

数据结构 —— C语言版

主讲：李艳娟

Tel: 13654550662

qq: 173919189

办公室：成栋楼1034



学时安排

● 上课：56学时

● 上机：16学时

- 地点：丹青楼9楼机房
- 时间：根据上课进度安排

● 教材：

- 《数据结构——C语言版》严蔚敏 吴伟民
清华大学出版社

● 参考：

- 《算法与数据结构考研试题精析（第3版）》陈守孔等 机械出版社 2015.3
- 《数据结构与算法基础》大连理工大学出版社
- 《算法与数据结构》陈守礼 机械工业出版社
- 《算法与数据结构》张乃孝 高教出版社



数据结构发展简史

- 作为独立课程国外1968年开始设立。
- 1968年美国KNUTH教授开创了数据结构的最初体系。
- 计算机、信管专业的专业基础课。
- 非计算机专业的主要选修课。



数据结构课程的内容

- 数据结构是介于数学、计算机硬件和计算机软件之间的一门计算机科学与技术专业的核心课程，是编译原理、操作系统、数据库、人工智能等课程的基础。同时，数据结构技术也广泛应用于信息科学、系统工程、应用数学以及各种工程技术领域。
- 数据结构课程的先修课程有高级语言程序设计(C语言)和离散数学。

数据结构课程的内容（续）

- 数据结构课程集中讨论软件开发过程中的设计阶段、分析阶段所涉及到的数据的**若干基本问题**（包括数据的**逻辑结构**、**存储结构**，及对数据施加的**操作**）。此外，为了构造出好的数据结构及其操作的实现，还需考虑数据结构及其实现的**评价与选择**。因此，数据结构的内容包括**三个层次**的**五个“要素”**，如下图所示：



数据结构课程内容体系

层次	数据表示	数据处理
抽象	逻辑结构	基本运算
实现	存储结构	算法
评价	不同结构的比较及算法分析	



数据结构课程的重要性

- 后续课程（数据库、操作系统、人工智能、编译等）的基础
- 硕士研究生考试课程



课程目的

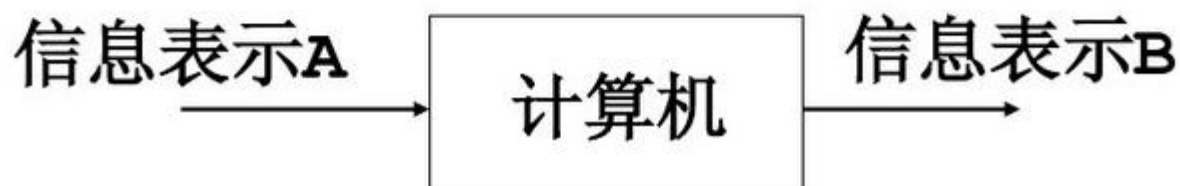
- 能够分析研究计算机加工的对象特性，获得其逻辑结构，根据需求，选择合适存储结构及其相应的算法；
- 学习一些常用的算法；
- 复杂程序设计的训练过程，要求编写的程序结构正确、清晰、易读；
- 初步掌握算法的时间分析和空间分析技术。

第1章 概论

- 实例介绍数据结构研究的内容
- 基本概念
- 抽象数据类型ADT
- 算法描述
- 算法分析
- 重点、难点
 - 数据、数据元素、数据结构等基本概念
 - 算法、算法描述、算法分析

数据结构讨论的范畴

为什么要学习数据结构？



电子计算机的主要用途：

- ☞ 早期：主要用于数值计算。
- ☞ 后来：处理逐渐扩大到非数值计算领域（能处理多种复杂的具有一定结构关系的数据）。

数值计算解决问题的一般步骤:

数学模型→选择计算机语言→编出程序→测试→最终解答。

数值计算的关键是：如何得出数学模型（方程）？

程序设计人员比较关注程序设计的技巧。

非数值计算问题:

数据元素之间的相互关系一般无法用数学方程加以描述，那什么是非数值问题呢？

例1 考生录取信息系统

考号	姓名	性别	报考专业	成绩
2300411	李闽志	男	计算机科学与技术	658
1000472	于惠芳	女	英语	632
1506302	刘 红	女	应用数学	617
2105902	宋大明	男	英语	600
0934785	高大庆	男	计算机科学与技术	601
0600807	何文丽	女	英语	611
0878529	隋文涛	男	应用数学	612
1690834	崔秀海	男	英语	602
1710641	于众群	女	计算机科学与技术	619
1900162	魏人民	男	英语	671



考生录取信息系统

- 计算机**处理**的**对象**是表
- **元素**间的**关系**是线性关系
- 施加于对象上的**操作**有查询、插入、删除等

例 2 : 书目自动检索系统

线性表

- 计算机处理的对象是表
- 元素间的关系是线性关系
- 施加于对象上的操作有查询、插入、删除等

书目卡片

登录号:
书名:
作者名:
分类号:
.....

001	高等数学	樊映川	S01
002	理论力学	罗远祥	L01
003	高等数学	华罗庚	S01
004	线性代数	栾汝书	S02
.....

书目文件

按书名

高等数学	001, 003.....
理论力学	002,
线性代数	004,
...	...

按作者名

樊映川	001,
华罗庚	002,
栾汝书	004,
...	...

索引表

按分类号

L	002,
S	001, 003.....
...	...

例3：人机对弈问题

Deep Blue的故事

计算机与人类两次真正意义上的正式的国际象棋比赛在美国举行。

对阵双方：

- 人类：国际象棋世界冠军俄罗斯的Kasparov
- 计算机：Deep Blue (“深蓝”)

	Game1	Game2	Game3	Game4	Game5	Game6
1996.2	Deep Blue wins	Kasparov wins	Draw	Draw	Kasparov wins	Kasparov wins
1997.5.3-5.11	Kasparov wins	Deep Blue wins	Draw	Draw	Draw	Deep Blue wins

