数据结构和算法

刘聪

2022/08/30

周二 3-4节
周四 3-4节

评分

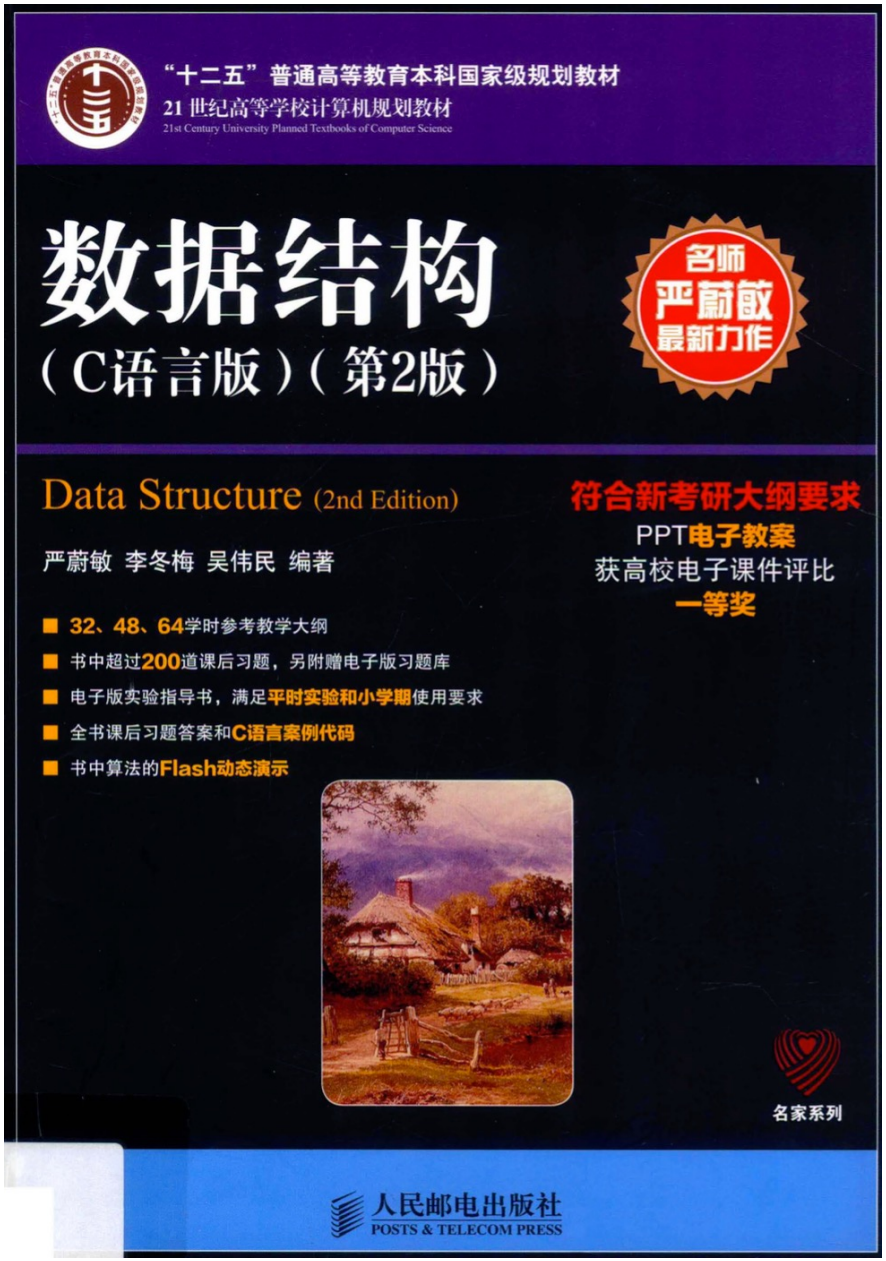
- 理论课
  - 平时: 40%
    - 作业 30%
    - 课堂练习(考勤) 10%
  - 期末统一试题闭卷考试: 60%
    - 不安排统一的期中考试

- 实验课
  - 平时: 70%
    - 课堂练习+考勤 10%
    - 三次以上的测试 60%
    - 全部在matrix系统上完成
  - 期末统一试题上机考试: 30%
    - 使用matrix系统

