数据结构
使用C语言

蔡明志著

科学出版社

北京希望电脑公司C语言技术丛书

数据结构——使用C语言

蔡明志 著

沈铁齐 鞠玉兰 改编

谷晓鸥 刘 军 审校

科学出版社

1993

(京) 新登字 092 号

内容提要

本书详细系统地介绍了数据结构的基础原理及实现算法，并在此基础上给出了 C 语言程序。本书内容全面，通俗易懂，不仅适合于初学者，而且对于有一定数据结构基础的读者也是一本很好的参考读物。

本书包括算法分析、数组、堆栈和队列、链表、动态存储器管理、树结构、图结构、排序与查找、二叉查找树、索引技术等内容。

本书叙述清晰，使用方便，是 C 语言程序设计人员极为有用的工具书，是计算机应用人员、大专院校师生必备的参考书。

需要本书的用户，请直接与北京 8721 信箱联系，邮政编码：100080，电话：2562329

版 权 声 明

数据结构--使用 C 语言

蔡明志 著

沈铁齐 鞠玉兰 改编

谷晓鸣 刘 军 审

责任编辑 杨家福

科学出版社出版

北京东黄城根北街 16 号

邮政编码：100717

双青印刷厂印刷

新华书店北京发行所发行 各地新华书店经售

*

1993 年 8 月第 一 版 开本：787×1092 1/16

1993 年 8 月第一次印刷 印张：21 1/4

印数：1—5000 字数：494 000

ISBN 7-03-003919-X / TP · 309

定价：29.00 元

前　言

数据结构是从事计算机科学及其应用的科技人员必须掌握的重要基础知识，也是计算机科学教学中一门核心课程.

本书是根据蔡明志先生（台湾）编著的《资料结构 — 使用 C 语言》一书改编而成的．本书在改编过程中力图在忠实原著的基础上，对原著中不正确之处做了适当的修改.

全书共分十章，内容丰富，通俗易懂．每一章都先讲述算法的步骤，然后用 C 语言来编写程序．各章都备有适量的思考题供读者练习以加深理解.

在本书的改编过程中得到了北京希望电脑公司秦人华老师的支持和指导，在此谨表谢意.

限于改编者的水平，书中错误和不妥之处，敬请读者批评指正.

1993 年 8 月

目 录

第一章 算法分析 …… 1
1.1 算法 …… 1
1.2 Big-oh …… 3
1.3 魔术方阵 …… 5
1.4 思考题 …… 8
1.5 程序设计 …… 10
第二章 数 组 …… 14
2.1 数组的表示法 …… 14
2.2 稀疏矩阵 …… 18
2.3 多项式表示法 …… 21
2.4 上三角形和下三角形表示法 …… 23
2.5 思考题 …… 24
第三章 堆栈与队列 …… 26
3.1 堆栈和队列基本概念 …… 26
3.2 堆栈的插入与删除 …… 26
3.3 队列的插入与删除 …… 27
3.4 堆栈与队列的应用 …… 30
3.5 多个堆栈 …… 33
3.6 思考题 …… 34
3.7 程序设计 …… 35
第四章 链表 …… 56
4.1 链表 …… 56
4.2 循环链表 …… 65
4.3 双向链表 …… 71
4.4 多项式相加 …… 72
4.5 思考题 …… 77
4.6 程序设计 …… 78
第五章 动态存储器管理 …… 128
5.1 最先适合法与最优满足法 …… 129
5.2 边界标志法 …… 131
5.3 伙伴系统 …… 134
5.4 思考题 …… 146
第六章 树结构 …… 147
6.1 树结构的一些术语 …… 147
6.2 二叉树 …… 148

6.3 二叉树的表示方法 …… 150
6.4 二叉树遍历 …… 151
6.5 穿线二叉树 …… 153
6.6 如何将一般树化为二叉树 …… 157
6.7 其他论题 …… 157
6.8 思考题 …… 161
6.9 程序设计 …… 164
第七章 图结构 …… 187
7.1 图的一些术语 …… 188
7.2 图数据结构表示法 …… 190
7.3 图遍历 …… 193
7.4 生成树 …… 197
7.5 最短路径 …… 201
7.6 拓扑排序 …… 204
7.7 思考题 …… 207
7.8 程序设计 …… 209
第八章 排序与查找 …… 214
8.1 冒泡排序 …… 214
8.2 选择排序(selection sort) …… 215
8.3 谢耳排序(shell sort) …… 216
8.4 二叉树排序(binary tree sort) …… 217
8.5 基数排序 …… 218
8.6 外部排序 …… 221
8.7 顺序查找 …… 223
8.8 二叉查找 …… 224
8.9 插补法查找 …… 225
8.10 斐波纳契查找 …… 225
8.11 思考题 …… 229
8.12 程序设计 …… 229
第九章 符号表 …… 252
9.1 二叉查找树 …… 252
9.2 动态树表 …… 259
9.3 高度平衡二叉树 …… 261
9.4 杂凑表 …… 280
9.5 思考题 …… 286
9.6 程序设计 …… 287
第十章 索引技术 …… 309
10.1 柱面一盘面索引 …… 309
10.2 杂凑索引 …… 312

