Data Structures and Algorithms in Multiple-Task

多任务下的数据结构与算法

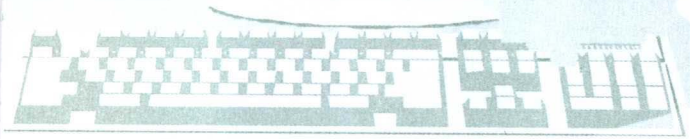
◎ 周伟明 / 著

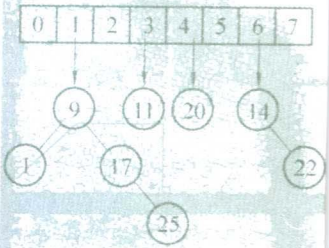
华中科技大学出版社
http://www.hustp.com

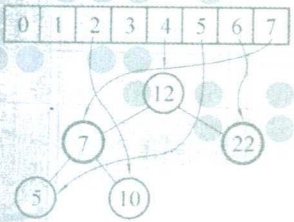
多任务下的数据结构与算法

Data Structures and Algorithms in Multiple-Task

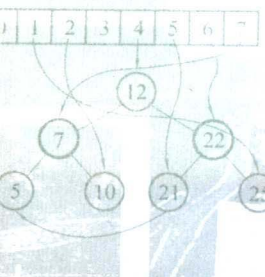
多任务下的数据结构与算法







◎ 周伟明／著



华中科技大学出版社

图书在版编目(CIP)数据

多任务下的数据结构与算法/周伟明　著
武汉:华中科技大学出版社,2006 年 4 月
ISBN 7-5609-3676-8

Ⅰ.多…
Ⅱ.周…
Ⅲ.数据结构
Ⅳ.TP31

多任务下的数据结构与算法　　周伟明　著

策划编辑:王红梅
责任编辑:刘　勤　　封面设计:刘　卉
责任校对:代晓莺　　责任监印:张正林

出版发行:华中科技大学出版社
武昌喻家山　邮编:430074　电话:(027)87557437

录　排:华中科技大学惠友文印中心
印　刷:湖北新华印务有限公司

开本:787×960　1/16　印张:24.75　插页:4　字数:466 000
版次:2006 年 4 月第 1 版　印次:2006 年 4 月第 1 次印刷　定价:58.00 元(含 1CD)
ISBN 7-5609-3676-8/TP・602

内容简介

本书和传统同类书籍的区别是除了介绍基本的数据结构容器如栈、队列、链表、树、二叉树、红黑树、AVL 树和图之外，引进了多任务；还介绍了将任意数据结构容器变成支持多任务的方法；另外，还增加了复合数据结构和动态数据结构等新内容的介绍。在复合数据结构中不仅介绍了哈希链表、哈希红黑树、哈希 AVL 树等容器，还介绍了复合数据结构的通用设计方法；在动态数据结构中主要介绍了动态环形队列、动态等尺寸内存管理算法。在内存管理中介绍了在应用程序层实现的内存垃圾回收算法、内存泄漏检查和内存越界检查的方法等。本书选取的内容均侧重于在实际中有广泛应用的数据结构和算法，有很好的商业使用价值。

本书大部分章节中都列举并介绍了应用实例，如用 AVL 树等容器实现的搜索引擎、用数组实现 HOOK 管理、用链表实现的短信息系统中的 CACHE 管理、用哈希表实现 WebServer 中的 CACHE 文件管理和用哈希 AVL 树实现抗 DoS/DDoS 攻击等。

书中重点介绍了软件的各种质量特性如时间效率和空间效率之间的关系，介绍了如何在各种质量特性间取得均衡的原则，并介绍了各种数据结构算法的应用场合和范围。

本书介绍的所有数据结构及算法都以不同复杂程度给出其编码实现。为了便于读者自学，每章末附有小结和思考练习题。

本书可供高校计算机及相关专业作为教学参考书，对从事软件开发与应用的科研人员、工程技术人员以及其他相关人员也具有较高的参考价值。

自 序

软件的核心技术是什么？一个软件要做出来后很难模仿才能称之为拥有核心技术。软件上市后，只要使用一下便知道有哪些功能，所以功能性需求是非常容易模仿的。比较难模仿的有两个方面：一是软件设计，二是数据结构与算法。好的算法可以申请专利，用作保护知识产权和限制竞争对手的重要手段，由此可见算法在软件中的重要意义。软件研发人员和测试人员的最大区别在于研发人员在数据结构与算法方面要掌握得更好。

