数据结构
(Data Structure)

2023-2024学年第一学期

2023.9

教学内容

- 第一章　绪论
- 第二章　线性表
- 第三章　栈和队列
- 第四章　串
- 第五章　数组和广义表
- 第六章　树和二叉树
- 第七章　图
- 第九章　查找
- 第十章　内部排序

注：教材中加 * *的章节一般不讲

第1章 绪论

1.1 什么是数据结构
1.2 基本概念和术语
1.3 抽象数据类型
1.4 算法和算法分析

1.1 什么是数据结构

1.1 什么是数据结构

- 用计算机解决问题：

  问题→抽象出数学模型→求模型的解

  两类问题：数值问题、非数值问题

  - 数值问题

  --程序处理的是纯粹的数值计算问题，其数学模型主要是数学方程。例如：用微分方程预测人口增长情况，求非线性方程的解等。

  - 非数值问题

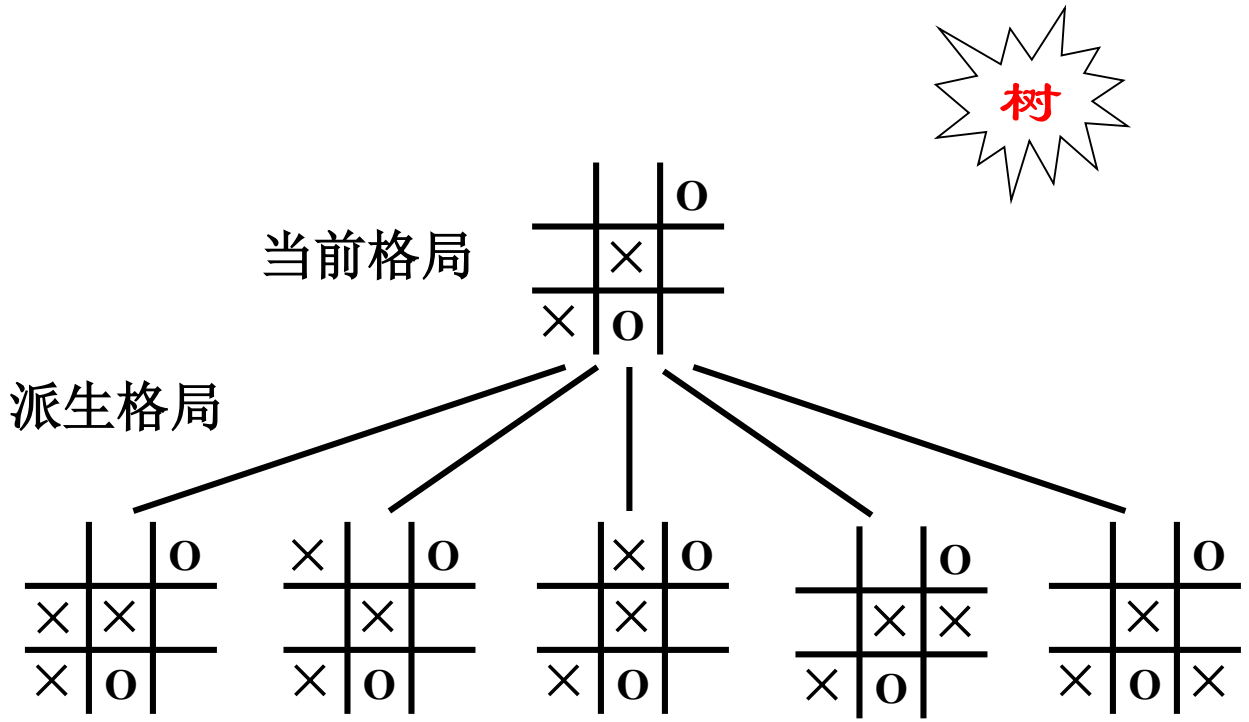
  --处理的对象是字符、表格、图像等各种具有一定结构关系的数据，其数学模型是表、树、图等数据结构。

1.1 什么是数据结构

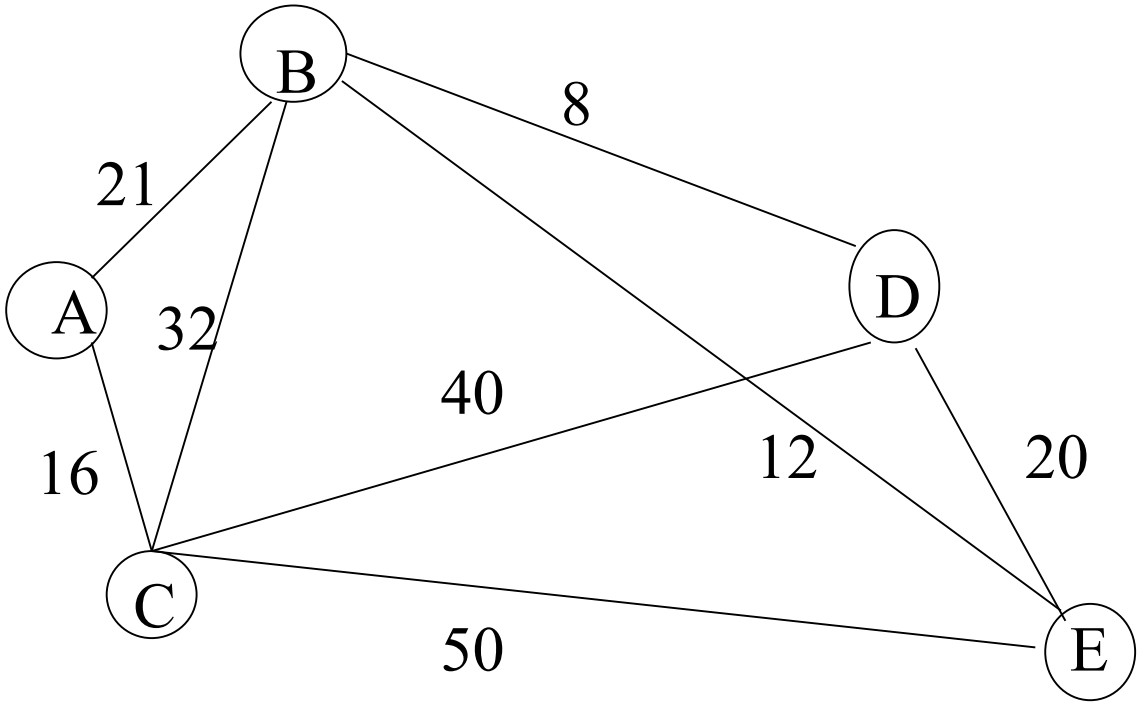
- 例1、编写一个程序，进行高精度整数的加、减、乘运算。输入：两个十进制大整数M和N（M、N最长可达200位），输出：计算结果。
  - 存在一个高精度整数如何表示的问题。

