UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science
Faculty of Engineering, Built Environment & IT
University of Pretoria

COS212 - Data structures and algorithms

Practical 3 Specifications: Splay Tree

Release date: 11-03-2024 at 06:00
Due date: 15-03-2024 at 23:59
Total marks: 190

Contents

1 General Instructions

- ***Read the entire assignment thoroughly before you start coding.***
- This assignment should be completed individually; no group effort is allowed.
- **To prevent plagiarism, every submission will be inspected with the help of dedicated software.**
- Be ready to upload your assignment well before the deadline, as no extension will be granted.
- You may not import any of the built-in Java data structures. Doing so will result in a zero mark. You may only use native 1-dimensional arrays where applicable. If you require additional data structures, you will have to implement them yourself.
- If your code does not compile, you will be awarded a mark of zero. Only the output of your program will be considered for marks, but your code may be inspected for the presence or absence of certain prescribed features.
- If your code experiences a runtime error, you will be awarded a zero mark. Runtime errors are considered unsafe programming.
- Read the entire specification before you start coding.
- **Ensure your code compiles with Java 8**
- The usage of ChatGPT and other AI-Related software is strictly forbidden and will be considered as plagiarism.

2 Plagiarism

The Department of Computer Science considers plagiarism a serious offence. Disciplinary action will be taken against students who commit plagiarism. Plagiarism includes copying someone else's work without consent, copying a friend's work (even with consent), and copying material (such as text or program code) from the Internet. Copying will not be tolerated in this course. For a formal definition of plagiarism, the student is referred to http://www.library.up.ac.za/plagiarism/index.htm (from the main page of the University of Pretoria site, follow the Library quick link, and then choose the Plagiarism option under the Services menu). **If you have any form of question regarding this, please ask one of the lecturers to avoid any misunderstanding.** Also note that the OOP principle of code reuse does not mean that you should copy and adapt code to suit your solution.

3 Outcomes

On completion of this practical, you will have gained experience with the following:

- Implementing a splay tree.
- Traversing and using a splay tree.
- Using key-value pairs for data.

4 Introduction

Splay trees are self-adjusting binary search trees. They are self-adjusting, as after every access, a series of rotations are performed, such that the node that was just accessed is rotated to the root. Self-adjusting trees are a variant of the move-to-root strategy which tries to create a more balanced tree. This strategy does not ensure that that tree is balanced, but its still efficient enough to be worthwhile. For this practical, we will be implementing a key-value pair splay tree. Thus, each node in the tree will have two values, a key and a value. The keys are used to ensure the BST property. For every key, there is a corresponding value. The keys are unique while the values don't necessarily need to be unique and also not in any particular order.

5 Tasks

5.1 Node

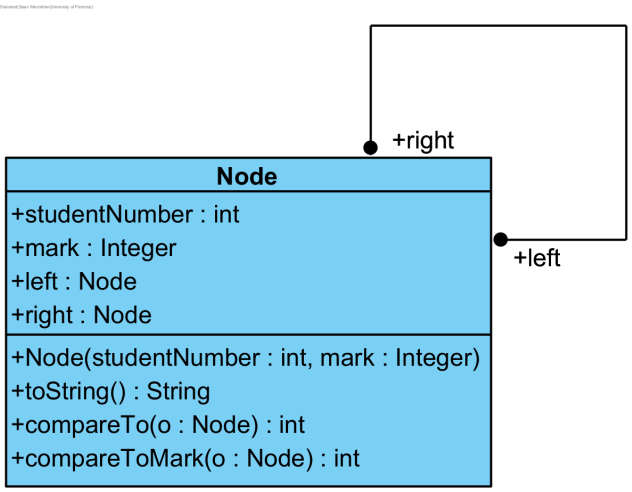

Figure 1: Node UML

- This is a *Node* class used in a key-value pair splay tree.
- The *studentNumber* is an int which is the key for the node.
- The *mark* is an Integer which represents the mark. Since this is an integer, it can be null. A mark of null is interpreted as a mark not read in yet. When comparing on marks, a mark of null is the lowest possible mark.
- This class is given to you, there is no need to change it. However, you can add members and functions if you need them.
- **Don't change any of the given functions as Fitchfork uses this to mark.**
- Node implements Comparable, this is setup to work with *studentNumbers*. Thus, when using *compareTo()*, the objects are compared based on student numbers. **Make sure you understand the difference between LHS.compareTo(RHS)==0 and LHS.equals(RHS)**.
- *compareToMark()* is also overwritten. This follows the same format as *compareTo()*, but this is for the *mark* variable. Note that *mark* is an Integer, thus it can be null. This is taken into account in the *compareTo()* function.

5.2 SplayTree

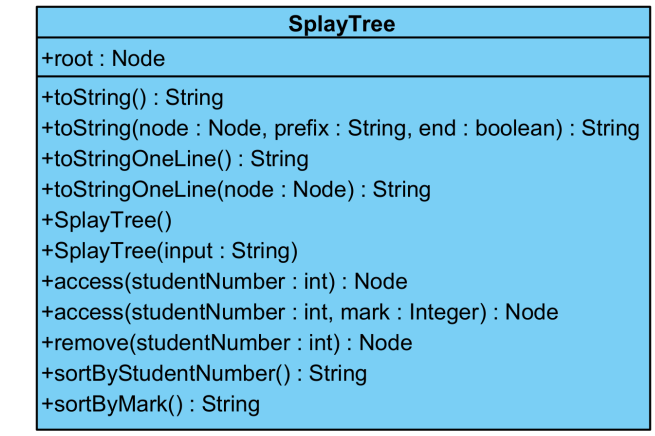

Figure 2: SplayTree UML

- Members
  - root: Node
    - This is the root of the splay tree.
    - If the tree is empty, this should be null.
- Functions
  - **toString()**, toString(node: Node, prefix: String, end: boolean) : String
    - These functions are given; **DO NOT** change them.
    - This is the normal toString function and a helper function for the toString.
    - This returns a readable version of the splay tree which can be used for visualization.
  - **toStringOneLine()**, toStringOneLine(node: Node, prefix: String, end: boolean) : String
    - These functions are given; **DO NOT** change them.
    - This returns a less readable version of the splay tree where the tree is squashed into one line.
    - This function is used for marking purposes, and also gives an easy way of reconstructing trees.
    - The string returned by this function is the exact string that should be passed in to the string constructor to recreate this splay tree.
  - SplayTree()
    - This constructor initialises the tree to an empty tree.

