자료구조 및 실습
보고서

[제09주] DS_09_201502273_김현종

2018.05.20.

201502273/김현종

1.내용

```
public class BinarySearchTree {
	BinaryNode root;

	/*
	 * BinarySearchTree의 Constructor
	 * root를 null로 초기화
	 */
	BinarySearchTree() {
		this.root = null;
	}

	/*
	 * x Node와 그 하위의 노드에서 value와 같은 값을 가지고있는 Node를 찾고
	 * toDelete값이 true이면 해당노드를 삭제한 뒤 반환하고
	 * toDelete값이 false이면 그냥 해당 노드를 반환한다.
	 */
	BinaryNode findNode(BinaryNode x, int value, boolean toDelete) {
		// TODO
		if (x != null && x.getValue() != value) {	//Leaf가 아니고 못찾았으면
			if (x.getValue() < value) {  //현재 노드 값보다 value가 더 크면
				return findNode(x.getRight(), value, toDelete);
//Right Child에서 이어 찾는다.
			} else {
				return findNode(x.getLeft(), value, toDelete);
			}
		} else if (x != null && x.getValue() == value) {	//찾았을 경우
			if (toDelete) {
				return deleteNode(x);	//노드 삭제 후 반환
			}
			return x;		//노드 그냥 반환
		} else {  //x == null, 찾지 못한경우
			return x;
		}
	}

	/*
	 * Tree에 Node를 삽입하는 메소드
	 * root Node 밑에 node를 삽입한다.
	 */
	void insertNode(BinaryNode root, BinaryNode node) {
		// TODO
		if (root == null) { //root가 null인경우
			this.root = node; //this.root를 node로 설정
		} else if (root.getValue() < node.getValue()) {//현재 노드 값보다 node의
값이 더 큰경우
			if (root.hasRight())		//Right Child가 있으면
				insertNode(root.getRight(), node);	//Right Child를
root로 Recursive하게 처리
			else {	//Right Child가 없으면
				root.setRight(node);		//node를 root의 Right
Child로 한다.
				node.setParent(root);
			}
		} else {  //현재 노드값보다 node의 값이 더 작거나 같은 경우
			if (root.hasLeft()) //Left Child가 있으면
				insertNode(root.getLeft(), node);	//Left Child를
root로 Recursive하게 처리
			else {	//Left Child가 없으면
				root.setLeft(node);		//node를 root의 Left Child로
한다.
				node.setParent(root);
			}
		}
	}

	/*
	 * 해당 노드와 하위 노드 중 가장 작은 값을 가진 노드를 반환하는 메소드
	 * BinarySearchTree에서는 Left Most Node가 가장 작은 값을 가진 노드이므로
	 * Left Most를 반환한다.
```

```
	 */
	BinaryNode treeMinimum(BinaryNode node) {
		// TODO
		if (node.hasLeft())
			return treeMinimum(node.getLeft());
		else
			return node;
	}

	/*
	 * 받은 노드와 같은 값을 갖는 Node를 삭제하는 메소드
	 */
	BinaryNode deleteNode(BinaryNode z) {
		// TODO
		if (z.hasLeft() && z.hasRight()) { // has both
			BinaryNode rightMinimum = treeMinimum(z.getRight());
//successor를 가져온다.

			if (rightMinimum.getParent() == z) {//successor가 z의 Right
child인 경우
				rightMinimum.setLeft(z.getLeft());	//successor의 Left
Child를 z의 Left Child로 설정
				z.getLeft().setParent(rightMinimum);
				transplant(z, rightMinimum);		//successor를 z에
transplant
			} else {  //successor가 z의 Right child가 아닌경우


rightMinimum.getParent().setLeft(rightMinimum.getRight());	//successor의 Left Child를
successor의 Right Child로 설정
				if (rightMinimum.hasRight())		//successor가 Right
Child를 갖고있으면

rightMinimum.getRight().setParent(rightMinimum.getParent());  //successor의 Right Child의
Parent를 successor의 Parent로 설정

				rightMinimum.setRight(z.getRight()); //z의 child를
successor에게 붙힌다.
				rightMinimum.setLeft(z.getLeft());
				z.getLeft().setParent(rightMinimum);
				z.getRight().setParent(rightMinimum);
				z.setLeft(null);	//z의 child를 삭제
				z.setRight(null);

				transplant(z, rightMinimum);		//successor를 z로
transplant
			}

			return z;//z를 반환
		} else if (z.hasLeft()) { // has only Left Child
			transplant(z, z.getLeft());	//z의 Left Child를 z에 transplant
			return z;//z를 반환
		} else if (z.hasRight()) { // has only Right Child
			transplant(z, z.getRight());  //z의 Right Child를 z에 transplant
			return z;//z를 반환
		} else { // Leaf
			transplant(z, null);		//null을 z에 transplant
			return z;//z를 반환
		}
	}

	/*
	 * source를 des의 Parent에 붙히는 메소드
	 */
	void transplant(BinaryNode des, BinaryNode source) {
		// TODO
		if (!des.hasParent()) {		//목적지에 Parent가 없으면
			this.root = source;		//source를 root로 설정
		} else if (des == des.getParent().getLeft()) {  //des가 Left Child이면
			des.getParent().setLeft(source);	//source를 des의 Parent의
Left Child로 설정
		} else {  //des가 Right Child이면
```

```
                des.getParent().setRight(source);      //source를 des의 Parent의
Right Child로 설정
            }
            if (source != null) {          //source가 null이 아니면
                source.setParent(des.getParent());    //source의 Parent를 des의
Parent로 설정
            }
        }

        /*
         * BinarySearchTree를 출력하는 메소드
         */
        void printTree(BinaryNode node, int depth) {
            // TODO
            if (node != null) {//node가 null이 아니면
                for (int i = 0; i < depth; i++) {       //depth만큼
                    System.out.print("\t");        //tab을 출력하고
                }
                System.out.println(node.getValue()); //node의 value를 출력하고
                printTree(node.getLeft(), depth + 1);          //node의 Left Child
출력
                printTree(node.getRight(), depth + 1);         //node의 Right
child 출력
            }
        }

        /*
         * root의 Getter
         */
        BinaryNode getRoot() {
            return this.root;
        }
}
```

