数据结构与算法
实验报告

第 三次

姓名	郭一阳
班级	软件 001 班
学号	2203312559
电话	18157631028
Email	1041357882@qq.com
日期	2021-11-20

目录

实验 1

1、 题目:

表达式树

2、 数据结构设计

3、
```
public BinNodePtr<Character> treeRootNode;
public BinNodePtr<Character>[] array=new BinNodePtr[30];
public Stack<Integer> stack=new Stack<>();//存储计算结果
public String result="";//存储树转化而来的后缀表达式
```

BinNodePtr 为自定义节点

```
public class BinNodePtr<T> {
    private T element;
    private int priority;
    private BinNodePtr<T> father;
    private BinNodePtr<T> left;
    private BinNodePtr<T> right;

    public BinNodePtr<T> father() {
        return father;
    }

    public void setFather(BinNodePtr<T> father) {
        this.father = father;
    }
    public BinNodePtr(){
        left=right=null;
    }

    public BinNodePtr(T val){
```

```
        left=right=null;
        element=val;
    }

    public BinNodePtr(T val,BinNodePtr left,BinNodePtr right){
        element=val;
        this.left=left;
        this.right=right;
    }

    public void setPriority(int a){
        priority=a;
    }

    public int getPriority(){
        return priority;
    }

    public T element() {
        return element;
    }


    public T setElement(T v) {
        return element=v;
    }


    public BinNodePtr<T> left() {
        return left;
    }


    public BinNodePtr<T> setLeft(BinNodePtr<T> p) {
        return left=p;
    }


    public BinNodePtr<T> right() {
        return right;
    }


    public BinNodePtr<T> setRight(BinNodePtr<T> p) {
        return right=p;
```

```
    }

    public boolean isLeaf() {
        return (left==null)&&(right==null);
    }
}
```

4、算法设计

任务 2 可采用老师所给的附件大众的方法建立二叉树；任务 3 可采用后序遍历二叉树，转化为周追表达式；任务 4 可采用层序遍历；任务 5 关键在于将前缀表达式转化为后缀表达式的算法

5、主干代码说明

任务二：

```
BinNodePtr<Character> readPostBinaryTree(String expression){
    int index=0;
    int sIndex=0;
    while(sIndex<expression.length()){
        array[index]=new BinNodePtr<>();
        char c=expression.charAt(sIndex);
        if(c>='0'&&c<='9'){
            array[index]=new BinNodePtr<Character>(c);
            index++;
        }
        else{
            BinNodePtr<Character> node=new BinNodePtr<>(c,array[index-
2],array[index-1]);
            index=index-2;
            array[index]=node;
            index++;
        }
        sIndex++;
    }
    return array[0];
}
```

任务三：

```
//辅助方法
 void postOrderResultHelp(BinNodePtr<Character> node){
    if(node.left()==null&&node.right()==null){
        result+=node.element()+" ";
        stack.push(Integer.parseInt(""+node.element()));
    }
    else{
        postOrderResultHelp(node.left());
        postOrderResultHelp(node.right());
        result+=node.element()+" ";
        int b=stack.pop();
        int a=stack.pop();
        stack.push(calculate(a,b,node.element()));
    }
}

//输出树转化为的后缀表达式
 void postOrderResult(BinNodePtr<Character> node){
    postOrderResultHelp(node);
    System.out.println(result+"="+stack.peek());
}
```

任务四：

```
//将树 2D 形式输出
void output(){
    int floor=0;
    int kongGe=64;
    BinNodePtr<Character>[] arrayNode=new BinNodePtr[50];
    BinNodePtr<Character>[] arrayNode1=new BinNodePtr[50];
    for (int i = 0; i < 50; i++) {
        arrayNode1[i]=new BinNodePtr<>();
        arrayNode[i]=new BinNodePtr<>();
    }
    arrayNode[0]=treeRootNode;
    int number=0;

    while(number<=10){
        kongGe=kongGe-32/(int)Math.pow(2,floor);
        for (int i = 0; i < kongGe; i++) {
            System.out.print(" ");
        }
```

```
        int index=0;
        for(int i=0;i<(int)(Math.pow(2,floor));i++){
            if(arrayNode[i]==null){
                arrayNode[i]=new BinNodePtr<>('o');
                System.out.print(arrayNode[i].element());
            }
            else
                System.out.print(arrayNode[i].element());
            for(int j=0;j<64/(int)Math.pow(2,floor);j++)
                System.out.print(" ");
            if(arrayNode[i].left()==null&&arrayNode[i].right()==null){
                arrayNode1[index]=null;
                index++;
                arrayNode1[index]=null;
                index++;
            }
            else{
                arrayNode1[index]=arrayNode[i].left();
                index++;
                arrayNode1[index]=arrayNode[i].right();
                index++;
            }
        }

        for (int i = 0; i < 50; i++) {
            if(arrayNode1[i]==null)
                arrayNode[i]=null;
            else
                arrayNode[i]=arrayNode1[i];
        }

        System.out.println();
        floor++;
        number++;
    }

}
```

