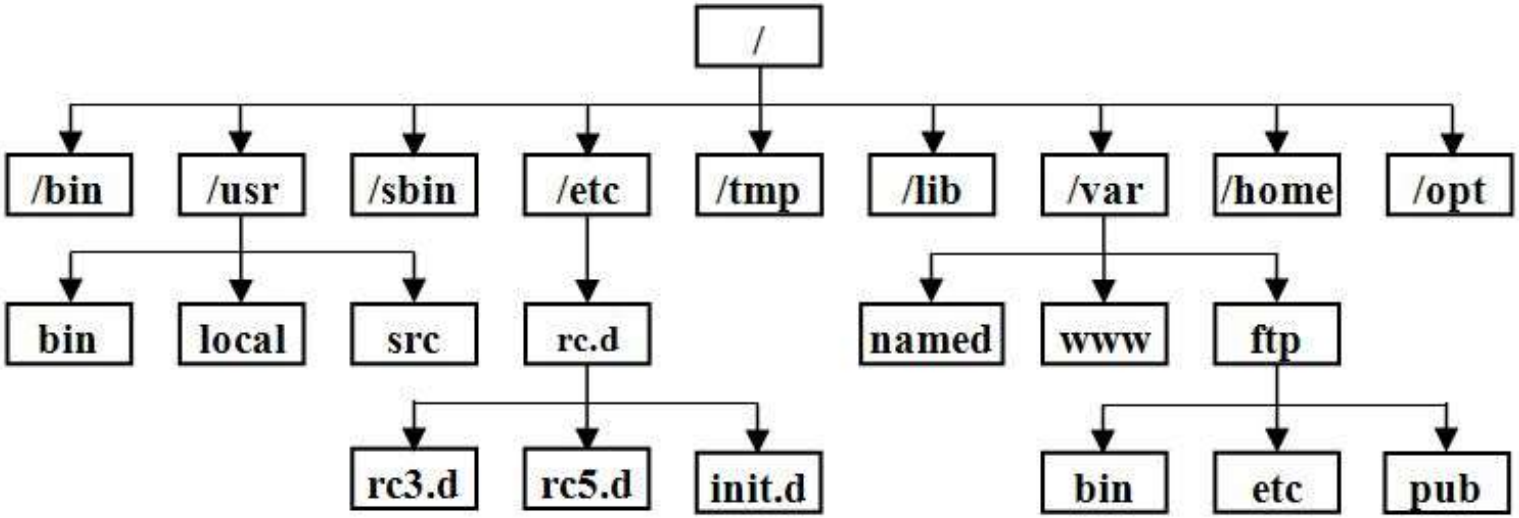
1.2 研究对象与基本概念(Cont.)

- 例2 树结构—人机对弈问题
  - 如何实现对弈？各格局之间是什么关系？



1.2 研究对象与基本概念(Cont.)

- 树结构—文件系统
  - 如何实现文件管理？各文件(目录)之间是什么关系？



- 其他树结构实例
  - 家谱
  - 组织机构
  - ……

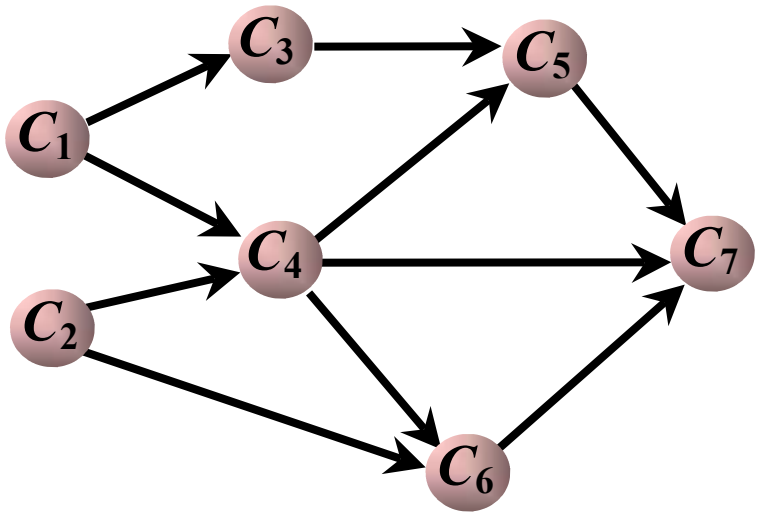
1.2 研究对象与基本概念(Cont.)

例3 图结构—教学计划编排问题

- 如何表示课程之间的先修关系？如何安排课表？

课程及其关系

课程编号	课程名称	先修课程
C1	工科数学分析	无
C2	计算机导论	无
C3	集合论与图论	C1
C4	程序设计	C1，C2
C5	数据结构与算法	C3，C4
C6	计算机组成原理	C2，C4
C7	数据库系统原理	C4，C5，C6



其他图结构实例

- 交通导航
- 社交网络
- ……

1.2 研究对象与基本概念(Cont.)

- 计算机求解问题
  - 问题→抽象出问题的模型→求模型的解
  - 问题——数值问题、非数值问题
    - 数 值 问 题→数学方程
    - 非数值问题→数据结构

- 数据结构与算法课程的研究对象
  - 是研究非数值计算问题中计算机的操作对象以及它们之间的关系和操作的学科。

1.2 研究对象与基本概念(Cont.)

数据结构

- 数据元素及其相互之间的相互关系，这种关系是抽象的，即并不涉及数据元素的具体内容。是数据元素及其相互间的关系的数学描述。
- 相互之间存在一定关系的数据元素的集合。
- 按照视点的不同，数据结构分为逻辑结构和存储结构。

数据的逻辑结构

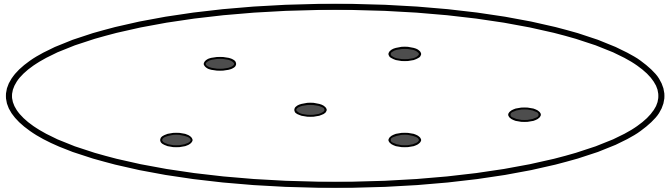
- 数据元素之间的抽象关系，数据元素之间逻辑关系的整体
  - 学籍管理问题中，表项之间的逻辑关系指的是什么？
  - 人机对弈问题中，格局之间的逻辑关系指的是什么？
  - 教学计划编排问题中，课程之间的逻辑关系指的是什么？
- 数据的逻辑结构是从具体问题抽象出来的数据模型。

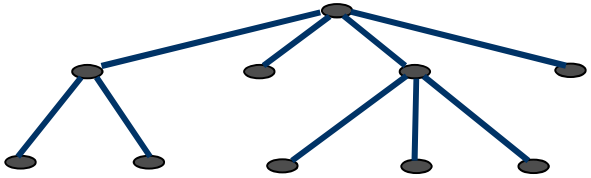
1.2 研究对象与基本概念(Cont.)