BinarySearchTree.java

```
public class BinaryNode {
    private int value;       //node의 value
    private BinaryNode parent;      //node의 Parent node
    private BinaryNode left;        //node의 Left Child
    private BinaryNode right;       //node의 Right Child

    /*
     * BinaryNode의 Constructor
     * 받은 값으로 value를 초기화하고
     * left, right, parent를 null로 초기화
     */
    BinaryNode(int value) {
        this.value = value;
        this.left = null;
        this.right = null;
        this.parent = null;
    }

    /*
     * value의 getter
     */
    public int getValue() {
        return value;
    }

    /*
     * parent의 setter
     */
    public void setParent(BinaryNode parent) {
        this.parent = parent;
    }

    /*
     * left의 setter
     */
    public void setLeft(BinaryNode left) {
```

```
        this.left = left;
    }

    /*
     * right의 setter
     */
    public void setRight(BinaryNode right) {
        this.right = right;
    }

    /*
     * parent의 getter
     */
    public BinaryNode getParent() {
        return this.parent;
    }

    /*
     * left의 getter
     */
    public BinaryNode getLeft() {
        return this.left;
    }

    /*
     * right의 getter
     */
    public BinaryNode getRight() {
        return this.right;
    }

    /*
     * parent의 여부를 반환하는 메소드
     */
    public boolean hasParent() {
        return this.parent != null;
    }

    /*
     * left의 여부를 반환하는 메소드
     */
    public boolean hasLeft() {
        return this.left != null;
    }

    /*
     * right의 여부를 반환하는 메소드
     */
    public boolean hasRight() {
        return this.right != null;
    }
}
```