10.3 树索引 …… 313
10.4 tric 索引 …… 323
10.5 多重链表文件与倒排文件 …… 328
10.6 思考题 …… 330
参考文献 …… **333**

第一章　算法分析

1.1　算　　法

在未进入数据结构主题之前，让我们先谈谈何谓算法及如何分析算法，以了解其所需要的执行时间.

在处理问题的过程中，一般人都会先将问题描述清楚以后，利用伪语言(pseudo－language)来写算法．常见的伪语言有 SPARKS, Pidgin－ALGOL, Pascal－like 等语言．算法写完以后，再利用高级语言或低级语言来执行.

何谓算法(algorithm)呢?算法是有限命令的集合，为了解决某一特定的任务．算法有下列几点特性:

(1) 输入(input): 可以有零个或多个以上的输入数据.

(2) 输出(output): 至少有一个输出数据.

(3) 明确性(definiteness): 每个命令必须要明确，不能模棱两可.

(4) 有限性(finiteness): 算法必须在经过有限的步骤时停止.

(5) 有效性(effectiveness): 可以在纸上做些分析以估计其结果.

从上述的特性可知程序(procedure)与算法不同，程序可以不具有限性，可能为无穷的循环，例如操作系统的任务调度(job schedule)是一个永远不会停止的运行程序.

如何去计算算法所需要的执行时间呢?在程序或算法中，每一语句(statement)的执行时间为此语句执行的次数与每一次执行所需的时间的乘积．由于每一语句所需的时间必须实际考虑到机器和编译器的功能，因此通常只考虑执行的次数．例如下列有三个程序，请计算 x←x+1 的执行次数:

```
.              for i←1 to n do          for i←1 to n do
.                  x←x+1                    for j←1 to n do
x←x+1              .                            x←x+1
.              end                          end
                                          end
(a)                (b)                      (c)
```

(a)的执行次数为 1 次，(b)的执行次数为 n 次，(c)的执行次数为 n^2 次.

在分析算法时，一般称语句的执行次数为数量阶(order of magnitude)，所以上述的(a)，(b)，(c)中，x←x+1 的数量阶分别为 1，n，n^2．而整个算法的数量阶为算法中每一语句的执行次数之和.

让我们来看看斐波纳契数(Fibonacci number)的算法:

```
1 :       procedure FIBONACCI
2 :        read (n)
3～4 :    if n < 0 then [print ('error'); stop]
5～6 :    if n = 0 then [print ('0'); stop]
7～8 :    if n = 1 then [print ('1'); stop]
9 :       fnm 2 ← 0; fnm 1 ← 1
10 :      for i ← 2 to n do
11 :          fn ← fnm1+fnm2
12 :          fnm2 ← fnm1
13 :          fnm1 ← fn
14 :       end {for}
15 :       print (fn)
16 :      end {procedure}
```

斐波纳契数为 0，1，1，2，3，5，8，13，21，34，55，...某一项为前二项之和，斐波纳契数的第零项为 0，第 1 项为 1，第 2 项为第零项加第 1 项等于 1，以此类推.

