“十二五”普通高等教育本科国家级规划教材

Data Structure

数据结构

（第2版）

主编　陈　越
编著　何钦铭　徐镜春　魏宝刚　杨　枨

高等教育出版社

“十二 科国家级规划教材

Data Structure

数据结构

（第2版）

主编 陈 越
编著 何钦铭 徐镜春 魏宝刚 杨 枨

高等教育出版社·北京

内容提要

本书的主要任务是介绍并探讨有关数据组织、算法设计、时间和空间效率的概念和通用分析方法，帮助读者学会数据的组织方法和现实世界问题在计算机内部的表示方法，针对问题的应用背景分析，选择合适的数据结构，从而培养高级程序设计技能。

本书第1章介绍了数据结构与算法的基本概念；第2章是对C语言关键内容的复习，为后续章节理解数据结构的实现做准备；第3章至第7章分别介绍了线性表、树、散列表、图、排序算法等经典数据结构与算法；最后在第8章通过对两个实际生活中提炼出的问题的解答，帮助读者更深刻地体会数据结构的应用。

本书可作为高等学校计算机类专业“数据结构”课程的教材。

图书在版编目(CIP)数据

数据结构/陈越主编；何钦铭等编著. --2版. --
北京：高等教育出版社，2016.6
ISBN 978-7-04-045110-8

Ⅰ.①数… Ⅱ.①陈… ②何… Ⅲ.①数据结构-高等学校-教材 Ⅳ.①TP311.12

中国版本图书馆CIP数据核字(2016)第070100号

策划编辑 张 龙	责任编辑 张 龙	封面设计 王 琰	版式设计 马敬茹
插图绘制 杜晓丹	责任校对 高 歌	责任印制 韩 刚	

出版发行	高等教育出版社	网 址	http://www.hep.edu.cn
社 址	北京市西城区德外大街4号		http://www.hep.com.cn
邮政编码	100120	网上订购	http://www.hepmall.com.cn
印 刷	涿州市星河印刷有限公司		http://www.hepmall.com
开 本	787mm×1092mm 1/16		http://www.hepmall.cn
印 张	20.25	版 次	2012年4月第1版
字 数	450千字		2016年6月第2版
购书热线	010-58581118	印 次	2016年6月第1次印刷
咨询电话	400-810-0598	定 价	33.00元

物料号 45110-00

前　言

“数据结构”是计算机类专业的重要专业基础课。它所讨论的知识内容和提倡的技术方法，无论对进一步学习计算机相关领域的其他课程，还是对从事大型信息工程的开发，都有着枢纽的作用。

解决问题往往有多种方法，且不同方法之间的效率可能相差甚远。解决问题方法的效率，与数据的组织方式有关，与空间的利用效率有关，也与方法的巧妙程度有关。本书的主要任务是介绍并探讨有关数据组织、算法设计、时间和空间效率的概念和通用分析方法，帮助读者学会数据的组织方法和现实世界问题在计算机内部的表示方法，针对问题的应用背景分析，选择合适的数据结构，从而培养高级程序设计技能。

本书的特点是从实际应用问题出发，导出各种经典数据结构的定义、实现（存储）方法以及操作实现，并以更丰富的综合应用案例帮助读者增强对理论的感性认识，从而明白这些数据结构为什么存在，以及在什么情况下可以最好地解决什么样的问题。

数据结构的思想和原理是不依赖于编程语言的，但对于每一个抽象概念的具体实现和应用则需要一种编程语言作为载体。本书根据国内多数学校计算机专业教学的实际情况，选择了C语言作为具体实现的语言，并提供了大量可以直接编译运行的源代码。不仅使得学生在学习时容易起步，可以在现有源代码的基础上不断修改扩充，从而解决更为复杂的问题，而且也为IT专业人士提供了方便的经典代码库。

本书第1章介绍了数据结构与算法的基本概念和两者的关联，重点介绍了抽象数据类型和算法复杂度的概念；第2章基本上是对C语言关键内容的复习，为后续章节理解数据结构的实现做准备；第3章介绍了线性表以及最基本的两种应用——堆栈和队列；第4章讨论一种重要的非线性结构——树，重点介绍了二叉树和搜索树，并将查找、哈夫曼树和集合表示等作为树形结构的应用进行了讨论；第5章通过对从海量信息中高效查找关键字问题的再思考，引出对散列表和经典哈希映射技术的讨论；第6章介绍图的各种表示方法和相关算法；第7章讨论了各种经典的排序算法；最后在第8章通过对两个实际生活中提炼出的问题的求解，帮助读者更深刻体会数据结构的应用。