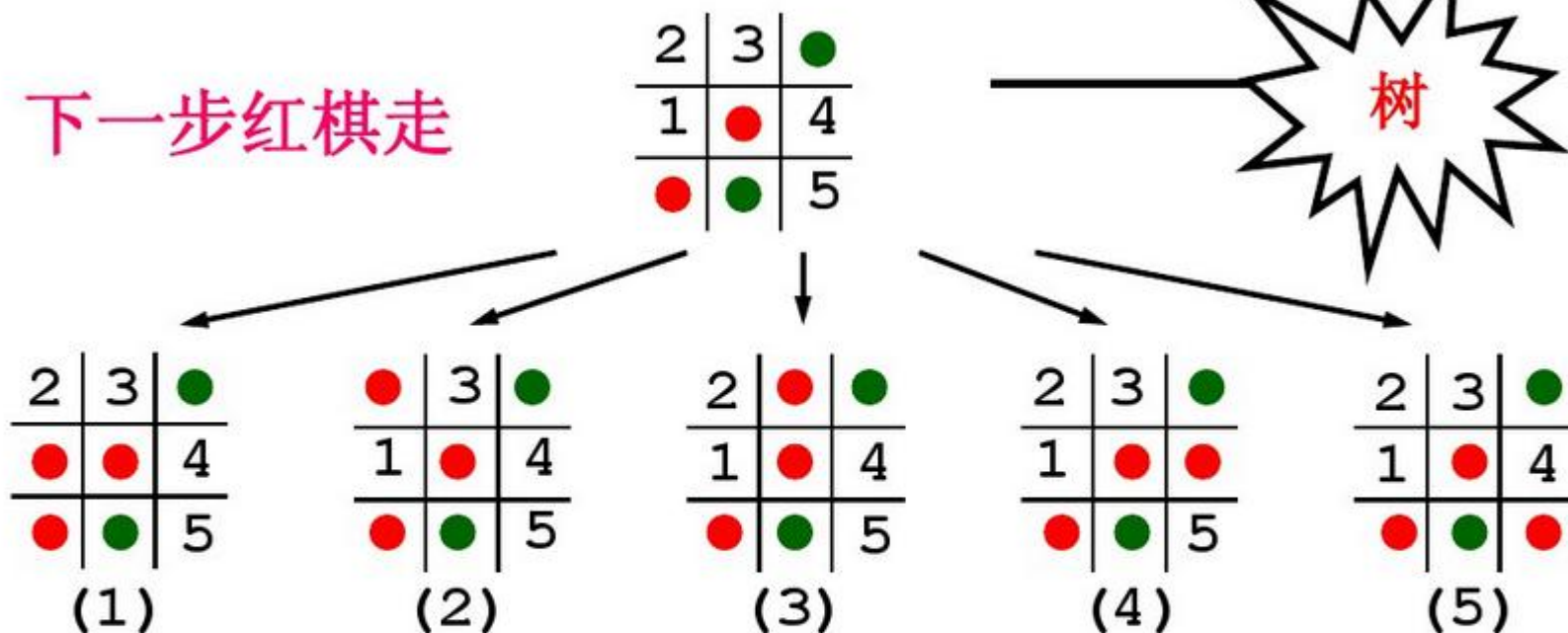
国际象棋

- Deep Blue是IBM耗资上千万美元，耗时8年建造起来的世界上最强大的会下国际下棋的计算机。本质上，Deep Blue是一个大规模并行的超级处理器的计算机系统。
- Deep Blue：2亿结点/秒，60万种棋局，评价函数有8000个参数；
- 计算机预见10-15步，心理学家认为，人类选手只能预测3-5步；
- 棋局中期计算机往后看75步。

例3: 人机对弈问题

算法: 对弈的规则和策略

下一步红棋走



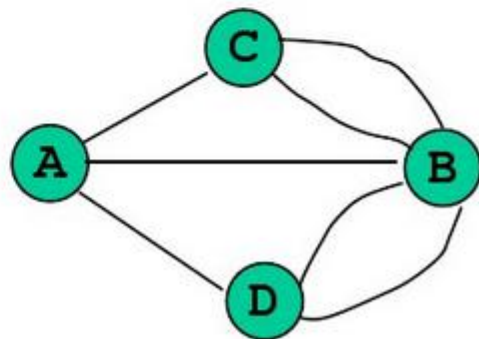
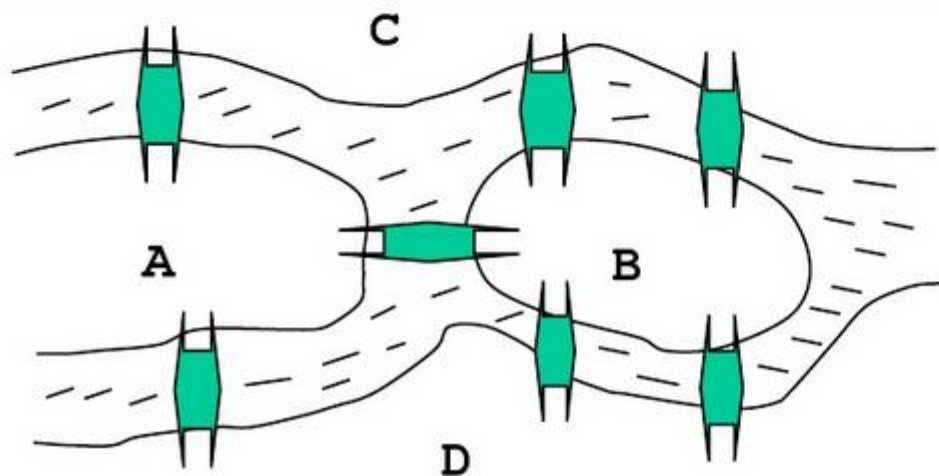
- 计算机处理的对象是树型结构
- 元素间的关系是层次关系
- 施加于对象上的操作有查询、插入、删除等

