Lab 4

Bruna Nunes

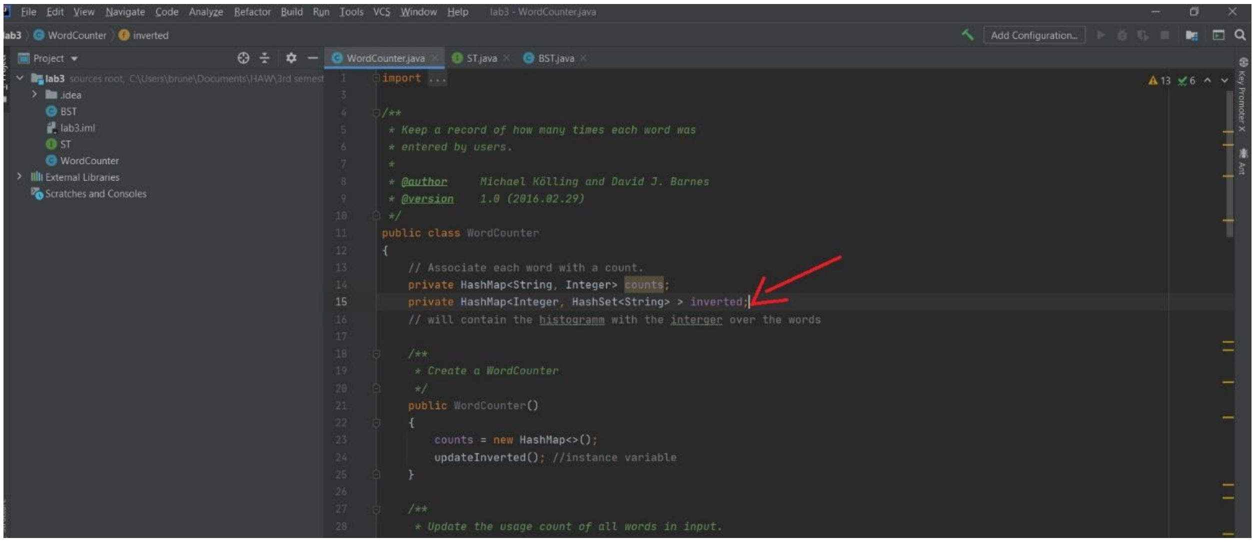
Anastasiya Purtova

- ## Preparation

1) We call that the BST invariant the fact that for any node n, every node in the left subtree of n has a value less than n's value, and every node in the right subtree of n has a value greater than n's value.

2) Implementation

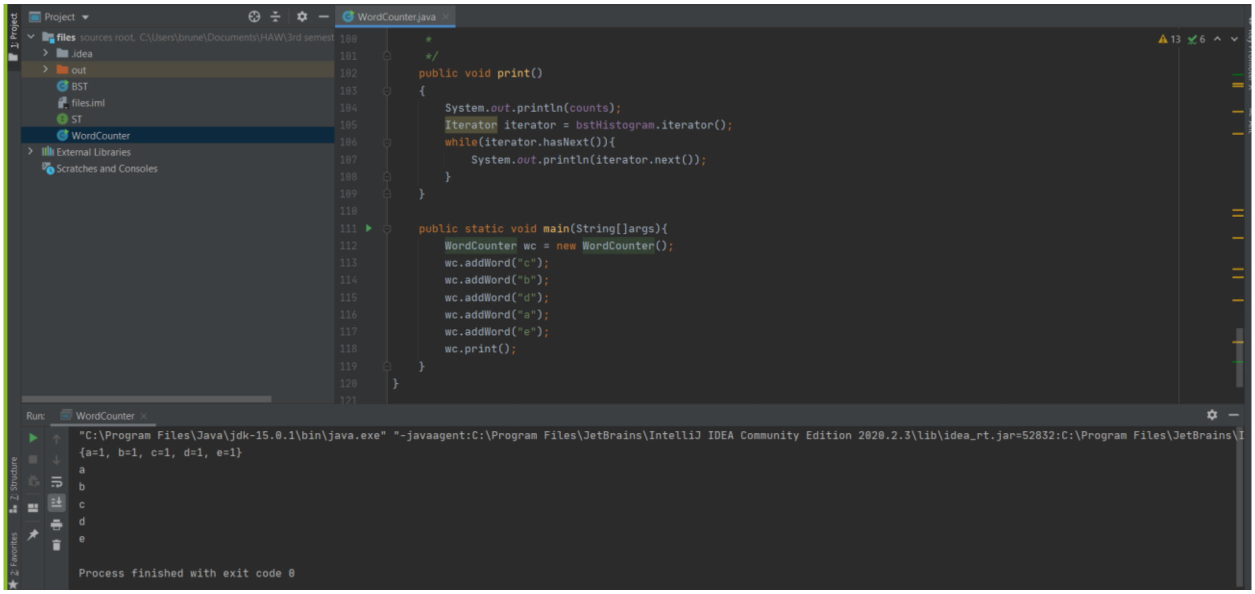
3)



```java
import ...

/**
 * Keep a record of how many times each word was
 * entered by users.
 *
 * @author     Michael Kölling and David J. Barnes
 * @version    1.0 (2016.02.29)
 */
public class WordCounter
{
    // Associate each word with a count.
    private HashMap<String, Integer> counts;
    private HashMap<Integer, HashSet<String> > inverted;
    // will contain the histogramm with the interger over the words

    /**
     * Create a WordCounter
     */
    public WordCounter()
    {
        counts = new HashMap<>();
        updateInverted(); //instance variable
    }

    /**
     * Update the usage count of all words in input.
```