读者可以根据自身的基础选择相应章节进行阅读，熟悉C语言的读者完全可以跳过第2章有关C语言基础的部分。目录中带*号的章节及习题中带*号的题目，是本书的扩展内容，读者可以在学习基础内容之后再阅读扩展内容。

本书作为第2版，除了修订第1版的错误外，还重新整理了全部源代码，提供了部分微视频，并提供了新的在线练习资源。希望读者能通过本书的学习提高实践能力，使数据结构与算法成

为用计算机解决实际问题的有效工具。

本书提供的学习资源如下。

(1) 本书全部源代码及配套电子课件可以通过发送邮件(jsj@pub.hep.cn)免费获取。

(2) 本书练习和习题中的程序设计题目部署在具有在线判题功能的 PTA(Programming Teaching Assistant)平台上,使用说明请阅读附录。读者使用本书封四提供的验证码即可登录 PTA 网站(http://pta.patest.cn)进行在线练习。

(3) PTA 同时接受任课教师的申请,提供自行出题或引用题库(现有千余题可供选用)、组织学生在线练习、考试等管理功能,并提供成绩一键下载等教学数据统计功能。采用本书作为教材的高校教师,可以发送邮件(jsj@pub.hep.cn)申请获得教师的管理权限。

(4) 针对学习难点和重点,书中使用二维码的形式提供了若干微视频,读者使用手机扫描即可观看。这些微视频引自本书作者在"中国大学 MOOC"平台上开设"数据结构"课程。建议读者在阅读本书的过程中,系统学习这门 MOOC 课程,可以取得更好的学习效果。

(5) 与本书配套的《数据结构学习与实验指导(第2版)》包含了实验指导、习题指导等丰富内容,同时大量的练习题目也部署在 PTA 平台上。读者可以使用 PTA 随时检测自己的学习效果与编程能力。

本书由浙江大学计算机科学与技术学院教师编写,由陈越教授组织并统稿。其中,第1、8章由陈越教授编写,第2、3章由何钦铭教授编写,第4章由魏宝刚教授编写,第5、6章由徐镜春副教授编写,第7章由杨枨副教授编写。新版主要由陈越教授与何钦铭教授共同修订,并补充微视频。

在改版过程中,得到各高校教师以及网友的大力支持与帮助,特别在此鸣谢青岛大学周强老师、广东东软学院罗先录老师,以及网友 bds1213、萌面大道、寻雾启示、love 易-水-寒的斧正。书中不当之处在所难免,敬请广大读者批评指正。

编著者

2015 年 12 月

目　录

第1章 概　论

1.1 引子

什么是数据结构？事实上，这个问题在计算机科学界至今没有标准的定义。

如果你的好奇心充分强，不妨打开各种版本的有关“数据结构”的教材首页，看到的会是五花八门的描述。而在你深入阅读本书之前，大多数的描述对你而言可能太过晦涩——例如 Sartaj Sahni 在他的《数据结构、算法与应用》一书中称：“数据结构是数据对象，以及存在于该对象的实例和组成实例的数据元素之间的各种联系。这些联系可以通过定义相关的函数来给出。” Clifford A.Shaffer 在《数据结构与算法分析》一书中的定义是：“数据结构是 ADT（抽象数据类型，Abstract Data Type）的物理实现。”互联网上的中文维基百科写道：“数据结构（Data Structure）是计算机中存储、组织数据的方式。通常情况下，精心选择的数据结构可以带来最优效率的算法。”

作为初学者，让我们暂且把那些由专业术语组成的各种定义抛开，先尝试解决下面几个简单的问题。在解决问题的过程中，或许可以得到对于数据结构的理解。

[**例 1.1**] 书店往往是书的海洋，图 1.1 显示了著名的圣保罗 Livraria da Vila 书店一角。如果你是书店的主人，该如何摆放你的书，才能让读者很方便地找到你手里这本《数据结构》？

