

数据结构与算法

Data Structures 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



2014/9/15



课程说明

- ➡ 课程编号: **13SC03100100**
- ➡ 授课学时: **48** (2至14周, 4学时/周)
- ➡ 实验学时: **12** (第6\8\10\12周, 3学时/周)
- ➡ 课程分类: 专业(技术)基础
- ➡ 答疑地点: 综合楼**804**, 每周1次
- ➡ 课程资源:
 - <http://cms.hit.edu.cn>
- ➡ 考核形式:
 - 期末笔试**70%**+平时成绩**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

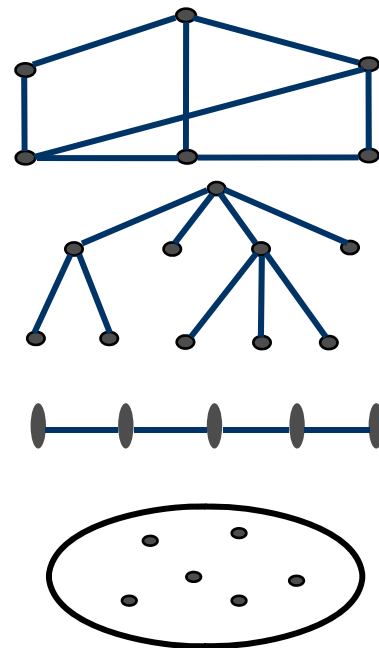
通俗地讲，

- **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

四种基本的
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：● **属于同一个集合**；



- **查找**：线性结构、树型结构和散列结构上查找。
- **排序**：内部排序和外部排序（包括文件）。





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！



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



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。

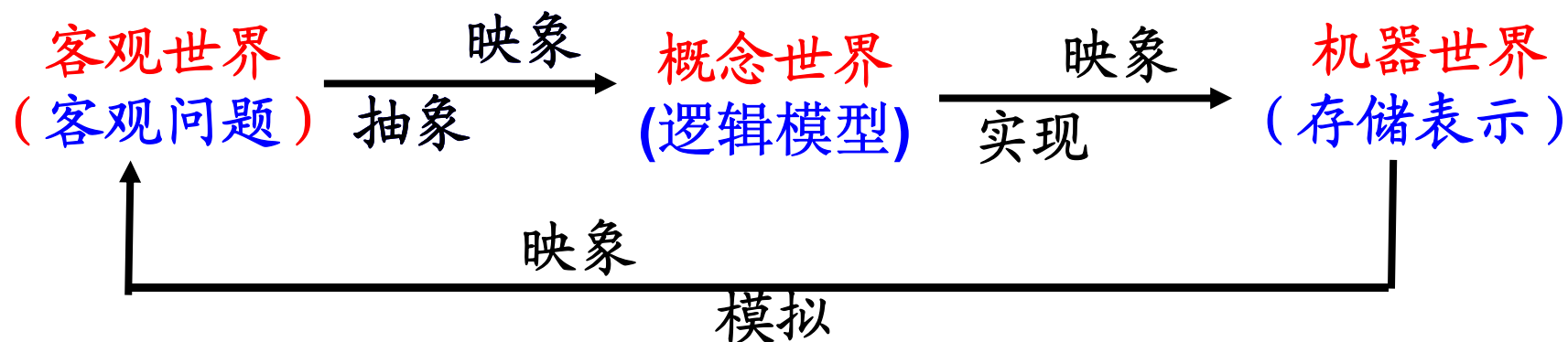




1.1 数据结构的兴起和发展

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。
- 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

➤ 数据结构的发展并未终结.....





1.2 研究对象与基本概念

例1 学籍管理问题——表结构

■ 完成什么功能？各表项之间是什么关系？

学号	姓名	性别	出生日期	政治面貌
0001	王晓东	男	1990/09/02	团员
0002	李明远	男	1989/12/25	党员
0003	张蔷薇	女	1991/03/26	团员
...

