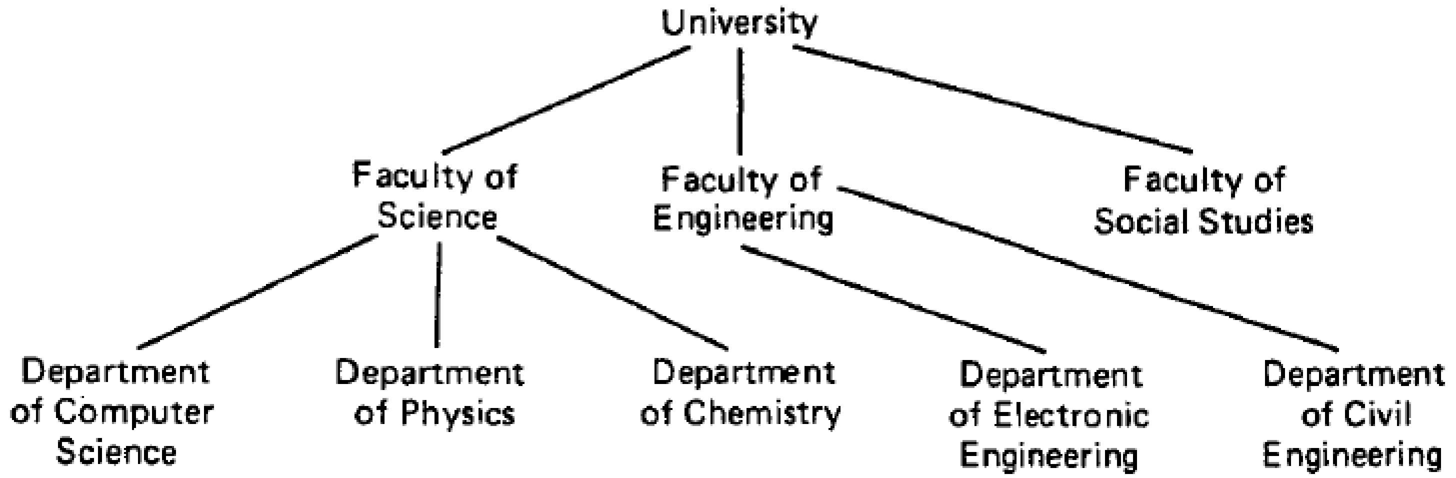
# 计算机问题求解 – 论题2-14

- 树及搜索树

2016年05月26日

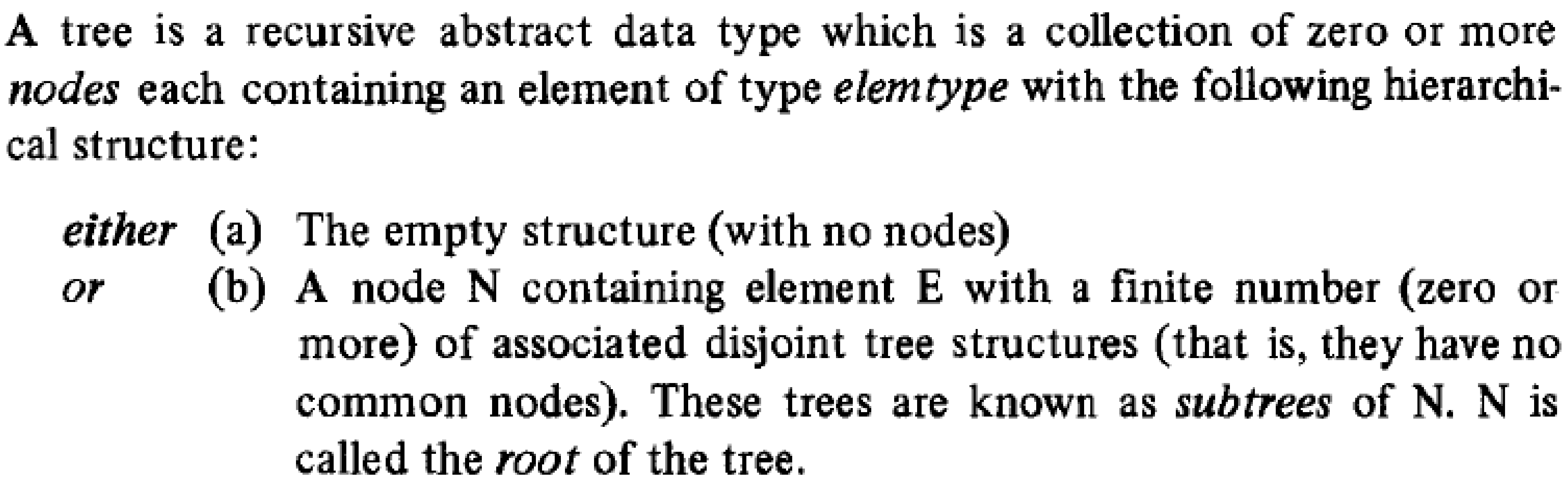
## 如何去定义表达如下结构的“树”数据结构？


University
Faculty of Science
Faculty of Engineering
Faculty of Social Studies
Department of Computer Science
Department of Physics
Department of Chemistry
Department of Electronic Engineering
Department of Civil Engineering

