**分析线索二叉树的前驱、后继求法**

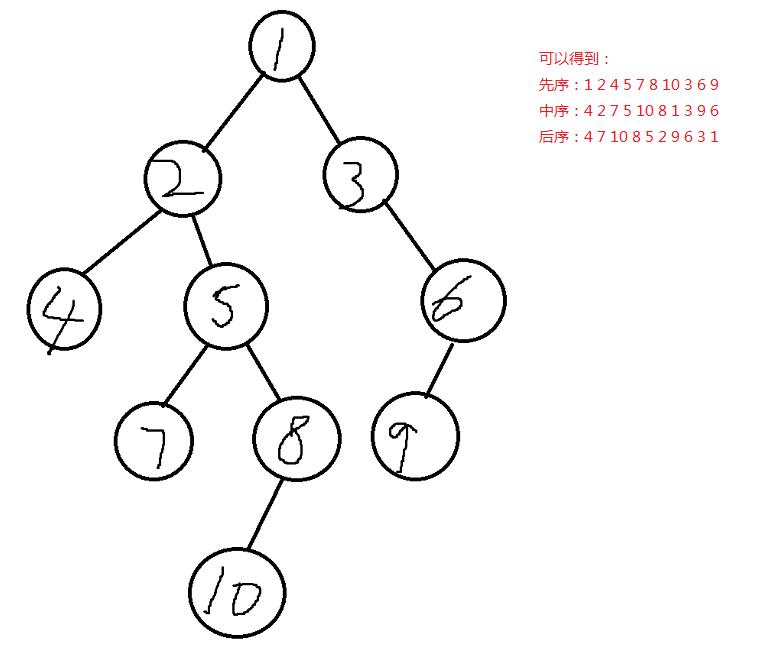


图1

如图可以得到该二叉树的

先序：1 2 4 5 7 8 10 3 6 9

中序：4 2 7 5 10 8 1 3 9 6

后序：4 7 10 8 5 2 9 6 3 1

给出该线索二叉树的结构体：

typedef int ElemType;

// 线索二叉树结构体

typedef struct Clue\_binary\_tree\_node

{

ElemType data; // 数据域

struct Clue\_binary\_tree\_node \*l\_child, \*r\_child; // 左孩子、右孩子

int l\_tag = 0, r\_tag = 0; // 标记

} Clue\_binary\_tree\_node, \*Clue\_binary\_tree;

**先来分析先序的情况：**

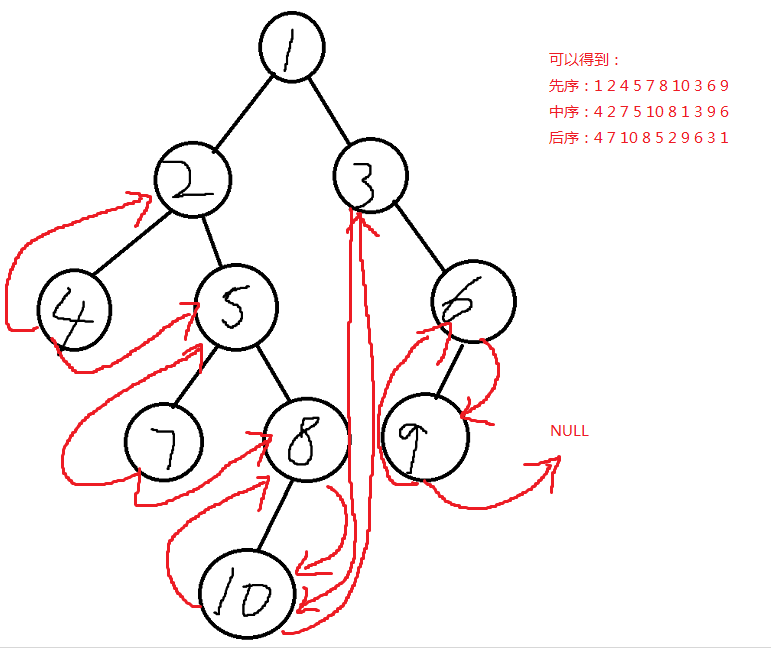


图2

（1）最简单的一种情况就是，它是叶子结点（如4、7、10），即无左右孩子，则，它的l\_tag及r\_tag全为1（默认为0是代表它有左右孩子），此时求**前驱和后继**只需要返回它的左右孩子即可，即：

if (1 == bt -> l\_tag) {

return bt -> l\_child;

