9.2 动态查找表

9.2.1 二叉排序树

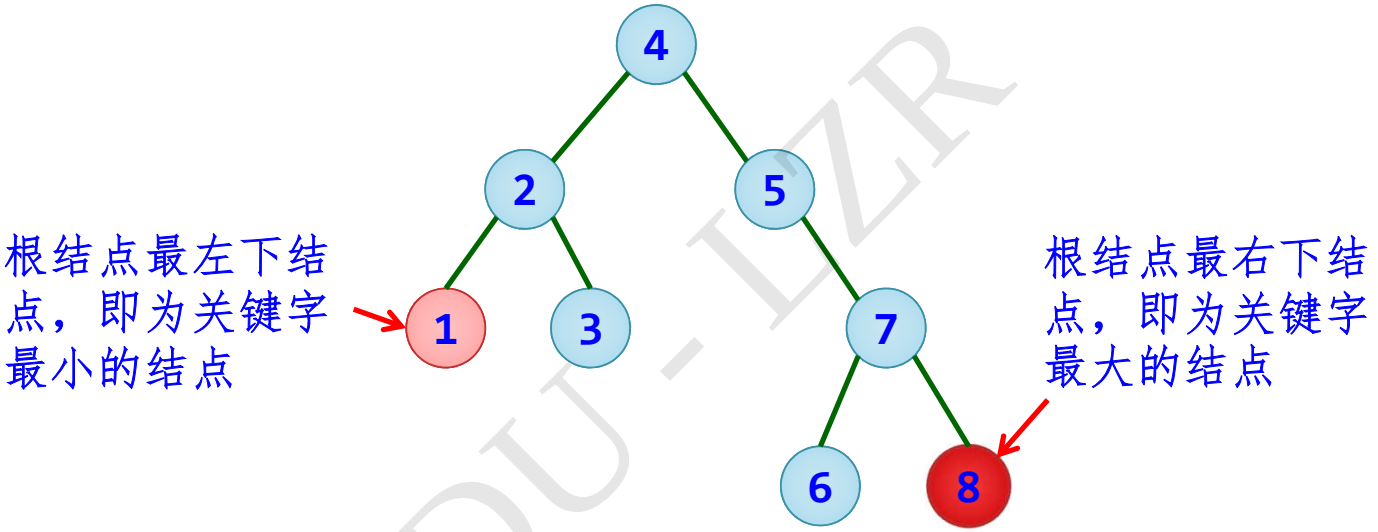
1. 二叉排序树的定义

二叉排序树（二叉搜索树或二叉查找树，BST）或者是一棵空树；或者是具有如下特性的二叉树：

- 1．若它的左子树不空，则左子树上所有结点的值均小于根结点的值；
- 2．若它的右子树不空，则右子树上所有结点的值均大于根结点的值；
- 3．它的左、右子树也都分别是二叉排序树。

注意：二叉排序树中没有相同关键字的结点。

一棵二叉排序树示例：



特点：

中序序列：1，2，3，4，5，6，7，8

中序序列是一个递增有序序列！

二叉排序树的二叉链存储结构的结点的类型声明如下：

```
//----- 数据元素类型的定义 -----
typedef  struct {
    KeyType  key ;              /* 关键字码  */
    otherinfo                   /* 其他域  */
} RecType ;

// ----- 二叉排序树的二叉链表存储表示 ------
typedef struct {
    RecType  data;
    BiTNode  *lchild, *rchild;  // 左右孩子指针
} BiTNode, *BiTree;
```

