Binary Tree and Recursive

Contents

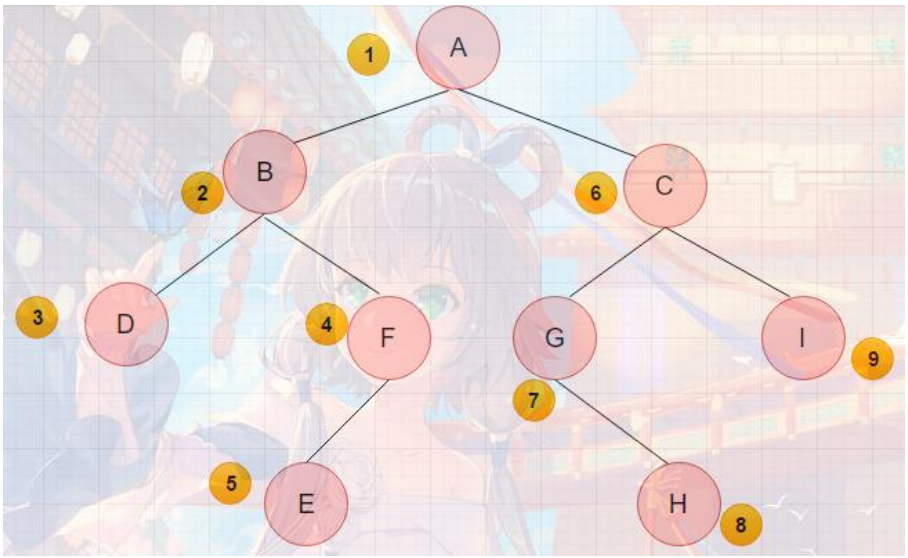
1. 二叉树的结构

```
// Definition for a binary tree node.
 typedef struct TreeNode {
     int val;
     struct TreeNode *left;
     struct TreeNode *right;
 }TreeNode_t;
```

2. 二叉树的遍历

前序遍历	先访问跟节点，在前序遍历左子树，再前序遍历右子树
中序遍历	中序遍历根节点的左子树，然后访问根节点，最后遍历右子树
后续遍历	从左到右先叶子节点再根节点
层序遍历	从根节点从上往下逐层遍历，在同一层，按从左到右对节点逐个访问。正好是一个 BFS 过程

a. 前序遍历 (根左右)

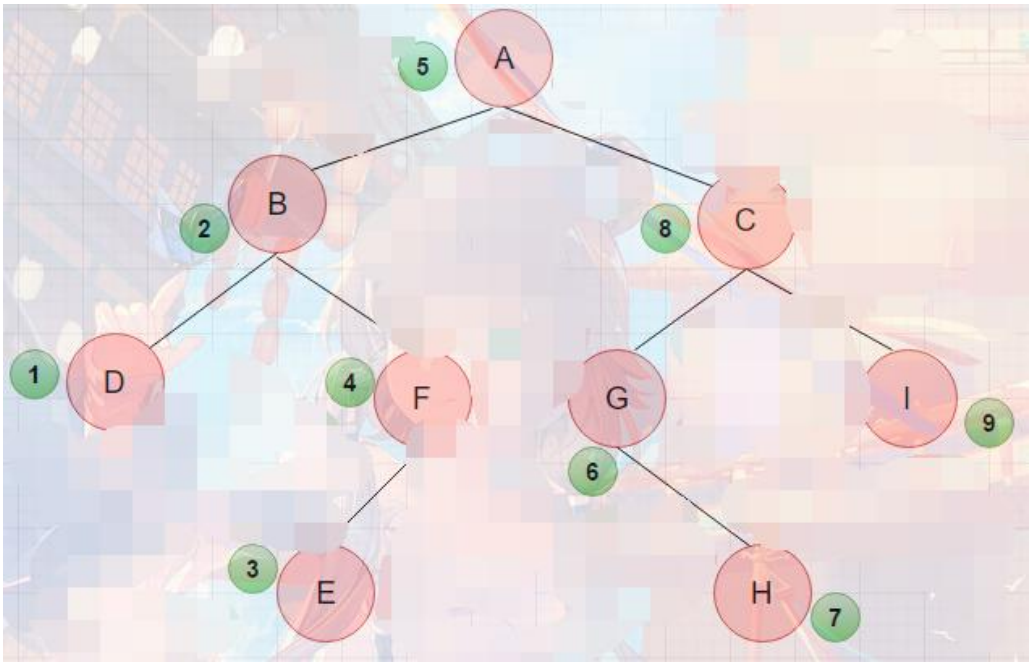



```
void pre_order_traverse(TreeNode_t *tree){
	if (tree == NULL){
		return;
	}

	printf("Node: %d. \n", tree->val);
	pre_order_traverse(tree->left);
	pre_order_traverse(tree->right);

	return;
}
```