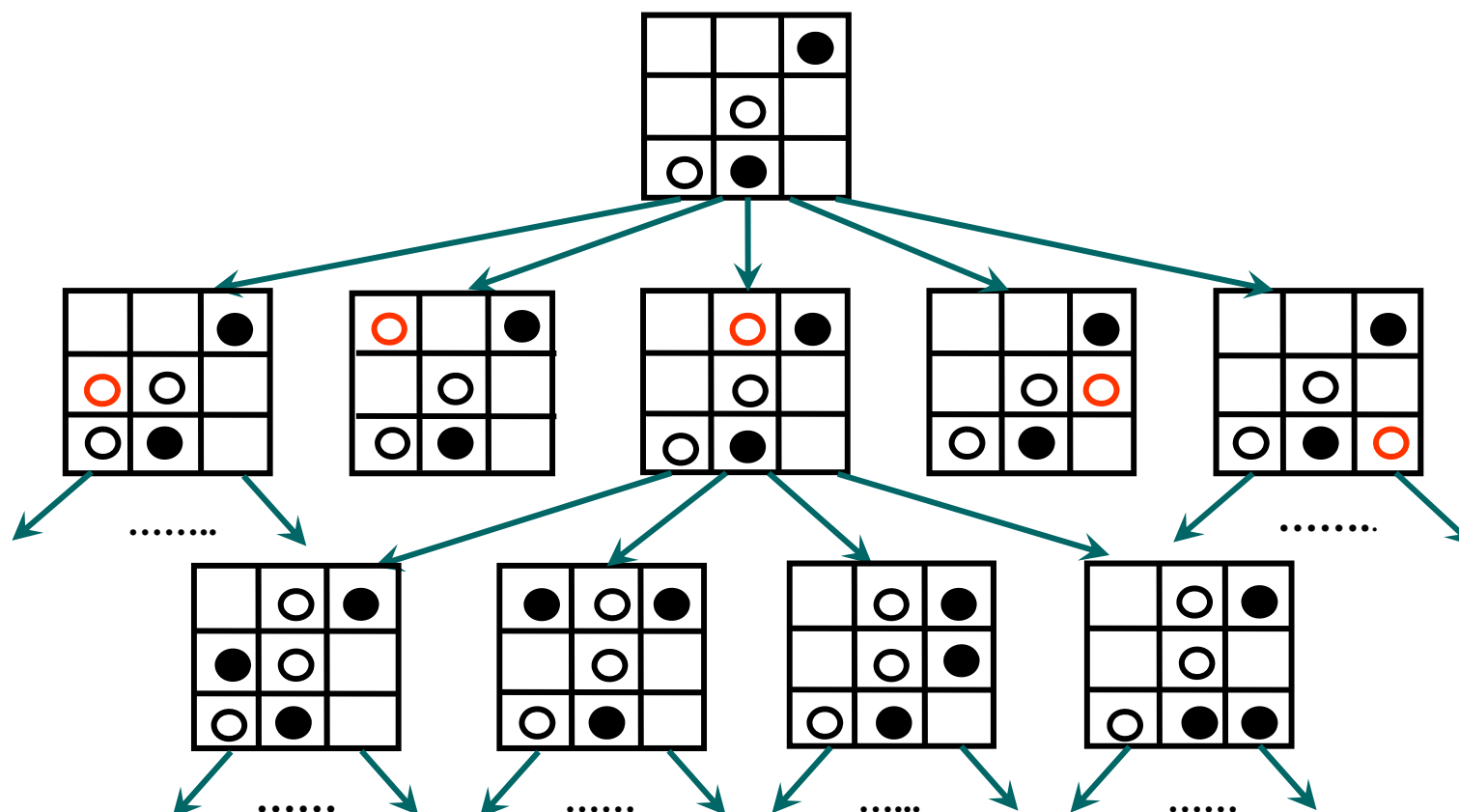




1.2 研究对象与基本概念(Cont.)

例2 人机对弈问题——树结构

■ 如何实现对弈？各格局之间是什么关系？



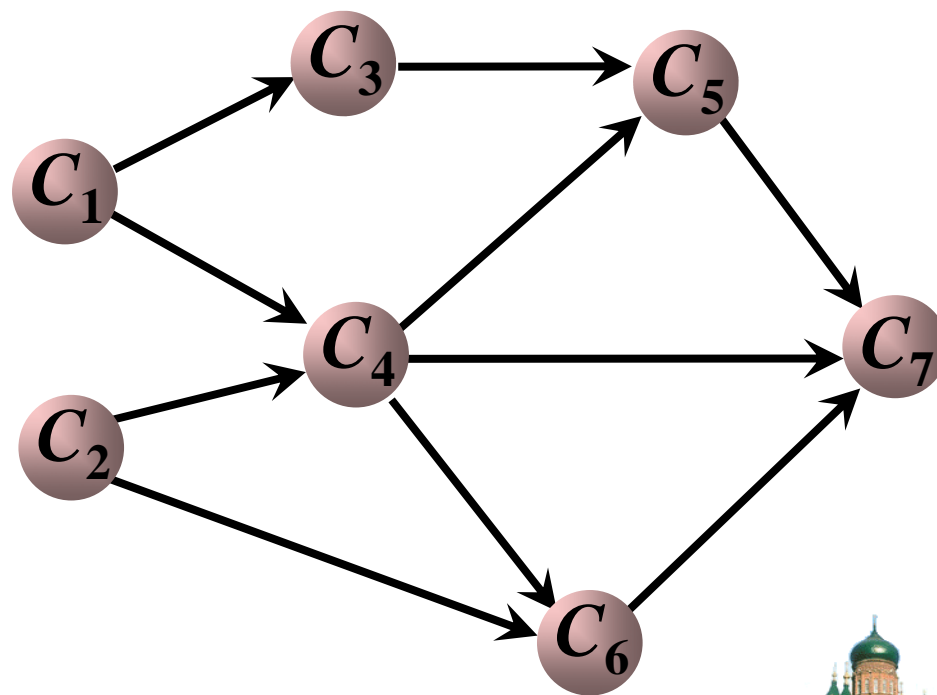


1.2 研究对象与基本概念(Cont.)

例3 教学计划编排问题——图结构

如何表示课程之间的先修关系？

编号	课程名称	先修课
C ₁	高等数学	无
C ₂	计算机导论	无
C ₃	离散数学	C ₁
C ₄	程序设计	C ₁ , C ₂
C ₅	数据结构	C ₃ , C ₄
C ₆	计算机原理	C ₂ , C ₄
C ₇	数据库原理	C ₄ , C ₅ , C ₆





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.)

数据结构的的基本概念

- **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(Cont.)