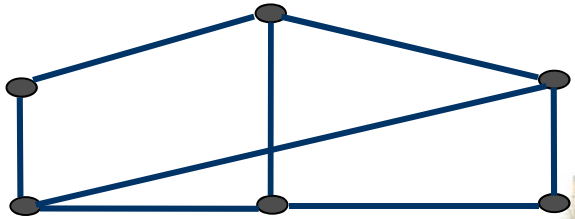
- 数据结构从逻辑上分为四类：

  即四种基本的逻辑结构

  - 集合：数据元素之间就是“属于同一个集合”；
  - 数据线性结构：数据元素之间存在着一对一的线性关系；
  - 树型结构：数据元素之间存在着一对多的层次关系；
  - 图型结构：数据元素之间存在着多对多的任意关系。



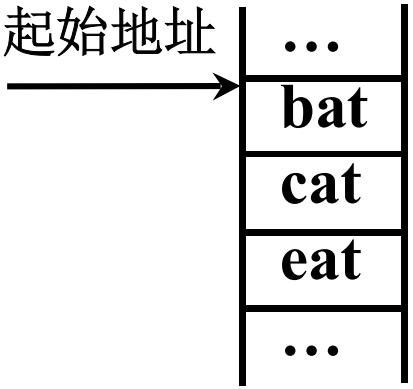




1.2 研究对象与基本概念(Cont.)

- 数据的存储结构
  - 又称物理结构，是数据及其逻辑结构在计算机中的表示。
  - 实质上是内存分配，以确定元素及元素之间关系的表示。
  - 在具体实现时，依赖于计算机语言。
- 两种基本的存储结构
  - 顺序存储结构
    - 用一组连续的存储单元依次存储数据元素，数据元素之间的逻辑关系由元素的存储位置来表示。

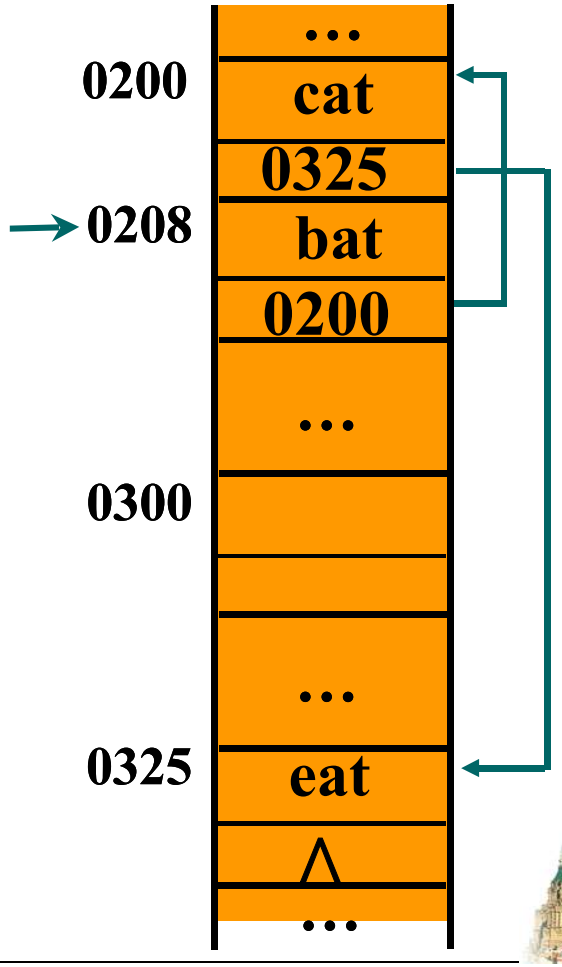
例：(bat, cat, eat)



1.2 研究对象与基本概念(Cont.)

- 两种基本的存储结构
  - 顺序存储结构
    - 用一组连续的存储单元依次存储数据元素，数据元素之间的逻辑关系由元素的存储位置来表示。
  - 链接存储结构
    - 用一组任意的存储单元存储数据元素，数据元素之间的逻辑关系用指针来表示 。

例：(bat, cat, eat)



1.2 研究对象与基本概念(Cont.)

- 数据的运算
  - 是定义在逻辑结构上的一组抽象操作（只知道做什么）
  - 只有在存储结构确定之后，才考虑如何实现这些操作或运算（如何做）
- 逻辑结构与存储结构的关系
  - 数据的逻辑结构属于用户视图，是面向问题的，反映了数据内部的构成方式；数据的存储结构属于具体实现视图，是面向计算机的。
  - 一种数据（的逻辑）结构可以用多种存储结构来表示，而采用不同的存储结构，其数据处理的效率往往是不同的。
  - 任何一个算法的设计取决于选定的数据（逻辑）结构，而算法的实现依赖于采用的存储结构。

1.2 研究对象与基本概念(Cont.)

- 数据结构与算法的学习内容
  - 数据对象的结构形式,各种数据结构的性质(逻辑结构);
  - 数据对象和“关系”在计算机中的表示(物理结构/存储结构);
  - 数据结构上定义的基本操作(算法)及其实现;
  - 算法的效率(时间和空间);
  - 数据结构的应用，如数据分类，检索等。

课程内容的组织体系

层次＼方面	数据表示	数据处理
抽象	逻辑结构	基本操作
实现	存储结构	算法实现
评价	不同数据结构的比较及算法的分析	

1.2 研究对象与基本概念(Cont.)