b. 中序遍历（左根右）

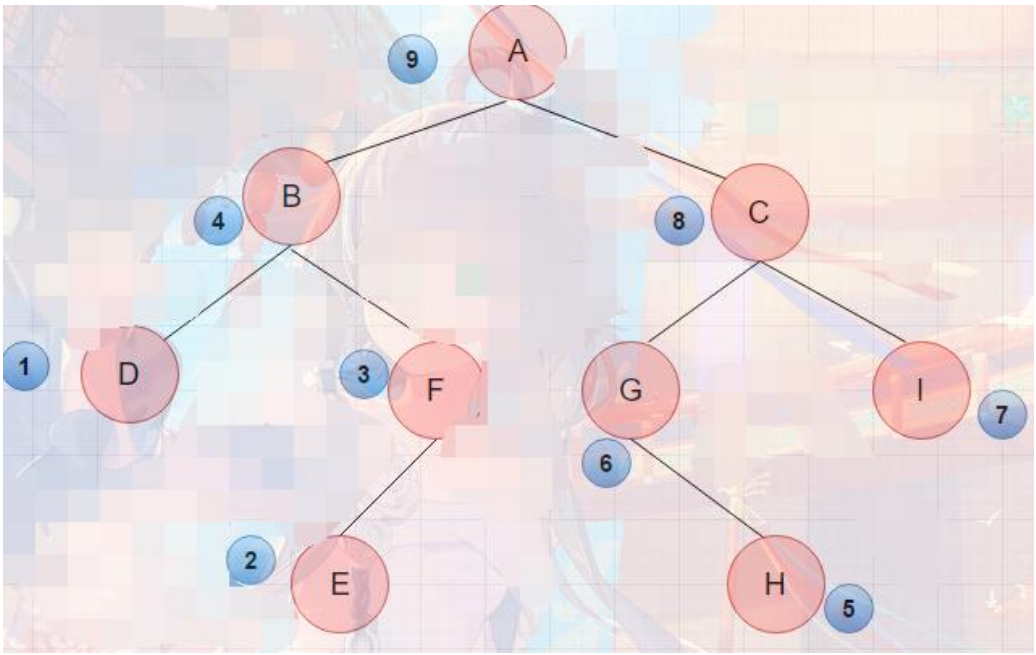



```
void in_order_traverse(TreeNode_t *tree){
	if (tree == NULL){
		return;
	}

	in_order_traverse(tree->left);
	printf("Node: %d. \n", tree->val);
	in_order_traverse(tree->right);

	return;
}
```

c. 后序遍历 （左右根）




```
void post_order_traverse(TreeNode_t *tree){
	if (tree == NULL){
		return;
	}

	post_order_traverse(tree->left);
	post_order_traverse(tree->right);
	printf("Node: %d. \n", tree->val);

	return;
}
```

3. 重建二叉树

已知前序遍历为{1,2,4,7,3,5,6,8}，中序遍历为{4,7,2,1,5,3,8,6}，它的二叉树是怎么样的?

