数据结构教程

蔡子经 施伯乐 编著

复旦大学出版社

数据结构教程

.992
42

数据结构教程

蔡子经　施伯乐　**编著**

复旦大学出版社

(沪)新登字 202 号

数据结构教程

蔡子经　施伯乐　编著

复旦大学出版社出版

(上海国权路 579 号)

新华书店上海发行所发行　复旦大学印刷厂印刷

开本 787×1092　1/16　印张 16.25　插页 0　字数 395,000

1994 年 12 月第 1 版　1994 年 12 月第 1 次印刷

印数 1—6,000

ISBN7-309-01408-1/T·116

定价: 14.00 元

内 容 提 要

本书共分八章，它们分别是线性表、串、排序、数组、树、树的查找及其应用、图和外部排序。复旦大学计算机科学系从1980年开始以此为教材，并不断更新、充实为一本比较完善的、系统的教科书。本书中大量算法都有实用价值，并且用目前最流行的C语言描述所有算法。

本书是计算机各专业及软件有关专业的基础教材，也可作为大专院校教师的教学参考书，以及数据处理工作者和软件应用技术人员的参考资料。

序　言

随着计算机科学和技术的发展，计算机的功能和运算速度不断地提高，其应用于信息处理的范围日趋扩大，需要表示、存贮和处理的信息也越来越多。因此，与之相应地，计算机的加工处理对象也从简单的数值发展到一般的符号，进而发展到更复杂的数据结构。

通常的数据结构可用一个二元组(D,R)表示，可写成 DS=(D,R)，其中 D 是数据集合，R 是 D 中数据元素之间所存在的关系的集合。对于数据集合 D，如果 D 中的数据元素之间存在着不同的关系集合 R_1 和 R_2，那么 $DS_1=(D,R_1)$ 和 $DS_2=(D,R_2)$ 是两个不同的数据结构。数据集合中的数据元素(也称结点)可以由若干个数据项(也称字段)所组成。数据结构就是根据各种不同的数据集合和数据元素之间的关系，研究如何表示、存贮、操作(查找、插入、删除、修改)这些数据的技术。因此，数据结构是计算机科学和技术中的一门基础课程。它是设计编译程序、操作系统、数据库系统及其他程序系统的重要基础。

数据结构的表示和操作都涉及到算法，如何描述数据的结构和讨论有关的算法，又涉及到程序设计语言。我们选择了 C 语言为工具，因为 C 语言不仅具有丰富的数据类型，也是一种较好地体现结构程序设计原则的语言，而且 C 语言在计算机界日益流行，且为计算机工作者所乐意使用。

全书共分八章。前三章介绍具有线性关系的数据结构，它是数据结构的基础。第一章介绍顺序存贮和链接存贮的线性表、栈和队列的表示和操作；第二章介绍一种特殊的线性表——串，重点介绍串的各种模式匹配算法；第三章介绍排序，讨论实现排序的各种算法，算法的好坏标准按照算法的时间复杂性来衡量。

第四～七章介绍具有非线性关系的数据结构。第四章介绍数组的顺序存贮和用于存取数组元素的地址计算公式，以及稀疏矩阵的表示及其操作；第五章介绍树，树是一种非常重要的非线性结构，它具有较大的实用价值。这一章介绍树的结构和有关的操作，对于二叉树作了更详细的讨论；第六章介绍树的查找及树的应用，讨论了查找树、解答树、B-树的表示及有关的算法；第七章介绍图，图是比树更复杂的数据结构，这一章讨论图的表示和图的遍历，以及图在最短路径、拓扑排序和关键路径等方面的应用。

最后的第八章介绍外部排序，讨论的内容涉及二级存贮设备，对于大型文件系统及数据库管理系统的研究有一定的价值。

本书是编者为讲授数据结构课程而编写的教材，是复旦大学计算机科学系的重点课程教材之一；同时，本书也可作为各类大专院校有关专业和计算机培训班的教材，为软件工作者所必备。

