浙江省高等教育重点建设教材

```
#include   <stdio.h>
#include <malloc.h>
typedef   struct   node
{
     int   data;
     struct   node   *link;
} NODE;

NODE* re_order(NODE *head)
{
     NODE   *h, *p,*q,*r;
     int   t;
     if   (head==NULL||head->link==NULL)   return(head);
     h=NULL;      t=head->data;      p=head;
     while   ( p->link   )
          if   (p->link->data<t)
          {
               q=p->link;
               p->link=q->link;
               if (h==NULL)   h=q;
               else   r->link = q;
               r=q;
          }else
               p = p->link;
     if   (h==NULL)
          return(head);
     else
     {
          r->link = head ;
          return(h);
     }
```

DATA STRUCTURE

数据结构

◎ 吴海燕 任午令 章志勇 编著

ZHEJIANG UNIVERSITY PRESS
浙江大学出版社

浙江省高等教育重点建设教材

数 据 结 构

吴海燕　任午令　章志勇 编著

ZHEJIANG UNIVERSITY PRESS
浙江大学出版社

图书在版编目（CIP）数据

数据结构 / 吴海燕等编著. —杭州：浙江大学出版社，2011.6
ISBN 978-7-308-08738-4

Ⅰ. ①数… Ⅱ. ①吴… Ⅲ. ①数据结构 Ⅳ. ①TP311.12

中国版本图书馆 CIP 数据核字（2011）第 107976 号

数据结构

吴海燕　任午令　章志勇 编著

责任编辑　杜希武
封面设计　刘依群
出版发行　浙江大学出版社
（杭州市天目山路 148 号　邮政编码 310007）
（网址：http://www.zjupress.com）
排　　版　杭州好友排版工作室
印　　刷　杭州半山印刷有限公司
开　　本　787mm×1092mm　1/16
印　　张　16.5
字　　数　401 千
版 印 次　2011 年 6 月第 1 版　2011 年 6 月第 1 次印刷
书　　号　ISBN 978-7-308-08738-4
定　　价　33.00 元

前　言

“数据结构”作为一门独立的课程在国外是从1968年才开始设立的。1968年美国唐·欧·克努特教授开创了数据结构的最初体系，他所著的《计算机程序设计技巧》第一卷《基本算法》是第一本较系统地阐述数据的逻辑结构和存储结构及其操作的著作。“数据结构”在计算机科学中是一门综合性的专业基础课程。数据结构是介于数学、计算机硬件和计算机软件三者之间的一门核心课程。数据结构这一门课的内容不仅是一般程序设计（特别是非数值性程序设计）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的重要基础。

本书从数据结构的逻辑结构、存储结构和数据的运算等几个方面去介绍了线性表、堆栈、队列、串、数组、树、图和文件等常用的数据结构，以及程序设计中经常出现的排序和查找算法。全书共分九个章节，第一章综述了数据、数据结构和抽象数据类型等基本概念；第二章至第六章介绍上述的几种数据结构及其应用；第七章至第八章讨论排序和查找的各种算法；第9章介绍常用的文件结构。全书采用C语言作为数据结构和算法的描述语言。

本书可作为计算机及其相关专业的本科或专科教材，也可以作为信息类或其他相关专业的选修教材，还可以作为其他一些课程如编译原理、操作系统、计算机图形学、数据库系统等的辅助读物。教师可根据课时、专业和学生的实际情况，解读或选讲书中的内容。本教材也是浙江省精品课程“数据结构”的教学用书。全书概念清楚，选材精练，叙述深入浅出，用了大量经典的应用实例和图表来说明基本概念和方法，直观易懂。

本教材的作者均为浙江工商大学承担数据结构课程的骨干教师，项目实践经验丰富，积累了不少的教学素材，本书由吴海燕负责全书的策划和组织，任午令教授和章志勇老师对全书进行了统稿和校对。由浙江工商大学的吴海燕，谢满德，柳虹和章志勇编写，其中吴海燕编写了第1,3,4,9章，章志勇和吴海燕共同编写了第2章和第5章，谢满德和柳虹共同编写了第6章，柳红编写了第7,8章。同时，要感谢浙江工商大学的金剑秋，庄毅、江照意、彭浩宇、杨文武、张亦舜等教师对本书提出的真诚和宝贵的修改建议。

