目录

[1. 代码架构 2](#_Toc117150581)

[2. 代码说明 2](#_Toc117150582)

[2.1. 树的相关代码 2](#_Toc117150583)

[2.1.1. 数据结构 2](#_Toc117150584)

[2.1.2. 函数声明 2](#_Toc117150585)

[2.1.3. 函数分析 2](#_Toc117150586)

[2.2. 二叉树的相关代码 2](#_Toc117150587)

[2.2.3. 函数分析 2](#_Toc117150588)

[2.3. 森林转二叉树的相关代码 2](#_Toc117150589)

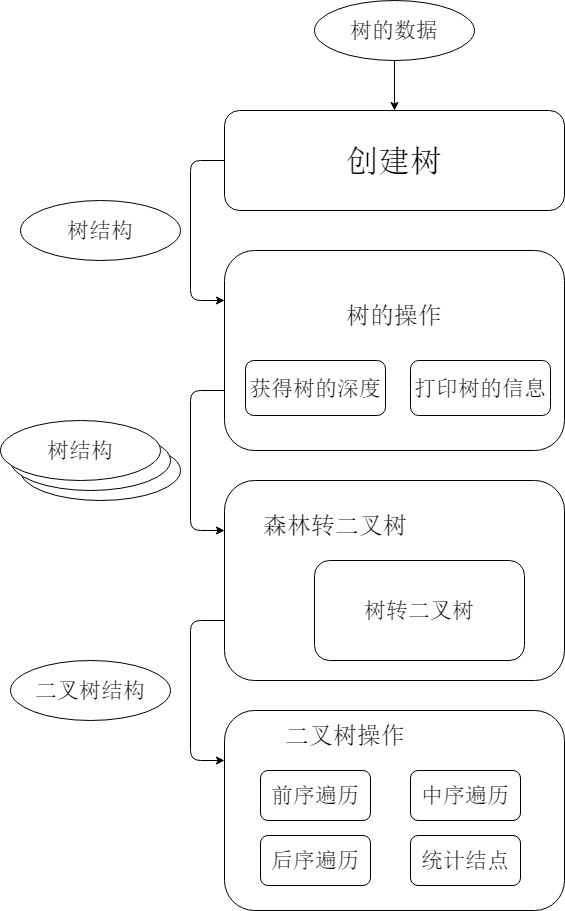
[2.3.1. 函数声明 2](#_Toc117150590)

[2.3.2. 函数分析 2](#_Toc117150591)

[3. 调试 2](#_Toc117150592)

# 代码架构

题目要求为森林转二叉树，通过过程中结构的不同分为四部分，分别为根据数据创建树结构，树的操作，根据一组树的结构创建一个二叉树以及二叉树的操作。



# 代码说明

## 树的相关代码

### 数据结构

定义树的结点TreeNode，包括数据，父节点，第一孩子结点。定义树的结构，包括数结点数组，以及存放结点数量的变量count。

1. /\* 树的结点 \*/
2. typedef struct PTNode{
3. char data;
4. int parent;
5. int firstchild;
6. }TreeNode;
7. /\* 树 \*/
8. typedef struct{
9. TreeNode nodes[MAX\_TREE\_SIZE];
10. int count = 0;
11. }Tree;

### 函数声明

1. /\* 树的相关函数 \*/
2. TreeNode\* CreateTreeNode(char nodeData, int parentData);
3. Tree\* CreatTree(int number,char dataInfo[], int parentInfo[]);
4. void SortTreeNode(int number, char\* dataInfo, int\* parentInfo);
5. int getTreeDeep(Tree\* tree);
6. int numlen(int num);
7. void PrintLayout(int data, int length);
8. void PrintLayout(char data, int length);
9. void ShowTree(Tree\* tree);