BinaryNode.java

```
import java.util.Scanner;

public class MainClass_09_201502273 {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        BinarySearchTree bst = new BinarySearchTree();
        int menu, insert;
        BinaryNode find;
        System.out.println("===============Start BinarySearchTree
Test===============");
        System.out.println("1.InsertValue    2.Find    3.Delete    4.PrintTree    9.Exit");
        System.out.print("Insert Menu > ");
        menu = sc.nextInt();

        while (menu != 9) {

            switch (menu) {
```

```
                    case 1:
                        System.out.println("Please Insert To Input Value");
                        insert = sc.nextInt();
                        bst.insertNode(bst.getRoot(), new BinaryNode(insert));
                        System.out.println("Inserting " + insert);
                        break;
                    case 2:
                        System.out.println("Please Insert To Find Value");
                        insert = sc.nextInt();
                        find = bst.findNode(bst.getRoot(), insert, false);
                        if (find != null) {
                            System.out.println("Find : " + find.getValue());
                        } else
                            System.out.println("Doesn't Exist Value " + insert);
                        break;
                    case 3:
                        System.out.println("Please Insert To Delete value");
                        insert = sc.nextInt();
                        find = bst.findNode(bst.getRoot(), insert, true);
                        if (find != null)
                            System.out.println("Delete : " + find.getValue());
                        else
                            System.out.println("Doesn't Exist Value " + insert);
                        break;
                    case 4:
                        if (bst.getRoot() == null)
                            System.out.println("Tree is Empty");
                        else
                            bst.printTree(bst.getRoot(), 0);
                        break;
                }
                System.out.println("\n\n1.InsertValue    2.Find    3.Delete    4.PrintTree
9.Exit");
                System.out.print("Insert Menu > ");
                menu = sc.nextInt();

            }
            System.out.println("===============End BinarySearchTree Test===============");
        }
}
```

MainClass_09_201502273.java

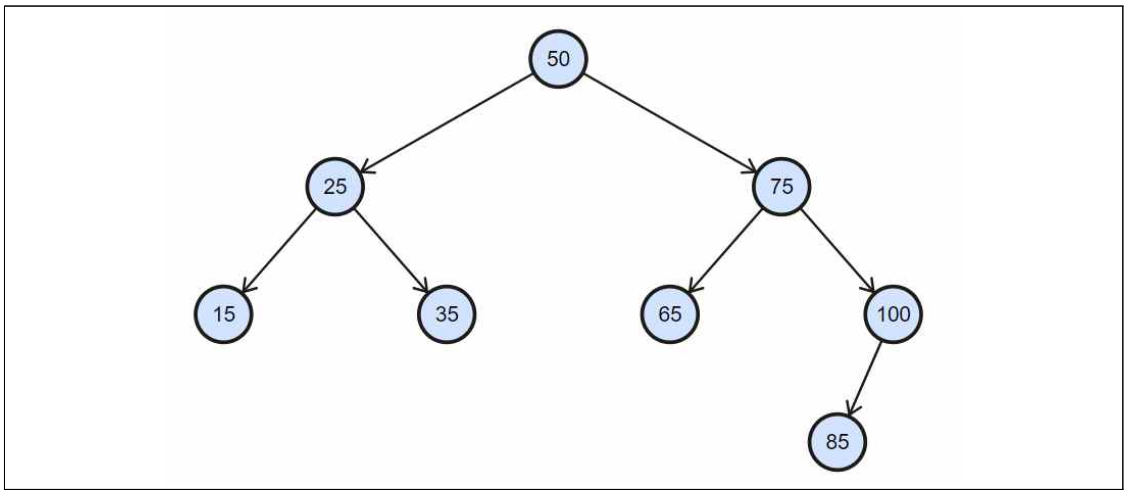
2.결과

```
===============Start BinarySearchTree Test================
1.InsertValue    2.Find    3.Delete    4.PrintTree    9.Exit
Insert Menu > 1
Please Insert To Input Value
50
Inserting 50


1.InsertValue    2.Find    3.Delete    4.PrintTree    9.Exit
Insert Menu > 1
Please Insert To Input Value
75
Inserting 75


1.InsertValue    2.Find    3.Delete    4.PrintTree    9.Exit
Insert Menu > 1
Please Insert To Input Value
65
Inserting 65


1.InsertValue    2.Find    3.Delete    4.PrintTree    9.Exit
Insert Menu > 1
Please Insert To Input Value
100
Inserting 100

1.InsertValue    2.Find    3.Delete    4.PrintTree    9.Exit
Insert Menu > 1
Please Insert To Input Value
85
Inserting 85


1.InsertValue    2.Find    3.Delete    4.PrintTree    9.Exit
Insert Menu > 1
Please Insert To Input Value
25
Inserting 25


1.InsertValue    2.Find    3.Delete    4.PrintTree    9.Exit
Insert Menu > 1
Please Insert To Input Value
15
Inserting 15


1.InsertValue    2.Find    3.Delete    4.PrintTree    9.Exit
Insert Menu > 1
Please Insert To Input Value
35
Inserting 35
```

