5.7 广义表操作的递归函数

递归函数

一个**含直接或间接调用本函数语句**的函数被称之为递归函数。

它必须满足以下两个条件：

1）在每一次调用自己时，必须是（在某种意义上）**更接近于解；（递归表达式）**

2）必须有一个**终止**处理或计算的**准则**。（递归出口）

例如： 汉诺塔的递归函数

```
void hanoi (int n, char x, char y, char z)
 {
   if (n==1)
     move(x, 1, z)；    //递归终止
   else {
     hanoi(n-1, x, z, y)；  //递归调用
     move(x, n, z)；
     hanoi(n-1, y, x, z)； //递归调用
   }
 }
```

如何设计递归函数？

主要采用**分治法**（Divide and Conquer），又称分割求解法。

分治法的设计思想为:

对于一个输入规模为 n 的函数或问题，用某种方法把输入分割成 **k(1<k≤n)**个子集，从而产生 L 个子问题，**分别求解**这 L 个问题，得出 L 个问题的子解，再用某种方法把它们**组合成原来问题的解。**若子问题还相当大，则可以反复使用分治法，直至最后所分得的子问题足够小，以至可以直接求解为止。

在利用分治法求解时，所得子问题的类型常常和原问题相同，因而很自然地导致递归求解。

例如: 汉诺塔问题: Hanoi(n, x, y, z)

将 n 个盘分成两个子集(1至n-1 和第 n 个），从而产生下列三个子问题：

1）将1至n-1号盘从 x 轴移动至 y 轴；

———— 可递归求解 Hanoi(n-1, x, z, y)

2）将 n号盘从 x 轴移动至 z 轴；

3）将1至n-1号盘从y轴移动至z轴；

———— 可递归求解 Hanoi(n-1, y, x, z)

广义表从结构上可以分解成：

广义表 = 表头 + 表尾

或者

广义表 = 子表1 + 子表2 + … + 子表n

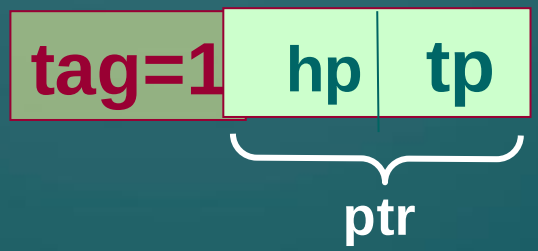
因此常利用分治法求解之。

算法设计中的关键问题是，**如何将 L 个子问题的解组合成原问题的解。**

广义表的头尾链表存储表示：

```
typedef enum {ATOM, LIST} ElemTag;
        // ATOM=0:原子, LIST=1:子表
typedef struct GLNode {
   ElemTag  tag;   // 标志域
   union{
     AtomType  atom;      // 原子结点的数据域
     struct {struct GLNode *hp, *tp;} ptr;
   };
} *GList
```