The word-frequency histogram-keeping Map is the HashMap inverted. I changed the HashMap for BST type and changed the instance variable name to bstHistogram and the method updateInverted to updateBSTHistogram. For every word inserted into the counts instance variable, a word was inserted into the new Binary Search Tree histogram.

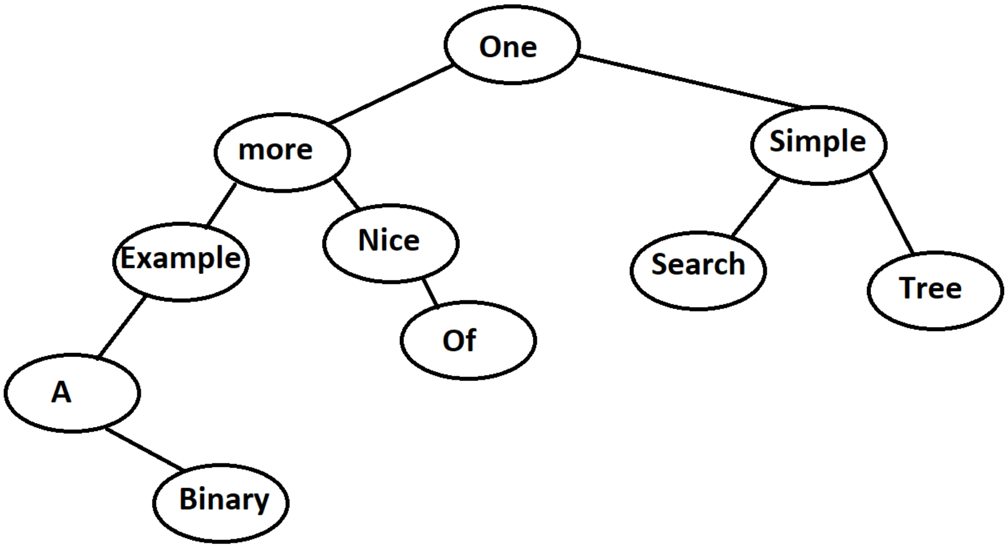
The print method was changed in a way that we can loop through the values of the tree and print out the values. We did this by retrieving the iterator attribute of the bstHistogram attribute and iterating through all of its stack values.