- 课程的研究对象，即我们要学习内容

五个方面

- 数据的逻辑结构
  - 线性结构
    - 线性表
    - 栈
    - 队列
    - ……
  - 非线性结构
    - 集合结构
    - 树型结构
    - 图型结构
- 数据的存储结构
  - 顺序存储
  - 链式存储
  - 索引存储
  - 散列存储
- 数据的操作(算法)及实现：检索、排序、插入、删除等
- 操作算法的性能：时间和空间效率—时间和空间复杂度
- 应用

1.3 抽象数据类型

- 抽象数据类型（Abstract Data Type）
  - 定义：一个数学模型和在该模型上定义的操作集合的总称
    - ADT是程序设计语言中数据类型概念的进一步推广和进一步抽象。
    - 例4：ADT int = ({ x|x∈Z }，{ + , − , * , / , % , ≤ , == })
    - 同一数学模型上定义不同的操作集，则它们代表不同的ADT。
  - 实现：用适当的数据结构来表示ADT中的数学模型，并用一组函数（方法）来实现该模型上的各种操作。
  - 定义独立于实现
    - 定义给出一个ADT客观存在的逻辑特征
    - 不必考虑如何在计算机中实现

1.3 抽象数据类型(Cont.)

- 数据类型、数据结构和ADT
  - 各自含义：
    - 数据类型是一组值的集合；
    - 数据结构则是数据元素之间的抽象关系；
    - 抽象数据型是一个数学模型及在该模型上定义的操作集的总称。
  - 相互关系
    - 数据型是根据数据结构分类的,同类型的数据元素的数据结构相同。
    - 数据结构则是抽象数据型中数学模型的表示;
    - **ADT**是数据类型的进一步推广和进一步抽象。

1.4 算法及算法分析

算法的概念

- 算法（**Algorithm**）:是对特定问题求解步骤的一种描述，是指令的有限序列。
- 算法的五大特性：
  - 输入：一个算法有零个或多个输入。
  - 输出：一个算法有一个或多个输出。
  - 有穷性：一个算法必须总是在执行有穷步之后结束，且每一步都在有穷时间内完成。
  - 确定性：算法中的每一条指令必须有确切的含义，对于相同的输入只能得到相同的输出。
  - 可行性：算法描述的操作可以通过已经实现的基本操作执行有限次来实现。

1.4 算法及算法分析(Cont.)

算法设计的要求

- 正确性
  - 对几组→精选苛刻→所有合法输入都能产生满足要求结果
- 可读性
  - 好算法首先便于理解和交流，其次才是机器执行
  - 坏算法晦涩难懂，易于隐藏错误且难于调试和修改
- 健壮性（鲁棒性）
  - 对非法数据的抵抗能力
  - 对非法数据能识别和处理，避免产生错误动作或陷入瘫痪
- 高效率和低存储量需求
  - 对于一个具体问题，通常有多个算法
  - 指行时间短的算法的效率就高
  - 存储量需求，是指算法执行过程中所需的最大存储空间
  - 两者都与问题规模有关

1.4 算法及算法分析(Cont.)

- 常用的算法设计策略
  - 递归（**recursion algorithm**）与分治（**Divide-and Conquer**）
  - 动态规划（**Dynamic Programming**）
  - 贪心法（**Greedy Technique**）
  - 回溯法（**Back Tracking Method**）
  - 分支界限（**Branch and Bound Method**）
  - ....
- 算法的描述方法
  - 自然语言描述、伪代码描述、程序语言描述
  - 例：欧几里德算法—求两个自然数 ***m*** 和 ***n*** 的最大公因数
    - 辗转相除法

1.4 算法及算法分析(Cont.)

- 例：欧几里德算法----辗转相除法
  - 自然语言描述

    输入：自然数*m* 和*n*，假设*m* >*n*

    输出：最大公因数

    ① 求*m*除以*n*的余数*r*；

    ② 若*r*等于0，则*n*为最大公约数，算法结束；

    否则执行第③步；

    ③ 将*n*的值放在*m*中，将*r*的值放在*n*中；

    ④ 重新执行第①步。

m	*n*	*r*
108	32	12
32	12	8
12	8	4
8	4	0

  - 优点：容易理解
  - 缺点：冗长、二义性

1.4 算法及算法分析(Cont.)

- 例：欧几里德算法
  - 伪代码描述

    输入：自然数*m* 和*n*，假设*m* >*n*

    输出：最大公因数

    1. r = m % n;

    2. 循环直到 r 等于0

    2.1 m = n;

    2.2 n = r;

    2.3 r = m % n;

    3. 输出 n ;
  - 表达能力强，抽象性强，容易理解

1.4 算法(Cont.)

- 例：欧几里德算法
  - 程序语言描述

```
int  gcd( int m, int n)
{
    int r=m%n;
    while(r>0) {
        m=n;
        n=r;
        r=m%n;
    }
     return n;
}
```

```
int  gcd( int m, int n)
{
    while(n) {
        m=m%n;
        swap(m,n);
    }
    return m;
}
```

```
int gcd( int m, int n)
{
    if(n==0)
        return m;
    return gcd(n,m%n);
}
```

  - 表达精确，可执行，
  - 不好理解，隐藏算法的关键部分

1.4 算法及算法分析(Cont.)

- 评价算法效率的方法
  - 事后统计法：将算法实现，测量其时间和空间开销。
  - 优点：准确的实测值
  - 缺点：
    - 编程序实现算法将花费较多的时间和精力
      - 虽然必须，但我们只选其一
    - 所得实测结果依赖于计算机的软、硬件等环境因素。
  - 事前分析法：对算法所消耗资源的一种估算方法。
- 算法分析：对算法所需计算机资源—时间和空间进行估算。
  - 时间复杂性（**Time Complexity**）
  - 空间复杂性（**Space Complexity**）

1.4 算法及算法分析(Cont.)

