

# DS–Autumn 2023 — Homework 3Solutions

王子隆, SID 2221411126

2023 年 11 月 3 日

## 摘要

二叉树实现及应用  
面向过程面向对象  
定义  
二叉树遍历（九种方法递归非递归 \* 2 \* 3  
preorder inorder postorder  
复杂度时间空间  
最坏情况最好情况最废空间最省空间  
二叉树的主要性质  
语法制导编辑器  
修改数据结构（不借助外力 stack queue  
三叉链表利用空指针（增加标识  
哈夫曼树定义性质应用  
二叉树分类  
BST  
ASL 平均查找长度

## 1. 二叉树实现

(a) 功能实现

(b) 应用

### 1.1 面向过程

**Solutions:**

1. 结构体 ArrayDeque.java

```
/** Array based list .  
 *   @author zilong  
 */
```

## 1.2 面向对象

### Solutions:

#### 1. interface BinaryTree.java

```

1      #include <iostream>
2      #include "../hw2/SeqStack.cpp"
3      // #define NULL nullptr
4      #define cin std::cin
5      #define cout std::cout
6      #define endl std::endl
7
8
9      template <class T>
10     class BinaryTreeNode{
11     private:
12         T data;
13         BinaryTreeNode<T>* left;
14         BinaryTreeNode<T>* right;
15
16
17     public:
18         BinaryTreeNode();
19         BinaryTreeNode(const T& elem);
20         BinaryTreeNode(const T& elem, BinaryTreeNode<T>* l,
21             BinaryTreeNode<T>* r);
22         ~BinaryTreeNode(){};
23         T value() const;
24         BinaryTreeNode<T>* leftchild() const;
25         BinaryTreeNode<T>* rightchild() const;
26         void setLeftchild(BinaryTreeNode<T>*);
27         void setRightchild(BinaryTreeNode<T>*);
28         void setValue(const T& val);
29         bool isLeaf() const;
30     };
31
32
33     template<class T>
34     BinaryTreeNode<T>::BinaryTreeNode(){
35         data = T();
36         left = nullptr;
37         right = nullptr;
38     }
39
40     template<class T>

```

```
41     BinaryTreeNode<T>::BinaryTreeNode(const T& elem){
42         data = elem;
43         left = nullptr;
44         right = nullptr;
45     }
46
47     template<class T>
48     BinaryTreeNode<T>::BinaryTreeNode(const T& elem, BinaryTreeNode<T>* l,
49         BinaryTreeNode<T>* r){
50         data = elem;
51         left = l;
52         right = r;
53     }
54
55     template<class T>
56     T BinaryTreeNode<T>::value() const{
57         return this->data;
58     }
59
60     template<class T>
61     BinaryTreeNode<T>* BinaryTreeNode<T>::leftchild() const{
62         return this->left;
63     }
64
65     template<class T>
66     BinaryTreeNode<T>* BinaryTreeNode<T>::rightchild() const{
67         return this->right;
68     }
69
70     template<class T>
71     void BinaryTreeNode<T>::setLeftchild(BinaryTreeNode<T>* Lchild){
72         this->left = Lchild;
73     }
74
75     template<class T>
76     void BinaryTreeNode<T>::setRightchild(BinaryTreeNode<T>* Rchild){
77         this->right = Rchild;
78     }
79
80     template<class T>
81     void BinaryTreeNode<T>::setValue(const T& Value){
82         this->data = Value;
83     }
84
85     template<class T>
```

```

86     bool BinaryTreeNode<T>::isLeaf() const{
87         return false;
88     }
89
90
91     enum Tag{L, R, M};
92     template <class T>
93     class StackNode{
94     public:
95         BinaryTreeNode<T>* pointer;
96         Tag tag;
97         StackNode() {pointer = nullptr; tag = L;}
98         StackNode(BinaryTreeNode<T>* ptr, Tag tg){pointer = ptr; tag =tg;}
99     };
100
101
102
103
104     template <class T>
105     class BinaryTree{
106     protected:
107         BinaryTreeNode<T>* root;
108     public:
109         BinaryTree() {root = nullptr;}
110         BinaryTree(BinaryTreeNode<T>* r) {root = r;}
111         ~BinaryTree() { DeleteBinaryTree(root); };
112         bool isEmpty() { return root==nullptr; };
113         void visit(const BinaryTree<T>& curr){cout << curr.root->value() << "
114             ";}
115         BinaryTreeNode<T>* Root() {return root;};
116         void CreateTree(const T& data, BinaryTreeNode<T>* lefttree,
117             BinaryTreeNode<T>* righttree);
118         void CreateTree(BinaryTreeNode<T> *r);
119         void DeleteBinaryTree(BinaryTreeNode<T>* root);
120
121         void PreOrder(BinaryTreeNode<T>* root);
122         void InOrder(BinaryTreeNode<T>* root);
123         void PostOrder(BinaryTreeNode<T>* root);
124         void PreOrderLikeRecursion(BinaryTreeNode<T>* root);
125         void InOrderLikeRecursion(BinaryTreeNode<T>* root);
126         void PostOrderLikeRecursion(BinaryTreeNode<T>* root);
127         void PreOrderWithoutRecursion(BinaryTreeNode<T>* root);
128         void InOrderWithoutRecursion(BinaryTreeNode<T>* root);
129         void PostOrderWithoutRecursion(BinaryTreeNode<T>* root);
130         void LevelOrder(BinaryTreeNode<T>* root);
131     };