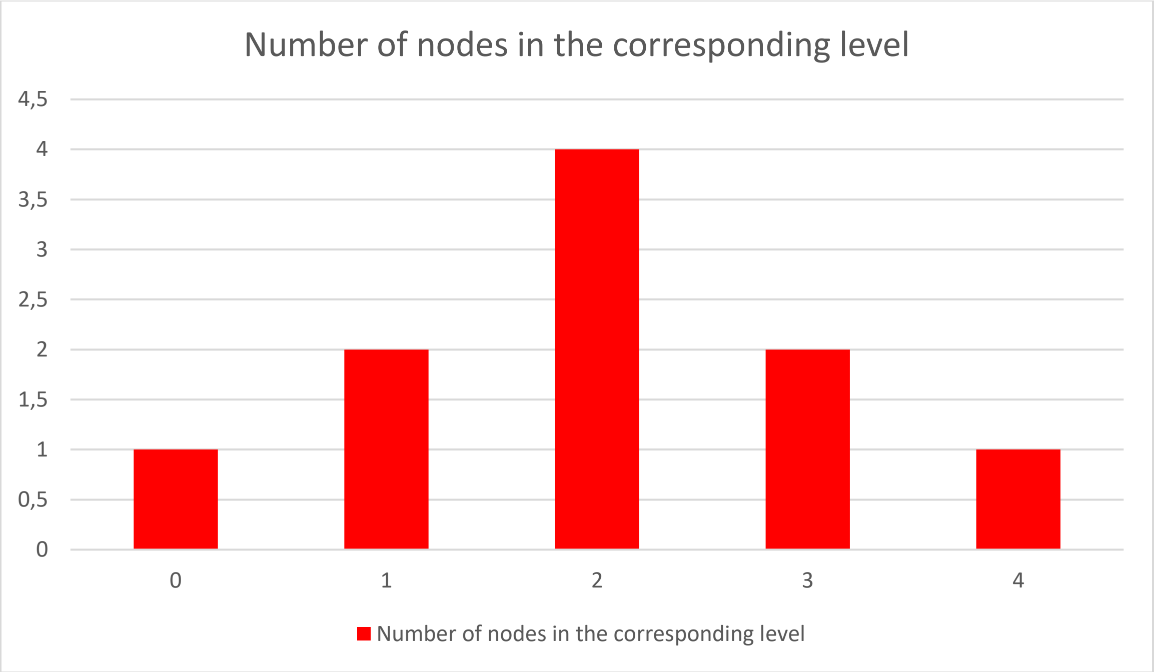
```java
     *
     */
    public void print()
    {
        System.out.println(counts);
        Iterator iterator = bstHistogram.iterator();
        while(iterator.hasNext()){
            System.out.println(iterator.next());
        }
    }

    public static void main(String[]args){
        WordCounter wc = new WordCounter();
        wc.addWord("c");
        wc.addWord("b");
        wc.addWord("d");
        wc.addWord("a");
        wc.addWord("e");
        wc.print();
    }
}
```

```
"C:\Program Files\Java\jdk-15.0.1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2020.2.3\lib\idea_rt.jar=52832:C:\Program Files\JetBrains\I
{a=1, b=1, c=1, d=1, e=1}
a
b
c
d
e

Process finished with exit code 0
```