动态集合中元素、元素之间的关系

这种集合上的基本操作及特殊操作

## 树中元素及元素之间的关系

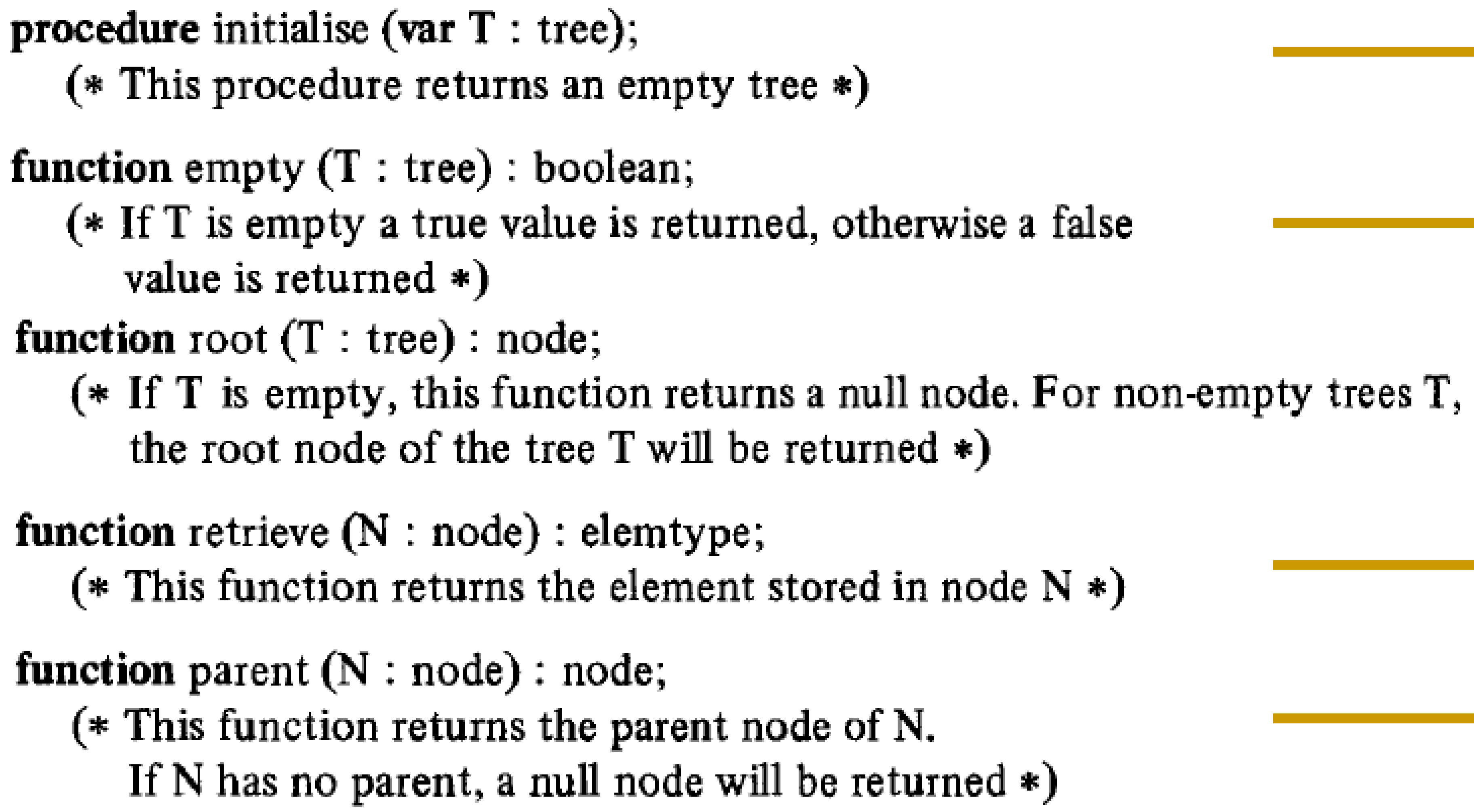

A tree is a recursive abstract data type which is a collection of zero or more *nodes* each containing an element of type *elemtype* with the following hierarchical structure:

*either* (a) The empty structure (with no nodes)

*or* (b) A node N containing element E with a finite number (zero or more) of associated disjoint tree structures (that is, they have no common nodes). These trees are known as *subtrees* of N. N is called the *root* of the tree.

你如何理解这两个概念：层次结构、递归ADT？

# 你能说清楚以下定义在树中元素上的操作哪些是基本操作(B)？哪些是特殊操作(S)吗？



```
procedure initialise (var T : tree);
    (* This procedure returns an empty tree *)

function empty (T : tree) : boolean;
    (* If T is empty a true value is returned, otherwise a false
       value is returned *)
function root (T : tree) : node;
    (* If T is empty, this function returns a null node. For non-empty trees T,
       the root node of the tree T will be returned *)

function retrieve (N : node) : elemtype;
    (* This function returns the element stored in node N *)

function parent (N : node) : node;
    (* This function returns the parent node of N.
       If N has no parent, a null node will be returned *)
```

