计算机专业教学辅导丛书

数据结构（C 语言篇）

——习题与解析

（修订版）

李 春 葆　编著

清 华 大 学 出 版 社
http://www.tup.tsinghua.edu.cn

第 7 章　广　义　表

广义表是 n(n≥0)个数据元素 a_1，a_2，…，a_n 的有限序列。其中 $a_i(1 \le i \le n)$其中或是单个数据元素（或称为原子），或仍然是一个广义表。通常记作 $GL=(a_1, a_2, \ldots, a_n)$。其中 GL 是广义表的名字，n 是其长度。如果 a_i 是一个广义表，则称它为 GL 的子表。a_1 是表头，其余元素组成的表称为表尾。广义表通常简称为表。

7.1　广义表的表示及其运算

7.1.1　广义表的表示

广义表通常用圆括号括起来，用逗号分隔广义表中的元素，本书中我们约定用小写字母表示原子，用大写字母表示广义表，例如：

```
A=(␣)
B=(a，b)
C=(c)
D=(B，c)=((a，b)，c)
E=(B，C)=((a，b)，(c))
```

其中 A 是一个空广义表，B 和 C 是两个仅包含原子的广义表，而 D 和 E 是两个包含子表的广义表。

广义表的长度指该广义表中所包含的元素（包括原子和子表）的个数。

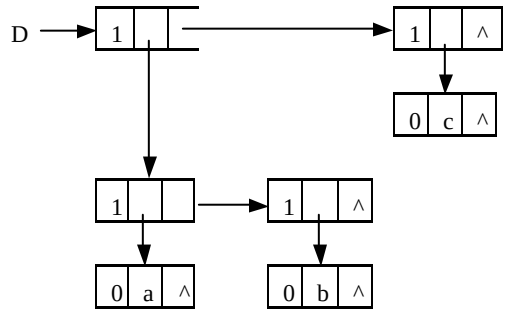
广义表的深度指该广义表中所包含括号的层数。

广义表中的结点具有不同的结构，即原子结点和子表元素结点，为了将两者统一，用了一个标志 tag，当其为 0 时表示是原子结点，其 data 域存储结点值，link 域指向下一个结点；当其 tag 为 1 时表示是子表结点，其 sublist 为指向子表的指针。因此，广义表采用如下结构存储：

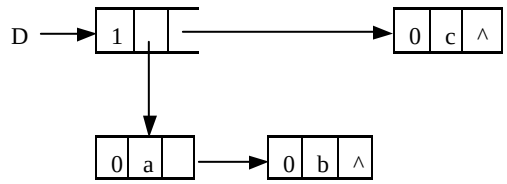
```
typedef struct linknode
      {
         int tag; /*在建立广义表时只能是 0 或 1*/
         struct linknode *link;
         union {
                 char data;
                 struct linknode *sublist;
               } val;
```

```
        } gnode;
```

广义表有两种类型的存储结构（参见文献[3]），即第一种存储结构和第二种存储结构。前者将广义表分为表头和表尾存储，后者在同层存储所有兄弟（又分为带头结点和不带头结点两种）。如图 7.1 所示，图（a）是第一种存储结构，图（b）是不带头结点的第二种存储结构。



（a）第一种存储结构



（b）第二种存储结构

图 7.1　广义表 D 的存储结构

7.1.2　广义表的基本运算

本节采用广义表第一种存储结构讨论其基本运算算法。

1．建立广义表算法

根据字符串 s 建立相应广义表。其递归定义如下：

基本项：
- 置空广义表　　当 s 为空表时
- 建原子结点的子表　　当 s 为单字符串时

归纳项：假设 subs 为脱去 s 中最外层括号对的子串，记为“ s_1，s_2，...，s ”，其中 s (i=1，2，...，n)为非空字符串。对每个 s 建立一个表结点，并令其 val.sublist 域的指针为 s 建立的子表的头指针，除最后建立的尾指针为 NULL 外，其余表结点的尾指针均这样指向。

函数 disastr(char *s，char *hstr)的功能是从字符串 s 中取出第 1 个','之前的子串赋给 hstr，并使 s 成为删除子串 hstr 和','之后的剩余串。若串 s 中没有字符','，则操作后的 hstr 即为操作前的 s，而操作后的 s 为空串。