```

```
130
131
132     template<class T>
133     void BinaryTree<T>::CreateTree(const T& data, BinaryTreeNode<T>*
        leftTree, BinaryTreeNode<T>* rightTree) {
134         root = new BinaryTreeNode<T>(data, leftTree, rightTree);
135         BinaryTree(root);
136     }
137
138
139
140
141     template<class T>
142     void BinaryTree<T>::DeleteBinaryTree(BinaryTreeNode<T>* Root){
143         if (Root != NULL) {
144             DeleteBinaryTree(Root->leftchild());
145             DeleteBinaryTree(Root->rightchild());
146             Root->~BinaryTreeNode();
147         }
148     }
```

## 2. 二叉树遍历

(a) 二叉树遍历（九种方法递归非递归 \* 2 \* 3）

preorder inorder postorder

复杂度时间空间

最坏情况最好情况最废空间最省空间

### preorder

#### recursive

**Solutions:** 遍历指走一遍只走一遍递归算法的关键——写好两个条件 1. 递归条件 2. 结束条件

```

1  template<class T>
2  void BinaryTree<T>::PreOrder(BinaryTreeNode<T>* root){
3      if (root == NULL) return;
4      visit(root);
5      PreOrder(root->leftchild());
6      PreOrder(root->rightchild());
7  }
```

#### nonrecursive

1. recursivelylike

**Solutions:**

```

1  template<class T>
2  void BinaryTree<T>::PreOrderLikeRecusion(BinaryTreeNode <T> * root){
3      SeqStack<BinaryTreeNode<T>*> tStack(10);
4      BinaryTreeNode<T>* pointer = root;
5      while(!tStack.IsEmpty() || pointer){
6          if (pointer){
7              visit(pointer);
8              tStack.Push(pointer);
9              pointer = pointer->leftchild();
10         } else{
11             pointer = tStack.Pop();
12             //tStack.Pop();
13             pointer = pointer->rightchild(); }
14     }
15 }
```

2. other way

**Solutions:**

```

1  template<class T>
2  void BinaryTree<T>::PreOrderWithoutRecursion(BinaryTreeNode <T> * root){
3      SeqStack<BinaryTreeNode<T>*> tStack(10);
4      BinaryTreeNode<T>* pointer = root;
5      if (!pointer) {return;};
6      tStack.Push(pointer);
7      while(!tStack.IsEmpty()){ //use the feature of stack
8          pointer = tStack.Pop();
9          visit(pointer);
10         if(pointer->rightchild()){
11             tStack.Push(pointer->rightchild()); // first in then out
12         }
13         if (pointer->leftchild()){
14             tStack.Push(pointer->leftchild());
15         }
16     }
17     //does queue do the same work?
18 }