} （后继同理）

（2）该结点有一个孩子（如8、3、6）

**在考虑线索的情况下**，先判断它的孩子是左孩子还是右孩子，如果是左孩子，那么它的**后继**，可以直接根据r\_child求出，如果是右孩子，那么它的前驱，可以直接根据l\_child求出。剩下另一半（前驱或后继）可以根据不考虑线索的情况下的求法求出。

**在不考虑线索的情况下**，先说**后继**，因为此时是**先序（先根后左再右）**，则无论该结点是有左孩子还是右孩子，它的后继都是它的孩子。再说**前驱**，同样因为是**先序（先根后左再右）**，如果它位于左孩子的位置，则它的顺序就应该在它的父亲结点之后，即它的前驱就是它的父亲结点；如果它位于右孩子的位置，则它的顺序就应该在它的左兄弟结点之后（若存在左兄弟，否则就是它的父亲结点），因为它的左兄弟结点可能有多个子孙，同样满足这个情况，故，该结点应该位于**左兄弟的子孙中（如果没有子孙就是左兄弟）最下且最靠右的结点的后面**，例如：

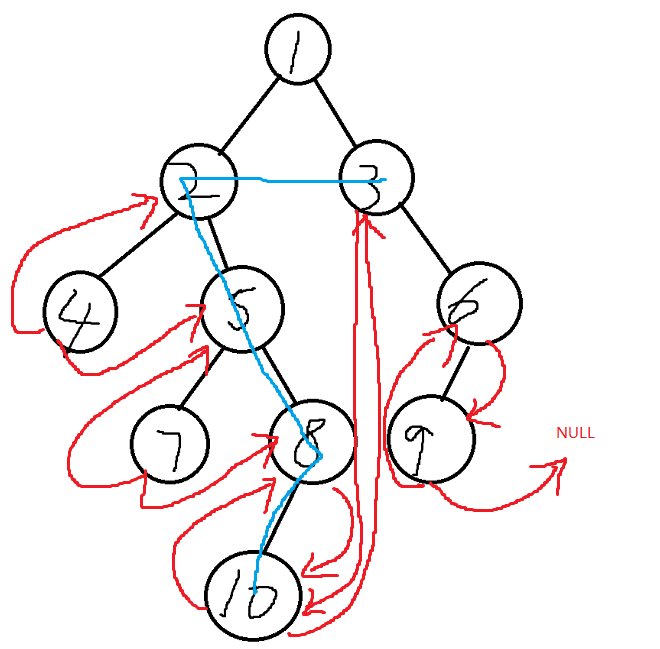
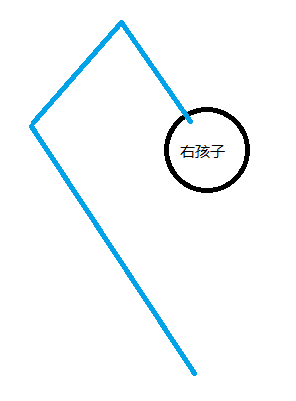


图3

如图中结点3的前驱就是它兄弟结点2的子孙中最下面且最靠右的结点10，同理结点6的前驱是结点3，因为结点6没有左兄弟，故它的前驱就是它的父亲结点。结点8的前驱是结点7，因为它的左兄弟结点7没有子孙，故它的前驱就是它的左兄弟。

