7.4 二叉树基本运算及其实现

7.4.1 二叉树的基本运算概述

归纳起来，二叉树有以下基本运算：

❶ 创建二叉树CreateBTNode(*b,*str)：根据二叉树括号表示法字符串str生成对应的二叉链存储结构b。

❷ 销毁二叉链存储结构DestroyBT(*b)：销毁二叉链b并释放空间。

❸ 查找节点FindNode(*b,*x*)：在二叉树b中寻找data域值为*x*的节点，并返回指向该节点的指针。

❹ 找孩子节点LchildNode(p)和Rchild-Node(p)：分别求二叉树中节点*p的左孩子节点和右孩子节点。

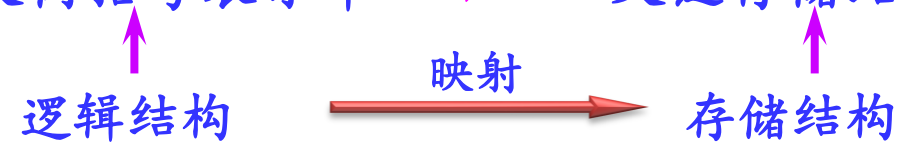
❺ 求高度BTNodeDepth(*b)：求二叉树b的高度。若二叉树为空，则其高度为0；否则，其高度等于左子树与右子树中的最大高度加l。

❻ 输出二叉树DispBTNode(*b)：以括号表示法输出一棵二叉树。

7.4.2 二叉树的基本运算算法实现

（1）创建二叉树CreateBTNode(*b,*str)