- 算法分析—时间复杂度分析

算法的执行时间＝每条语句执行时间之和

↓

每条语句执行次数之和

↓

基本语句的执行次数

↓ 单位时间

执行次数×执行一次的时间

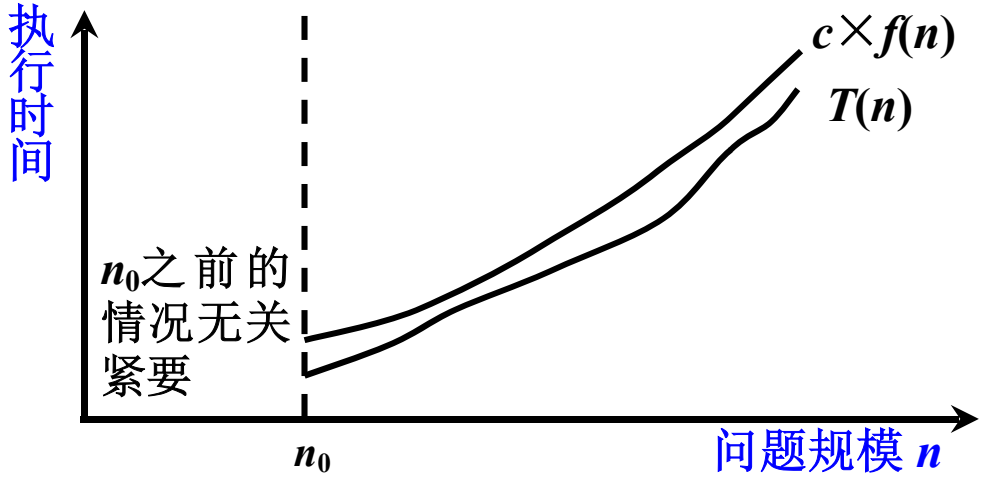
↓

指令系统、编译的代码质量

例：
```
for (i=1; i<=n; i++)
    for (j=1; j<=n; j++)
        x++;
```

1.4 算法及算法分析(Cont.)

- 算法分析—渐近上界记号O
  - 设 $T(n)$ 和 $f(n)$ 是定义在正整数集合上的两个函数，若存在一个正的常数 c 和正整数 n_0，使得对于任意 $n \geq n_0$，都有 $T(n) \leqslant c \times f(n)$，则称$T(n) = O(f(n))$



当问题规模充分大时，执行时间在渐近意义下的阶

- 其他渐近记号及相互关系： O 和 Ω、 Θ、 o 和 ω

1.4 算法及算法分析(Cont.)

- 算法分析—算法的时间复杂度（性）
  - 算法的执行时间，是基本（操作）语句重复执行的次数，它是问题规模的一个函数。我们把这个函数的渐近阶称为该算法的时间复杂度。
  - 问题规模：输入量的大小。
  - 基本语句：是执行次数与整个算法的执行时间成正比的操作指令。

例：
```
for (i=1; i<=n; i++)
    for (j=1; j<=n; j++)
        x++;
```

  - 问题规模：n
  - 基本语句：x++
  - 时间复杂性：O（n^2）

1.4 算法及算法分析(Cont.)

- 算法分析—大*O*记号性质
  - 定理：若$A(n)=a_m n^m+a_{m-1}n^{m-1}+\ldots+a_1 n+a_0$是一个$m$次多项式，则$A(n)=O(n^m)$。
  - 说明：在计算算法时间复杂度时，可以忽略所有低次幂（低阶）项和最高次幂（最高阶）项的系数。
  - 例：

```
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];
    }
```

问题规模：n

基本语句：*

复杂度：$O(n^3)$

1.4 算法及算法分析(Cont.)

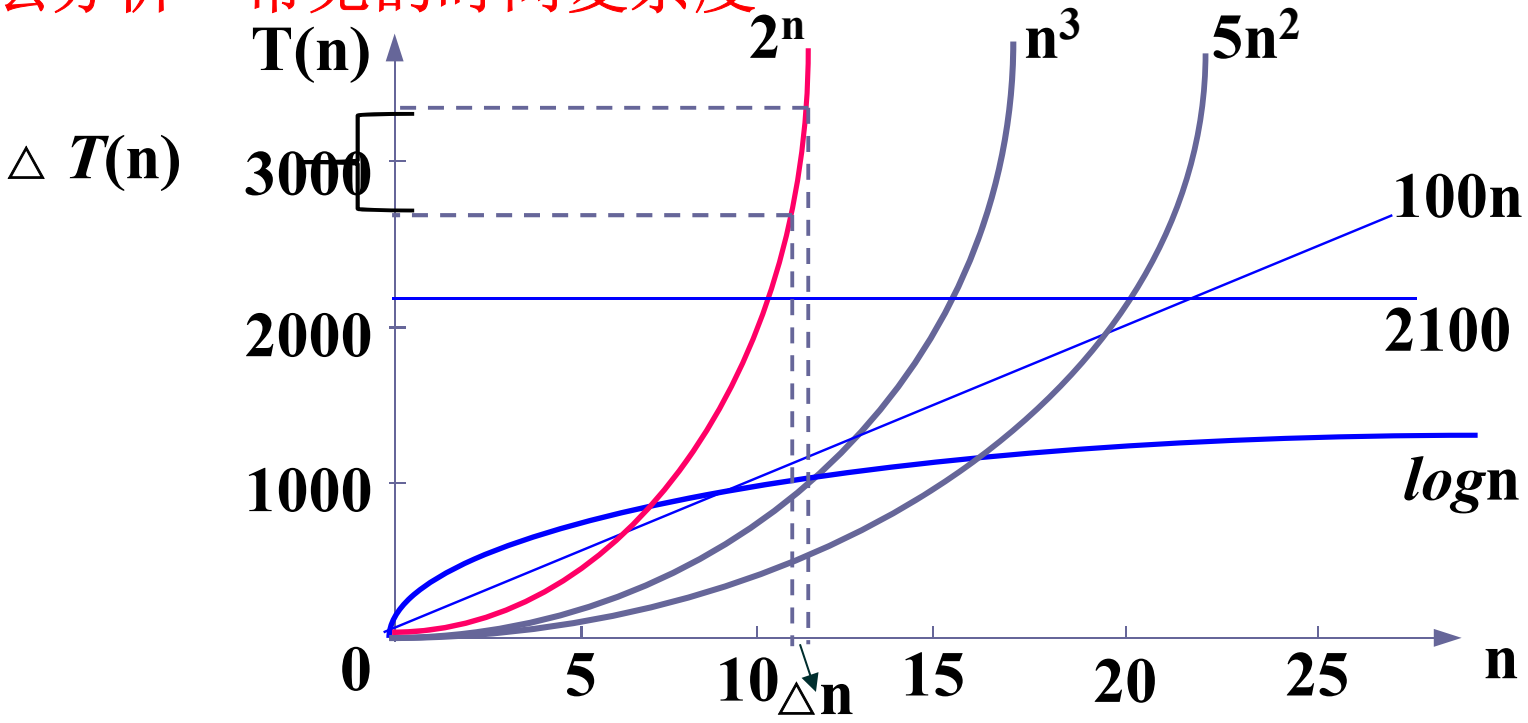
- 算法分析—最好情况、最坏情况、平均情况
  - 例：在一维整型数组A[n]中顺序查找与给定值k相等的元素（假设该数组中有且仅有一个元素值为k）。

```
int Find(int A[ ], int k, int n)
{
    for (i=0; i<n; i++)
       if (A[i]= =k) break;
    return i;
}
```

  - 基本语句的执行次数是否只与问题规模有关？
  - 结论：如果问题规模相同，时间代价与输入数据（的分布）有关，则需要分析最好情况、最坏情况、平均情况

1.4 算法及算法分析(Cont.)