➡ 数据结构

- 数据元素之间的**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。



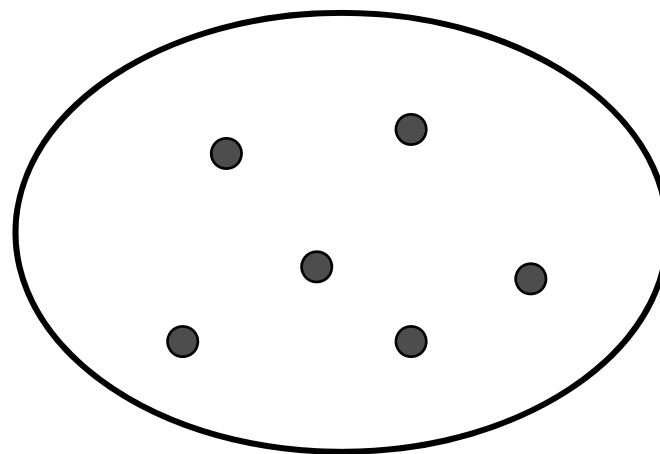


1.2 研究对象与基本概念(Cont.)

➤ 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；



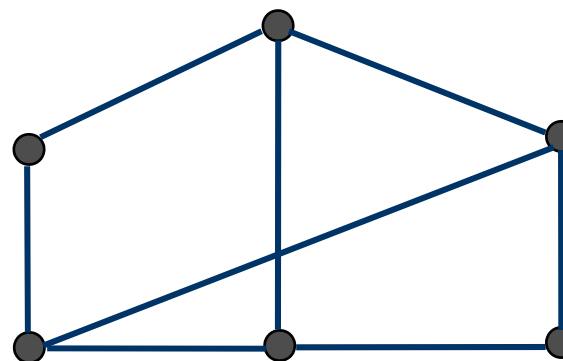
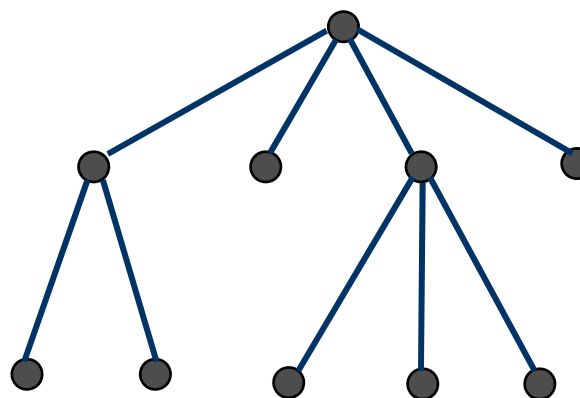


1.2 研究对象与基本概念(Cont.)

➡ 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**。





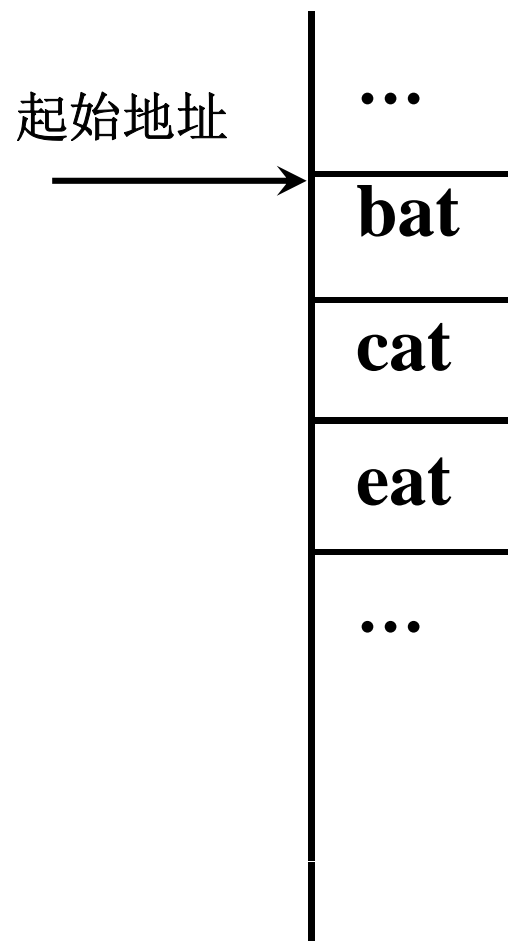
1.2 研究对象与基本概念(Cont.)

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

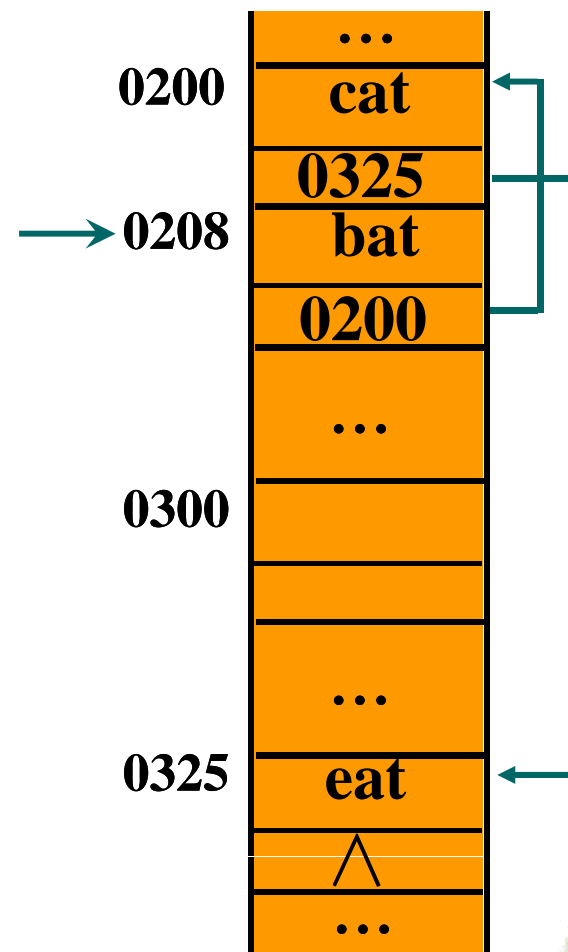
■ 顺序存储结构

- 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.)