满足如下图的样式

图4

图中蓝色的线最多连到哪，它的前驱就到哪（最后可能是左孩子也可能是右孩子）。

（3）该结点有2个孩子（如结点2、5），发现，它的**前驱**，同样满足上述规则，当它为左孩子时，它的前驱就是它的父亲结点，当它是右孩子时，它的前驱就是它的左兄弟子孙中最靠下靠右的结点。它的**后继**，更简单，就是它的左孩子。

**总结：当线索二叉树是先序线索二叉树时，要你求某一个结点的前驱时，分两种大情况：**

1. **它的(l\_tag == 1)，直接返回 l\_child 即可**
2. **它的(l\_tag != 1)，沿着图4的轨迹找即可**

**要你求某一个结点的后继时，分两大种情况：**

1. **它的(r\_tag == 1)，直接返回 r\_child 即可**
2. **它的(r\_tag != 1)，有左孩子则是左孩子，没有左孩子则是右孩子。**

**具体代码实现如下：**

bool find\_node\_parent(Clue\_binary\_tree target, Clue\_binary\_tree root, Clue\_binary\_tree &temp);

Clue\_binary\_tree get\_pre\_node(Clue\_binary\_tree bt, Clue\_binary\_tree root);

Clue\_binary\_tree get\_suc\_node(Clue\_binary\_tree bt);

// 求前驱

Clue\_binary\_tree get\_pre\_node(Clue\_binary\_tree bt, Clue\_binary\_tree root) {

// 如果该结点为空，则返回空

if (NULL == bt) {

return NULL;

}

// 如果该节点的左孩子存储的就是它的前驱，则返回左孩子

if (1 == bt -> l\_tag) {

return bt -> l\_child;

}

Clue\_binary\_tree temp = NULL;

// 如果找不到该节点的父亲节点，则该节点为根，根在先序遍历中是不可能有前驱

if (!find\_node\_parent(bt, root, temp)) {

return NULL;

}

// 此时temp即为bt的父亲结点，如果它的左孩子为空，则它就是该结点的前驱

if (!temp -> l\_child) {

return temp;

} else {

temp = temp -> l\_child

while(temp -> r\_child) {

temp = temp -> r\_child;

}

if (temp -> l\_child) return temp -> l\_child;

else return temp;

}

}

// 求后继

Clue\_binary\_tree get\_suc\_node(Clue\_binary\_tree bt) {

if (NULL == bt) {

return NULL;

}

if (1 == bt -> r\_tag) {

return bt -> r\_child;

}

if (bt -> l\_child) {

return bt -> l\_child;

} else {

return bt -> r\_child;

}

}

bool find\_node\_parent(Clue\_binary\_tree target, Clue\_binary\_tree root, Clue\_binary\_tree &temp) {

if ((root -> l\_child == target) || (root -> r\_child == target)) {

temp = root;

return true;

} else {

if (find\_node\_parent(target, root -> l\_child, temp)) return true;

if (find\_node\_parent(target, root -> r\_child, temp)) return true;