[**分析**] 解决的办法有很多，下面只列举 3 种最简单的。

方法 1：随便放。

这种方法使得放书非常方便，任何时候有新书进来，哪里有空就把书插到哪里。但是这种方法显然使得查找非常痛苦。最不走运的时候，是你的书架上根本没有这本书，但是你需要翻遍整个书架的每一本书，才能确定地说真的找不到。

方法 2：按照书名的拼音字母顺序排放。

这种方法使得查找方便了一些。我们可以随便抽取一本书，检查书名的拼音首字母。例如书名是 L 开头的《离散数学》，我们就知道以 S 开头的《数据结构》一定排在 L 的后面；再从它后面随便抽取一本书，例如是 W 开头的《网络技术基础》，那么《数据结构》一定排在 W 的前面，我们的查找范围就迅速缩小到 L 和 W 之间的区域内。

但是这种方法会使得新书的插入成为一种痛苦。如果买的新书是 Z 开头的《“做中学”程序员攻略》还好，如果新买的一本是 A 开头的《阿 Q 正传》就惨了，为了给新书腾出空间，要把多少本书向后挪动啊！

图 1.1 圣保罗的 Livraria da Vila 书店一角

方法 3:把书架划分成几块区域,每块区域指定摆放某种类别的图书;在每种类别内,按照书名的拼音字母顺序排放。

这种方法与方法 2 相比,无论是查找还是插入,工作量都减少很多,因为类别一旦确定,要处理的书架范围就大大缩小了。但是仍然存在问题——因为我们不可能事先知道每种类别的图书会有多少本,所以划分区域的时候最好给每种类别预留足够的新书空间,这可能造成空间上的浪费。

另一方面,类别分得越细,属于同一类的书就越少,在某一类内部查找或插入的工作量就越小。但是如果类别太多,要找到某一类所在的区域又会成为一件麻烦事…… 你还有更好的解决方案吗?

[**例 1.2**] 写程序实现一个函数 PrintN,使得传入一个正整数为 N 的参数后,能顺序打印从 1 到 N 的全部正整数。

[**分析**] 只要略有编程基础的人都可以很容易实现这个函数。代码 1.1 给出了一个用 C 语言循环语句实现的版本。

```
void PrintN(int N)
{  /* 打印从 1 到 N 的全部正整数 */
   int i;
   for(i=1;  i<=N;  i++)
      printf("%d\n",  i);
   return;
}
```

代码 1.1 用 C 语言循环语句实现的 PrintN 函数

另一个用C语言递归语句实现的版本看上去更简洁，甚至不需要临时变量的帮助，如代码1.2所示。

```
void PrintN(int N)
{/* 打印从 1 到 N 的全部正整数 */
    if(N>0){
        PrintN(N-1);
        printf("%d\n", N);
    }
}
```

代码 1.2 用C语言递归语句实现的 PrintN 函数

问题看上去很简单，上述两种方法似乎都可以完成任务。然而，事实真的如此吗？我们可以运行代码1.3，来比较一下两种实现方法。

```
#include <stdio.h>

void PrintN(int N);

int main()
{   /* 读入整数 N，并调用 PrintN 函数 */
    int N;

    scanf("%d", &N);
    PrintN(N);
    return 0;
}
```

代码 1.3 函数 PrintN 的测试程序

把代码1.1和代码1.2分别（不是同时）贴到代码1.3的尾部，分别编译运行。测试输入N为100、1000、10000、100000的情况——如果还不能发现问题，那么继续测试更大的N……终于，我们将发现，对于充分大的N，代码1.2中的递归函数拒绝工作了！而此时代码1.1仍然正常运行。

为什么会这样？请读者思考其中的原因。

[例 1.3] 一元多项式的标准表达式可以写为：$f(x)=a_0+a_1x+\cdots+a_{n-1}x^{n-1}+a_nx^n$。现给定一个多项式的阶数 n，并将全体系数 $\{a_i\}_{i=0}^{n}$ 存放在数组 $a[\,]$ 里。请写程序计算这个多项式在给定点 x 处的值。

[分析] 最直接的办法是根据多项式的标准表达式 $f(x)=\sum_{i=0}^{n} a_i x^i$ 通过循环累计求和来实现这个函数。代码 1.4 给出了这个直接实现的版本。

```
double f(int n, double a[], double x)
{   /* 计算阶数为 n,系数为 a[0]... a[n]的多项式在 x 点的值 */
    int i;
    double p=a[0];
    for(i=1; i<=n; i++)
        p+=a[i]*pow(x, i);
    return p;
}
```