表结点



5.7.1 求广义表的深度

广义表的深度定义为广义表中括弧的层数。

将广义表分解成 n 个子表，分别（递归）求得每个子表的深度。

广义表的深度=Max {子表的深度} +1

可以直接求解的两种简单情况为：

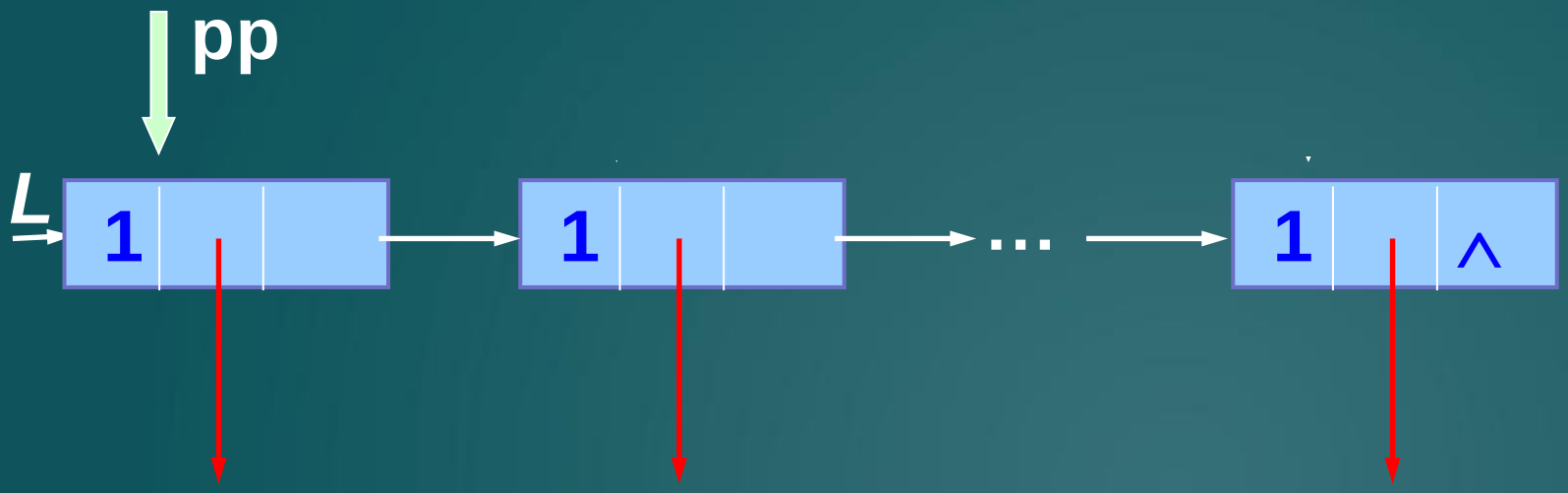
空表的深度 = 1

原子的深度 = 0

```
int GlistDepth(Glist L) {
    // 返回指针L所指的广义表的深度
    if (!L) return 1；
    if (L->tag == ATOM) return 0；
   for (max=0, pp=L; pp; pp=pp->ptr.tp){
      dep = GlistDepth( pp->ptr.hp)；
      if (dep > max) max = dep；
   }
   return max + 1；
} // GlistDepth
```

算法　5.5

例如:




```
for (max=0, pp=L; pp; pp=pp->ptr.tp){
        dep = GlistDepth(pp->ptr.hp);
        if (dep > max) max = dep;
    }
```

具体例子，讲义P114 图5.13

5.7.2 复制广义表

前面提到，任何一个非空广义表均可分解成表头和表尾；反之，一对确定的表头和表尾可惟一确定一个广义表。

将广义表分解成表头和表尾两部分，分别（递归）复制求得新的表头和表尾，

新的广义表由新的表头和表尾构成。

可以直接求解的两种简单情况为：

空表复制求得的新表自然也是空表；

原子结点可以直接复制求得。

复制求广义表的算法描述如下:

若 ls= NIL 则 newls = NIL

否则

构造结点 newls,

由 表头ls->ptr.hp 复制得 newhp

由 表尾 ls->ptr.tp 复制得 newtp

并使 newls->ptr.hp = newhp,

newls->ptr.tp = newtp

```
Status CopyGList(Glist &T, Glist L) {
    if (!L) T = NULL;  // 复制空表
    else {
      if ( !(T = (Glist)malloc(sizeof(GLNode))) )
          exit(OVERFLOW);  // 建表结点
      T->tag = L->tag ;
      if (L->tag == ATOM)
          T->atom = L->atom ;   // 复制单原子结点
      else {   分别复制表头和表尾    }
    } // else
    return OK;
} // CopyGList
```

算法 5.6

复制表头和表尾为：

CopyGList(T->ptr.hp, L->ptr.hp)；

// 复制求得表头L->ptr.hp的一个副本T->ptr.hp

CopyGList(T->ptr.tp, L->ptr.tp)；

// 复制求得表尾L->ptr.tp 的一个副本T->ptr.tp

语句 CopyGList(T->ptr.hp, L->ptr.hp)；

等价于

CopyGList(newhp, L->ptr.tp)；

T->ptr.hp = newhp；

5.7.3 建立广义表的存储结构

从上述两种广义表操作的递归算法中可以发现：在对广义表进行操作的递归定义时，可有两种分析方法：一种是把广义表分解成表头和表尾；另一种是把广义表看成是含有n个并列子表（假设原子也作为子表）的表。

