4.2 二叉树的变形

4.2.1 二叉排序树（BST）

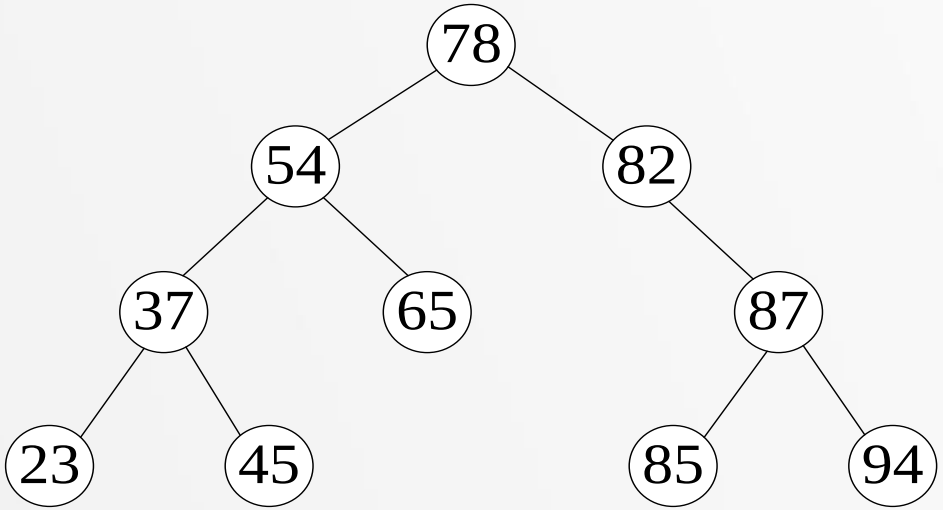
4.2.2 平衡二叉树(AVL)