建立广义表算法的函数如下：

此为试读，需要完整PDF请访问：www.ertongbook.com

```
void disastr(char s[], char hstr[])
{
   int i=0, j=0, k=0, r=0;
   char rstr[100];
   while (s[i] && (s[i]!=', ' || k))
   {
      if (s[i]=='(') k++;
      else if (s[i]==')') k--;
      if (s[i]!=', ' || s[i]==', ' && k)
      {
          hstr[j]=s[i];
          i++;
          j++;
      }
   }
   hstr[j]='\0';
   if (s[i]==', ') i++;
   while (s[i])
   {
      rstr[r]=s[i];
      r++;
      i++;
   }
   rstr[r]='\0';
   strcpy(s, rstr);
}

gnode *gcreat(char s[])
{
   gnode *p, *q, *r, *gh;
   char subs[100], hstr[100], rstr[100];
   int len;
   len=strlen(s);
   if (!strcmp(s, "()") || len == 0 ) gh=NULL;
   else
   if (len==1) /*原子的情况*/
   {
      gh=(gnode *)malloc(sizeof(gnode));     /*建立一个新结点*/
      gh->tag=0;                             /*构造原子结点*/
      gh->val.data=*s;
      gh->link=NULL;
   }
```

```
    else                                          /*子表的情况*/
    {
       gh=(gnode *)malloc(sizeof(gnode));         /*建立一个新结点*/
       gh->tag=1;
       p=gh;
       s++;
       strncpy(subs，s，len-2);                   /* 除掉前面的一个'(' */
       subs[len-2]='\0';
       do
       {
          disastr(subs，hstr);                    /*将 subs 分为表头和表尾*/
          r=gcreat(hstr);                         /* 表尾处理 */
          p->val.sublist=r;
          q=p;
          len=strlen(subs);
          if (len>0)
          {
             p=(gnode *)malloc(sizeof(gnode));
             p->tag=1;
             q->link=p;
          }
       } while (len>0);
       q->link=NULL;
    }
    return(gh);
}
```

2．打印广义表算法

对给定存储结构的广义表，打印出其表示格式。

采用的算法思想是：先打印出一个'('号，若遇到 tag=0 的结点 p，是一个原子结点，如果有后续结点，则打印该结点值之后加一个','，','号，若没有后续结点，则只打印该结点值。如果 tag=1，为子表的情况，则递归调用本函数打印其该子表，子表打印完后，最后打印一个')'号。

实现上述算法的函数如下：

```
void prtlist(gnode *h)
{
   gnode *p，*q;
   printf("(");
   if (h)
   do
   {
```


计算机专业教学教辅丛书

```
        p=h->val.sublist;
        q=h->link;
        while (q && p && !p->tag) /*为原子结点且无后续结点的情况，*/
        {
           printf("%c，", p->val.data);
           p=q->val.sublist;
           q=q->link;
        }
        if (p && !p->tag) /*为原子结点且无后续结点的情况*/
        {
           printf("%c", p->val.data);
           break;
        }
        else /*为子表的情况*/
        {
           if (!p) printf("()");
           else prtlist(p);  /*为子表的情况*/
           if (q) printf("，");
           h=q;
        }
     } while (h);
     printf(")");
  }
```

3．查找算法

在给定的广义表中查找数据域为 x **的结点。**

算法思想是：若遇到 tag=0 **的原子结点** p，**如果是要找的结点**(p->val.data=x)，**则查找成功；若遇到** tag=1 **的结点** p，**则递归调用本函数在该子表中查找；若未找到数据域为** x **的结点且还有所续元素，则递归调用本函数查找后续每个元素，直至遇到** link **域为** NULL **的元素。**

设 f(p，x)**为查找函数，当成功时为** true，**否则为** false，**则有如下递归模型：**

$$f(p,x)=\begin{cases} true & \text{若 } p\text{->tag}=0 \text{ 且 } p\text{->val.data}=x \\ f(p->link,x) & \text{若 } p\text{->tag}=0 \text{ 且 } p\text{->val.data}\neq x \\ f(p->val.sublist,x)\ or\ f(p->link,x) & \text{若 } p\text{->tag}=1 \end{cases}$$