例4：哥尼斯堡七桥问题

问题：怎样才能够从某块陆地出发，经过每座桥一次且仅一次最后回到出发点。

图

- 计算机处理的对象是图
- 元素间的关系是复杂的图形或网状关系
- 施加于对象上的操作有查询、插入、删除等



求解非数值计算的问题：

主要考虑的是设计出合适的数据结构及相应的算法。

即：首先要考虑对相关的各种信息如何表示、组织和存储？

什么是数据结构？

数据结构是一门研究**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**的学科。

Q：学习数据结构有什么用？

答：计算机内的数值运算依靠方程式，而非数值运算（如表、树、图等）则要依靠数据结构。

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。

程序设计的实质是对实际问题选择一个好的数据结构，加之设计一个好的算法。而好的算法在很大程度上取决于描述实际问题的数据结构。

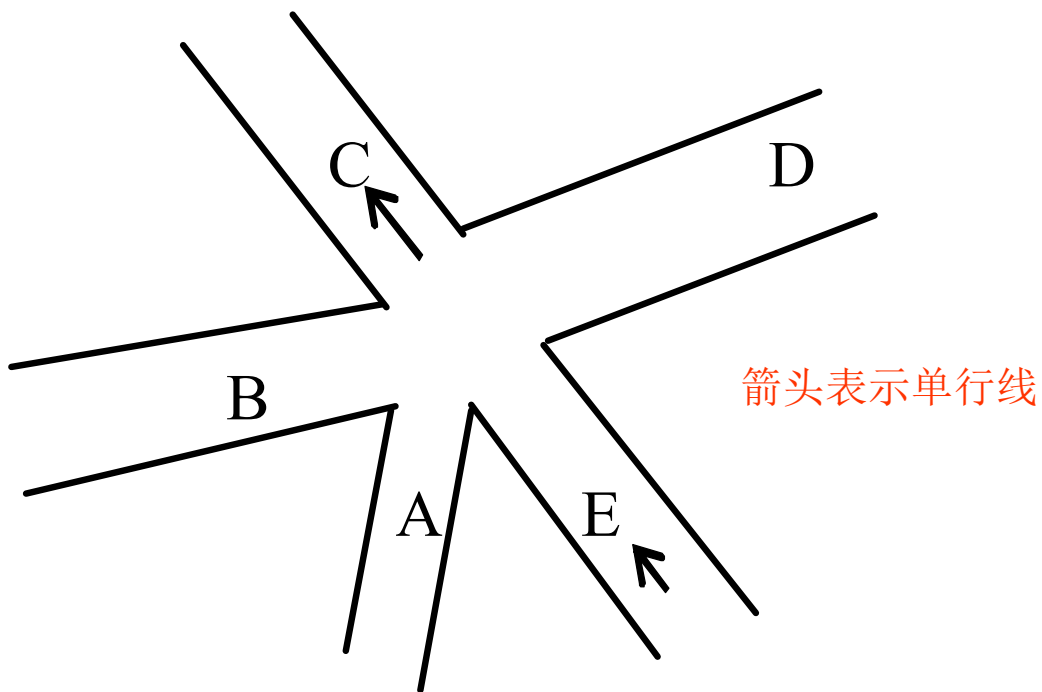
例 多叉路口交通灯的管理问题

问题： 为一个多叉路口设计信号灯管理系统。

因为：

不同行驶路线之间可能出现冲突。

解决思路： 需要对所有的可能行驶路线作某种分组，使每个组内各方向行驶的车辆可以同时安全行驶，不会发生阻挡或碰撞。

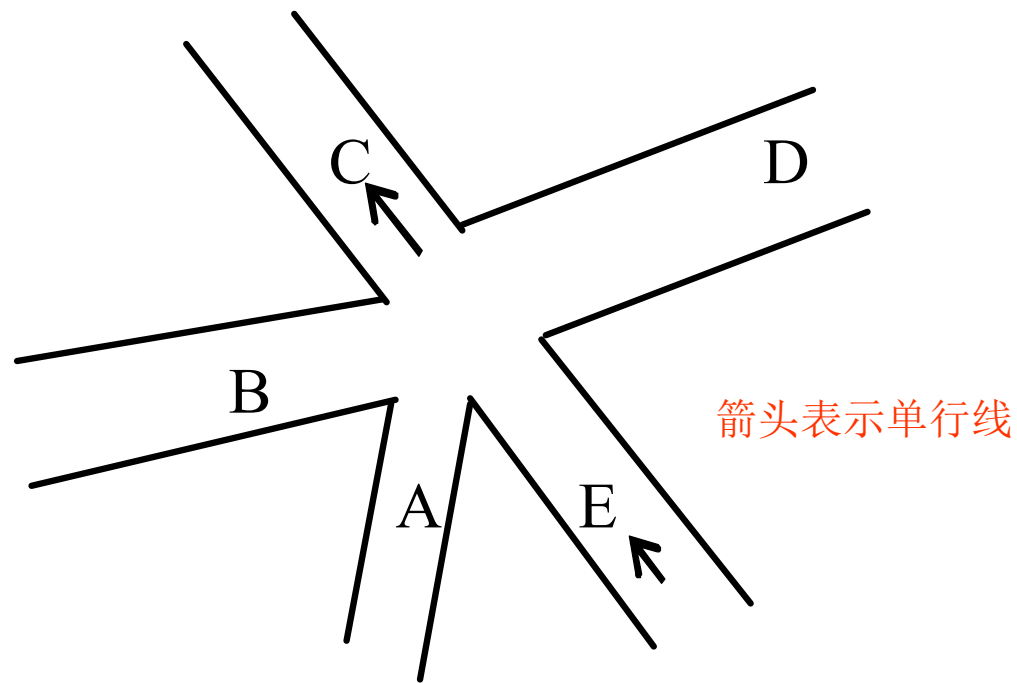


一个交叉路口的模型

1 问题分析

可通行方向

$A \rightarrow B$	$A \rightarrow C$	$A \rightarrow D$
$B \rightarrow A$	$B \rightarrow C$	$B \rightarrow D$
$D \rightarrow A$	$D \rightarrow B$	$D \rightarrow C$
$E \rightarrow A$	$E \rightarrow B$	$E \rightarrow C$
$E \rightarrow D$		