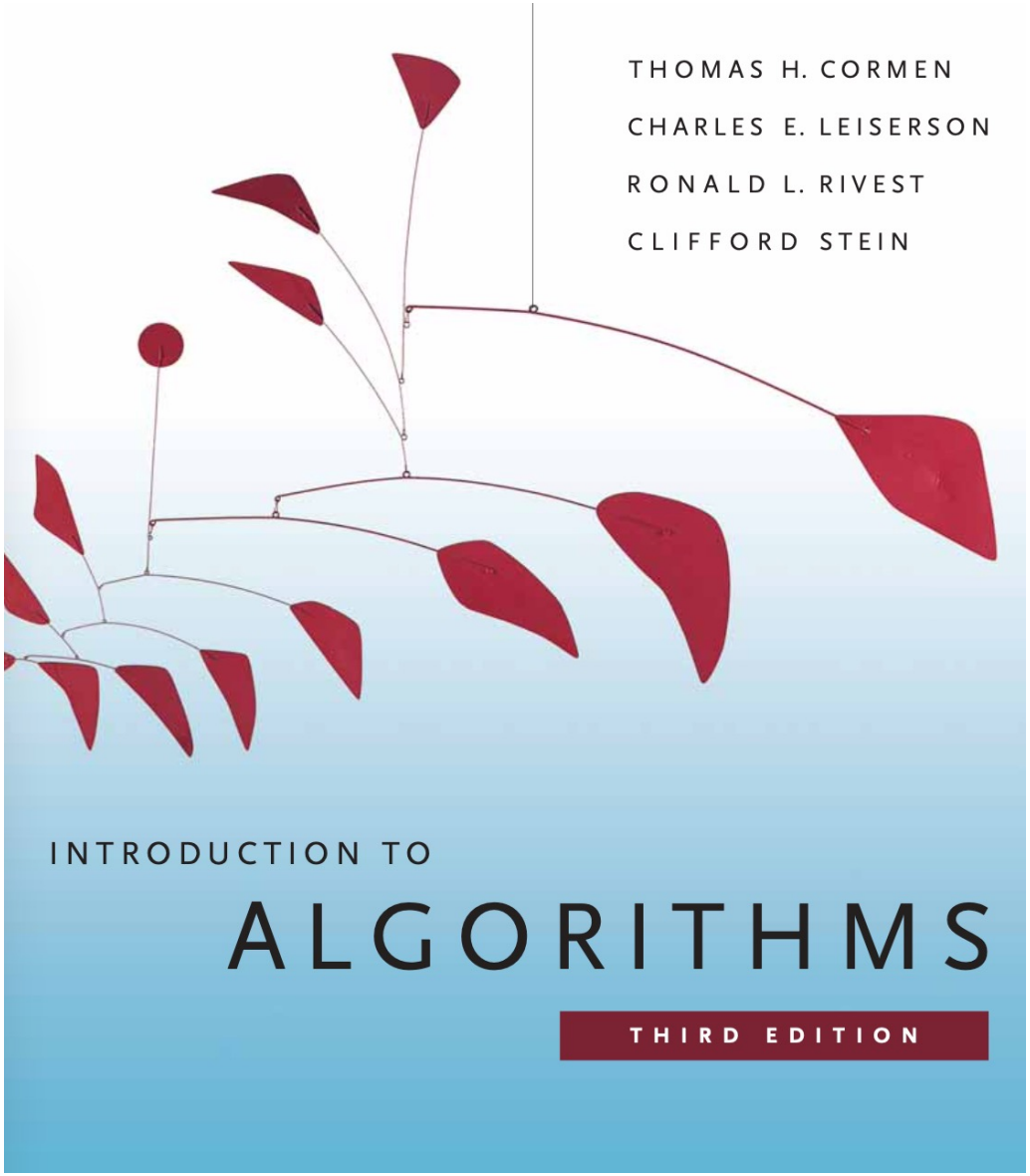
工具

- 课程网站
  - 课堂上链接
    - 给定Wi-Fi热点
    - 浏览器输入给定的ip地址
- 课件
  - 每次在课堂网站上下载
- 作业
  - 在matrix系统上完成
    - matrix.sysu.edu.cn
  - 理论课
    - 上课当前晚上10点前完成
  - 实验课
    - 下课前完成
- 课堂练习
  - 在课堂网站上完成较简单的题目
  - 成绩作为考勤分数

参考书


“十二五”普通高等教育本科国家级规划教材
21世纪高等学校计算机规划教材
21st Century University Planned Textbooks of Computer Science
数据结构
（C语言版）（第2版）
名师
严蔚敏
最新力作
Data Structure (2nd Edition)
严蔚敏 李冬梅 吴伟民 编著
符合新考研大纲要求
PPT电子教案
获高校电子课件评比
一等奖
32、48、64学时参考教学大纲
书中超过200道课后习题，另附赠电子版习题库
电子版实验指导书，满足平时实验和小学期使用要求
全书课后习题答案和C语言案例代码
书中算法的Flash动态演示
名家系列
人民邮电出版社
POSTS & TELECOM PRESS

参考书



- https://github.com/aliaamohamedali/Algorithms

本章内容

- 数据结构的研究内容
- 数据结构的例子
- 基本概念和术语
- 抽象数据类型的表示与实现
- 算法和算法分析

数据结构的研究内容

- 数据结构这门课所要研究的问题
  - 计算机是一门研究用计算机进行信息表示和处理的科学。这里面涉及到两个词题:信息的表示,信息的处理。
  - 信息的表示和组织又直接关系到处理信息的程序的效率。

数据结构的研究内容

- 《算法与数据结构》是计算机科学中的一门综合性专业基础课
- 是介于数学、计算机硬件计算机软件三者之间的一门核心课程
- 不仅是一般程序设计的基础，而且是设计和实现编译程序、操作系统、数据库系统及其他系统程序和大型应用程序的重要基础。

本章内容

- 数据结构的研究内容
- 数据结构的例子
- 基本概念和术语
- 抽象数据类型的表示与实现
- 算法和算法分析

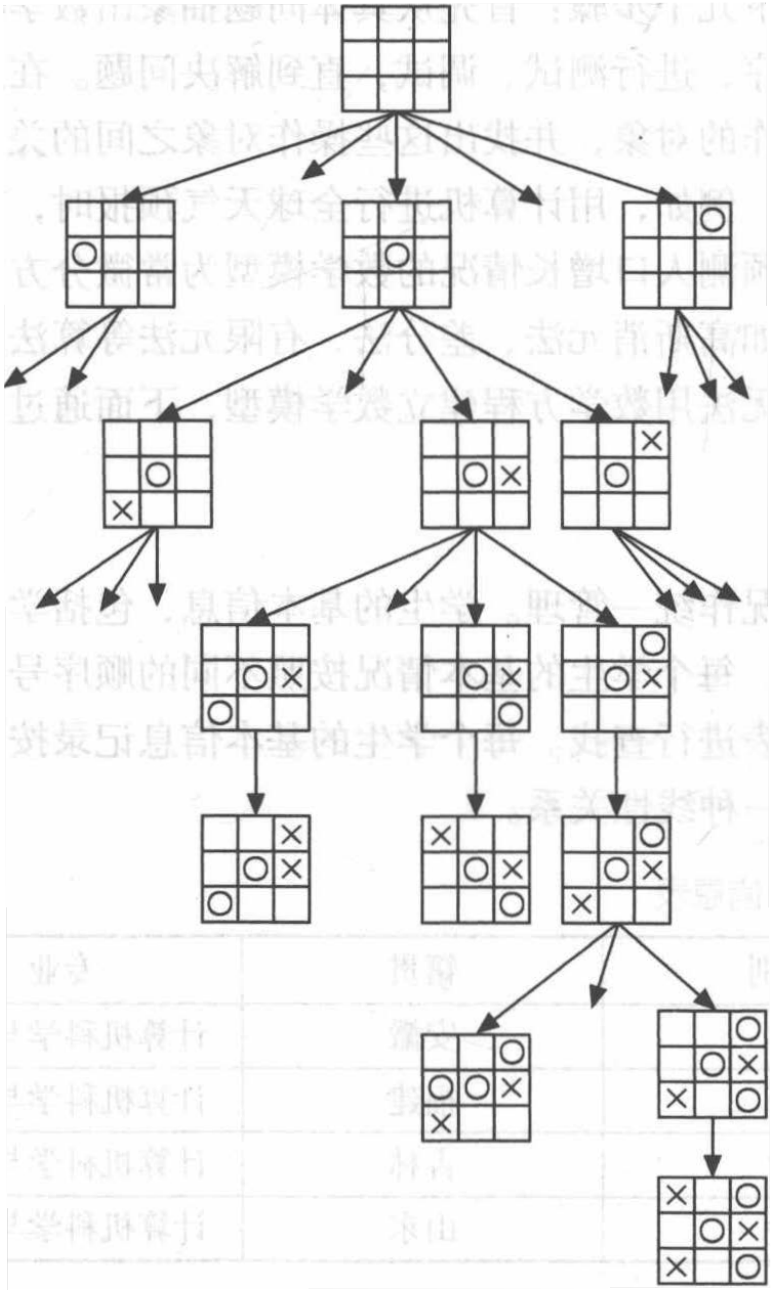
数据结构的例子

- 学生学籍管理系统
  - 高等院校教务处使用计算机对全校的学生情况作统一管理。学生的基本信息，包括学生的学 号、姓名、性别、籍贯、专业等。每个学生的基本情况按照不同的顺序号，依次 存放在 “学生基本信息表” 中，根据需要对这张表进行查找。每个学生的基本信息记录按顺序号 排列，形成了学生基本信息记录的线性序列，呈一种线性关系。