### 函数分析

#### 初始化树的结点

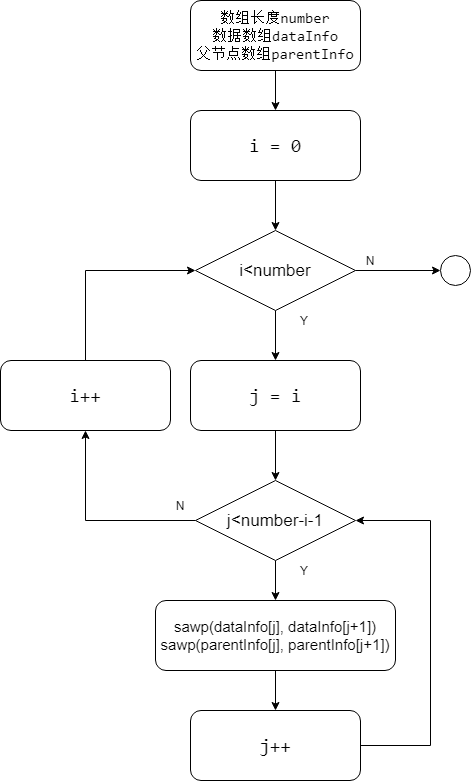
创建并初始化树的结构结点，将输入的数据信息和父结点信息录入到结点中，并将第一孩子结点赋值为默认值-1，并返回该结点。

1. TreeNode\* CreateTreeNode(char nodeData, int parentData) {
2. TreeNode\* newNode = new TreeNode;
3. newNode->data = nodeData;
4. newNode->parent = parentData;
5. newNode->firstchild = -1;
6. return newNode;
7. }

#### 将树结点信息排序

因为树结构中的结点出现的顺序代表了每一层从左到右的顺序，所以整个树对结点的顺序有极高的要求，才需要对有可能混乱的结点以父结点信息进行排序，这里使用冒泡排序，操作完之后不会对原本的顺序打乱。

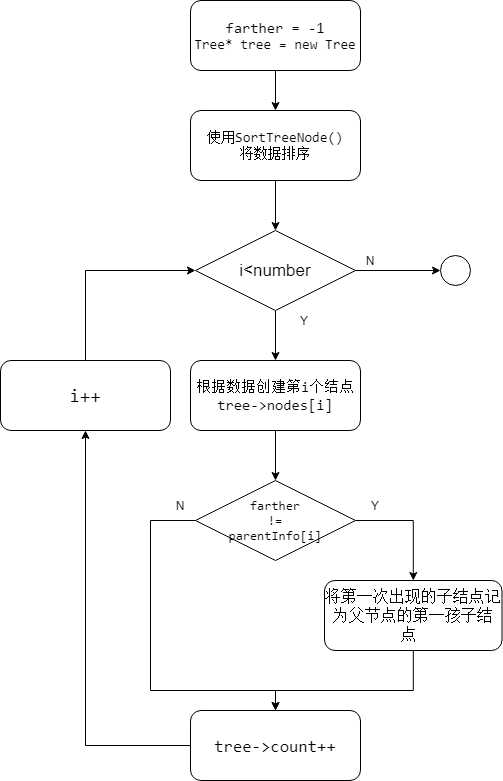
1. void SortTreeNode(int number, char\* dataInfo, int\* parentInfo) {
2. int i, j;
3. int intTemp;
4. char charTemp;
5. // 冒泡排序
6. for (i = 0; i < number; i++) {
7. for(j = i; j < number - i - 1; j++){
8. if (parentInfo[j] > parentInfo[j+1]) {
9. charTemp = dataInfo[j + 1];
10. dataInfo[j + 1] = dataInfo[j];
11. dataInfo[j] = charTemp;
13. intTemp = parentInfo[j + 1];
14. parentInfo[j + 1] = parentInfo[j];
15. parentInfo[j] = intTemp;
16. }
17. }
18. }
19. }



#### 创建树结构

创建一个树结构（双亲表示法加第一孩子结点），并将信息录入。需要先对输入的数据排序。再将数据信息和父结点信息通过初始化的方式创建，如果第一次出现新的父结点，则将该结点的父结点的第一孩子结点指向该结点的下标。即将所有有子树的结点的第一孩子结点信息录入。