十年前，当我初次来到深圳开始职业软件生涯时，出于对数学的热爱，闲暇时经常看一些数据结构与算法方面的书籍和资料，常从其中受到启发。经过七年的日积月累后，忽然发现CPU的速度已快到达极限，多核CPU已投入实际使用，未来将是多核CPU的天下，对多任务编程提出了更高的要求，而目前数据结构与算法方面的书籍均没有涉足多任务方面，乃下定决心写作本书，在历时三年的写作过程中又有一些新的心得写入了本书中。

数据结构与算法已经成为软件开发工程师的必备的基础知识之一，在学校里，它已经成为计算机学科的重要课程，同时也成为许多其他专业的热门选修课，社会上大多数公司在招聘软件开发人员时必然会考察应聘人员数据结构与算法的掌握程度，并将此作为衡量应聘者水平的重要依据。本书是为那些已从事软件开发或即将从事软件开发的人员而写，也可以供专业人士参考。本书注重实践，注重软件的设计与实现，为软件开发人员向职业化方向发展和进一步提升打好基础。

本书重点讲解了多任务方面的内容，讲解了使数据结构支持多任务的算法。讲解了很多新的数据结构与算法，也讲解了数据结构的设计思想，如复合数据结构和动态数据结构的设计思想，本书中的哈希红黑树和哈希AVL树的设计就是复合数据结构设计的典范，而

动态等尺寸内存管理算法则是动态数据结构设计的代表。传统的垃圾收集算法在进行垃圾内存收集时应用程序会被停住，本书提供的算法在进行垃圾内存收集时不影响应用程序的运行，并且可以在应用程序层使用，效率很高。本书中的实例也都是选用比较热门的商业应用如搜索引擎、短信息系统、抗DoS攻击，WebServer等。

为了使更多的人能看懂此书，本书代码基本都使用C语言来实现，只有少数代码使用C++实现，因此具有C语言基础知识就可以阅读本书，本书的代码也很容易改写成C++，喜欢使用C++的朋友可以按照光盘中的版权申明要求将本书代码改成用C++实现，本书光盘附带的代码是可以免费使用和修改的。

本书最终得以完成有赖于许多人的帮助。首先感谢我的妻子和两岁多的儿子，在忘我投入写作的无数个周末和假日，儿子也渐渐地从襁褓长大到能四处探险。在写作过程中，他虽经常给我捣乱，但也给我带来很多欢乐，他的成长也伴随了这本书的成型。为了使我能完成自己的夙愿，妻子放弃了自己的工作，默默承担起照顾孩子和家庭的重担，有时还要帮我做一些书稿的录入工作。

感谢华中科技大学出版社对此书的重视，感谢编辑王红梅和刘勤老师等人，他们的工作大大改善了本书的质量。

感谢最初引导我进入软件行业的邓耀斌、唐玉天两人。感谢当年在 Santa Cruz 一起工作过的Neil Readshaw，九年前正是在他的指导和帮助下，我对多任务下的数据结构和算法的理解有了较大的飞跃。

本书的写作过程中参考了国内外许多有关数据结构与算法方面的文献，在此，我谨向那些卓有建树的专家、学者致以诚挚的感谢！本书的写作还得到了许多朋友的帮助和鼓励，温野、唐文宁、罗巍等人帮助校对过部分章节的内容，并给出了很好的改进建议。一些朋友多年来持续给予了我鼓励和帮助，包括章俊良、邓耀斌、张莉、洪建明等，章俊良博士还帮我查阅过关于哈密顿圈算法的一些论文资料，还有很多人请恕我在这里不能把他们一一列出，在此对他们一并表示感谢！