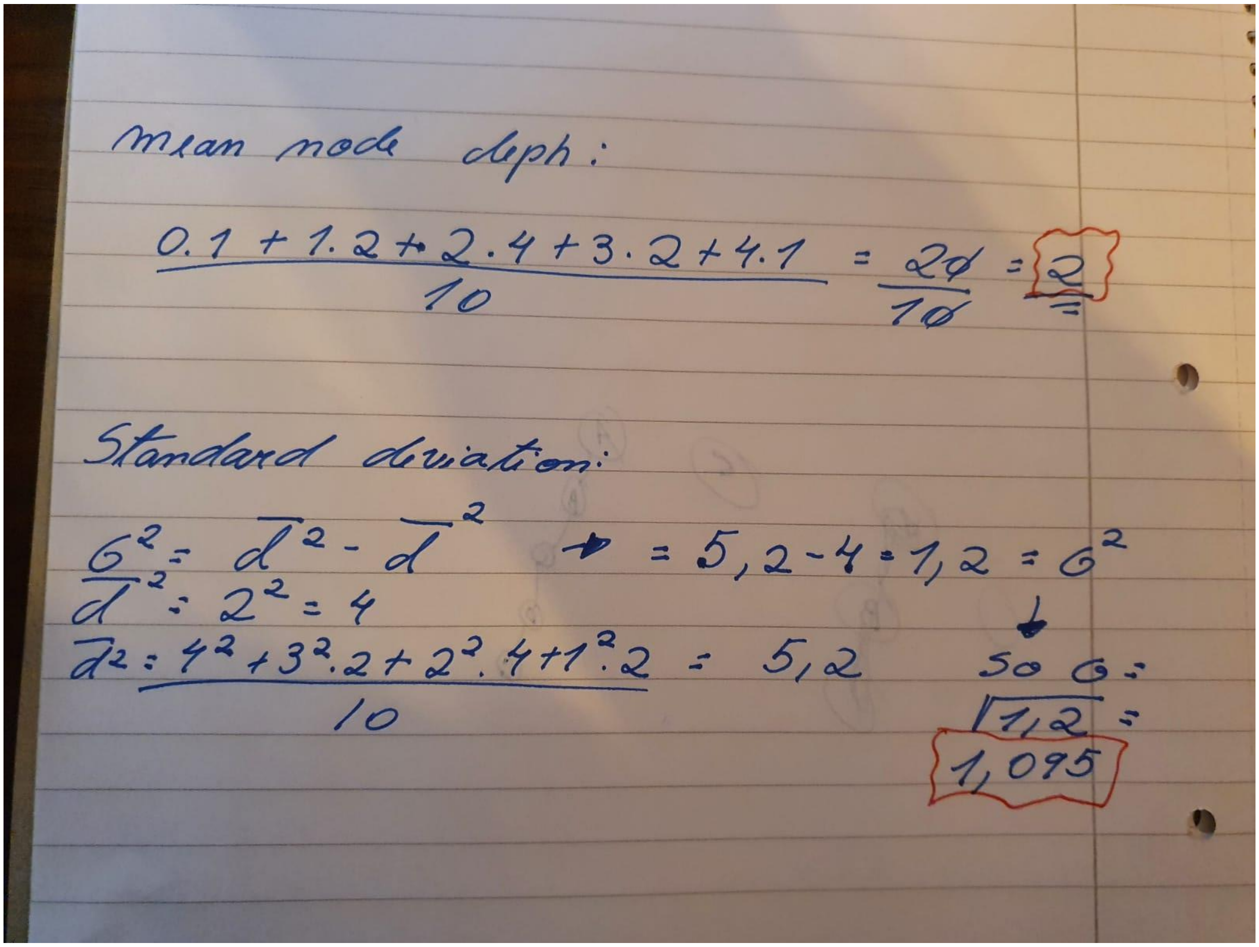
4)Tree:



We can see its maximum depth is 4.



Calculating the average node depth and the standard deviation:



mean node deph:

$$\frac{0.1+1.2+2.4+3.2+4.1}{10} = \frac{2\cancel{0}}{1\cancel{0}} = \boxed{2}$$

Standard deviation:

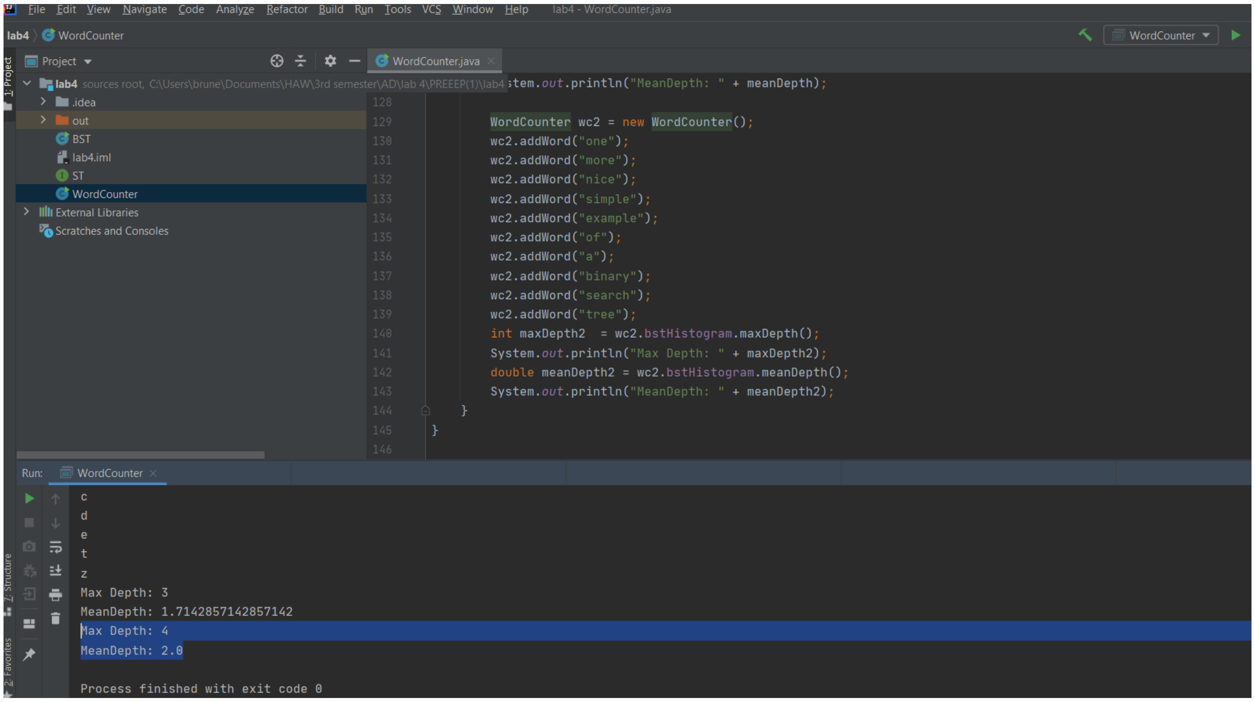
$$\sigma^2 = \overline{d^2} - \overline{d}^2 \rightarrow = 5,2-4 = 1,2 = \sigma^2$$