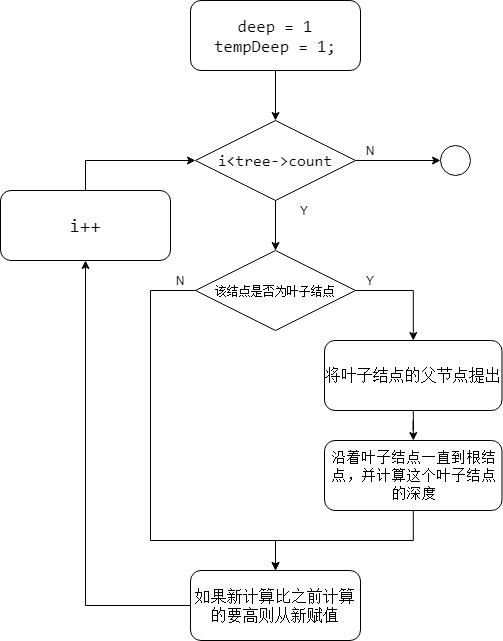
1. Tree\* CreatTree(int number,char dataInfo[], int parentInfo[]) {
2. int farther = -1;
3. Tree\* tree = new Tree;
4. // 先将结点排序
5. SortTreeNode(number, dataInfo, parentInfo);
6. // 遍历所有结点数组信息
7. for (int i = 0; i < number; i++) {
8. // 创建一个树的结点，将输入录入
9. tree->nodes[i] = \*CreateTreeNode(dataInfo[i], parentInfo[i]);
10. // 将第一次出现的子结点记为父节点的第一孩子结点
11. if (farther != parentInfo[i]) {
12. farther = parentInfo[i];
13. tree->nodes[farther].firstchild = tree->count;
14. }
15. tree->count++;
16. }
17. return tree;
18. }



#### 获得树的深度

遍历整个树结构，找到所有的叶子结点，并从叶子结点开始返回根结点，统计需要几次才能达到，即是该叶子结点的高度。将所有叶子结点都统计后，返回高度最大的叶子结点，就是该树的高度。

1. int getTreeDeep(Tree\* tree) {
2. int i;
3. int deep = 1;
4. int tempDeep = 1;
5. // 遍历整个树结构
6. for (i = 0; i < tree->count; i++) {
7. // 找到所有叶子结点
8. if (tree->nodes[i].firstchild == -1) {
9. // 将叶子结点的父节点提出
10. int perantIndex = tree->nodes[i].parent;
11. // 沿着叶子结点一直到根结点，并计算这个叶子结点的深度
12. while (perantIndex != -1) {
13. perantIndex = tree->nodes[perantIndex].parent;
14. tempDeep++;
15. }
16. }
17. // 如果新计算比之前计算的要高则从新赋值
18. if (tempDeep > deep) {
19. deep = tempDeep;
20. }
21. tempDeep = 1;
22. }
23. return deep;
24. }



#### 计算数的长度

计算输入的数占多少位（包括符号位）。如果输入的是0则直接返回1，如果是负数则长度先加一。通过除以10的方式知道数小于0，最后返回数的长度。

1. int numlen(int num) {
2. int len = 0;
3. // 如果数为0则直接返回1
4. if (num == 0) return 1;
5. // 如果数为负数，则长度加1且对数取绝对值
6. if (num < 0) {
7. len++;
8. num = abs(num);
9. }
10. // 计算出数的长度
11. for(; num > 0; ++len)
12. num /= 10;
13. return len;
14. }

#### 格式化打印树的信息

这里使用重载的方法，因为输出的数据可能是int型也可能是char型。主要实现了数据的长度如何，最后打印出的总长度永远是length。是为了打印的美观性。

1. void PrintLayout(int data, int length) {
2. int i;
3. cout<< "| " << data;
4. for (i = 0; i < length - numlen(data); i++) {
5. cout<< ' ';
6. }
7. }
8. void PrintLayout(char data, int length) {
9. int i;
10. cout<< "| " << data;
11. for (i = 0; i < length - strlen(&data); i++) {
12. cout<< ' ';
13. }
14. }