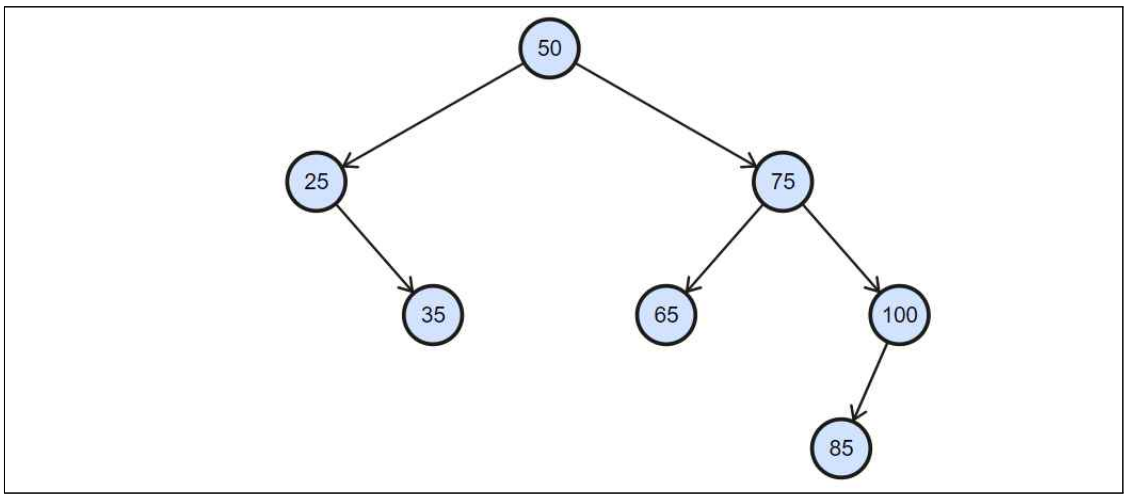
```
1.InsertValue    2.Find    3.Delete    4.PrintTree    9.Exit
Insert Menu > 4
50
        25
                15
                35
        75
                65
                100
                        85


1.InsertValue    2.Find    3.Delete    4.PrintTree    9.Exit
Insert Menu > 2
Please Insert To Find Value
85
Find : 85


1.InsertValue    2.Find    3.Delete    4.PrintTree    9.Exit
Insert Menu > 3
Please Insert To Delete value
15
Delete : 15


1.InsertValue    2.Find    3.Delete    4.PrintTree    9.Exit
Insert Menu > 4
50
        25
                35
        75
                65
                100
                        85


1.InsertValue    2.Find    3.Delete    4.PrintTree    9.Exit
Insert Menu > 3
Please Insert To Delete value
25
Delete : 25


1.InsertValue    2.Find    3.Delete    4.PrintTree    9.Exit
Insert Menu > 4
50
        35
        75
                65
                100
                        85
```

```
1.InsertValue    2.Find    3.Delete    4.PrintTree    9.Exit
Insert Menu > 3
Please Insert To Delete value
75
Delete : 75


1.InsertValue    2.Find    3.Delete    4.PrintTree    9.Exit
Insert Menu > 4
50
        35
        85
                65
                100


1.InsertValue    2.Find    3.Delete    4.PrintTree    9.Exit
Insert Menu > 3
Please Insert To Delete value
85
Delete : 85


1.InsertValue    2.Find    3.Delete    4.PrintTree    9.Exit
Insert Menu > 4
50
        35
        100
                65


1.InsertValue    2.Find    3.Delete    4.PrintTree    9.Exit
Insert Menu > 3
Please Insert To Delete value
100
Delete : 100


1.InsertValue    2.Find    3.Delete    4.PrintTree    9.Exit
Insert Menu > 4
50
        35
        65


1.InsertValue    2.Find    3.Delete    4.PrintTree    9.Exit
Insert Menu > 2
Please Insert To Find Value
65
Find : 65


1.InsertValue    2.Find    3.Delete    4.PrintTree    9.Exit
Insert Menu > 9
===============End BinarySearchTree Test===============
```