$$\overline{d}^2 = 2^2 = 4$$

$$\overline{d^2} = \frac{4^2+3^2.2+2^2.4+1^2.2}{10} = 5,2$$

so $\sigma = \sqrt{1,2} = \boxed{1,095}$

6)



```
stem.out.println("MeanDepth: " + meanDepth);

        WordCounter wc2 = new WordCounter();
        wc2.addWord("one");
        wc2.addWord("more");
        wc2.addWord("nice");
        wc2.addWord("simple");
        wc2.addWord("example");
        wc2.addWord("of");
        wc2.addWord("a");
        wc2.addWord("binary");
        wc2.addWord("search");
        wc2.addWord("tree");
        int maxDepth2  = wc2.bstHistogram.maxDepth();
        System.out.println("Max Depth: " + maxDepth2);
        double meanDepth2 = wc2.bstHistogram.meanDepth();
        System.out.println("MeanDepth: " + meanDepth2);
    }
}

```

```
c
d
e
t
z
Max Depth: 3
MeanDepth: 1.7142857142857142
Max Depth: 4
MeanDepth: 2.0

Process finished with exit code 0
```

I provided a method in BST called maxDepth that returns the maximum depth of the given tree. I also implemented a method called meanDepth which calculates the average of all the given tree depths.

2-3-4 trees and red-black tree implementation

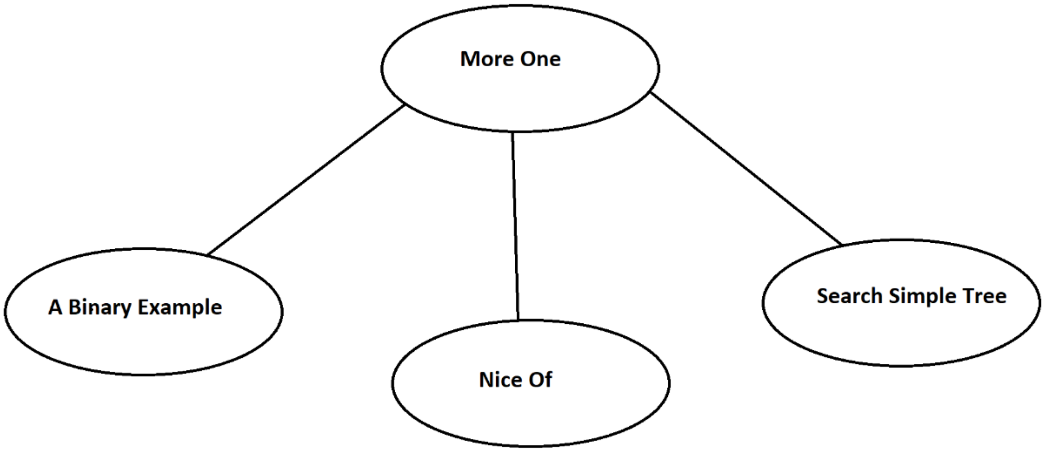
1) 2-3-4 tree invariants:

Every node (leaf or internal) is a 2-node, 3-node or a 4-node, and holds one, two, or three data elements, respectively.