周伟明 2006年4月

目　录

CONTENTS

1 绪 论

本章介绍了一些阅读后续内容所需的预备知识，包括 C 语言编程常见问题分析、任意类型数据处理和自任务的一些基础知识，软件设计简介等。

1.1 引言

本书建议读者：

有 C 语言基础知识，如果读过一本或多本其他数据结构与算法书籍则更佳。

本书多任务那一章要求有多任务编程的基础知识。

当我准备开始写这本书的时候，曾经和朋友谈论过，当时朋友很惊讶地说："数据结构与算法发展到今天，已有的相关书籍如此之多，并且在 C++中还有一个标准模板库 STL，难道你还能写出新的东西吗？为什么不写其他方面的书呢？"的确，数据结构与算法发展到现在，一般软件工程师经常要用的数据结构与算法不知有多少人写过了。还有 STL，也有很多家公司实现了 Free 的源代码，甚至有些公司的编码实现被一些人奉为优秀代码的典范，在此情况下要写一本数据结构与算法的书籍确实不是一件容易的事情。在经过充分的权衡后，我还是决定要写这本关于数据结构与算法方面的书，主要基于以下考虑。

1）目前数据结构与算法虽然看上去很成熟了，但随着软件技术越来越多地应用到工程实践中，新的应用会对它提出很多新的需求。特别是多核 CPU 的发展对多任务编程提出了更高的要求，这样数据结构的发展空间还是很大的，如 STL 目前还不能支持多任务。举个简单例子，大家可以考虑一下，如果一个链表有多个任务在操作，有些任务在做插入、删除操作，有些任务在遍历，能不能做到在遍历的过程中进行插入、删除操作？目前在 STL 中肯定是做不到的。本书会有专门的章节来讲述多任务下的数据结构与算法设计。

2) 现在虽然有很多 Free 的数据结构算法源代码，但是这些代码普遍存在的一个问题是可读性较差，没有注释，而且很多代码的命名风格很偏，即使是资深的专业人士看起来也很费力。本书的代码均追求高的可读性，体现良好的程序设计风格。

3) 本书探讨了一些复合数据结构的设计概念，如哈希链表和哈希红黑树。这些数据结构在服务器软件等方面会有很高的实用价值。

4) 本书会对所讲到的绝大部分数据结构与算法给出完整的源代码，即使是复杂的 AVL 树和红黑树也给出了完整代码。

5) 有一句网络名言叫“专业的程序员用 C 语言”，没错，本书代码主要是用 C 语言实现，只有少部分用 C++实现。其主要的设计思想还是采用面向对象的思想，C 语言实现的部分可以很方便地改写成 C++语言，但对大家学习来说可以屏蔽 C++语言复杂的语法，比 C++语言更容易理解和掌握，同时也让初学者明白面向对象与语言无关，并不是只有 C++、Java 这类支持面向对象的语言才可以写面向对象的程序。(其实许多初学者用 C++、Java 写的程序都不是面向对象的。)

6) 本书虽然是专门为专业人士而撰写的，但即使是在校学生也可以很容易看懂，只要有 C 语言基础知识就可以阅读本书。

7) C++标准委员会把模板引入 STL 中，使得 C++中的模板使用非常普及。实际应用中在内存受限的情况下，模板库开销过大使得人们将注意力集中到如何解决类型无关指针的问题。本书是用 void 指针来实现类型无关的，虽然理论上类型转换方面没有模板安全，但数据结构方面的类型转换实际上一般是不会有问题的，因为类型转换错误，执行马上就会异常，只要代码通过测试是不会有问题的。同时模板生成的执行代码庞大，对于内存受限系统，STL 的裁剪是一个非常令人头疼的问题。

对于模板库，或者任何一种技术都得看应用的场合。其实，任何一种思想和方法的滥用都会导致严重的后果，就像设计模式的滥用一样。比如，COM 的滥用导致现在软件质量的严重下降。COM 的设计可以说是软件史上最严重的一次设计失误，COM 的设计违反了软件设计的基本原理，但就这样一个设计居然被推广到整个业界使用，造成的危害实在太大了。我们可以看到，很多常用的软件的 BUG 越来越多，动不动就死掉或要发送错误报告什么的，很多情况估计都是拜 COM 所赐。虽然微软现在再也不敢提 COM 了，但它的副作用短期内仍然难以消除，特别是在中国，谈到设计时很多人现在还在言必称组件。