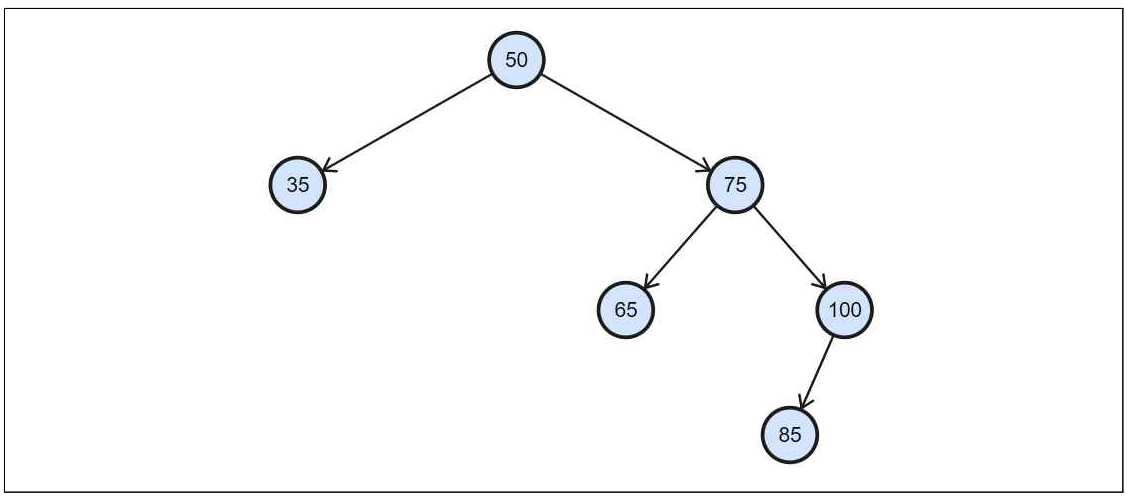
50, 75, 65, 100, 85, 25, 15, 35를 순서대로 Tree에 넣은 것을 볼 수 있으며 다음과 같은 트리가 만들어졌다.



그리고 85를 찾으면 잘 찾는 것을 볼 수 있으며 Leaf에 해당하는 15를 삭제하면 잘 작동하여 다음과 같은 Tree가 되는 것을 볼 수 있다.

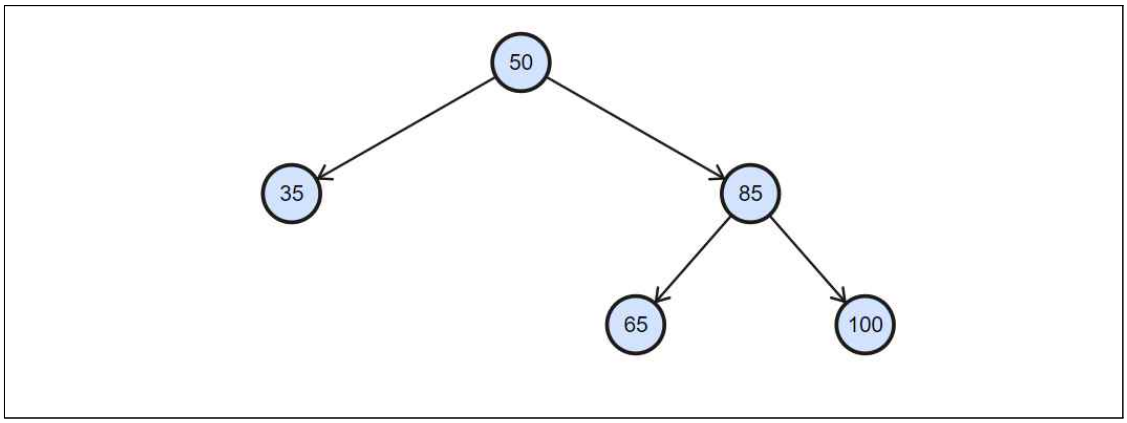


그리고 Right Child만 가지고 있는 25를 삭제하면 다음과 같은 Tree로 바뀌는 것을 볼 수 있다.