学号	姓名	性别	籍贯	专业
060214201	杨阳	男	安徽	计算机科学与技术
060214202	薛林	男	福建	计算机科学与技术
060214215	王诗萌	女	吉林	计算机科学与技术
060214216	冯子晗	女	山东	计算机科学与技术

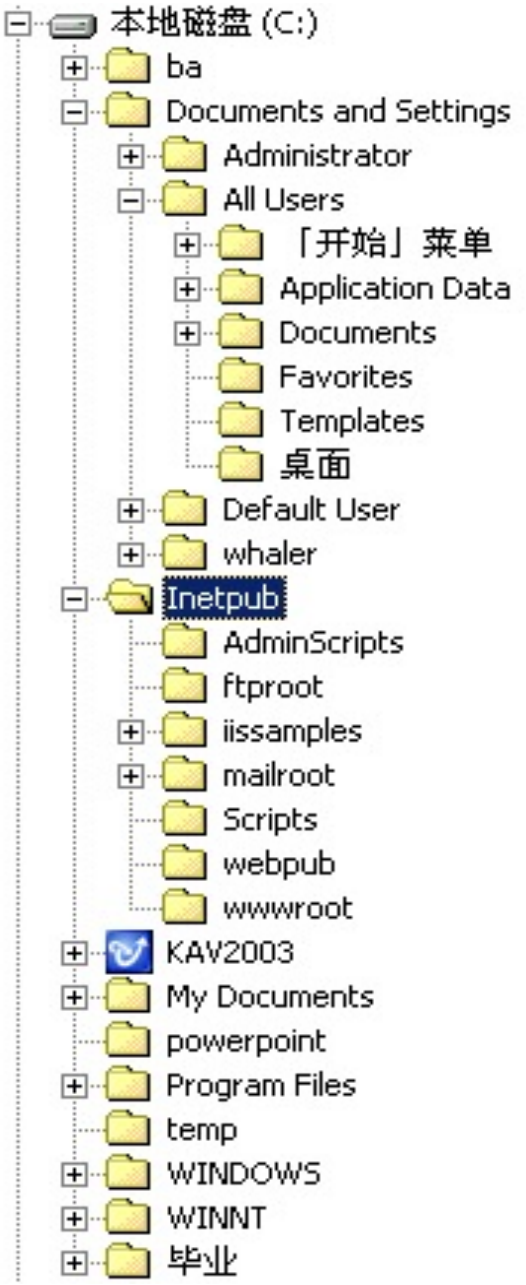
数据结构的例子

- 人机对弈问题
  - 计算机之所以能和人对弈是因为已经将对弈的策略在计算机中存储好。由于对弈的过程是在一定规则下随机进行的,所以,为使计算机能灵活对弈,就必须把对弈过程中所有可能发生的情况及相应的对策都加以考虑。以最简单的井字棋为例,初始状态是一个空的棋盘格局。对弈开始后,每下一步棋,则构成-个新的棋盘格局,且相对于上一个棋盘格局的可能选择可以有多种形式。因而整个对弈过程如同图 所示的“一棵倒长的树”。在这棵""树" 中,从初始状态(根)到某一最终格局(叶子)的一条路径,就是一次具体的对弈过程。



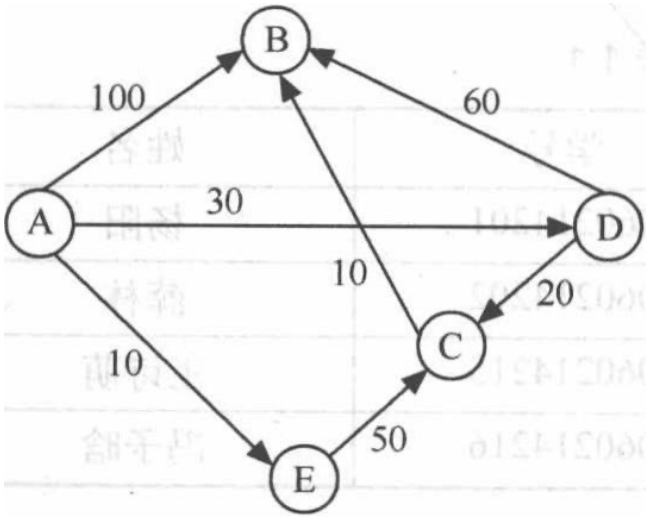
数据结构的例子

- 磁盘目录文件系统
  - 磁盘根目录下有很多子目录及文件，每个子目录里又可以包含多个子目录及文件，但每个子目录只有一个父目录，依此类推：
  - 本问题是一种典型的树型结构问题，如图，数据与数据成一对多的关系，是一种典型的非线性关系结构—树形结构。



数据结构的例子

- 最短路径问题
  - 从城市 A 到城市 B 有多条线路，但每条线路的交通费不同，那么，如何选择一条线路，使得 从城市A到城市B的交通费用最少呢?解决的方法是，可以把这类问题抽象为图的最短路径问 题。如图 1.2 所示， 图中的顶点代表城市， 有向边代表两个城市之间的通路，边上的权值代表两 个城市之间的交通费。求解 A 到 B 的最少交通费用，就是要在有向图中 A 点(源点)到达 B 点 (终点)的多条路径中， 寻找一条各边权值之和最小的路径，即最短路径。
  - 最短路径问题的数学模型就是图结构，算法是求解两点之间的最短路径。诸如此类的图结构 还有网络工程图和网络通信图等， 在这类问题中，元素之间是多对多的网状关系，施加于对象上的操作依然有查找、插入和删除等。这类数学模型称为 “图” 的数据结构。



本章内容

- 数据结构的研究内容
- 数据结构的例子
- 基本概念和术语
- 抽象数据类型的表示与实现
- 算法和算法分析

基本概念和术语

- 20世纪60年代初期，“数据结构“有关的内容散见于操作系统、编译原理等课程中。1968 年，“数据结构”作为一门独立的课程被列入美国一些大学计算机科学系的教学计划，同年，著名 计算机科学家D.E.Knuth 教授发表了《计算机程序设计艺术》第一卷《基本算法》。这是第一本较 系统地阐述“数据结构“基本内容的著作。之后，随着大型程序和大规模文件系统的出现，结构化程序设计成为程序设计方法学的主要研究方向，人们普遍认为程序设计的实质就是对所处理的问题选 择一种好的数据结构，并在此结构基础上施加一种好的算法，著名科学家Wirth教授的《算法+数据结构=程序》正是这种观点的集中体现。