4.2.3 哈夫曼树及哈夫曼编码

4.2.4 堆排序

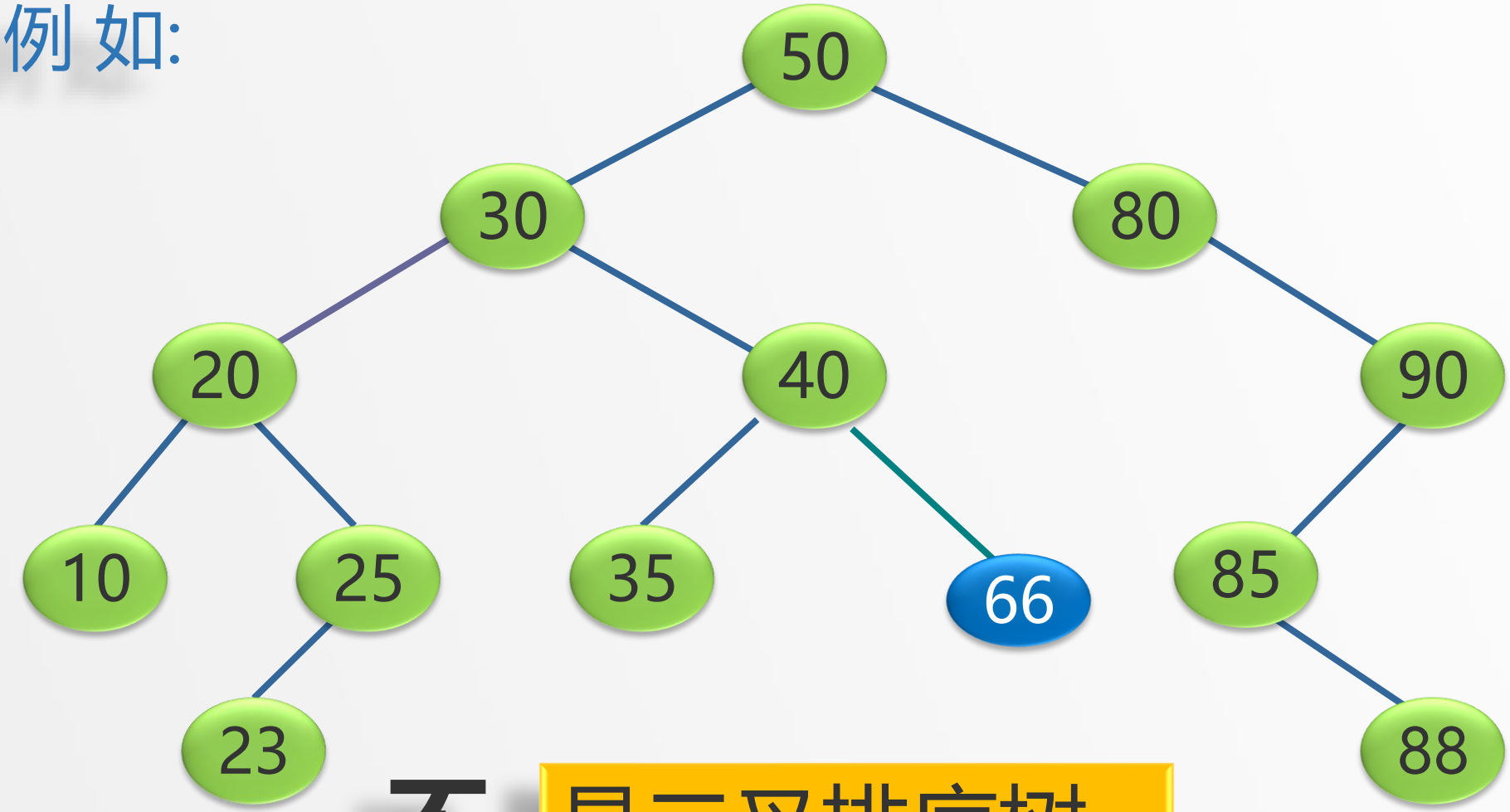
4.2 二叉树的变形 | 4.2.1二叉排序树（BST）

1. 定义



中序遍历序列：
23, 37, 45, 54, 65, 78, 82, 85, 87, 94

例 如:

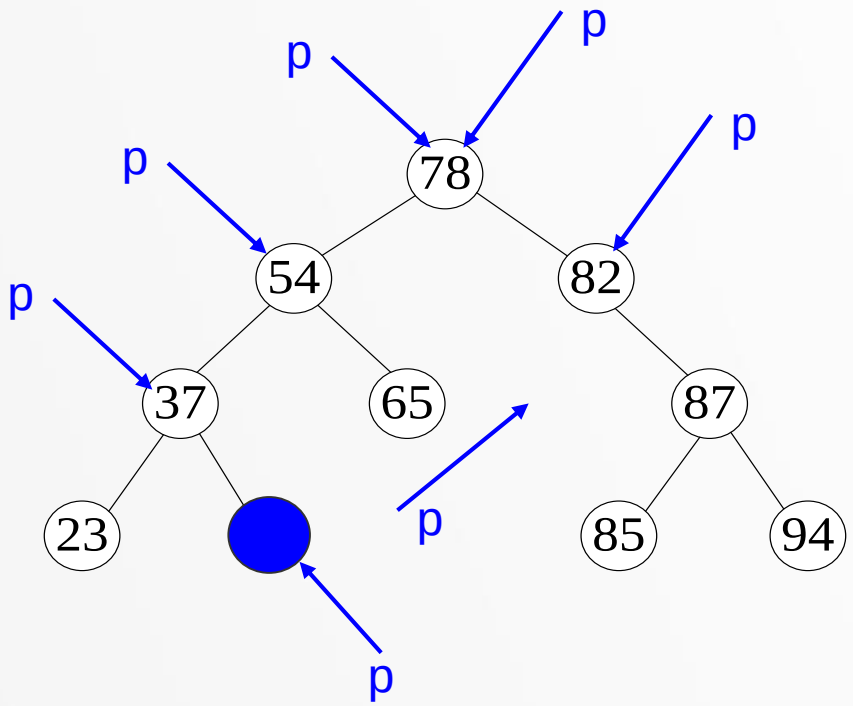


不 是二叉排序树。

4.2.1 二叉排序树 | 查找

key=45 ✓

key=81 ✗



3．二叉排序树的插入算法

“插入”操作在**查找不成功**时才进行；

当二叉排序树中不存在关键字等于 e.key 的数据元素时，插入元素值为 e 的结点，并返回 TRUE; 否则，不进行插入并返回FALSE

```
Status Insert BST(BiTree &T, ElemType e )
{
    if (!SearchBST ( T, e.key, p ))
    {     ...     ...     }
  else return FALSE;
} // Insert BST
```

