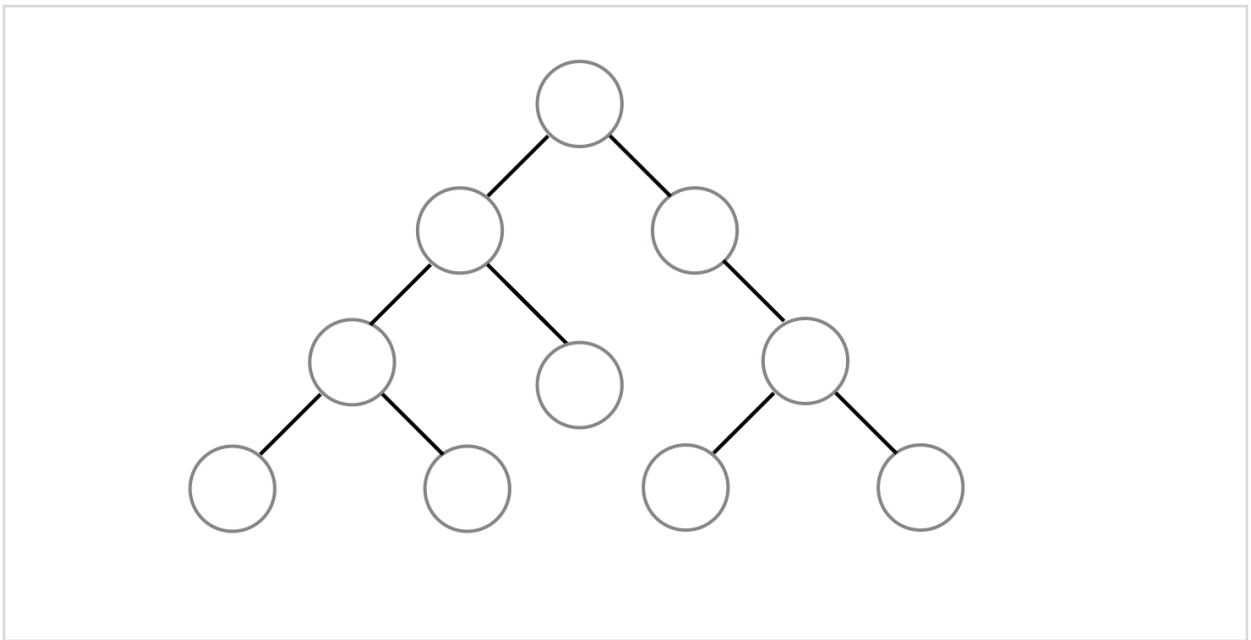
基础算法与数据结构（二） 树（1）基础二叉树

树的基本数据结构

```
//tree
 struct normalTreeNode {
      int val;
      TreeNode* sons[];
  };
//binary tree
  struct TreeNode {
      int val;
      TreeNode *left;
      TreeNode *right;
      TreeNode(int x) : val(x), left(NULL), right(NULL) {}
  };
```

树的基本概念

1. 根、子树、叶结点
2. 节点的度、分支节点、孩子、双亲、子孙、深度
3. 二叉树：每个节点至多只有两棵子树（度不大于2）



二叉树的典型架构

二叉树的性质

二叉树可以使用数组存储。

1. 在二叉树的第i层上最多只有2^(i-1)个结点
2. 深度为k的二叉树至多有2^k - 1 个结点
3. 对任何一棵二叉树T，如果其终端结点数为n0，度为2的节点数为n2，则n0=n2+1
4. 具有n个结点的完全二叉树的深度为[log2n]+1

二叉树的遍历

二叉树的遍历分为三种： 前序(root-left-right)、中序(left-root-right)、后序(left-right-root) （以root所在的位置为基准）

三种遍历均可用递归和非递归的方式进行，前序中序的非递归主要用一个栈来模拟递归的出入顺序，代码分别如下
所有结果都存在vector中返回
括号内数字为对应leetCode题目编号