#### 打印数的所有信息

遍历整个树结构，将树结构中的所有信息通过格式化的方式打印。

1. void ShowTree(Tree\* tree) {
2. int i, j;
3. cout << "树的内容为:" << endl;
4. cout << "==============================" << endl;
5. cout << "|index| data|parent|fistchild|" << endl;
6. for (i = 0; i < tree->count; i++) {
7. PrintLayout(i, 4);
8. PrintLayout(tree->nodes[i].data, 4);
9. PrintLayout(tree->nodes[i].parent, 5);
10. PrintLayout(tree->nodes[i].firstchild, 8);
11. cout << "|" << endl;
12. }
13. cout << "==============================" << endl;
14. }

## 二叉树的相关代码

#### 数据结构

定义二叉树的结构BinaryTree，包括数据，左右孩子结点结点。

1. /\* 二叉树 \*/
2. typedef struct BinaryTree {
3. char data;
4. BinaryTree \*Lchild, \*Rchild;
5. }BinaryTree;

#### 函数声明

1. /\* 二叉树的相关函数 \*/
2. BinaryTree\* CreateTreeNode(char data);
3. BinaryTree\* CreateTree(Tree\* tree, int i);
4. int GetNodeNumber(BinaryTree\* root);
5. void ShowTreeMLR(BinaryTree\* root);
6. void ShowTreeLMR(BinaryTree\* root);
7. void ShowTreeLRM(BinaryTree\* root);

### 函数分析

#### 创建二叉树结点

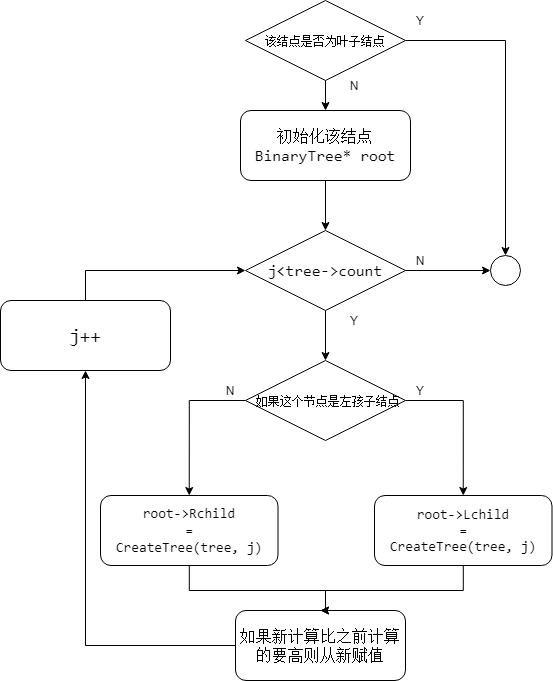
创建并初始化二叉树树的结构结点，将输入的数据信息录入到结点中，并将左右孩子结点赋值为默认值NULL，并返回该结点。

1. BinaryTree\* CreateTreeNode(char data) {
2. BinaryTree\* tree = new BinaryTree;
3. tree->data = data;
4. tree->Lchild = NULL;
5. tree->Rchild = NULL;
6. return tree;
7. }

#### 创建二叉树结构

将树结构和对应结点下标传入（第一次调默认下标为0），将该树的对应下标结点创建为二叉树结构。遍历树中这个节点下面的所有结点，如果出现了是该结点的孩子结点时，再判断该孩子结点是不是父节点的第一孩子结点，如果是且结点的左子树为空，则将结点的左子树指向该结点，并将孩子结点的数据和下标传入函数递归调用；如果不是第一孩子结点且右子树为空，则将结点右子树指向该结点，进行递归调用。直至出入的结点是叶子结点，则结束递归开始返回。