B

B

S

B

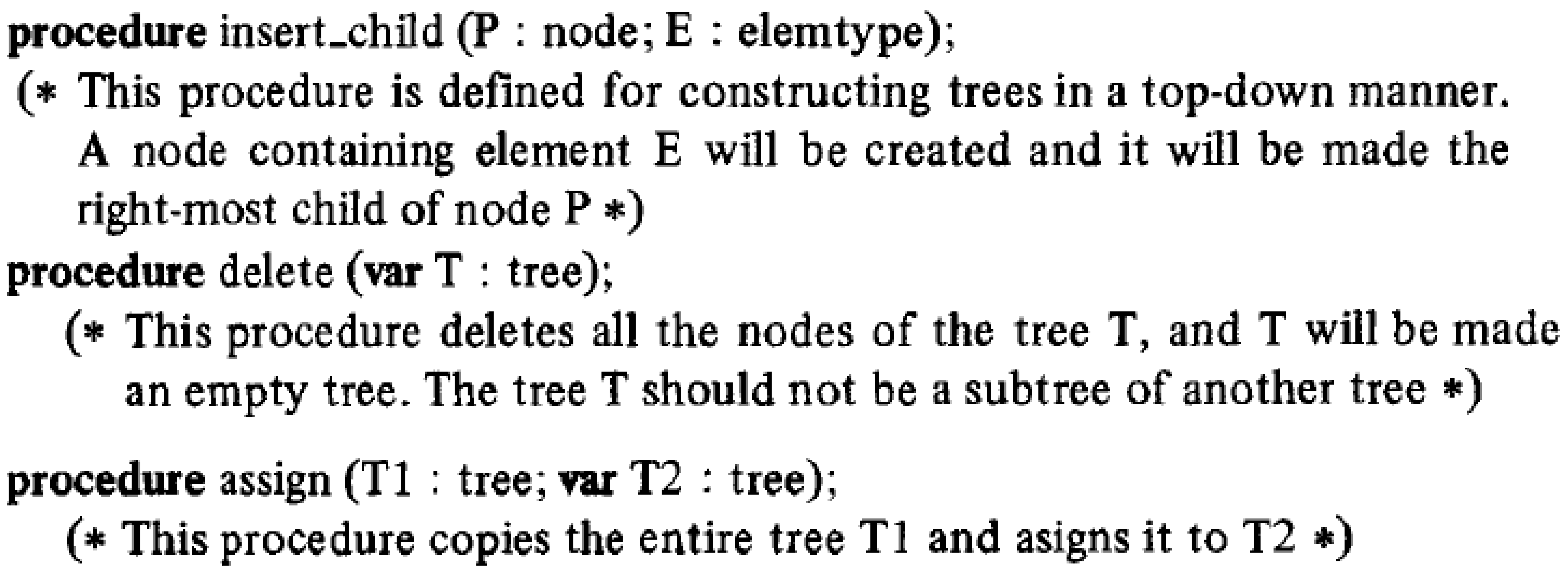
S

**function** left_most_child (N : node) : node;
(* This function returns the left-most child of the node N.
If there are no children, a null node will be returned *) S

**function** right_sibling (N : node) : node;
(* If node N has a sibling node, it will be returned as the value of this function. Otherwise, a null node will be returned *) S

**function** null_node (N : node) : boolean;
(* This function returns a true value if and only if N is a null node *) B/S

**function** cons_tree$_n$ (E : elemtype; $T_1, T_2, \ldots, T_n$: tree) : tree;
(* This function constructs a tree with a node N as its root containing element E and the trees $T_1, T_2, \ldots, T_n$ as its children, in that order. The trees $T_1, T_2, \ldots, T_n$ should not be used again after this operation to construct other trees. If copies of these subtrees are required, they can be constructed using the assign(T, T′) operation. This is meant for space-efficiency so that this function need not copy these subtrees in order to make a new tree *) B



```
procedure insert_child (P : node; E : elemtype);
 (* This procedure is defined for constructing trees in a top-down manner.
    A node containing element E will be created and it will be made the
    right-most child of node P *)
procedure delete (var T : tree);
   (* This procedure deletes all the nodes of the tree T, and T will be made
      an empty tree. The tree T should not be a subtree of another tree *)

procedure assign (T1 : tree; var T2 : tree);
   (* This procedure copies the entire tree T1 and asigns it to T2 *)
```

