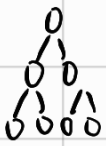
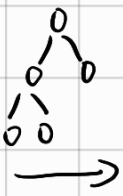
一、复习

种类：1、满二叉树



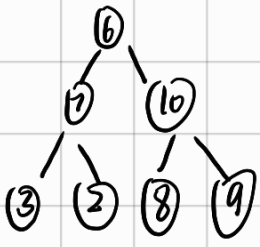
每棵树都是完整树

2、完全二叉树



从左到右每棵子树完整

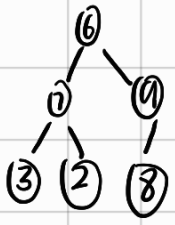
3、二叉搜索树



每棵左子树小于右子树

对结点完整性不作要求

4、平衡二叉搜索树



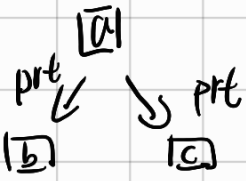
左子树和右子树高度差不过1

例：C++中 map，set，multi map，multi set

查询、插入 $\log n$ 级别

储存时：

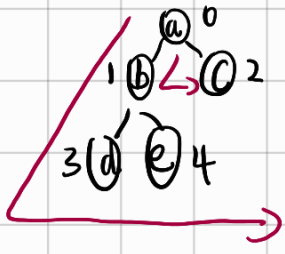
1、链式



2、顺序

a	b	c	d	e

这也可以先序遍历



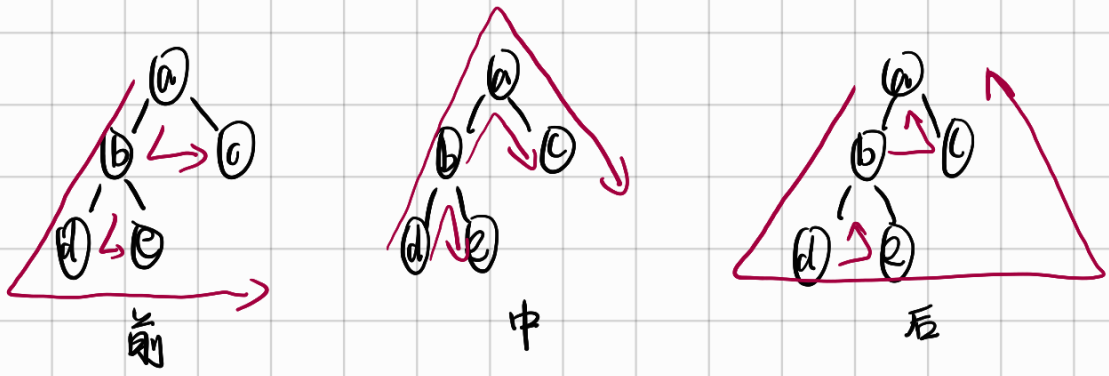
问左孩子，$2*i+1$，$2*i+2$

遍历方式：

？没用过

深度优先：前中后序　　迭代法、和递归法

广度优先：层序　　迭代法



二叉树节点的定义：

```
typedefine struct TreeNode {
        int value;
        TreeNode* leftChild;
        TreeNode* rightChild;
} TreeNode
```

二、遍历方式

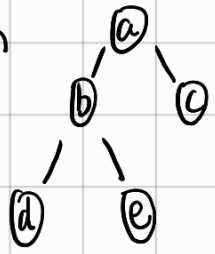
递归法：

```
void traversal (TreeNode* node, vector<int>& vec) {// 前序
    if(node == nullptr) return;
    vec.push_back(node->value); // 头
    traversal(node->leftChild); // 左
    traversal(node->rightChild); // 右
}
void traversal (TreeNode* node, vector<int>& vec) {// 中序
    if(node == nullptr) return;
    traversal(node->leftChild); // 左
    vec.push_back(node->value); // 头
    traversal(node->rightChild); // 右
}
```

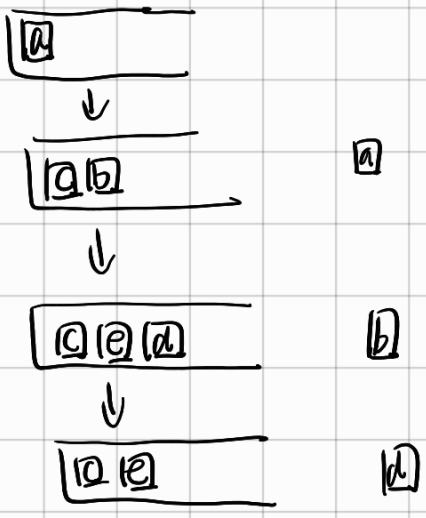
```
void traversal (TreeNode* node, vector<int>& vec) { // 后序
    if (node == nullptr) return;
    traversal (node->leftChild); // 左
    traversal (node->rightChild); // 右
    vec.push_back(node->value); // 头
}
```

迭代法：

先序



通过一个栈




```
vector<int> pretraversal (TreeNode* node) {
    stack <TreeNode*> stru;
    vector<int> result;
    stru.push_back(node);  -> if (node == nullptr) return result
    while(!stru.empty()) {
        TreeNode* cur = stru.top();
        stru.pop();
        result.push_back(cur->value);
        if (cur->rightChild != nullptr) stru.push_back(cur->rightChild);
        if (cur->leftChild != nullptr) stru.push_back(cur->leftChild);
    }
    return result;
}
```