一个交叉路口的模型

有些通行方向不能同时行驶，在这样的结点间画一条连线。

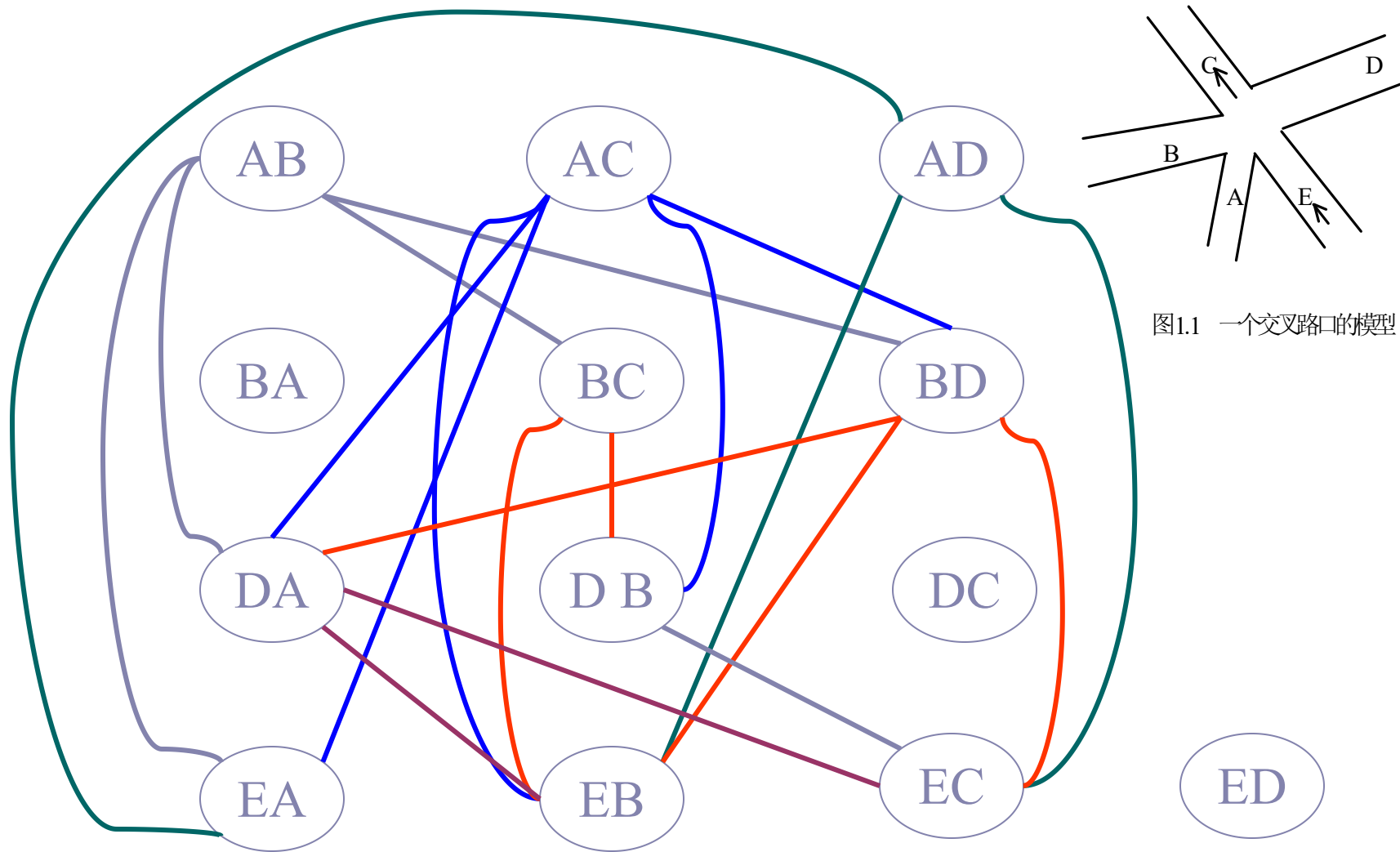


图1.1 一个交叉路口的模型

交叉路口的图示模型（冲突图）

应用地图着色的原理：为节点间没连线的点（即可以同时通行的点）标上同一种颜色

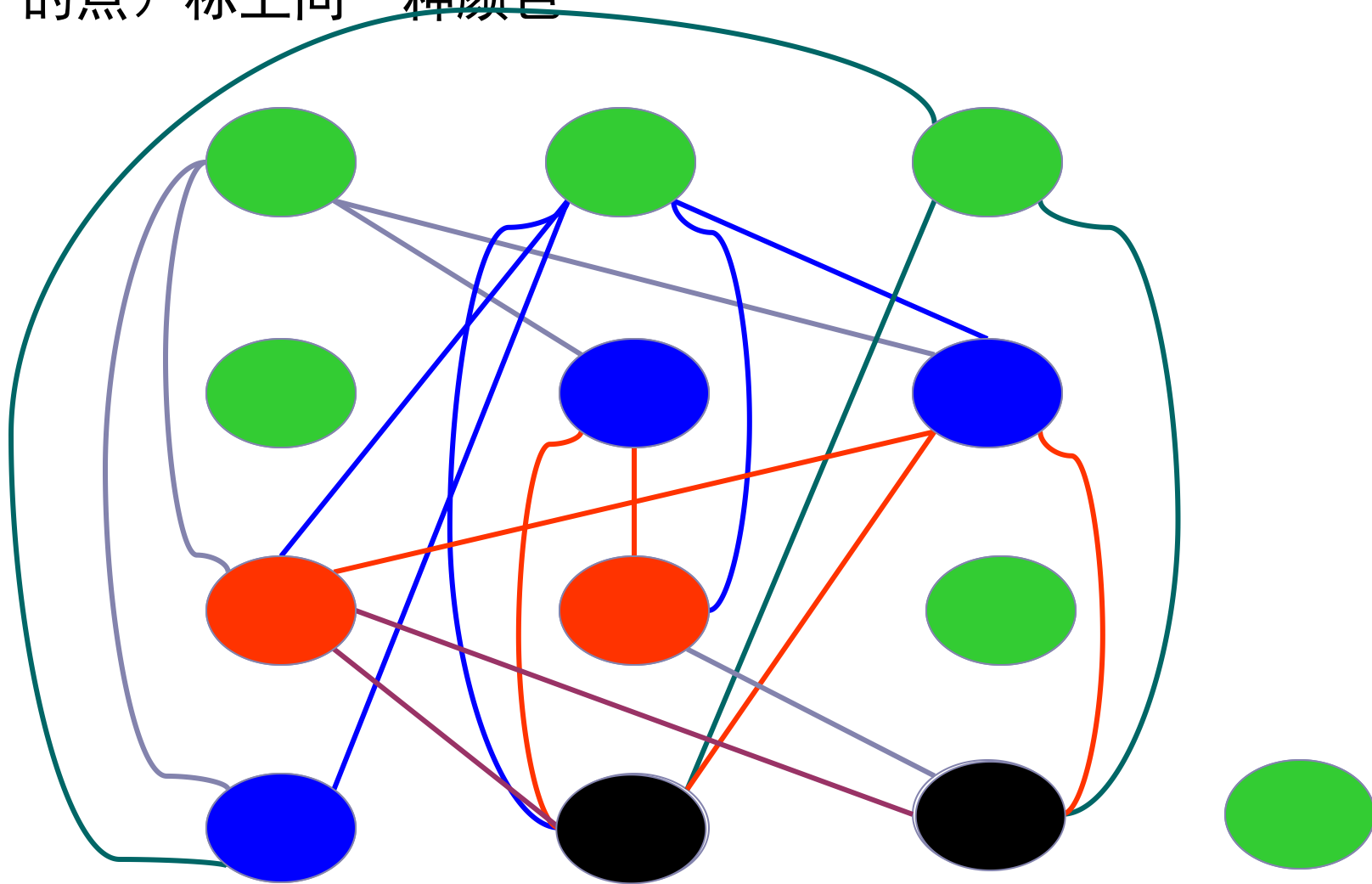


图1.2 交叉路口的图示模型



得到下面的分组：

绿色： AB, AC, AD, BA, DC, ED

蓝色： BC, BD, EA

红色： DA, DB

黑色： EB, EC

一、名词术语

数据

描述客观事物的数字、字符以及一切能够输入到计算机中，并且能够被计算机程序处理的**符号的集合**。包括文字、表格、图像等。

• 例如：

- 文字信息：整数{0,1,12,36}、实数、字母{'A', 'B', ... 'Z'}
- 声音、图形、图像

图书信息：

登录号	书名	借阅者编号
001	理论力学	9002
002	高等数学	9001
...