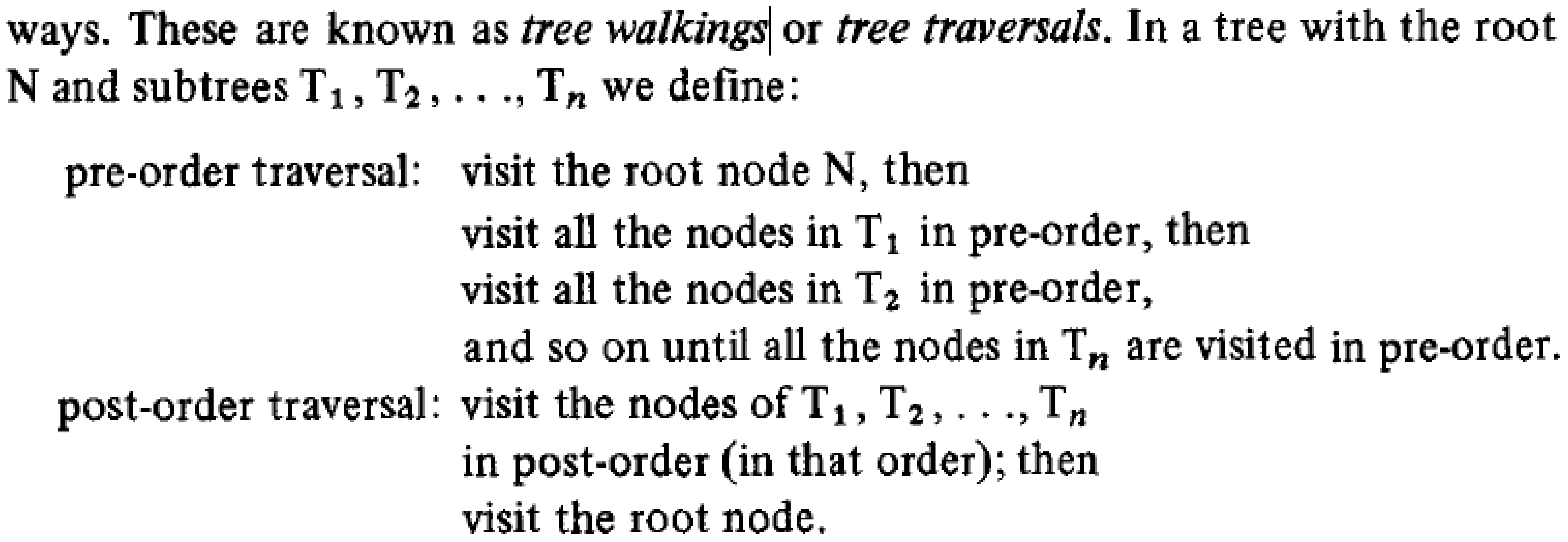
B

B

S/B

你能够从你的判断中得到一个一般性的规律吗：
当你定义一个数据结构时，哪些操作是必须的，哪些操作是特定的？

## 你有没有感觉到，在树这样的数据结构中，递归如呼吸般自然？

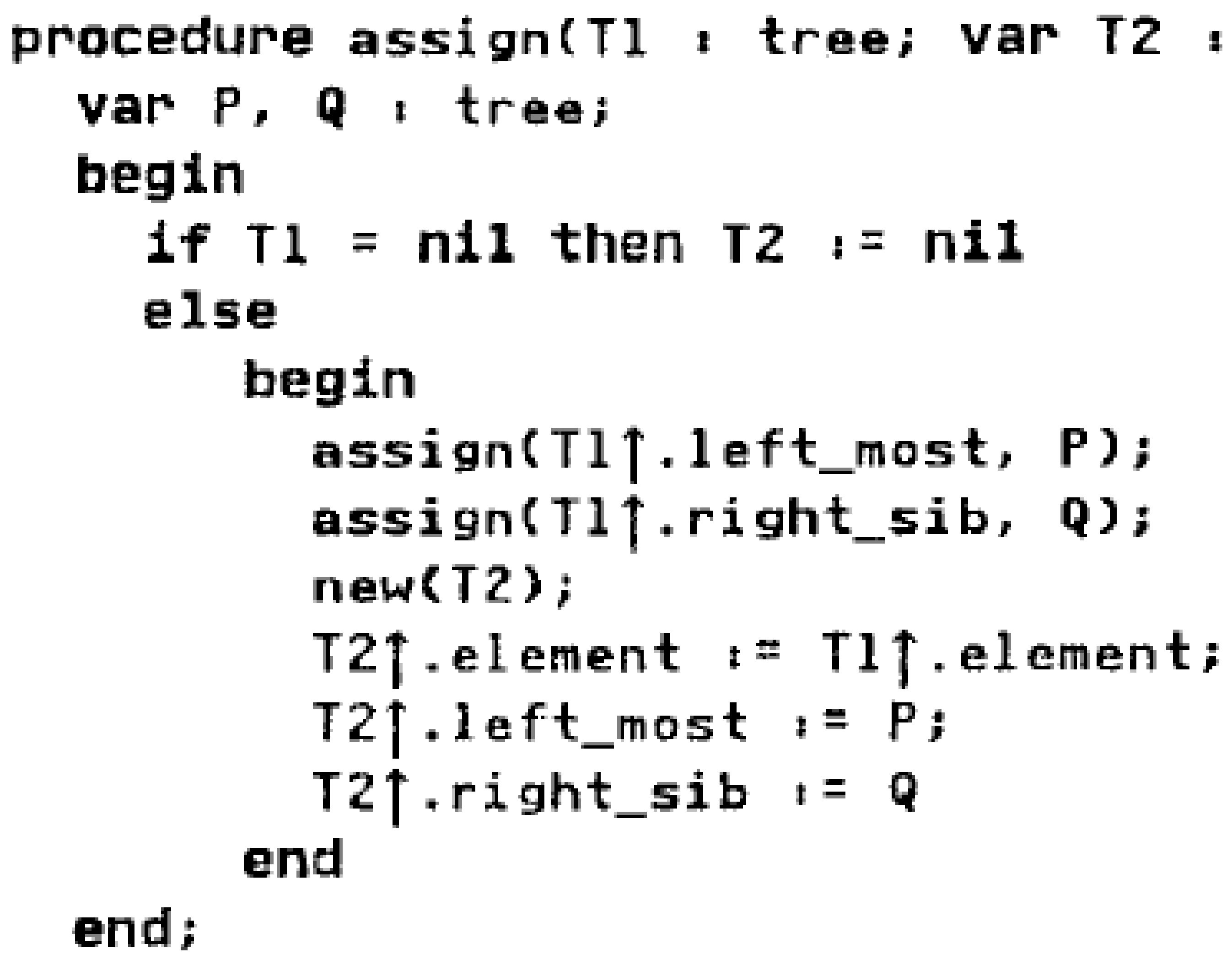


ways. These are known as *tree walkings* or *tree traversals*. In a tree with the root N and subtrees $T_1$, $T_2$, . . ., $T_n$ we define:

pre-order traversal: visit the root node N, then
visit all the nodes in $T_1$ in pre-order, then
visit all the nodes in $T_2$ in pre-order,
and so on until all the nodes in $T_n$ are visited in pre-order.

post-order traversal: visit the nodes of $T_1$, $T_2$, . . ., $T_n$
in post-order (in that order); then
visit the root node.