基本概念和术语

- 数据 (Data) 是客观事物的符号表示，是所有能输入到计算机中并被计算机程序处理的符号的总称。如数学计算中用到的整数和实数，文本编辑中用到的字符串，多媒体程序处理的图形、图像、声音及动画等通过特殊编码定义后的数据。
- 数据元素 (Data Element) 行是数据的基本单位，在计算机中通常作为一个整体进行考虑和处理。在有些情况下，数据元素也称为元素、记录等。数据元素用于完整地描述一个对象，如前一节示例中的一名学生记录，树中棋盘的一个格局(状态)，以及图中的一个顶点等。
- 数据项 (Data Item) 格子 是组成数据元素的、有独立含义的、不可分割的最小单位。例如，学生基本信息表中的学号、姓名、性别等都是数据项。
- 数据对象 (Data Object) 是性质相同的数据元素的集合 表，是数据的一个子集。例如:整数数据对象是集合N={0,±1,±2,…}，字母字符数据对象是集合C={'A','B',…,'Z','a','b',…,'z'}，学生基本信息表也可以是一个数据对象。由此可以看出，不论数据元素集合是无限集(如整数集)，或是有限集(如字母字符集)，还是由多个数据项组成的复合数据元素(如学生表)的集合，只要集合内元素的性质均相同，都可称之为一个数据对象

基本概念和术语

- 数据结构 (Data Structure) 是相互之间存在一种或多种特定关系的数据元素的集合。换句话 说，数据结构是带 ” 结构" 的数据元素的集合，“结构” 就是指数据元素之间存在的关系。
- 数据的逻辑结构是从逻辑关系上描述数据，它与数据的存储无关，是独立于计算机的。因此，数据的逻辑结构可以看作是从具体问题抽象出来的数学模型。

基本概念和术语

- 数据的逻辑结构是从逻辑关系上描述数据，它与数据的存储无关，是独立于计算机的。因此，数据的逻辑结构可以看作是从具体问题抽象出来的数学模型。
- 数据的逻辑结构有两个要素：一是数据元素；二是关系。数据元素的含义如前所述，关系是指数据元素间的逻辑关系。根据数据元素之间关系的不同特性，通常有四类基本结构

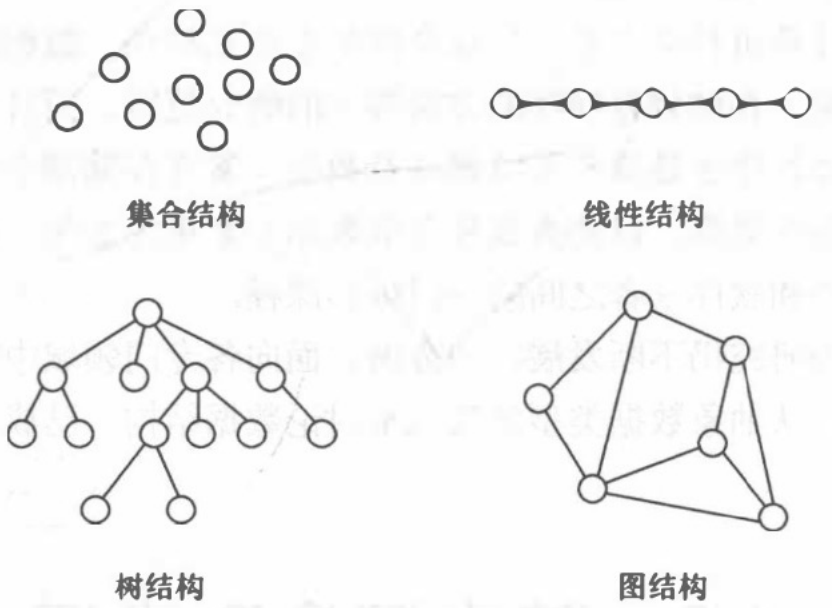
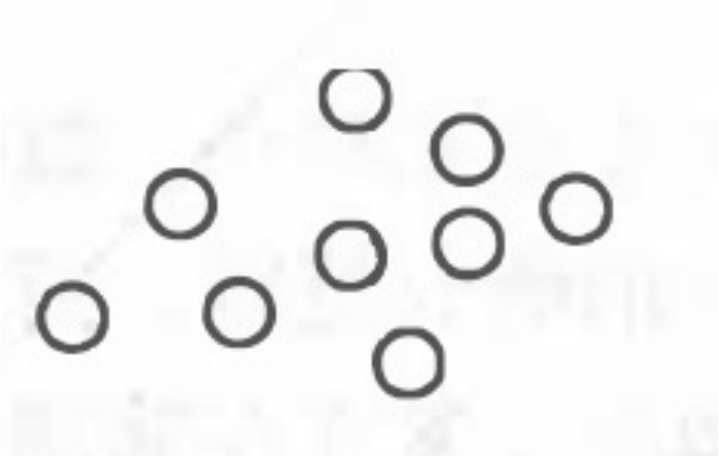