由此，实现上述算法的函数如下：

```
int locate(gnode *p，char x)
{
   int find=0;
   if (p!=NULL)
```

```
    {
       if (!p->tag && p->val.data==x)     /* 当前结点为原子结点且其 data 为 x */
         return(1);
       else if (p->tag)
         find=locate(p->val.sublist, x); /* 查找子表 */
       if (find) return(1);
       else return(locate(p->link, x));   /* 查找后续结点 */
    }
    else
      return(0);
}
```

7.2 基 本 题

7.2.1 单项选择题

1．广义表((a)，a)的表头是__①__，表尾是__②__。

A. a　　B. b

C. (a)　　D. ((a))

答：① C　② C

2．广义表((a))的表头是__①__，表尾是__②__。

A. a　　B. (a)

C. (␣)　　D. ((a))

答：① B　② C

3．广义表((a，b)，c，d)的表头是__①__，表尾是__②__。

A. a　　B. b

C. (a，b)　　D. (c，d)

答：① C　② D

4．广义表(a，b，c，d)的表头是__①__，表尾是__②__。

A. a　　B.b

C. (a，b)　　D. (b，c，d)

答：① A　② D

5．广义表((a，b，c，d))的表头是__①__，表尾是__②__。

A. a　　B. (␣)

C. (a，b，c，d)　　D. ((a，b，c，d))

答：① C　② B

6．一个广义表的表头总是一个广义表，这个断言是_____。

A. 正确的　　B. 错误的

答：B

计算机专业教学教辅丛书

此为试读，需要完整PDF请访问：www.ertongbook.com

7．一个广义表的表尾总是一个广义表，这个断言是_____。

A. 正确的　　B. 错误的

答：A

7.2.2 填空题（将正确的答案填在相应的空中）

1．广义表(((a)))的表头是__①__，表尾是__②__。

答：① ((a))　② (␣)

2．广义表((a)，((b)，c)，(((d))))的表头是__①__，表尾是__②__。

答：① (a)　② (((b)，c)，(((d))))

3．广义表((a)，((b)，c)，(((d))))的长度是__①__，深度是__②__。

答：① 3　② 4

4．广义表(a，(a，b)，d，e，((i，j)，k))的长度是__①__，深度是__②__。

答：① 5　② 3

5．设 HAED[p]为求广义表 p 的表头函数，TAIL[p]为求广义表 p 的表尾函数，其中[]是函数的符号，给出下列广义表的运算结果：

HEAD[(a，b，c)]的结果是__①__。

TAIL[(a，b，c)]的结果是__②__。

HEAD[((a)，(b))]的结果是__③__。

TAIL[((a)，(b))]的结果是__④__。

HEAD[TAIL[(a，b，c)]的结果是__⑤__。

TAIL[HEAD((a，b)，(c，d))]的结果是__⑥__。

HEAD[HEAD[(a，b)，(c，d))]]的结果是__⑦__。

TAIL[TAIL[(a，(c，d))]]的结果是__⑧__。

答：① a　② (b，c)　③ (a)　④ ((b))　⑤ b　⑥ (b)　⑦ a　⑧ (␣)

7.3 习 题 解 析

*1．假设用下述两种结点的链表作广义表的存储结构：

表结点：

tag=1	dlink	rlink

元素结点：

tag=0	data

（1）画出下列广义表的存储结构：((((a)，b))，(((␣)，(d))，(e，f)));