代码 1.4 计算多项式函数值的直接法

然而早在 800 年前,中国南宋的数学家秦九韶就提出了一种更快的算法,他通过不断提取公因式 x 来减少乘法的运算次数,把多项式改写为:

$$f(x)=a_0+x(a_1+x(\cdots(a_{n-1}+x(a_n))\cdots)) \qquad \text{(公式 1.1)}$$

代码 1.5 给出了按照公式 1.1 编程的多项式求值算法。

```
double f(int n, double a[], double x)
{   /* 计算阶数为 n,系数为 a[0],…,a[n]的多项式在 x 点的值 */
    int i;
    double p=a[n];
    for(i=n; i>0; i--)
        p=a[i-1]+x*p;
    return p;
}
```

代码 1.5 计算多项式函数值的秦九韶法

微视频 1-1 clock()工具的使用

看上去代码 1.4、代码 1.5 两个版本的程序一样简单,都只要 5 行语句就可以了。问题是,秦九韶算法究竟比简单的直接算法快了多少?要回答这个问题,我们需要先学习 clock()工具的使用。

要获得一个程序的运行时间,常用的方法是调用头文件 time.h,其中提供了 clock()函数,可以捕捉从程序开始运行到 clock()被调用时所耗费的时间。这个时间单位是 clock tick,即"时钟打点",在 C/C++中定义的数据类型是 clock_t。同时还有一个常数 CLK_TCK(或是 CLOCKS_PER_SEC),给出了机器时钟每秒所走的时钟打点数。代码 1.6 给出了一个常用范例。

```
#include <stdio.h>
#include <time.h>

clock_t  start,  stop;  /* clock_t 是 clock()函数返回的变量类型 */
double  duration;  /* 记录被测函数运行时间,以秒为单位 */

int main()
{ /* 不在测试范围内的准备工作写在 clock()调用之前 */

    start=clock();  /* 开始计时 */
    MyFunction();   /* 把被测函数加在这里,使用时这个函数必须被替换 */
    stop=clock();   /* 停止计时 */
    duration=((double)(stop-start))/CLK_TCK;  /* 计算运行时间 */
    /* 注意 CLK_TCK 是机器时钟每秒所走的时钟打点数, */
    /* 在某些 IDE 下也可能叫 CLOCKS_PER_SEC。      */

    /* 其他不在测试范围的处理写在后面,例如输出 duration 的值 */
    return 0;
}
```

代码 1.6 测试函数 function()的运行时间

下面我们可以通过一个具体多项式函数值的计算,来比较秦九韶算法与直接法的效率差别:令 $f(x)=\sum_{i=0}^{9} i\cdot x^i$,计算 $f(1.1)$ 的值。代码 1.7 给出了测试函数。

```
#include <stdio.h>
#include <time.h>
#include <math.h>

clock_t start,  stop;
double duration;
#define MAXN 10  /* 多项式最大项数,即多项式阶数+1 */
#define MAXK 1e7 /* 被测函数最大重复调用次数 */

double f1(int n, double a[],  double x)
{ /* 代码 1.4 的算法 */
    int i;
    double p=a[0];
    for(i=1;  i<=n;  i++)
```

```
        p+=(a[i] * pow(x, i));
    return p;
}

double f2(int n, double a[], double x)
{ /* 代码1.5的算法 */
    int i;
    double p=a[n];
    for(i=n; i>0; i--)
        p=a[i-1]+x*p;
    return p;
}

void run(double(*f)(int, double*, double), double a[], int case_n)
{/*此函数用于测试被测函数(*f)的运行时间,并且根据case_n输出相应的结果 */
 /*case_n是输出的函数编号:1代表函数f1;2代表函数f2                    */
    int i;

    start=clock();
    for(i=0; i<MAXK; i++)  /*重复调用函数以获得充分多的时钟打点数*/
        (*f)(MAXN-1, a, 1.1);
    stop=clock();

    duration=((double)(stop-start))/CLK_TCK;
    printf("ticks%d=%f\n", case_n, (double)(stop-start));
    printf("duration%d=%6.2e\n", case_n, duration);
}

int main()
{
    int i;
    double a[MAXN];  /*存储多项式的系数*/

    /*为本题的多项式系数赋值,即a[i]=i */
    for(i=0; i<MAXN; i++)a[i]=(double)i;

    run(f1, a, 1);
```

```
    run(f2, a, 2);
    return 0;
}
```