图 1.3　四类基本逻辑结构关系图

基本概念和术语

- 数据元素之间除了“属于同一集合”的关系外，别无其他关系。例如，确定一名学生是否为班级成员，只需将班级看做一个集合结构

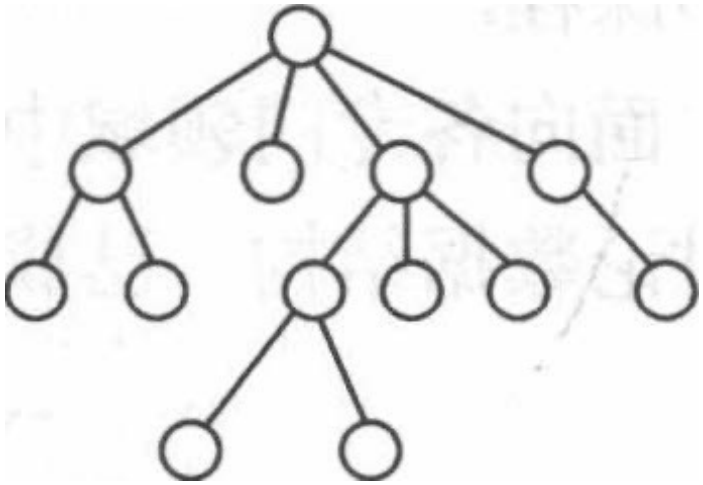


基本概念和术语

- 数据元素之间存在一对一的关系。例如， 将学生信息数据按照其入学报到的时间先后顺序进行排列， 将组成一个线性结构

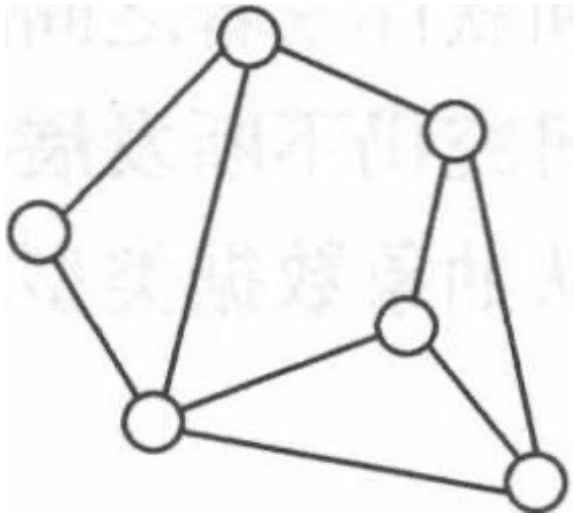
基本概念和术语

- 数据元素之间存在一对多的关系。例如，在班级的管理体系中，班长管理多个组长，每位组长管理多名组员，从而构成树形结构



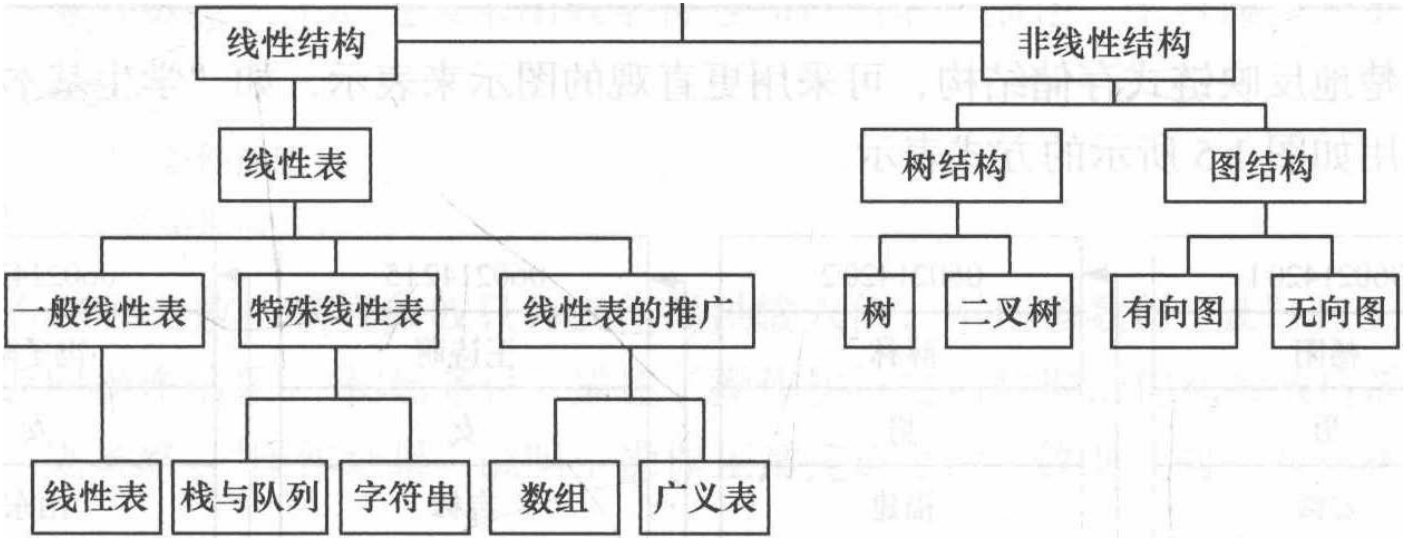
基本概念和术语

- 数据元素之间存在多对多的关系。例如，多位同学之间的朋友关系，任何两位同学都可以是 朋友，从而构成 或网状结构。



基本概念和术语

- 线性结构 /非线性结构
  - 线性结构包括线性表(典型的线性结构，如例1.1中的学生基本信息表)、栈和队列(具有特 限制的线性表，数据操作只能在表的一端或两端进行)、字符串(也是特殊的线性表，其特殊性 表现在它的数据元素仅由一个字符组成)、数组(是线性表的推广，它的数据元素是一个线性表)、广义表(也是线性表的推广，它的数据元素是一个线性表，但不同构，即或者是单元素，或者是 线性表)。
  - 非线性结构包括树(具有多个分支的层次结构)和二叉树(具有两个分支的层次结构)、有向图(一种图结构，边是顶点的有序对)和无向图(另一种图结构，边是顶点的无序对)。这几 种逻辑结构可以用一个层次图描述



基本概念和术语

- 数据对象在计算机中的存储表示称为数据的存储结构，也称为物理结构。把数据对象存储到 计算机时， 通常要求既要存储各数据元素的数据，又要存储数据元素之间的逻辑关系， 数据元素在计算机内用一个结点来表示。数据元素在计算机中有两种基本的存储结构，分别是顺序存储结构和链式存储结构

基本概念和术语

- 顺序存储结构是借助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，通常借助 程序设计语言的数组 类型来描述

表 1.2　　顺序存储结构

地址	学号	姓名	性别	籍贯	专业
0	060214201	杨阳	男	安徽	计算机科学与技术
50	060214202	薛林	男	福建	计算机科学与技术
100	060214215	王诗萌	女	吉林	计算机科学与技术
150	060214216	冯子晗	女	山东	计算机科学与技术

基本概念和术语

- 顺序存储结构要求所有的元素依次存放在一片连续的存储空间中，而链式存储结构，无需占用一整块存储空间。但为了表示结点之间的关系，需要给每个结点附加指针字段，用于存放后继元素的存储地址。所以链式存储结构通常借助于程序设计语言的指针类型来描述。

链式存储结构

表 1.3　　链式存储结构

地址	学号	姓名	性别	籍贯	专业	后继结点的首地址
0	060214201	杨阳	男	安徽	计算机科学与技术	100
50	060214216	冯子晗	女	山东	计算机科学与技术	^
100	060214202	薛林	男	福建	计算机科学与技术	150
150	060214215	王诗萌	女	吉林	计算机科学与技术	50

图 1.5　链式存储结构示意图

基本概念和术语

- 在程序设计语言中，每一个数据都属于某种数据类型。类型明显或隐含地规定了数 据的取值范围、存储方式以及允许进行的运算，数据类型是一个值的 集合和 定义在这个值集上的 一组操作的总称。
- 抽象就是抽取出实际问题的本质。在计算机中使用二进制数来表示数据，在汇编语言中则可 给出各种数据的十进制表示，它们是二进制数据的抽象，使用者在编程时可以直接使用，不必考 虑实现细节。在高级语言中，则给出更高一级的数据抽象，出现了数据类型，如整型、实型、字 符型等，可以进一 步利用这些类型构造出线性表、栈、队列、树、图等复杂的抽象数据类型。
- 抽象数据类型 (Abstract Data Type, ADT) 一般指由用户定义的、表示应用问题的数学模型，以及定义在 这个模型上的一组操作的总称，具体包括三部分:数据对象、数据对象上关系的集合 以及对数据对象的基本操作的集合