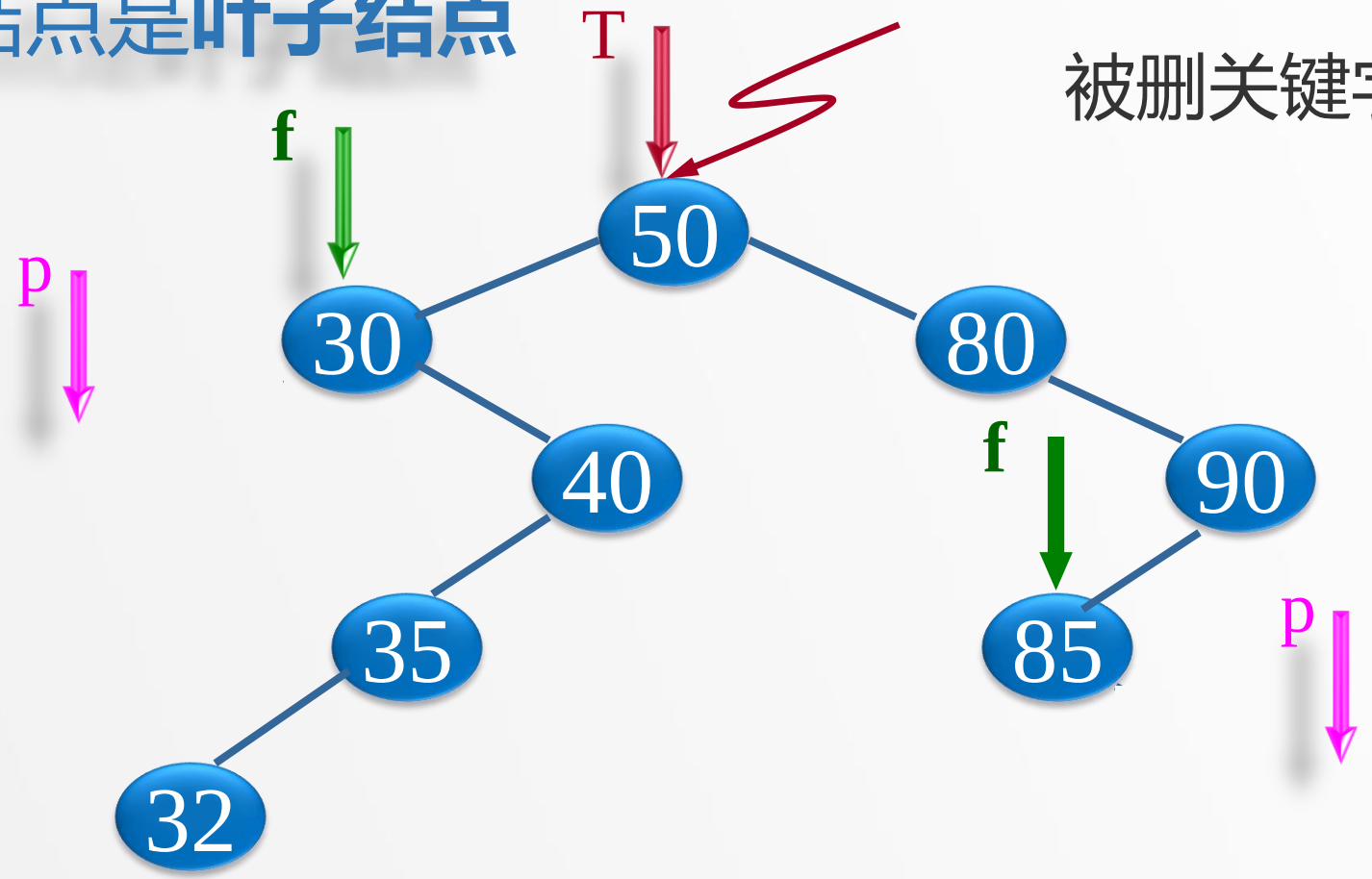
```
s = (BiTree) malloc (sizeof (BiTNode));
                    /*为新结点分配空间*/
s->data = e;
s->lchild = s->rchild = NULL;

if ( !p ) T = s;     /* 插入 s 为新的根结点*/

else   if (e.key< p->data.key)
       p->lchild = s;     /* 插入 *s 为 *p 的左孩子*/
else  p->rchild = s;  /* 插入 *s 为 *p 的右孩子*/

return TRUE;     /* 插入成功*/
```