（2）给出如图 7.2 所示的存储结构对应的广义表。

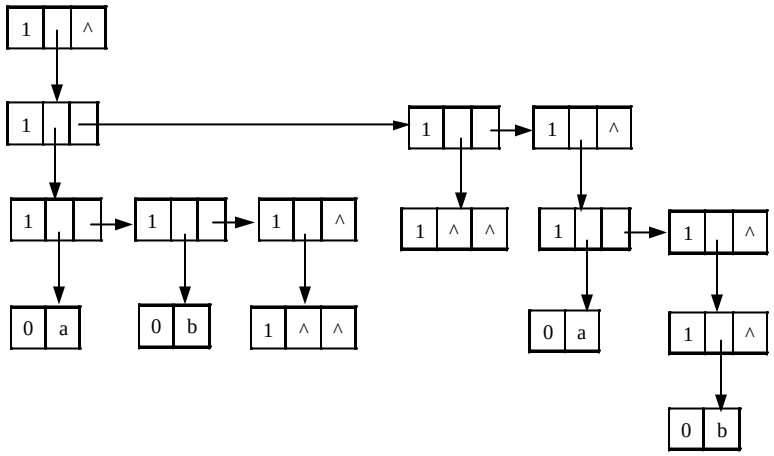


图 7.2　广义表对应的存储结构

解：依题意，某存储结构为广义表第二种存储结构。

（1）本小题的广义表对应的不带头结点的存储结构如图 7.3 所示。

（2）图 7.2 的存储结构为带头结点的第二种存储结构，对应的广义表为：(((a，b，(␣))，(␣)，(a，(b)))，(␣))。

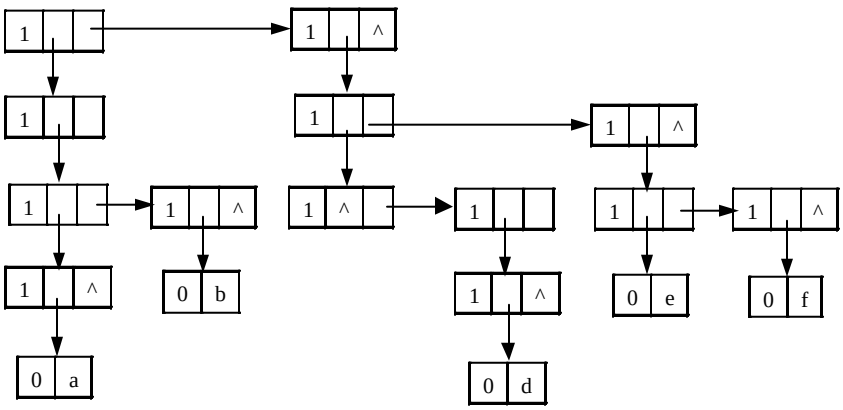


图 7.3　广义表表示

2．根据表头和表尾的定义，设计分别实现求表头和表尾操作的 head()**和** tail()**函数，并编写一个求广义表表头和表尾的程序。**

解：一个广义表的表头指的是该广义表的第一个元素。由此，实现上述算法的函数如下：

```
gnode *head(gnode *p)
{
   return(p->val.sublist);
}
```

一个广义表的表尾指的是除去该广义表的第一个元素后的所有剩余的部分。由此，实

计算机专业教学教辅丛书

现上述算法的函数如下：

```
gnode *tail(gnode *p)
{
   return(p->link);
}
```

由此得到一个求广义表表头和表尾的程序如下：

```
main()
{
   char *s;
   gnode *glist，*h，*t;
   strcpy(s，"(a，(b，c，d))");
   glist=gcreat(s);
   printf("\n广义表:");
   prtlist(glist);
   h=head(glist);
   printf("\n表头:"); /*为原子的情况，要进行特殊处理*/
   if (h->tag==0) printf("%c"，h->val.data);
   else prtlist(h);
   t=tail(glist);
   printf("\n表尾:");
   prtlist(t);
}
```

本程序的执行结果如下:

```
广义表:(a，(b，c，d))
表头:a
表尾:((b，c，d))
```

3．编写一个函数计算一个广义表的长度。例如一个广义表为(a，(b，c)，((e)))，其长度为 3。

解：依题意，本题可用直接求解和递归求解两种方法。直接求解的算法思想：扫描通过广义表的第一层的每个结点，直至遇到 link 域为 NULL 的元素，每扫描一个结点，长度累加器增 1。采用递归求解的递归模型如下：

$$f(p)=\begin{cases} 0 & \text{若 } p=\text{NULL 或空表} \\ f(\text{tail}(p))+1 & \text{若 } p\neq\text{NULL} \end{cases}$$

例如，计算广义表(a，(b，c)，((e)))的长度的递归函数如下：

```
f[(a，(b，c)，((e)))]→求值为 3
          ↓
  f[((b，c)，((e)))]→求值为 2
```

```
        ↓
   f(((e)))→求值为1
        ↓
     f[(␣)]
        ↓
     返回0
```

因此，实现本题功能的直接求解函数如下：

```
int length(gnode *p)
{
   int n=0;
   if (p!=NULL && p->tag==1)
   {
      while (p!=NULL)
      {
         p=p->link;
         n++;
      }
      return(n);
   }
   else return(0);
}
```