## 将一棵树赋值给另一棵树，原来也很简单

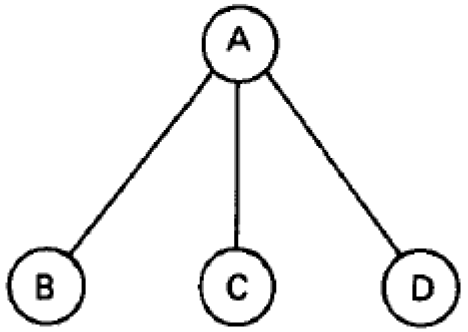


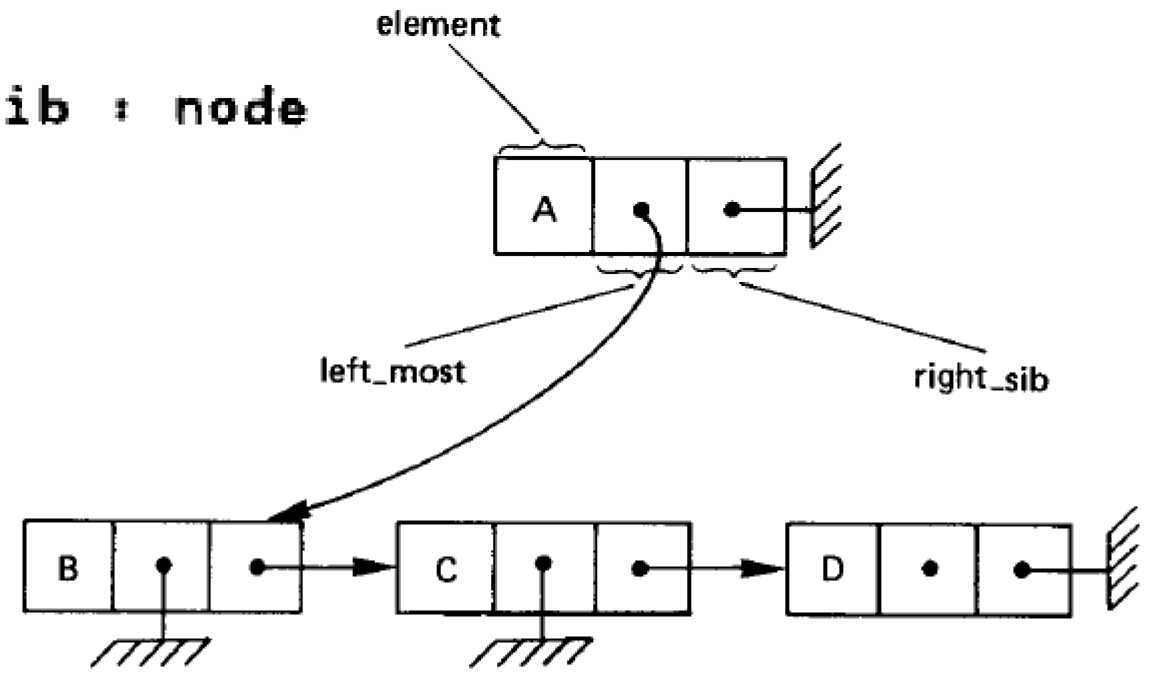
```
procedure assign(T1 : tree; var T2 : tree);
   var P, Q : tree;
   begin
      if T1 = nil then T2 := nil
      else
         begin
            assign(T1↑.left_most, P);
            assign(T1↑.right_sib, Q);
            new(T2);
            T2↑.element := T1↑.element;
            T2↑.left_most := P;
            T2↑.right_sib := Q
         end
   end;
```

你的第一想法是循环还是递归？
你最终的做法是循环还是递归？

# 你能从下图的一般的树的链表实现中，设计一个二叉树的链表实现吗？

```
type node = ↑node_cell
     node_cell = record
                    element : elmtype,
                    left_most, right_sib : node
                 end;
     tree = node;
```

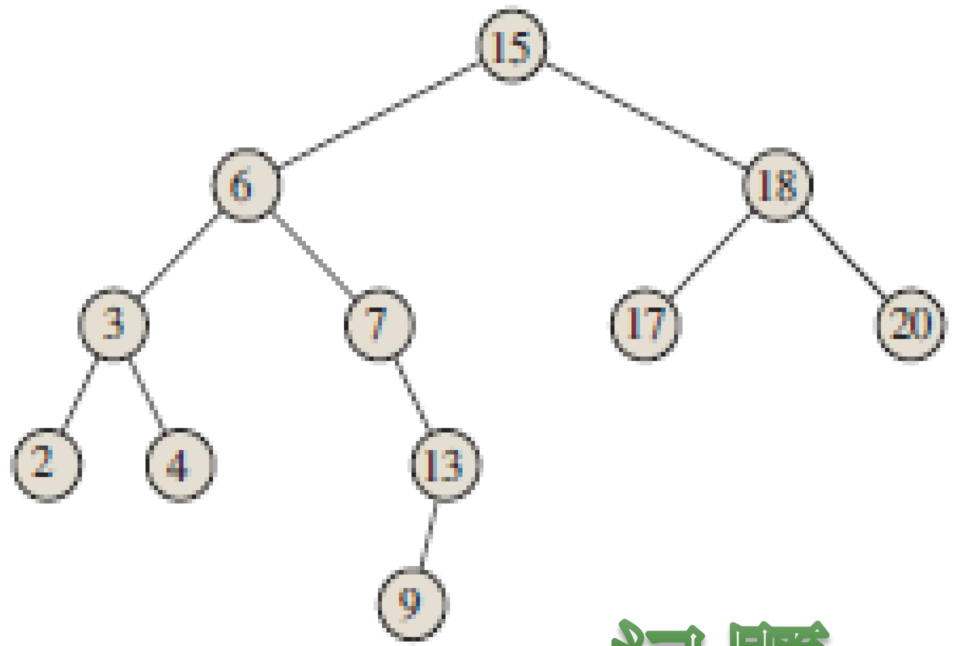


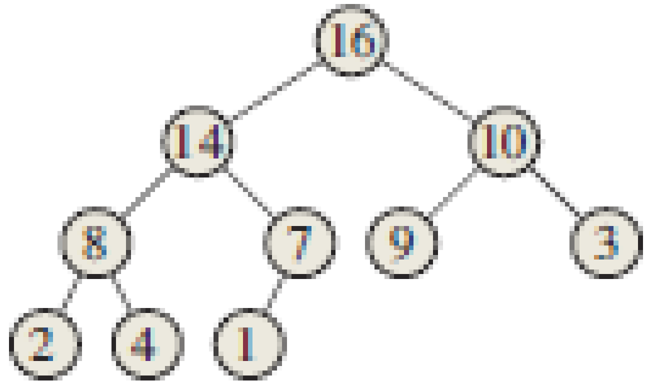


```
class Node
   {
   int iData;                          // data used as key value
   float fData;                        // other data
   node leftChild;                     // this node's left child
   node rightChild;                    // this node's right child

   public void displayNode()
      {
      // (see Listing 8.1 for method body)
      }

   }
```

```
class Tree
   {
   private Node root;            // the only data field in Tree

   public void find(int key)
      {
      }
   public void insert(int id, double dd)
      {
      }
   public void delete(int id)
      {
      }
   // various other methods

   }  // end class Tree
```

```
class TreeApp
   {
   public static void main(String[] args)
      {
      Tree theTree = new Tree;         // make a tree

      theTree.insert(50, 1.5);         // insert 3 nodes
      theTree.insert(25, 1.7);
      theTree.insert(75, 1.9);

      node found = theTree.find(25);  // find node with key 25
      if(found != null)
         System.out.println("Found the node with key 25");
      else
         System.out.println("Could not find node with key 25");
      }  // end main()

   }  // end class TreeApp
```





问题:

这是什么结构? 他们有什么相同与不同之处?

```
INORDER-TREE-WALK(x)
1  if $x \neq$ NIL
2      INORDER-TREE-WALK(x.left)
3      print x.key
4      INORDER-TREE-WALK(x.right)
```

## 问题:

这个过程将得到一个binary search tree中所有元素的按序输出。你能证明吗?你能想到几种方法去证明?

