二叉树的遍历包括先序遍历，中序遍历，后序遍历，层次遍历等等。本文对此进行整理。

二叉树结构定义如下：

**[cpp]** [view plain](http://blog.csdn.net/luckyxiaoqiang/article/details/8701926) [copy](http://blog.csdn.net/luckyxiaoqiang/article/details/8701926)

1. //Definition for binary tree
2. **struct** TreeNode {
3. **int** val;
4. TreeNode \*left;
5. TreeNode \*right;
6. TreeNode(**int** x) : val(x), left(NULL), right(NULL) {}
7. };

## 1. 先序遍历

先序遍历就是先访问根节点，然后再先序遍历左子树，最后先序遍历右子树。先序遍历也就是深度优先搜索（DFS）。

### 先序遍历递归实现：

**[cpp]** [view plain](http://blog.csdn.net/luckyxiaoqiang/article/details/8701926) [copy](http://blog.csdn.net/luckyxiaoqiang/article/details/8701926)

1. vector<**int**> preorderTraversal(TreeNode \*root)
2. {
3. vector<**int**> vals;
4. preorderTraversal(root, vals);
5. **return** vals;
6. }
7. **void** preorderTraversal(TreeNode \*root, vector<**int**> &vals)
8. {
9. **if**(root == NULL) **return**;
10. vals.push\_back(root->val);
11. preorderTraversal(root->left, vals);
12. preorderTraversal(root->right, vals);
13. }

### 先序遍历非递归实现：

这个代码好：

**void PreOrder\_Nonrecursive(BiTree T)** //先序遍历的非递归

{

if(!T)

return ;

stack<BiTree> s;

s.push(T);

while(!s.empty())

{

BiTree temp = s.top();

cout<<temp->data<<" ";

s.pop();

if(temp->rchild)

s.push(temp->rchild);

if(temp->lchild)

s.push(temp->lchild);

}

}

**[cpp]** [view plain](http://blog.csdn.net/luckyxiaoqiang/article/details/8701926) [copy](http://blog.csdn.net/luckyxiaoqiang/article/details/8701926)

1. vector<**int**> preorderTraversal(TreeNode \* root)
2. {
3. vector<**int**> vals;
4. stack<TreeNode\*> s;
5. TreeNode \* p = root;
6. **while**(!s.empty() || p)
7. {
8. **if**(p == NULL)
9. {
10. **while**(!s.empty() && (p == s.top()->right || s.top()->right == NULL))
11. {   // 右子树已访问，出栈
12. p = s.top();
13. s.pop();
14. }
15. **if**(s.empty()) **break**;
16. // 左子树已访问，右子树尚未访问，访问右子树
17. p = s.top()->right;
18. }
19. **else**
20. {
21. vals.push\_back(p->val);
22. s.push(p);
23. p = p->left;
24. }
25. }
26. **return** vals;
27. }

## 2. 中序遍历

中序遍历就是先中序遍历左子树，然后访问根节点，最后中序遍历右子树。

### 中序遍历递归实现：

这个代码好

void InOrderTraverse(BiTree T) // 中序遍历的非递归

{

if(!T)

return ;

stack<BiTree> s;

BiTree curr = T->lchild; // 指向当前要检查的节点

s.push(T);

while(curr != NULL || !s.empty())

{

while(curr != NULL) // 一直向左走

{

s.push(curr);

curr = curr->lchild;

}

curr = s.top();

s.pop();

cout<<curr->data<<" ";

curr = curr->rchild;

}

}

**[cpp]** [view plain](http://blog.csdn.net/luckyxiaoqiang/article/details/8701926) [copy](http://blog.csdn.net/luckyxiaoqiang/article/details/8701926)

1. vector<**int**> inorderTraversal(TreeNode \*root)
2. {
3. vector<**int**> vals;
4. inorderTraversal(root, vals);
5. **return** vals;
6. }
7. **void** inorderTraversal(TreeNode \*root, vector<**int**> &vals)
8. {
9. **if**(root == NULL) **return**;
10. inorderTraversal(root->left, vals);
11. vals.push\_back(root->val);
12. inorderTraversal(root->right, vals);
13. }

### 中序遍历非递归实现：

**[cpp]** [view plain](http://blog.csdn.net/luckyxiaoqiang/article/details/8701926) [copy](http://blog.csdn.net/luckyxiaoqiang/article/details/8701926)

