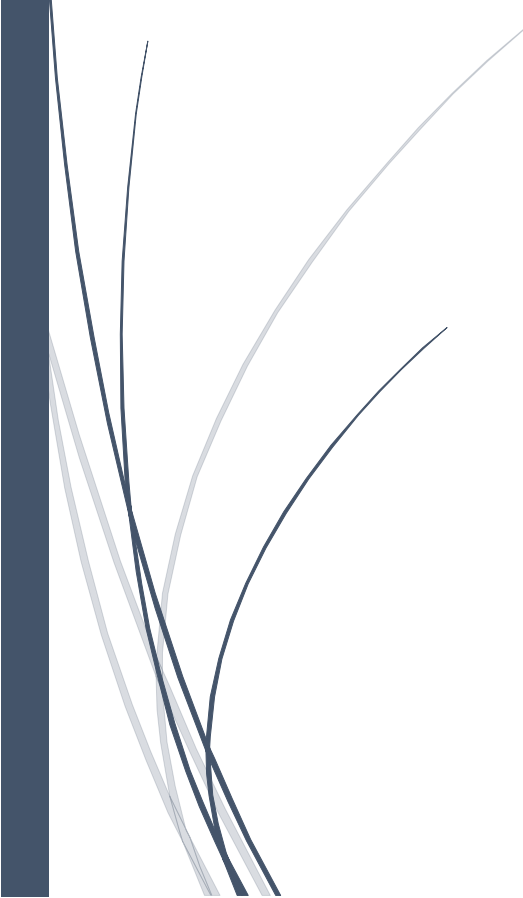


A dark blue vertical bar on the left side of the page, with a blue arrow pointing right from its center.

Assignment2

RedBlackTree

Mahmoud fathy aboeleneen 47
abdelmonem badr elrawy 26
MICROSOFT

Several thin, curved lines in dark blue and light gray originating from the bottom left corner and extending upwards and to the right.

Problem Statement:

A red black tree is a kind of self-balancing binary search tree in computer science. Each node of the binary tree has an extra bit, and that bit is often interpreted as the color (red or black) of the node. These color bits are used to ensure the tree remains approximately balanced during insertions and deletions. Balance is preserved by painting each node of the tree with one of two colors in a way that satisfies certain properties, which collectively constrain how unbalanced the tree can become in the worst case. When the tree is modified, the new tree is subsequently rearranged and repainted to restore the coloring properties. The properties are designed in such a way that this rearranging and recoloring can be performed efficiently.

Assumptions:

- I have a problem with one test(delete random element In tree).

Data structure:

- linkedList in tree map implementation

Code Snapshots:

- Insert:

```
@Override
public void insert(T key, v value) {
    // TODO Auto-generated method stub
    if(key == null || value == null) {
        throw new RuntimeException(new Error());
    }
    INode<T,v> newNode = new Node<T, v>(key,value);
    INode<T,v> curNode = root ;
    INode<T,v> previousNode = NIL ;

    while(!curNode.isNull()) {
        previousNode = curNode;
        if(key.compareTo(curNode.getKey())>0) {
            curNode = curNode.getRightChild();
        }else if(key.compareTo(curNode.getKey())< 0) {
            curNode = curNode.getLeftChild();
        }else {
            curNode.setValue((v) value);
            return;
        }
    }
    newNode.setParent(previousNode);
    if(previousNode.isNull()) {
        this.root=newNode;
    }else if (newNode.getKey().compareTo(previousNode.getKey()) > 0) {
        previousNode.setRightChild(newNode);
    }else {
        previousNode.setLeftChild(newNode);
    }
    newNode.setLeftChild(this.NIL);
    newNode.setRightChild(this.NIL);
    newNode.setColor(true);
    FIX(newNode);
}
```

- Delete:

```

@Override
public boolean delete(T key) {
    // TODO Auto-generated method stub
    if(key==null) {
        throw new RuntimeException(new Error());
    }
    if(! contains(key)) return false;
    INode<T,v> ZNode = getNode(key);
    INode<T,v> XNode ;
    INode<T,v> YNode = ZNode ;
    Boolean YColor = YNode.getColor();
    if(ZNode.getLeftChild().isNull()) {
        XNode=ZNode.getRightChild();
        this.TransPlant(ZNode, ZNode.getRightChild());
    }
    else if (ZNode.getRightChild().isNull()) {
        XNode = ZNode.getLeftChild();
        this.TransPlant(ZNode, ZNode.getLeftChild());
    }else {
        YNode =this.getMinimum(ZNode.getRightChild());
        YColor = YNode.getColor();
        XNode = YNode.getRightChild();
        if(YNode.getParent()==ZNode) {
            XNode.setParent(YNode);
        }else {
            this.TransPlant(YNode, YNode.getRightChild());
            YNode.setRightChild(ZNode.getRightChild());
            YNode.getRightChild().setParent(YNode);
        }
        this.TransPlant(ZNode, YNode);
        YNode.setLeftChild(ZNode.getLeftChild());
        YNode.getLeftChild().setParent(YNode);
        YNode.setColor(ZNode.getColor());
    }
    if(!YColor) {
        FIX_DELETE(XNode);
    }

    return true;
}