编　者

一九九四年四月

试读结束：需要全本请在线购买：　www.ertongbook.com

目　录

第一章 线 性 表

线性表是最常用、最简单的一种线性结构。因为它的应用范围十分广泛，所以有必要在这里作较详细的介绍。

1.1 线性表及其基本运算

1.1.1 线 性 表

线性表是具有相同类型的 n 个数据元素 $k_0,k_1,\cdots,k_{n-1}$ 的有限序列，记为 $(k_0,k_1,\cdots,k_{n-1})$。元素个数 n 称为线性表的长度，称长度为零的线性表为空的线性表（简称为空表）。当 $n\geqslant 1$ 时，k_0 是最前面的一个元素，k_{n-1} 是最后一个元素，线性表中数据元素的相对位置是确定的。因为线性表的数据元素构成一个序列，在序列中，k_i 排在 k_{i+1} 前面，我们称 k_i 是 k_{i+1} 的前趋元素；k_{i+1} 排在 k_i 后面，我们称 k_{i+1} 是 k_i 的后继元素。k_0 没有前趋元素，k_{n-1} 没有后继元素；当 $n\geqslant 2$ 时，k_0 有后继元素 k_1，k_{n-1} 有前趋元素 k_{n-2}；当 $n\geqslant 3$ 时，$k_i(0<i<n-1)$ 既有前趋元素 k_{i-1}，也有后继元素 k_{i+1}。

对于给定的线性表 $(k_0,k_1,\cdots,k_{n-1})$，它的数据元素集合为 $\{k_0,k_1,\cdots,k_{n-1}\}$，而数据元素之间的关系由数据元素出现在线性表中的位置所确定。我们可用数据元素 k_i,k_{i+1} 构成的偶对表示数据元素 k_i 和 k_{i+1} 所存在的这种关系。因此，可用数据元素的偶对的集合表示线性表中数据元素之间的关系，这种关系记为 $\{(k_i,k_{i+1})\mid 0\leqslant i<n-1\}$。对于相同数据元素集合的两个线性表，如果数据元素出现在表中的次序不相同，那么这两个线性表是不相同的。

线性表中的数据元素也称为结点，或称为记录，它可以是一个整数、一个实数、一个字符或一个字符串；也可以由若干个数据项组成，其中每个数据项可以是一般数据类型，也可以是构造类型。数据项也称为字段，或称为域。

比如，表 1.1.1 的线性表用于记录最近一周每天的平均气温。每个结点有两个字段：一个是星期，用于指明是星期几，它的数据类型是由三个字符组成的字符串；另一个是温度，用于指明平均气温，它的数据类型是实数。

表 1.1.1 一周内每天的平均气温记录表

星期	Mon	Tue	Wed	Thu	Fri	Sat	Sun
温度	15.5	16.0	16.5	15.7	15.0	16.1	16.4

再比如，表 1.1.2 也是一个线性表，它是一个企业的职工工资表。该表由职工号、姓名、性别、年龄和工资等五个字段所组成，其中职工号、年龄的数据类型是整数；姓名和性别是字符串；工资是实数。

表 1.1.2 职工工资表

职工号	姓 名	性 别	年 龄	工 资
001	Wang	male	35	160.50
002	Cai	male	32	150.00
003	Zhang	female	28	130.00
⋮	⋮	⋮	⋮	⋮

线性表中的结点可由若干个字段组成，我们称能唯一标识结点的字段为关键字，或简称为键。如表 1.1.1 的线性表的关键字是星期，而表 1.1.2 的线性表的关键字是职工号。在本书中，为了讨论方便，往往只考虑结点的关键字，而忽略其他字段。这样的假设也不失一般性，只要知道存放某结点的存贮单元，就能取到该结点的其他字段上的值。

1.1.2 线性表的基本运算

上面介绍了线性表的基本概念，下面介绍线性表的一些基本运算。

线性表结构可以动态地增长或收缩，在线性表的任何位置上可以访问、插入或删除结点。我们把线性表常用的运算分成几类，每类包含若干种运算。

1. 查找

(1) 查找线性表的第 i($0 \leqslant i \leqslant n-1$)个结点。

(2) 在线性表中查找值为 x 的结点。

2. 插入

(1) 把新结点插在线性表的第 i($0 \leqslant i \leqslant n$)个位置上。

(2) 把新结点插在值为 x 的结点的前面(或后面)。

3. 删除

(1) 在线性表中删除第 i(($0 \leqslant i \leqslant n-1$)个结点。

(2) 在线性表中删除值为 x 的结点。

4. 其他运算

(1) 统计线性表中结点的个数。

(2) 打印线性表中所有结点。

(3) 复制一份线性表。

(4) 把一个线性表拆成几个线性表。

(5) 把几个线性表合并成一个线性表。

(6) 根据结点的某个字段值按升序(或降序)重新排列线性表。

1.2 顺序存贮的线性表

在计算机内，可以利用不同的存贮方式表示线性表，其中最常用、最简单的方式是顺序存贮。所谓线性表的顺序存贮，就是用一组连续的存贮单元依次存贮线性表中的结点。

因为线性表中所有结点的数据类型是相同的，所以每个结点在存贮器中占用大小相同

试读结束：需要全本请在线购买： www.ertongbook.com

的空间。如果每个结点占用计算机中按机器字编址或按字节编址的 s 个地址的存贮单元，并假设存放结点 $k_i(0\leqslant i\leqslant n-1)$ 的开始地址为 αk_i，那么结点 k_i 的地址 αk_i 可用整数 i，以及地址计算公式