在本书的编写过程中，参考了部分图书资料和网站资料，在此向其作者表示感谢。由于作者水平有限，书中难免出现遗漏和不足之处，恳请社会各界同仁及读者朋友提出宝贵意见和真诚的批评。

作　者

2011年5月于浙江工商大学

目　录

第一章 绪 论

数据结构是计算机程序设计的重要理论技术基础，它不仅是计算机及相关专业考研和水平等级考试的必考科目，也是众多理工专业的热门选修课。通过数据结构的学习，培养数据抽象能力，在实际应用中有效合理地组织、存储和处理数据，正确地设计算法以及分析和评价算法能力。

1.1 为什么要学习数据结构

数据结构是一门综合性的专业课程，是介于数学、计算机硬件、计算机软件之间的一门核心课程，是设计和实现编译系统、操作系统、数据库系统及其他系统程序和大型应用程序的基础。

当给出一个问题描述后，用计算机帮助解题的过程通常是这样的，首先将具体问题转换为数学模型，然后给出算法，根据根据算法再进行程序设计，求解问题。然而在现实社会中存在着许多非数值计算问题，其数学模型难以用数学方程描述。我们来看几个例子：

(1)人机对弈问题(GamePlaying)

在对弈问题中，计算机处理的对象是一个个格局。计算机能与人对弈，是因为人们将博弈规则存入计算机，利用高速运算实现对弈过程。所有可能出现的格局是一棵倒置的树。这涉及数据结构中树的问题。

(2)旅游线路的安排问题

如需要到全国 9 个城市旅游，每个城市只走一次，由于地理环境不同等因素使各条旅游线路所需耗费不同，如何使耗费成本最低？这是一个讨论图的生成树的问题。

(3)求 n 个整数中的最大值问题

最大值求解问题中，如果这些整数的值有可能达到 10^{12}，那么对 32 位的计算机来说，就存在一个如何表示这些数的问题。

(4)员工信息检索系统

当我们需要查找某个员工的有关情况时，或者想查询某个部门或职务的员工的有关情况时，我们可以在信息检索系统中建立一张按工号顺序排列的员工信息表和分别按姓名、部门、职务等属性顺序排列的索引表。由这些表构成的文件就是员工信息检索的数学模型。

从以上例子中我们可以看到，描述这类非数值计算问题的数学模型不再是数学方程，而是诸如表、树和图之类的数据结构。因此，简单地说，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。

1.2 基本概念和术语

接下来，我们介绍数据结构中常用的一些术语。

数据(Data)：是指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。数据是计算机加工的对象，是数据元素的集合。

数据元素(DataElement)：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。有些情况下，数据元素也称为元素、结点、顶点、记录。有时数据元素可以由若干数据项(也称为字段、域、属性)组成，数据项是数据的有独立含义的最小标识单位。

数据类型：(DataType)：是一个值的集合和定义在这个值集上的所有的操作。例如，整数类型。数据类型可分为：原子数据类型和结构数据类型。原子类型的值是不可分解的，结构类型的值是由若干成分按某种结构组成的。

抽象数据类型(Abstract Data Type)：可理解为数据类型的进一步抽象。即把数据类型和数据类型上的运算捆在一起，进行封装。引入抽象数据类型的目的是把数据类型的表示和数据类型上运算的实现与这些数据类型和运算在程序中的引用隔开，使它们相互独立。对于抽象数据类型的描述，除了必须描述它的数据结构外，还必须描述定义在它上面的运算(过程或函数)。抽象数据类型上定义的过程和函数以该抽象数据类型的数据所应具有的数据结构为基础。一个 ADT 可描述为：

```
ADT ADT-Name{
    数据对象: 数据说明
    数据关系: 数据元素之间逻辑关系的描述
    基本操作/操作说明:
      Operation1://操作1, 它通常可用C或C++的函数原型来描述
      Operation2://操作2
    ......
  }//ADT
```