1. 前序（144）

```
//递归
vector<int> route;
void PreOrderTraversal(TreeNode* root){
    if(!root){
        route.push_back(root->val);
        PreOrderTraversal(root->left);
        PreOrderTraversal(root->right);
    }
}

//非递归
vector<int> PreOrderTraversal(TreeNode* root){
    vector<int> route;
    stack<TreeNode*> s;
    TreeNode* cur = root;
    while(cur || s.empty()){
```

```
            while(cur){
                route.push_back(cur->val);
                s.push(cur);
                //go to the most left
                cur = cur->left;
            }
            if(!s.empty()){
                cur = s.top();
                s.pop();
                //back to right
                cur = cur->right;
            }
        }
        return route;
    }
```

2. 中序（94）

```
    //递归
    vector<int> route;
    void InOrderTraversal(TreeNode* root){
        if(!root){
            InOrderTraversal(root->left);
            route.push_back(root->val);
            InOrderTraversal(root->right);
        }
    }

    //非递归
    vector<int> InOrderTraversal(TreeNode* root){
        vector<int> route;
        stack<TreeNode*> s;
        TreeNode* cur = root;
        while(cur || s.empty()){
            while(cur){
                s.push(cur);
                //go to the most left
                cur = cur->left;
```

```
            }
            if(!s.empty()){
                cur = s.top();
                s.pop();
                route.push_back(cur->val);
                //back to right
                cur = cur->right;
            }
        }
        return route;
    }
```

3. 后序（145）

```
    //递归
    vector<int> route;
    void PostOrderTraversal(TreeNode* root){
        if(!root){
            PostOrderTraversal(root->left);
            PostOrderTraversal(root->right);
            route.push_back(root->val);
        }
    }

    //非递归
    vector<int> PostOrderTraversal(TreeNode* root){
         stack<TreeNode> s;
         vector<int> ans;
          TreeNode* pre,cur;
          pre = NULL;
          if(root == NULL) return ans;
          s.push(root);
          while(!s.empty()){
              cur = s.top();
            //要么左右均为空，那么就是叶子节点放入route，或者前一个是左结点或者右结点回溯回来的，此时也是已经访问过左右放入父节点
              if((!cur->left&&!cur->right) || (pre && (pre == cur->left || pre == cur->right))){
```

```
                s.pop();
                ans.push_back(cur.val);
                pre = cur;
            }else{
               //向栈内先放右，再放左，先进后出
                if(cur->right) s.push(cur->right);
                if(cur->left) s.push(cur->left);
            }
        }
        return ans;
    }
```

二叉树典型题

一般二叉树的题目都可以用递归的方法解决。（例如 求树的深度，树的最浅深度以及最深度等问题）

翻转二叉树 （leetCode 226）

这道题有一句名言那。。。

> Google: 90% of our engineers use the software you wrote (Homebrew), but you can’t invert a binary tree on a whiteboard so fuck off.!

分别翻转左右即可。。

```
TreeNode* invertTree(TreeNode* root) {
        if(root==NULL) return NULL;
        root->left = invertTree(root->left);
        root->right = invertTree(root->right);

        TreeNode* tmp = root->left;
        root->left = root->right;
        root->right = tmp;
        return root;
    }
```

利用前序/中序构造二叉树

根据前序中序的特征可以来模拟写出程序。

先找到前序中的第一个，然后利用这个在中序中找到这个值，然后这个值之前的在左子树中，之后的在右子树中，然后分别递归再次利用这个方法继续建树。

同样的可以利用中序/后序构建二叉树。

```
TreeNode *buildTree(vector<int> &preorder, vector<int> &inorder){
        if(preorder.empty() || inorder.empty()) return NULL;
        TreeNode* root = new TreeNode(preorder[0]);
        if(preorder.size()==1 && inorder.size()==1) return root;
        int inOrderNum = FindNode(inorder,preorder[0]);
        vector<int> a(preorder.begin()+1,preorder.begin()+inOrderNum+1);
        vector<int> b(inorder.begin(),inorder.begin()+inOrderNum+1);
        vector<int> c(preorder.begin()+inOrderNum+1,preorder.end());
        vector<int> d(inorder.begin()+inOrderNum+1,inorder.end());
        root->left = buildTree(a,b);
        root->right = buildTree(c,d);
        return root;
    }

    int FindNode(vector<int> &v,int target){
        int size = v.size();
        for(int i=0;i<size;i++){
            if(v[i] == target) return i;
        }
        return -1;
    }
```