任务五：

```
BinNodePtr<Character> readInBinaryTree(String expression){
    String result=new Caculator().parse(expression);
    return readPostBinaryTree(result);
}

//辅助方法
```

```
public String parse(String str){
    class Stack<T>{
        private LinkedList<T> list = new LinkedList<T>();

        public void push(T t){
            list.addLast(t);
        }

        public T pop(){
            return list.removeLast();
        }

        public T top(){
            return list.peekLast();
        }

        public boolean isEmpty(){
            return list.isEmpty();
        }
    }

    Stack<Character> stack = new Stack<>();
    StringBuilder sb = new StringBuilder();
    char[] cs = str.toCharArray();

    for(int i=0; i<cs.length; ++i){
        char c = cs[i];
        if( isNumber(c)){
            sb.append(c);

        }else if( isLeftBracket(c)){
            stack.push(c);

        }else if( isOperator(c)){
            //sb.append("'");//分开左右两个数
            while( leftPirorityIsNotLess(stack.top(), c)){
                sb.append( stack.pop());
            }
            stack.push(c);

        }else if( isRightBracket(c)){
            while( !isLeftBracket( stack.top())){
                sb.append(stack.pop());
            }
            stack.pop();
```

```
        }
    }

    while( !stack.isEmpty()){
        sb.append(stack.pop());
    }

    return sb.toString();
```

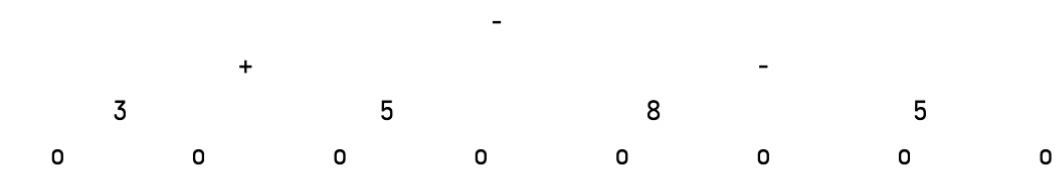
6、运行结果展示

```
CTree tree=new CTree();
//后缀转化为树
tree.treeRootNode= tree.readPostBinaryTree("356-+67+6**");
//树转化为后缀并输出
tree.postOrderResult(tree.treeRootNode);
//输出 2D 图
tree.output();
System.out.println();

//中缀转化为树
tree.treeRootNode=tree.readInBinaryTree("(3+5)-(8-5)");
//树转化为后缀并输出
tree.postOrderResult(tree.treeRootNode);
//输出 2D 图
tree.output();
```

```
3 5 6 - + 6 7 + 6 * * =156
                                *
                +                                       *
        3                -                +                       6
    o       o       5       6       6       7       o       o
```

```
3 5 + 8 5 - - =5
```



实验 2

1、题目：

Treap

2、数据结构设计

bst

```
private BinNodePtr<Integer> rootNode;
```

treap

```
private Node<Integer> root;  // 根节点
private final Node<Integer> nullNode;  // 空节点
```

3、算法设计

插入，删除：关键在于 Treap 的左右旋操作。分多种情况。

求树高：关键在于灵活运用队列数据结构。

搜索：类似于二分查找。

4、主干代码说明

实现插入操作：

Bst:

```
void insertHelp(BinNodePtr<Integer> node,int a){
    if(rootNode==null){
        rootNode=new BinNodePtr<Integer>(a);
    }
    else if(a<node.element()){
        if(node.left()==null)
            node.setLeft(new BinNodePtr<Integer>(a));
        else
            insertHelp(node.left(),a);
    }
    else{
        if(node.right()==null)
            node.setRight(new BinNodePtr<Integer>(a));
        else
            insertHelp(node.right(),a);
    }
}

void insert(int a){
   insertHelp(rootNode,a);
}
```

Treap:

```
private Node<Integer> insert(Integer element, Node<Integer> node) {
    if (node == nullNode) {
        return new Node<>(element, nullNode, nullNode);  // 插入到叶子节
点中
    }
    int compareResult = element.compareTo(node.element);
    if (compareResult < 0) {
        node.left = insert(element, node.left);  // 按二叉查找树的性质插入
```

```
        if (node.left.priority < node.priority) {  // 按priority的堆序性质（小顶堆）进行调整
            node = rotateWithLeftChild(node);
        }
    } else if (compareResult > 0) {
        node.right = insert(element, node.right);
        if (node.right.priority < node.priority) {
            node = rotateWithRightChild(node);
        }
    }
    return node;
}

// 左一字型的单旋转
private Node<Integer> rotateWithLeftChild(Node<Integer> t) {
    Node<Integer> tmp = t.left;
    t.left = tmp.right;
    tmp.right = t;
    return tmp;
}

// 右一字型的单旋转
private Node<Integer> rotateWithRightChild(Node<Integer> t) {
    Node<Integer> tmp = t.right;
    t.right = tmp.left;
    tmp.left = t;
    return tmp;
}
```