$$\alpha k_i=\alpha k_0+i\cdot s$$

计算出来。

对于顺序存贮的线性表，因为可以利用地址计算公式直接计算出 k_i 的开始地址 αk_i，所以存取第 $i(0\leqslant i\leqslant n-1)$ 个结点特别方便。

如果用 C 语言的数组表示线性表 $(k_0,k_1,\cdots,k_{n-1})$，我们可使用如下的说明：

```
#define MAXSIZE 100
int list[MAXSIZE];
int n;
```

其中 MAXSIZE 是数组 list 中元素个数的最大值，n 是线性表中当前的结点个数。这里假设线性表的结点值是整数，所以数组元素也是整数，当然可根据需要取其他类型。线性表的结点 $k_0,k_1,\cdots,k_{n-1}$ 依次存放在数组元素 list[0],list[1],…,list[n−1]中。这样，存放线性表各个结点的地址与数组 list 各个下标变量的地址有如下的对应关系：

αk_0	αk_1	……	αk_{i-1}	αk_i	……	αk_{n-1}
↕	↕		↕	↕		↕
&list[0]	&list[1]	……	&list[i−1]	&list[i]	……	&list[n−1]

有了上述的对应关系，可得到下面计算 k_i 的地址的公式，这里我们假设 s=sizeof(int)：

因　　$\alpha k_i=$&list[i]

而　　&list[i]=&list[0]+i·s

故　　$\&k_i=$&list[0]+i·s

1.2.1 线性表的插入

这里所讲的插入是在具有 n 个结点的线性表中，把新结点插在线性表的第 $i(0\leqslant i\leqslant n)$ 个位置上，使原来长度为 n 的线性表变成长度为(n+1)的线性表。在把新结点放进线性表前，必须把原来位置号(序号)为(n−1)至位置号为 i 的结点依次往后移一个位置，然后把新结点放在第 i 个位置上，此时共移动(n−i)个结点。对于 i=n，只要把新结点插在第 n 个位置上，此时无需移动结点。

下面用一个 C 函数 sq_insert( )实现上述的插入算法。此函数在具有 n 个结点的线性表 list 中，把值为 x 的结点插在第 $i(0\leqslant i\leqslant n)$ 个位置上。若插入位置 i 不在可以插入的位置上(即 i<0 或 i>n)，则返回 1；若 n=MAXSIZE(即线性表已满)，此时 list 数组没有存贮单元存放新结点，则返回 2；若插入成功，则返回 0。在函数的参数中，有一个指针变量 p_n，在调用时，把存放线性表的当前结点个数的变量 n 的地址赋给指针变量 p_n，以此来实现插入后线性表长度 n 增加 1。

```
int sq _ insert(list, p _ n, i, x)
  int list[ ], x;
  int *p _ n, i;
  {  int j;
     if (i<0 || i>*p _ n)  return(1);
     if (*p _ n==MAXSIZE)  return(2);
     for (j=*p _ n; j>i; j--)
        list[j]=list[j-1];
     list[i]=x;
     (*p _ n)++;
     return(0);
  }
```

对于存放在数组 list 中的、具有 n 个结点的线性表，为了把值为 x 的结点插在线性表第 i(0≤i≤n)个位置上，可用如下的调用语句：

```
k=sq _ insert(list, &n, i, x);
```

在具有 n 个结点的线性表中，插入一个新结点时，其执行时间主要花费在移动结点的循环上。假设把新结点插在第 i(0≤i≤n)个位置上的概率为 p_i，各个 p_i 应满足约束条件 $\sum_{i=0}^{n} p_i = 1$。因为把一个新结点插在第 i 个位置上需移动(n−i)个结点，所以移动结点的平均次数为

$$\sum_{i=0}^{n} p_i(n-i)$$

如果线性表中每个可插入位置的插入概率相同，即有

$$p_0=p_1=\cdots=p_{n-1}=p_n=\frac{1}{n+1}$$

那么，上式可改写为

$$\frac{1}{n+1}\sum_{i=0}^{n}(n-i)=\frac{1}{n+1}\sum_{i=1}^{n}i=\frac{1}{(n+1)}\cdot\frac{n(n+1)}{2}=\frac{n}{2}$$