代码 1.7 测试多项式求值函数的运行时间

因为我们要比较两种算法的效率，又不想把相似的测试代码重复写两遍，所以把测试函数运行时间的代码写成了一个函数 run，将被测函数(* f)作为参数(函数指针类型)传入，测试其运行时间，并且根据 case_n 输出相应的结果。

注意到被测函数运行一次所花费的时间有可能小于两次时钟打点的间隔，这时我们就有可能得到 stop-start=0 的情况，从而测不出真正的运行时间。

解决这个问题的方法是，让被测函数重复运行充分多次，使得测出的总的时钟打点间隔充分长，最后计算被测函数平均每次运行的时间即可。在代码 1.7 中，我们令函数运行10^7次，读者可以根据自己机器配置选择其他的 MAXK 值。

注意到测试结果取决于机器的配置，在不同的机器上运行，得到的具体数据是不一样的。但可以肯定的是，秦九韶算法的计算速度明显比直接法快了一个数量级。

为什么会这样？请读者思考其原因。

通过对上面 3 个例子的研究，我们可以发现，即使解决一个非常简单的问题，往往也有多种方法，且不同方法之间的效率可能相差甚远。解决问题方法的效率，跟数据的组织方式有关(如例 1.1)，跟空间的利用效率有关(如例 1.2)，也跟算法的巧妙程度有关(如例 1.3)。

本章将要向大家介绍的，就是有关数据组织、算法设计、时间和空间效率的概念和通用分析方法，是后续所有数据结构及其相关算法的基础。

1.2 数据结构

1.2.1 定义

从例 1.1 中我们发现用不同方法摆放图书，会直接影响查找、插入等工作的效率。在计算机的世界里，“图书”就是待处理的“数据对象”，“查找”、“插入”等工作就是对数据进行的“操作”，完成这些操作所用的方法就是“算法”。

“数据结构”的定义，首先应该包含数据对象在计算机中的组织方式——这类似于图书的摆放方法。另一方面，数据对象必定与一系列加在数据对象上的操作相关联，就如我们在书架上摆放图书是为了能找到想要的书，或者是插入一本新买的书。我们讨论数据对象的各种不同的组织方式，是为了得到处理这些数据对象的最高效的算法。所以我们在讨论“数据结构”这个概念的时候，关心的不仅仅是数据对象本身以及它们在计算机中的组织方式，还要关心与它们相关联

的一个操作集,以及实现这些操作的最高效的算法。

关于数据对象在计算机中的组织方式,其实还包含了两个概念:一是数据对象集的逻辑结构;二是数据对象集在计算机中的物理存储结构。

例如我们把一本书看成一个数据对象,如果所有的书是一本挨一本排成一大排的,从最左边第1本书开始向右顺序编号,每本书的位置可以由它的编号唯一确定,那么这个数据对象集的逻辑结构就被称为是"线性(Linear)"的,因为数据对象都串在一条线上,并且编号跟书是"1对1"的关系。当我们把这些书的信息存进计算机时,可以设计一个结构体来记录一本书,而书的集合可以用结构体的数组来存储,也可以用结构体的链表来存储。数组或者链表就是数据对象集在计算机中的物理存储结构。

在后面的章节中,大家还会见识到更多样的数据对象逻辑结构。例如在例1.1的解决方法3中,把图书先按类别编号,在同一类中再按字母序编号,那么一个类别编号就对应多本图书,类别编号跟书是"一对多"的关系。这种数据对象集的逻辑结构就是"树(Tree)"状的,将在第4章中讨论。如果还需要统计买书人的兴趣关系,即买了某本图书的人同时还买了哪些其他的书,那么这些图书之间就构成了一个"多对多"的关系网,这种逻辑结构被称为"图(Graph)",是第6章中将要介绍的内容。而如何在计算机中有效地存放"树"和"图"这样的结构,则是这两章要讨论的另一个有趣的话题。