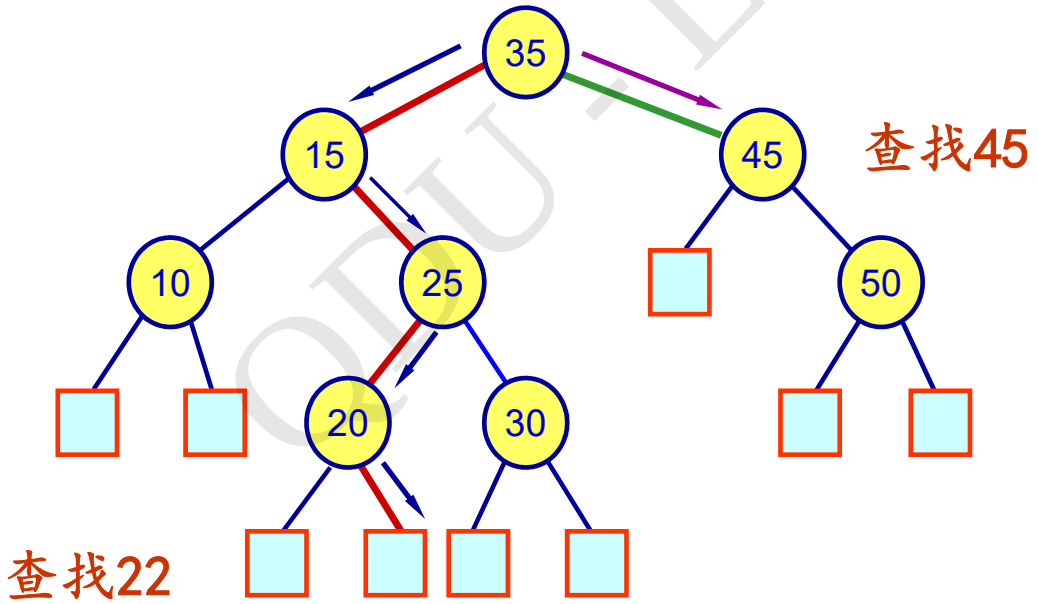
2. 二叉排序树基本算法实现

（1）查找算法

在二叉排序树T中查找关键字为*k*的结点，找到后返回该结点的指针，找不到返回NULL。

- 若当前结点p的关键字等于*k*，则返回p。
- 否则若*k*小于当前结点的关键字，则在左子树中查找。
- 否则若*k*大于当前结点的关键字，则在右子树中查找。

