

```

1:  /*-----+
2:  |  leetcode 144 pre_order      |
3:  |  leetcode 145 post_order    |
4:  |  leetcode 94  in_order      |
5:  +-----*/
6:
7:  <PART 1.>
8:  对于二叉树的遍历，递归式遍历很好写出
9:  /*前序：根->左->右*/
10: void _pre_recur(TreeNode *root, vector<int>&ans)
11: {
12:     if (root){
13:         ans.push_back(root->val);
14:         _pre_recur(root->left, ans);
15:         _pre_recur(root->right, ans);
16:     }
17: }
18: /*后序：左->右->根*/
19: void _post_recur(TreeNode *root, vector<int>&ans)
20: {
21:     if (root){
22:         _post_recur(root->left, ans);
23:         ans.push_back(root->val);
24:         _post_recur(root->right, ans);
25:     }
26: }
27: /*中序：左->根->右*/
28: void _in_recur(TreeNode *root, vector<int>&ans)
29: {
30:     if (root){
31:         _in_recur(root->left, ans);
32:         ans.push_back(root->val);
33:         _in_recur(root->right, ans);
34:     }
35: }
36:
37: <PART 2.>
38: 对于迭代式的遍历，则需要借助栈这一结构。首先对于前序遍历而言，每次处理的都是
39: 根节点，然后对其左右子树操作，因此，由于栈的结构原因，入栈的时候先入栈右子树
40: 因此就可以得到前序序列。
41: void _pre_it(TreeNode *root, vector<int>&ans)
42: {
43:     stack<TreeNode*> s;
44:     s.push(root);
45:     while (!s.empty()){
46:         TreeNode *cur=s.top();
47:         s.pop();
48:         if (cur!= nullptr){
49:             ans.push_back(cur->val);
50:             s.push(cur->right);
51:             s.push(cur->left);
52:         }
53:     }
54: }
55: 对于后序遍历，只需入栈的顺序改一下，先入栈左节点，从而ans中便是根->右->左
56: 再将ans反转便是左->右->根
57: void _post_it(TreeNode *root, vector<int>&ans)
58: {
59:     stack<TreeNode*> s;
60:     s.push(root);
61:     while (!s.empty()){
62:         TreeNode *cur=s.top();
63:         s.pop();
64:         if (cur!= nullptr){
65:             ans.push_back(cur->val);
66:             s.push(cur->left);
67:             s.push(cur->right);
68:         }
69:     }
70:     reverse(ans.begin(), ans.end());
71: }
72: 但是对于中序遍历，这个过程有些许的变化，不仅需要栈，还需要指针的遍历。中序
73: 遍历首先需要找到最左边的节点。之后取出这个节点，往右边走。

```

```

74: void _in_it(TreeNode *root, vector<int>&ans)
75: {
76:     stack<TreeNode*>s;
77:     TreeNode *cur=root;
78:     while (cur!= nullptr||!s.empty()){
79:         if(cur!=nullptr){
80:             s.push(cur);
81:             cur=cur->left;
82:         }else{
83:             /*最左边的节点*/
84:             TreeNode *leftest=s.top();
85:             ans.push_back(leftest->val);
86:             s.pop();
87:             cur=leftest->right;
88:         }
89:     }
90: }
91:
92: <PART 3.>
93: 然而，如果没有统一的套路，迭代式的写法就有些许麻烦，可以使用标记法，将迭代
94: 纳入统一的框架。也就是在需要处理的节点后面放入一个空指针进行标记，在碰到空
95: 节点时就统一处理该节点。
96: void _pre_uni(TreeNode *root, vector<int>&ans)
97: {
98:     stack<TreeNode *> s;
99:     if (root) s.push(root);
100:    while (!s.empty()) {
101:        TreeNode *cur = s.top();
102:        if (cur) {
103:            s.pop();
104:            /*右*/
105:            if (cur->right) s.push(cur->right);
106:            /*左*/
107:            if (cur->left) s.push(cur->left);
108:            /*根*/
109:            s.push(cur);
110:            s.push(nullptr);
111:        } else {
112:            /*统一处理*/
113:            s.pop();
114:            cur = s.top();
115:            s.pop();
116:            ans.push_back(cur->val);
117:        }
118:    }
119: } « end_pre_uni »
120: void _post_uni(TreeNode *root, vector<int>&ans)
121: {
122:     stack<TreeNode*>s;
123:     if (root) s.push(root);
124:     while(!s.empty()){
125:         TreeNode *cur=s.top();
126:         if (cur){
127:             s.pop();
128:             s.push(cur);
129:             s.push(nullptr);
130:             if (cur->right) s.push(cur->right);
131:             if (cur->left) s.push(cur->left);
132:         }else{
133:             s.pop();
134:             cur=s.top();
135:             s.pop();
136:             ans.push_back(cur->val);
137:         }
138:     }
139: } « end_post_uni »
140: void _in_uni(TreeNode *root, vector<int>&ans)
141: {
142:     stack<TreeNode*>s;
143:     if (root) s.push(root);
144:     while (!s.empty()){
145:         TreeNode *cur=s.top();
146:         if (cur){

```

```

147:         s.pop();
148:         if (cur->right) s.push(cur->right);
149:         s.push(cur);
150:         s.push(nullptr);
151:         if (cur->left) s.push(cur->left);
152:     }else{
153:         s.pop();
154:         cur=s.top();
155:         s.pop();
156:         ans.push_back(cur->val);
157:     }
158: }
159: } « end _in_uni »
160: 可以看到，只有入栈地方的次序不一样，别的地方都是一样的框架。
161:

```