- 例2、交叉路口的红绿灯管理。如今十字路口横竖两个方向都有三个红绿灯，分别控制左拐、直行和右拐，那么如何控制这些红绿灯既使交通不堵塞，又使流量最大呢?
  - 若要编制程序解决问题，首先要解决一个如何表示研究对象的问题。

- 例3、计算机和人对奕问题
- 数据的逻辑结构：表示棋局之间的演化关系



- 例4、煤气管道的铺设问题。
- 如需为城市的各小区之间铺设煤气管道，对 n 个小区只需铺设 n-1 条管线，由于地理环境不同等因素使各条管线所需投资不同(如图上所标识)，如何使投资成本最低？这是一个讨论图的生成树的问题。



■例5、图书馆的书目检索系统自动化问题

- 建立一张按登录号（关键字）顺序排列的书目文件和一些按书名、作者名和分类号顺序排列的索引表。
- 由这4张表构成的文件便是书目自动检索的数学模型。计算机的主要操作按照某个特定要求对书目文件进行查询

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	…
⋮	⋮	⋮	⋮	⋮

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

樊映川	001,…
华罗庚	003,…
栾汝书	004,…
⋮	

L	002,…
S	001,003,…
⋮	

1.1 什么是数据结构

- 以上例子表明，数据结构是便于有效存储和操作的组织数据的方式
- 数据结构起源于程序设计，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及它们之间的关系和操作的学科。
- “数据结构”的概念最早是由 C. A. R. Hoare 和 N. Wirth 在 1966 年提出。大量关于程序设计理论的研究表明：为了系统而科学地构造大型复杂的程序，必须对这些程序中所包含的数据结构进行深入的研究。

- 1968年D.E.Knuth教授开创了数据结构课程的最初体系，《计算机程序设计技巧》第一卷《基本算法》首次系统地研究并整理了当时经常使用的主要数据结构与相关的算法，为数据结构课程的开设提供了丰富的素材。
- 我国1980’年代初，开设《数据结构》。
- 数据结构是程序设计的基础，是实现编译程序、操作系统、数据库系统等系统程序和大型应用程序的重要基础。
- 是介于数学、计算机硬件和计算机软件之间的一门核心专业基础课。

1.2 基本概念和术语

1.基本术语

- （1）数据：描述客观事物的数字、字符以及所有能输入到计算机中并被计算机程序处理的符号的集合。（数字、字符、声音、图形、图像等等）
- （2）数据元素：数据的基本单位，在计算机程序中常常作为一个整体进行考虑和处理，如记录/结构。
- （3）数据项：数据的不可分割的最小单位，如结构中的域。
- （4）数据对象：性质相同的数据元素的集合，是数据的一个子集。

■例如，

学号	姓名	班号	性别	出生日期	入学成绩
001	刘影	01	女	19840417	622
002	李恒	01	男	19831211	679
003	陈诚	02	男	19840910	638
004	…	…	…	…	…

数据元素

整个表是学生成绩数据对象

1.2 基本概念和术语

2. 数据结构

- 是相互之间存在一种或多种特定关系的数据集合。
- 是按照逻辑关系组织起来的一批数据，按一定的存储方法把它存储在计算机中，并在这些数据上定义了一个运算的集合。
- 数据结构的三个方面：
  - 逻辑结构：数据元素之间的逻辑关系，与计算机的存储无关。
  - 存储结构：数据结构在计算机中的存储表示（或称映像），又称物理结构。它包括数据元素的表示和关系的表示。
  - 运算：不同的数据结构其操作集不同

1.2 基本概念和术语

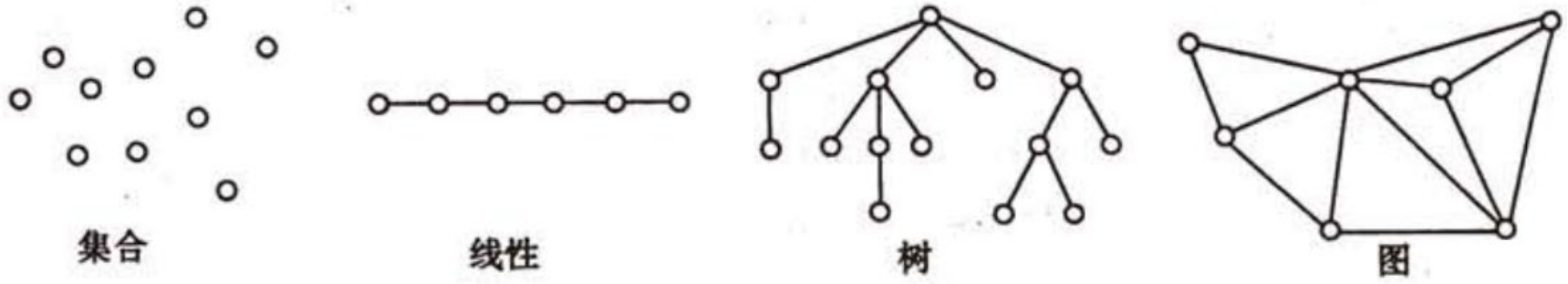
- 逻辑结构：数据元素之间的关系
  - 4种基本逻辑结构：元素之间关系的不同特性

（1）线性结构：存在一对一的关系，序列相邻，次序关系。

（2）树型结构：存在一对多发关系，层次关系。

（3）图状结构（网状结构）：存在多对多的关系。

（4）集合结构：元素属于同一个集合，无其他关系。



1.2 基本概念和术语

- 数据结构可用一个二元组表示：

DS=(D,S)

其中：D是数据元素的有限集合，

S是D上关系的有限集合。

例如：线性结构的二元组表示形式：

DS=（D，R）

D={a_i | a_i ∈ ElemSet，i=1,2,…,n,n>0}

R={<a_{i-1}, a_i> | a_{i-1}, a_i ∈D，i=2,..,n,n>0}

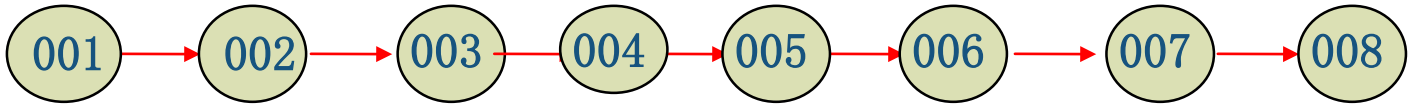
例1：以下是一个学生情况表的二元组表示（D,R），画出该数据结构，指出其逻辑结构是否是线性结构。

S=(D,R)