- **查找过程**。其中内结点是二叉排序树中原有结点，外部结点是失败结点，代表树中没有的数据。
- 查找不成功时检测指针停留在某个失败结点。



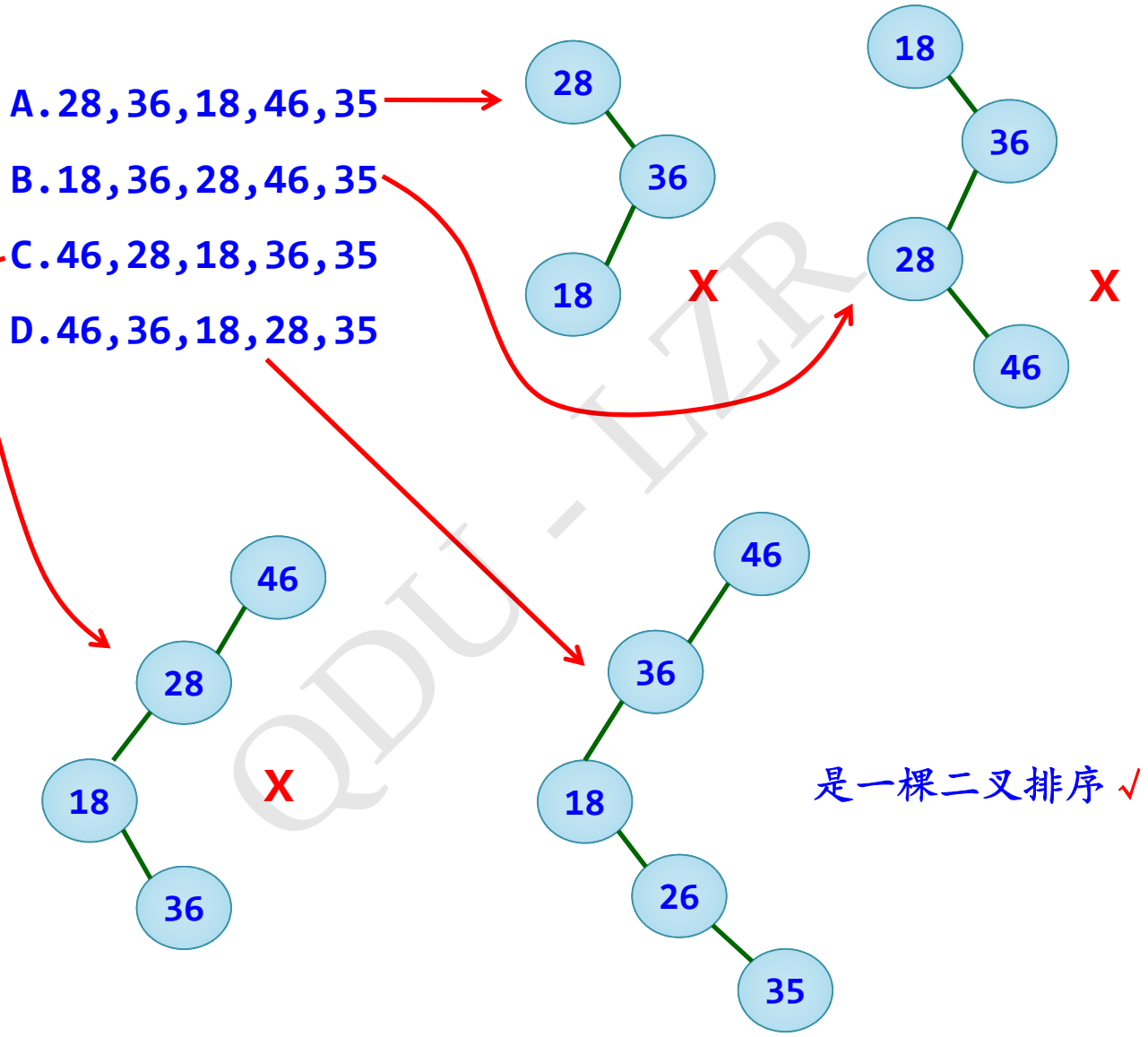
```
BiTree SearchBST(BiTree T, KeyType key)
{
    // 在根指针T所指二叉排序树中递归地查找某关键字等于key的数据元素，
    // 若查找成功，则返回指向该数据元素结点的指针，否则返回空指针。算法9.5(a)
    if(!T || EQ(key, T->data.key))
        return T;                          // 查找结束
    else
        if LT(key, T->data.key)            // 在左子树中继续查找
            return SearchBST(T->lchild, key);
        else
            return SearchBST(T->rchild, key); // 在右子树中继续查找
}
```

【示例-1】在含有27个结点的二叉排序树上，查找关键字为35的结点，以下哪些是可能的关键字比较序列？

A.28,36,18,46,35　　B.18,36,28,46,35

C.46,28,18,36,35　　D.46,36,18,28,35

判断标准：在二叉排序树中的查找路径是原来二叉排序树的一部分，也一定构成一棵二叉排序树。


A.28,36,18,46,35
B.18,36,28,46,35
C.46,28,18,36,35
D.46,36,18,28,35
28
36
18
X
18
36
28
46
X
46
28
18
36
X
46
36
18
26
35
是一棵二叉排序 √