本章内容

- 数据结构的研究内容
- 数据结构的例子
- 基本概念和术语
- 抽象数据类型的表示与实现
- 算法和算法分析

抽象数据类型的表示与实现

- 运用抽象数据类型描述数据结构，有助于在设计一 个软件系统时，不必首先考虑其中包含的 数据对象，以及操作在不同处理器中的表示和实现细节，而是在构成软件系统的每个相对独立的 模块上定义一组数据和相应的操作，把这些数据的表示和操作细节留在模块内部解决，在更高的 层次上进行软件的分析和设计
- 抽象数据类型的概念与面向对象方法的思想是一 致的。抽象数据类型独立于具体实现，将数 据和操作封装在一起，使得用户程序只能通过抽象数据类型定义的某些操作来访问其中的数据，从而实现了信息隐藏。抽象数据类型和类的概念实际上反映了程序或软件设计的两层抽象：抽象数据类型相当于在 概念层(或称为抽象层) 上描述问题，而类相当于在实现层上描述问题。此外，C++中的类只是一 个由用户定义的普通类型，可用它来定义变量(称为对象或类的实例)。因此，在 C++中，最 终是通过操作对象来解决实际问题的，所以我们可将该层次看做是应用层。例如，main程序就可 看做是用户的应用程序。

抽象数据类型的表示与实现

```
/* 定义部分

ADT Complex {
数据对象:D={e1,e2 | e1,e2 in R,R 是实数集}
数据关系: S={<e1,e2> | e1 是复数的实部, e2 是复数的虚部} 基本操作:
    Complex_init(C, x, y)
        操作结果:构造复数C, 其实部和虚部分别被赋以参数x和y的值。
    Complex_real(C)
        初始条件:复数C已存在。
        操作结果:返回复数C的实部值。
    Complex_imaginary(C)
        初始条件:复数C已存在。
        操作结果:返回复数c的虚部值。
    Complex_add(Cl,C2)
        初始条件:Cl, C2是复数。
        操作结果:返回两个复数 Cl 和 C2 的和。
    Complex_add(Cl,C2)
        初始条件:Cl, C2是复数。
        操作结果:返回两个复数 Cl 和 C2 的差。
} ADT Complex
*/
```

抽象数据类型的表示与实现

```
23    // 表示部分
24    typedef struct
25    {
26        float real;
27        float imaginary;
28    } Complex;
```

抽象数据类型的表示与实现

```
30  // 实现部分
31  void Complex_init(Complex * this, float real, float imaginary) {
32      this->real = real;
33      this->imaginary = imaginary
34  }
35
36  float Complex_real(Complex * this) {
37      return this->real;
38  }
39
40  float Complex_imaginary(Complex * this) {
41      return this->imaginary;
42  }
43
44  Complex Complex_add(Complex * this, Complex * other) {
45      Complex sum;
46      sum.read = this->real + other->real;
47      sum.imaginary = this->imaginary + other->imaginary;
48      return sum;
49  }
50
51  Complex Complex_sub(Complex * this, Complex * other) {
52      Complex diff;
53      diff.read = this->real - other->real;
54      diff.imaginary = this->imaginary - other->imaginary;
55      return diff;
56  }
```

算法和算法分析

- 数据结构与算法之间存在着本质联系
- 算法是研究数据结构的重要途径

算法和算法分析

- 算法 (Algorithm) 是为了解决某类问题而规定的一个有限长的操作序列。一个算法必须满足以下五个重要特性。
- (1)有穷性。一个算法必须总是在执行有穷步后结束，且每一步都必须在有穷时间内完成。
- (2) 确定性。对于每种情况下所应执行的操作，在算法中都有确切的规定，不会产生二义性，使算法的执行者或阅读者都能明确其含义及如何执行。
- (3) 可行性。算法中的所有操作都可以通过已经实现的基本操作运算执行有限次来实现。
- (4) 输入。一个算法有零个或多个输入。当用函数描述算法时，输入往往是通过形参表示的，在它们被调用时，从主调函数获得输入值。
- (5) 输出。一个算法有一个或多个输出，它们是算法进行信息加工后得到的结果，无输出的 算法没有任何意义。当用函数描述算法时，输出多用返回值或引用类型的形参表示

算法和算法分析

- 评价算法优劣的基本标准
- 一个算法的优劣应该从以下几方面来评价。
- (1)正确性。在合理的数据输入下，能够在有限的运行时间内得到正确的结果。

- (2) 可读性。一个好的算法，首先应便于人们理解和相互交流，其次才是机器可执行性。可 读性强的算法有助于人们对算法的理解，而难懂的算法易于隐藏错误，且难于调试和修改。
- (3) 健壮性。当输入的数据非法时，好的算法能适当地做出正确反应或进行相应处理，而不 会产生一些莫名其妙的输出结果。
- (4) 高效性。高效性包括时间和空间两个方面。时间高效是指算法设计合理，执行效率高，可以用时间复杂度来度量;空间高效是指算法占用存储容量合理，可以用空间复杂度来度量。时 间复杂度和空间复杂度是衡量算法的两个主要指标。

算法和算法分析

- 算法的时间复杂度
  - 算法效率分析的目的是看算法实际是否可行，并在同一问题存在多个算法时，可进行时间和 空间性能上的比较，以便从中挑选出较优算法。
  - 衡量算法效率的方法主要有两类:事后统计一法和事前分析估算法。事后统计法需要先将算法 实现，然后测算其时间和空间开销。这种方法的缺陷很显然，一是必须把算法转换成可执行的程 序，二是时空开销的测算结果依赖于计算机的软硬件等环境因素，这容易掩盖算法本身的优劣。 所以我们通常采用事前分析估算法，通过计算算法的渐近复杂度来衡量算法的效率

算法和算法分析

- 不考虑计算机的软硬件等环境因素，影响算法时间代价的最主要因素是问题规模。问题规模 是算法求解问题输入量的多少，是问题大小的本质表示，一般用整数n表示。
- 问题规模n对不 同的问题含义 不同，例如，在排序运算中n为参加排序的记录数，在矩阵运算中n为矩阵的阶 数，在多项式运算中n为多项式的项数，在集合运算中n为集合中元素的个数，在树的有关运 算中n为树的结点个数，在图的有关运算中n为图的顶点数或边数。显然，n越大算法的执行时 间 越长。