4．二叉排序树的删除算法

删除可分**三种情况**讨论：

（1）被删除的结点是叶子
（2）被删除的结点只有左子树或者只有右子树
（3）被删除的结点既有左子树，也有右子树

（1）被删除的结点是**叶子结点**

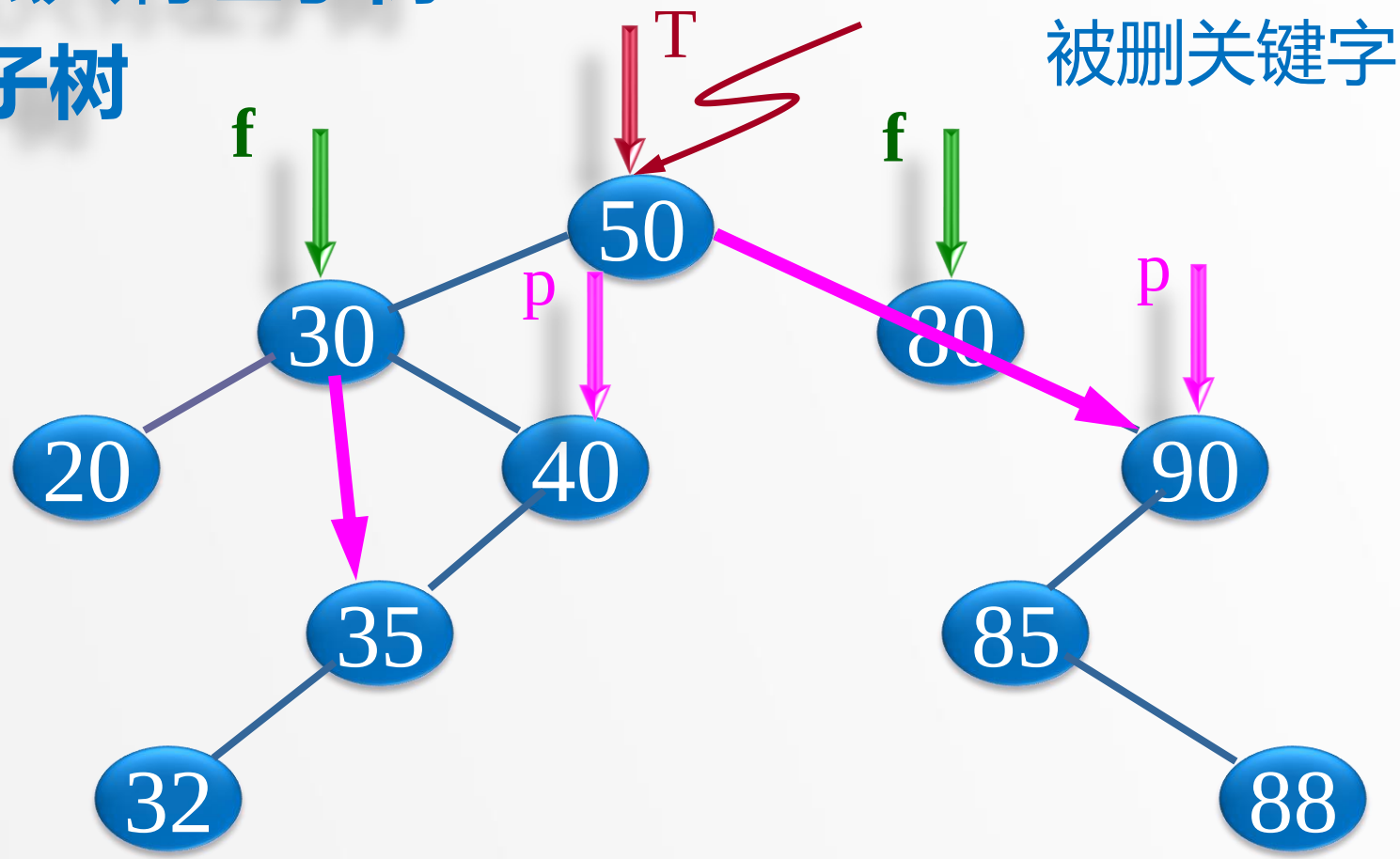


被删关键字 = **88**

其双亲结点中相应指针域的值改为“空”

