链滴

二叉树遍历的常用方法

作者：vcjmhg
原文链接：https://ld246.com/article/1603678020363
来源网站：链滴

概述

二叉树的遍历可以说是解决二叉树问题的基础。我们常用的遍历方式无外乎就四种前序遍历、中序遍历、续遍历、层次遍历这四种。其中前三种遍历方式在实现时，即便采用不同的实现方式（递归方式、非归），它们的算法结构是有很大的相似性。因而针对前三种的遍历我们会总结出对应通用的解决框架便于在解决二叉树问题时进行使用。

递归方式

递归方式遍历二叉树时，无论是前序遍历、中序遍历还是后续遍历的方式，它们最大的区别就是**对节数据的访问位置不同。**除此之外其结构完全一致，因而我们总结出如下的框架结构：

```
void traverse(TreeNode root) {
    //终止条件
    if(root == null) return;
    // 前序遍历
    traverse(root.left);
    // 中序遍历
    traverse(root.right);
    // 后序遍历
}
```

对应注释的位置访问数据就可以实现不同的遍历方式。

例如，前序遍历：

```
void traverse(TreeNode root) {
    if(root == null) return;
    visit(root);
    traverse(root.left);
```

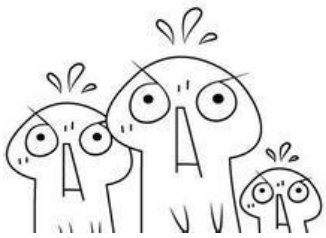
```
    traverse(root.right);
}
```

同样的中序遍历：

```
void traverse(TreeNode root) {
    if(root ==null) return;
    traverse(root.left);
    visit(root);
    traverse(root.right);
}
```

后续遍历：

```
void traverse(TreeNode root) {
    if(root ==null) return;
    traverse(root.left);
    traverse(root.right)
}
```



是否非常easy！！

非递归方式

二叉树非递归遍历说实话有**很多种实现方式**，但本质上都是**模拟整个遍历的过程来实现**的。

为了便于理解，其中前序遍历、中序遍历、后序遍历我们采用一套类似的算法框架。

整个算法框架如下：

```
 public void traverse(TreeNode root) {
    // 边界判断
    if (root == null) {
      return;
    }
    Stack<TreeNode> stack = new Stack<>();
    TreeNode current = root;
    while (current != null || !stack.isEmpty()) {
      //节点非空时，证明父节点的左侧节点非空，直接入栈
      if (current != null) {
        //前序遍历 visit(current)
        stack.push(current);
        current = current.left;
      } else {
```

```
        //节点为空，证明左侧节点为空，出栈，更换游标节点方向
        current = stack.pop();
        //中续遍历 visit(current);
        current = current.right;
      }
    }
  }
```

后序遍历它的遍历顺序为**"左-->右-->根"，**较之与前序遍历的**"根-->左-->右"，**好像是有很大的相性，我们能否针对上边的框架进行修改，使由前序遍历转换成后序遍历？？**

答案是肯定的，我们可以观察到，可以先求出遍历顺序是"根-->右-->左"**"的节点序列，再倒序，刚好是后序遍历的顺序：左右根。而遍历顺序是根右左的话，很好办，从前序遍历的代码中改两行就了。

故而，可以选择使用两个栈，其中一个用于遍历，另一个用于结果的倒序。

实现代码如下：

```
//使用双栈来实现后序遍历
  public void postOrderTraverse(TreeNode root){
    Stack<TreeNode> stack = new Stack<>();
    Stack<Integer> res = new Stack<>();
    TreeNode cur = root;
    while (cur!=null || !stack.isEmpty()) {
      if (cur!=null){
        stack.push(cur);
        res.push(cur.val);
        cur = cur.right; //修改处
      }else{
        cur = stack.pop();
        cur = cur.left;  // 修改处
      }
    }
    while (!res.isEmpty()){
      visit(res.pop());
    }
  }
```

至此，非递归遍历完成，是不是也很easy！！

下边我们可以看一下最后一种层次遍历

层次遍历

层次遍历本质上就是阉割版**广度优先遍历**，关于BFS我之前也写了一篇文章BFS与DFS套路总结，感兴的小伙伴可以去读一下。我们此处就直接给出BFS算法的框架：

```
/**
* 给定起始节点start和目标节点target，返回其最短路径长度
**/
int BFS(Node start,Node target){
    Queue<Node> q; //核心数据结构
    Set<Node> visited: //某些情况下可以通过byte数组来进行代替
    int step = 0; //记录扩散步数
```

```
    //起始节点入队列
    q.add(start);
    visited.offer(start);
    while(q not empty) {
        //必须要用sz来保存q.size()，然后扩散sz不能直接使用q.size()
        int sz = q.size();
        //将队列中的节点进行扩散
        for(int i =0 ; i < sz; i++) {
            Node cur = q.poll();
            // 目标节点判断
            if(cur is target) {
                return step;
            }
            // 邻接结点入队列
            for(Node n:cur.adjs) {
                //未访问节点入队列
                if(n is not int visited) {
                    visitd.add(n);
                    q.offer(n);
                }
            }
        }
        // 更新步数
        step++;
    }
}
```

此处我们借助BFS的框架，直接给出其实现方法：

```
void LevelOrder(TreeNode root){
    //初始化栈，并放入
    Queue<TreeNode> queue;
    queue.add(root);
    while( !queue.isEmpty()) {
        //出栈
        TreeNode cur = queue.poll();
        //访问节点
        visit(cur);
        //向下一层级扩散
        if(cur.left !=null) queue.add(cur.left);
        if(cur.right !=null) queue.add(cur.right);
    }
}
```

较之于BFS，我们会发现，层次遍历，少了好多东西，比如不需要visited来标记已访问的节点（二叉本身结构的特点，不可能出现重复遍历），也不需要将队列中的节点进行扩散等。

总结

至此，二叉树的四种遍历方式总结完成。我们发现其实二叉树所有的遍历方式都有一种通用的算法框，只要掌握算法本身的框架还是比较容易能够写出实现代码的。

参考

1. https://www.cnblogs.com/kangna/p/11846156.html
2. https://zhuanlan.zhihu.com/p/80578741