**二叉树递归遍历存在的问题**

如果我们的二叉树只有左子树，而且树的高度还很深的时候，这个时候递归调用遍历的时候，栈帧空间开辟的较大，很可能造成栈溢出。但是我们一个程序中，为堆分配的空间要比栈大的多，这个时候我们可以实现一个二叉树遍历的非递归的实现形式，模拟栈帧调用的过程，自己模拟一个栈用于保存节点，然后实现遍历。

**非递归实现树的遍历**

前序

首先设置一个指针cur指向根节点，访问cur然后让cur入栈，并且是在循环中使cur一直指针它的的左子树，入栈的时候就访问了，当我们出栈的时候就使cur指向它的右子树，这里我们需要两层循环，一层主要是判断当前的栈是否为空的，一层主要是判断一直往左走的时候什么时候到达了左子树的尽头了。

void PrevNoROeder()

{

stack<Node\*> **sn***;*

Node\* cur = \_root*;*

**while** (cur **||** !**sn**.empty())

{

**while** (cur)

{

cout << cur->\_**data** << " "*;*

**sn**.**push(cur);**

cur = cur->\_left*;*

}

cur = (**sn**.top())->\_right*;*

**sn**.**pop();**

}

cout << endl*;*

}

其实可以这样子理解的，我们的前序实际上是把所有的问题都转化成了访问了左子树，即使是访问右子树的时候，也好像是访问左子树一样，就是把这个树当成了一个树根

中序

同样是上面的内容，这个时候也是可以使用cur内容的，但是这个时候我们不是先访问，而是先进栈，当我们出栈的时候，我们在访问，然后出栈，出栈的时候把出来的这个节点的右子树当成是一个cur，这个时候还是一直当成是一个左子树来访问了

void InNoROrder()

{

stack<Node\*> **sn***;*

Node\* cur = \_root*;*

**while** (cur **||** !**sn**.empty())

{

**while** (cur)

{

**sn**.**push(cur);**

cur = cur->\_left*;*

}

cout << (**sn**.top())->\_**data**<<" "*;*

cur = (**sn**.top())->\_right*;*

**sn**.**pop();**

}

cout << endl*;*

}

后序

首先让cur指向的是一个root，然后这个时候让cur一直往左走，当走到NULL的时候，我们想要出栈但是还不能出栈，因为我们还不知道当前节点的右子树是不是已经访问了呢。所以我们加上了一层判断，就是当我们的当前节点的右子树不是空的时候，并且当前节点的右子树不是刚刚出栈的节点的时候，我们就需要把当前节点的右子树。这里因为需要判断当前节点的右子树是不是刚刚出栈的节点，所以我们每次出栈的时候都需要标记一下当前出栈的节点，以便后来的比较使用。如果不是之前访问的节点，我们就可以出栈访问了。

void EndNoROrder()

{

stack<Node\*> **sn***;*

Node\* cur = \_root*;*

Node\* prev = \_root*;*

**while** (cur **||** !**sn**.empty())

{

**if** (cur) //让所有的左子树入栈

{

**sn**.**push(cur);**

cur = cur->\_left*;*

}

**else** //所有的左子树都入栈了

{

**if** (((**sn**.top())->\_right != NULL) && ((**sn**.top())->\_right != prev))

{

cur = (**sn**.top())->\_right*;*

}

**else** //所有的左右子树都访问完毕了

{

cout << (**sn**.top())->\_**data**<<" "*;*

prev = **sn**.top()*;*

**sn**.**pop();**

}

}

}

cout << endl*;*

}

通过上面的非递归三种方式的比较我们可以发现，三个的一个共同点是，一开始都是让所有的左子树节点先入栈，其中先序遍历的时候，入栈的时候就直接访问了，中序遍历的时候是出栈的时候才访问的，还有一个点就是，出栈的时候让cur指向栈顶的右子树，先序遍历和中序遍历是直接入栈的，但是后序遍历是需要一层判断，就是如果右子树**不是**刚刚访问过的，说明当前右子树还没有访问过，这个时候就需要访问右子树，如果是刚刚访问过的，这个时候就可以直接让cur出栈访问就好了，就是我们在后序遍历的时候，在即将出栈的时候需要加上一层判断就是当前的右子树是否已经被访问过了，如果没有被访问过，我们就需要是当前的右子树入栈如果访问过了，我们就可以使当前的top出栈了访问他了