(2) 被删除的结点只有**左子树**或者只有**右子树**

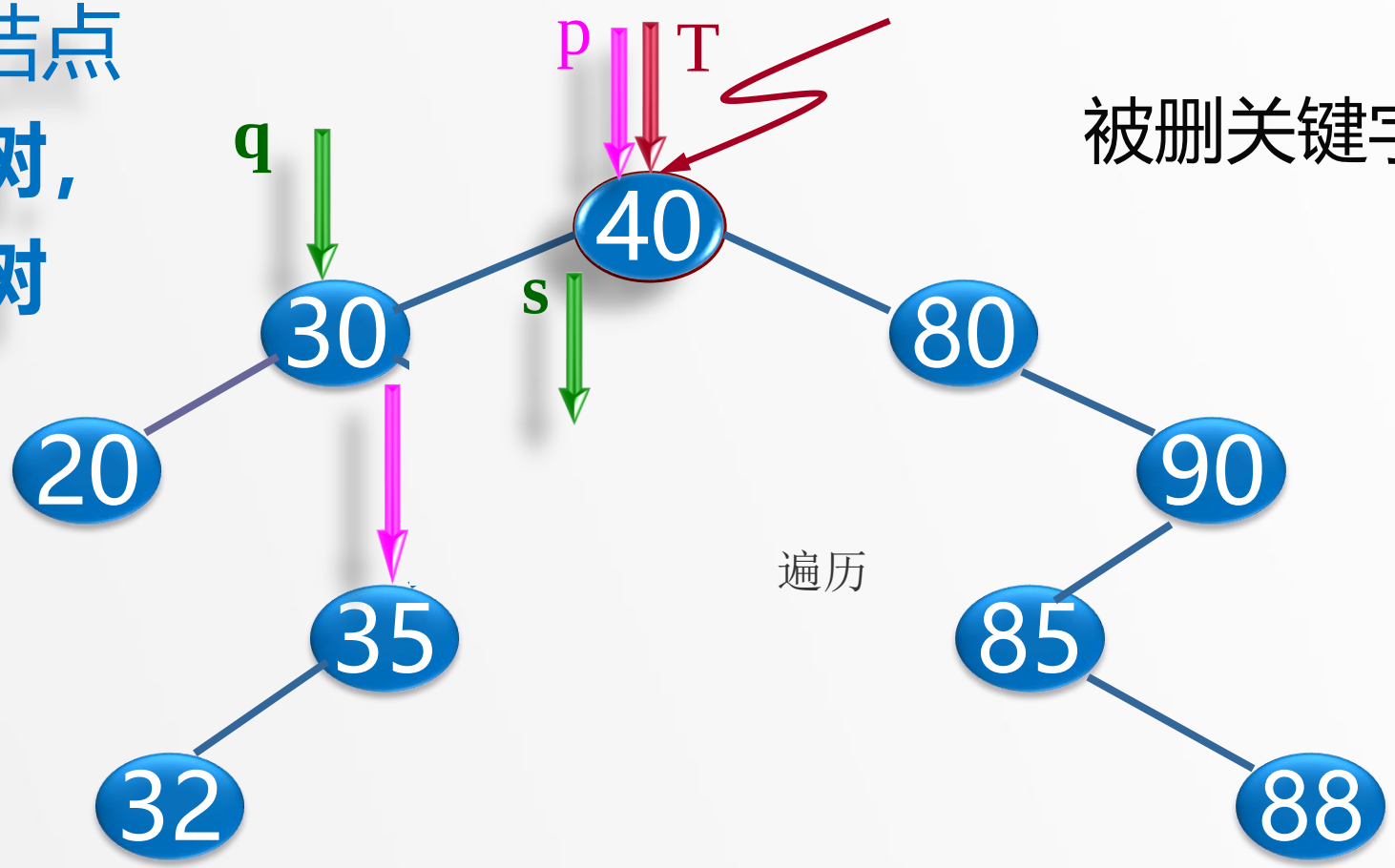
被删关键字 = **80**



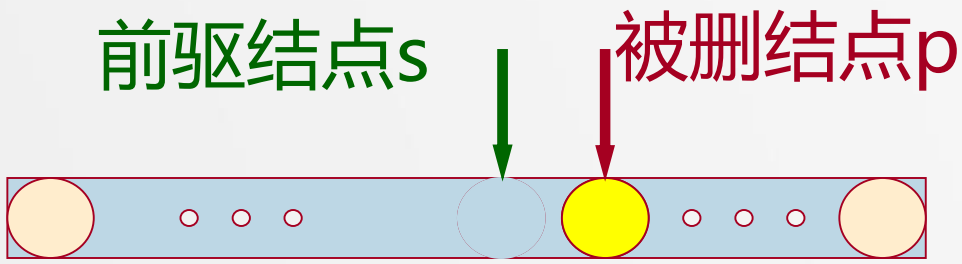
其双亲结点的相应指针域的值改为 “指向被删除结点的左子树或右子树”。

(3) 被删除的结点

既有左子树，
也有右子树



被删关键字 = 50



以其前驱替代之，然后再删除该前驱结点

讨 论

被删除的结点既有左子树，也有右子树时,能否用被删除结点的后继结点进行替换操作?

算法描述如下：

```
Status DeleteBST (BiTree &T, KeyType key ) {
/* 若二叉排序树 T 中存在其关键字等于 key 的*/
/* 数据元素，则删除该数据元素结点，并返回*/
/*函数值 TRUE，否则返回函数值 FALSE*/