}

}

**再来分析中序的情况 ：**

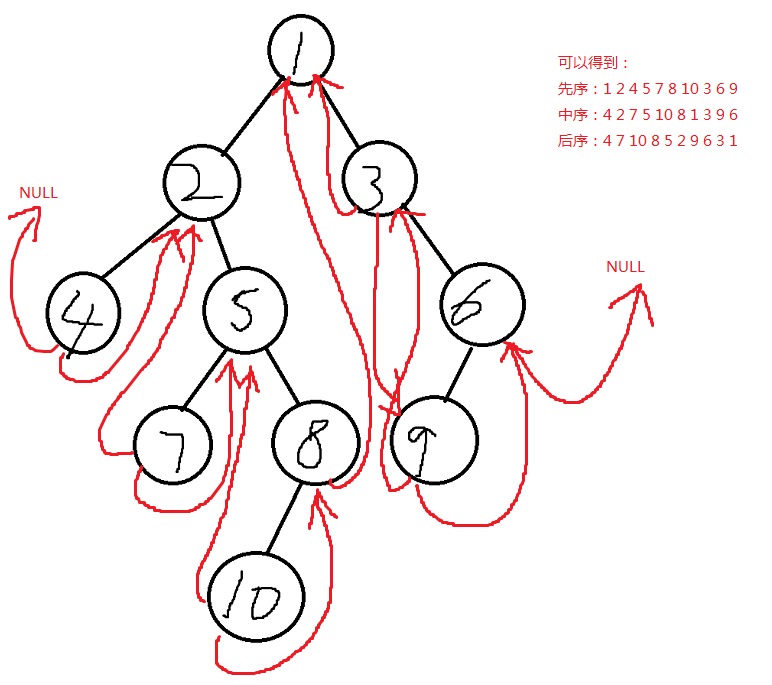
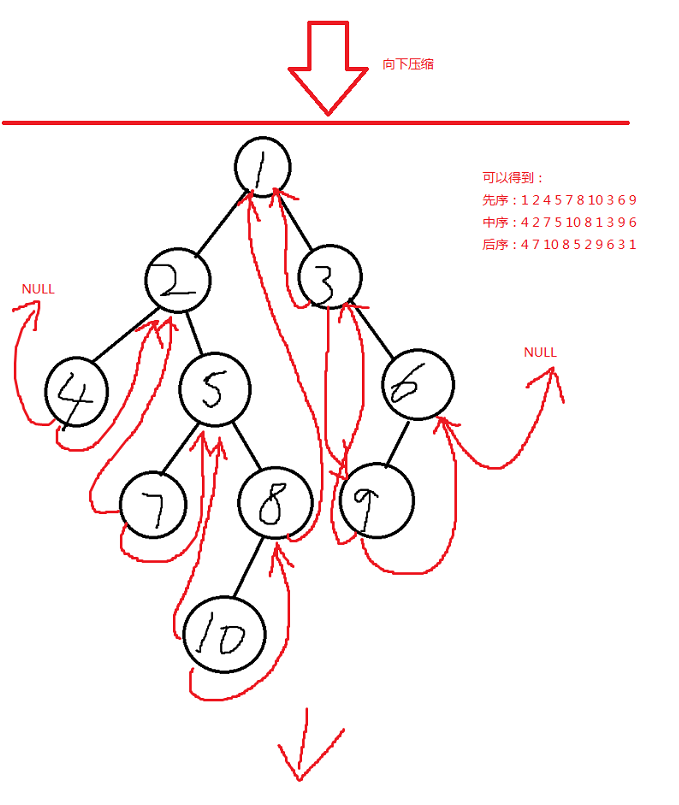
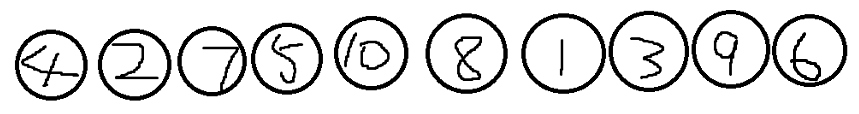
****

图5

如图5是中序线索二叉树

**如果不用程序来求中序，简便方法：**





正好就是它的中序序列。

下面还是分3种情况来分析：

1. 叶子结点：因为它的左右孩子就是存储的它中序前驱后继的信息，故直接返回左右孩子即可得到该结点的前驱后继。
2. 只有1个孩子的结点：先来分析如果它只有**右孩子**，因为**中序（先左后根再右）**，若该结点只有右孩子，那么它的**前驱**一定是它的父亲结点（信息就存储在它的左孩子），但是它的**后继**，则不单纯是它的右孩子，应该是它右子孙中**最靠左的那个结点**；如果它只有左孩子，那么它的**前驱**则是它左子孙中**最靠右的那个结点**，它的后继的信息则存储在它的右孩子中。
3. 有2个孩子的结点：它的**前驱**是它的左子孙中**最靠右的那个结点**，它的**后继**是它的右子孙中**最靠左的那个结点**。