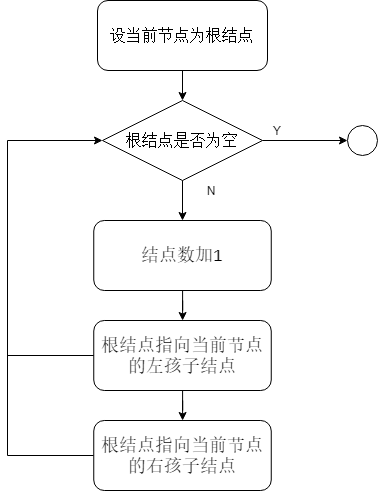
1. BinaryTree\* CreateTree(Tree\* tree, int i = 0) {
2. // 不能是叶子结点
3. if (i != -1) {
4. // 初始化该结点
5. BinaryTree\* root = CreateTreeNode(tree->nodes[i].data);
6. // 遍历数结构
7. for (int j = i; j < tree->count; j++) {
8. // 找到是该结点的孩子结点
9. if (tree->nodes[j].parent == i) {
10. //cout << root->data;
11. if (tree->nodes[tree->nodes[j].parent].firstchild == j) {
12. // 如果这个节点是左孩子结点
13. if (root->Lchild == NULL) {
14. //cout << "L" << tree->nodes[j].data << endl;
15. root->Lchild = CreateTree(tree, j);
16. }
17. } else {
18. // 如果这个节点是右孩子结点
19. if (root->Rchild == NULL) {
20. //cout << "R" << tree->nodes[j].data << endl;
21. root->Rchild = CreateTree(tree, j);
22. }
23. }
24. }
25. }
26. return root;
27. } else {
28. return NULL;
29. }
30. }



#### 获得树中的结点数

变量整个二叉树结构，只要该结点不为空，就将变量加一。分别递归对应该结点的左右子树。

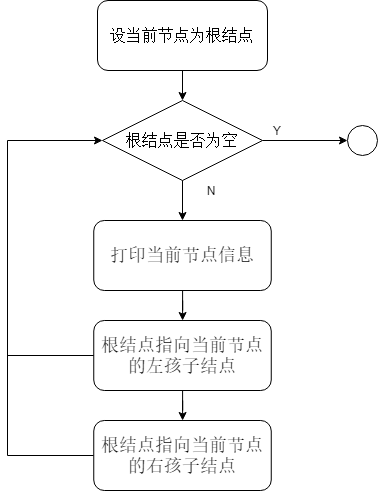
1. int GetNodeNumber(BinaryTree\* root) {
2. if (root != NULL) {
3. nodeNumber++;
4. GetNodeNumber(root->Lchild);
5. GetNodeNumber(root->Rchild);
6. }
7. return nodeNumber;
8. }



#### 获得前序中序后序遍历结果

三种遍历方式本质一样，只是递归调用的时间不同。通过控制何时打印该结点数据，就可以得到前序中序后序遍历结果。

1. /\* 先序遍历 \*/
2. void ShowTreeMLR(BinaryTree\* root) {
3. if (root != NULL) {
4. cout << root->data;
5. ShowTreeMLR(root->Lchild);
6. ShowTreeMLR(root->Rchild);
7. }
8. }
10. /\* 中序遍历 \*/
11. void ShowTreeLMR(BinaryTree\* root) {
12. if (root != NULL) {
13. ShowTreeLMR(root->Lchild);
14. cout << root->data;
15. ShowTreeLMR(root->Rchild);
16. }
17. }
19. /\* 后序遍历 \*/
20. void ShowTreeLRM(BinaryTree\* root) {
21. if (root != NULL) {
22. ShowTreeLRM(root->Lchild);
23. ShowTreeLRM(root->Rchild);
24. cout << root->data;
25. }
26. }