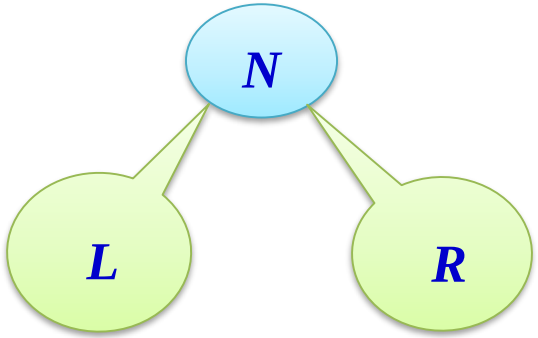
由正确的二叉树括号表示串 ⇨ 二叉链存储结构



正确的二叉树括号表示串中只有4类字符：

- 单个字符：节点的值
- (：表示一棵左子树的开始
-)：表示一棵子树的结束
- ,：表示一棵右子树的开始

算法设计：



- 先构造根节点N，再构造左子树L，最后构造右子树R
- 构造右子树R时，找不到N了，所以需要保存N
- 而节点是按最近原则匹配的，所以使用一个栈保存N

用ch扫描采用括号表示法表示二叉树的字符串：

① 若ch='('：则将前面刚创建的节点作为双亲节点进栈，并置*k*=1，表示开始处理左孩子节点；

② 若ch=')'：表示栈顶节点的左、右孩子节点处理完毕，退栈；

③ 若ch=',': 表示开始处理右孩子节点，置*k*=2；

④ 其他情况（节点值）：

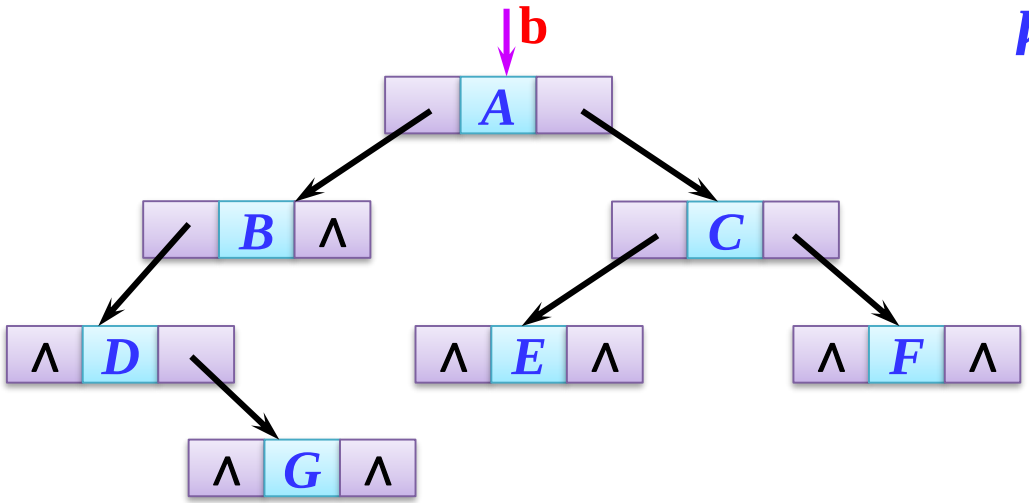
创建*p节点用于存放ch；

当*k*=1时，将*p节点作为栈顶节点的左孩子节点；

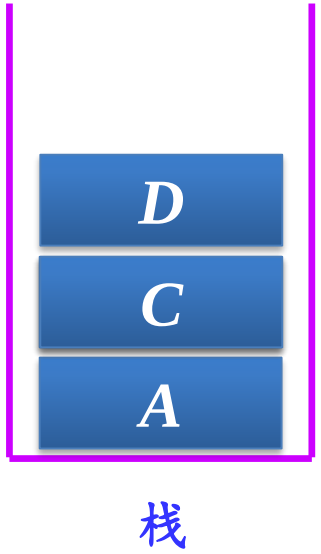
当*k*=2时，将*p节点作为栈顶节点的右孩子节点。

根据括号表示法字符串构造二叉链的演示

A (B (D (, G)) , C (E , F))



$k = 2$



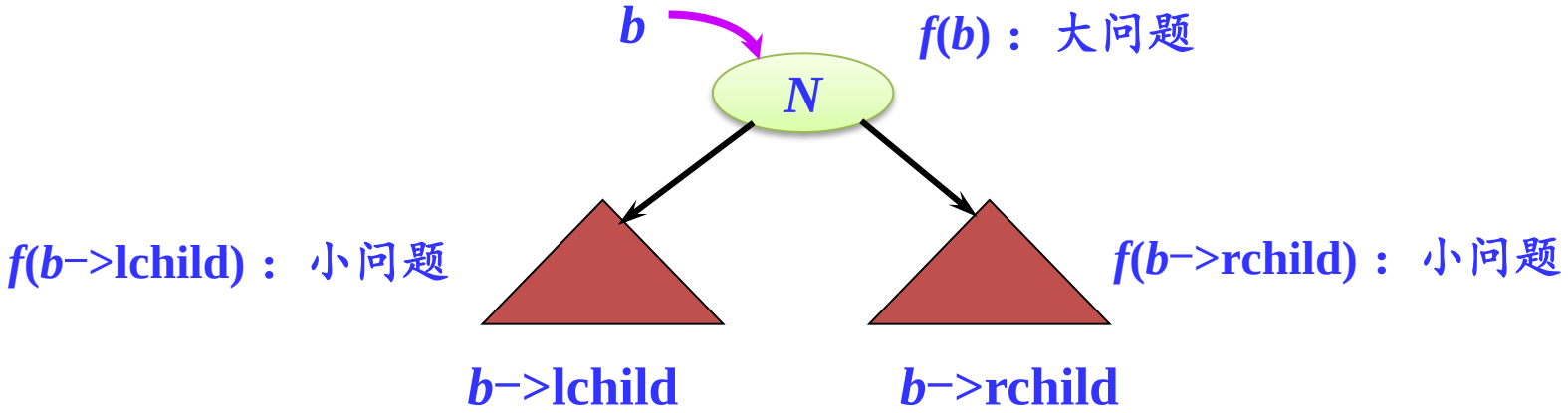
二叉链创建完毕

```
void CreateBTNode(BTNode * &b,char *str)
{    //由str ⇨ 二叉链b
     BTNode *St[MaxSize], *p;
     int top=-1,  k , j=0;
     char ch;
     b=NULL;                    //建立的二叉链初始时为空
     ch=str[j];
     while (ch!='\0')           //str未扫描完时循环
     {    switch(ch)
          {
            case '(': top++; St[top]=p; k=1; break;          //可能有左孩子节点，进栈
            case ')': top--;  break;
            case ',': k=2;  break;                           //后面为右孩子节点
```

```
            default:                                    //遇到节点值
                p=(BTNode *)malloc(sizeof(BTNode));
                p->data=ch;  p->lchild=p->rchild=NULL;
                if  (b==NULL)                           //p为二叉树的根节点
                    b=p;
                else                                    //已建立二叉树根节点
                {     switch(k)
                      {
                       case 1:  St[top]->lchild=p;  break;
                       case 2:  St[top]->rchild=p;  break;
                      }
                }
            }
            j++;  ch=str[j];                            //继续扫描str
        }
    }
```

