June 30th, 2020

# Exercises for Programming Practice 2

Note:
- ✓ Do not create “module-info.java”, when you create a Java Project.
- ✓ Do not set Package name in the window “New Java Class”.
- ✓ Do not use the title of exercise for “contents of the program”. Think about “contents of the program” yourself.

The deadline for submitting the programs is 18:00 on July 2nd, 2020.

In exercise 18 and 19, the binary search tree shown in Fig. 1 is used. In this binary tree, the values associated with keys are not diagramed.

The binary search tree is implemented in TestData class shown in Fig. 2 using TreeNode class shown in Fig. 3. The implemented tree can be obtained with getTree method of TestDate class. In the showTree method, the tree is displayed based on the algorithm of inorder traversal.

The classes “TestData.java” and “TreeNode.java” can be obtained from “Week 9” of Resource page, “Programming Practice 2” course, manaba+R.

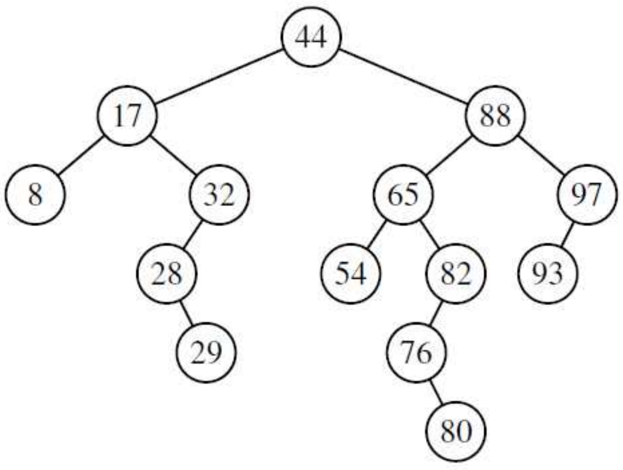

Fig. 1 A binary search tree with integer keys.

```java
public class TestData {
	TreeNode tree;

	public TestData() {
		tree = new TreeNode(44,
			new TreeNode(17,
				new TreeNode(8, null, null),
				new TreeNode(32,
					new TreeNode(28,
						null,
						new TreeNode(29, null, null)
					),
					null
				)
			),
			new TreeNode(88,
				new TreeNode(65,
					new TreeNode(54, null, null),
					new TreeNode(82,
						new TreeNode(76,
							null,
							new TreeNode(80, null, null)
						),
						null
					)
				),
				new TreeNode(97,
					new TreeNode(93, null, null),
					null)
			)
		);
	}

	public TreeNode getTree() {
		return tree;
	}

	public void showTree(TreeNode tree) {
		if(tree != null) {
			showTree(tree.getLeft());
			show(tree);
			showTree(tree.getRight());
		}
	}

	public void show(TreeNode tree) {
		System.out.print(" " + tree.getKey());
	}
}
```

Fig. 2 TestData.java

```
public class TreeNode {
      private String data;
      private int key;
      private TreeNode left;
      private TreeNode right;

      public TreeNode(int key_, TreeNode left_, TreeNode right_){
            key = key_;
            left = left_;
            right = right_;
      }

      public int getKey() {
            return key;
      }

      public TreeNode getLeft() {
            return left;
      }

      public TreeNode getRight() {
            return right;
      }

      public void setLeft(TreeNode node) {
            left = node;
      }

      public void setRight(TreeNode node) {
            right = node;
      }

      public void setKey(int key_) {
            key = key_;
      }
}
```

Fig. 3 TreeNode.java

Exercise 18 (file name “Insert.java”)

Complete method “public TreeNode insertKey(TreeNode tree, int key)” (of “Insert.java”), in which the following algorithm is realized.

(1) If tree is null, make new TreeNode using key, in which the values of left child and right child are null, and assign it to tree.
(2) If key is less than the key value of tree, set the result of insertKey(tree.getLeft(), key) to left child of tree.
(3) Otherwise, set the result of insertKey(tree.getRight(), key) to right child of tree.

Incomplete “Insert.java” can be downloaded from “Week 9” of Resource page, “Programming Practice 2” course, manaba+R.

The outline of “Insert.java” is as follows.

```
/* comments */

public class Insert {
    TreeNode tree;
    TestData testData;

    public static void main(String[] args) {
        new Insert();
    }

    public Insert() {
        testData = new TestData();
        tree = testData.getTree();
        testData.showTree(tree);
        System.out.println();
        tree = insertKey(tree, 68);
        testData.showTree(tree);
        System.out.println();
    }

    public TreeNode insertKey(TreeNode tree, int key) {
        System.out.print(key + " >> ");
        testData.showTree(tree);
        System.out.println();

        /* complete this part */

        return tree;
    }
}
```

Expected result:

```
8 17 28 29 32 44 54 65 76 80 82 88 93 97
68 >> 8 17 28 29 32 44 54 65 76 80 82 88 93 97
68 >> 54 65 76 80 82 88 93 97
68 >> 54 65 76 80 82
68 >> 76 80 82
68 >> 76 80
68 >>
8 17 28 29 32 44 54 65 68 76 80 82 88 93 97
```

Exercise 19 (file name “Delete.java”)

Complete method “public TreeNode deleteKey(TreeNode tree, int key)” (of “Delete.java”), in which the following algorithm is realized.

(1) If tree is null, do nothing.

(2) If key is less than the key value of tree, set the result of deleteKey(tree.getLeft(), key) to left child of tree.

(3) If key is larger than the key value of tree, set the result of deleteKey(tree.getRight(), key) to right child of tree.

(4) Otherwise

 (4-1) If there is no left child, assign the right child to tree.

 (4-2) If there is no right child, assign the left child to tree.

 (4-3) Otherwise, assign the value of minValue(tree.getRight()) to key of tree.

Incomplete “Delete.java” can be downloaded from “Week 9” of Resource page, “Programming Practice 2” course, manaba+R.

The outline of “Delete.java” is as follows.

```java
/* comments */

public class Delete {
    TreeNode tree;
    TestData testData;

    public static void main(String[] args) {
        new Delete();
    }

    public Delete() {
        testData = new TestData();
        tree = testData.getTree();
        testData.showTree(tree);
        System.out.println();
        tree = deleteKey(tree, 32);
        testData.showTree(tree);
        System.out.println();
    }

    public TreeNode deleteKey(TreeNode tree, int key) {
        System.out.print(key + " >> ");
        testData.showTree(tree);
        System.out.println();

        if(tree == null) {
            /* complete this part */
        }
        return tree;
    }
    public int minValue(TreeNode tree) {
        int minv = tree.getKey();

        while(tree.getLeft() != null) {
            minv = tree.getLeft().getKey();
            tree = tree.getLeft();
        }
        return minv;
    }
}
```

Expected result:

```
8 17 28 29 32 44 54 65 76 80 82 88 93 97
32 >> 8 17 28 29 32 44 54 65 76 80 82 88 93 97
32 >> 8 17 28 29 32
32 >> 28 29 32
8 17 28 29 44 54 65 76 80 82 88 93 97
```