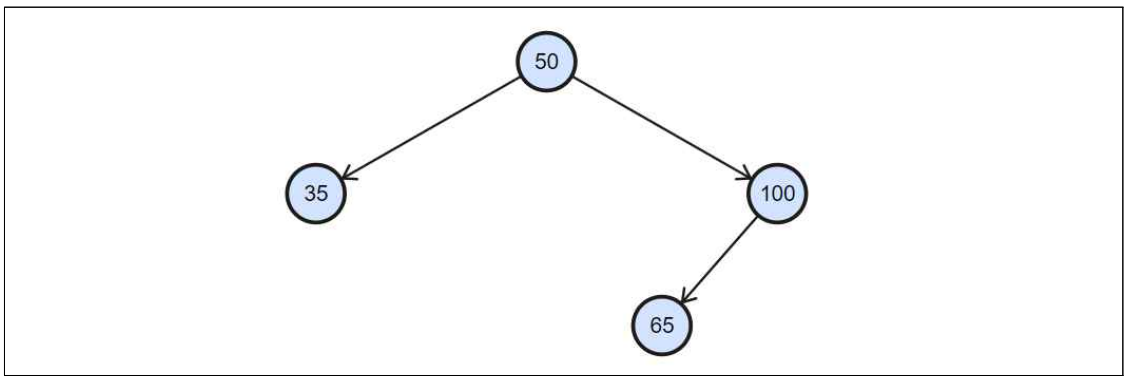


Left Child와 Right Child를 모두 가지고 있는 75를 삭제하면 다음과 같이 변하는 것을 볼

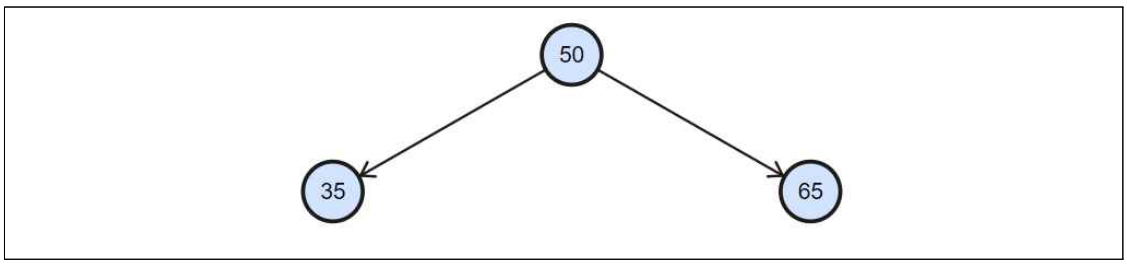
수 있다.



그리고 Left Child와 Right Child를 모두 가지고 있으면서 해당 Node의 Successor가 자신의 Right Child인 85를 삭제해도 잘 작동해 다음과 같은 Tree가 나오는 것을 볼 수 있다.



마지막으로 Left Child만 가지고 있는 100을 삭제해도 제대로 작동하는 것을 볼 수 있다.



깨달은 점 및 결론

삭제하는 경우에 삭제할 노드가 자식을 둘 다 가지고 있으면 삭제할 노드의 Successor를 삭제할 노드의 위치에 넣어야하는데 Successor가 삭제할 노드의 Right Child인 경우와 그렇지 않은 경우로 나누지 않아서 Successor가 삭제할 노드의 Right Child인 경우에 루프가 형성되어 Print할 경우 무한루프를 도는 문제가 있었는데 경우를 나눔으로서 해결할 수 있었다.