上式表明，在顺序存贮的线性表中，插入一个新结点，平均需要移动一半结点。当线性表的结点很多，且每个结点的数据量相当大时，花费在移动结点上的执行时间就会很长。

1.2.2 线性表的删除

这里所讲的删除是在具有 n 个结点的线性表中，删除第 i(0≤i≤n−1)个位置上的结点，使原来长度为 n 的线性表变成长度为(n−1)的线性表。删除时，要把位置号为(i+1)至位置号为(n−1)的结点都依次向前移动一个位置，此时共需移动(n−i−1)个结点。

下面用一个 C 函数 sq _ delete( )实现上述的删除算法。此函数在具有 n 个结点的线性表 list 中，删除第 i(0≤i≤n−1)个位置上的结点。若删除的结点不在可删除的位置上(即 i<0 或 i≥n)，则返回 1；若删除成功，则返回 0。在函数的参数中，有一个指针变量 p _ n，在

调用时，把存放线性表当前结点个数的变量 n 的地址赋给指针变量 p _ n，以此来实现删除后线性表的长度 n 减少 1。

```
int sq _ delete(list, p _ n, i)
  int list[ ];
  int *p _ n, i;
  { int j;
    if ( i<0 || i>= *p _ n) return(1);
    for (j=i+1; j< *p _ n; j++)
        list[j-1]=list[j];
    (*p _ n)--;
    return(0);
  }
```

调用时，可使用如下的语句：

```
k=sq _ delete(list, &n, i);
```

在具有 n 个结点的线性表中，删除一个结点所需的执行时间主要花费在移动结点的循环上。假设删除第 i(0≤i≤n−1)个位置上的结点的概率为 p_i，每个 p_i 应满足约束条件 $\sum_{i=0}^{n-1} p_i = 1$。因为删除第 i(0≤i≤n−1)个位置上的结点需移动(n−i−1)个结点，所以移动结点的平均次数为

$$\sum_{i=0}^{n-1} p_i(n-i-1)$$

如果假设删除线性表任何一个结点的概率都相同，即有

$$p_0=p_1=\cdots=p_{n-1}=\frac{1}{n}$$

那么上面的和式可改写为

$$\frac{1}{n}\sum_{i=0}^{n-1}(n-i-1)=\frac{1}{n}\sum_{j=1}^{n-1}j=\frac{1}{n}\cdot\frac{n(n-1)}{2}=\frac{n-1}{2}\simeq\frac{n}{2}$$

上式表明，在顺序存贮的线性表中，删除一个结点，平均大约需要移动一半结点。当线性表的结点很多，且每个结点的数据量相当大时，花费在移动结点上的执行时间就会很长。

1.2.3 用顺序存贮的线性表表示多项式

一般代数多项式可写成

$$A(x)=a_mx^{e_m}+a_{m-1}x^{e_{m-1}}+\cdots+a_1x^{e_1}+a_0x^{e_0}$$

其中每个 $a_i(0\leqslant i\leqslant m)$ 是 A(x)的非零系数；次数 $e_i(0\leqslant i\leqslant m)$ 是递减的，即 $e_m>e_{m-1}>\cdots>e_1>e_0\geqslant 0$。

我们可用包含 coef 和 exp 两个字段的结点表示多项式的非零项，其中 coef 给出系数，exp 给出次数。这样，可用数组 poly[MAXN]表示多项式。因此，可使用如下的说明：

```
#define MAXN 100
typedef struct term {   float coef;
                        int exp;
                    }   TERM;
TERM poly[MAXN];
```

其中 MAXN 是数组 poly 可存放结点的最大个数。有时，我们可把 MAXN 取得适当大，使得多个多项式可共享数组 poly。

例如，对于下面两个多项式：

$$A(x)=8x^{60}+6x^{50}+4x^{25}+2x^{10}+1$$

$$B(x)=7x^{60}-6x^{50}+3x^{20}$$

它们在数组 poly 中的存贮情况如图 1.2.1 所示。

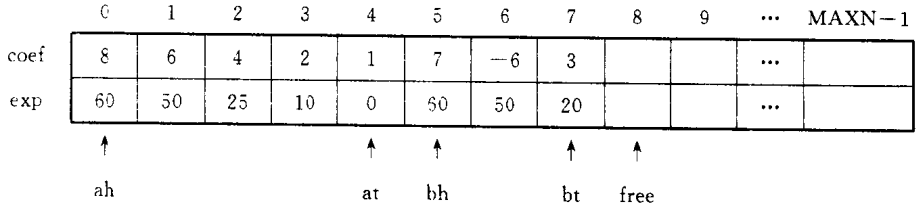

图 1.2.1　多项式的存贮

我们使用了指针 ah 和 bh，它们分别给出 A(x)和 B(x)的第一项在数组中的存放位置，而指针 at 和 bt 分别给出 A(x)和 B(x)的最后一项在数组中的存放位置。在图 1.2.1 中，我们取 ah=0，at=4，bh=5，bt=7。另外，指针 free 给出数组中空闲存区的开始位置，取 free=8。