对应广义表的**不同**定义方法相应地有**不同**的创建存储结构的算法。

假设以字符串 S = ′(α_1, α_2, …, α_n)′ 的形式定义广义表 **L**，建立相应的存储结构。

由于 S 中的每个子串 α_i 定义 **L** 的一个**子表**，从而产生 **n** 个子问题，即分别由这 **n**个子串（**递归**）建立 **n** 个子表，再**组合**成一个广义表。

可以直接求解的两种简单情况为：

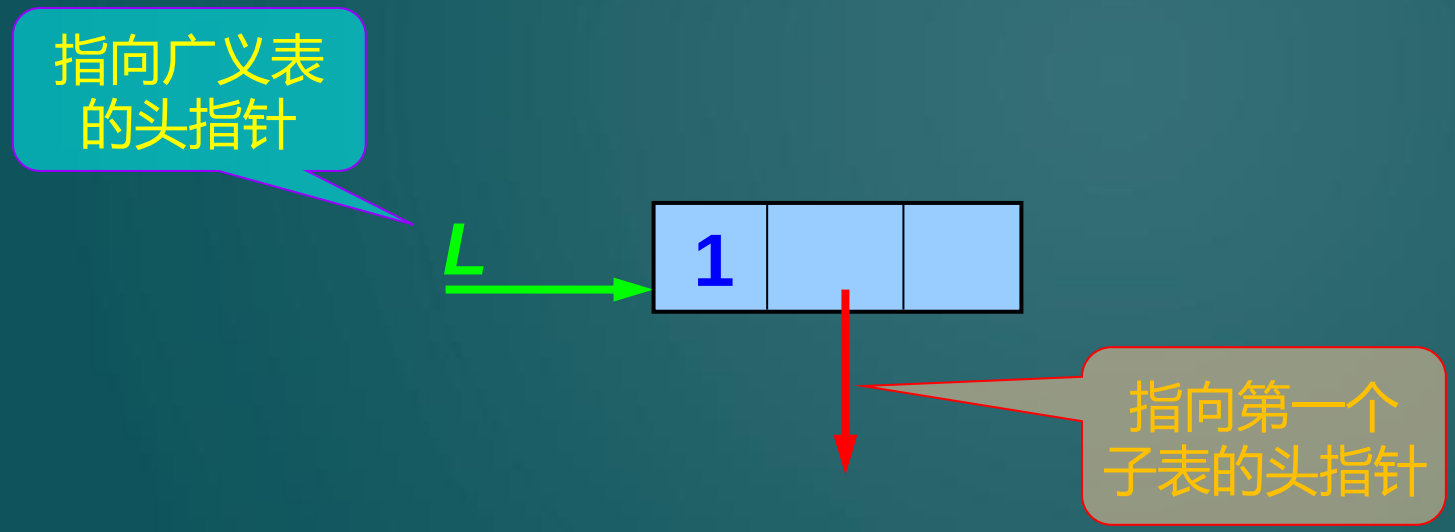
由串′( )′建立的广义表是空表；

由单字符建立的子表只是一个原子结点。

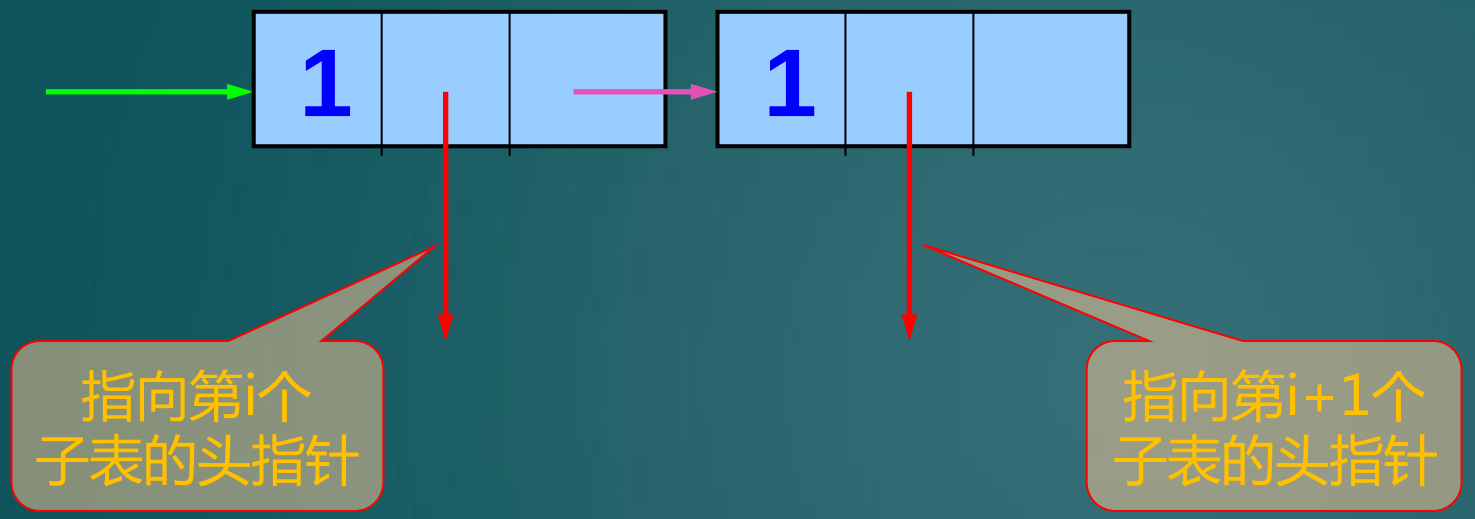
如何由子表组合成一个广义表？

首先分析广义表和子表在存储结构中的关系。

先看第一个子表和广义表的关系:



再看相邻两个子表之间的关系:



可见，两者之间通过表结点相链接。

若 S = '( )' 则 L = NIL；

否则，构造第一个表结点*L，

并从串S中分解出第一个子串α_1，对应创建第一个子广义表

L->ptr.hp；

若剩余串非空，则构造第二个表结点**L->ptr.tp**，并从串S中分解出第二个子串 α_2，对应创建第二个子广义表……；

依次类推，直至剩余串为空串止。

```
void CreateGList(Glist &L, SString S) {
    if  (strcompare(S,emp))  L = NULL ;   // 创建空表
    else {
      if(!(L=(Glist) malloc(sizeof(GLNode))))
          exit(overflow) ;   //建表结点
      if(Strlength(s)==1) {L->tag=ATOM ; L->atom=s ; }
      else {                              //创建单原子广义表
      L->tag=List ;    p=L ;
      sub=SubString(S,2,StrLength(S)-2) ;
                                  //脱去串S的外层括弧
        由sub中所含 n 个子串建立 n 个子表 ;
    } // else
    return OK
