# TÖL212M Rökstudd Forritun - Einstaklingsverkefni 6

Andri Fannar Kristjánsson

23. febrúar 2025

## Einstaklingsverkefni 6

## 1

Sækið skrána `E6-skeleton.java` og vistið hana hjá ykkur en breytið nafni hennar í E6.java. Klárið að forrita klasann og keyrið einnig forritið og sýnið útkomuna.

### 1.1 Svar:

Hér fyrir neðan má sjá kóðann leystu útgáfuna. Þegar skráin er keyrð fæst einfaldlega `Testing is done`. Einnig er hægt að skoða skrána hér.

```
// Author of solution: Andri Fannar Kristjánsson, afk6@hi.is

// Vistið þessa skrá með nafninu E6.java og klárið að
// forrita föllin searchRecursive, searchLoop og
// searchTailRecursive. Þið skuluð líka þýða og keyra
// klasann svona (þið þurfið einnig skrána AVL.java):
//    javac AVL.java E6.java
//    java E6
// Í forrituninni skuluð þið fylgja þeim stöðulýsingum
// sem gefnar eru.

// Save this file with the name E6.java and finish
// programming the functions searchRecursive, searchLoop,
// and searchTailRecursive. You should also compile and
// run the class like this (you also need the file AVL.java):
//    javac AVL.java E6.java
//    java E6
// In the programming you should fulfil the state descriptions
// given.

class E6 {
    public static class Pair<T extends Comparable<? super T>> {
        public AVL<T> le, gt;
    }

    // Notkun: searchRecursive(t,x,p);
    // Fyrir:  t er AVL<T> tré, x er T gildi, p er Pair<T>.
    //         Ekkert þessara gilda (nema e.t.v. t) er null og t er
    //         í vaxandi röð.
    // Eftir:  p.le vísar á aftasta hnút í t með gildi <=x,
    //         ef slíkur hnútur er til. Ef slíkur hnútur er
    //         ekki til þá er p.le jafnt null.
    //         p.gt vísar á fremsta hnút í t með gildi >x,
    //         ef slíkur hnútur er til. Ef slíkur hnútur er
    //         ekki til þá er p.gt jafnt null.
```

```
// Usage: searchRecursive(t,x,p);
// Pre:   t is an AVL<T> tree, x is a T value, p is a Pair<T>.
//        None of those values (except maybe t) are null and t is in
//        ascending order.
// Post:  p.le refers to the rightmost node in t with a value
//        <=x, if such a node exists. If no such node exists
//        then p.le is null.
//        p.gt refers to the leftmost node in t with a value
//        >x, if such a node exists. If no such node exists
//        then p.gt is null.
public static <T extends Comparable<? super T>> void
            searchRecursive(AVL<T> t, T x, Pair<T> p) {
    // We set p.le and p.gt to null if t is empty
    if (t == null) {
        p.le = null;
        p.gt = null;
    } else if (AVL.rootValue(t).compareTo(x) <= 0) {
        // If the root value of t is less than or equal to x,
        // we search the right subtree of t.
        searchRecursive(AVL.right(t), x, p);
        // When going back, we check if p.le is null,
        // and if it is, we set it to t, as then it is the
        // rightmost node with a value <= x.
        if (p.le == null) {
            p.le = t;
        }
    } else {
        // If x is greater than the root value of t,
        // we search the left subtree of t.
        searchRecursive(AVL.left(t), x, p);
        // When going back, we check if p.gt is null,
        // and if it is, we set it to t, as then it is the
        // leftmost node with a value > x.
        if (p.gt == null) {
            p.gt = t;
        }
    }
}

// Notkun: searchTailRecursive(t,x,p);
// Fyrir:  t er AVL<T> tré, x er T gildi, p er Pair<T>.
//         Ekkert þessara gilda er null (nema e.t.v. t) og t er
//         í vaxandi röð.
// Eftir:  p.le vísar á aftasta hnút í t með gildi <=x,
//         ef slíkur hnútur er til. Ef slíkur hnútur er
//         ekki til þá er p.le óbreytt.
//         p.gt vísar á fremsta hnút í t með gildi >x,
//         ef slíkur hnútur er til. Ef slíkur hnútur er
//         ekki til þá er p.gt óbreytt.

// Usage: searchTailRecursive(t,x,p);
// Pre:   t is an AVL<T> tree, x is a T value, p is a Pair<T>.
//        None of those values (except maybe t) are null and t is in
//        ascending order.
// Post:  p.le refers to the rightmost node in t with a value
//        <=x, if such a node exists. If no such node exists
//        then p.le is unchanged.
```

```
//        p.gt refers to the leftmost node in t with a value
//        >x, if such a node exists. If no such node exists
//        then p.gt is unchanged.
public static <T extends Comparable<? super T>> void
            searchTailRecursive(AVL<T> t, T x, Pair<T> p) {
    // If t is null, we're done traversing the tree.
    // We don't have to change p.le or p.gt, as the postcondition
    // states that they should be unchanged if no node with the
    // correct value is found.
    // p.le and p.gt then contain the right- and leftmost nodes with
    // values <= x and > x, respectively, if there were any found
    // (if t is not null) at the start.
    if (t == null)
        return;
    if (AVL.rootValue(t).compareTo(x) <= 0) {
        // If x is less than or equal to the root value of t,
        // we set p.le to t and search the right subtree of t.
        p.le = t;
        searchTailRecursive(AVL.right(t), x, p);
    } else {
        // If x is greater than the root value of t,
        // we set p.gt to t and search the left subtree of t.
        p.gt = t;
        searchTailRecursive(AVL.left(t), x, p);
    }
}

// Notkun: searchLoop(t,x,p);
// Fyrir:  t er AVL<T> tré, x er T gildi, p er Pair<T>.
//         Ekkert þessara gilda (nema e.t.v. t) er null og t er
//         í vaxandi röð.
// Eftir:  p.le vísar á aftasta hnút í t með gildi <=x,
//         ef slíkur hnútur er til. Ef slíkur hnútur er