（2）销毁二叉链DestroyBT(*b)

设*f*(*b*)销毁二叉链*b*：大问题。则*f*(*b*->lchild)销毁左子树，*f*(*b*->rchild)销毁右子树：两个小问题。



递归模型如下：

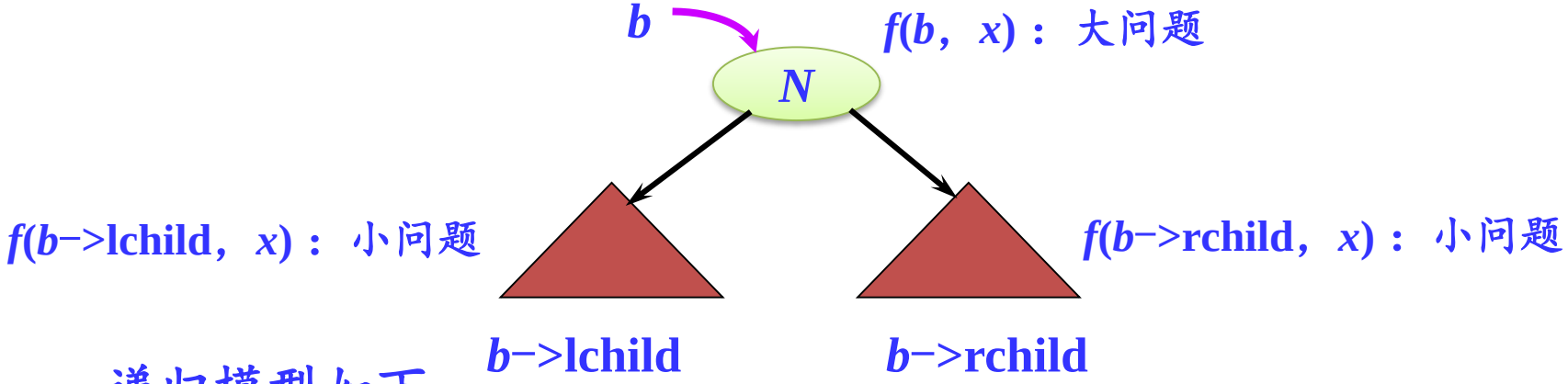
f(*b*) ≡ 不做任何事件　　若*b*=NULL
f(*b*) ≡ *f*(*b*->lchild)；*f*(*b*->rchild)；　　其他情况
释放**b*节点

对应的递归算法如下：

```
void   DestroyBT(BTNode *&b)
{      if (b==NULL) return ;
       else
       {     DestroyBT(b->lchild);
             DestroyBT(b->rchild);
             free(b);     //剩下一个节点*b，直接释放
       }
}
```

（3）查找节点FindNode(*b,x)

设*f*(*b*，*x*)在二叉树*b*中查找值为*x*的节点（唯一）。找到后返回其指针，否则返回NULL。



递归模型如下：

f(*b*,*x*) = NULL	若*b*=NULL
f(*b*,*x*) = *b*	若*b*->data==*x*
f(*b*,*x*) = *p*	若在左子树中找到了，即*p*=*f*(*b*->lchild,*x*)且*p*!=NULL
f(*b*,*x*) = *f*(*b*->rchild,*x*)	其他情况

对应的递归算法如下：

```
BTNode *FindNode(BTNode *b,ElemType x)
{    BTNode *p;
     if (b==NULL) return NULL;
     else if (b->data==x) return b;
     else
     {    p=FindNode(b->lchild,x);
          if (p!=NULL) return p;
          else return FindNode(b->rchild,x);
     }
}
```