思路： 根据前、中序遍历的特点，（根左右、左根右），先根据前序遍历确定根节点，然后在中序遍历知道该根节点的左右树的数量，反推出前序遍历中左子树的结点有哪些。根据该思路进行递归即可完成二叉树的重建。

```
static int preV[8] = {1,2,4,7,3,5,6,8};
static int inV[8] = {4,7,2,1,5,3,8,6};


TreeNode_t* reConstruct(int preV_start, int preV_end, int inV_start, int inV_end){
	TreeNode_t *Tree_node;
	int preV_start_left, preV_end_left, inV_start_left, inV_end_left;
	int preV_start_right, preV_end_right, inV_start_right, inV_end_right;
	int i, len;

	Tree_node = (TreeNode_t *) malloc(sizeof(TreeNode_t));
	Tree_node->val = preV[preV_start];

	if ((preV_start == preV_end) || (inV_start == inV_end)){
		Tree_node->left = NULL;
		Tree_node->right = NULL;
		return Tree_node;
	}

	for (i=inV_start;i<=inV_end;i++)
	{
		if (preV[preV_start] == inV[i]){
			break;
		}
	}

	len = i - inV_start;


	if (len == 0){
		Tree_node->left = NULL;
		//return NULL;

	} else {
		preV_start_left = preV_start + 1;
		preV_end_left = preV_start_left + len-1;
		inV_start_left = inV_start;
		inV_end_left = i - 1;

		Tree_node->left = reConstruct(preV_start_left, preV_end_left,
inV_start_left, inV_end_left);
	}

	if (len == (inV_end - inV_start)){
		Tree_node->right = NULL;
		//return NULL;

	}else {
```

```
            preV_start_right = preV_start + len + 1;
            preV_end_right = preV_end;
            inV_start_right = i + 1;
            inV_end_right = inV_end;

            Tree_node->right = reConstruct(preV_start_right, preV_end_right,
inV_start_right, inV_end_right);
      }

      return Tree_node;
}
```

4. 二叉树的下一节点

给定一个二叉树的节点，如何找出中序遍历的下一节点。有两个指向左右子树的指针，还有一个指向父节点的指针。
思路：求中序遍历的下一节点，就要分各种情况（明确中序遍历下一结点在二叉树中的位置有哪些），然后对某种情况详细分析。

下一结点可能存在的情况：

- 有右子节点
  - 右子节点有无左子节点
    - 无 —— 右子节点就是当前结点下一节
    - 有 —— 递归寻找右子节点的左子节点就是下一节点
- 无右子节点
  - 无父节点 —— 无下一结点
  - 有父节点
    - 当前结点作为父节点的左子节点 —— 下一结点为父节点
    - 当前结点作为父节点的右子节点 —— 向父节点递归寻找作为左子节点的结点就是下一节点

```
TreeNode_with_Parent_t *nextNode(TreeNode_with_Parent_t *tree_node_p){
    TreeNode_with_Parent_t *ptree, *parentNode;
    /* Case 1:
     有右子节点
    右子节点有无左子节点
           无 —— 右子节点就是当前结点下一节
           有 —— 递归寻找右子节点的左子节点就是下一节点
    */

    if (tree_node_p->right != NULL){
           if (tree_node_p->right->left == NULL){
                  return tree_node_p->right;
           }else {
                  ptree = tree_node_p->right->left;
                  while (ptree->left!=NULL)
                         ptree = ptree->left;

                  return ptree;
           }
    }

    /*
           无右子节点
                  无父节点 —— 无下一结点
                  有父节点
                         - 当前结点作为父节点的左子节点 —— 下一结点为父节点
                         - 当前结点作为父节点的右子节点 —— 向父节点递归寻找作为左子节点的
结点就是下一节点
    */

    if (tree_node_p->right == NULL){
           if (tree_node_p->parentNode == NULL){
                  return NULL;
           }else {
                  ptree = tree_node_p; //->parentNode;
                  while (ptree->parentNode != NULL){
                         if (ptree->parentNode->left == ptree){
                                return ptree->parentNode;
                         }

                         if (ptree->parentNode->right == ptree){
                                       ptree = ptree->parentNode;
                         }
                  }
           }
    }

    return NULL;
}
```