- 算法分析—常见的时间复杂度



程序运行时间比较　$T(n)=O(f(n))$

- 常见阶的比较：
- $O(1)<O(\log_2 n)<O(n)<O(n\log_2 n)<O(n^2)<O(n^3)$
  $<\ldots<O(2^n)<O(n!)$

1.4 算法及算法分析(Cont.)

- 算法分析—时间复杂性分析的基本方法
  - 时间复杂性的运算法则

设$T_1(n)=O(f(n))$，$T_2(n)=O(g(n))$，则

- ①加法规则:$T_1(n)+T_2(n)=O(\max\{f(n), g(n)\})$
- ②乘法规则:$T_1(n)*T_2(n)=O(f(n)\cdot g(n))$

  - 时间复杂性的分析方法
    - 首先求出程序中各语句、各模块的运行时间,
    - 再求整个程序的运行时间。
    - 各种语句和模块分析应遵循的规则是:

1.4 算法及算法分析(Cont.)

- 算法分析—各种语句和模块分析应遵循的规则
  - (1)赋值语句或读/写语句:
    - 运行时间通常取O(1) .有函数调用的除外，此时要考虑函数的执行时间。
  - (2)语句序列:
    - 运行时间由加法规则确定，即该序列中耗时最多的语句的运行时间。
  - (3)分支语句:
    - 运行时间由条件测试（通常为O(1) ）加上分支中运行时间最长的语句的运行时间
  - (4)循环语句:

1.4 算法及算法分析(Cont.)

- 算法分析—各种语句和模块分析应遵循的规则
  - (4)循环语句:
    - 运行时间是对输入数据重复执行n次循环体所耗时间的总和
    - 每次重复所耗时间包括两部分：一是循环体本身的运行时间；二是计算循环参数、测试循环终止条件和跳回循环头所耗时间。后一部分通常为O(1)。
    - 通常，将常数因子忽略不计，可以认为上述时间是循环重复次数n和m的乘积，其中m是n次执行循环体当中时间消耗最多的那一次的运行时间(乘法规则)
    - 当遇到多重循环时，要由内层循环向外层逐层分析。因此，当分析外层循环的运行时间是，内层循环的运行时间应该是已知的。此时，可以把内层循环看成是外层循环的循环体的一部分。

1.4 算法及算法分析(Cont.)

- 算法分析—各种语句和模块分析应遵循的规则
  - (5)函数调用语句:
    - ①若程序中只有非递归调用，则从没有函数调用的被调函数开始，计算所有这种函数的运行时间。然后考虑有函数调用的任意一个函数P，在P调用的全部函数的运行时间都计算完之后，即可开始计算P的运行时间
    - ②若程序中有递归调用，则令每个递归函数对应于一个未知的时间开销函数T(n)，其中n是该函数参数的大小，之后列出关于T的递归方程并求解之。

1.4 算法及算法分析(Cont.)

- 算法分析—空间复杂性
  - 算法的空间复杂性是指算法在执行过程中的存储量需求
  - 一个算法的存储量需求除了存放算法本身所有的指令、常数、变量和输入数据外，还包括对数据进行操作的工作单元和存储实现计算所需信息的辅助空间
  - 算法的存储量需求与输入的规模、表示方式、算法采用的数据结构、算法的设计以及输入数据的性质有关
  - 算法的执行的不同时刻，其空间需求可能是不同的
  - 算法的空间复杂性是指算法在执行过程中的最大存储量需求
  - 空间复杂性的渐近表示----空间复杂度

    $S(n) = O(f(n))$ 其中，n为问题的输入规模

1.4 算法及算法分析(Cont.)

- 算法分析—例:分析下述“冒泡”排序程序的时间复杂性。

```
void BubbleSort( int A[], int n )
{   int i, j, temp;
(1)     for (i=0; i<n-1; i++)
(2)         for (j=n-1; j>=i+1; j--)
(3)             if (A[j-1]>A[j]) {
(4)                 temp=A[j-1];
(5)                 A[j-1]=A[j];
(6)                 A[j]=temp;
                }
}
```

•时间复杂度：

$$O(\sum_{i=0}^{n-2}(n-i-1)) \le O(n(n-1)/2) = O(n^2)$$

•空间复杂度：

O（1）

1.4 算法及算法分析(Cont.)

- 算法分析—例:编写求n!的程序，并分析其时间复杂性。

●求n!的递归算法

```
long  fact ( int  n)
{   if ( n==0 ) || ( n ==1 )
        return( 1 );
    else
        return( n * fact(n – 1));
}
```

●时间复杂性的递归方程

$$T(n)=\begin{cases} C & \text{当 } n=0,\ n=1 \\ G+T(n-1) & \text{当 } n>1 \end{cases}$$

●解递归方程:

$$\begin{aligned} T(n) &= G+T(n-1) \\ T(n-1) &= G+T(n-2) \\ T(n-2) &= G+T(n-3) \\ &\cdots\cdots \\ T(2) &= G+T(1) \\ +\ T(1) &= C \\ \hline T(n) &= G(n-1)+C \end{aligned}$$