算法和算法分析

- 一个算法的执行时间大致上等于其所有语句执行时间的总和，而语句的执行时间则为该条语 句的重复执行次数 和执行一次所需时间的乘积。
- 一条语句的重复执行次数称作语句频度(Frequency Count)。
- 所谓的算法分析并非精确统计算法实际执行所需时间，而是针对算法中语句的执行次数做出估计，从中得到算法执行时间的信息。
- 设每条语句执行一次所需的时间均是单位时间，则一个算法的执行时间可用该算法中所有语句频度之和来度量。

算法和算法分析

- 例1.4 求两个n阶矩阵的乘积算法

```
for (i=1;i<=n;i++)                                  //频度为 n
    for(j=1;j<=n;j++) {                             //  频度为 n^2
        c[i][j]=0;                                  //频度为 n^2
        for(k=1;k<=n;k++)                           //频度为 n^3
            c[i][j]=c[i][j]+a[i][k]*b[k][j];        //频度为 n^3
    }
```

- 该算法中所有语句频度之和， 是矩阵 阶数n的函数，用f(n)表示之。换句话说， 上例算法的 执行时间与f(n)成正比。
- $f(n) = 2n^3 + 3n^2 + 2n + 1$

算法和算法分析

- 算法的时间复杂度定义
  - 对于例1.4这种较简单的算法，可以直接计算出算法中所有语句的频度，但对于稍微复杂一些的算法，则通常是比较困难的，即便能够给出，也可能是个非常复杂的函数。
  - 因此，为了客观地反映一个算法的执行时间，可以只用算法中的“基本语句"的执行次数来度量算法的工作量。所谓“基本语句”指的是算法中重复执行次数和算法的执行时间成正比的语句，它对算法运行时间的贡献最大。
  - 通常，算法的执行时间是随问题规模增长而增长的，因此对算法的评价通常只需考虑其随问题规模增长的趋势。这种情况下，我们只需要考虑当问题规模充分大时，算法中基本语句的执行次数在渐近意义下的阶。

本章内容

- 数据结构的研究内容
- 数据结构的例子
- 基本概念和术语
- 抽象数据类型的表示与实现
- 算法和算法分析

算法和算法分析

- 算法的时间复杂度定义
  - 如例1.4矩阵的乘积算法. 当n趋向无穷大时, 显然有

$$\lim_{n\to\infty} f(n)/n^3 = \lim_{n\to\infty} (2n^3 + 3n^2 + 2n + 1)/n^3 = 2$$

  - 即当n充分大时
    - f(n)和n^3 之比是一个不等于零的常数。
    - 即f(n)和n^3是同阶的,
    - 或者说f(n)和n^3的数量级(Order of Magnitude)相同。

算法和算法分析

- 算法的时间复杂度定义
  - 一般情况下， 算法中基本语句重复执行的次数是问题规模n的某个函数f(n), 算法的时间量 度记作
    - $T(n)= 0(f(n)) = O(n^3)$
      大-O 表示法
  - 它表示随问题规模n的增大， 算法执行时间的增长率和 f(n的) 增长率相同， 称做算法的渐近时间复杂度，简称时间复杂度(Time Complexity)

算法和算法分析

- 数学符号"O" 的严格定义
  - 若T(n)和j(n是) 定义在正整数集合上的两个函数，则T(n)=0(f(n))表示存在正的常数C和n_0,使得当n≥ n_0时都满足O ≤T(n) ≤Cf(n)。
  - 该定义说明了函数 T(n)和f(n)具有相同的增长趋势,并且T(n)的增长至多趋向于函数f(n)的增长。符号"O"用来描述增长率的上限，它表示当问题规模n≥ n_0时，算法的执行时间不会超过f(n), 其直观的含义如图1.6所示

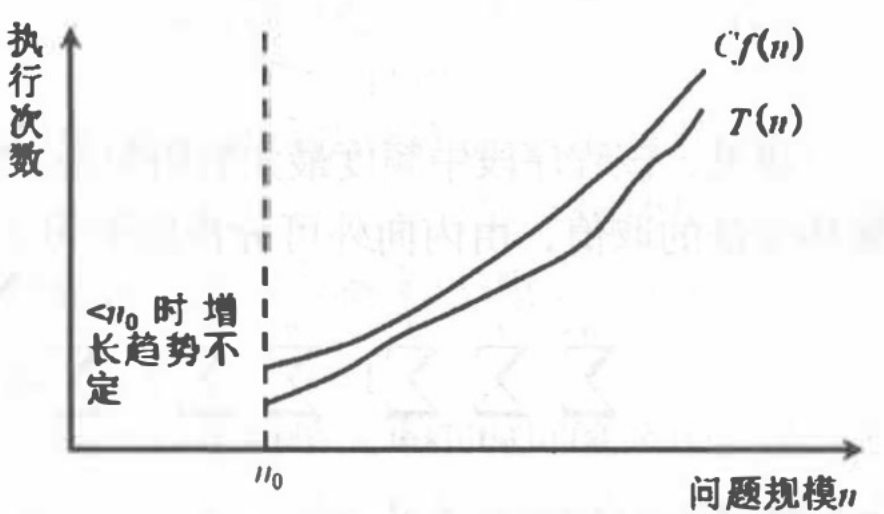


图 1.6　符号 “O” 的直观含义

算法和算法分析

- 算法的时间复杂度分析举例
  - 析算法时间复杂度的基本方法为:找出所有语句中语句频度最大的那条语句作为基本语句，计算基本语句的频度得到问题规模n的某个函数f(n), 取其数量级用符号"O"表示即可。
  - {x++;s=0;}

算法和算法分析

- 算法的时间复杂度分析举例
  - 析算法时间复杂度的基本方法为:找出所有语句中语句频度最大的那条语句作为基本语句，计算基本语句的频度得到问题规模n的某个函数f(n), 取其数量级用符号"O"表示即可。
  - {x++;s=0;}
  - T(n)=0(1), 称为常量阶

算法和算法分析

- for(i=O;i<lOOOO;i++) {x++;s=O;}

算法和算法分析

- for(i=0;i<10000;i++) {x++;s=0;}
  - 实际上，如果算法的执行时间不随问题规模n的增加而增长，算法中语句频度就是某个常数。 即使这个常数再大， 算法的时间复杂度都是 0(1)

算法和算法分析

- for(i=0;i<n;i++) {x++;s=0;}

算法和算法分析

- for(i=0;i<n;i++) {x++;s=0;}
  - 循环体内两条基本语句的频度均为f(n) =n, 所以算法的时间复杂度为T(n) = O(n), 称为线性阶
  -

算法和算法分析

- ```
  (1)  x=0;y=0;
  (2)  for(k=1;k<=n;k++)
  (3)     x++;
  (4)  for(i=1;i<=n;i++)
  (5)     for(j=1;j<=n;j++)
  (6)        y++;
  ```
  ```

算法和算法分析