5. 树的子结构

输入两棵二叉树 A 和 B，判断 B 是不是 A 的子结构。
思路： 通过判断两棵树的根节点否相同，如果相同，则递归判断树剩余的结点是否相同。如果不相同，则递归树的左右子节点进行对比找到相同的根节点。

```
/*Need two iterations here*/
/* Is treeB is subtree of treeA */
int IsBSubTreeofA(TreeNode_t *treeA, TreeNode_t *treeB){
        if (treeB == NULL){
              return 1;
        }

       if (treeA == NULL){
             return 0;
       }

       if (treeA->val != treeB->val){
             return 0;
       }


       if (treeA->val == treeB->val){
             return (IsBSubTreeofA(treeA->left, treeB->left) &&
IsBSubTreeofA(treeA->right, treeB->right));
       }

       return 0;
}

/*treeA combinations*/
int TreeA_Combinations(TreeNode_t *treeA, TreeNode_t *treeB){
       if (treeA == NULL){
                    return 0;
       }

        if (treeB == NULL){
                     return 1;
        }

        if (treeA->val == treeB->val){
                    return (IsBSubTreeofA(treeA->left, treeB->left) &&
IsBSubTreeofA(treeA->right, treeB->right));
        }

       return (TreeA_Combinations(treeA->left, treeB) ||
TreeA_Combinations(treeA->right, treeB));
}
```

6. 二叉树的镜像

请完成一个函数，如果一个二叉树，该函数输出它的镜像。
思路：根节点的左右子节点相互交换，继续递归遍历，将子节点的左右结点进行交换，直到遇到叶子节点。

```
void mirror_binary_tree(TreeNode_t *tree){
	TreeNode_t *temp_tree;

	if (tree == NULL)
		return;

	temp_tree = tree->left;
	tree->left = tree->right;
	tree->right = temp_tree;

	image_binary_tree(tree->left);

	image_binary_tree(tree->right);


	return;
}
```

7. 对称二叉树

思路

1、首先，观察一个对称的二叉树有什么特点？

结构上：在结构上实对称的，某一节点的左子节点和某一节点的右子节点对称。

规律上：我们如果进行前序遍历（根、左、右），然后对前序遍历进行改进（根、右、左），如果是对称的二叉树，他们的遍历结果是相同的。

2、考虑其他情况

结点数量不对称

结点值不对称

```
int Symmetric_tree(TreeNode_t *tree){
	if ((tree->left == NULL) && (tree->right == NULL))
		return 1;

	if ((tree->left == NULL) && (tree->right != NULL))
		return 0;

	if ((tree->left != NULL) && (tree->right == NULL))
		return 0;

	if (tree->left->val != tree->right->val)
		return 0;

	return ((Symmetric_tree(tree->left))&&Symmetric_tree(tree->right));
}
```

8. 从上到下打印二叉树

从上到下打印出二叉树的每个节点，同一层的节点按照从左到右的顺序打印。（按层遍历二叉树）

思路:

从根节点开始按层遍历打印结点（自左往右），下一层的遍历是上一层的字节点，但是我们发现想要获取到上层结点的子节点时，上层的父节点已经遍历过去可，想要在获取到，必须存储父节点。然后下层遍历的时候，自左往右取出父节点，依次打印子节点。