数据结构(Data Structure)：数据之间的相互关系，即数据的组织形式。包括以下三方面：

(1)数据元素之间的逻辑关系，也称为数据的逻辑结构；

(2)数据元素及其关系在计算机存储器内的表示，称为数据的存储结构；

(3)数据的运算，即对数据施加的操作。

数据的逻辑结构：在不产生混淆的前提下，常将数据的逻辑结构简称为数据结构。数据的逻辑结构有两大类：线性结构和非线性结构。线性结构的逻辑特征是：若结构是非空集，则有且仅有一个开始结点和一个终端结点，并且所有结点都最多只有一个直接前趋和一个直接后继。线性表是一个典型的线性结构。栈、队列、串等都是线性结构。非线性结构的逻辑特征是：一个结点可能有多个直接前趋和直接后继。数组、广义表、树和图等数据结构都是非线性结构。

数据的存储结构：数据结构在计算机中的表示(又称映象)称为数据的物理结构，又称存

储结构，是数据元素及其关系在计算机存储器的表示。用于表示数据元素的位串称之为元素或结点，用于表示数据项的位串称之为数据域。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。数据有四种存储结构：

(1)顺序存储结构：把逻辑上相邻的结点存储在物理位置上相邻的存储单元里，结点间的逻辑关系由存储单元的邻接关系来体现。通常顺序存储结构是借助于语言的数组来描述的。

(2)链式存储结构：不要求逻辑上相邻的结点物理上也相邻，结点间的逻辑关系是由附加的指针字段表示的，通常要借助于语言的指针类型来描述。

(3)索引存储方法：该方法通常在储存结点信息的同时，还建立附加的索引表。索引表由若干索引项组成。若每个结点在索引表中都有一个索引项，则该索引表称之为稠密索引。若一组结点在索引表中只对应一个索引项，则该索引表称为稀疏索引。索引项的一般形式是：

(关键字、地址)

关键字是能唯一标识一个结点的那些数据项。稠密索引中索引项的地址指示结点所在的存储位置；稀疏索引中索引项的地址指示一组结点的起始存储位置。

(4)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。

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。选择何种存储结构来表示相应的逻辑结构，视具体要求而定，主要考虑运算方便及算法的时空要求。

1.3 算法描述

算法(Algorithm)是对特定问题求解步骤的一种描述，它是指令的有限序列。每条指令表示一个或多个操作。

算法具有五个重要特性：有穷性、确定性、可行性、输入、输出。有穷性指算法执行有穷步后结束，不能无止境地执行下去；确定性是指算法的描述必须是清晰的，不具有二义性；可行性指算法原则上能精确地进行，用纸和笔有限次完成；一个算法必须有输入，算法结束后必须有输出。

对一个算法设计的要求包含以下几点：正确性、可读性、健壮性和效率与低存储量。

若一个算法对于每个输入实例均能终止并给出正确的结果，则称该算法是正确的。正确的算法解决了给定的计算问题，一个不正确的算法是指对某些输入实例不终止，或者虽然终止但给出的结果不是所渴望得到的答案，一般只考虑正确的算法。算法的健壮性是指算法在碰到一个非法数据时的处理能力。

算法的描述形式有自然语言表示法，伪代码表示法，流程图表示法，结构化流程图(N—S图)，计算机程序语言或其他语言表示法等等，唯一的要求是该说明必须精确地描述计算过程。

一般而言，描述算法最合适的语言是介于自然语言和程序语言之间的伪语言。它的控制结构往往类似于 Pascal、C 等程序语言，但其中可使用任何表达能力强的方法使算法表达更加清晰和简洁，而不至于陷入具体的程序语言的某些细节。从易于上机验证算法和提高实际程序设计能力考虑，在本书中大多数算法我们采用 C 语言的形式描述。

