DS–Autumn 2023 — Homework 3Solutions

王子隆, SID 2221411126

2023 年 10 月 24 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 = nullptr;
    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>
```

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





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(BinaryTree<T>& curr){cout <<curr->data << "␣";}
    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 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 = BinaryTreeNode<T>(data, leftTree, rightTree);
    BinaryTree(root);
}




template<class T>
void BinaryTree<T>::DeleteBinaryTree(BinaryTreeNode<T>* Root){
```

```
        if (Root != NULL) {
            DeleteBinaryTree(Root->left);
            DeleteBinaryTree(Root->right);
            delete Root;
        }
    }

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


    template<class T>
    void BinaryTree<T>::PreOrderWithoutRecusion(BinaryTreeNode <T> * root){
        SeqStack<BinaryTreeNode<T>* > tStack;
        BinaryTreeNode<T>* pointer = root;
        while(!tStack.empty() || pointer){
            if (pointer){
                visit(pointer->value());
                tStack.push(pointer);
                pointer = pointer->leftchild();
            } else{
                pointer = tStack.top();
                tStack.pop();
                pointer = pointer->rightchild(); }
        }
    }
```

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:

nonrecursive

1. recursivelike

   Solutions:

2. other way

   Solutions:

inorder

recursive

Solutions:

nonrecursive

1. recursivelike

   Solutions:

2. other way

   Solutions:

postorder

recursive

Solutions:

nonrecursive

1. recursivelike

   Solutions:

2. other way

   Solutions:

复杂度分析

1. 复杂度时间空间
2. 最坏情况最好情况

层次遍历

1. 复杂度时间空间
2. 借助外力 stack queue 逻辑

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
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