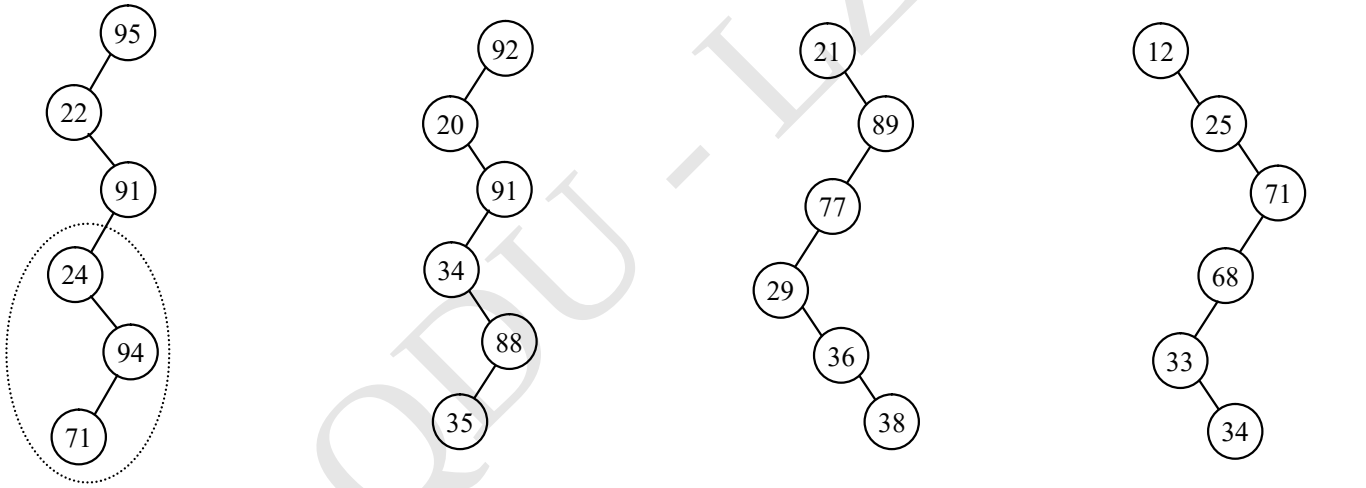
【示例-2】对于下列关键字序列，不可能构成某二叉排序树中一条查找路径的序列是___。

A. 95,22,91,24,94,71　　B. 92,20,91,34,88,35

C. 21,89,77,29,36,38　　D. 12,25,71,68,33,34

说明：本题为2011年全国考研题。

解：各选项对应的查找过程如下图所示，从中看到只有选项A对应的查找树不构成一棵二叉排序树，图中虚线圆框内部分表示违背二叉排序树的性质的子树。本题答案为A。



（a）选项 A 对应的查找过程　（b）选项 B 对应的查找过程　（c）选项 C 对应的查找过程　（d）选项 D 对应的查找过程

（2）插入结点算法

- 二叉排序树是一种动态查找树。其特点是，树的结构通常不是一次生成的，而是在查找过程中，当树中不存在关键字等于给定值的结点时再进行插入。新插入的结点一定是一个新添加的叶子结点，并且是查找不成功时查找路径上访问的最后一个结点的左孩子或右孩子结点。
- 为了实现插入操作，需将查找算法9.5(a)改写成算法9.5(b)，以便能在查找不成功时返回插入位置。
- 插入算法如算法9.6所示。

（2）插入结点算法

- 根据动态查找表的定义，“插入”操作在查找不成功时才进行；
- 若二叉排序树为空树，则新插入的结点为新的根结点；
- 否则，新插入的结点必为一个新的叶子结点，其插入位置由查找算法SearchBST()得到。

```
Status SearchBST(BiTree &T, KeyType key, BiTree f, BiTree &p)
{
    //算法9.5(b)。在根指针T所指二叉排序树中递归地查找其关键字等于key的数据元素，
    //若查找成功，则指针p指向该数据元素结点，并返回TRUE，否则指针p指向查找路径上
    // 访问的最后一个结点并返回FALSE，指针f指向T的双亲，其初始调用值为NULL
    if(!T) {                // 查找不成功
        p = f;
        return FALSE;
    }
    else
        if EQ(key, T->data.key) {       //查找成功
            p = T;
            return TRUE;
        }
        else
            if LT(key, T->data.key)
                return SearchBST(T->lchild, key, T, p); //在左子树中继续查找
            else
                return SearchBST(T->rchild, key, T, p); //在右子树中继续查找
}
```

```
Status InsertBST(BiTree &T, RecType e)
{
    // 当二叉排序树T中不存在关键字等于e.key的元素时，插入e并返回TRUE，
    // 否则返回FALSE。算法9.6
    BiTree p, s;
    if(!SearchBST(T, e.key, NULL, p)) {          // 查找不成功
        s = (BiTree)malloc(sizeof(BiTNode));
        s->data = e;
        s->lchild = s->rchild = NULL;
        if(!p)
            T = s;                               // 被插结点*s为新的根结点
        else
            if LT(e.key, p->data.key)
                p->lchild = s;                   // 被插结点*s为左孩子
            else
                p->rchild = s;                   // 被插结点*s为右孩子
        return TRUE;
    }
    else
        return FALSE;                // 树中已有关键字相同的结点，不再插入
}
```