1.4 算法分析

求解同一问题可能有许多不同的算法，究竟如何来评价这些算法的好坏以便从中选出较好的算法呢？选用的算法首先应该是“正确”的。此外，主要考虑如下三点：

(1)执行算法所耗费的时间；

(2)执行算法所耗费的存储空间，其中主要考虑辅助存储空间；

(3)算法应易于理解，易于编码，易于调试等等。

一个占存储空间小、运行时间短、其他性能也好的算法是很难做到的。原因是上述要求有时相互抵触：要节约算法的执行时间往往要以牺牲更多的空间为代价；而为了节省空间可能要耗费更多的计算时间。因此我们只能根据具体情况有所侧重：若该程序使用次数较少，则力求算法简明易懂；对于反复多次使用的程序，应尽可能选用快速的算法；若待解决的问题数据量极大，机器的存储空间较小，则相应算法主要考虑如何节省空间。

接下来我们重点讨论算法的时间性能分析。

一个算法所耗费的时间＝算法中每条语句的执行时间之和；

每条语句的执行时间＝语句的执行次数(即频度(Frequency Count))×语句执行一次所需时间；

算法转换为程序后，每条语句执行一次所需的时间取决于机器的指令性能、速度以及编译所产生的代码质量等难以确定的因素。若要独立于机器的软、硬件系统来分析算法的时间耗费，则设每条语句执行一次所需的时间均是单位时间，一个算法的时间耗费就是该算法中所有语句的频度之和。

例如，求两个 n 阶方阵的乘积 C＝A×B，其算法如下：

```
# define n 100                                  // n 可根据需要定义,这里假定为100
void MatrixMultiply(int A[n][n], int B [n][n], int C[n][n])
{                                               //右边注释列为各语句的频度
    int i ,j ,k;
(1) for(i=0; i<n;j++)                           // n+1
(2)   for (j=0;j<n;j++) {                       // n(n+1)
(3)     C[i][j]=0;                              //  n²
(4)     for (k=0; k<n; k++)                     //  n²(n+1)
(5)       C[i][j]=C[i][j]+A[i][k]*B[k][j];      //  n³
      }
 }
```

该算法中所有语句的频度之和(即算法的时间耗费)为：

$$T(n)=2n^3+3n^2+2n+1$$

其中，语句(1)的循环控制变量 i 要增加到 n，测试到 i＝n 成立才会终止。故它的频度是 n＋1。但是它的循环体却只能执行 n 次。语句(2)作为语句(1)循环体内的语句应该执行 n 次，但语句(2)本身要执行 n＋1 次，所以语句(2)的频度是 n(n＋1)。同理可得语句

(3),(4)和(5)的频度分别是 n^2,$n^2(n+1)$和 n^3。算法 MatrixMultiply 的时间耗费 T(n)是矩阵阶数 n 的函数。

算法求解问题的输入量称为问题的规模(Size),一般用一个整数 n 表示。例如,矩阵乘积问题的规模是矩阵的阶数。一个图论问题的规模则是图中的顶点数或边数。一个算法的时间复杂度(Time Complexity,也称时间复杂性)T(n)是该算法所求解问题规模 n 的函数。当问题的规模 n 趋向无穷大时,时间复杂度 T(n)的数量级(阶)称为算法的渐进时间复杂度。

例如上面的算法 MatrixMultidy 的时间复杂度 $T(n)=2n^3+3n^2+2n+1$,当 n 趋向无穷大时,显然有

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

这表明,当 n 充分大时,T(n)和 n^3 之比是一个不等于零的常数。即 T(n)和 n^3 是同阶的,或者说 T(n)和 n^3 的数量级相同。记作 $T(n)=O(n^3)$是算法 MatrixMultiply 的渐近时间复杂度。数学符号"O"的严格的数学定义如下:

若 T(n)和 f(n)是定义在正整数集合上的两个函数,则 $T(n)=O(f(n))$表示存在正的常数 C 和 n0,使得当 $n\geqslant n0$ 时都满足 $0\leqslant T(n)\leqslant C\cdot f(n)$。