1.2.2 抽象数据类型

顾名思义,抽象数据类型(Abstract Data Type)是一种对"数据类型"的描述,这种描述是"抽象"的。

首先,"数据类型"描述两方面的内容:一是数据对象集;二是与数据集合相关联的操作集。

"抽象"的意思是指,我们描述数据类型的方法是不依赖于具体实现的,即数据对象集和操作集的描述与存放数据的机器无关、与数据存储的物理结构无关、与实现操作的算法和编程语言均无关。简而言之,抽象数据类型只描述数据对象集和相关操作集"是什么",并不涉及"如何做到"的问题。

[例1.4] "矩阵"的抽象数据类型定义

类型名称:矩阵(Matrix)

数据对象集:一个 $m \times n$ 的矩阵 $\boldsymbol{A}_{m\times n}=(a_{ij})(i=1,\cdots,m;j=1,\cdots,n)$ 由 $m \times n$ 个三元组 $<a,i,j>$ 构成,其中 a 是矩阵元素的值,i 是元素所在的行号,j 是元素所在的列号。

操作集:对于任意矩阵 A、B、C ∈Matrix,以及整数 i、j、M、N,仅列出几项有代表性的操作。更多关于矩阵的操作不是我们讨论的重点,故在此略去。

1. Matrix Create(int M, int N):返回一个 M×N 的空矩阵;
2. int GetMaxRow(Matrix A):返回矩阵 A 的总行数;
3. int GetMaxCol(Matrix A):返回矩阵 A 的总列数;
4. ElementType GetEntry(Matrix A, int i, int j):返回矩阵 A 的第 i 行、第 j 列的元素;
5. Matrix Add(Matrix A, Matrix B):如果 A 和 B 的行、列数一致,则返回矩阵 C=A+B,否则

返回错误标志；

6. Matrix Multiply(Matrix A, Matrix B)：如果 A 的列数等于 B 的行数，则返回矩阵 C=AB，否则返回错误标志；

7. ……

通过例 1.4，我们可以这样理解“抽象”的含义：

(1) 当我们在数据对象集中描述矩阵元素的时候，刻画了它的取值和二维位置，但是这个描述并没有规定矩阵元素是整数还是浮点数，这个元素甚至可能是一个特殊的结构体！但无论什么类型的矩阵元素，都可以用这个数据对象集来描述。相应于数据对象的抽象描述，操作 4 的类型描述被写为 ElementType，即“元素类型”，意味着当具体实现某一种矩阵的时候，这个类型可以用相应的具体类型替换掉。而其他操作如加法、乘法的具体实现也可能需要随着元素类型的不同而不同——想一想，如果矩阵元素是某种特殊的结构体，我们怎么定义两个结构体的相加？这样的描述方法，忽略元素类型这种细节问题，适用于任何一种类型的矩阵。

(2) 对于数据对象的描述不依赖于其在计算机中具体的存储方法。例如我们可以用二维数组存储，也可以用一维数组存储，还可以用十字交叉的链表来存储一个矩阵。抽象数据类型的描述不涉及这样的细节，但是适用于任何具体的存储方式。

(3) 在描述操作的时候，我们只描写了这个操作是做什么用的，并不涉及操作的具体实现方法。例如矩阵相加的时候，我们是先按行加还是先按列加？抽象数据类型的描述也不涉及这样的细节，更与实现操作的编程语言没有关系。

综上所述，抽象数据类型描述的重要特征是“抽象”。抽象是计算机求解问题的基本方式和重要手段，它使得一种设计可以应用于多种场景。而且通过抽象可以屏蔽底层的细节，使设计更加简单、理解更加方便。

抽象数据类型的描述方法与面向对象的思想是一致的，它把数据对象和相关操作封装在一起，对于需要调用这个数据类型的用户而言，无论内部的具体实现如何改变，只要对外描述的接口不变，就不影响使用。

在后面的章节中，每当我们介绍一种新的数据结构时，会首先用抽象数据类型来描述这个结构，以方便读者理解。

1.3 算法

1.3.1 定义

“算法”(Algorithm)一词是由 Algorism 衍生而来，而 Algorism 源自一本波斯数学教材，原意为“算术”。算法的设计是一门艺术。解决同一个问题，一般有多种算法，但漂亮的算法与其他算法相比往往有天壤之别。