1. vector<**int**> inorderTraversal(TreeNode \*root) {
2. vector<**int**> vals;
3. stack<TreeNode \*> s;
4. TreeNode \*p = root;
5. **while**(!s.empty() || p)
6. {
7. **while**(p)
8. {
9. s.push(p);
10. p = p->left;
11. }
12. p = s.top();
13. s.pop();
14. vals.push\_back(p->val);
15. p = p->right;
16. }
17. **return** vals;
18. }

## 3. 后序遍历

后序遍历就是先后序遍历左子树，再后序遍历右子树，最后访问根节点。  
  
**后序遍历递归实现：**

**[cpp]** [view plain](http://blog.csdn.net/luckyxiaoqiang/article/details/8701926) [copy](http://blog.csdn.net/luckyxiaoqiang/article/details/8701926)

1. vector<**int**> postorderTraversal(TreeNode\* root)
2. {
3. vector<**int**> vals;
4. postorderTraversal(root, vals);
5. **return** vals;
6. }
7. **void** postorderTraversal(TreeNode \*root, vector<**int**> &vals)
8. {
9. **if**(root == NULL)
10. **return**;
11. postorderTraversal(root->left, vals);
12. postorderTraversal(root->right, vals);
13. vals.push\_back(root->val);
14. }

### 后序遍历非递归实现：

使用两个栈

void InOrderTraverse(BiTree T) // 中序遍历的非递归

{

if(!T)

return ;

stack<BiTree> s;

BiTree curr = T->lchild; // 指向当前要检查的节点

s.push(T);

while(curr != NULL || !s.empty())

{

while(curr != NULL) // 一直向左走

{

s.push(curr);

curr = curr->lchild;

}

curr = s.top();

s.pop();

cout<<curr->data<<" ";

curr = curr->rchild;

}

}

1. vector<**int**> postorderTraversal(TreeNode\* root)
2. {
3. vector<**int**> vals;
4. stack<TreeNode\*> s;
5. TreeNode \* pre = NULL;
6. TreeNode \* p = root;
7. **while**(!s.empty() || p)
8. {
9. **while**(p)
10. {
11. s.push(p);
12. p = p->left;
13. }
14. pre = NULL;
15. **while**(!s.empty() && pre == s.top()->right)
16. {
17. pre = s.top();
18. vals.push\_back(s.top()->val);
19. s.pop();
20. }
21. **if**(s.empty()) **break**;
22. p = s.top()->right;
23. }
24. **return** vals;
25. }

## 4. 层次遍历

层次遍历就是先遍历第一层节点，然后遍历第二层节点，。。。也就是广度优先搜索（BFS）。

### 队列实现：

**[cpp]** [view plain](http://blog.csdn.net/luckyxiaoqiang/article/details/8701926) [copy](http://blog.csdn.net/luckyxiaoqiang/article/details/8701926)

1. vector<**int**> levelOrderTraversal(TreeNode \*root)
2. {
3. vector<**int**> vals;
4. **if**(root == NULL)
5. **return** vals;
6. queue<TreeNode\*> q;
7. q.push(root);
8. **while**(!q.empty())
9. {
10. TreeNode \* p = q.front();
11. q.pop();
12. vals.push\_back(p->val);
13. **if**(p->left)
14. q.push(p->left);
15. **if**(p->right)
16. q.push(p->right);
17. }
18. **return** vals;
19. }

### 层次遍历递归实现：

**[cpp]** [view plain](http://blog.csdn.net/luckyxiaoqiang/article/details/8701926) [copy](http://blog.csdn.net/luckyxiaoqiang/article/details/8701926)

1. vector<**int**> levelOrderTraversal(TreeNode \*root)
2. {
3. vector<**int**> vals;
4. height = getTreeHeight(root);
5. **for**(**int** level = 1; level <= height; level++)
6. levelOrderTraversal(root, level, vals);
7. **return** vals;
8. }
9. **int** getTreeHeight(TreeNode \* root)
10. {
11. **if**(root == NULL) **return** 0;
12. **int** heightLeft = getTreeHeight(root->left);
13. **int** heightRight = getTreeHeight(root->right);
14. **return** max(heightLeft, heightRight)+1;
15. }
16. **void** levelOrderTraversal(TreeNdoe \*root, **int** level, vector<**int**> &vals)
17. {
18. **if**(root == NULL) **return**;
19. **if**(level == 1)
20. {
21. vals.push\_back(root->val);
22. **return**;
23. }
24. levelOrderTraversal(root->left, level-1, vals);
25. levelOrderTraversal(root->right, level-1, vals);
26. }