这里顺便指出一点，ah，at，bh，bt 及 free 等，它们其实不是指针，只是一种游标。这是因为存放在 ah，at，bh，bt 及 free 内的不是数组元素在存贮器中的地址，而是数组元素的下标值，但由于使用上的习惯，我们仍然称它们为指针，希望读者加以区别。

上面介绍如何用数组存放多项式的方法，下面我们讨论如何用 C 函数实现多项式相加的算法。在下面的程序中，函数polyadd( )实现C(x)=A(x)+B(x)的运算；而函数 append( )把系数 c 和次数 e 构成的项存放到由指针 free 所指出的位置上，成为结果多项式的一个项，同时 free 加 1，为下次添加结点提供存放位置。下面给出 C 语言的说明及实现运算的函数。

```
#define MAXN 100
typedef struct { int coef;
                 int exp;
               } TERM;
TERM poly[MAXN];
int ah, at, bh, bt, ch, ct, free;
```

```
int append(c, e)
  int c;
  { if (free>=MAXN) return (1);
    poly[free] . coef=c;
    poly[free++] . exp=e;
    return(0);
  }
int  poly _add(ah, at, bh, bt, p _ch, p _ct)
  int ah, at, bh, bt;
  int *p _ch, *p _ct;
  { int p, q, pe, qe;
    int c;
    char ch;
    p=ah;
    q=bh;
    *p _ch=free;
    while (p<=at && q<=bt)
      { pe=poly[p] .exp;
        qe=poly[q] .exp;
        if  (pe==qe) ch='=';
        else if (pe<qe) ch='<';
             else ch='>';
        switch (ch)
          { case '=' :
              c=poly[p]. coef+poly[q]. coef;
              if  (c)
                  if  (append(c,pe))
                      return(1);
              p++;
              q++;
              break;
            case '<' :
              if  (append(poly[q] .coef, qe))
                  return(1);
              q++
              break;
            case '>' :
              if  (append(poly[p] .coef, pe))
                  return(1);
```

```
            p++;
            break;
        }
    }
  while (p<=at)
    { if (append(poly[p].coef, poly[p].exp))
        return(1);
      p++;
    }
  while (q<=bt)
    { if (append(poly [q].coef, poly[q].exp))
        return(1)
      q++;
    }
  *p_ct=free-1;
  return(0);
}
```

可用如下的调用语句执行A(x)和B(x)两个多项式的相加：

```
if ( polyadd (ah, at, bh, bt, &ch, &ct ))
    printf("ERROR\n");
```

如果打印出ERROR，那么说明数组poly不够用；如果没有打印ERROR，但调用后出现ct<ch，那么说明相加后得到一个零多项式。

现在分析polyadd( )的执行时间。首先应指出，执行函数append( )的时间是O(1)。在polyadd( )中，执行三个循环之外的几个语句的时间也是O(1)。所以，执行时间主要花费在三个循环上。设A(x)和B(x)非零项的个数分别为m和n，对于第一个循环，当A(x)和B(x)的各项的次数都不相同时，执行循环的次数最多，共执行(m+n)次，故其执行时间最多为O(m+n)；第二和第三个循环最多分别执行m次和n次，故其执行时间最多分别为O(m)和O(n)。这样，执行polyadd( )的时间为O(1)+O(m+n)+O(m)+O(n)=O(m+n)。

1.3 顺序存贮的栈和队列

栈和队列都是特殊的线性表。这两种结构比一般线性表更简单，但非常有用，所以有必要对它们进行仔细讨论。

1.3.1 栈及其基本运算

栈是只允许在一端进行插入和删除的线性表。称允许插入和删除的一端为栈顶；称不允许插入和删除的一端为栈底。若给定栈$S=(s_0, s_1, \cdots, s_{n-1})$，则$s_0$是栈底结点，$s_{n-1}$是栈顶结

试读结束：需要全本请在线购买： www.ertongbook.com