D = { 001，002，003，004，005，006，007，008}

R= {<001,002>,<002,003>,<003,004>,<004,005>,<005,006>,
<006,007>,<007,008> }

解：逻辑结构是线性结构



例2：以下是一个家庭成员的二元组表示（D,R），画出该数据结构，指出其是哪一种逻辑结构。

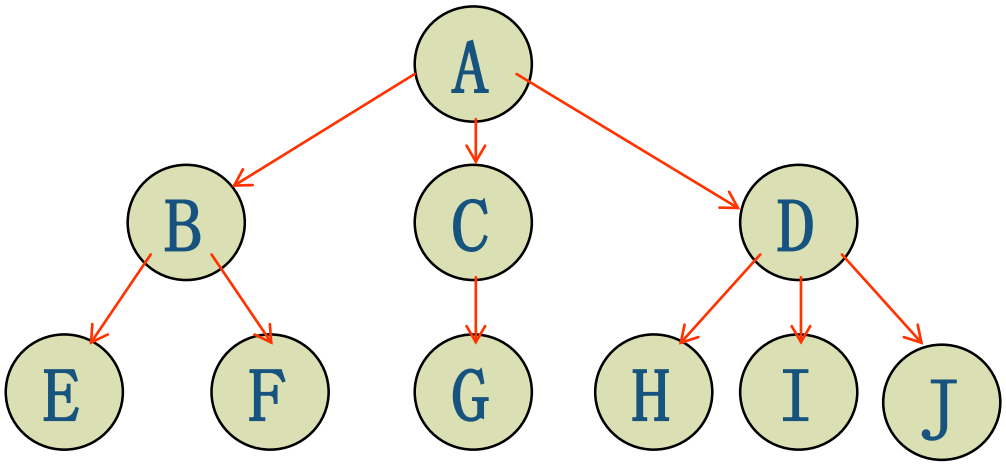
S=(D,R)

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

R =

{<A,B>,<A,C>,<A,D>,<B,E>,<B,F>,<C,G>,<D,H>,<D,I>,<D,J> }

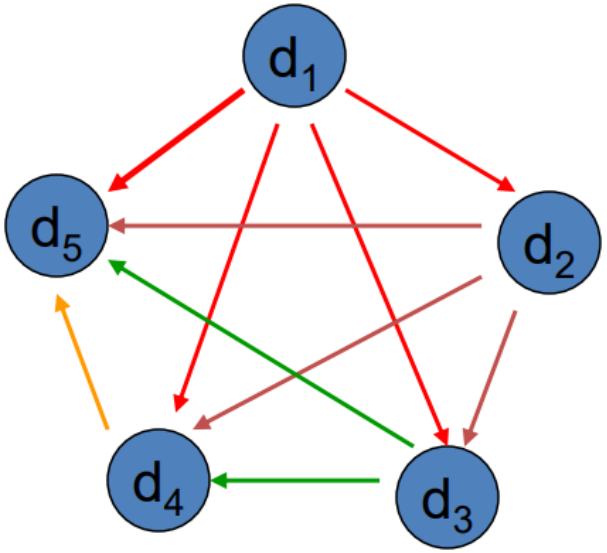
解：逻辑结构是树状结构



例3：已知以下数据结构的二元组表示（D,R），画出该数据结构，指出其是哪一种逻辑结构。

S = (D,R)
D = { d_i | 1≤i≤5}
R = { ⟨d_i, d_j⟩, i<j }

解：图状结构



例4：给出复数的数据结构的二元组表示。

解：定义如下： Complex=(D，R)

其中：D是含两个实数的集合D=｛C1，C2｝，分别表示复数的实部和虚部。R={P}，P是定义在集合上的一种关系={〈C1，C2〉}。

例5：编制一个事务管理程序，管理学校科研小组的各项事务。为科研小组设计一个数据结构。假设每个小组由一位教师、1~3名研究生及1~6名本科生组成，小组成员之间的关系是：教师指导研究生，每位研究生指导1~2名本科生。

解： 数据结构定义：Group=(P，R)

其中：$P=\{T，G_1，\ldots，G_n，S_{11}，\ldots，S_{nm}\}_{1\le n\le 3，\ 1\le m\le 2}$，$R=\{R_1，\ R_2\}$，$R_1=\{<T，G_i>\mid_{1\le i\le n，\ 1\le n\le 3}\}$，$R_2=\{<G_i，S_{ij}>\mid 1\le i\le n，\ 1\le j\le m，1\le n\le 3，\ 1\le m\le 2\}$

例6：以下计算机系人事表，根据不同的关系，可设计出多种数据结构。

工号	职务	教研室	工作时间	发表论文
01	系主任	软件	1981.1	A,B
02	教研室主任	软件	1985.1	B,C,E,F
03	教师	软件	1990.8	C,D
04	教师	应用	1987.8	A,G
05	教师	应用	1975.9	E,I
06	教师	应用	1992.2	F,J
07	教师	软件	1983.8	D,L
08	教研室主任	应用	1986.7	G,H
09	教师	应用	1995.8	H,I,J,K
10	教师	软件	1989.2	L,K

- D = { 01,02,03,04,05,06,07,08,09,10}

- 按照工作时间排列，形成一个线性结构：
  R={<05,01>, <01,07>, <07,02>, <02,08>,
  <08,04>, <04,10>, <10,03>, <03,06>,<06,09>}
- 需要表示上下级关系，形成树型结构：
  R={<01,02>,<01,08>,<02,03>,<02,10>,<02,07>,
  <08,04>,<08,05>,<08,06>,<08,09>}
- 需要表示论文合作情况，图状结构：
  R={(01,02),(01,04),(02,05),(02,06),(02,03),
  (03,07),(04,08),(05,09),(06,09),(07,10),(08
  ,09),(09,10)}

1.2 基本概念和术语

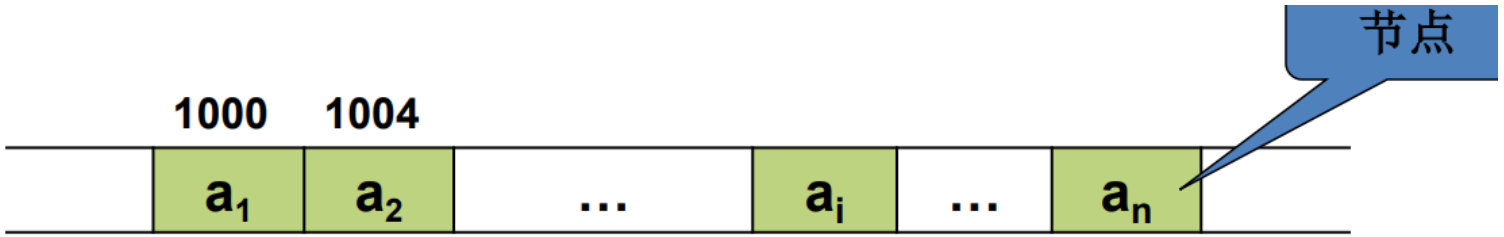
- 存储结构：逻辑结构到存储器的一个映像。
  - 存储器模型：

    一个存储器M是一系列固定大小的存储单元，每个单元U有一个唯一的地址A(U)，该地址被连续地编码。每个单元U有一个唯一的后继单元U’ =succ(U)
  - 数据元素的表示：节点