上方的解题思路中父节点的存储和遍历让我们想到一个熟悉的数据结构，对了，“先进先出”的思想，那就是队列。在遍历上一层结点的时候，先打印结点值，然后判断是够存在左右子树，如果存在，将给结点入队，直到该层的结点全部遍历完成。然后队列出队，分别打印结点，循环此步骤。

```
void levelOrder(TreeNode_t *tree){
	TreeNode_t *queueNode[100];
	int enqueueNodeCnt = 0;
	int outqueueNodeCnt = 0;
	int i;

	queueNode[enqueueNodeCnt++] = tree;

	while (outqueueNodeCnt < enqueueNodeCnt){
		if (tree->left != NULL){
			queueNode[enqueueNodeCnt++] = tree->left;
		}

		if (tree->right != NULL){
			queueNode[enqueueNodeCnt++] = tree->right;
		}

		tree = queueNode[++outqueueNodeCnt];
	}


	for (i=0;i<enqueueNodeCnt;i++)
	{
		printf("%d ", queueNode[i]->val);
		fflush(stdout);
	}

	printf("\n");
	fflush(stdout);
}
```

需要两个计数器： enqueueIdx 和 outqueueIdx, 当 enqueue <= outqueue 就退出循环

9. 二叉树的后序遍历序列

输入一个整数数组，判断该数组是不是某二叉搜索树的后续遍历。如果是返回 true，如果不是返回 false。假设输入的任意两个数字互不相同。

思路:

根据后续遍历的规律和二叉树具备的特点，可以找到的规律就是（左、右、根）序列的最后一个数为根节点，又根据二叉树的特点，左子节点小于根节点，右子节点大于根节点，分离出左右子节点，根据上边的规律，**递归**剩下的序列。

根据后序遍历的顺序，我们可以知道数组的最后一个节点一定是二叉树的根节点，那么前面的序列，一部分是左子树（都比根节点小），另一部分是右子树（都比根节点大）。此后，可以递归地进行判断，左子树序列中的最后一个元素是左子树跟节点，比它小的是其左子树，比它大的是其右子树。……以此类推，如果存在在左子树序列中出现比根节点大的，或者右子树序列中比根节点小的情况，那么就说明不是二叉搜索树的后序遍历序列。

“二叉搜索树的特点：对于树中的每个节点X，它的左子树中所有关键字值小于X 的关键字值，而它的右子树中所有关键字值大于X 的关键字值。

根据这个性质，对一个二叉树进行中序遍历，如果是单调递增的，则可以说明这个树是二叉搜索树”

```
int isPostOrder(int *array, int start, int end){
	int i, j;
	int left_start, left_end, right_start, right_end;
	int isLeftPostOrder, isRightPostOrder;

	if (start>end){
		return 1;
	}

	for (i=start;i<end;i++){
		if (array[i] > array[end]){
			break;
		}
	}

	left_start = start;
	left_end = i-1;


	right_start = i;
	right_end = end -1;


	for (j=right_start;j<end;j++){
		if (array[j] < array[end])
			return 0;
	}


	isLeftPostOrder = isPostOrder(array, left_start, left_end);
```

```
        isRightPostOrder = isPostOrder(array, right_start, right_end);

        return (isLeftPostOrder && isRightPostOrder);
}
```

10. 二叉树和为某一值路径

输入一棵二叉树和一个整数，打印出二叉树中节点值的和为输出整数的所有路径。从树的根节点开始往下一直到叶子节点所经过的节点形成一条路径。(不是子树的根节点到叶子节点)
思路:

本题目的是找出所有符合条件的路径，因此要遍历二叉树，使用先序遍历可以从根节点开始。

需要一个辅助空间存储遍历过的树节点。当程序遍历到叶子节点的时候，如果满足所有路径和为指定值，打印路径。

在递归遍历节点过程中，当程序遍历完一条完整的路径，需要退回上一级父节点，此时也在路径中删除当前节点。

```
int stack[20]; // stack
int stack_sum = 0; // sum in stack elements
int stack_cnt = 0; // number of element in stack

int path_num = 0;  // number of path
int savedPath[10][20]; // max number of path, path length
int savedPathLen[10];

void treeSum(TreeNode_t *Tree_node, int expected_sum){

        stack[stack_cnt++] = Tree_node->val;
        stack_sum += Tree_node->val;

        if ((Tree_node->left == NULL) && (Tree_node->right == NULL)){
                if (stack_sum == expected_sum){
                        savedPathLen[path_num] = stack_cnt;
                        memcpy(savedPath[path_num], stack, stack_cnt * sizeof(int));
                        path_num++; // increase number of path
                }
        }

        if (Tree_node->left != NULL)
                treeSum(Tree_node->left, expected_sum);
```

```
        if (Tree_node->right != NULL)
                treeSum(Tree_node->right, expected_sum);


        stack_cnt--; // decrease the number of element in stack
        stack_sum -= Tree_node->val;

        return;
}
```

11. 序列化二叉树

请实现两个函数，分别用来序列化二叉树和反序列化二叉树。

思路

1、序列化：遍历二叉树，遇到叶子节点，将其转化为 $ 表示。

2、反序列化：根据前序遍历的特点（根、左、右），进行二叉树的还原。

```
int NodeVal[100];
int serialize_cnt = 0;

void serialize(TreeNode_t *tree){
        if (tree == NULL){
                NodeVal[serialize_cnt++] = 1000;
                return;
        }

        NodeVal[serialize_cnt++] = tree->val;
        serialize(tree->left);
        serialize(tree->right);

        return;
}


int NodeVal_deS[13] = {0, 1, 3, 1000, 1000, 4, 5, 1000, 1000, 1000, 2, 1000, 1000};
int deS_cnt = 0;

TreeNode_t* deSerialize(){
        TreeNode_t *Tree_node;

        if (deS_cnt == 13)
                return NULL;

        if (NodeVal_deS[deS_cnt]==1000){
                deS_cnt++;
                return NULL;
```

```
        }


        Tree_node = (TreeNode_t *) malloc(sizeof(TreeNode_t));

        Tree_node->val = NodeVal_deS[deS_cnt++];

        Tree_node->left = deSerialize();

        Tree_node->right = deSerialize();



        return Tree_node;
}
```

12. 二叉树的第 K 大节点

给定一棵二叉搜索树，请找出其中的第 K 大节点。

思路

要想找到第 K 大结点必要要知道排序，二叉树的前、中、后遍历中的中序遍历就是从小到大排序。然后遍历的同时计数找到第 K 大节点。

```
int cnt = 0;
int K = 4;
int KstNode(TreeNode_t *tree){
        if (tree==NULL)
                return 0;

        KstNode(tree->left);
        cnt++;

        //if (cnt == K){
                printf("the %d'th Node is %d. \n", cnt, tree->val);
                fflush(stdout);
        //}

        KstNode(tree->right);

        return 0;
}
```

13. 二叉树的深度

输入一棵二叉树的根节点，求该树的深度。从根节点到叶子节点依次经过的节点（包含根、叶子节点）形成树的一条路径，最长路径的长度树的深度。

思路

1、思路一：按层遍历，对按层遍历的算法进行改进，每遍历一次层进行加一。

2、思路二：寻找最长路径，借助遍历最长路径的设计思路记性改进。只需记录两个子树最深的结点为主。

```
#define MAX(x,y) (x)>(y)?(x):(y)

int MaxTreeDepth(TreeNode_t *tree){
	int leftDepth;
	int rightDepth;
	int depth;

	if (tree == NULL)
		return 0;

	leftDepth = MaxTreeDepth(tree->left);
	rightDepth = MaxTreeDepth(tree->right);

	depth = MAX(leftDepth, rightDepth);
	depth += 1;
	return depth;
}
```

参考资源：

【1】https://github.com/luxiangqiang/Blog/blob/master/articel/数据结构与算法系列/数据结构与算法之二叉树系列%5B 题型篇%5D.md

【2】https://www.cnblogs.com/du001011/p/11229170.html