## “扫描”BST的代价是线性的

If $x$ is the root of an $n$-node subtree, then the call INORDER-TREE-WALK$(x)$ takes $\Theta(n)$ time.

为什么你会猜出这么一个结论？

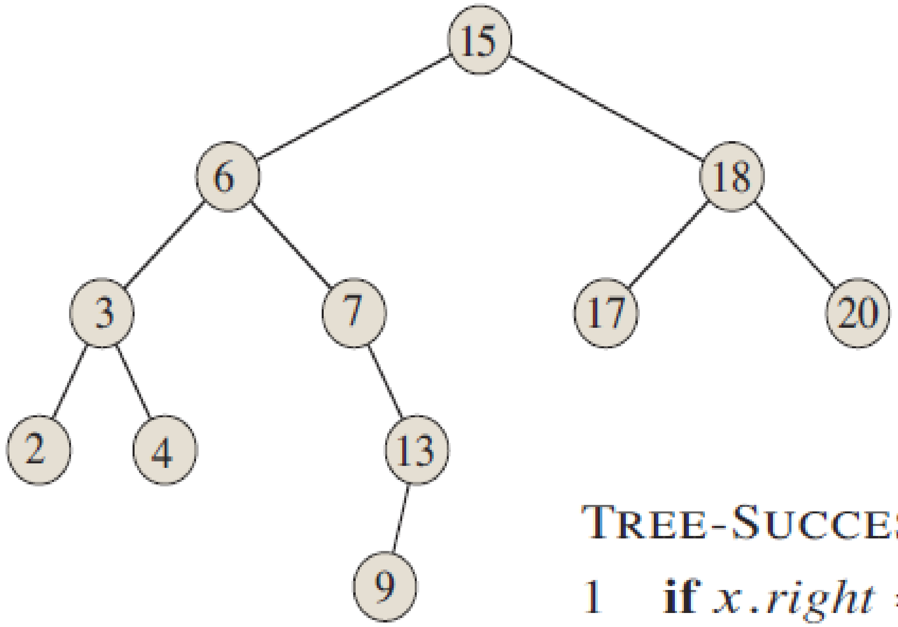
We use the substitution method to show that $T(n) = O(n)$ by proving that $\boxed{T(n) \le (c+d)n+c}$. For $n = 0$, we have $(c+d)\cdot 0+c = c = T(0)$. For $n > 0$, we have

$$\begin{aligned} T(n) &\le T(k)+T(n-k-1)+d \\ &= ((c+d)k+c)+((c+d)(n-k-1)+c)+d \\ &= (c+d)n+c-(c+d)+c+d \\ &= (c+d)n+c\,, \end{aligned}$$

其实是数学归纳法

其实我们还需要使用数学归纳法来证明我们的猜测

## BST中结点的“后继”



问题：

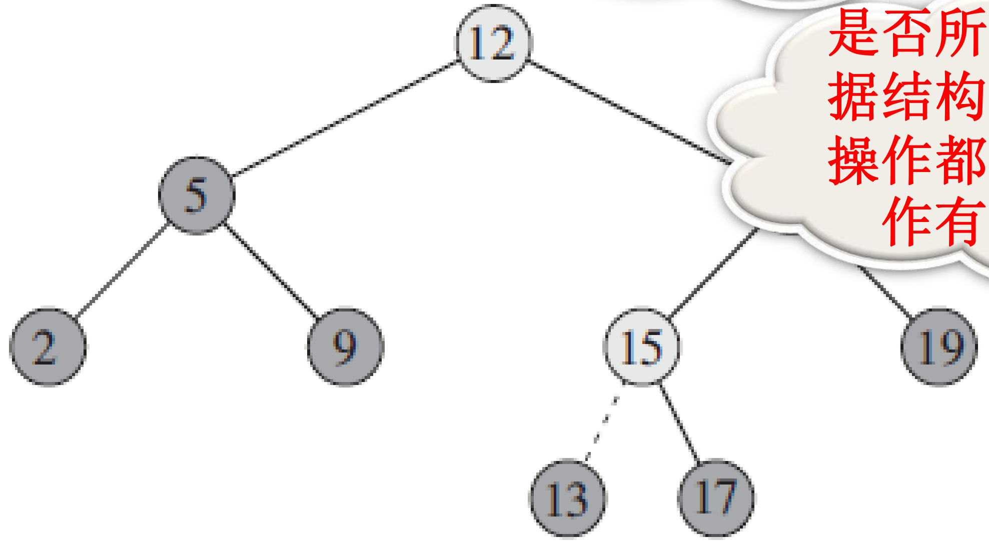
你能否结合左图，解释下面的过程，特别是红框中的部分？

```
TREE-SUCCESSOR(x)
1  if x.right ≠ NIL
2      return TREE-MINIMUM(x.right)
3  y = x.p
4  while y ≠ NIL and x == y.right
5      x = y
6      y = y.p
7  return y
```

# 插入一个元素

```
TREE-INSERT(T, z)
y = NIL
x = T.root
while x ≠ NIL
    y = x
    if z.key < x.key
        x = x.left
    else x = x.right
z.p = y
if y == NIL
    T.root = z
elseif z.key < y.key
    y.left = z
else y.right = z
```

插入操作和查询操作有什么关联？



是否所有的的数据结构上的插入操作都和查询操作有关联？

runs in $O(h)$ time on a tree of height $h$.