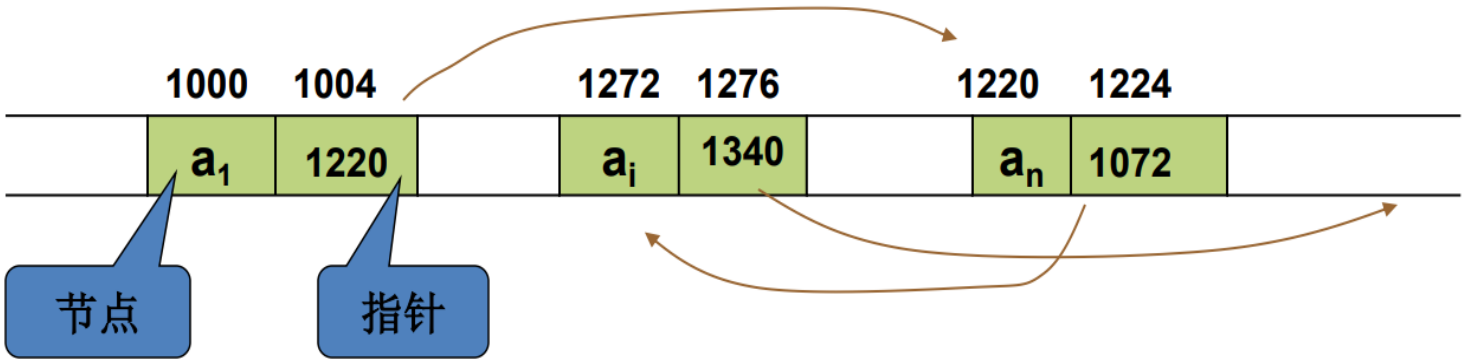
    二进制位串 数据类型
  - 关系的表示：
    - 顺序存储：物理的相对位置表示关系
    - 链式存储：用附加信息（指针）表示关系

顺序存储和链式存储示意图

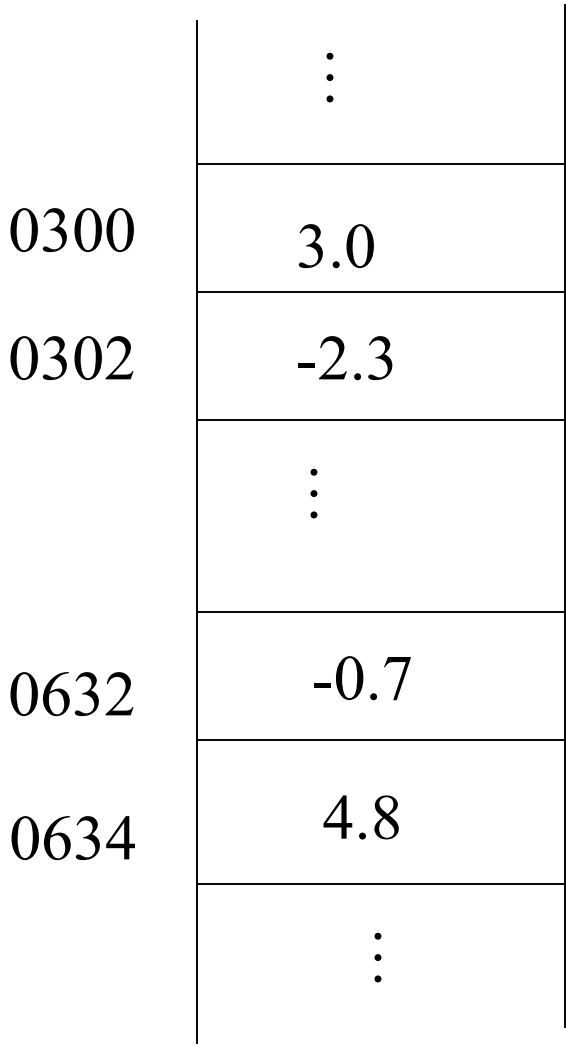
- （1）顺序存储：数据元素依次存放在连续的存储单元中。



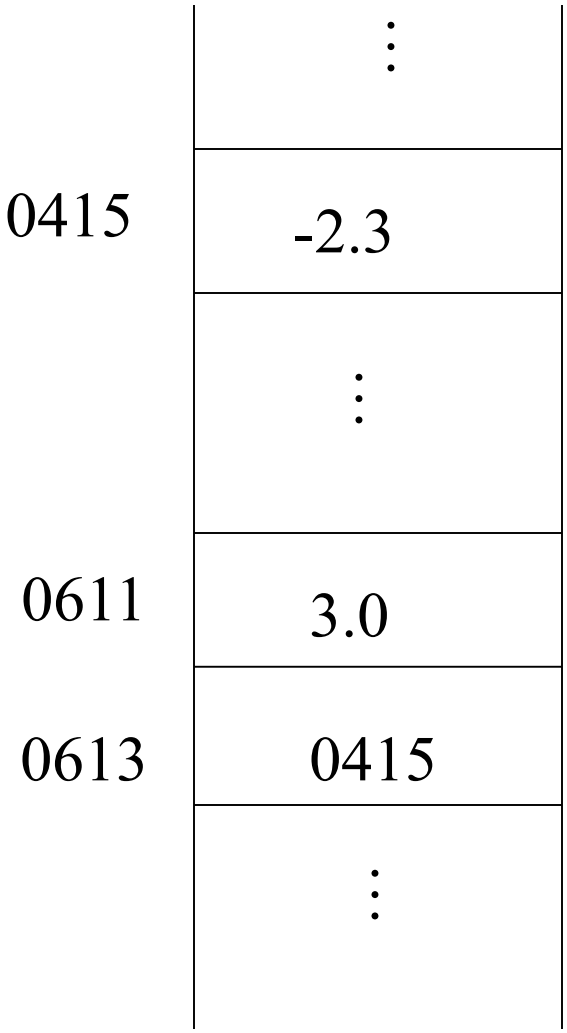
- （2）链式存储：在存储节点中增加指针域，记录后继节点的存储地址（指针）。



例：复数z1=3.0-2.3i和z2=-0.7+4.8i 的存储结构示意图



顺序存储结构



链式存储结构

1.2 基本概念和术语

- 数据结构的运算
  - 运算是施加于数据上的操作，如查找、添加、修改、删除等。主要运算包括：
    - (1) 建立(Create)一个数据结构；
    - (2) 消除(Destroy)一个数据结构；
    - (3) 从一个数据结构中删除(Delete)一个数据元素；
    - (4) 把一个数据元素插入(Insert)到一个数据结构中；
    - (5) 对一个数据结构中的数据元素进行访问(Access)；
    - (6) 对一个数据结构中的数据元素进行修改(Modify)
    - (7) 对一个数据结构进行排序(Sort)；
    - (8) 对一个数据结构进行查找(Search)。
- 运算定义在逻辑结构上，而实现在存储结构上

数据结构的三个方面：

- 数据的逻辑结构
  - 线性结构
    - 线性表
    - 栈
    - 队
    - 串
    - 数组
    - 广义表
  - 非线性结构
    - 树型结构
    - 图状结构
    - 集　　合
- 数据的存储结构
  - 顺序存储
  - 链式存储
- 数据的运算：检索、排序、插入、删除、修改等