当 n < 0 或 n = 0 或 n = 1 时，FIBONACCI 的算法中每一语句所执行的次数如下:

列 号	执行次数		
	n<0	n=0	n=1
1	1	1	1
2	1	1	1
3	1	1	1
4	1	0	0
5	0	1	1
6	0	1	0
7	0	0	1
8	0	0	1
9～15	0	0	0

当 n > 1 时 FIBONACCI 的算法如下:

列 号	执行次数	列 号	执行次数
1	1	9	2
2	1	10	n
3	1	11	n−1
4	0	12	n−1

5	1	13	n−1
6	0	14	n−1
7	1	15	1
8	0	16	1

由于第 9 列有二个语句故执行次数一共为 2，第 10 列执行次数为 n，是因为当执行到 n+1 时才会跳出此循环，由 2，3，4，…n+1，共有 n 次．第 11，12，13，14 都执行 n−1 次，因此 FIBONACCI 算法总共执行次数是 5n+5．

再看一个例子．下面是两个矩阵相乘的算法:

```
1 : procedure MATRIX
2 :   for i = 1 to n
3 :       for j = 1 to n
4 :         sum ← 0
5 :         for k = 1 to n
6 :           sum ← sum+A[i,  k] * B[k,  j]
7 :         end
8 :         C[i,  j] ← sum
9 :       end
10 :   end
11 : end {procedure}
```

列　号	执行次数	列　号	执行次数
1	1	6	n^3
2	n+1	7	n^3
3	$n(n+1)=n^2+1$	8	n^2
4	n^2	9	n^2
5	$n^2(n+1)=n^3+n^2$	10	n
		11	1

所以 MATRIX 算法的执行次数共 $3n^3+4n^2+2n+2$．

1.2　Big−oh

计算完算法的执行次数后，通常利用 Big−oh 来计算此算法执行的时间．

当且仅当有两个常数 c 与 n_0，当所有 $n>n_0$ 都满足$|f(n)|<c|g(n)|$，则 $f(n)=O(g(n))$ 为此算法所需的时间．如斐波纳契数算法的执行时间为 O(n)，因为 $5n+n<10n$，故矩阵相乘算法执行时间为 $O(n^3)$．O(g(n))就是程序执行时所需最坏的时间

一般常见的 Big−oh 有下列几种情形:

1. O(1)称为常数(constant)时间
2. O(n)称为线性(linear)时间
3. O($\log_2 n$)称为次线性(sub-linear) 时间
4. O(n^2)称为平方(quadratic)时间
5. O(n^3)称为立方(cubic)时间
6. O(2^n)称为指数(exponential)时间
7. O($n\log_2 n$)

当 n>16 时，其执行时间长短的顺序如下：

$O(1)<O(\log_2 n)<O(n)<(n\log_2 n)<O(n^2)<O(n^{[3]})<O(2^n)$.

可由图 1-1 得知，当 n 愈大时，更能显示其间的差异，如图 1-2 所示。

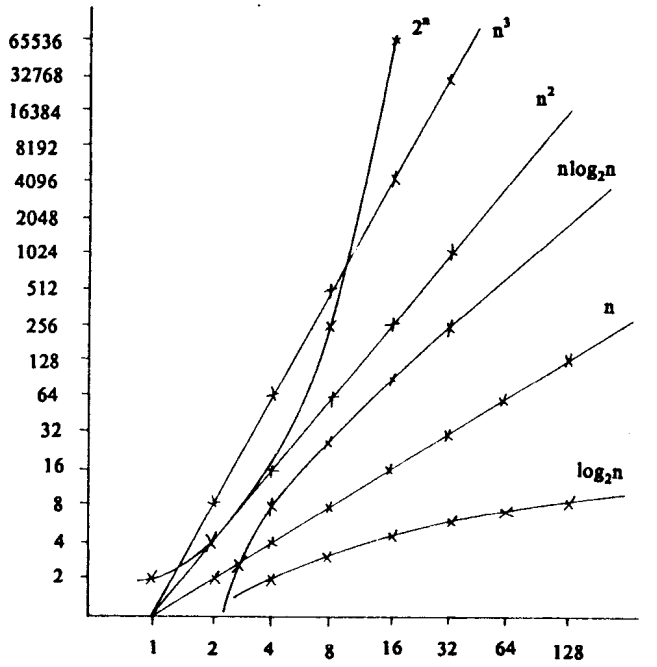


图 1-1

$\log_2 n$	n	$n\log_2 n$	n^2	n^3	2^n
0	1	0	1	1	2
1	2	2	4	8	4
2	4	8	16	64	16
3	8	24	64	512	256
4	16	64	256	4096	65536
5	32	160	1024	32768	2,147,483,6648

图 1-2

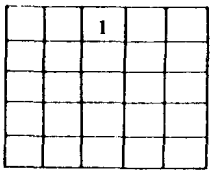
1.3 魔术方阵

最后以一个游戏来做为本章的结束．读者可以从此游戏的算法计算出 Big-oh 有一 $n \times n$ 的方阵，其中 n 为奇数，请将 1 到 n^2 的整数填入 $n \times n$ 的魔术方阵中，使其各列、各行及对角线之和都相等．

