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

```
#include <iostream>
#include "../hw2/SeqStack.cpp"
//#define NULL nullptr
#define cin std::cin
#define cout std::cout
#define endl std::endl


template <class T>
class BinaryTreeNode{
private:
    T data;
    BinaryTreeNode<T>*  left;
    BinaryTreeNode<T>*  right;


public:
    BinaryTreeNode();
    BinaryTreeNode(const T& elem);
    BinaryTreeNode(const T& elem, BinaryTreeNode<T>* l,
        BinaryTreeNode<T>* r);
    ~BinaryTreeNode(){};
    T value() const;
    BinaryTreeNode<T>*  leftchild() const;
    BinaryTreeNode<T>*  rightchild() const;
    void  setLeftchild(BinaryTreeNode<T>*);
    void  setRightchild(BinaryTreeNode<T>*);
    void  setValue(const T& val);
    bool  isLeaf() const;
};



template<class T>
BinaryTreeNode<T>::BinaryTreeNode(){
    data = T();
    left = nullptr;
    right = nullptr;
}

template<class T>
```

```
    BinaryTreeNode<T>::BinaryTreeNode(const T& elem){
        data = elem;
        left = nullptr;
        right = nullptr;
    }

    template<class T>
    BinaryTreeNode<T>::BinaryTreeNode(const T& elem, BinaryTreeNode<T>* l,
        BinaryTreeNode<T>* r){
        data = elem;
        left = l;
        right = r;
    }


    template<class T>
    T BinaryTreeNode<T>::value() const{
        return this->data;
    }

    template<class T>
    BinaryTreeNode<T>*  BinaryTreeNode<T>::leftchild() const{
        return this->left;
    }

    template<class T>
    BinaryTreeNode<T>*  BinaryTreeNode<T>::rightchild() const{
        return this->right;
    }

    template<class T>
    void BinaryTreeNode<T>::setLeftchild(BinaryTreeNode<T>* Lchild){
        this->left = Lchild;
    }

    template<class T>
    void BinaryTreeNode<T>::setRightchild(BinaryTreeNode<T>* Rchild){
        this->right = Rchild;
    }

    template<class T>
    void BinaryTreeNode<T>::setValue(const T& Value){
        this->data = Value;
    }

    template<class T>
```

```cpp
bool BinaryTreeNode<T>::isLeaf() const{
    return false;
}


enum Tag{L, R, M};
template <class T>
class StackNode{
public:
    BinaryTreeNode<T>* pointer;
    Tag tag;
    StackNode() {pointer = nullptr; tag = L;}
    StackNode(BinaryTreeNode<T>* ptr,Tag tg){pointer = ptr; tag =tg;}
};




template <class T>
class BinaryTree{
protected:
    BinaryTreeNode<T>* root;
public:
    BinaryTree() {root = nullptr;}
    BinaryTree(BinaryTreeNode<T>* r) {root = r;}
    ~BinaryTree() { DeleteBinaryTree(root); };
    bool isEmpty() { return root==nullptr; };
    void visit(const BinaryTree<T>& curr){cout << curr.root->value() << " 
        ";}
    BinaryTreeNode<T>*& Root() {return root;};
    void CreateTree(const T& data, BinaryTreeNode<T>* lefttree,
        BinaryTreeNode<T>* righttree);
    void CreateTree(BinaryTreeNode<T> *&r);
    void DeleteBinaryTree(BinaryTreeNode<T>* root);

    void PreOrder(BinaryTreeNode<T>* root);
    void InOrder(BinaryTreeNode<T>* root);
    void PostOrder(BinaryTreeNode<T>* root);
    void PreOrderLikeRecusion(BinaryTreeNode<T>* root);
    void InOrderLikeRecusion(BinaryTreeNode<T>* root);
    void PostOrderLikeRecusion(BinaryTreeNode<T>* root);
    void PreOrderWithoutRecusion(BinaryTreeNode<T>* root);
    void InOrderWithoutRecusion(BinaryTreeNode<T>* root);
    void PostOrderWithoutRecusion(BinaryTreeNode<T>* root);
    void LevelOrder(BinaryTreeNode<T>* root);
};
```

```
template<class T>
void BinaryTree<T>::CreateTree(const T& data, BinaryTreeNode<T>*
    leftTree, BinaryTreeNode<T>* rightTree) {
    root = new BinaryTreeNode<T>(data, leftTree, rightTree);
    BinaryTree(root);
}



template<class T>
void BinaryTree<T>::DeleteBinaryTree(BinaryTreeNode<T>* Root){
    if (Root != NULL) {
        DeleteBinaryTree(Root->leftchild());
        DeleteBinaryTree(Root->rightchild());
        Root->~BinaryTreeNode();
    }
}
```