1.2 基本概念和术语

3. 数据结构的划分

- （1）按性质划分：逻辑结构和物理结构
- （2）按存储划分：顺序存储结构和链式存储结构
- （3）按操作划分：静态结构、半静态结构、动态结构

1.2 基本概念和术语

- （1）按性质划分：逻辑结构和物理结构

例题：以下哪种结构是逻辑结构，而与存储和运算无关：（　　）
A.双向链表　B.数组　C. 队列　D.顺序表

下列数据结构都是逻辑结构：
　　线性表、栈、队列、串、树、图、集合
下列数据结构与存储和运算有关：
　　链表（单向链表、双向链表、循环链表）、顺序表、顺序栈、链栈、循环队列、链队列、二叉排序树、B-树，等等。

1.2 基本概念和术语

- （2）按存储方式来划分
  - 顺序存储结构——借助元素在存储器的相对位置来表示数据元素之间的逻辑关系。
  - 链式存储结构——借助指示元素存储地址的指针表示数据元素之间的逻辑关系。
  - 索引存储方法——在存储结点的同时，还建立附加的索引表，索引表中的每一项称为索引项，形式为：关键字，地址。
  - 散列存储方法——根据结点的关键字直接计算出该结点的存储地址。

说明：四种存储方法可结合起来对数据结构进行存储映像。

1.2 基本概念和术语

- （3）按数据结构的操作来划分
  - 静态结构——经过操作后，数据的结构特征保持不变（如数组）。
  - 半静态结构——经过操作后，数据的结构特性只允许很小变迁（如栈、队列）。
  - 动态结构——经过操作后，数据的结构特性变化比较灵活，可随机地重新组织结构（如指针）。

1.3 抽象数据类型的表示和实现

1.3 抽象数据类型的表示和实现

- 数据类型
  - 定义：是一个“值”的集合和定义在这个值集上的“一组操作”的总称。
  - C语言的数据类型分为原子类型和构造类型。
  - **数据类型**是对数据对象的**抽象**
  - **数据结构**不同于数据类型，也不同于数据对象，它不仅要描述数据对象，而且要描述数据对象各元素之间的相互关系。

1.3 抽象数据类型的表示和实现

- 抽象数据类型：
  - 定义：指基于一个逻辑类型的数据模型以及定义在该模型上的一组操作。
  - 例如，矩阵的抽象数据类型定义为，矩阵是一个由 m * n 个数排成 m 行 n 列的表，它可以进行初等变换、相加、相乘、求逆、……等运算
  - 形式化表示：三元组(D,S,P) ，D是数据对象，S是D上的关系集，P是对D的基本操作集。

例：三元组的抽象数据类型定义

- ADT Triplet{
  - **数据对象**：D={e1,e2,e3| e1,e2,e3 ∈ ElemSet}
  - **数据关系**：S={<e1,e2>,<e2,e3>}
  - **基本操作**：
    - InitTriplet(&T,v1,v2,v3)
      操作结果：构造了三元组T，e1,e2,e3分别等于v1,v2,v3的值
    - DestroyTriplet(&T)
      操作结果：三元组T被销毁
    - Get(T,i,&e)
      初始条件：三元组T已存在，1<=i<=3。
      操作结果：用e返回T的第i元的值。
    - 。。。。。。
- }

1.3 抽象数据类型的表示和实现

- 一个含抽象数据类型的软件模块，通常应包含：定义、表示、实现3个部分。
- 抽象数据类型的表示和实现
  - 抽象数据类型可通过固有数据类型来表示和实现。

例：三元组的抽象数据类型的表示和实现

- //--采用动态分配的顺序存储结构--
- typedef ElemType *Triplet; //
- //--基本操作的函数原型说明
- Status InitTriplet(Triplet &T,ElemType v1,ElemType v2,ElemType v3);//
- 。。。

1.3 抽象数据类型的表示和实现

- 抽象数据类型的作用：
  - 是模块化思想的发展，隐藏了运算实现的细节和内部数据结构，便于软件复用
- 对抽象数据类型的理解：
  - 抽象数据类型（ADT）指描述数据模型和操作的方法不依赖于具体实现
  - ADT强调数据抽象与过程抽象,将数据的表示与实现隐藏
  - 操作以一个严格定义的过程接口的方式提供,每一个操作由输入和输出定义，与算法编程语言均无关。
  - 数据结构是ADT的物理实现。

1.4 算法和算法分析

1.4 算法和算法分析

1.4.1 算法

是对特定问题求解步骤的一种描述，指一系列**确定的**而且是在**有限步骤**内能完成的操作。

例如：求两个正整数 m，n 的最大公因数

欧几里得算法（Euclid’s algorithm）：

- step 1. 求余数Divide m by n and r be the remainder.(We will have 0<=r<n)
- step 2. 是0？ if r==0,the algorithm terminateds; n is the answer
- step 3. Reduce. Set m←n,n←r, and go back to step 1

1.4 算法和算法分析

- 算法的特性
  - (1)**有穷性**： 能在执行有穷步后结束
  - (2)**确定性**： 每条指令有确切的含义，对于相同的输入执行相同的路径，得到相同的输出
  - (3)**有效性**(可行性)：用于描述算法的操作都是足够基本的，即：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** 0至多个输入
  - (5)**输出** 1至多个输出

- 思考：算法是不是等于程序?

算法与程序的区别

- 算法是解决问题的一种方法或一个过程，考虑如何将输入转化成输出，一个问题可以有多种算法
- 程序是用某种程序设计语言对算法的具体实现。（指令的序列）
- 主要区别：有穷性、正确性和描述方法
  - 程序可以是无穷的，例如OS
  - 程序可以是错误的，但算法必须是正确的，作为指导程序的编写依据
  - 程序是用程序设计语言描述，在机器上可以执行；算法可以用框图、自然语言等方式描述

- 算法的描述----本书采用类C语言
  - 所有算法均以函数（或过程）的形式表示
    - 返回类型 函数名（参数表）
    - {//算法说明;
    - 语句体;
    - return 返回值;
    - }
  - 算法的输入数据来自参数表;
  - 参数表的参数在算法之前均进行类型说明;
  - 有关结点结构的类型定义,以及全局变量的说明等均在算法之前进行说明。

- 数据结构的表示(存储结构)都用类型定义(typedef) 的方式描述。
  - 基本数据元素类型约定为 ElemType，由用户在使用该数据类型时再自行具体定义。
  - 除了函数的参数需要说明类型外，算法中使用的辅助变量可以不作变量说明，必要时对其作用给予注释。
- 语句序列中仅包含C语言的三种基本结构：顺序结构、选择结构和循环结构