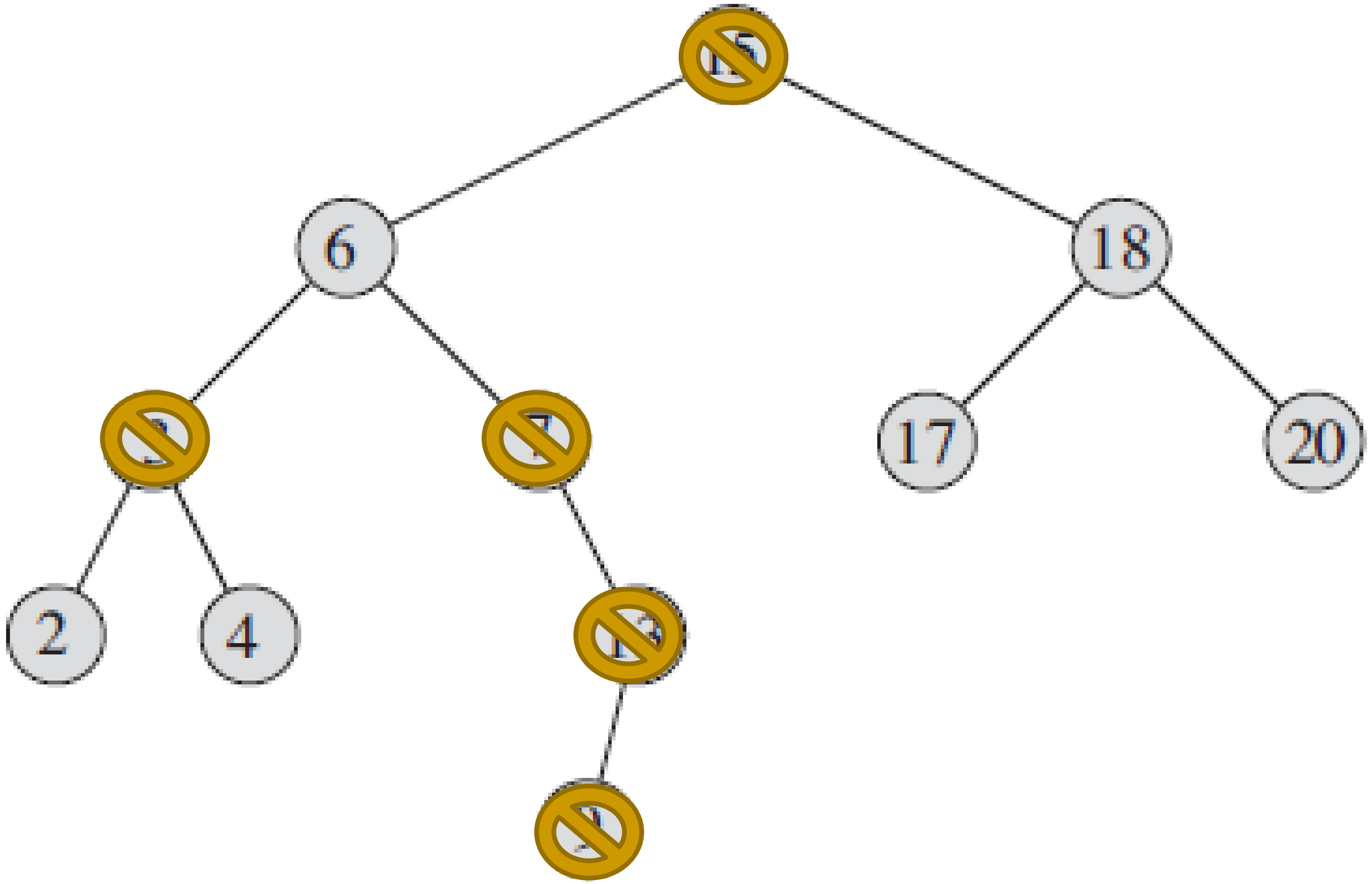
## 问题:

为什么在搜索树中删除比插入复杂?

问题0.1:
插入一个元素，必定是增加一个叶节点?

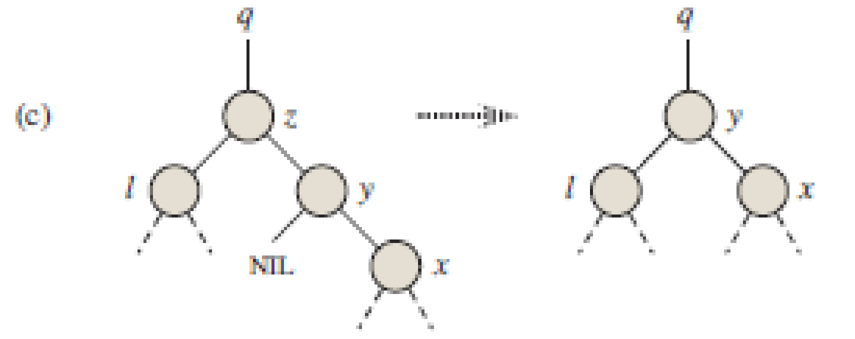
问题0.2:
什么叫删除和插入一个元素?