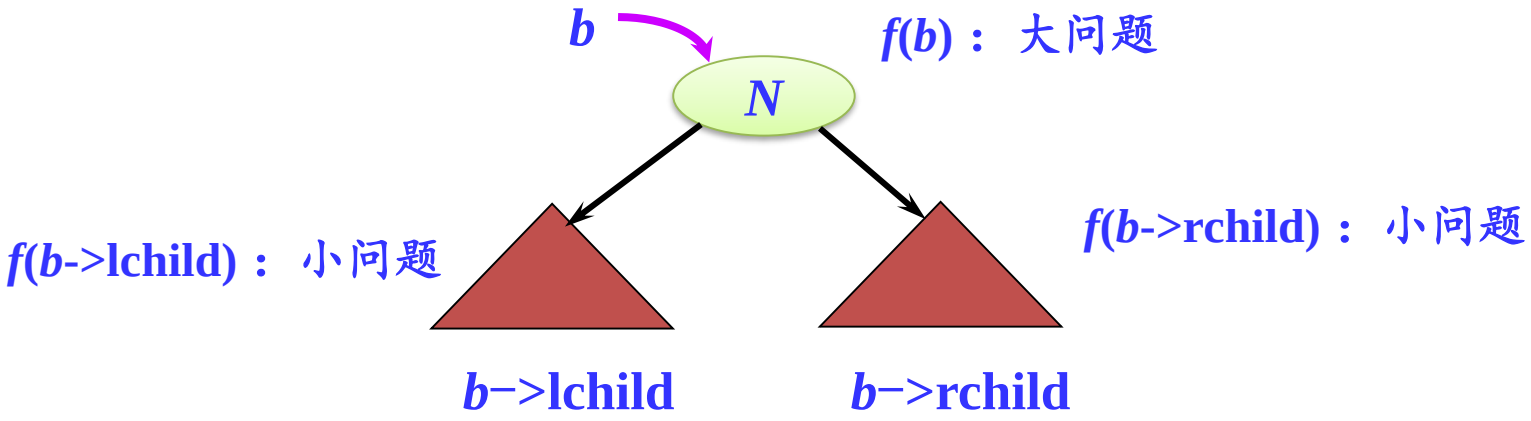
（4）找孩子节点LchildNode(p)和RchildNode(p)

直接返回*p节点的左孩子节点或右孩子节点的指针。

```
BTNode *LchildNode(BTNode *p)
{
    return p->lchild;
}

BTNode *RchildNode(BTNode *p)
{
    return p->rchild;
}
```

（5）求高度BTNodeDepth(*b)



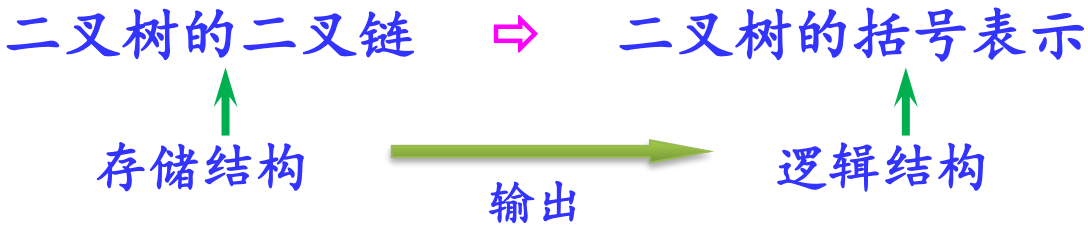
求二叉树的高度的递归模型f(b)如下：

f(b) = 0	b=NULL
f(b) = MAX{f(b->lchild),f(b->rchild)}+1	其他情况

对应的递归算法如下：

```
int BTNodeDepth(BTNode *b)
{    int lchilddep,rchilddep;
     if (b==NULL) return(0);          //空树的高度为0
     else
     {    lchilddep=BTNodeDepth(b->lchild);
                                      //求左子树的高度为lchilddep
          rchilddep=BTNodeDepth(b->rchild);
                                      //求右子树的高度为rchilddep
          return(lchilddep>rchilddep)? (lchilddep+1):(rchilddep+1));
     }
}
```

（6）输出二叉树DispBTNode(*b)



根节点（左子树，右子树） ⟵ 括号表示

```
void  DispBTNode(BTNode *b)
{   if (b!=NULL)
    {     printf("%c",b->data);
          if (b->lchild!=NULL || b->rchild!=NULL)
          {   printf("(");
              DispBTNode(b->lchild);   //递归处理左子树
              if (b->rchild!=NULL) printf(",");
              DispBTNode(b->rchild);   //递归处理右子树
              printf(")");
          }
    }
}
```

思考题：

本讲算法都是以二叉链为存储结构，如果采用顺序存储结构，算法如何改写？

——本讲完——