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的视图，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存储，而采用不同的存储结构，其数据处理的效率往往是不同的。

数据结构与算法的学习内容

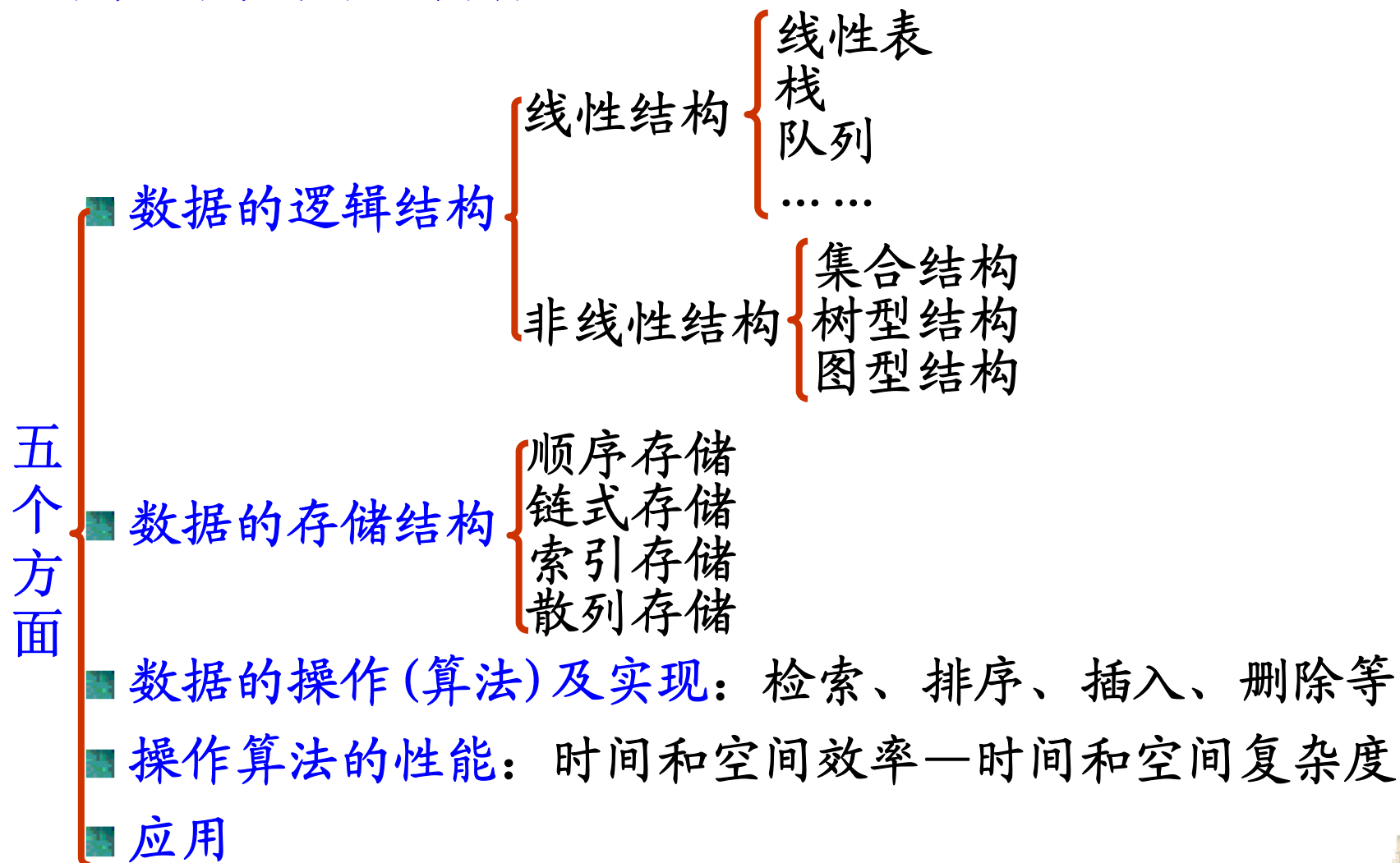
- 数据对象的结构形式,各种数据结构的性质(逻辑结构);
- 数据对象和“关系”在计算机中的表示(物理结构/存储结构);
- 数据结构上定义的基本操作(算法)及其实现;
- 算法的效率（时间和空间）;
- 数据结构的应用,如数据分类,检索等.





1.2 研究对象与基本概念(Cont.)

➔ 研究对象即学习内容





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: $\text{ADT int} = (\{x | x \in \mathbb{Z}\}, \{+, -, *, /, \%, \leq, ==\})$
 - 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**。





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

算法的相关概念

- **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.)

➡ 例5：欧几里德算法——辗转相除法求两个自然数 m 和 n 的最大公约数。

■ 自然语言描述

- ①输入 m 和 n ;
- ② 求 m 除以 n 的余数 r ;
- ③ 若 r 等于0，则 n 为最大公约数，算法结束；否则执行第④步；
- ④ 将 n 的值放在 m 中，将 r 的值放在 n 中；
- ⑤ 重新执行第②步。

■ 优点：容易理解

■ 缺点：冗长、二义性





1.4 算法及算法分析(Cont.)

➡ 例5：欧几里德算法——辗转相除法求两个自然数 m 和 n 的最大公约数。

■ 伪代码描述

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 算法及算法分析(Cont.)

