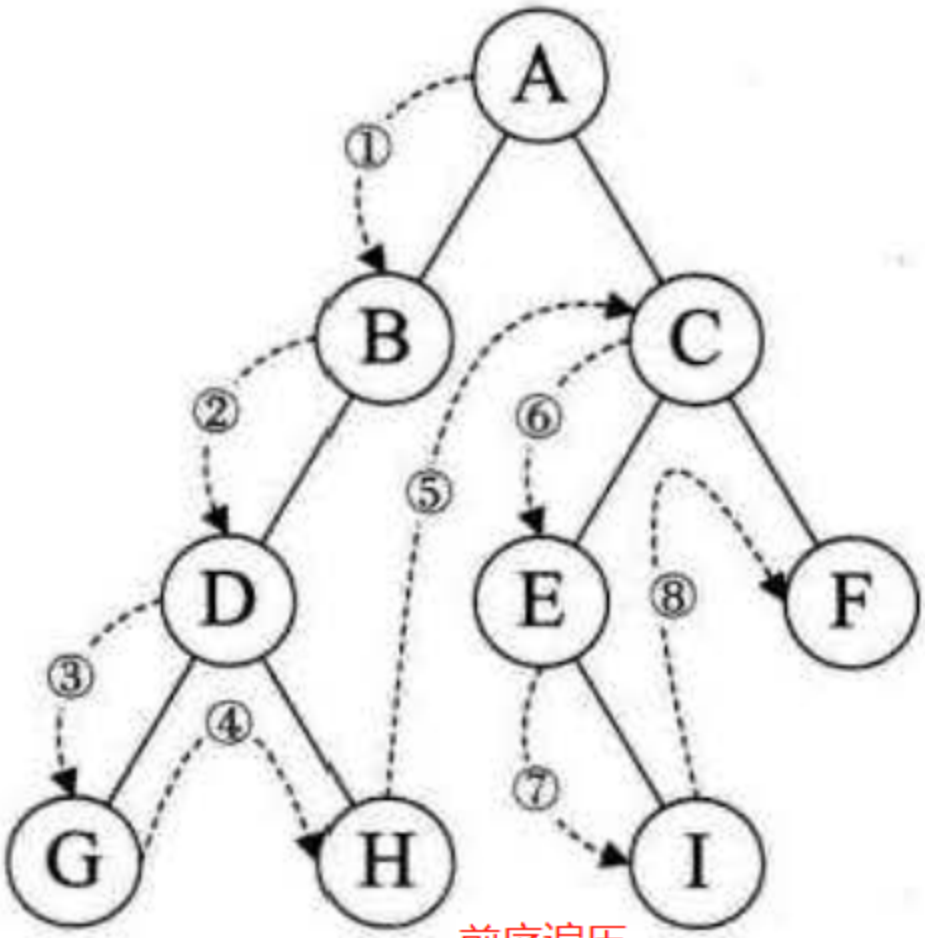
二叉树遍历

对于二叉树遍历常用方式： **前序遍历，中序遍历，后序遍历。**

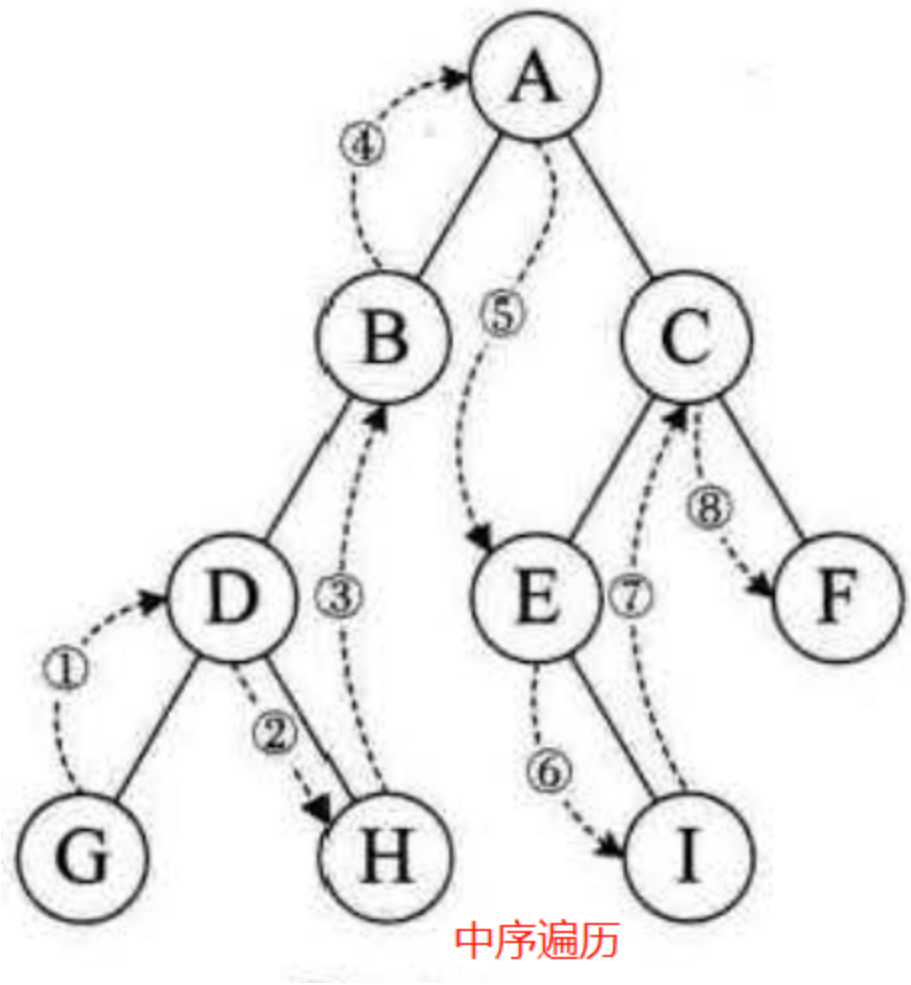
核心：

前序遍历：先遍历根结点，然后左子树，再右子树
中序遍历：先遍历左子树，然后根结点，再右子树
后续遍历：先遍历左子树，然后右子树，再根结点

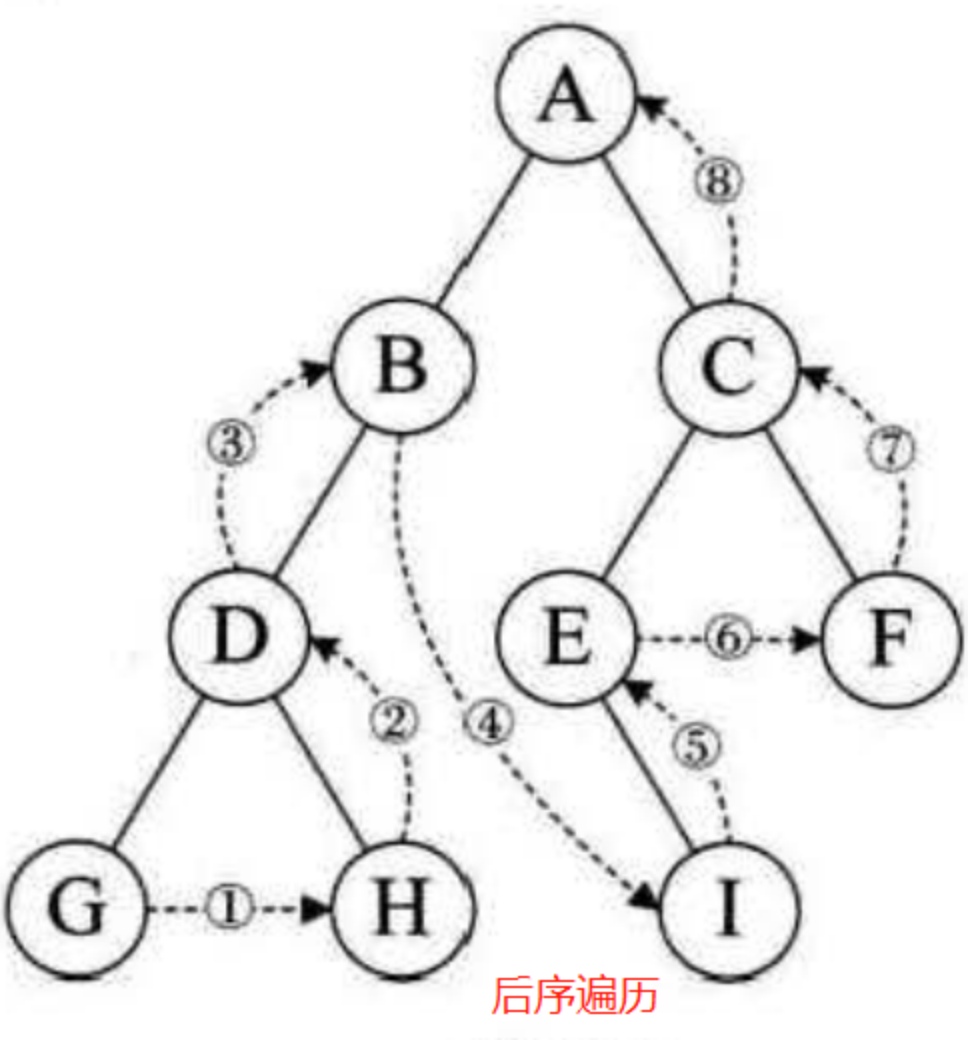
前序过程如下：



中序过程如下：



后序过程如下：



代码实现：

我们规定有一颗这样的二叉树：

```
class TreeNode {
    constructor(value, left, right) {
        this.value = value;
        this.left = left;
        this.right = right;
    }
}

let node7 = new TreeNode(7);
let node6 = new TreeNode(6);
let node5 = new TreeNode(5);
```

```
let node4 = new TreeNode(4);
let node3 = new TreeNode(3, node6, node7);
let node2 = new TreeNode(2,node4, node5);
let node1 = new TreeNode(1, node2, node3);
// node1 就是我们的根节点。
/**
 *    1
 *  2  3
 * 4 5 6 7
 *
 */
```

前序遍历（递归方式）：

```
var preorderTraversal = function(root) {
    let arr = [];
    let traverse = (root) => {
      // 先处理根节点
      if(root == null) return;
      arr.push(root.value);
      // 递归处理左节点
      traverse(root.left);
      // 递归处理右节点
      traverse(root.right);