}
```

算法 5.7

```
do {              //重复建n个子表
    sever(sub, hsub)；// 从sub中分离出表头串hsub
    CreateGlist(p->ptr.hp,hsub)； q=p；
    if (!StrEmpty(sub) {    //表尾不空
     if(!( p=(GLNode*)malloc(sizeof(GLNode))))
          exit(overflow)；
     p->tag=LIST； q->ptr.tp=p；
    }//if
} while (!StrEmpty(sub))；
q->ptr.tp = NULL；    // 表尾为空表
}//else
```

```
Status server(Sstring  &str, SString &hstr) {
//将非空串str分割成两部分：hstr为第一个 ‘，’ 之前的子串，
str为之后的子串
  n=Strlength(str); i=0; k=0;//k记尚未配对的左括号个数
      Do {          //搜索最外层的第一个逗号
        ++i ;
          SubString ( ch , str , i , 1);
          if (ch== ‘( ‘)  ++k;
          else if (ch== ‘) ‘)  --k;
        } while(i<n&&(ch!= ‘,’ ||k!=0));
        if (i<n)
       { SubString(hstr,str,1,i-1); SubString(str,str,i+1,n-i) }
        else {strcopy(hstr,str);clearstring(str) }
  } server
```

算法 5.8

本章内容小结

1、了解数组的两种存储表示方法，并掌握数组在以行为主的存储结构中的地址计算方法。

2、掌握对特殊矩阵进行压缩存储时的下标变换公式。

3、了解稀疏矩阵的两类压缩存储方法的特点和适用范围，领会以三元组表示稀疏矩阵时进行矩阵运算采用的处理方法。

4、 掌握广义表的结构特点及其存储表示方法，掌握对非空广义表进行分解的两种分析方法：即可将一个非空广义表分解为表头和表尾两部分或者分解为n个子表。

5、学习利用分治法的思想设计递归算法的方法。