（3）创建二叉排序树算法

由包含*n*个关键字的数组*a*建立相应的二叉排序树。依次扫描*a*数组的所有元素，调用InsertBST()将其插入到二叉排序树T中。

```
void CreateBST(BiTNode *&T, KeyType a[], int n)
{
    T = NULL;            //初始时T为空树
    int i = 0;
    while(i < n) {
        InsertBST(T, a[i]); //将关键字a[i]插入到二叉排序树T中
        i++;
    }
}
```

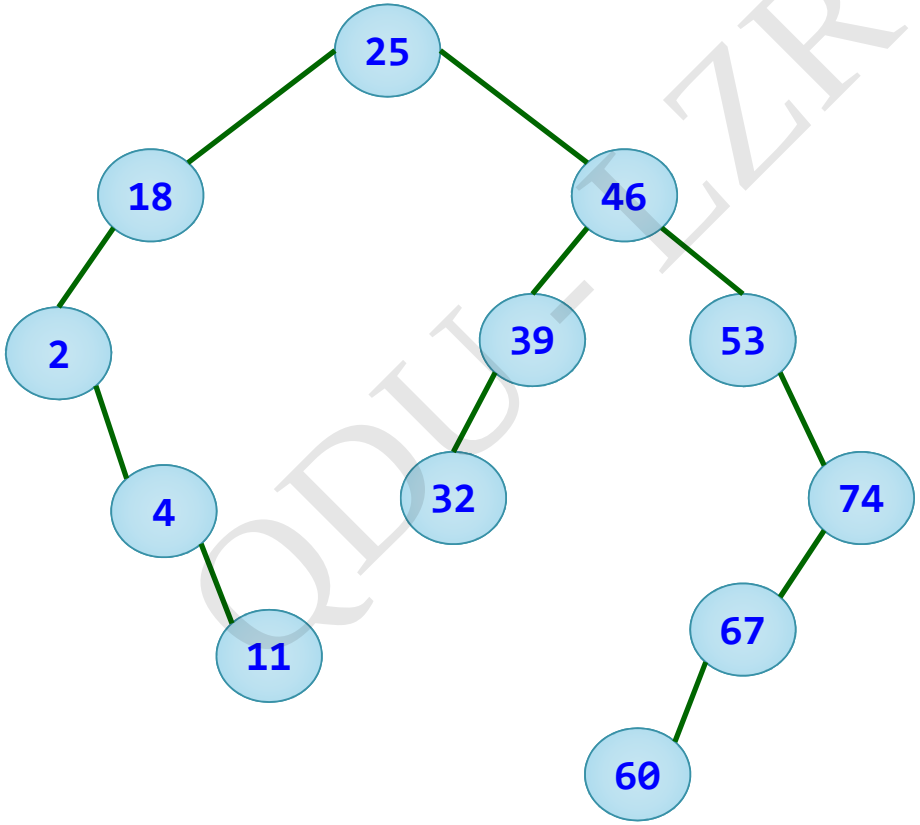
【示例-3】 已知一组关键字为

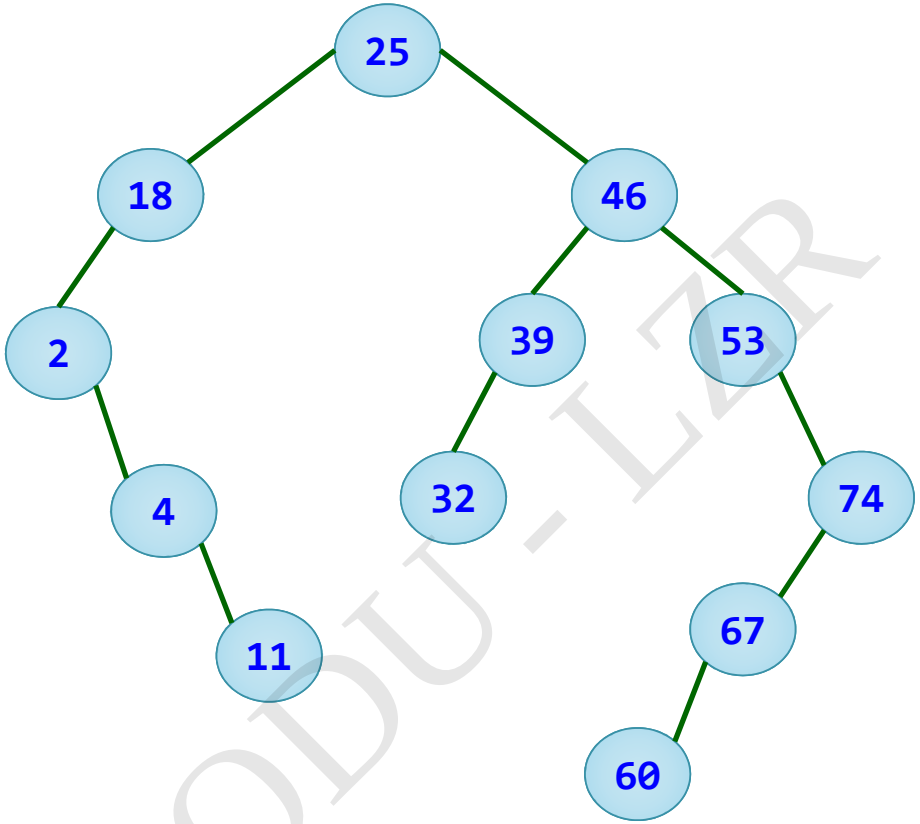
（25,18,46,2,53,39,32,4,74,67,60,11）

按表中的元素顺序依次插入到一棵初始为空的二叉排序树中，画出该二叉排序树，并求在等概率的情况下查找成功的平均查找长度和查找不成功的平均查找长度。