    }
    traverse(root);
    return arr;
};
```

递归方式很好理解， 根左右即可。接下来看看非递归算法，这里面就需要利用到栈结构的知识了。

前序遍历（非递归方式）：

```
var preorderTraversal1 = function(root) {
    if(root == null) return [];
    let stack = [], res = [];
```

```
    stack.push(root);
    while(stack.length) {
        let node = stack.pop();
        res.push(node.value);
        // 左孩子后进先出，进行先左后右的深度优先遍历
        if(node.right) stack.push(node.right);
        if(node.left) stack.push(node.left);
    }
    return res;
};
```

接下来看看中序遍历。

中序遍历（递归方式）：

```
let inorderTraversal = function(root) {
    let arr = [];
    let traverse = (root) => {
      if(root == null) return;
      traverse(root.left);
      arr.push(root.value);
      traverse(root.right);
    }
    traverse(root);
    return arr;
};
```

中序遍历（非递归方式）：

```
let inorderTraversal = function(root) {
    if(root == null) return [];
    let stack = [], res = [];
    let p = root;
    while(stack.length || p) {
        while(p) {
            stack.push(p);
```

```
            p = p.left;
        }
        let node = stack.pop();
        res.push(node.value);
        p = node.right;
    }
    return res;
};
```

接下来看看后序遍历。

后序遍历（递归方式）：

```

var postorderTraversal = function(root) {
    let arr = [];
    let traverse = (root) => {
      if(root == null) return;
      traverse(root.left);
      traverse(root.right);
      arr.push(root.value);
    }
    traverse(root);
    return arr
};
```

后续递归（非递归方式）：

```
var postorderTraversal = function(root) {
    if(root == null) return [];
    let stack = [], res = [];
    let visited = new Set();
    let p = root;
    while(stack.length || p) {
        while(p) {
            stack.push(p);
```

```
            p = p.left;
        }
        let node = stack[stack.length - 1];
        // 如果右孩子存在，而且右孩子未被访问
        if(node.right && !visited.has(node.right)) {
            p = node.right;
            visited.add(node.right);
        } else {
            res.push(node.value);
            stack.pop();
        }
    }
    return res;
};
```