实现本题功能的递归求解函数如下:

```
int length1(gnode *p)
{
   if (p==NULL) return(0);
   else return(length1(tail(p))+1); /*tail()为上题的函数*/
}
```

4．编写一个函数计算一个广义表的原子结点个数。例如一个广义表为(a，(b，c)，((e)))，其原子结点个数为 4。

解：依题意，得到计算一个广义表的原子结点个数的递归模型如下：

$$f(p)=\begin{cases}0 & \text{若 } p=NULL\\ 1+f(p->link) & \text{若 } p->tag=0\\ f(p->val.sublist)+f(p->link) & \text{若 } p->tag=1\end{cases}$$

因此，实现本题功能的函数如下：

```
int counter(gnode *p)
{
   int m，n;
   if (p==NULL) return(0);
```


此为试读，需要完整PDF请访问：www.ertongbook.com

```
    else
    {
       if (p->tag==0) n=1;
       else n=counter(p->val.sublist);
       if (p->link!=NULL)
          m=counter(p->link);
        else m=0;
        return(m+n);
    }
}
```

5．编写一个函数计算一个广义表的所有原子结点数据域(数据域为整数型)之和。例如一个广义表为((3，4)，5，((6，3)))，其所有原子结点数据域之和为 21。

解：依题意，得到计算一个广义表的所有原子结点数据域之和的递归模型如下：

$$f(p)=\begin{cases}0 & \text{若 } p=NULL\\ p\text{->val.data}+f(p\text{->link}) & \text{若 } p\text{->tag}=0\\ f(p\text{->val.sublist})+f(p\text{->link}) & \text{若 } p\text{->tag}=1\end{cases}$$

因此，实现本题功能的函数如下：

```
int sum(gnode *p)
{
   int m，n;
   if (p==NULL) return(0);
   else
   {
      if (p->tag==0) n=p->val.data-'0';    /* 将数字字符转换成对应数字 */
      else n=sum(p->val.sublist);
      if (p->link!=NULL)
         m=sum(p->link);
      else m=0;
      return(m+n);
   }
}
```

*6．编写一个函数接受任一无共享子表的非递归表 L，求出此表的深度，设表以链表形式存放，每个结点有三个域：

tag	data/sublist	link

tag= { 0 表示该结点为原子
       1 表示该结点为表

例如 L=((A，B)，C，((D，E)，F))**的深度为** 3。

解：依题意，本题采用的算法思想：扫描通过广义表的第一层的每个结点（使用 p=p->link **语句），对每个结点递归调用计算出其子表的深度，取最大的子表深度，然后加** 1 **即为该广义表的最大深度，即递归模型如下：**

$$\begin{cases} maxdh(p)=0 & p \text{ 为原子即 } p\text{->tag}=0 \\ maxdh(p)=1 & p \text{ 为空表即 } p\text{->tag}=1 \text{ 且 } p\text{->val.sublist}=NULL \\ maxdh(p)=max(maxdh(p_1)，\ldots，maxdh(p_n))+1 & \text{否则，} p=(p_1，p_2，\ldots，p_n) \end{cases}$$

因此，实现本题功能的函数如下：

```
int depth(gnode *p)
{
   int h，maxdh;
   gnode *q;
   if (p->tag==0) return(0);  /*若表头结点 tag 为 0，则表示该表为原子*/
   else if (p->tag==1 && p->val.sublist==NULL) return 1;  /*空表的情况*/
      else  /*否则，要进行递归求解*/
      {
         maxdh=0;/*赋初值*/
         while (p!=NULL)
         {   /*循环扫描广义表的第一层的每个结点，对每个结点求其子表深度*/
            q=p->val.sublist;
            h=depth(q);
            if (h>maxdh) maxdh=h; /*取最大的子表深度*/
                p=p->link;
         };
         return(maxdh+1); /*最大子表深度加 1 即为该广义表的深度*/
      }
}
```

7．编写一个函数将两个广义表合并成一个广义表。合并是指元素的合并，例如两个广义表((a，b)，(c))**与**(a，(e，f))**合并后的结果是**((a，b)，(c)，a，(e，f))。

