Filipe de Oliveira Ataide

Mat.: 20181014040022

Árvore Binária de Pesquisa

Main.java

```
public class Main {
	public static void printTree(Tree t) {
		int h = t.height(t.getRoot());
		t.inOrder(t.getRoot());
		int w = t.nodes.size();
		int[][] vector = new int[h][w];
		for (int i = 0; i < w; i++) {
			Node n = t.nodes.get(i);
			vector[t.depth(n)][i] = n.getKey();
		}
		for (int l = 0; l < h; l++) {
			for (int c = 0; c < w; c++) {
				System.out.print((vector[l][c] != 0 ? vector[l][c] : "") + "\t");
			}
			System.out.println("");
		}
	}
	public static void main(String[] args) {
		Tree t = new Tree();
		t.insert(13);
		t.insert(6);
		t.insert(24);
		t.insert(5);
		t.insert(41);
		t.insert(7);
		t.insert(12);
		t.insert(27);
		t.insert(57);
		t.insert(44);
		t.remove(13);
		t.remove(7);

		printTree(t);
	}
}
```

Tree.java

```java
import java.util.ArrayList;

public class Tree {
    Node root;
    public ArrayList<Node> nodes = new ArrayList<>();

    public Node getRoot() {
        return root;
    }

    public void insert(int key) {
        Node n = new Node(key);
        if (root == null) {
            root = n;
        } else {
            Node node = root;
            Node parent;
            while (true) {
                parent = node;
                if (key < node.getKey()) {
                    node = node.getLeft();
                    if (node == null) {
                        parent.setLeft(n);
                        n.setParent(parent);
                        return;
                    }
                } else {
                    node = node.getRight();
                    if (node == null) {
                        parent.setRight(n);
                        n.setParent(parent);
                        return;
                    }
                }
            }
        }
    }
```

```
public boolean remove(int key) {
        Node node = root;
        Node parent = root;
        boolean ItsLeft = true;

        while (node.getKey() != key) {
                parent = node;
                if (key < node.getKey()) {
                        ItsLeft = true;
                        node = node.getLeft();
                        node.setParent(parent);
                } else {
                        ItsLeft = false;
                        node = node.getRight();
                        node.setParent(parent);
                }
                if (node == null) {
                        return false;
                }
        }
        if (node.getLeft() == null && node.getRight() == null) {
                if (node == root) {
                        root = null;
                } else if (ItsLeft) {
                        parent.setLeft(null);
                } else {
                        parent.setRight(null);
                }
        } else if (node.getRight() == null) {
                if (node == root) {
                        root = node.getLeft();
                        root.setParent(null);
                } else if (ItsLeft) {
                        parent.setLeft(node.getLeft());
                        node.getLeft().setParent(parent);
                } else {
                        parent.setRight(node.getLeft());
                        node.getLeft().setParent(parent);
                }
        } else if (node.getLeft() == null) {
```

```
        if (node == root) {
            root = node.getRight();
            root.setParent(null);
        } else if (ItsLeft) {
            parent.setLeft(node.getRight());
            node.getRight().setParent(parent);
        } else {
            parent.setRight(node.getRight());
            node.getRight().setParent(parent);
        }
    } else {
        Node replace = replace(node);
        if (node == root) {
            root = replace;
        } else if (ItsLeft) {
            parent.setLeft(replace);
            replace.setParent(parent);
        } else {
            parent.setRight(replace);
            replace.setParent(parent);
        }
        replace.setLeft(node.getLeft());

    }
    return true;
}

public Node replace(Node rNode) {
    Node rParent = rNode;
    Node r = rNode;

    Node node = rNode.getRight();

    while (node != null) {
        rParent = r;
        r = node;
        node = node.getLeft();
    }
    if (r != rNode.getRight()) {
        rParent.setLeft(r.getRight());
```

```
                r.setRight(rNode.getRight());
        }
        return r;
}

public Node search(int key) {
        Node node = root;
        while (node.getKey() != key) {
                if (key < node.getKey()) {
                        node = node.getLeft();
                } else {
                        node = node.getRight();
                }
                if (node == null) {
                        return null;
                }
        }
        return node;
}

public void inOrder(Node node) {
        if (node != null) {
                inOrder(node.getLeft());
                nodes.add(node);
                inOrder(node.getRight());
        }
}

public int height(Node node) {
        int lh = 0, rh = 0;
        if (node.getLeft() != null) {
                lh = height(node.getLeft());
        }
        if (node.getRight() != null) {
                rh = height(node.getRight());
        }
        return 1 + Math.max(lh, rh);
}

public int depth(Node node) {
		if (node == root) {
			return 0;
		} else {
			return 1 + depth(node.getParent());
		}
	}
}
```

Node.java

```
public class Node {
	private int key;
	private Node parent, left, right;

	public Node(int key) {
		this.key = key;
	}

	public int getKey() {
		return key;
	}
	public Node getParent() {
		return parent;
	}
	public Node getLeft() {
		return left;
	}
	public Node getRight() {
		return right;
	}

	public void setKey(int key) {
		this.key = key;
	}
	public void setParent(Node parent) {
		this.parent = parent;
	}
	public void setLeft(Node left) {
		this.left = left;
	}
	public void setRight(Node right) {
		this.right = right;
	}
}
```