1.4 算法和算法分析

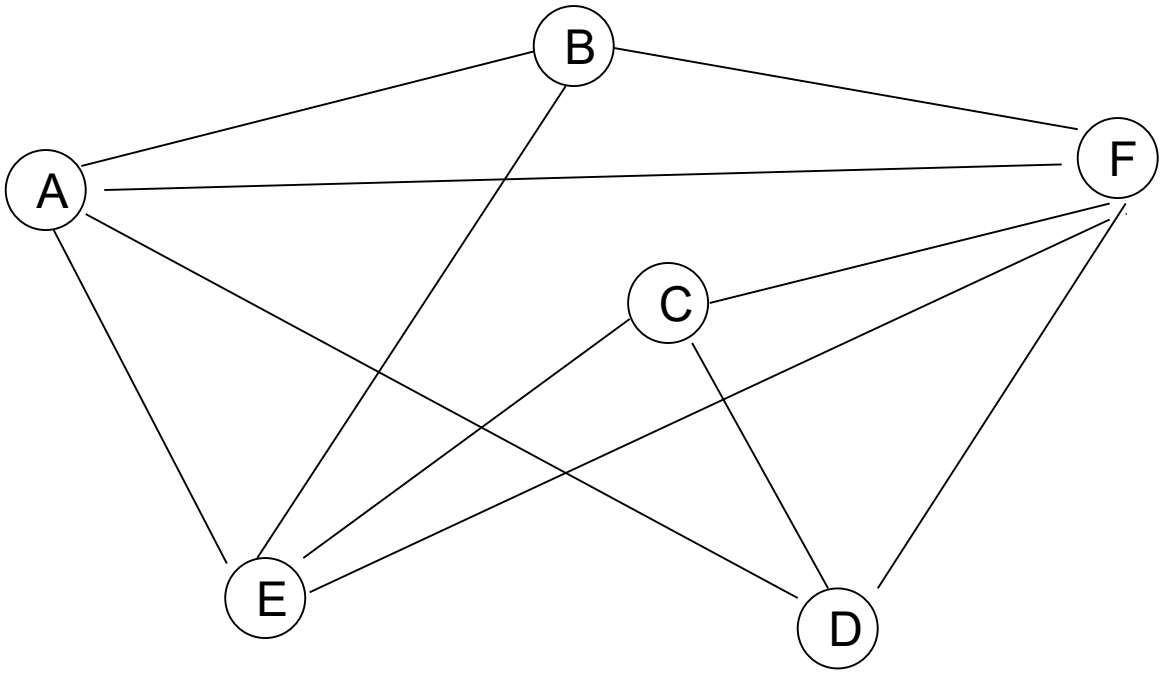
- 算法与数据结构的关系
  - 计算机科学家沃斯（N.Wirth）提出的“算法+数据结构=程序”揭示了程序设计的本质：对实际问题选择一种好的数据结构，加上设计一个好的算法，而好的算法很大程度上取决于描述实际问题的数据结构。
  - 算法与数据结构是互相依赖、互相联系的。
  - 任何一个算法的设计取决于选定的数据（逻辑）结构，而算法的实现依赖于采用的存储结构。

- 例1：查询某城市某人的电话号码
  - 建立一张登记表，存放2个数据项：姓名+Tel
  - 好的算法取决于这张表的结构及存储方式：
    - （1）按照登记次序存储在顺序表中
    - （2）按照姓名顺序有序地存储在顺序表中
    - （3）在（1）的基础上，再建立一张姓名索引表：
      姓名　顺序表中的地址

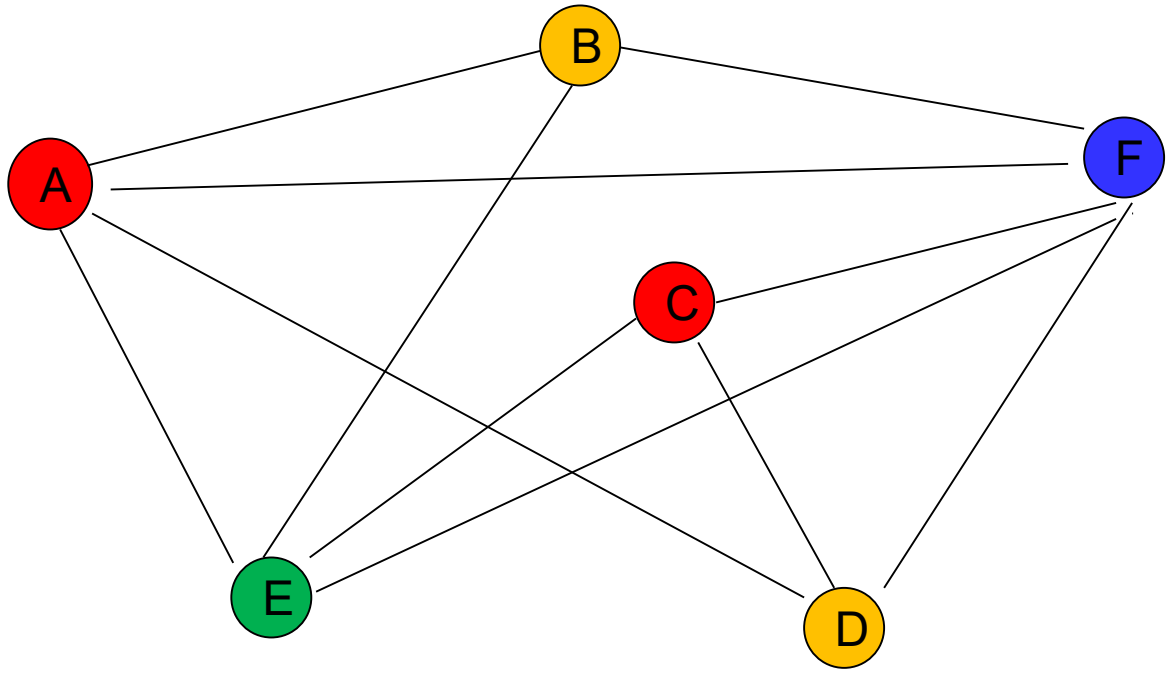
- 例2: 设计一个考试日程安排表，使在尽可能短的时间内安排完考试，要求同一个学生选修的课程不能安排在同一个时间内。
- 方案之一： 1： A,C
  2:B,D 3:E 4: F

姓名	选修1	选修2	选修3
Alex	A	B	E
Bill	C	D	
Jim	C	E	F
Joe	D	F	A
Lisa	B	F	

- 解决该问题，首先选择一个合适的数据结构。用无向图表示，图中的顶点表示课程，不能同时考试的课程之间连上一条边。



- 则该问题就抽象成对该无向图进行“着色”操作，即用尽可能少的颜色去给图中每个顶点着色，使得任意两个相邻的顶点着不同的颜色。同一种颜色表示一个考试时间。
- 答案：1： A,C 2:B,D 3:E 4: F



- 所以，解决问题的关键步骤是先选取合适的数据结构表示问题，才能写出有效的算法。

1.4 算法和算法分析

1.4.2. 算法设计的要求

- （1）正确性　四层含义

  （a）不含语法错误（b）对于几组输入数据能够得出预期的结果（c）对于精心选择的典型、苛刻而带有刁难性的几组输入数据能够得出预期的结果（d）对于一切合法的输入数据都能产生预期的结果