  if (!T)  return FALSE;
          /*不存在关键字等于key的数据元素*/
  else {     …… ……       }
} /* DeleteBST*/
```

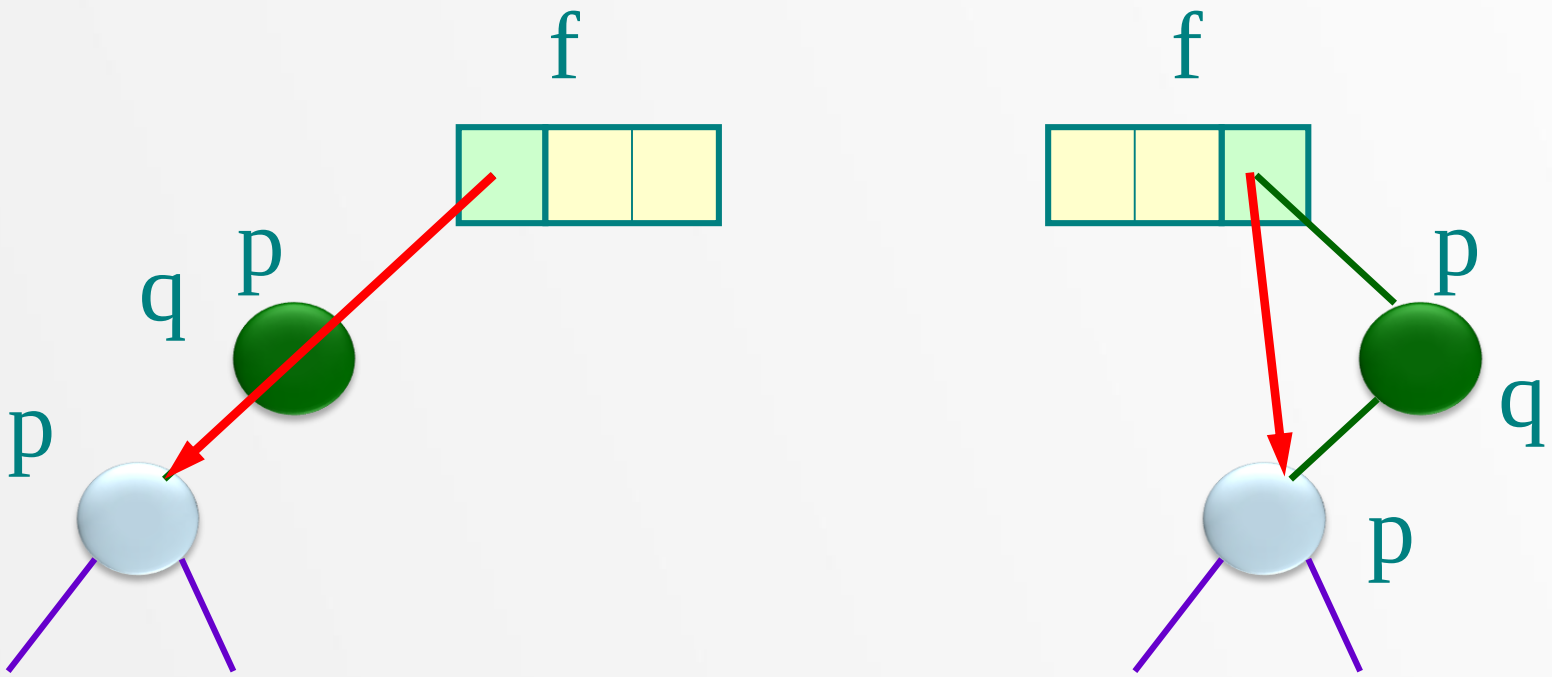
```
if (key==T->data.key )
    { Delete (T);   return TRUE; }
        /* 找到关键字等于key的数据元素*/
else if (key<T->data.key))

else     DeleteBST ( T->lchild, key );
        /* 继续在左子树中进行查找*/
         DeleteBST ( T->rchild, key );
        /*继续在右子树中进行查找*/
```