## 森林转二叉树的相关代码

### 函数声明

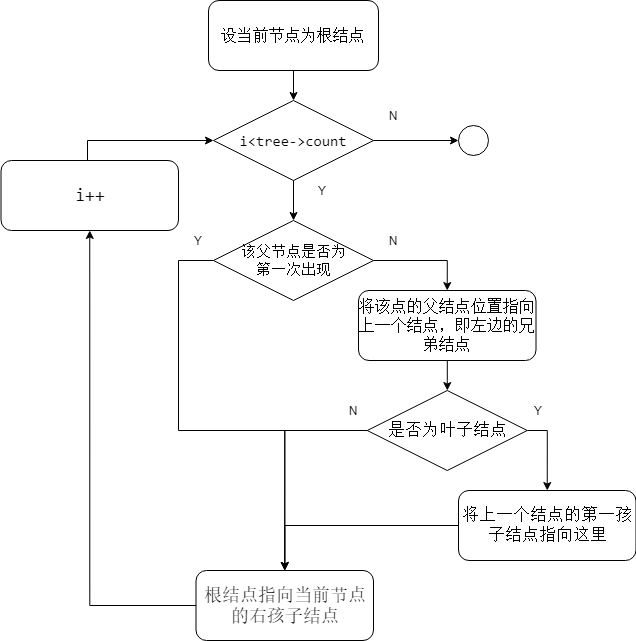
1. /\* 树转二叉树的相关函数 \*/
2. void TreeToBTree(Tree\* tree);
3. BinaryTree\* ForestToBinaryTree(Tree\* tree[], int count);

### 函数分析

#### 树转二叉树

遍历整数树结点，从根结点下的第一个结点开始，如果新出现的一个父结点，从第二次出现该父节点开始依次将后面的父节点指向前面的下标，直到出现新的父节点。如果遇到叶子结点，就将上一个结点的第一孩子结点指向这个结点。

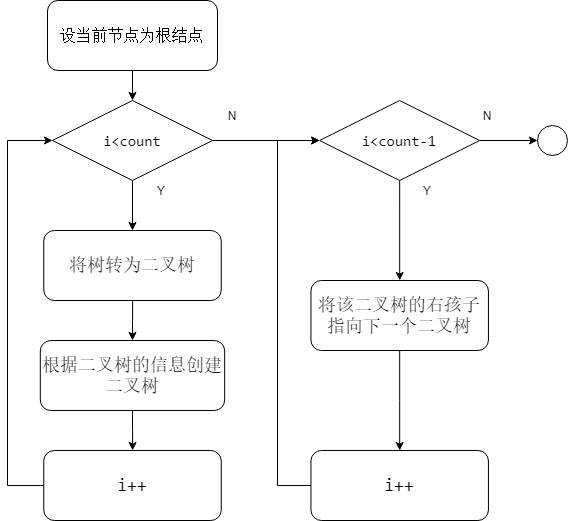
1. void TreeToBTree(Tree\* tree) {
2. int i;
3. int index = 0;
4. bool change = false;
5. int nodeCount;
6. for (i = 1; i < tree->count; i++) {
7. if (index == tree->nodes[i].parent) {
8. // 更改第二次及以后出现相同的父结点的结点信息（即除了第一孩子结点）
9. if (change) {
10. // 将该点的父结点位置指向上一个结点，即左边的兄弟结点
11. tree->nodes[i].parent = i-1;
12. // 如果这个点是叶子结点，就将上一个结点的第一孩子结点指向这里
13. if (tree->nodes[i-1].firstchild == -1) {
14. tree->nodes[i-1].firstchild = i;
15. }
16. } else {
17. change = true;
18. }
19. } else {
20. index = tree->nodes[i].parent;
21. }
22. }
23. }



#### 森林转二叉树

遍历森林结构，即树的数组。将每个数都转换为二叉树，并创建二叉树结构，存入二叉树数组。在将每个二叉树根节点的右子树指向下一个二叉树的根节点，生成一颗二叉树。并返回第一个二叉树结构。

1. BinaryTree\* ForestToBinaryTree(Tree\* tree[], int count) {
2. BinaryTree\* Btree[count];
3. for (int i = 0; i < count; i++) {
4. //ShowTree(tree[i]);
5. TreeToBTree(tree[i]);
6. //ShowTree(tree[i]);
7. Btree[i] = CreateTree(tree[i]);
8. }
9. for (int i = 0; i < count-1; i++) {
10. Btree[i]->Rchild = Btree[i+1];
11. }
12. return Btree[0];
13. }