评价算法效率的方法

- **事后统计**：将算法实现，测算其时间和空间开销。
- **优点**：准确的实测值
- **缺点**：
 - 编写程序实现算法将花费较多的时间和精力；
 - 所得实验结果依赖于计算机的软、硬件等环境因素。

- **事前分析**：对算法所消耗资源的一种估算方法。

算法分析：对算法所需要的计算机资源——时间和空间进行估算。

- **时间复杂性 (Time Complexity)**
- **空间复杂性 (Space Complexity)**

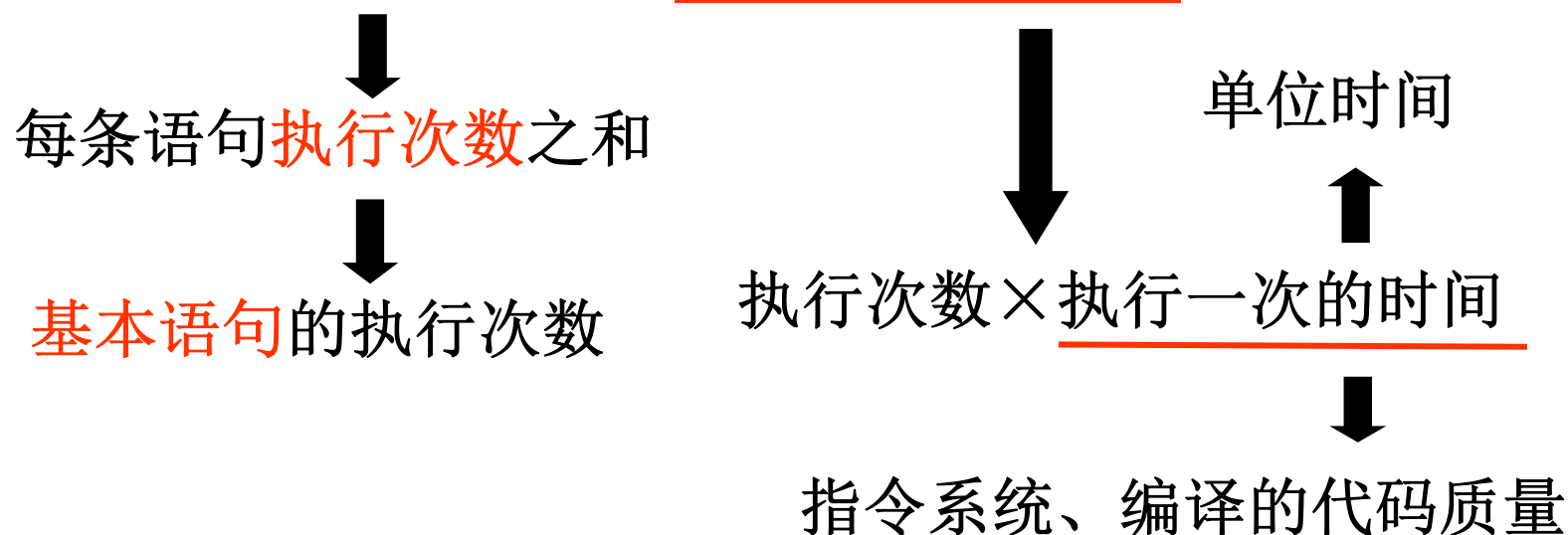




1.4 算法及算法分析(Cont.)

➤ 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



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----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：输入量的多少。
- 基本语句：是执行次数与整个算法的执行次数成正比的操作指令。

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 (n²)

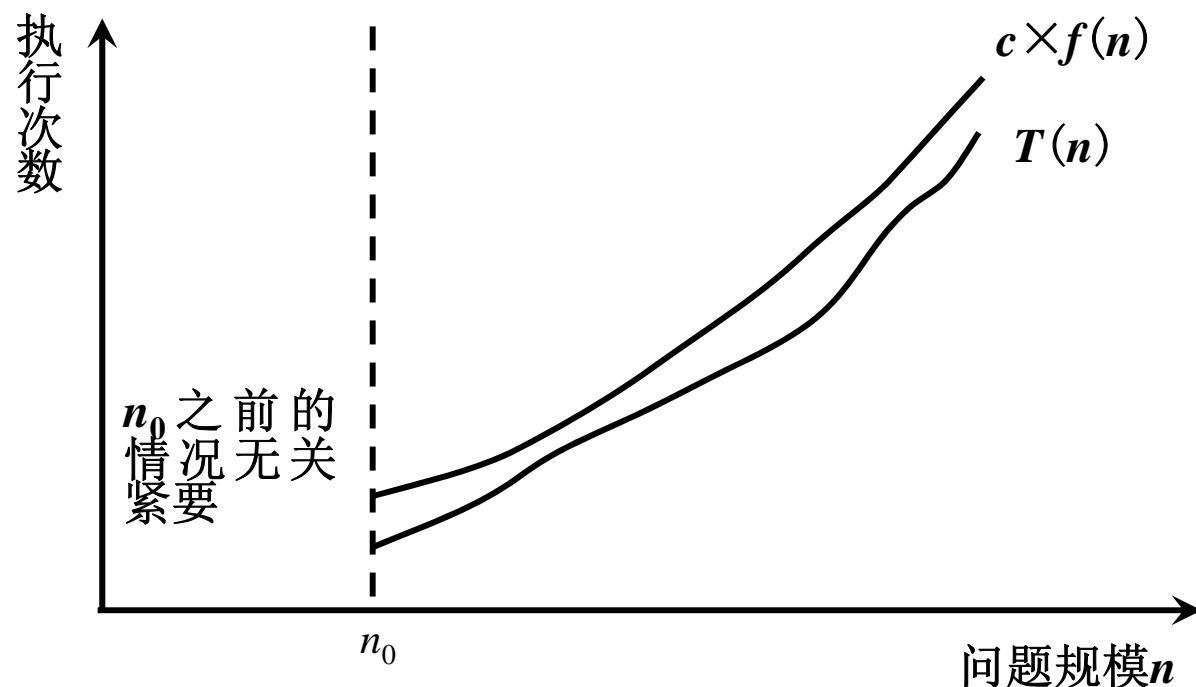




1.4 算法及算法分析(Cont.)