读者信息：

读者编号	姓名	所借图书登录号
9001	李红	002
9002	张小林	001
...

数据元素

数据这个集合中的一个一个的元素，是数据的基本单位，在计算机程序中通常作为一个**整体**进行考虑和处理。

• 例如：

一个数据元素

整数数据：{0, 1, 12, 36}

字母字符数据：{'A', 'B', ..., 'Z'}

图书信息：

登录号	书名	借阅者编号
001	理论力学	9002
002	高等数学	9001
...

一个数据元素
(一条记录)

数据项

- 数据的不可分割的最小单位
- 一个数据元素可由一个或多个数据项构成

3个数据项

一个数据元素
(一条记录)



登录号	书名	借阅者编号
001	理论力学	9002
002	高等数学	9001
...

数据对象

- 具有相同特性的数据元素的集合
- 是数据(全集)的子集

例:

整数数据对象 $A = \{0, 1, 12, 36\}$

字母字符数据对象 $C = \{'A', 'B', \dots 'Z'\}$

图书信息表 $G = \{(001, \text{理论力学}, 9002),$
 $(002, \text{高等数学}, 9001), \dots\}$

结构

- 数据元素之间具有的关系。(联系)

二. 数据结构的定义1

1. **数据结构**就是相互之间存在一种或多种特定关系的数据元素的集合。

2. **数据结构**是一个二元组

$$\text{Data_Structure} = (D, R)$$

其中, $D = \{d_i \mid 1 \leq i \leq n, n \geq 1\}$ 是数据元素的有限集合

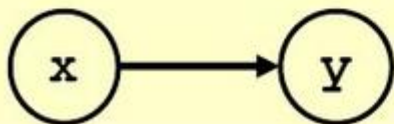
$R = \{r_j \mid 1 \leq j \leq m, m \geq 1\}$ 是D上的关系的集合

d_i 表示集合中第i个数据元素, r_j 表示集合中第j个关系。

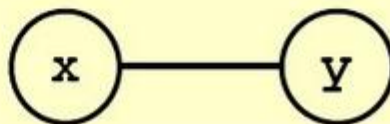
D上二元关系R是序偶的集合。对于R中的任一序偶 $\langle x, y \rangle$ ($x, y \in D$)， x 称为序偶的第一元素， y 称为序偶的第二元素，又称序偶的第一元素是第二元素的直接前驱；第二元素为第一个元素的直接后继。

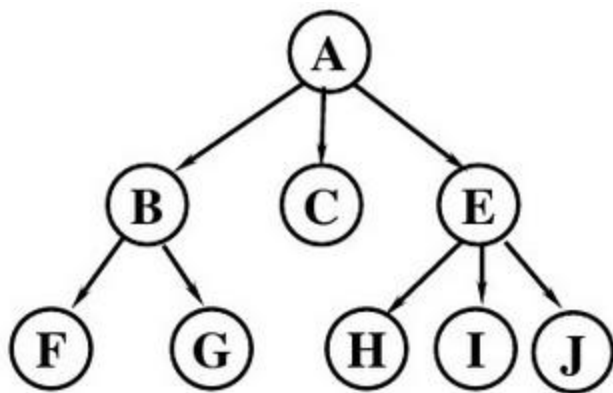
对称序偶用 (x, y) ($x, y \in D$) 来表示。

$\langle x, y \rangle$ 的逻辑图形为：



(x, y) 的逻辑图形为：





【例】树型结构中的圆圈代表数据元素，带箭头的连线表示元素之间的关系，试用二元组表示法表示其逻辑结构。

解：二元组为 (D, R) ，其中：

$D = \{A, B, C, E, F, G, H, I, J\};$

$R = \{ \langle A, B \rangle, \langle A, C \rangle, \langle A, E \rangle, \langle B, F \rangle, \langle B, G \rangle, \langle E, H \rangle, \langle E, I \rangle, \langle E, J \rangle \}$

• 例1

用图形表示下列数据结构，并指出它们是属于线性结构还是非线性结构。

$$S = (D, R)$$

$$D = \{ a, b, c, d, e, f \}$$

$$R = \{ (a, e), (b, c), (c, a), (e, f), (f, d) \}$$

解：上述表达式可用图形表示为：

$$b \longrightarrow c \longrightarrow a \longrightarrow e \longrightarrow f \longrightarrow d$$

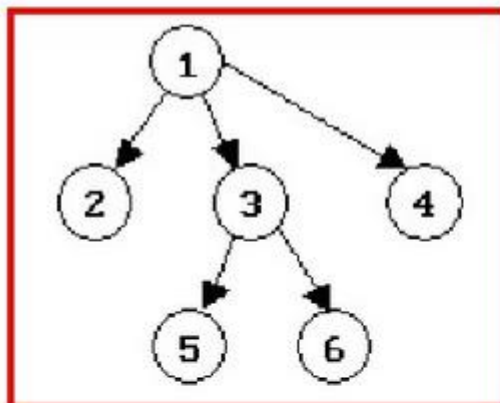
此结构为线性的。

• 例2

$D = \{1, 2, 3, 4, 5, 6\}$,

$R = \{ \langle 1, 2 \rangle, \langle 1, 3 \rangle, \langle 1, 4 \rangle, \langle 3, 5 \rangle, \langle 3, 6 \rangle \}$,

试画出它们对应的逻辑图形表示, 并指出它们属于何种逻辑结构。



分析:

题中数据元素间的关系是一对多的关系, 其中只有一个元素没有直接前驱, 其余元素有且仅有一个直接前驱, 而元素的直接后继可以有一或多个, 也可以没有。

树形结构

二. 数据结构的定义2

数据结构是按某种逻辑关系组织起来的一批数据应用计算机语言并按一定的存储表示方式把它们存储在计算机的存储器中，并在其上定义了一个运算的集合。

具体来说，数据结构包含三个方面的内容，即数据的**逻辑结构**，数据的**存储结构**和对数据所施加的**运算**（操作）。

这三个方面的关系为：

(1) 数据的逻辑结构独立于计算机，是数据本身所固有的。**（设计算法——构造数学模型）**

(2) 存储结构是逻辑结构在计算机存贮器中的映像，必须依赖于计算机。**（存储结构，算法的实现）**

(3) 运算是指所施加的一组操作总称。运算的定义直接依赖于逻辑结构，但运算的实现必须依赖于存贮结构。

逻辑结构的分类

数据的逻辑结构是本质，可以分为：
线性结构和**非线性结构**，
也可以分为

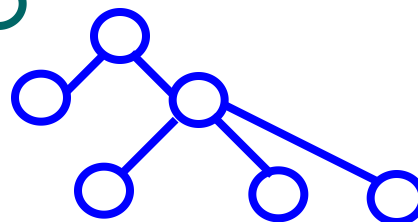
➤ 集合：



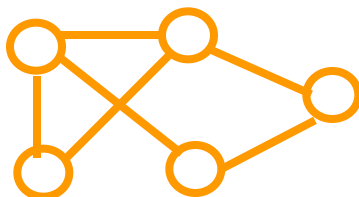
➤ 线性结构：



➤ 树形结构：

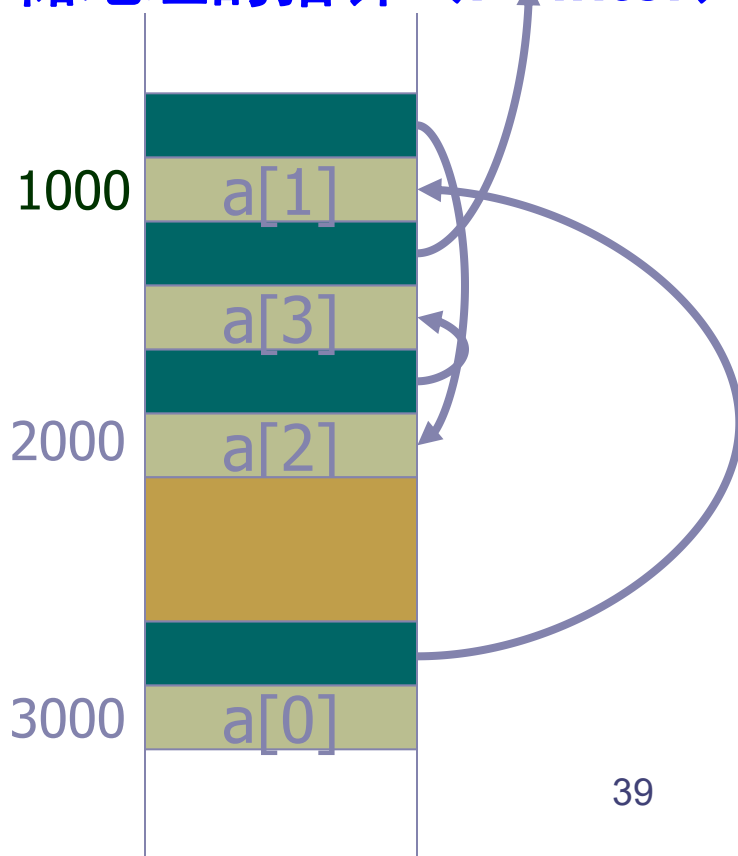
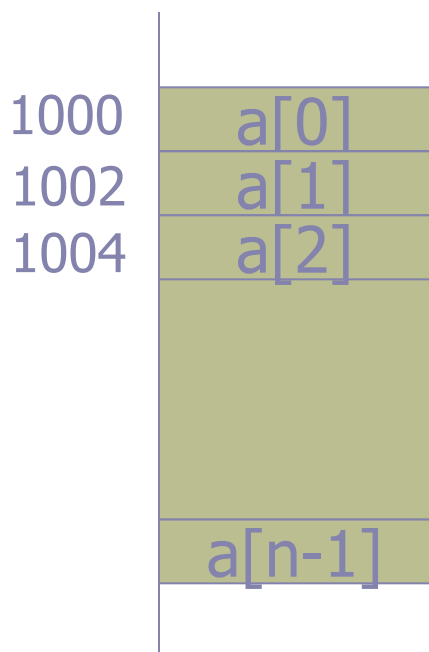


➤ 图状结构：



存储结构

- 存储结构(物理结构)指数据结构在计算机中的表示。
 - **顺序存储结构**：借助元素在存储器中的相对位置表示数据元素之间的关系。
 - **链式存储结构**：借助指示元素存储地址的指针（Pointer）表示数据元素之间的逻辑关系。





物理（存储）结构的分类

● 物理（存储）结构的分类

- 顺序存储结构：
- 链式存储结构：
- 索引存储结构：
- 散列存储结构



1.3 抽象数据类型


- 数据类型是一个值的集合和定义在该值上的一组操作的总称。
- 数据类型：
 - 简单类型（原子类型）
 - 结构类型：由简单类型数据按照一定的规则构造而成。如数组等。



抽象数据类型

- **抽象数据类型: (Abstract Data Type)**
ADT: 一个数学模型以及定义在该模型上的一组操作。

抽象数据类型的定义取决于它的一组逻辑特性，而与其在计算机内部如何表示和实现无关。即不论其内部结构如何变化，只要它的数学特性不变，都不影响其外部的使用。

- 
- 使用抽象数据类型可以更容易的描述现实世界。
 - 例如：
 - 用线性表抽象数据类型描述学生成绩表
 - 用树抽象数据类型描述遗传关系
 - 用图抽象数据类型描述城市道路交通图

抽象数据类型的定义

● 抽象数据类型的定义格式:

ADT 抽象数据类型名{

数据对象: {数据对象定义}

数据关系: {数据关系定义}

基本操作: {基本操作定义}

}ADT 抽象数据类型名

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

- 数据对象和数据关系的定义用伪码描述,
- 基本操作的定义格式为:

 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>

例.抽象数据类型"复数"的定义为:

ADT Complex {

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

数据关系: $R1 = \{\langle e1, e2 \rangle \mid e1 \text{ 是复数的实部, } e2 \text{ 是复数的虚部}\}$

基本操作:

InitComplex(**&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**)