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

```
template<class T>
void BinaryTree<T>::PreOrder(BinaryTreeNode<T>* root){
    if (root == NULL)  return;
    visit(root);
    PreOrder(root->leftchild());
    PreOrder(root->rightchild());
}
```

nonrecursive

1. recursivelike

   Solutions:

```
template<class T>
void BinaryTree<T>::PreOrderLikeRecusion(BinaryTreeNode <T> * root){
    SeqStack<BinaryTreeNode<T>*> tStack(10);
    BinaryTreeNode<T>* pointer = root;
    while(!tStack.IsEmpty() || pointer){
        if (pointer){
            visit(pointer);
            tStack.Push(pointer);
            pointer = pointer->leftchild();
        } else{
            pointer = tStack.Pop();
            //tStack.Pop();
            pointer = pointer->rightchild(); }
    }
}
```

2. other way

   Solutions:

```
template<class T>
void BinaryTree<T>::PreOrderWithoutRecusion(BinaryTreeNode <T> * root){
    SeqStack<BinaryTreeNode<T>*> tStack(10);
    BinaryTreeNode<T>* pointer = root;
    if (!pointer) {return;};
    tStack.Push(pointer);
    while(!tStack.IsEmpty()){  //use the feature of stack
        pointer = tStack.Pop();
        visit(pointer);
        if(pointer->rightchild()){
            tStack.Push(pointer->rightchild());// first in then out
        }
        if (pointer->leftchild()){
            tStack.Push(pointer->leftchild());
        }
    }
    //does queue do the same work?
}
```

inorder

recursive

Solutions:

```
template<class T>
void BinaryTree<T>::InOrder(BinaryTreeNode<T>* root){
    if (root == NULL) return;
    InOrder(root->leftchild());
    visit(root);
    InOrder(root->rightchild());
}
```

nonrecursive

1. recursivelike

   与前序指针行进类似，访问次序不同

   Solutions:

```
template<class T>
void BinaryTree<T>::InOrderLikeRecusion(BinaryTreeNode <T> * root){
    SeqStack<BinaryTreeNode<T>*> tStack(10);
    BinaryTreeNode<T>* pointer = root;
    while(!tStack.IsEmpty() || pointer){
        if (pointer){
```

```
            tStack.Push(pointer);
            pointer = pointer->leftchild();
        } else{
            pointer = tStack.Pop();
            visit(pointer);
            //tStack.Push(pointer);
            //tStack.Pop();
            pointer = pointer->rightchild(); }
    }
}
```

2. other way

   Solutions:

```
template<class T>
void BinaryTree<T>::InOrderWithoutRecusion(BinaryTreeNode <T> * root){
    SeqStack<StackNode<T>*> tStack(10);
    //BinaryTreeNode<T>* pointer = root;
    StackNode<T>* Tagptr = new StackNode<T>();
    Tagptr->pointer = root;

    if (!Tagptr->pointer) {return;};
    tStack.Push(Tagptr);
    while(!tStack.IsEmpty()){
        Tagptr = tStack.Pop();
        if (Tagptr->tag == L) {
            if(Tagptr->pointer->rightchild()){
                StackNode<T>* tem = new
                    StackNode<T>(Tagptr->pointer->rightchild(), L);
                tStack.Push(tem);
            }
            StackNode<T>* tem = new StackNode<T>(Tagptr->pointer, R);
            tStack.Push(tem);
            if (Tagptr->pointer->leftchild()){
                StackNode<T>* tem = new
                    StackNode<T>(Tagptr->pointer->leftchild(), L);
                tStack.Push(tem);
            }
        } else visit(Tagptr->pointer);
    }
}
```