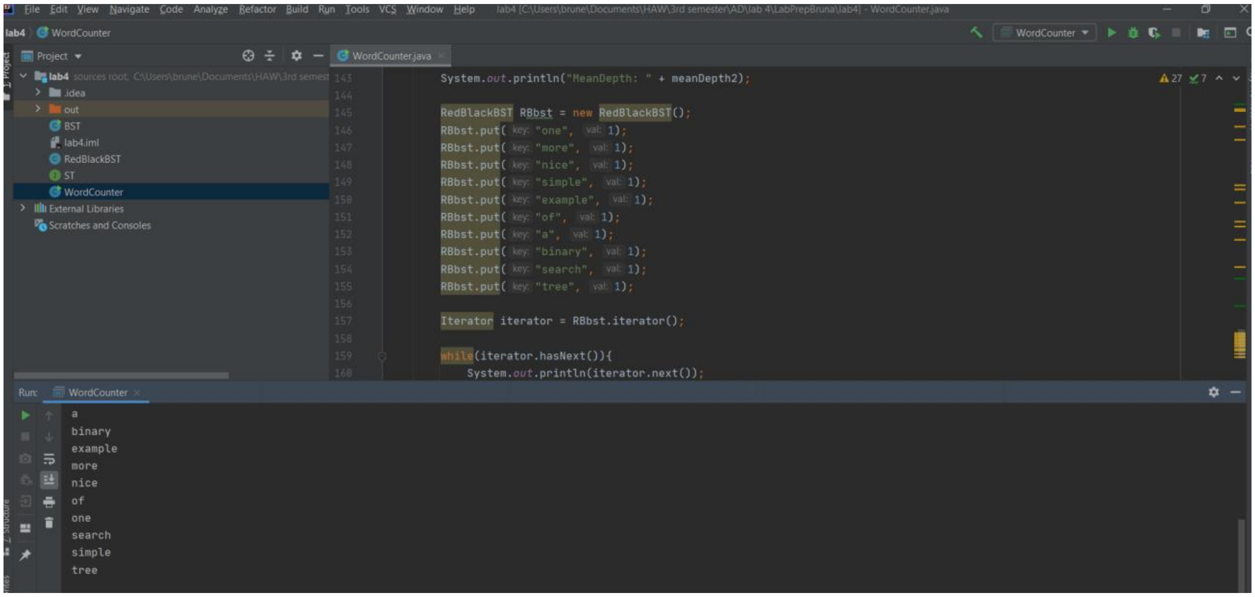
All leaves are at the same depth (the bottom level).

All data is kept in sorted order.

2) For the word sequence **"one more nice simple example of a binary search tree":**



3)


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
lab4 [C:\Users\brune\Documents\HAW\3rd semester\AD\lab 4\LabPrepBruna\lab4] - WordCounter.java
lab4 WordCounter
Project
lab4 sources root, C:\Users\brune\Documents\HAW\3rd semest
.idea
out
BST
lab4.iml
RedBlackBST
ST
WordCounter
External Libraries
Scratches and Consoles
WordCounter.java
Run: WordCounter

```
        System.out.println("MeanDepth: " + meanDepth2);

        RedBlackBST RBbst = new RedBlackBST();
        RBbst.put( key: "one",  val: 1);
        RBbst.put( key: "more",  val: 1);
        RBbst.put( key: "nice",  val: 1);
        RBbst.put( key: "simple",  val: 1);
        RBbst.put( key: "example",  val: 1);
        RBbst.put( key: "of",  val: 1);
        RBbst.put( key: "a",  val: 1);
        RBbst.put( key: "binary",  val: 1);
        RBbst.put( key: "search",  val: 1);
        RBbst.put( key: "tree",  val: 1);

        Iterator iterator = RBbst.iterator();

        while(iterator.hasNext()){
            System.out.println(iterator.next());
```

```
a
binary
example
more
nice
of
one
search
simple
tree
```

I implemented the class RedBlackBST that is a red-black binary search tree. Then tested it by iterating through the ordered word sequence given before.

References:

Lecture material from Dr.Prof. Renz

https://en.wikipedia.org/wiki/2%E2%80%933%E2%80%934_tree#Properties