初始条件: 复数已存在。

操作结果: 复数Z被销毁。

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



1.3 抽象数据类型的表示和实现（续）

二.抽象数据类型的表示：伪语言/类语言

类C语言：介于伪码和C语言之间

伪码：用常规语言或文字符号（即非编程语言）写的代码算法。

1. 借用程序设计语言的结构描述——简洁、严谨
2. 忽略程序设计语言的细节——简洁
3. 类C语言与C语言的区别：P10

类C语言：

- ① 数据结构的表示：typedef

格式：typedef <老类型> <新类型>

例：typedef int status

- ② 赋值：成组赋值、交换赋值
- ③ 选择语句：switch的扩展
- ④ 输入/输出：可以忽略格式化
- ⑤ 头文件、辅助变量定义：忽略
- ⑥ C的扩展：引入C++的引用参数来表示变参



1.4 算法和算法分析

- 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。
其中每一条指令表示一个或多个操作。

算法应该具有下列五个特性

- (1) **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。
- (2) **确定性**：算法中每一条指令必须有确切的含义。不存在二义性。即对于相同的输入只能得出相同的输出。
- (3) **可行性**：一个算法是可行的。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
- (4) **输入**：一个算法有零个或多个输入，这些输入取自于某个特定的对象集合。
- (5) **输出**：一个算法有一个或多个输出，这些输出同输入之间有着某些特定的关系。

例题：考虑下列两段描述

```
(1) void exam1 ()  
{ int n=2;  
  while (n%2==0)  
  { n+=2;  
    printf ("%d\n",n) ;  
  }  
}
```

```
(2) void exam2 ()  
{ int y=0;  
  int x=5/y;  
  printf ("%d,%d",x,y) ;  
}
```

以上两段算法是否能满足算法的特性，如不满足，则违反了那些特性？

答案：(1) 是死循环，违反了有穷性。

(2) 包含除零错误，违反了算法的可行性。



算法描述

- 算法可用多种方式描述：
 - 自然语言
 - 高级语言
 - 伪码语言
 - 框图
- 算法≠程序



算法≠程序

- **算法**是供人阅读的
- **程序**是让机器执行的
- **算法**用计算机语言实现时就是程序
- **程序**有时不需要具有算法的有穷性
- 在本课程中只讨论满足有穷性的程序，因此“算法”和“程序”是通用的。



“好” 算法的设计要求

● 正确性(correctness)

- 没有语法错误，对于所有合法的输入数据能够得出符合要求的结果。

● 可读性强(readability)

- 容易读懂和交流。便于调试和修改。

● 健壮性 (robustness)

- 算法不但对于合法的输入数据能够输出符合要求的结果，而且当输入数据非法时，算法也能适当地作出反应或进行处理，而不会产生莫明其妙的结果。

● 高效性(effectiveness)

- 算法应有效使用存储空间和有较高的时间效率。

算法效率的衡量方法

- 衡量算法效率的方法主要有两大类：
 - 事后统计：利用计算机的时钟；必须执行，其它因素掩盖算法本质。
 - 事前分析估算：用高级语言编写的程序运行的时间主要取决于如下因素：
 - 算法选用的策略；
 - 问题规模；
 - 编写程序的语言：级别越高，效率越低；
 - 编译程序所产生的目标代码的质量；
 - 机器执行指令的速度；

问题规模

- 随着处理问题的数据增大，处理会越来越困难复杂，把描述数据增大程度的量叫做**问题规模**。
- 规模越大，耗费时间越多
 - 求100以内的素数和求10000以内的素数的执行时间**显然不同**
 - 10×10 的两矩阵相乘和 1000×1000 的两矩阵相乘的执行时间**相差很远**

时间复杂度

- 算法的运行工作量的大小就只依赖于问题的规模（通常用正整数 n 表示），或者说它是问题规模 n 的函数，记 $T(n)$ ，称为该算法的时间复杂度。当问题规模 n 趋向无穷大时，时间复杂度 $T(n)$ 的数量级（阶）称为算法的渐近时间复杂度，记做 $O(f(n))$ ，它表示随问题规模 n 的增大，算法执行时间的增长率和函数 $f(n)$ 的增长率相同。
- 算法主要由程序的控制结构（顺序，分支，循环）和原操作（必须的操作）构成，算法的时间主要取决于两者。
- 通常做法：统计基本操作的执行次数

时间复杂度（举例）

➤ `++x;`

➤ `s=0;`

● 语句频度为1，时间复杂度为 $O(1)$ 。

➤ `for(j=1;j<=n;++j)`

➤ `for(k=1;k<=n;++k)`

➤ `{++x;s+=x;}`

● 语句频度为 $n \times n$ ，时间复杂度为 $O(n^2)$ 。



时间复杂度（举例续）

- `for(j=1;j<=n;++j)`
- `for(k=1;k<=j;++k)`
- `{++x;s+=x;}`

● 语句频度为近似于 n^2 ，所以时间复杂度仍为 $O(n^2)$ 。

- `s=0;`
- `for(j=1;j<=n;j*=2)`
- `for(k=1;k<=n;++k)`
- `{s++;}`

● 时间复杂度为 $O(n\log_2 n)$

时间复杂度（举例续）

● 例：矩阵加法： $n + n$

- `for(i = 0; i < n; i++)`
- `{for(j = 0; j < n; j++)`
- `c[i][j] = a[i][j] + b[i][j];`
- `}`

● 语句的频度：重复执行的次数： $n*n$;

- $T(n) = O(n^2)$
- 即：矩阵加法的运算量问题的规模 n 的平方是同一个量级；

时间复杂度（举例续）

● 例：矩阵乘法： $n \times n$

```
➤      for( i = 0; i < n; i++)           //(n+1)
➤      for( j = 0; j < n; j++)           //n(n+1)
➤      { c[i][j] = 0;                     //n²
➤        for( k= 0; k< n; j++)           // n²(n+1)
➤        c[i][j] = c[i][j]+a[i][k]* b[k][j]; // n³
➤      }
```

● 说明：各语句行后的数字是该语句重复执行的次数；

● 本算法时间复杂度为 $O(n^3)$

时间复杂度（举例续）

- $k = 0;$
- $\text{for}(i = 0; i < n; i++)$
- $\quad \text{for}(j = 0; j \leq i; j++)$
- $\quad \quad a[i][j] = ++k;$

● 执行次数： $n*(n+1)/2$

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

时间复杂度（举例续）

● 例：冒泡排序

```
➤ for( i = n - 1, change = TRUE; i >= 1 && change; --i)
➤     {change = FALSE;
➤       for( j = 0; j < i; ++j )
➤         if( a[j] > a[j+1] ) )
➤           { a[j] ↔ a[j+1];
➤             change = TRUE;
➤           }
➤     }
```