解：依题意，本题采用的算法思想：先找到第一个广义表的最后一个结点，将其链接到第二个广义表的首元素上即可。因此，实现本题功能的函数如下：

```
gnode *append(gnode *p，gnode *q)
{
   gnode *r;
   if (p!=NULL && p->tag==1)
   {
      r=p;
      while (r!=NULL) r=r->link;  /*r 指向最后一个结点*/
      if (q!=NULL && q->tag==1) r->link=q;
   }
```

```
    return(p);
}
```

8．编写一个函数复制一个广义表，包括该广义表的所有原子结点和非原子结点。

解：将广义表 p 复制到广义表 q 的递归模型如下：

copy(p，q)→q=NULL　　**当 p=NULL 时**

copy(p，q)→q->tag=q->tag，q->val.data=p->val.data

当 p 只有一个原子结点时，直接复制

copy(p，q)→copy(p->val.sublist，q-val.sublist)，copy(p->link，q->link)

否则，先复制第一个元素的子表（若存在），然后复制其后续元素

因此，实现本题功能的函数如下：

```
void copy(gnode *p，gnode *q)
{
   if (p==NULL) q=NULL;
   else
   {
      q=(gnode *)malloc(sizeof(gnode));
      q->tag=p->tag;
      if (p->tag==0) q->val.data=p->val.data; /*原子结点直接复制*/
      else
      {
         copy(p->val.sublist，q->val.sublist); /*子表结点要递归调用复制子表*/
         copy(p->link，q->link);               /*复制该结点的后续表*/
      }
   }
}
```

9．编写一个与二叉树前序遍历的递归算法相似的遍历广义表的算法，按照广义表的逻辑结构顺序，打印广义表所有元素结点上的数据域。

解：依题意，得到前序遍历广义表的递归模型如下：

preorder(p)→write(p->val.data)，preorder(p->link)

若 p 为原子结点，显示该结点数据域值，然后递归调用本函数遍历所有后续元素

preorder(p)→preorder(p->val.sublist)，preorder(p->link)

若 p 是子表结点，递归调用本函数遍历该子表，然后递归调用本函数遍历所有后续元素

因此，实现本题功能的函数如下：

```
preorder(gnode *p)
{
```

```
    if (p!=NULL)
    {
       if (p->tag==0) printf("%c ", p->val.data);
       else preorder(p->val.sublist);
       if (p->link!=NULL)
          preorder(p->link);
    }
}
```

例如，广义表(a，(b，c，d)，e，((f)))**的输出结果是：**

```
a b c d e f
```

10．**编写一个函数判定两个广义表是否相等。相等的含义是指两个广义表具有相同的存储结构，对应的原子结点的数据域值也相同。**

解：依题意，得到判定两个广义表是否相等的递归模型如下：

$$
\begin{cases}
\text{same}(p,q)=\text{same}(p\text{->link},q\text{->link}) & \text{若 p->tag=0 且 q-tag=0 且 p->val.data=q->val.data} \\
\text{same}(p,q)=\text{same}(p\text{->val.sublist},q\text{->val.sublist}) \text{ and } \text{same}(p\text{->link},q\text{->link}) & \text{若 p->tag=1 且 q-tag=1} \\
\text{same}(p,q)=\text{false} & \text{若 p->tag=0 且 q->tag=0 且 p->val.data}\neq\text{q->val.data 或 p-tag=0 且 q->tag=1 或 p->tag=1 且 q->tag=0} \\
\text{same}(p,q)=\text{false} & \text{若 p=NULL 且 q}\neq\text{NULL 或 p}\neq\text{NULL 且 q=NULL}
\end{cases}
$$

因此，实现本题功能的函数如下：

```
int same(gnode *p，gnode *q)
{
   int flag=1;
   if (p!=NULL && q!=NULL)
   {
     if (p->tag==0 && p->tag!=0)
       if (p->val.data!=q->val.data) flag=0;
     else if (p->tag==1 && q->tag==1)
           flag=same(p->val.sublist，q->val.sublist);
        else flag=0;
       if (flag) flag=same(p->link，q->link);
   }
   else
   {
      if (p==NULL && q!=NULL) flag=0;
      if (p!=NULL && q==NULL) flag=1;
   }
   return(flag);
```