```

- Right rotation

```
private void RightRotation(INode<T,v> Xnode) {
    /*if (Xnode.isNull() || Xnode.getLeftChild().isNull()) {
        return;
    }*/
    INode<T,v> Ynode = Xnode.getLeftChild();
    Xnode.setLeftChild(Ynode.getRightChild());
    if(!Ynode.getRightChild().isNull()) {
        Ynode.getRightChild().setParent(Xnode);
    }

    Ynode.setParent(Xnode.getParent());
    if(Xnode.getParent().isNull()) {
        root = Ynode;
    }
    else if(Xnode == Xnode.getParent().getLeftChild()) {
        Xnode.getParent().setLeftChild(Ynode);
    }else {
        Xnode.getParent().setRightChild(Ynode);
    }
    Ynode.setRightChild(Xnode);
    Xnode.setParent(Ynode);
}
```

- Left rotation

```
private void LeftRotation(INode<T,v> Xnode) {
    /*if (Xnode.isNull() || Xnode.getRightChild().isNull()) {
        return;
    }*/
    INode<T,v> Ynode = Xnode.getRightChild();
    Xnode.setRightChild(Ynode.getLeftChild());
    if(!Ynode.getLeftChild().isNull()) {
        Ynode.getLeftChild().setParent(Xnode);
    }
    Ynode.setParent(Xnode.getParent());
    if(Xnode.getParent().isNull()) {
        root=Ynode;
    }else if (Xnode == Xnode.getParent().getLeftChild()) {
        Xnode.getParent().setLeftChild(Ynode);
    }else {
        Xnode.getParent().setRightChild(Ynode);
    }
    Ynode.setLeftChild(Xnode);
    Xnode.setParent(Ynode);
}
```

- Insert fix

```
private void FIX(INode<T,v> curNode) {
    if (curNode==root) {
        root.setColor(false);
    }
    INode<T,v> node = curNode;
    while( node.getParent().getColor() ) {
        if(node.getParent() == node.getParent().getParent().getLeftChild()) {
            // case one when uncle is red so we do color flip
            if(node.getParent().getParent().getRightChild().getColor()) {
                node.getParent().setColor(false);
                node.getParent().getParent().getRightChild().setColor(false);
                node.getParent().getParent().setColor(true);
                node=node.getParent().getParent();
            }

            // case two when uncle is black so we need rotation
            else {
                if (node == node.getParent().getRightChild()){
                    node = node.getParent();
                    LeftRotation(node);
                }
                node.getParent().setColor(false);
                node.getParent().getParent().setColor(true);
                RightRotation(node.getParent().getParent());
            }
        }
        else {
            // case one when uncle is red so we do color flip
            if(node.getParent().getParent().getLeftChild().getColor()) {
                node.getParent().setColor(false);
                node.getParent().getParent().getLeftChild().setColor(false);
                node.getParent().getParent().setColor(true);
                node=node.getParent().getParent();
            }

            // case two when uncle is black so we need rotation
            else {
                if (node == node.getParent().getLeftChild()){
                    node = node.getParent();
                    RightRotation(node);
                }
            }
        }
    }

    root.setColor(false);
}
```

- Delete fix:

```
private void FIX_DELETE(INode<T,v> node) {
    while(node!=root&&!node.getColor()) {
        // part one when node is left child
        if(node == node.getParent().getLeftChild()) {
            INode<T,v> sibling = node.getParent().getRightChild();
            if(sibling.getColor()) { // red
                sibling.setColor(false);
                node.getParent().setColor(true);
                LeftRotation(node.getParent());
                sibling = node.getParent().getRightChild();
            }

            if(!sibling.getLeftChild().getColor()&&!sibling.getRightChild().getColor()) {
                sibling.setColor(true);
                node=node.getParent();
            }

            else {
                if (!sibling.getRightChild().getColor()) {
                    sibling.getLeftChild().setColor(false);
                    sibling.setColor(true);
                    this.RightRotation(sibling);
                    sibling=node.getParent().getRightChild();
                }

                sibling.setColor(node.getParent().getColor());
                node.getParent().setColor(false);
                sibling.getRightChild().setColor(false);
                LeftRotation(node.getParent());
                node = root;}

        }else {
```

```

    }else {

        INode<T,v> sibling = node.getParent().getLeftChild();
        if(sibling.getColor()) { // red
            sibling.setColor(false);
            node.getParent().setColor(true);
            RightRotation(node.getParent());
            sibling = node.getParent().getLeftChild();
        }

        if(!sibling.getLeftChild().getColor() && !sibling.getRightChild().getColor()) {
            sibling.setColor(false);
            node = node.getParent();
        }

        else {
            if (!sibling.getLeftChild().getColor()) {

                sibling.getRightChild().setColor(false);
                sibling.setColor(true);
                LeftRotation(sibling);
                sibling = node.getParent().getLeftChild();
            }

            sibling.setColor(node.getParent().getColor());
            node.getParent().setColor(false);
            sibling.getLeftChild().setColor(false);
            RightRotation(node.getParent());
            node = root;

        }
    }

    node.setColor(false);
}

```