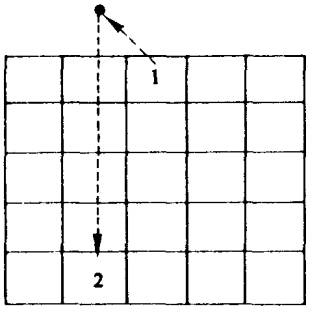
做法很简单，首先将 1 填入最上列的中间格，然后往左上方走：(a)以 1 的级数增加其值，并将此值填入空格；(b)假使方格已填满，则在原地的下一方格填上数字，并继续做；(c)若超出方阵，则往下到最底层或往右到最右方，看两者那一个有方格，则将数字填上此方格；(d)若两者都无方格，则在原地的下一方格填上数字．

例如有一 5×5 的方阵，其形成魔术方阵的步骤如下，并以上述(a)，(b)，(c)，d)规则来说明：

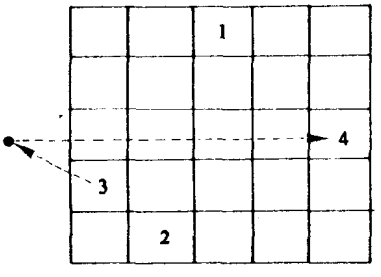
(1) 将1填入此方阵最上列的中间方格，如下所示：



(2) 接(1)往左上方走，由于超出方阵，依据规则(c)发现往下的最底层有空格，因此将 2 填上，如下所示：



(3) 接(2)步往左上方，依据规则(a)将 3 填上，然后再往左上方，此时，超出方阵依据规则(c)将 4 填在最右方的方格，如下所示：



(4) 接(3)步往左上方，依据规则(a)将 5 填上，再往左上方时，此时方格已有数字，依据规则(b)往 5 的下方填，如下所示：

		1		
			5	
			6	4
3				
	2			

(5) 以此类推，依据上述的四个规则继续填，填到 15 的结果如下：

15	8	1		
	14	7	5	
		13	6	4
3			12	10
9	2			11

(6) 接(5)步此时往左上方，发现往下的最底层和往右的最右方都无空格，依据规则(d)在原地的下方，将此数字填上，如下所示：

15	8	1		
16	14	7	5	
		13	6	4
3			12	10
9	2			11

(7) 继续往下填，并依据规则(a)，(b)，(c)，(d)，最后的结果如下：

15	8	1	24	17
16	14	7	5	23
22	20	13	6	4
3	21	19	12	10
9	2	25	18	11

此时读者可以算算各行、各列及对角线之和是否都相等，答案是肯定的，其和都为 65。魔术方阵的算法如下：

```
1 : procedure MAGIC (square,  n)
2 :  / / square (0: n-1,  0: n-1) / /
3 :  if n is even then [print ('input error'); stop]
```

```
4 :  square ← 0
5 :  square (0,  (n−1) / 2) ← 1
6 :  key ← 2; i ← 0; j ← (n−1) / 2
7 :  while key ≤ n²do
8 :     (k,  l) ← ((i−1) mod n,  (j−1) mod n)
9 :     if square (k,  l) ≠0 then
10 :       i ← (i+1) mod n
11 :    else (i,  j) ← (k,  l)
12 :    square (i,  j) ← key
13 :    key ← key+1
14 : end {while}
15 : print (n,  square)
16 :end {procedure}
```

说明如下:

列号	代 表 意 义
2	此方阵是一n×n的方阵
3	假使n是偶数，则打印出错误的信息
4	将此方阵的每一方格都清除为0
5	在最上列的中间方格填上1
6	i，j表示当前的位置
8～9	表示往左上方，并测试在最右方或最下层的方格是空的
9～10	往左上方移，发现此方格没有数字，则在原来的位置下方填上数字
12	往左上方移，此方格没有数字，则将此数字填上
13	数字以1的级数增加

以上述的 5×5 方阵为例来说明魔术方阵的算法:

(1) 首先将 1 放在 square(0, (n−1) / 2)的方格上，若 n=5，则此方格为第 0 列、第 2 行.

(2) 将当前的方格所代表的第 n 列和第 n 行存放在 i 与 j 中(第 6 行)，此时 i=0，j=2，并将 2 赋值给 key.