```

## inorder

### recursive

#### Solutions:

```

1  template<class T>
2  void BinaryTree<T>::InOrder(BinaryTreeNode<T>* root){
3      if (root == NULL) return;
4      InOrder(root->leftchild());
5      visit(root);
6      InOrder(root->rightchild());
7  }

```

### nonrecursive

#### 1. recursivelylike

与前序指针行进类似，访问次序不同

#### Solutions:

```

1  template<class T>
2  void BinaryTree<T>::InOrderLikeRecursion(BinaryTreeNode <T> * root){
3      SeqStack<BinaryTreeNode<T>*> tStack(10);
4      BinaryTreeNode<T>* pointer = root;
5      while(!tStack.IsEmpty() || pointer){
6          if (pointer){

```

```

7         tStack.Push(pointer);
8         pointer = pointer->leftchild();
9     } else{
10        pointer = tStack.Pop();
11        visit(pointer);
12        //tStack.Push(pointer);
13        //tStack.Pop();
14        pointer = pointer->rightchild(); }
15    }
16 }

```

## 2. other way

### Solutions:

```

1  template<class T>
2  void BinaryTree<T>::InOrderWithoutRecusion(BinaryTreeNode <T> * root){
3      SeqStack<StackNode<T>*> tStack(10);
4      //BinaryTreeNode<T>* pointer = root;
5      StackNode<T>* Tagptr = new StackNode<T>();
6      Tagptr->pointer = root;
7
8      if (!Tagptr->pointer) {return;};
9      tStack.Push(Tagptr);
10     while(!tStack.IsEmpty()){
11         Tagptr = tStack.Pop();
12         if (Tagptr->tag == L) {
13             if(Tagptr->pointer->rightchild()){
14                 StackNode<T>* tem = new
15                     StackNode<T>(Tagptr->pointer->rightchild(), L);
16                 tStack.Push(tem);
17             }
18             StackNode<T>* tem = new StackNode<T>(Tagptr->pointer, R);
19             tStack.Push(tem);
20             if (Tagptr->pointer->leftchild()){
21                 StackNode<T>* tem = new
22                     StackNode<T>(Tagptr->pointer->leftchild(), L);
23                 tStack.Push(tem);
24             }
25         } else visit(Tagptr->pointer);
26     }
27 }

```



**postorder****recursive****Solutions:**

```

1      template<class T>
2 void BinaryTree<T>::PostOrder(BinaryTreeNode<T>* root){
3     if (root == NULL) return;
4     PostOrder(root->leftchild());
5     PostOrder(root->rightchild());
6     visit(root);
7 }

```

**nonrecursive**

## 1. recursivelylike

**Solutions:**

```

1  template<class T>
2 void BinaryTree<T>::PostOrderLikeRecursion(BinaryTreeNode <T> * root){
3     SeqStack<StackNode<T>* > tStack(10);
4     StackNode<T>* Tagptr = new StackNode<T>(root, L);
5
6     while (!tStack.IsEmpty() || Tagptr->pointer != nullptr ){
7         while (Tagptr->pointer != nullptr) {
8             StackNode<T>* tem = new StackNode<T>(Tagptr->pointer, L);
9             tStack.Push(tem);
10            Tagptr->pointer = Tagptr->pointer->leftchild();
11        }
12        if (!tStack.IsEmpty()) {
13            Tagptr = tStack.Pop();
14            if (Tagptr->tag == L ) {
15                StackNode<T>* tem = new StackNode<T>(Tagptr->pointer, R);
16                tStack.Push(tem);
17                Tagptr->pointer = Tagptr->pointer->rightchild();
18            } else {
19                visit(Tagptr->pointer);
20                Tagptr->pointer = nullptr;
21            }
22        }
23    }
24 }