本书还有一些小的特点请读者在阅读本书时自行体会，限于时间和水平，本书的不足之处和缺陷相信也很多，请大家不吝赐教。

1.2 C 语言编程常见问题分析

C 语言中的一般语法和一些编程技巧在很多书里都讲过了，下面主要讲一些别的

书很少讲到、但是又非常重要的问题。这些都是作者在编程过程中总结出的一些经验教训，可以说职业程序员每天编程时都要遇到这些问题。

1.2.1 参数校验问题

在C语言的函数中，一般都要对函数的参数进行校验，但是有些情况下不在函数内进行校验，而由调用者在外部校验，到底什么情况下应该在函数内进行校验，什么情况下不需要在函数内进行校验呢？下列原则可供读者参考。

1) 对于需要在大的循环里调用的函数，不需要在函数内对参数进行校验。

例如链表的逐个遍历函数 void *List_EnumNext(LIST *pList)。

在链表的逐个遍历函数里，要不要对pList参数的合法性进行校验呢？答案是否定的。为什么呢？因为链表的逐个遍历函数通常是要在一个循环里使用的，比如一个链表有10 000个节点，逐个遍历就要遍历10 000次。如果上面的函数对参数pList进行了校验，那么对整个链表的逐个遍历过程将校验10 000次，不如由调用者在调用函数前校验一次就够了。因此，像这种可能频繁地被调用，且在外面校验只要校验一次就够的函数参数是不需要在函数内部进行校验的。

2) 底层的函数调用频度都比较高，一般不校验。

3) 对于调用频度低的函数，参数要校验。

4) 执行时间开销很大的函数，在参数校验相对整个函数来讲占的比例可以忽略不计的情况下，一般最好对函数参数进行校验。

5) 可以大大提升软件的稳定性时，函数参数要进行校验。

1.2.2 return语句的问题

在函数里，使用return语句可能也是一个值得探讨的问题。有些人认为应该让函数的出口尽量的少，因此要减少return语句的使用；特别是在函数中有分配资源操作时，在之后的每个return语句里都需要进行对应的释放操作，在编程时很容易出现遗漏而出现资源泄漏。但是我们也知道，如果要减少return语句，又会带来另外一些问题。首先，很多情况下要减少return语句会增加编程的难度；其次，有时候减少了return语句又使得大括号嵌套的层数增加不少，使得代码行长度增加不少，很容易因超过80字符而使得代码阅读困难。

那么，到底什么情况下可以减少return语句，什么情况下又不能减少return语句呢？下列原则供读者参考。

1) 对参数进行校验，校验失败，一般要使用return语句，这样做可使程序逻辑清晰，阅读也方便，又减少了大括号嵌套的层数。

2) 对于函数内部的不同出错情况，要有不同的return语句；否则对外部调用者来

说，无法区分出错的真正原因。

3) 对于函数内部的同类出错情况，尽量只使用一个 return 语句，即尽量不要让两个 return 语句返回相同的返回值。

1.2.3 while 循环和 for 循环的问题

1. 两种循环在构造死循环时的区别

用 while 构造死循环时，一般会使用 while(TRUE)来构造死循环；而用 for 来构造死循环时，则使用 for(;;)来构造死循环。这两个死循环的区别是：while 循环里的条件被看成表达式，因此，当用 while 构造死循环时，里面的 TRUE 实际上被看成永远为真的表达式，这种情况容易产生混淆，有些工具软件如 PC-Lint 就会认为出错了，因此构造死循环时，最好使用 for(;;)来进行。

2. 两种循环在普通循环时的区别

对一个数组进行循环时，一般来说，如果每轮循环都是在循环处理完后才讲循环变量增加的话，使用 for 循环比较方便；如果循环处理的过程中就要将循环变量增加时，则使用 while 循环比较方便；还有在使用 for 循环语句时，如果里面的循环条件很长，可以考虑用 while 循环进行替代，使代码的排版格式好看一些。

1.2.4 if 语句的多个判断问题

在对参数进行校验时，经常需要校验函数的几个参数，是对每个参数都使用一个单独的 if 语句进行一次校验，还是多个语句都放在一个 if 语句里用逻辑或运算来进行校验呢？还是用实例来说明吧。

例 1-1 参数校验举例。

```
TABLE *CreateTable1( int nRow, int nCol )
{
    if ( nRow > MAX_ROWS )
    {
        return NULL;
    }
    if ( nCol >= MAX_COLS )
    {
        return NULL;
    }
    ……
}
```