postorder

recursive

Solutions:

```
        template<class T>
void BinaryTree<T>::PostOrder(BinaryTreeNode<T>* root){
    if (root == NULL) return;
    PostOrder(root->leftchild());
    PostOrder(root->rightchild());
    visit(root);
}
```

nonrecursive

1. recursivelike

   Solutions:

```
template<class T>
void BinaryTree<T>::PostOrderLikeRecusion(BinaryTreeNode <T> * root){
    SeqStack<StackNode<T>* > tStack(10);
    StackNode<T>* Tagptr = new StackNode<T>(root, L);

    while (!tStack.IsEmpty() || Tagptr->pointer != nullptr ){
        while (Tagptr->pointer != nullptr) {
            StackNode<T>* tem = new StackNode<T>(Tagptr->pointer, L);
            tStack.Push(tem);
            Tagptr->pointer = Tagptr->pointer->leftchild();
        }
        if (!tStack.IsEmpty()) {
            Tagptr = tStack.Pop();
            if (Tagptr->tag == L ) {
                StackNode<T>* tem = new StackNode<T>(Tagptr->pointer, R);
                tStack.Push(tem);
                Tagptr->pointer = Tagptr->pointer->rightchild();
            } else {
                visit(Tagptr->pointer);
                Tagptr->pointer = nullptr;
            }
        }
    }
}
```

2. other way

   Solutions:

```cpp
template<class T>
void BinaryTree<T>::PostOrderWithoutRecusion(BinaryTreeNode <T> * root){
    SeqStack<StackNode<T>*> tStack(10);
    //BinaryTreeNode<T>* pointer = root;
    StackNode<T>* Tagptr = new StackNode<T>();
    Tagptr->pointer = root;

    if (!Tagptr->pointer) {return;};
    tStack.Push(Tagptr);
    while(!tStack.IsEmpty()){
        Tagptr = tStack.Pop();
        if (Tagptr->tag == L) {
            StackNode<T>* tem = new StackNode<T>(Tagptr->pointer, R);
            tStack.Push(tem);
            if (Tagptr->pointer->rightchild()){
                StackNode<T>* tem = new
                    StackNode<T>(Tagptr->pointer->rightchild(), L);
                tStack.Push(tem);
            }
            if (Tagptr->pointer->leftchild()){
                StackNode<T>* tem = new StackNode<T>
                    (Tagptr->pointer->leftchild(), L);
                tStack.Push(tem);
            }
        } else if (Tagptr->tag == R) {
            StackNode<T>* tem = new StackNode<T>(Tagptr->pointer, M);
            tStack.Push(tem);
        } else {
            visit(Tagptr->pointer);
        }
    }
}
```

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

```

template<class T>
void BinaryTree<T>::LevelOrder(BinaryTreeNode <T> * root){
    SeqQueue<BinaryTreeNode<T>*> Queue(16);
    Queue.InQueue(root);
    while(!Queue.IsEmpty()) {
        //cout << "Starting Level Order Traversal..." << endl;
        BinaryTreeNode<T>* pointer = Queue.OutQueue();
        if (pointer->leftchild()){
            Queue.InQueue(pointer->leftchild());
        }
        if (pointer->rightchild()){
            Queue.InQueue(pointer->rightchild());
        }
        visit(pointer);
    }
}
```

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

7.Binary searching/sorting Tree

8.BinaryTree categories

9.BinaryTree in java