- （2）可读性

  首先是给人读，然后才是机器执行

- （3）健壮性　容错性

  当输入数据非法时，算法仍能做出反应或进行处理

- （4）效率与低存储量需求

1.4 算法和算法分析

1.4.3.算法效率的度量

- 评价标准：
  - 1）算法运行所花费的时间（时间特性）
  - 2）算法运行所占用的存储空间（空间特性）
  - 3）算法的可读性、健壮性、简单性等（非功能特性）

1.4 算法和算法分析

1.4.3.算法效率的度量

- 度量算法执行时间的两种方法

1）事后统计法（计时）

- 此方法有两个缺陷：(必须依据算法写成程序；时间的统计依赖于硬件、软件等环境)

2）事前分析估算法

- 此方法取决于多个因素：算法策略、问题规模、程序语言、编译质量、机器速度等

1.4 算法和算法分析

■1.4.4. 时间复杂度

(1) 语句的频度：语句重复执行的次数。

- {++x; s=0; } ++x 的频度为1
- for (i=1; i<=n; ++i) {++x; s+=x;}

  ++x的频度为n, 时间复杂度 O(n)
- for (j=1;j<=n;++j)

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

  ++x的频度为n^2
- for(i=2; i<=n; ++i)

  for(j=2; j<=i-1; ++j) {++x; a[i] [j]=x;}

++x语句频度为： 1+2+3+…+n-2= n^2-3n+2

1.4 算法和算法分析

(2)时间复杂度:

- 定义：设算法中所有语句的语句频度为T(n)，$f(n)$是当n趋于无穷大时与T(n)是同阶无穷大，算法的时间量度记作　$T(n) = O(f(n))$
- 其中：n是问题的规模

*f(n)*是运算时间随n增大时的增长率，表示随问题规模n的增大，T(n)执行时间的增长率和$f(n)$的**增长率相同**，*f(n)*称作算法的**渐进时间复杂度**，简称时间复杂度。

“*O*”的数学含义是，若存在两个常量 *m* 和 *n0*，当 *n>n0* 时，|*T(n)*| <= m*|*f(n)*|，

则记作 *T(n) = O(f(n))*

例1.1 两个 n*n 的矩阵相乘。其中矩阵的"阶" n 为问题的规模。

- 算法 1.1

```
void Mult_matrix( int c[][], int a[][], int b[][], int n){// a、b 和 c 均为 n 阶方阵，且 c 是 a 和 b 的乘积
    for (i=1; i<=n; ++i)  //n次
        for (j=1; j<=n; ++j) {  //n次
            c[i,j] = 0;    //n*n次
            for (k=1; k<=n; ++k)
                c[i,j] += a[i,k]*b[k,j];  //n*n*n次
        }
}// Mult_matrix
```

- 算法的时间复杂度为O (n^3) 。

例1.2 对 n 个整数的序列进行选择排序。其中序列的"长度" n 为问题的规模。

- 算法 1.2

```
    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 ) {  //n-1次
      j = i;   //n-1次
      for ( k = i+1; k < n; ++k ) //n(n-1)/2
        if (a[k] < a[j] ) j = k;  //n(n-1)/2
      if ( j != i ) { w = a[j]; a[j] = a[i]; a[i] = w;}//n-1
} // select_sort
```

- 算法的时间复杂度为O (n^2) 。

例1.3 对 n 个整数的序列进行起泡排序。

算法1.3

```
void  bubble_sort(int a[ ]，int n){
 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] ←→a[j+1];//基本操作:
              change=TURE;  }
         }}
```

- 算法的时间复杂度为O (n^2) 。

- 从这三个例子可见，算法时间复杂度取决于最深循环内包含基本操作的语句的频度。
- 有的情况下，算法中基本操作重复执行的次数还随问题的输入数据集不同而不同。例如：冒泡排序需要分情况讨论：最好，最坏，平均
  - 最好情况（初始序列从小至大排列）：0次
  - 最坏情况（初始序列从大至小排列）：$n(n-1)/2$
  - 平均时间复杂度为：$O(n^2)$