主要用算法时间复杂度的数量级(即算法的渐近时间复杂度)评价一个算法的时间性能。

例如,有两个算法 A1 和 A2 求解同一问题,时间复杂度分别是 $T1(n)=100n^2$,$T2(n)=5n^3$。

(1)当输入量 $n<20$ 时,有 $T1(n)>T2(n)$,后者花费的时间较少。

(2)随着问题规模 n 的增大,两个算法的时间开销之比 $5n^3/100n^2=n/20$ 亦随着增大。即当问题规模较大时,算法 A1 比算法 A2 要有效得多。它们的渐近时间复杂度 $O(n^2)$和 $O(n^3)$从宏观上评价了这两个算法在时间方面的质量。在算法分析时,往往对算法的时间复杂度和渐近时间复杂度不予区分,而经常是将渐近时间复杂度 $T(n)=O(f(n))$简称为时间复杂度,其中的 f(n)一般是算法中频度最大的语句频度。例如上文的算法 MatrixMultiply 的时间复杂度一般为 $T(n)=O(n^3)$,$f(n)=n^3$ 是该算法中语句(5)的频度。下面再举例说明如何求算法的时间复杂度。

(1) 交换 i 和 j 的内容。

```
temp=i;          // (1次)
i=j;             // (1次)
j=temp;          // (1次)
```

以上三条单个语句的频度均为 1,该程序段的执行时间是一个与问题规模 n 无关的常数。算法的时间复杂度为常数阶,记作 $T(n)=O(1)$。如果算法的执行时间不随着问题规模 n 的增加而增长,即使算法中有上千条语句,其执行时间也不过是一个较大的常数。此类算法的时间复杂度是 O(1)。

（2）求和。

```
sum=0;                    //（1次）
for(i=1;i<=n;i++)         //（n次 ）
   for(j=1;j<=n;j++)      //（n²次 ）
       sum++;             //（n²次 ）
```

因此，$T(n)=2n^2+n+1=O(n^2)$。一般情况下，对步进循环语句只需考虑循环体中语句的执行次数，忽略该语句中步长加 1、终值判别、控制转移等成分。因此，以上程序段中频度最大的语句是 sum＋＋，其频度为 $f(n)=n^2$，所以该程序段的时间复杂度为 $T(n)=O(n^2)$。

当有若干个循环语句时，算法的时间复杂度是由嵌套层数最多的循环语句中最内层语句的频度 f(n)决定的。

（3）
```
for(i=0;i<n;i++)
      for(j=0;j<i;j++)
        for(k=0;k<j;k++)
          x=x+2;
```

当 i 的值为某个 x，j 的值为某个 y 的时候，内层循环的次数为 y。而当 i 的值为某个 x 时，j 可以取 0，1，…，x－1，所以这里最内循环共进行了 0＋1＋…＋x－1＝(x－1)x/2 次，因此，当 i 从 0 取到 n 时，则循环共进行了：0＋(1－1)＊1/2＋…＋(n－1)n/2＝n(n＋1)(n－1)/6 所以时间复杂度为 $O(n^3)$。

常见的时间复杂度按数量级递增排列依次为：常数 O(1)、对数阶 O(logn)、线形阶 O(n)、线形对数阶 O(nlogn)、平方阶 $O(n^2)$ 立方阶 $O(n^3)$、…、k 次方阶 $O(n^k)$、指数阶 $O(2^n)$、阶乘阶 O(n!)，显然，时间复杂度为阶乘阶 O(n!)的算法效率极低，当 n 值稍大时就无法应用。

最坏情况下的时间复杂度称最坏时间复杂度。如无特别说明，讨论的时间复杂度均是最坏情况下的时间复杂度。这样做的原因是：最坏情况下的时间复杂度是算法在任何输入实例上运行时间的上界，这就保证了算法的运行时间不会比任何更长。平均时间复杂度是指所有可能的输入实例均以等概率出现的情况下，算法的期望运行时间。

类似于时间复杂度的讨论，一个算法的空间复杂度(Space Complexity)S(n)定义为该算法所耗费的存储空间，它也是问题规模 n 的函数。渐近空间复杂度也常常简称为空间复杂度。算法的时间复杂度和空间复杂度合称为算法的复杂度。