算法分析----大O符号

■ **定义** 若存在两个正的常数 c 和 n_0 , 对于任意 $n \geq n_0$, 都有 $T(n) \leq c \times f(n)$, 则称 $T(n) = O(f(n))$



当问题规模充分大时在渐近意义下的阶





1.4 算法及算法分析(Cont.)

算法分析----大O符号

■ 定理：若 $A(n)=a_m n^m+a_{m-1}n^{m-1}+\dots+a_1n+a_0$ 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 n)
{
    for (i=0; i<n; i++)
        if (A[i]==k) break;
    return i;
}
```

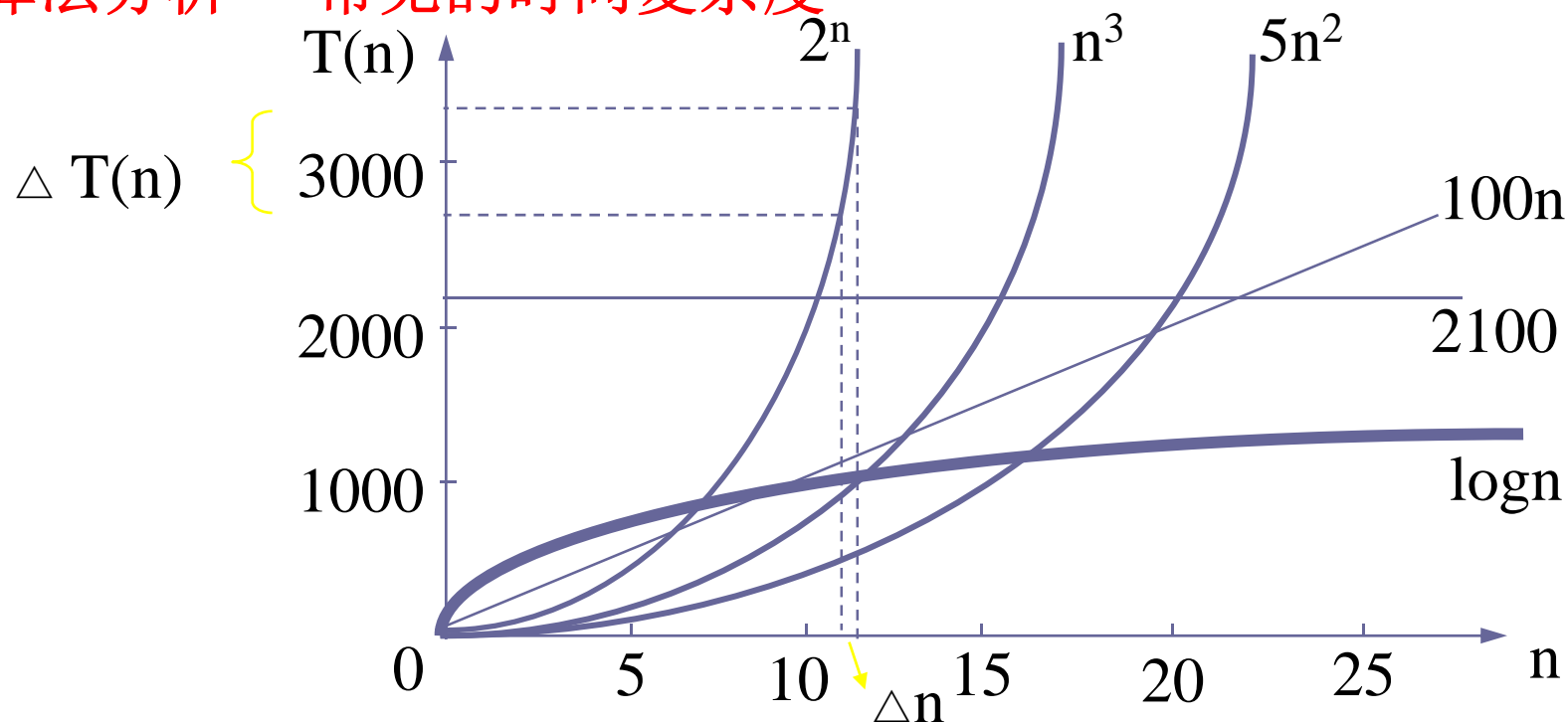
- 基本语句的执行次数是否只和问题规模有关？
- **结论**：如果**问题规模相同**，时间代价与**输入数据**（的**分布**）有关，则需要分析**最好情况、最坏情况、平均情况**





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:

■ $O(1) < O(\log_2 n) < O(n) < O(n \log_2 n) < O(n^2) < O(n^3)$
 $< \dots < O(2^n) < O(n!)$





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函数开销函数 $T(n)$ ，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**—例:分析下述“气泡”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;
```

```
           }
```

```
 }
```