**总结：当线索二叉树是中序线索二叉树时，要你求某一个结点的前驱时，分两种大情况：**

1. **它的 (1 == l\_tag)时，直接return l\_child。**
2. **它的 (0 == l\_tag)时，找它的左子孙中最靠右的那个结点。**

**要你求某一个结点的后继时，分两大种情况：**

1. **它的 (1 == r\_tag)时，直接return r\_child**
2. **它的 (0 == r\_tag)时，找它的右子孙中最靠左的那个结点。**

**具体代码实现如下：**

// 求前驱

Clue\_binary\_tree get\_pre\_node(Clue\_binary\_tree bt) {

if (NULL == bt) {

return NULL;

}

if (1 == bt -> l\_tag) {

return bt -> l\_child;

}

Clue\_binary\_tree temp = bt -> l\_child;

while (temp -> r\_child) {

temp = temp -> r\_child;

}

return temp;

}

// 求后继

Clue\_binary\_tree get\_suc\_node(Clue\_binary\_tree bt) {

if (NULL == bt) {

return NULL;

}

if (1 == bt -> r\_tag) {

return bt -> r\_child;

}

Clue\_binary\_tree temp = bt -> r\_child;

while (temp -> l\_child) {

temp = temp -> l\_child;

}

return temp;

}

**最后分析后序的情况 ：**

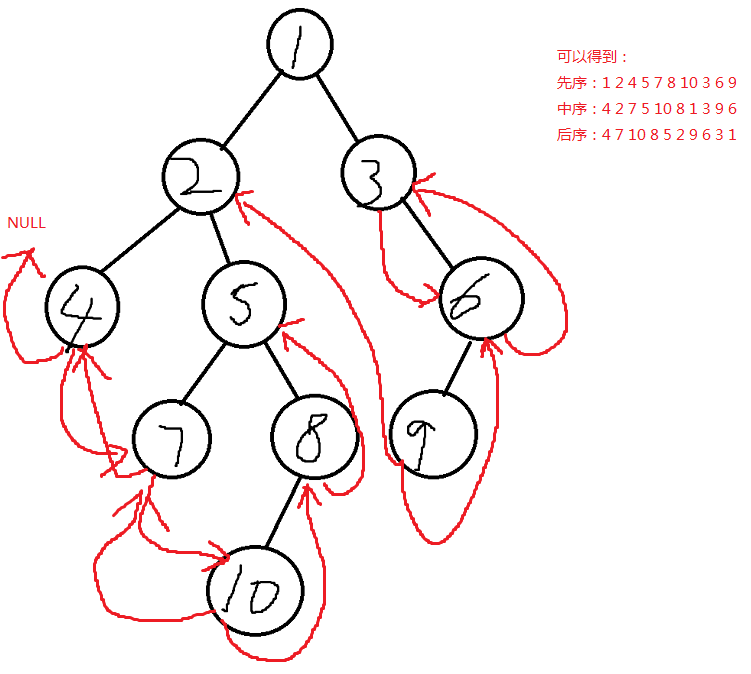
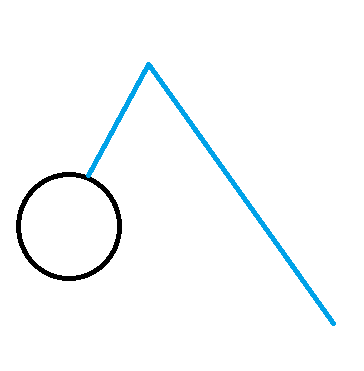
****

图8

还是分3种情况分析：

1. 叶子结点，同上，返回l\_child或r\_child即可
2. 只有1个孩子的结点：由于**后序（先左后右再根）**，它的**前驱**一定是它的那个孩子。它的**后继**，则是它的**右边兄弟的子孙里最靠下靠左的那个结点**，借鉴先序。



（3） 有两个孩子的结点：它的**先驱**即是它的子孙中**最靠下靠右的那个结点**，