## 删除一个元素和分情形证明法：

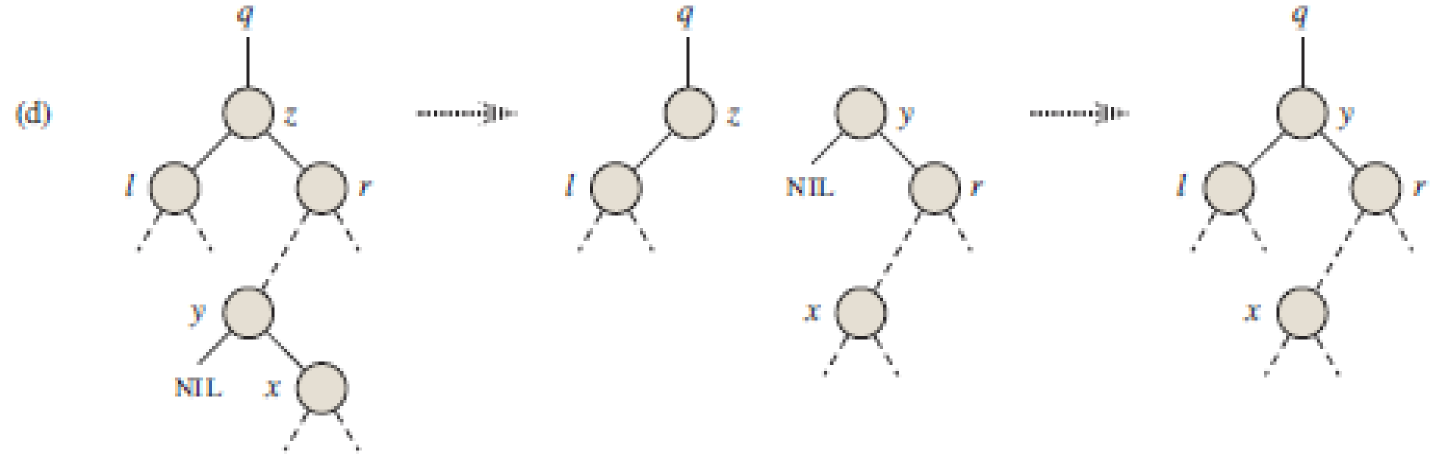


# 从BST中删除

假设待删除元素所在结点左右子树皆非空：



关键是：待删除元素的后继是否为其右子结点（即右子树的根）。



## 问题6:

## BST是否能够有效地实现动态集合,为什么?

平衡程度是关键!

## 随机构造的BST的高度

***Theorem 12.4***

The expected height of a randomly built binary search tree on $n$ distinct keys is $O(\lg n)$.

## 定义：

- 定义随机变量：
  - $X_n$： 树的高度；
  - $R_n$： 根在以它为根的树中所有元素的排位；
    - 如果$R_n$总是在中间，树的高度最小；
  - $Z_{n,i}$： $R_n=i$的指示器随机变量

$$\mathrm{E}[Z_{n,i}] = 1/n$$

- 定义辅助随机变量：指数高度 $Y_n = 2^{X_n}$
  - 如果$R_n=i$， $Y_n = 2 \cdot \max(Y_{i-1}, Y_{n-i})$
  - 同时我们还有：

$$Y_n = \sum_{i=1}^{n} Z_{n,i} \left(2 \cdot \max(Y_{i-1}, Y_{n-i})\right)$$

We shall show that $\mathrm{E}[Y_n]$ is polynomial in $n$, which will ultimately imply that $\mathrm{E}[X_n] = O(\lg n)$.

$$\begin{aligned}
\mathrm{E}[Y_n] &= \mathrm{E}\left[\sum_{i=1}^{n} Z_{n,i}\,(2 \cdot \max(Y_{i-1}, Y_{n-i}))\right] \\
&= \sum_{i=1}^{n} \mathrm{E}[Z_{n,i}\,(2 \cdot \max(Y_{i-1}, Y_{n-i}))] && \text{(by linearity of expectation)} \\
&= \sum_{i=1}^{n} \mathrm{E}[Z_{n,i}]\,\mathrm{E}[2 \cdot \max(Y_{i-1}, Y_{n-i})] && \text{(by independence)} \\
&= \sum_{i=1}^{n} \frac{1}{n} \cdot \mathrm{E}[2 \cdot \max(Y_{i-1}, Y_{n-i})] && \text{(by equation (12.1))} \\
&= \frac{2}{n} \sum_{i=1}^{n} \mathrm{E}[\max(Y_{i-1}, Y_{n-i})] && \text{(by equation (C.22))} \\
&\le \frac{2}{n} \sum_{i=1}^{n} (\mathrm{E}[Y_{i-1}] + \mathrm{E}[Y_{n-i}]) && \text{(by Exercise C.3-4)}\ . \\
&\le \frac{4}{n} \sum_{i=0}^{n-1} \mathrm{E}[Y_i]
\end{aligned}$$

Using the substitution method, we shall show that for all positive integers $n$, the recurrence (12.2) has the solution

$$\mathrm{E}[Y_n] \le \frac{1}{4}\binom{n+3}{3} .$$

$$\begin{aligned} 2^{\mathrm{E}[X_n]} &\le \mathrm{E}\left[2^{X_n}\right] \\ &= \mathrm{E}[Y_n] , \end{aligned}$$

as follows:

$$2^{\mathrm{E}[X_n]} \le \frac{1}{4}\binom{n+3}{3}$$

Taking logarithms of both sides gives $\mathrm{E}[X_n] = O(\lg n)$

## Open topics

- 请两两组队设计一个可使用搜索树的算法问题并解决它：一个同学定义并实现一个“搜索树”数据结构，另一个同学编写一个使用该结构解决该问题的算法；

- 请证明中根遍历算法能够将一个BST中的所有元素进行一次按序输出
  - 你能给出几种证明方法？

## 课外作业

- TC pp.289-: ex.12.1-2, 12.1-5
- TC pp.293-: ex.12.2-5, 12.2-8, 12.2-9
- TC pp.299-: ex.12.3-5
- TC pp.303-: prob.12-1
- TC pp.311-: ex.13.1-5, 13.1-6, 13.1-7
- TC pp.313-: ex.13.2-2
- TC pp.322-: ex.13.3-1, 13.3-5
- TC pp.330-: ex. 13.4-1, 13.4-2, 13.4-7