二叉树的遍历算法（递归、非递归的先序、中序、后序和层次遍历）

http://blog.sina.com.cn/s/blog_6f611c300102ux55.html

标签： 二叉树 先序 后序 中序 非递归　　分类： 算法

二叉树的递归遍历和非递归遍历

首先定义：

```
#define MaxSize 100
typedef char ElemType;
typedef struct node
{
    ElemType data;
    struct node * lchild;
    struct node * rchild;
}BTNode ;
```

1.先序遍历

递归：

```
void PreOrder(BTNode *b)
{
    if (b!= NULL)
    {
        printf("%c", b->data);
        PreOrder(b->lchild);
        PreOrder(b->rchild);
    }
}
```

非递归：

借助一个栈，因为每次都是栈顶出栈，即栈顶都是先访问的节点，先序遍历的思想是先根，再左孩子，再右孩子。

故访问完当前节点后，应该先将右孩子入栈，再左孩子入栈即可。

```
void PreOrder1(BTNode *b)
{
    BTNode *St[MaxSize], *p;
    int top = -1;
    if (b!=NULL)
    {
        top++;
        St[top] = b;
        while(top > -1)       //the stack is not empty then loop
        {
            p = St[top];
            top--;
            printf("%c", p->data);
            if(p->rchild) St[top++] = p->rchild;
            if(p->lchild) St[top++] = p->lchild;
        }
    }
    printf("\n");
}
```

2.中序遍历

递归:

```
void InOrder(BTNode * b)
{
    if(b!=NULL)
    {
        InOrder(b->lchild);
        printf("%c", b->data);
        InOrder (b->rchild);
    }
}
```

非递归:

中序遍历的思想是先左孩子，再父节点，再右孩子，

故先将所有左孩子节点入栈，再输出最后一个入栈的节点，再访问他的右孩子。

```
void InOrder2(BTNode *b)
{
    BTNode *St[MaxSize], *p;
    int top = -1;
    if(b != NULL)
    {
        p = b;
        while(top> -1 || p!= NULL)
        {
            while(p!= NULL)
            {
                top++;
                St[top] = p;
                p = p->lchild;
            }
            if(top > -1)
            {
                p = St[top];
                top--;
                printf("%c", p->data);
                p = p->rchild;
            }
        }
        printf("\n");
    }
}
```

3.后序遍历

递归：

```
void PostOrder(BTNode* b)
{
    if(b!=NULL)
    {
        PostOrder(b->lchild);
        PostOrder(b->rchild);
        printf("%c", b->data);
    }
}
```

非递归：

后序遍历的思想是先左孩子，再右孩子，再父节点。

故也是先将所有左孩子遍历，这时需判断他是否有右孩子，如果右孩子节点不存在或者右孩子已经访问过了，这时需要一个标记当前节点的前一个访问节点。

```
void PostOrder2(BTNode* b)
{
    BTNode* St[MaxSize], *p;

    int top = -1;
    if(b!= NULL)
    {
            do
            {
                while(b!= NULL)            //send all left child into stack
                {
                    top++;
                    St[top] = b;
                    b = b->lchild;
                }
                p = NULL;                                    //p point to
the previous visited node of current node
                flag = 1;                                    //note the
node of b has been visited
                while(top != -1 && flag)
                {
                    b = St[top];                       //Get the current
node
                    if( b->rchild == p)           // if right node is not
existed or has been visited, then visit the current node
                    {
                        printf("c", b->data);          //vistited current
node
                        top--;
                        p = b;
                    }
                    else
                    {
                        b = b->rchild;          //b point to the right
child
                        flag = 0;                   //set not visited
                    }
                }
            }while( top != -1)
    }
}
```

4.层次遍历

```
void TravLevel(BTNode* b)
{
    BTNode *Qu[MaxSize];
    int front, rear;
    front = rear = 0;
    if( b!= NULL)
        printf("%c", b->data);
    rear++;
    Q[rear]= b;
    while(rear != front)
    {
        front= (front+ 1)%MaxSize;        //front head come out;
        b = Qu[front];
        if(b->lchild != NULL)                   //print left child, and
enter stack
        {
            printf("%c", b->lchild->data);
            rear = (rear+1)%MaxSize;
            Qu[rear] = b->lchild;
        }
        if(b->rchild != NULL)                   //print right child, and
enter stack
```

```
        {
            printf("%c", b->rchild->data);
            rear = (rear+1)%MaxSize;
            Qu[rear]=b->rchild;
        }
    }
    printf("\n");
}
```