● 最佳情况：最差情况：平均情况

● 时间复杂度（最坏情况下的时间复杂度）



以下六种计算算法时间的多项式是最常用的。
其关系为：

$$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3)$$

指数时间的关系为：

$$O(2^n) < O(n!) < O(n^n)$$

当 n 取值很大时，指数时间算法和多项式时间算法在所需时间上非常悬殊。因此，只要有人能将现有指数时间算法中的任何一个算法化简为多项式时间算法，那就取得了一个伟大的成就。

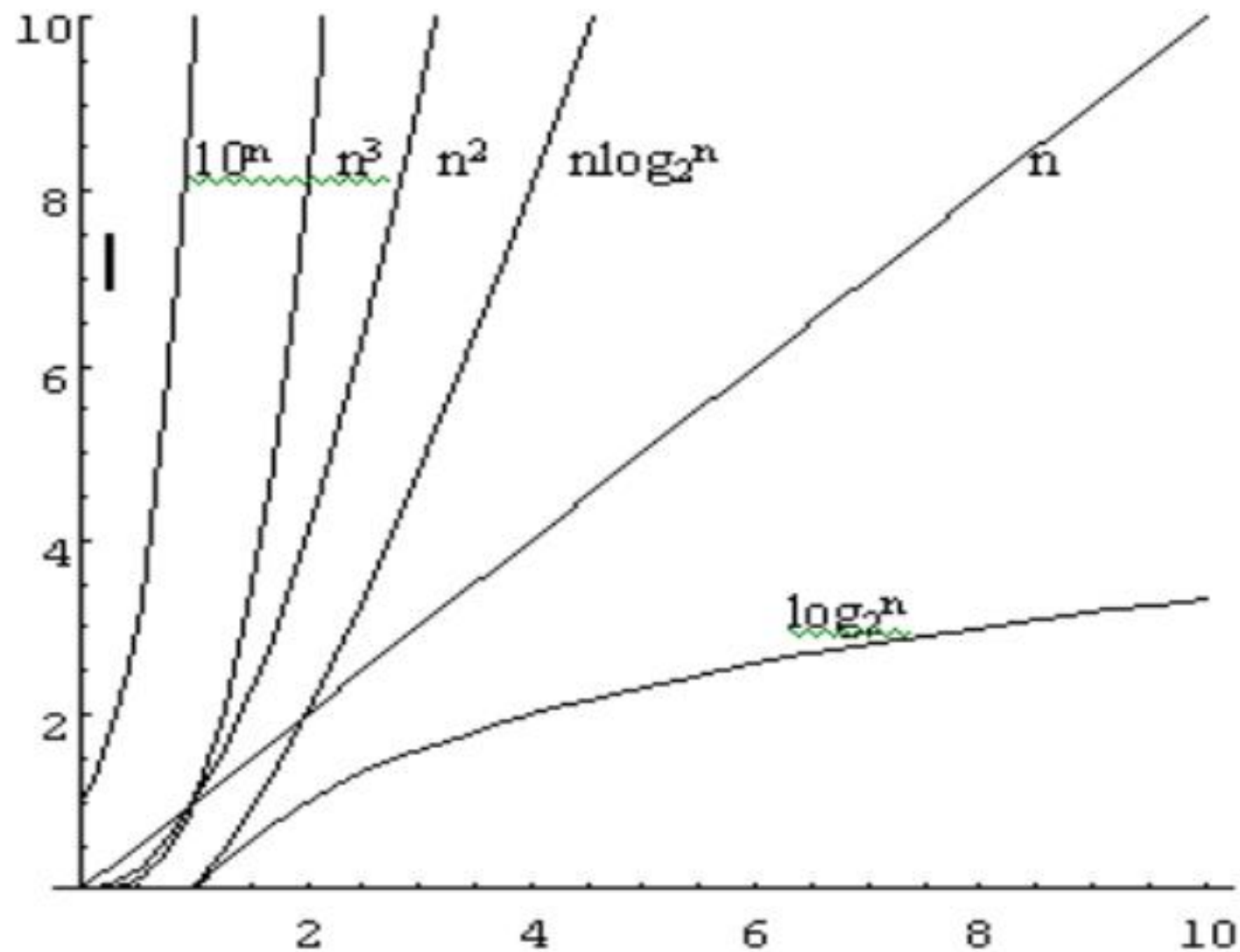


图 1.3 $f(n)$ 函数曲线变化速度的比较

算法的重要性

● 问题：百元买百笔。钢笔3元一支，圆珠笔2元一支，铅笔5角一支。给出解决方案。

● 方案1：

- `for(i = 0; i <=100; i++)`
- `for(j = 0; j <=100; j++)`
- `for(k= 0; k<=100; j++)`
- `if(i+j+k==100 &&3*i+2*j+0.5*k==100)`
- `printf(“i=%d, j=%d, k=%d”,i,j,k)`

算法的重要性(续)

● 方案2:

- `for(i = 0; i < =20; i++)`
- `for(j = 0; j < =34-i; j++)`
- `if(3*i+2*j+(100-i-j) *0.5==100)`
- `printf(“i=%dj=%dk=%d”,i,j, 100-i-j);`

● 方案1 内层循环超过100万次，在某机器上运行了50分钟；

● 方案2 的if语句执行525次，运行了2秒钟，相差1500倍。

算法的空间复杂度

● 算法的空间复杂度

- 算法的空间复杂度是指解决问题的算法在执行时所占用的存储空间。再具体一些，也就是我们编程序时，程序的存储空间、变量占用空间、系统堆栈的使用空间等等。也正由此，空间复杂度的度量分为两个部分：**固定部分和可变部分**。
- 若所用额外存储空间相对于输入数据量来说是常数，则称此算法为**原地（就地）工作**。

本章知识点和难点重点

● 知识点

- 基本概念
- 算法描述
- 类C语言

● 重点、难点

- 数据、数据元素、数据结构等基本概念
- 抽象数据类型
- 算法、算法描述、算法分析



本章学习目标

- (1) 领会数据、数据元素、数据项、数据结构、数据类型等概念及其相互关系；
- (2) 清楚数据结构的逻辑结构、存储结构、数据的运算的含义及相互关系；
- (3) 理解抽象数据类型的概念；
- (4) 掌握进行简单算法分析的方法。

练习

1.

```
for(i=1;i<=n;i++)  
    for(j=1;j<=i;j++)  
        s++;
```
2.

```
for(i=1;i<=n;i++)  
    for(j=i;j<=n;j++)  
        s++;
```
3.

```
i=0;s=0;  
while(s<n)  
{ i++;s=s+i;}
```