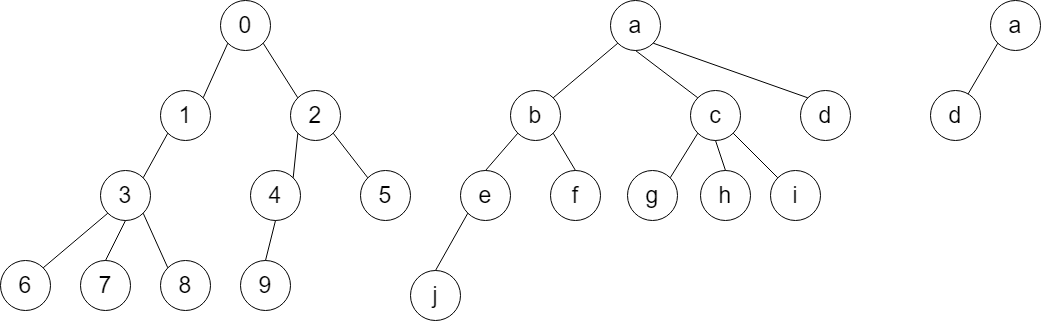
# 调试

## 测试数据

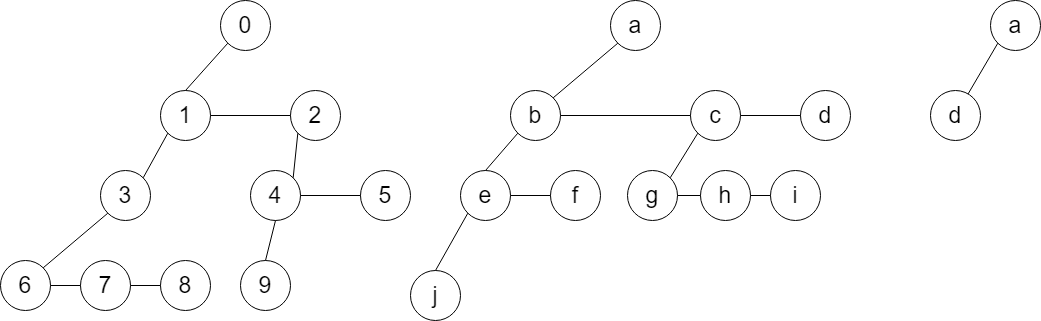
通过三个树测试。

1. char dataInfo1[] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'};
2. int parentInfo1[] = {-1, 0, 0, 1, 2, 2, 3, 3, 3, 4};
4. char dataInfo2[] = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j'};
5. int parentInfo2[] = {-1, 0, 0, 0, 2, 2, 3, 3, 3, 4};
7. char dataInfo3[] = {'a', 'b'};
8. int parentInfo3[] = {-1, 0};

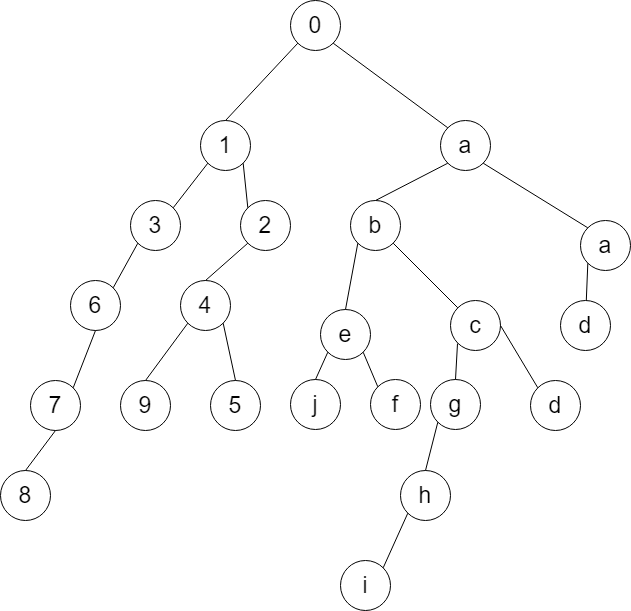
创建出来的树的结构：



树转二叉树后的结构：



组合二叉树达到森林转二叉树的效果：



## 测试结果