计算机专业教学教辅丛书

```
}
```

*11．编写一个函数删除广义表中所有值为 x 的元素。例如删除广义表((a，b)，a，(d，a))中所有 a 的结果是广义表((b)，(d))。

解：依题意，得到删除广义表中所有值为 x 的元素的递归模型如下：

$$
f(p,x)=\begin{cases}
\text{NULL} & \text{若 p->tag=0 且 p->val.data=x 且 p->link=NULL}\\
f(p,x)=f(tail(p),x) & \text{若 p->tag=0 且 p->val.data=x 且 p->link!=NULL}\\
f(p,x)=head(p) & \text{若 p->tag=0 且 p->val.data}\neq x\text{ 且 p->link=NULL}\\
f(p,x)=append(head(p),f(tail(p),x)) & \text{若 p->tag=0 且 p->val.data}\neq x\text{ 且 p->link!=NULL}\\
f(p,x)=f(head(p),x) & \text{若 p->tag=1 且 p->link=NULL}\\
f(p,x)=append(f(head(p),x),f(tail(p),x)) & \text{若 p->tag=1 且 p->link}\neq\text{NULL}
\end{cases}
$$

这里的 append(a，b)功能是将广义表 a 与 b 作为元素的广义表连接起来。

例如：求解 f[((a，b)，a，(d，a))]的逆归调用和求值过程如图 7.4 所示。

```
gnode *delall(gnode *p,char x)
{
   gnode *s,*t,*q;
   if (p==NULL)
       q=NULL;
   else
   {
      if (p->tag==0)   /*为原子结点的情况*/
      {
         if (p->val.data!=x)
         {
            q=(gnode *)malloc(sizeof(gnode));  /*复制该原子结点*/
            q->tag=0;
            q->val.data=p->val.data;
            q->link=NULL;
         }
         else q=NULL;                          /*返回空表*/
      }
      else
      {
         delall(p->val.slist,x,s);             /*删除表头中的 x 得到 s*/
         if (s==NULL)                          /*新表的表头为空的情况*/
         {
            p=p->link;
            if (p!=NULL)
            {
```

```
                delall(p->val.slist,x,t);          /*从表尾的表头产生 t*/
                q=(gnode *)malloc(sizeof(gnode)); /*创建整个新表的表头结点 q*/
                q->tag=1;
                q->link=NULL;
                q->val.slist=t;            /*将 t 作为新表的表头*/
                if (p->link!=NULL)         /*若还有后续元素,删除 x 后作为新表表尾*/
                {
                   delall(p->link,x,t);
                   q->link=t;
                }
             }
          }
          else q=NULL;  /*返回空表*/
       }
       else
       {
          delall(p->link,x,t);                  /*删除表尾中的 x 得到 t*/
          q=(gnode *)malloc(sizeof(gnode));     /*创建整个新表的表头结点 q*/
          q->tag=1;
          q->val.slist=s;
          q->link=t;
       }
    }
    }
    return q;
}
```

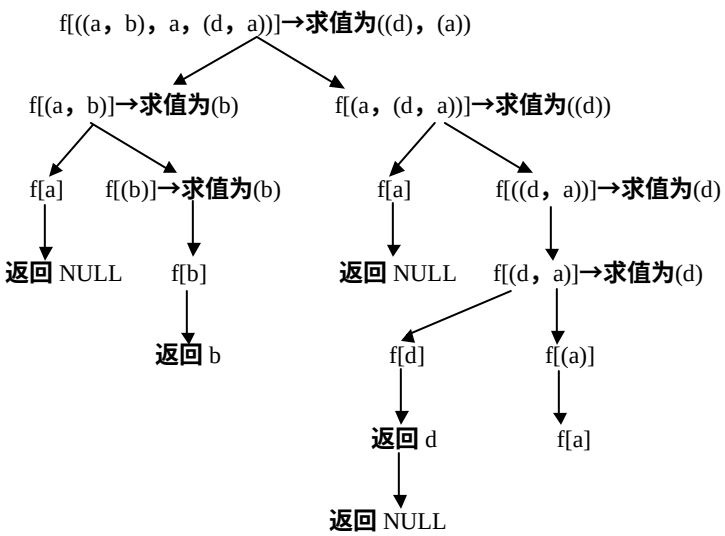


图 7.4　递归调用和求值过程

此为试读,需要完整PDF请访问: www.ertongbook.com