取 $f(n)=n$

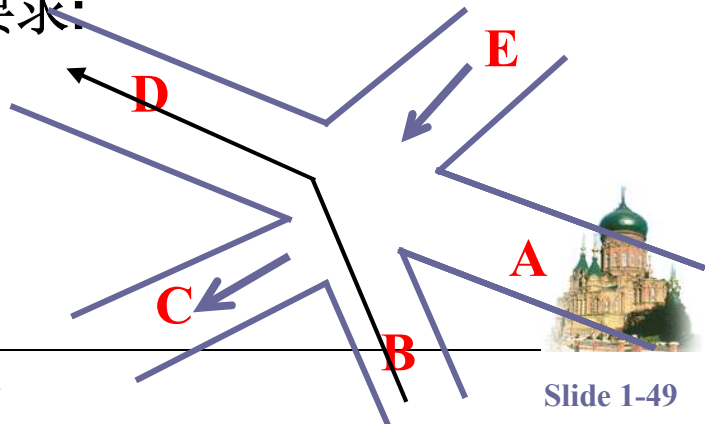
$\therefore T(n)=O(f(n))$

$=O(n)$

•空间复杂度：O（n）

1.5 逐步求精的程序设计方法

- 例:交叉路口的交通安全管理问题
  - 问题描述:
    - 一个具有多条通路的交叉路口，当允许某些通路上的车辆在交叉路口“拐弯”时，必须对其他一些通路上的 车辆加以限制，不允许同时在交叉路口“拐弯”,以免发生碰撞。所有这些可能的“拐弯”组成一个集合。
  - 基本要求:
    - 把这个集合分成尽可能少的 组,使得每组中所有的“拐弯”都能同时进行而不发生碰撞。这样，每组对应一个指挥灯，因而实现了用尽可能少的指挥灯完成交叉路口的管理。
  - 实例:
    - 一有5条路组成的交叉路口,其中有2条路是单行道，把从一条路到另一条路的通路称为“拐弯”，有的“拐弯”可以同时通过，有些“拐弯”不能同时通过，共有13个“拐弯”。要求:
    - (1)设置一组交通灯,实现安全管理;
    - (2)使交通灯的数目最少。

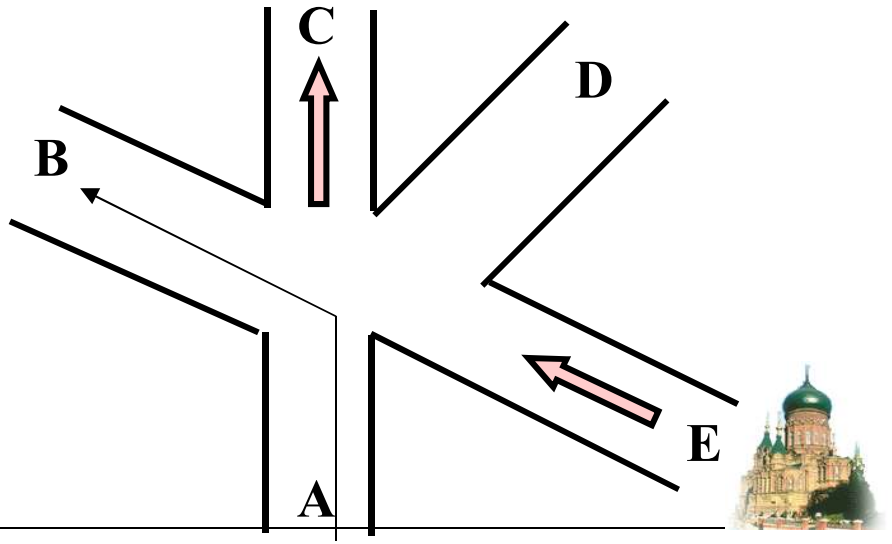


1.5 逐步求精的程序设计方法(Cont.)

- 1.模型化（建模）
  - 对实际问题进行分析，选择适当的模型来描述问题，即建模；
- 2.确定算法
  - 根据模型，找出解决问题的方法，即算法；
- 3.逐步求精
  - 对用自然语言等描述的算法逐步细致化、精确化和形式化，这一阶段可能包括多步求精。
  - 当逐步求精到某一步时，根据程序中所使用的数据形式，定义若干ADT，并且用ADT中的操作代替对应的非形式语句。
- 4.ADT的实现
  - 对每个ADT，选择适当的数据结构表示数学模型，并用相应的函数实现每个操作。

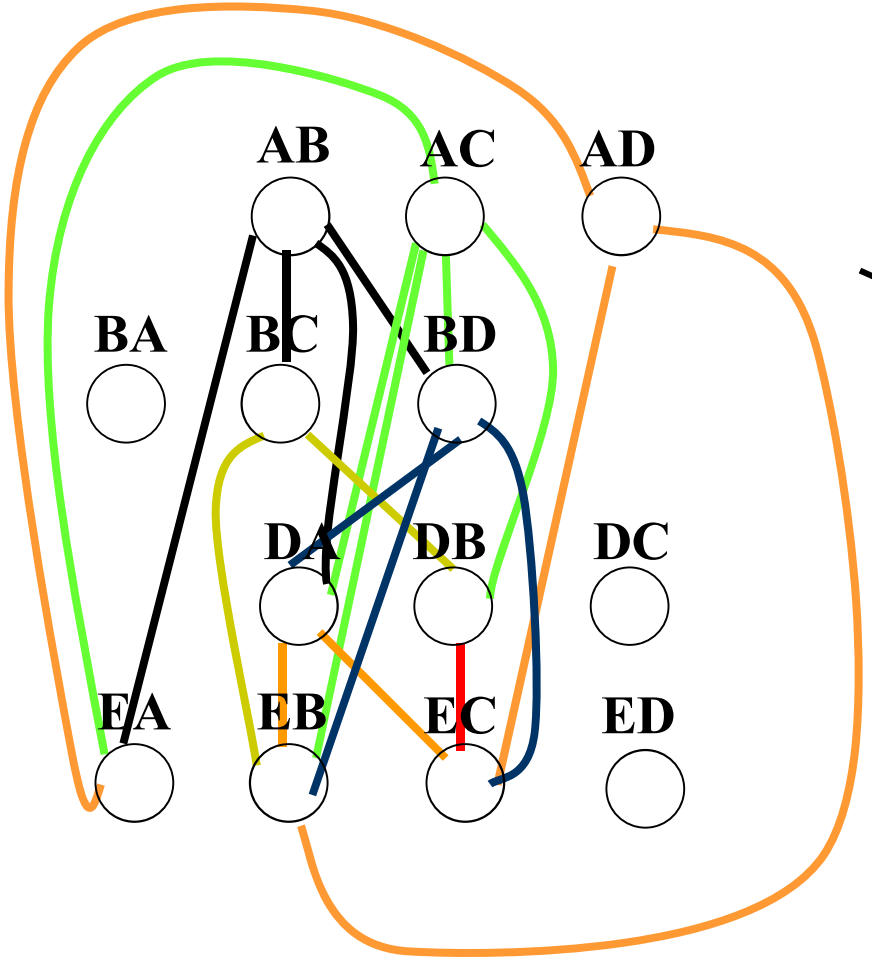
1.5 逐步求精的程序设计方法(Cont.)