实现删除操作：

Bst:

```
int getMin(BinNodePtr<Integer> node){
    if(node.left()==null)
        return node.element();
    else
        return getMin(node.left());
}

BinNodePtr<Integer> deleteMin(BinNodePtr<Integer> node){
    if(node.left()==null)
        return node.right();
    else{
```

```
        node.setLeft(deleteMin(node.left()));
        return node;
    }
}

BinNodePtr<Integer> removeHelp(int a,BinNodePtr<Integer> node){
    if(node==null) return null;

    if(a<node.element())
        node.setLeft(removeHelp(a,node.left()));
    else if(a>node.element())
        node.setRight(removeHelp(a,node.right()));
    else {
        if(node.left()==null)
            node=node.right();
        else if(node.right()==null)
            node=node.left();
        else{
            node.setElement(getMin(node.right()));
            node.setRight(deleteMin(node.right()));
        }
    }
    return node;
}
void remove(int a){
    removeHelp(a,rootNode);
}
```

Treap:

```
// 左一字型的单旋转
private Node<Integer> rotateWithLeftChild(Node<Integer> t) {
    Node<Integer> tmp = t.left;
    t.left = tmp.right;
    tmp.right = t;
    return tmp;
}

// 右一字型的单旋转
private Node<Integer> rotateWithRightChild(Node<Integer> t) {
    Node<Integer> tmp = t.right;
    t.right = tmp.left;
```

```
private Node<Integer> remove(Integer element, Node<Integer> node) {
    if (node != nullNode) {
        int compareResult = element.compareTo(node.element);
        if (compareResult < 0) {
            node.left = remove(element, node.left);
        } else if (compareResult > 0) {
            node.right = remove(element, node.right);
        } else {  // compareResult = 0，即找到对应项
            if (node.left.priority < node.right.priority) {
                node = rotateWithLeftChild(node);
            } else {
                node = rotateWithRightChild(node);
            }
            if (node != nullNode) {
                node = remove(element, node);
            } else {
                node.left = nullNode;
            }
        }
    }
    return node;
}
```

实现搜索操作：

Bst:

```
int searchHelp(BinNodePtr<Integer> node,int a){

    if(a<node.element()){
        if(node.left()==null)
            return -1;
        else
            return searchHelp(node.left(),a);
    }
    else if(a>node.element()){
        if(node.right()==null)
            return -1;
        else
            return searchHelp(node.right(),a);
    }
    else
        return a;
}
```

```
int search(int a){
    return searchHelp(rootNode,a);
}
```

Treap:

和 BST 基本相同

计算树高操作：

Bst:

原本用递归，后来发现如果数据规模很大计算很慢，所以后来借助队列实现

```
int hight1(){
    int hight=0;
    Queue<BinNodePtr<Integer>> queue=new LinkedList<>();
    if(rootNode==null)
        return 0;
    else{
        int numbers=1;
        int numbersNext=0;
        queue.offer(rootNode);
        while(!queue.isEmpty()){
            while(numbers>0){
                if(queue.peek().left()!=null){
                    queue.offer(queue.peek().left());
                    numbersNext++;
                }
                if(queue.peek().right()!=null){
                    queue.offer(queue.peek().right());
                    numbersNext++;
                }
                numbers--;
                queue.poll();
            }
            numbers=numbersNext;
            numbersNext=0;
            hight++;
```

```
        }
    }
    return hight;
}
```

Treap:

和 BST 类似

5、运行结果展示

		有序序列数据			随机序列数据		
		1000	10000	100000	1000	10000	100000
BST	插入所有元素时间总计	17	293	栈溢出	2	8	86
	树高	1000	10000	100000	24	38	153
	查找所有元素时间总计	9	268	栈溢出	1	2	7
	删除所有元素时间总计	5	168	栈溢出	1	3	17
Treap	插入所有元素时间总计	3	5	45	5	16	43
	树高	25	33	44	24	31	40
	查找所有元素时间总计	1	3	14	0	2	4
	删除所有元素时间总计	1	2	20	0	3	4

总结：对于随机数据，数据规模较小时侯 BST 和 TREAP 性能差别不大。数据规模较大时，Treap 可实现更低的树高，更短的查找，删除，插入操作时间。对于有序序列数据，不论数据规模如何，TREAP 性能明显优于 BST。