其中**删除操作**过程如下所描述：

```
void Delete ( BiTree &p ){
  /* 从二叉排序树中删除结点 p，*/
  /* 并重接它的左子树或右子树*/
  if (!p->rchild) { …… } /*只有左子树*/
  else if (!p->lchild) { …… } /*只有右子树*/
  else { …… }                 /*左右子树均有*/
} /* Delete*/
```

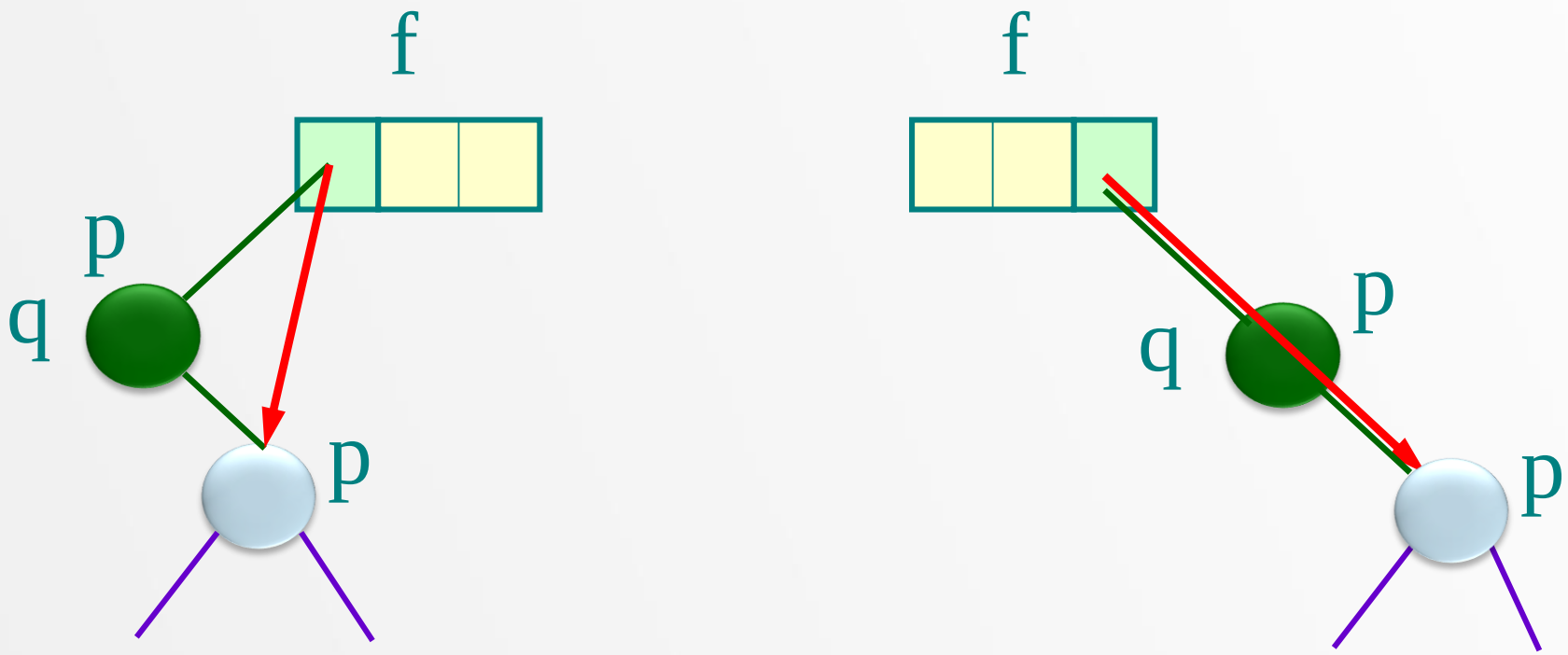
/* 右子树为空树则只需重接它的左子树*/

q = p; p = p->lchild; f-> lchild=p;free(q);
(f-> rchild=p)



/* 左子树为空树只需重接它的右子树*/

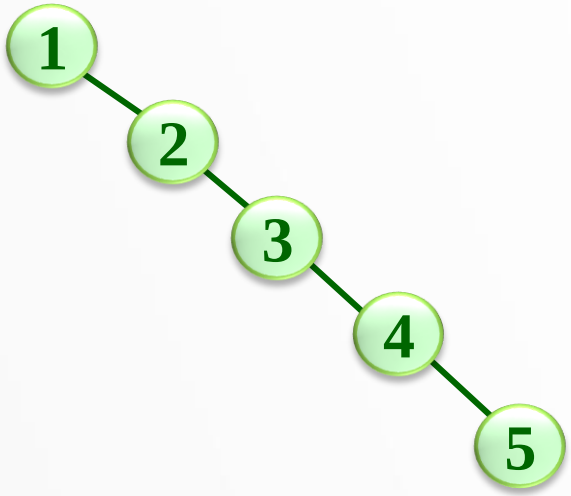
q = p; p = p->rchild; f-> lchild=p; free(q);
(f-> rchild=p)



```
                /* 左右子树均不空*/
q = p;  s = p->lchild;
while (s->rchild) { q = s;  s = s->rchild; }
    /* s 指向被删结点p的前驱,q与s同步*/

p->data = s->data;
if (q != p )  q->rchild = s->lchild;
else  q->lchild = s->lchild;
                        /* 重接*q的左子树*/
free(s);/*S的内容已复制到P上，即P实际上已变成S*/
```

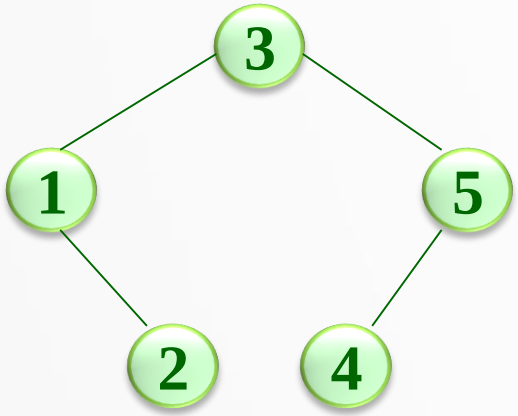
5．查找性能的分析

例如：由关键字序列 1，2，3，4，5构造而得的二叉排序树，



$$ASL = (1+2+3+4+5) / 5$$
$$= 3$$

由关键字序列 3，1，2，5，4构造而得的二叉排序树，



$$ASL = (1+2+3+2+3) / 5$$
$$= 2.2$$

作业:

给定一组元素{17,28,36,54,30,27,94,15,21,83,40},画出由此生成的二叉排序树.