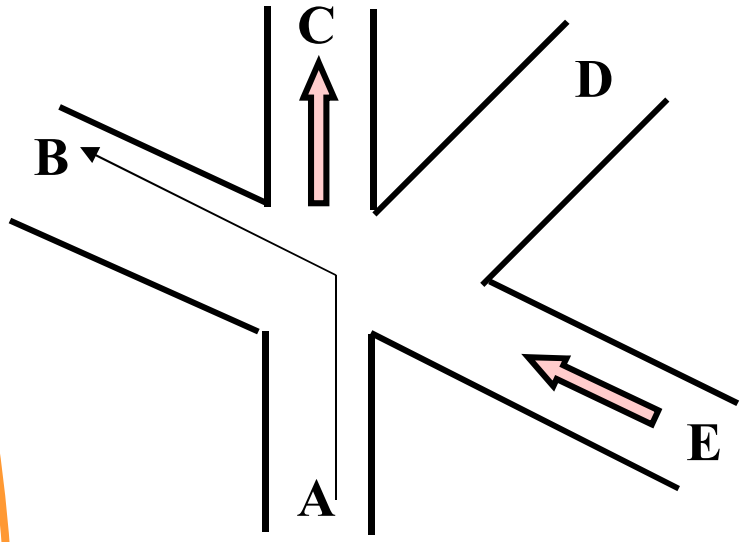
- 例:交叉路口的交通安全管理问题
  - 1.模型化：
    - 用图作为交叉路口的数学模型;
    - 每个“拐弯”对应图中的一个顶点;
    - 若两个“拐弯”不能同时进行,则用用一条边把这两个“拐弯”所对应的两个结点连接起来，并且说这两个顶点是相邻的。



1.5 逐步求精的程序设计方法(Cont.)

- 1.模型化：





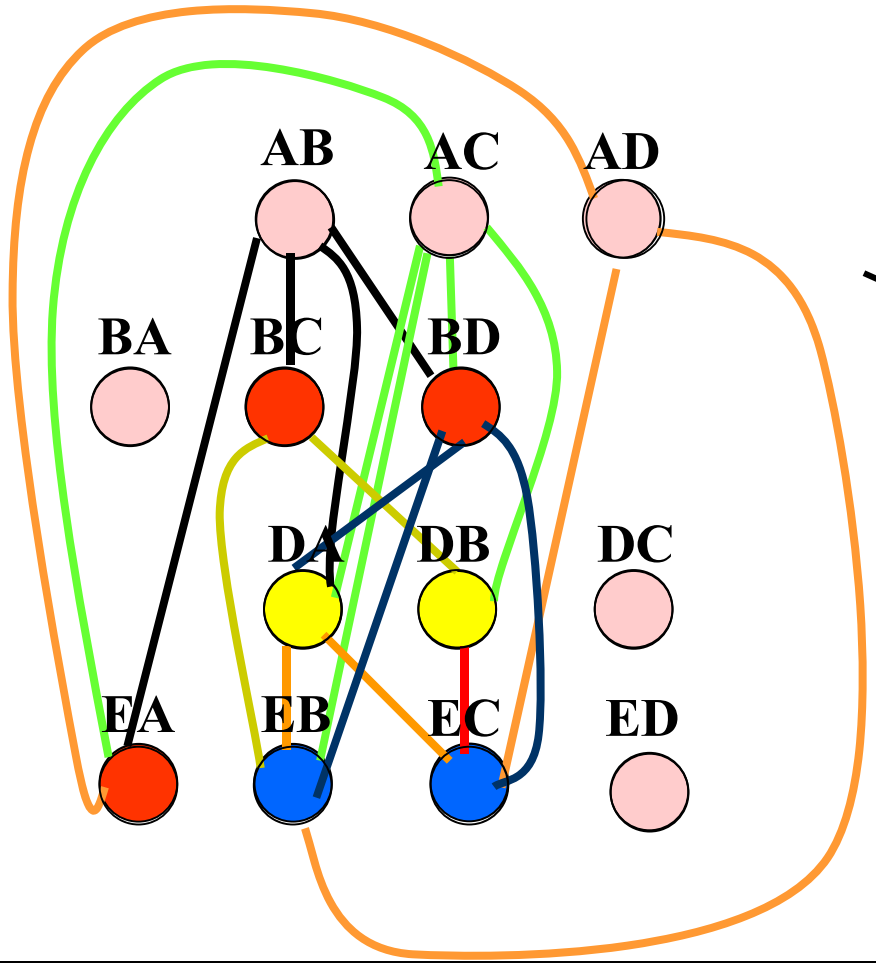
1.5 逐步求精的程序设计方法(Cont.)

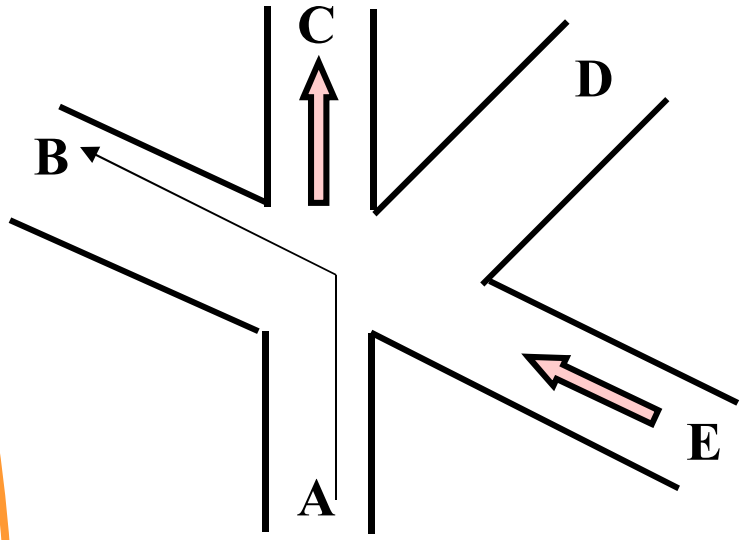
- **2.确定算法：**
  - (1)穷举法；(2)试探法；(3)贪心法
  - “贪心”算法的思想是
    - 首先用第一种颜色对图中尽可能多的顶点着色(尽可能多表现出“贪心”)；
    - 然后用第二种颜色对余下的顶点中尽可能多的顶点着色；
    - 如此等等，直至所有的顶点都着完色。

1.5 逐步求精的程序设计方法(Cont.)