(3) 当 key≤5^2 时，将(i−1) mod n 即(0−1)mod 5=4；(j−1) mod n 即(2−1) mod 5=1，然后将 4 赋值给 k，1 赋值给 l(第 8 行)，也即方格在第 4 列、第 1 行. 由于(4, 1)=0，故将 4 赋给 j(第 11 行)，然后将 key 的值放在(i, j)=(4, 1)方格上(第 12 行)，key=key+1(第 13 行).

(4) 如果 key=6，此时的(i, j)为(1, 3). 因为 square(0, 2)已有数字，即 square(k, l)≠0，则将(1+1) mod 5=2(第 10 行)，将此数字赋值给 i，即此方格为(2, 3)，此表示在原来的方格往下移一格.

(5) 利用同样的方法即可完成魔术方阵。

在此节开始时就谈到一些魔术方阵的规则，共有四个，其中第(2)规则相当算法的第 8，9，11，12，13 行，而规则(3)，(4)相当算法的第 8，9，10，12，13 行。读者可以自己以一个 7×7 的方阵来填写，相信会使您更明了。

1.4 思考题

1.1 何谓算法(algorithm)，它与程序(procedure)有何区别?

1.2 何谓 Bin-oh，试说明其含义。

1.3 请计算下列 n 个片断程序中 x←x+1 的执行次数。

(1)
```
1: for i ← 1 to n
2:     for j ← 1 to i
3:         for k ← 1 to j
4:             x ← x+1
5:         end
6:     end
7: end
```

(2)
```
1: i ← 1
2: while i < n do
3:     x ← x+1
4:     i ← i+1
5: end
```

(3)
```
1: for i ← 1 to n do
2:     for j ← 1 to n do
3:         x ← x+1
4:     end
5: end
```

(4)
```
1: for i ← 1 to n do
2:     for j ← 1 to n do
3:         for k ← 1 to n do
4:             x ← x+1
5:         end
6:     end
7: end
```

(5)
```
1: for i ← 1 to n do
2:     j ← i
3:     for k ← j+1 to n do
4:         x ← x+1
5:     end
6: end
```

(6)
```
1: for i = 1 to n do
2:     j = i
3:     while j > = 2 do
4:             j = j div 5
5:             x ← x+1
6:     end
7: end
```

(7)
```
1: k = 100, 000
2: while k < > 5 do
3:     k = k div 10
4:     x ← x+1
5: end
```

(8)
```
1: for i = 1 to n
2:     k ← i+1
3:     repeat
4:         x ← x+1
5:     until k < n
6: end
```

1.4 假设数组 A 有 10 个元素分别为 2，4，6，8，10，12，14，16，18，20，请问若查找 1，3，13 和 21 四个数，在下列程序中 repeat–until 内的语句(statement)一共执行多少次(列出单个查找 1，3，13 和 21 所需的次数，然后再求和)?

```
1:    i ← 1; j ← n
2:    repeat k ← [(i+j) / 2]
3:      if A (k) < x then
4:            i ← k+1
5:      else
6:            j ← k−1
7:      until i > j
```

1.5 二叉查找法的算法如下:

```
1:    procedure BINSRCH (A, n, x, j)
2:        lower ← 1; upper ← n
3:        while lower < upper do
4:            mid ← [(lower + upper) / 2]
5:            case
6:                x > A (mid): lower ← mid+1
7:                x < A (mid): upper ← mid−1
8:                else: j ← mid; return
9:            end {case}
10:       end {while}
11:   end {procedure}
```

请列出每一行所需执行的次数，并计算此算法所需花费的时间(即 Big-oh)。

1.5 程序设计

程序一：奇数魔术方阵

```
1 / * ODD MAGIC MATRIX     —> use C88 V2.51, MicroSoft C V4.0 * /
2
3 #include "stdio.h"
4 #define MaxN 32+1 / * define the maximum number = 32 * /
5 int A [MaxN] [MaxN], N;
6 int x, y, i, j, count;
7 main()
8   {
9   begin:
10  printf ("\nPlease input an odd positive.");
11  scanf ("%d", &N);
12  if ((N%2 = = 0)     (N < = 0))
13    {
14    printf ("\n%d is not an odd positive!", N);
15    goto begin;
16    }
17  printf ("Please input the starting number. ");
18  scanf ("%d", &count);
19  i = 1;
20  j = (N+1) / 2;
21  for (x = 1; x < = N; x++)
22    {
23    for (y = 1; y < = N; y++)
```