```

## 2. other way

**Solutions:**

```

1  template<class T>
2  void BinaryTree<T>::PostOrderWithoutRecursion(BinaryTreeNode <T> * root){
3      SeqStack<StackNode<T>*> tStack(10);
4      //BinaryTreeNode<T>* pointer = root;
5      StackNode<T>* Tagptr = new StackNode<T>();
6      Tagptr->pointer = root;
7
8      if (!Tagptr->pointer) {return;};
9      tStack.Push(Tagptr);
10     while(!tStack.IsEmpty()){
11         Tagptr = tStack.Pop();
12         if (Tagptr->tag == L) {
13             StackNode<T>* tem = new StackNode<T>(Tagptr->pointer, R);
14             tStack.Push(tem);
15             if (Tagptr->pointer->rightchild()){
16                 StackNode<T>* tem = new
17                     StackNode<T>(Tagptr->pointer->rightchild(), L);
18                 tStack.Push(tem);
19             }
20             if (Tagptr->pointer->leftchild()){
21                 StackNode<T>* tem = new StackNode<T>
22                     (Tagptr->pointer->leftchild(), L);
23                 tStack.Push(tem);
24             }
25         } else if (Tagptr->tag == R) {
26             StackNode<T>* tem = new StackNode<T>(Tagptr->pointer, M);
27             tStack.Push(tem);
28         } else {
29             visit(Tagptr->pointer);
30         }
31     }
32 }

```

## 复杂度分析

### 1. 复杂度时间空间

$n$  为节点数每个节点都在栈里走了两遍故为 $2n$

空间为栈的长度，树的高度

traversalMethod	Time	Space
PreOrder	$2N$	$height$
InOrder	$2N$	$height$
PostOrder	$3N$	$\log_2 N$

### 2. 最坏情况最好情况

traversalMethod	Bestcase	worstcase
PreOrder	only rightchild	only leftchild
InOrder	only rightchild	only leftchild
PostOrder	FBT with least height	only leftchild

## 层次遍历

### 1. 借助外力 stack queue 逻辑

```

1
2  template<class T>
3  void BinaryTree<T>::LevelOrder(BinaryTreeNode <T> * root){
4      SeqQueue<BinaryTreeNode<T>*> Queue(16);
5      Queue.InQueue(root);
6      while(!Queue.IsEmpty()) {
7          //cout << "Starting Level Order Traversal..." << endl;
8          BinaryTreeNode<T>* pointer = Queue.OutQueue();
9          if (pointer->leftchild()){
10             Queue.InQueue(pointer->leftchild());
11         }
12         if (pointer->rightchild()){
13             Queue.InQueue(pointer->rightchild());
14         }
15         visit(pointer);
16     }
17 }

```

### 3. 二叉树的定义、主要性质、定理

- (a) 递归定义及基本术语
- (b) 分类顺序存储链式存储
- (c) 性质\*5

#### definition

##### Solutions:

1. a recursive definition

#### category

##### Solutions:

1. sequential structure
2. list structure

#### properties quality character

##### Solutions:

1.  $i \leq 2^{i-1}$  floor
2.  $k \leq 2^k - 1$  all the tree
3.  $n_0 = n_2 + 1$
4.  $\text{height} = \lceil \log_2 n \rceil + 1$
5. structure of root, Lchild, Rchild

## 4.证明

YOUR ANSWER GOES HERE

something to prove

**Solutions:**

1. 给定tree 遍历序列唯一给定位置唯一存在（顺序唯一  
idea: 给定位置唯一存在
2. 中序遍历猜想 + 先序猜想
3. 遍历时的性质\*5
4.  $n$  node  $2*n$  pointer  $N-1$  in use

## 5.修改数据结构

YOUR ANSWER GOES HERE

- (a) 利用空指针（增加标识
- (b) 三叉链表

### 1.1 利用空指针

**Solutions:**

1. 结构体 ArrayDeque.java

```
/** Array based list .  
 *   @author zilong  
 */
```

### 1.2 三叉链表

**Solutions:**

1. interface List.java

```
public interface List<Item>
```

## 6.Huffman tree

- (a) 递归定义及基本术语
- (b) 分类顺序存储链式存储
- (c) 应用\*4

### definition

#### Solutions:

1. a recursive definition

### category

#### Solutions:

1. sequential structure
2. list structure

### properties quality character

#### Solutions:

1.  $i \leq 2^{i-1}$  floor
2.  $k \leq 2^k - 1$  all the tree
3.  $n_0 = n_2 + 1$
4.  $\text{height} = \lceil \log_2 n \rceil + 1$
5. structure of root, Lchild, Rchild

## 7.Binary searching/sorting Tree



## 8.BinaryTree categories

## 9.BinaryTree in java