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





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:

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

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

$$T(n-2) = G + T(n-3)$$

... ..

$$T(2) = G + T(1)$$

$$+ T(1) = C$$

$$T(n) = G(n-1) + C$$



$$\text{取 } f(n) = n$$

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





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**

$T(n) = O(f(n))$ 其中， n 为问题的输入规模





1.5 逐步求精的程序设计方法

➡ 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

■ 问题描述:

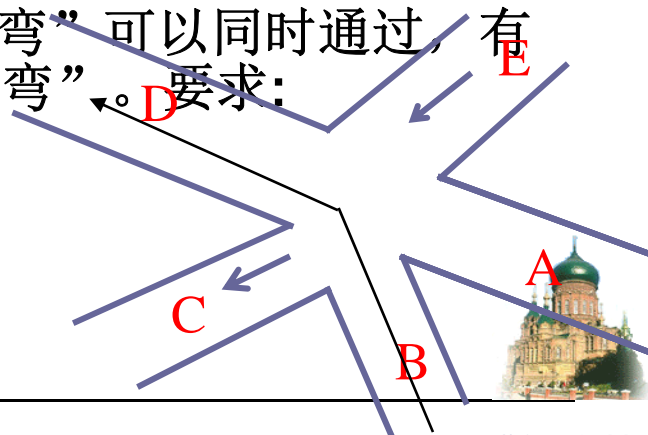
- 一个具有多条通路的交叉路口,当允许某些通路上的车辆在交叉路口“拐弯”时,必须对其他一些通路上的车辆加以限制,不允许同时在交叉路口“拐弯”,以免发生碰撞。所有这些可能的“拐弯”组成一个集合。

■ 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.)

✦ 例:交叉路口的交通安全管理问题

■ 1.模型化:

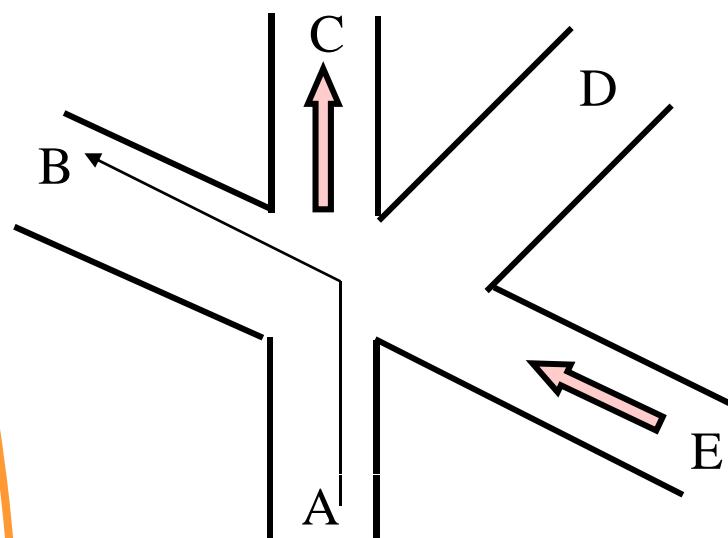
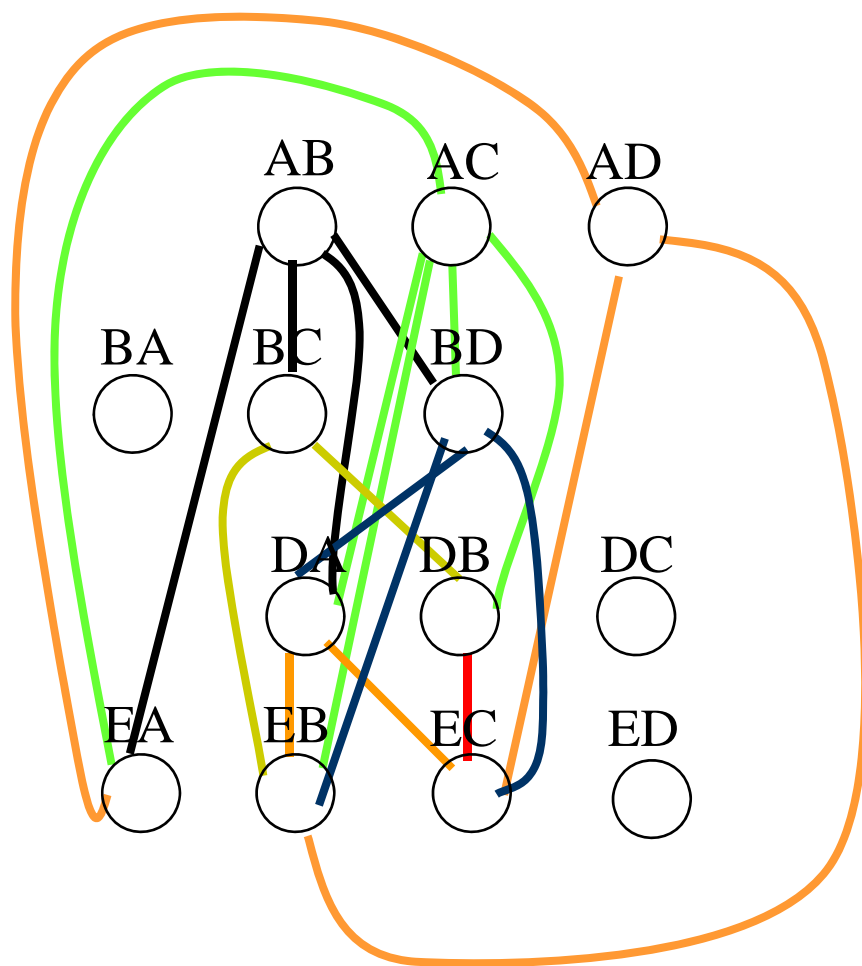
- 用图作为交叉路口的数学模型;
- 每个“拐弯”对应图中的一个顶点;
- 若两个“拐弯”不能同时进行,则用一条边把这两个“拐弯”所对应的两个结点连接起来,并且说这两个顶点是相邻的。