2.确定算法：

- 贪心法





1.5 逐步求精的程序设计方法(Cont.)

- 求解问题的"贪心"算法如下:

```
void greedy(GRAPH G, SET newclr)
/*把G中可以着同一色的顶点放入newclr*/
{
(1)  newclr=Φ ;
(2)  while (G中有未着色的顶点v)
(3)        if(v不与newclr中的任何顶点相邻){
(4)              对v着色;
(5)              将v放入newclr;  }
 }
```

- 其中，G是被着色的图，newclr的初值为空，算法执行的结果形成可以着相同颜色的顶点的集合newclr。
- 只要重复调用greedy算法，直到图中的所有顶点都被着色为止，即可求出问题的解。

1.5 逐步求精的程序设计方法(Cont.)

- **3.逐步求精：** 第一步求精：

```
    void greedy(GRAPH G, SET newclr)
(1) {   newclr=Φ; int found;
(2)     while(G中有未着色的顶点v){
(3.1)       found=0;  /*found的初值为false*/
(3.2)       for (newclr中的每一个顶点w）
(3.3)           if(v与w相邻)
(3.4)               found=1;
(3.5)       if(found==0){  /*v与newclr中的任何顶点都不相邻 */
(4)             对v着色;
(5)             将v放入newclr;  }
            }
        }
```

1.5 逐步求精的程序设计方法(Cont.)

- **3.逐步求精：** 第二步求精：

```
void greedy(GRAPH G, SET newclr)
{   newclr=Φ ; int found;
    v=G中第一个未着色的顶点;
    while (v!=0){ /*G中还有未着色的顶点v*/
        found=0;
        w=newclr中的第一个顶点;
        while(w!=0){ /*newclr中的顶点还没取尽*/
            if(v与w相邻)
                found=1;
            w=newclr中的下个顶点;
        }
        if(found==0){
            对v着色;
            将v放入newclr;
        }
        v=G中下一个未着色的顶点;
    }
}
```

1.5 逐步求精的程序设计方法(Cont.)

- **3.逐步求精：** 第三步求精：

由上一步求精的结果可见，算法中大部分操作都归结为对图和集合的操作。设G和S分别是抽象数据型GRAPH和SET的实例，

- 在G上规定如下操作：
  - **FIRSTG(G)**：返回G中的第一个未加标记的（未着色的）元素；若G中没有这样的元素存在，则返回NULL
  - **EDGE(v,w,G)**：若v和w在G中相邻，则返回true，否则返回false
  - **MARK(v,G)**：标记G中的元素v
  - **ADDG(v,G)**：将元素v放入G中
  - **NEXTG(G)**：返回G中下一个未标记得元素，若G中没有这样的元素存在，则返回NUL。
- 在S上规定如下操作：
  - **MAKENULL(S)**将集合S置空
  - **FIRSTS(S)**：返回S中的第一个元素；若S为空集，则返回NULL
  - **NEXTS(S)**：返回S中的下一个元素；若S中没有下一个元素，则返回NULL
  - **ADDS(v,S)**：将v放入S中

1.5 逐步求精的程序设计方法(Cont.)

- **3.逐步求精：** 第三步求精：

```
void greedy(GRAPH G, SET newclr)
/*类型GRAPH和SET有待说明*/
{
    int found;
    elementtype v,w;/*elementtype可以自定义*/
    MAKENULL(newclr);
    v=FIRSTG(G);
    while(v!=NULL){
        found=0;
        w=FIRSTS(newclr);
        while(w!=NULL){
            if(EDGE(v,w,G))
                found=1;
            w=NEXTS(newclr);
        }
        v=NEXTG(G);
    }
}
```

4.ADT的实现：

最后一步就是给出类型 **elementtype**的定义和实现抽象数据型**GRAPH**及**SET**。此后，上述函数就是可执的程序了.

1.5 逐步求精的程序设计方法(Cont.)

- 算法与数据结构的关系

  两者既有联系又有区别。

  - 联系是：程序 = 算法 + 数据结构
    - 数据结构是算法实现的基础，
    - 算法总是要依赖某种数据结构来实现的。
    - 算法的操作对象是数据结构。
  - 区别是：
    - 数据结构关注的是数据的逻辑结构、存储结构以及基本操作，
    - 而算法更多的是关注如何在数据结构的基础上解决实际问题。
    - 算法是编程思想，数据结构则是这些思想的基础。

本章小结

- 数据结构的基本概念、相关术语和研究对象
- 抽象数据型定义和实现
- 算法及其复杂性分析
- 解决问题的一般过程和算法的逐步求精方法

数据结构与算法的五个方面

- 数据的逻辑结构
  - 线性结构
    - 线性表
    - 栈
    - 队列
  - 非线性结构
    - 树型结构
    - 图型结构
- 数据的存储结构
  - 顺序存储
  - 链式存储
  - 散列存储
  - 索引存储
- 数据的操作(算法)：检索、排序、插入、删除、修改等
- 时间和空间效率
- 数据结构与算法的应用

毛竹的启示

毛竹用了4年的时间，仅长了3 厘米，在第五年开始以每天 30 cm的速度疯狂的生长，仅用了 6周时间就长到了15米。其实在前面的四年，竹子将根在土壤里延伸了数百平米。做人做事亦是如此。不要担心你此时此刻的付出得不到回报，因为这些付出都是为了扎根，人生需要储备！多少人，没熬过那三厘米！

•竹生空野外，梢云耸百寻。无人赏高节，徒自抱贞心。

— 咏竹 (梁.刘孝先)

•咬定青山不放松，立根原在破岩中。千磨万击还坚劲，任尔东西南北风。

— 竹石 （清.郑燮）