解：生成的二叉排序树如图所示。

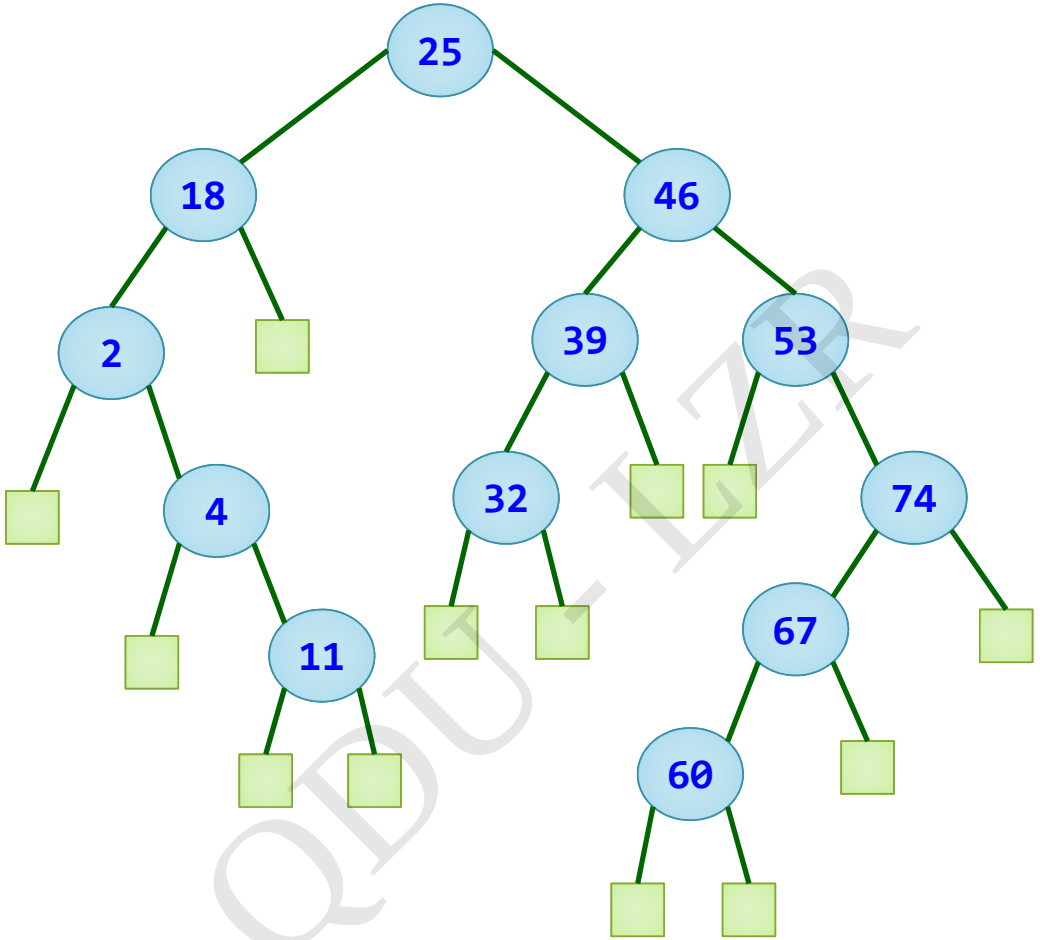
25　18　46　2　53　39　32　4　74　67　60　11





在等概率的情况下，查找成功的平均查找长度为：

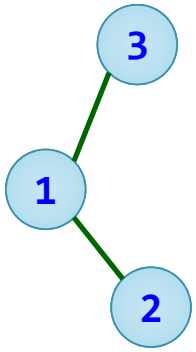
$$ASL_{succ} = \frac{1\times1+2\times2+3\times3+3\times4+2\times5+1\times6}{12} = 3.5$$



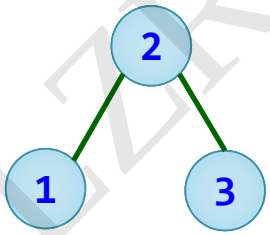
在等概率的情况下，查找不成功的平均查找长度为：

$$ASL_{unsucc} = \frac{1\times 2 + 3\times 3 + 4\times 4 + 3\times 5 + 2\times 6}{13} = 4.15$$

由*n*个关键字构成的二叉排序树可能有多棵，如*n*=3：



$ASL_{succ}=(1+2+3)/3=2$



$ASL_{succ}=(1+2*2)/3=1.67$

- 平均查找长度和树的形态有关。

- 因此，含有n个结点的二叉排序树的平均查找长度和树的形态有关。当先后插入的关键字有序时，构成的二叉排序树蜕变为单支树。树的深度为n，其平均查找长度为(n+1)/2(和顺序查找相同），这是最差的情况。
- 显然，最好的情况是二叉排序树的形态和折半查找的判定树相同，其平均查找长度和$\log_2 n$成正比。
- 在随机的情况下，二叉排序树的平均查找长度和$\log_2 n$是等数量级的，然而，这种情况出现的概率约为53.5%。

二叉排序树小结:

1. 中序遍历二叉排序树可得到关键字有序序列。

2. 在构造二叉排序树时，每次插入的新结点都是新的叶子结点，则进行插入时，不必移动其它结点。

3. 二叉排序树不但拥有类似于折半查找的特性，又采用了链表作存储结构，因此是动态查找表的一种适宜表示。

END