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。

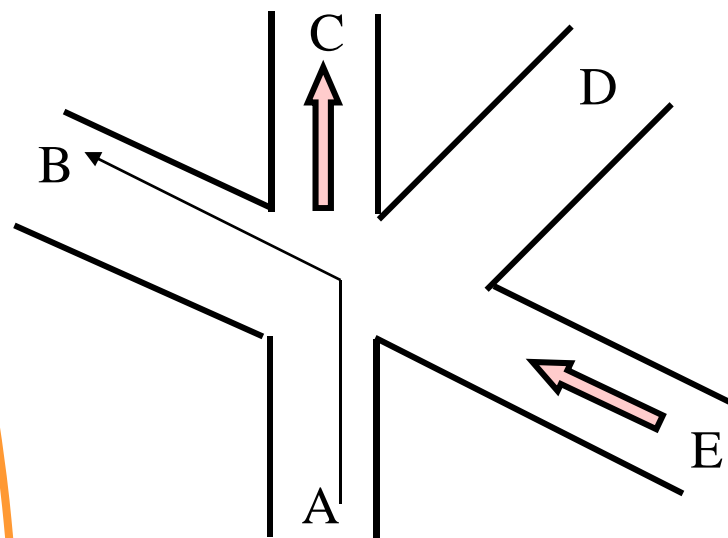
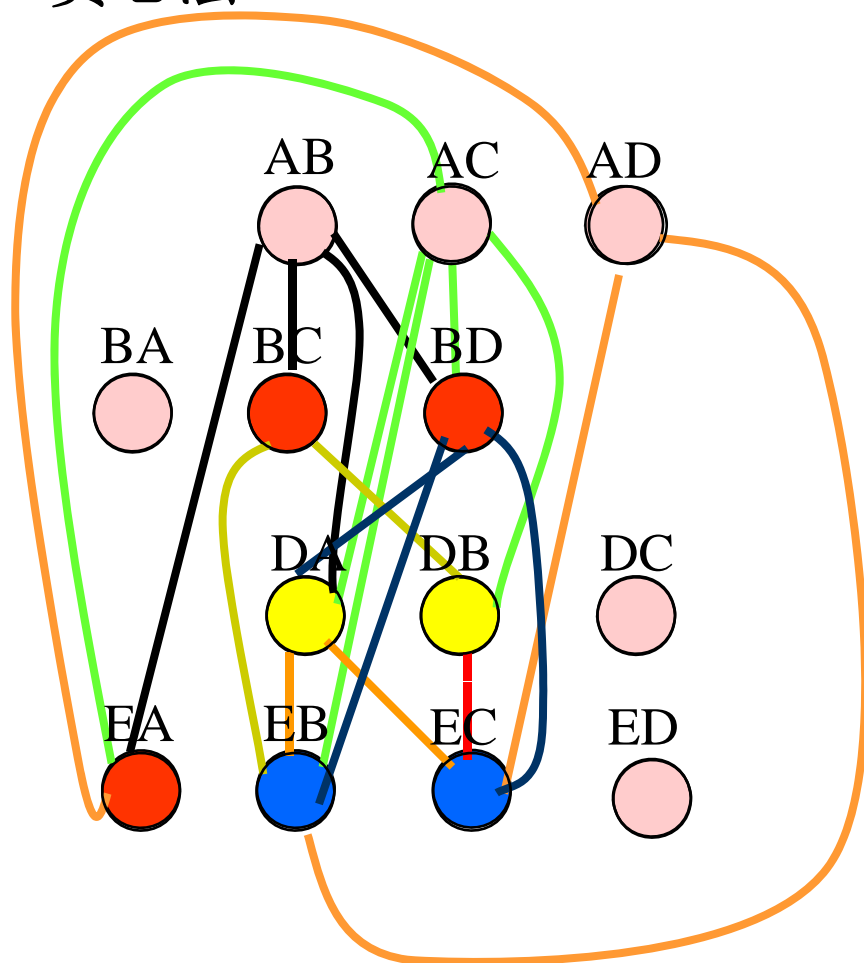




1.5 逐步求精的程序设计方法(Cont.)

2. 确定算法:

贪心法





1.5 逐步求精的程序设计方法(Cont.)

➔ 用C语言描述的"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= $\Phi$  ; 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，我们在**G**上规定如下操作：

(1)**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**。

(2)**EDGE(v,w,G)**若**v**和**w**在**G**中相邻，则返回**true**，否则返回**false**

(3)**MARK(v,G)**标记**G**中的元素**v**。

(4)**ADDG(v,G)**将元素**v**放入**G**中。

(5)**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：

(1)**MAKENULL(S)**将集合**S**置空。

(2)**FIRSTS(S)**返回**S**中的第一个元素；若**S**为空集，则返回**NULL**。

(3)**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**。

(4)**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, SETnewclr)

➔ /*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：

按上述函数，最后一步工作就是
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。





本章小结

- 数据结构的基本概念、相关术语和研究对象
- 抽象数据类型定义和实现
- 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