```
(1)  x=0;y=0;
(2)  for(k=1;k<=n;k++)
(3)     x++;
(4)  for(i=1;i<=n;i++)
(5)     for(j=1;j<=n;j++)
(6)        y++;
```

- 对循环语句只需考虑循环体中语句的执行次数，以上程序段中频度最大的语句是(6)，其频 度为$f(n)=n^2$，所以该算法的时间复杂度为$T(n)=O(n^2)$，称为平方阶。

算法和算法分析

- ```
  (1)  x=1;
  (2)  for(i=1;i<=n;i++)
  (3)      for(j=1;j<=i;j++)
  (4)          for(k=1;k<=j;k++)
  (5)              x++;
  ```
  ```

算法和算法分析

```
(1)  x=1;
(2)  for(i=1;i<=n;i++)
(3)      for(j=1;j<=i;j++)
(4)          for(k=1;k<=j;k++)
(5)              x++;
```

- 该算法的时间复杂度为 $T(n) = O(n^3)$, 称为立方阶

算法和算法分析

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

算法和算法分析

```
for(i=1;i<=n;i=i*2){x++;s=0;}
```

- $T(n)= O(\log_2 n)$, 称为对数阶

算法和算法分析

- 常见的时间复杂度按数量级递增排列依次为
  - 常量阶 0(1) 、
  - 对数阶 O(log_2n) 、
  - 线性阶 O(n) 、
  - 线性对数 阶O(log_2n)、
  - 平方阶O(n^2)、
  - 立方阶O(n^3)、……、
  - K 次方阶 O (n^k) 、
  - 指数阶 0(2^n)等
- 应该 尽可能选择使用多项式阶阶 O (n^k)的算法，而避免使用指数阶的算法。

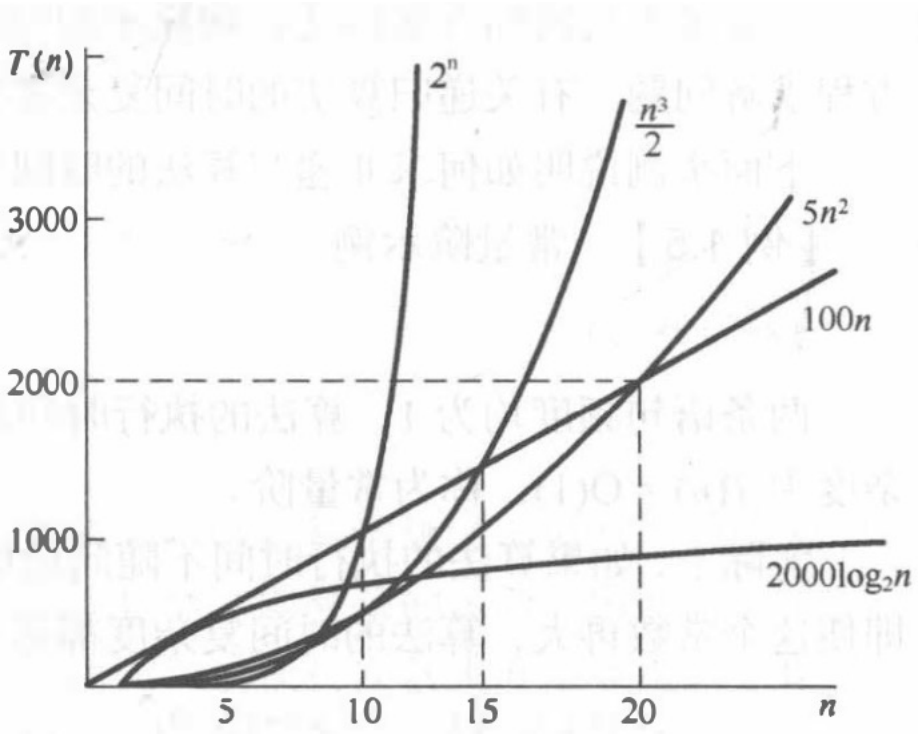


图 1.7　常见函数的增长率

算法和算法分析

- 最好、最坏和平均时间复杂度

```
(1)  for(i=0;i<n;i++)
(2)      if(a[i]==e) return i+1;
(3)  return 0;
```

  - 此例说明，算法的时间复杂度不仅与问题的规模有关，还与问题的其他因素有关。再如某些 排序的算法，其执行时间与待排序记录的初始状态有关。因此，有时会对算法有最好、 最坏以及 平均时间复杂度的评价。
  - 对算法时间复杂度的度量，人们更关心的是最坏情况下和平均情况下的时间复杂度。然而在 很多情况下，算法的平均时间复杂度难于确定。因此，通常只讨论算法在最坏情况下的时间复杂 度，即分析在最坏情况下，算法执行时间的上界。 在本书后面内容中讨论的时间复杂度，除特别 指明外，均指最坏情况下的时间复杂度。

算法和算法分析

- 渐近空间复杂度(Space Complexity)
  - 若算 法执行时所需要的辅助空间相对于输入数据 量而言是个常数， 则称这个算法为原地工作，辅助 空间为0(1),

【算法 1】

```
for(i=0;i<n/2;i++)
{   t=a[i];
    a[i]=a[n-i-1];
    a[n-i-1]=t;
}
```

【算法 2】

```
for(i=0;i<n;i++)
    b[i]=a[n-i-1];
for(i=0;i<n;i++)
    a[i]=b[i];
```

算法和算法分析

- 渐近空间复杂度(Space Complexity)
  - 对于一个算法，其时间复杂度和空间复杂度往往是相互影响的，当追求一个较好的时间复杂度时，可能会导致占用较多的存储空间，即可能会使空间复杂度的性能变差，反之亦然。不过，通常情况下，鉴于运算空间较为充足，人们都以算法的时间复杂度作为算法优劣的衡 址指标。
  -

小结

- 介绍了
  - 数据结构的基本概念和术语
    - 定义：数据、数据元素、数据项、数据对象、 数据结构
    - 数据结构所含两个层次：数据的逻辑结构和存储结构
      - 同一逻辑结构采用不同的 存储方法， 可以得到不同的存储结构
      - 逻辑结构是从具体问题抽象出来的数学模型，从逻辑关系上描述数据，它与数据的存储无 关
      - 通常有四类基本逻辑结构:集合结构、线性结构、 树形 结构和图状结构
      - 存储结构是 逻辑结构在计算机中的存储表示，有两类存储结构:顺序存储结构和链式存储 结构
    - 抽象数据类型是指由用户定义的、 表示应用问题的数学模型
      - 包括三部分:数据对象、数据对象上关系的集合， 以及对数据对象的基本 操作的集合
    - 算法是为了解决某类问题而规定的一个有限长的操作序列
      - 算法具有五个特性:有穷性、 确定性、可行性、输入和输出
      - 一个算法的优劣应该从以下四方面来评价:正确性、可读性、健 壮性和高效性
  - 算法和算法时间复杂度的分析方法