1.4 算法和算法分析

- 渐进上界记号（大O）：
  - $f(n), g(n)$是正整数n的函数，若存在正常数c和n_0使得对所有$n \geq n_0$有：

$$0 \leq f(n) \leq cg(n)$$

成立，则称$f(n)$ 的渐进上界是$g(n)$,记作：

$$f(n) = O(g(n))$$

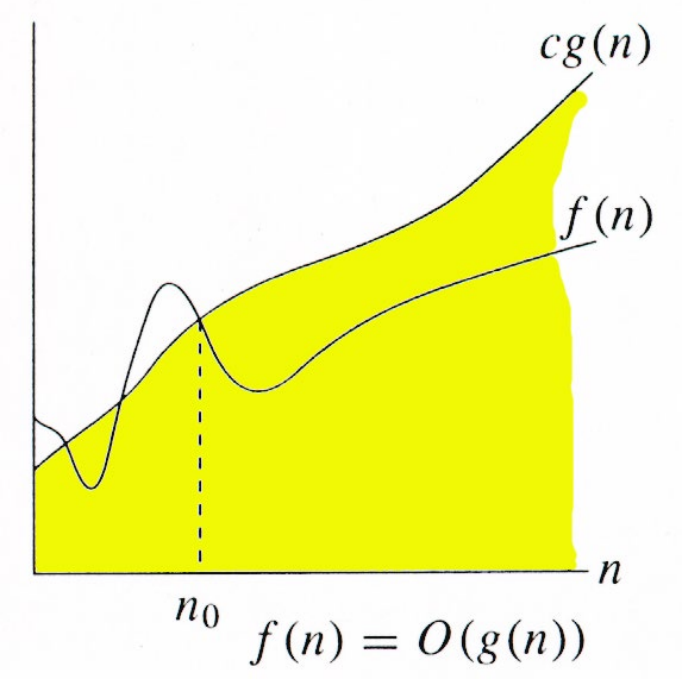
例如：f(n)=$4n^2$+3n+5

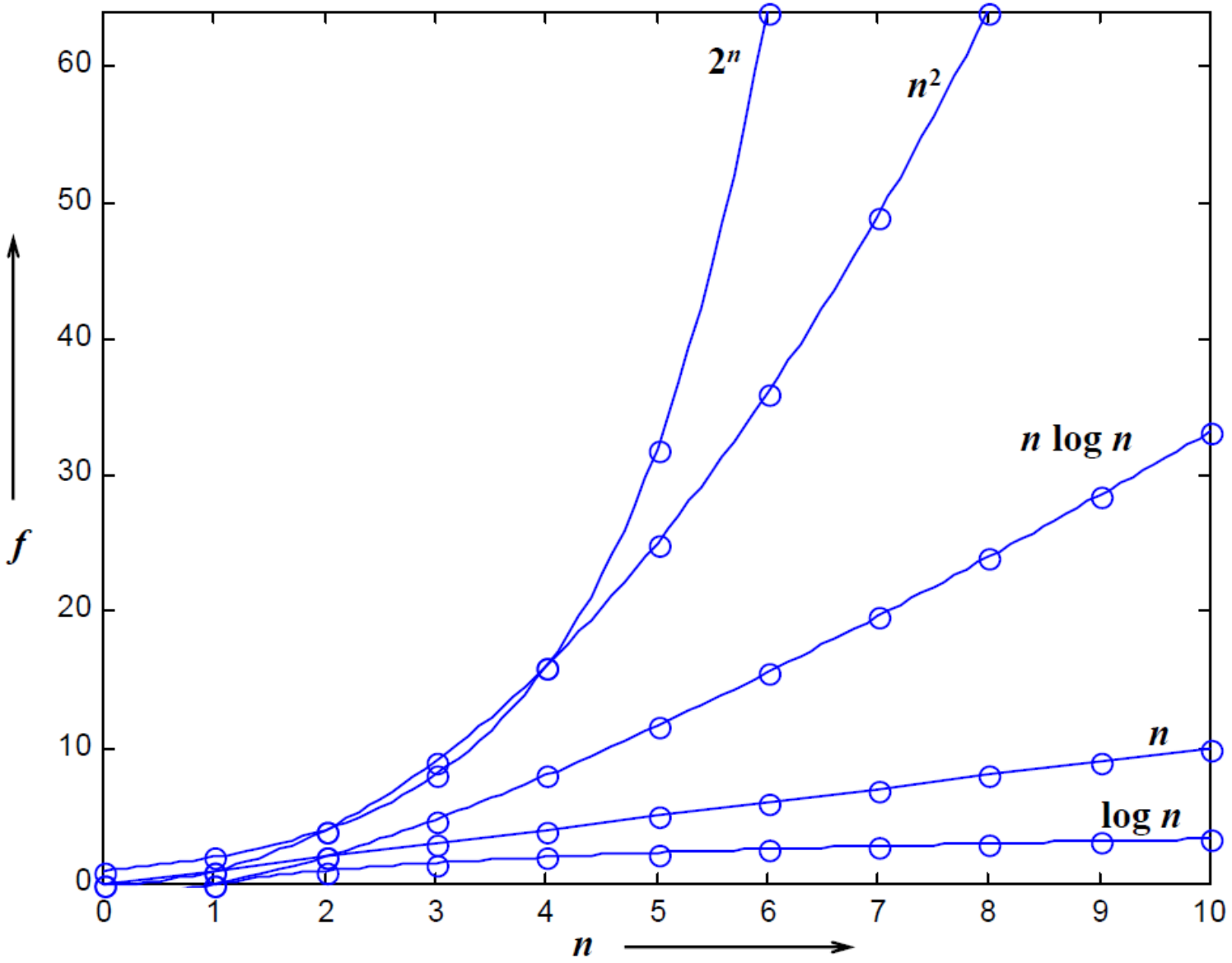
g(n)=?

g(n)=n ✗

√ g(n)=n^2

√ g(n)=n^3





$$O(1) < O(logn) < O(n) < O(nlogn) < O(n^2) < O(2^n)$$

常数 对数 线性 平方 **指数**

大O运算规则

规则1：$kf(n) = O(f(n))$ 大O忽略常数因子

规则2：$if\ \ f(n) = O(g(n)) and\ g(n) = O(h(n))\ then\ f(n) = O(h(n))$ 传递性

规则3（加法规则，顺序复合）：$f(n) + g(n) = O(\max\{f(n), g(n)\})$

规则4（乘法规则，循环）：$if\ \ f1(n) = O(g1(n))\ and\ f2(n) = O(g2(n)) then\ f1(n) * f2(n) = O(g1(n) * g2(n))$

定理：若$A(n)=a_m n^m+a_{m-1} n^{m-1}+\cdots+a_1 n+a_0$是一个m次多项式，则$A(n)=O(n^m)$　　证略。

表：时间复杂度和算法运行时间的关系（假定每秒1M指令）

T(n) / n	O(logn)	O(n)	O(nlogn)	$O(n^2)$	$O(n^3)$	$O(n^5)$	$O(2^n)$	O(n!))
20	4.3us	20us	86.4us	400us	8ms	3.2s	1.05s	771世纪
40	5.3us	40us	213us	1600us	64ms	1.7min	12.7天	2.59*1032世纪
60	5.9us	60us	354us	3600us	216ms	13min	366世纪	2.64*1066世纪

- 以下六种计算算法时间的多项式是最常用的。其关系为：

$$O(1) < O(logn) < O(n) < O(nlogn) < O(n^2) < O(n^3)$$

- 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。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。
- 不同数量级时间复杂度的形状图示（p16）

例如：求最大子段和 [2,-1,-3,1,5,-2,6,-5]

```
int MaxSubseqSum2( int A[], int N )
{ int ThisSum, MaxSum = 0;int i, j;
        for(i=0;i<N;i++) {
                ThisSum = 0; /* ThisSum是从A[i]到A[j]的子列和*/
                for(j=i;j<N;j++) {/*j是子列右端*/
                        ThisSum += A[j];
                        if( ThisSum > MaxSum )
                                MaxSum = ThisSum;
                } /* j循环结束*/
        } /* i循环结束*/
        return MaxSum;
```

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

1.4 算法和算法分析

1.4.5.空间复杂度

- （1）存储算法本身所占用的空间
- （2）算法的输入/输出数据占用的空间
- （3）算法在运行过程中临时占用的辅助空间

- 原地工作：若辅助空间相对于输入数据量是常数，则称此算法是原地工作。
- 若所占空间量依赖于特定的输入，按最坏情况来分析。

1.4 算法和算法分析

- 结论：
  - （1）当f(n)为对数函数、幂函数、或它们的乘积时，算法的运行时间是可以接受的，称这些算法是有效算法；当f(n)为指数函数或阶乘函数时，算法的运行时间是不可接受的，称这些算法是无效的算法。
  - （2）无法精确计算基本操作的执行次数（频度）时，只须求出其关于n的增长率。随着n值的增大，增长速度各不相同，n足够大时，存在下列关系：

    对数函数<幂函数<指数函数

1.4 算法和算法分析

- （3）算法的时间复杂度除了与问题规模有关外，还与被处理的数据的分布情况有关。不同的数据分布有不同的运行时间函数。因此算法分析中，通常用最好情况、最坏情况、平均情况的时间复杂度来描述。除特别指明外，均指最坏情况下的时间复杂度。

如冒泡程序，可考虑平均时间复杂度和最坏情况下的时间复杂度

本章小结

- 数据结构
  - 包含数据的逻辑结构和物理结构
  - 对每种结构定义相关的各种运算
  - 设计相应的算法
  - 分析算法的效率
- 学完这门课应达到的要求：
  - 掌握各类基本数据结构和相应的存储结构
  - 提高算法和编写算法的能力
  - 能针对给定问题，选择相应的数据结构，并能设计和分析算法