//         ekki til þá er p.le jafnt null.
//         p.gt vísar á fremsta hnút í t með gildi >x,
//         ef slíkur hnútur er til. Ef slíkur hnútur er
//         ekki til þá er p.gt jafnt null.

// Usage: searchLoop(t,x,p);
// Pre:   t is an AVL<T> tree, x is a T value, p is a Pair<T>.
//        None of those values (except maybe t) are null and t is in
//        ascending order.
// Post:  p.le refers to the rightmost node in t with a value
//        <=x, if such a node exists. If no such node exists
//        then p.le is null.
//        p.gt refers to the leftmost node in t with a value
//        >x, if such a node exists. If no such node exists
//        then p.gt is null.
public static <T extends Comparable<? super T>> void
            searchLoop(AVL<T> t, T x, Pair<T> p) {
    // We create a new variable s that contains the current
    // working subtree.
    AVL<T> s = t;
    // We also set p.le and p.gt to null at the start.
    p.le = null;
    p.gt = null;
    while (s != null) {
        // s er undirtré t með einhverja trjáslóð sp.
```

```
            // Allt í PreSeq(t,sp) er <=x.
            // Allt í PostSeq(t,sp) er >x.
            // Ef til er hnútur í t fyrir framan s undirtréð
            // (þ.e. PreSeq(t,sp) er ekki tómt) þá vísar
            // p.le á aftasta slíkan hnút. Annars er p.le
            // jafnt null.
            // Ef til er hnútur í t fyrir aftan s undirtréð
            // (þ.e. PostSeq(t,sp) er ekki tómt) þá vísar
            // p.gt á fremsta slíkan hnút. Annars er p.gt
            // jafnt null.

            // s is a subtree of t with some treepath sp.
            // All in PreSeq(t,sp) is <=x.
            // All í PostSeq(t,sp) is >x.
            // If there exists a node in t to the left of the
            // s subtree (i.e. PreSeq(t,sp) is not empty) then
            // p.le refers to the rightmost such node. Otherwise
            // p.le is null.
            // If there exists a node in t to the right of the
            // s subtree (i.e. PostSeq(t,sp) is not empty) then
            // p.gt refers to the leftmost such node. Otherwise
            // p.gt is null.

            if (AVL.rootValue(s).compareTo(x) <= 0) {
                // If the root value of s is less than or equal to x,
                // we set p.le to s and set s to the right subtree of old s.
                p.le = s;
                s = AVL.right(s);
            } else {
                // If the root value of s is greater than x,
                // we set p.gt to s and set s to the left subtree of old s.
                p.gt = s;
                s = AVL.left(s);
            }
        }
    }

    private static void
            test_LastLE_FirstGT(java.util.function.Function<Integer,
                                                            Pair<Integer>> f,
                                AVL<Integer> t) {
        for (int x = -1; x != 2000; x++) {
            Pair<Integer> p = f.apply(x);
            if (p.le != null) {
                if (AVL.rootValue(p.le) > x)
                    throw new Error("" + x);
                if (x >= 1998 && AVL.rootValue(p.le) != 1998)
                    throw new Error("" + x);
                if (x <= 1998 && x / 2 * 2 != AVL.rootValue(p.le))
                    throw new Error("" + x);
                if (AVL.right(p.le) != null &&
                    AVL.rootValue(AVL.right(p.le)) <= x)
                    throw new Error("" + x);
            } else {
                if (x >= 0)
                    throw new Error("" + x);
                if (AVL.find(t, x) || AVL.find(t, x - 1))
                    throw new Error("" + x);
```

```
            }
            if (p.gt != null) {
                if (AVL.rootValue(p.gt) <= x)
                    throw new Error("" + x);
                if (x < 0 && AVL.rootValue(p.gt) != 0)
                    throw new Error("" + x);
                if (x >= 0 && (x + 2) / 2 * 2 != AVL.rootValue(p.gt))
                    throw new Error("" + x);
                if (AVL.left(p.gt) != null &&
                    AVL.rootValue(AVL.left(p.gt)) > x)
                    throw new Error("" + x);
            } else {
                if (x < 1998)
                    throw new Error("" + x);
                if (AVL.find(t, x + 1) || AVL.find(t, x + 2))
                    throw new Error("" + x);
            }
        }
    }

    public static void main(String[] args) {
        AVL<Integer> t = null;
        for (int n = 0; n != 10; n++)
            for (int i = 0; i != 1000; i++)
                t = AVL.insert(t, 2 * i);
        final AVL<Integer> s = t;
        final Pair<Integer> p = new Pair<Integer>();
        try {
            test_LastLE_FirstGT(i -> {
                searchRecursive(s, i, p);
                return p;
            }, s);
        } catch (Error e) {
            System.out.println("Error in searchRecursive searching for "
                               + e.getMessage());
        }
        try {
            test_LastLE_FirstGT(i -> {
                p.le = p.gt = null;
                searchTailRecursive(s, i, p);
                return p;
            }, s);
        } catch (Error e) {
            System.out.println("Error in searchTailRecursive searching for "
                               + e.getMessage());
        }
        try {
            test_LastLE_FirstGT(i -> {
                searchLoop(s, i, p);
                return p;
            }, s);
        } catch (Error e) {
            System.out.println("Error in searchLoop searching for "
                               + e.getMessage());
        }
        System.out.println("Testing is done");
    }
}
```