习　题

1. 设 n 为正整数，利用大“O”记号，将下列程序段的执行时间表示为 n 的函数。

（1）
```
i=1; k=0;
while(i<n)
  { k=k+10*i;i++;
  }
```

```
(2) i=0; k=0;
    do{
       k=k+10*i; i++;
      }
    while(i<n);
(3) i=1; j=0;
    while(i+j<=n)
      {
        if (i>j) j++;
        else i++;
      }
(4)x=n; // n>1
   while (x>=(y+1)*(y+1))
     y++;
(5) x=91; y=100;
    while(y>0)
        if(x>100)
          {x=x-10;y--;}
        else x++;
```

2. 按增长率由小至大的顺序排列下列各函数：

2^{100}，$(3/2)^n$，$(2/3)^n$，n^n，$n^{0.5}$，$n!$，2^n，$\lg n$，$n^{\lg n}$，$n^{(3/2)}$

3. 简述下列概念：数据、数据元素、数据类型、数据结构、逻辑结构、存储结构、线性结构、非线性结构。

4. 试举一个数据结构的例子、叙述其逻辑结构、存储结构、运算三个方面的内容。

5. 评价一个好的算法，可以从哪几方面来考虑？

6. 根据数据元素之间的逻辑关系，一般有哪几类基本的数据结构？

7. 有实现同一功能的两个算法 A1 和 A2，其中 A1 的时间复杂度为 $T1=O(2^n)$，A2 的时间复杂度为 $T2=O(n^2)$，仅就时间复杂度而言，请具体分析这两个算法哪一个好。

第二章　线性表

线性结构是简单且常用的数据结构，而线性表是一种典型的线性结构。一般情况下，如果需要在程序中存储数据，最简单、最有效的方法是把它们存放在一个线性表中。只有当需要组织和搜索大量数据时，才考虑使用更复杂的数据结构。本章讨论和分析了一般线性表的表示，并介绍了不同的实现线性表的方法。

2.1　线性表的概念

在所有的数据结构中，最简单的是线性表(Linear List)。通常，定义线性表为 n(n≥0)个数据元素的一个有限的序列。记为 $L=(a_1,a_2,\cdots,a_i,a_{i+1},\cdots,a_n)$。其中，L 是线形表的名称，$a_i$ 是表中的数据元素，是不可再分割的原子数据，有时又称为结点或表项。n 是线形表中数据元素的个数，也称为表的长度。若 n=0 叫做空表，此时，表中一个元素也没有。线性表的第一个元素称为表头(head)，最后一个元素称为表尾(tail)。

线性表是一个有限线性序列，这意味着表中的各个数据元素是相继排列的，且除了第一个数据元素和最后一个数据元素外，每两个相邻的数据元素之间都构成了直接前驱和直接后继的相互关系，也就是说，线性表存在唯一的第一个数据元素和最后一个数据元素。除第一个数据元素外，其他所有的数据元素有且仅有一个直接前驱，第一个元素没有前驱；除最后一个数据元素外，其他所有的数据元素有且仅有一个直接后继，最后一个元素没有后继。

直接前驱和直接后继从不同的角度深刻地刻画了线形表结点之间的一种逻辑关系(即邻接关系)。在线性结构中，这种相互邻接关系是 1 对 1 的，即除了首尾的数据元素外，每个结点只有一个直接前驱并且只有一个直接后继。而所有结点按 1 对 1 的邻接关系构成的整体就是线性结构。

线性表中的每一个元素都有自己的数据类型。虽然在概念上允许线性表中各个元素可以有不同的数据类型(参看有关广义表的讨论)，但为简单起见，本章讨论的线性表的程序实现中，表中所有的数据元素都具有相同的数据类型。下面是几个线性表的例子。

COLOR=('Red','Orange','Yellow','Green','Blue','Black')

DEPT=(通信，计算机，自动化，微电子，建筑与城市规划，生命科学，精密仪器)

SCORE=(667,664,662,659,59,659,657,654,653,652,650,650)

线性表中元素的值与它的位置之间可以有联系，也可以没有联系，这个取决于实际的各种应用。例如，有序线性表(sorted list)中的各个数据元素按照值的递增顺序排列，而无序线性表(unsorted list)的各个数据元素的值与位置之间就没有特殊的联系。线性表的抽象数据类型定义如下所示：