- SplayTree(input: String): String
  - This should initialise the tree to the passed-in string. There is no need to splay the tree for this function, just recreate the tree passed in.
  - You may assume that all input passed into the function is valid. This means that any input sent to the function which does not follow the rules below, should crash your program as it is illegal input.
  - This function recreates the output from *toStringOneLine* as a new splay tree. Thus calling the constructor and passing in the *toStringOneLine* creates a copy of the splay tree.
  - If the tree is empty, the following will be passed in

```
Empty Tree
```

  - A node is represented as follows:

```
{<node.toString()><node.left><node.right>}
```

    An empty node is represented as:

```
{}
```

  - Examples
    - If the root is a node with student number 10, mark of 50 and no children then the string will be

```
{[u10:50%]{}{}}
```

    - If the previous root had a left child with student number 5, mark of 40 and no children the string will be

```
{[u10:50%]{[u5:40%]{}{}}{}}
```

    - If the root in the previous example had a right child with student number 15, mark 60 and no children the string will be

```
{[u10:50%]{[u5:40%]{}{}}{[u15:60%]{}{}}}
```

  - Hints:
    - It is easier to implement this function recursively.
    - A node has the same number of open and closing curly braces.
- access(studentNumber: int): Node
  - This function calls the other access function with a mark of null.
- access(studentNumber: int,mark: Integer): Node
  - This function is used for insertion and accessing.
  - If the passed-in student number is already in the tree, then the corresponding node is splayed to the root of the tree using the same rules as the lecture slides and textbook. If the passed-in mark is null, then no changes should be made to the node. If the passed-in mark is not null, then update the mark of the node to reflect the new passed-in mark.
  - If the passed-in student number is not already in the tree, then add a new node with the passed-in parameters, according to the normal BST insertion rules. After the node is inserted, then splay the node to the root of the tree using the same rules as the lecture slides and textbook.

- remove(studentNumber: int): Node
  - The removal algorithm described below is similiar to the one found in the textbook. The steps are as follows
    1. Call the *access* function, using the passed-in student number. *Note that this implies that if a node that was not in the tree to begin with, is removed, that this node will be created and splayed. This is the intended behaviour. This means that calling remove with a student number not in the tree,* ***might change the structure of the tree.***
    2. Once the deleted node has been accessed, it will now be the root. The left child of this node is refered to as the left tree, and the right child is refered to as the right tree. The root of the original tree is then set to the left child.
    3. Find the largest element in the left tree. After this is found, then access that node, such that it becomes the new root of the tree.
    4. Once it has been splayed, it can't have a right child anymore, since it is the largest element in the left tree. Now, set the right child of the root node to the right tree.
    5. *Note that this algorithm is missing a few edge cases. You should try to find and implement them according to what makes sense. If you are unsure what the correct answer is, use the visualizer to see what the correct step is.*
- sortByStudentNumber(): String
  - This should return a string, where the nodes are sorted by student numbers in ascending order.
  - If the tree is empty, return the string

```
Empty Tree
```

  - If the tree is not empty, then use the toString of the nodes and concatenate them together. You should not add any formatting or spaces.
- sortByMark(): String
  - This should return a string, where the nodes are sorted by marks in ascending order.
  - If the tree is empty, return the string:

```
Empty Tree
```

  - If the tree is not empty, then use the *toString* of the nodes and concatenate them together. You should not add any formatting or spaces.
  - If two nodes have the same mark, then the node with the smaller student number must be placed first.

6 Testing

As testing is a vital skill that all software developers need to know and be able to perform daily, approximately 10% of the assignment marks will be assigned to your testing skills. To do this, you will need to submit a testing main (inside the Main.java file) that will be used to test an Instructor-provided solution. You may add any helper functions to the Main.java file to aid your testing. In order to determine the coverage of your testing the jacoco tool. The following set of commands will be used to run jacoco:

```
javac *.java
rm -Rf cov
mkdir ./cov
java -javaagent:jacocoagent.jar=excludes=org.jacoco.*,destfile=./cov/output.exec
    -cp ./ Main
mv *.class ./cov
java -jar ./jacococli.jar report ./cov/output.exec --classfiles ./cov --html
    ./cov/report
```

This will generate output which we will use to determine your testing coverage. The following coverage ratio will be used:

$$\frac{\text{number of lines executed}}{\text{number of source code lines}}$$

and we will scale this ratio according to the size of the class.
The mark you will receive for the testing coverage task is determined using table 1:

Coverage ratio range	% of testing mark
0%-5%	0%
5%-20%	20%
20%-40%	40%
40%-60%	60%
60%-80%	80%
80%-100%	100%

Table 1: Mark assignment for testing

Note the top boundary for the Coverage ratio range is not inclusive except for 100%. Also, note that only the function stipulated in this specification will be considered to determine your mark. Remember that your main will be testing the instructor-provided code and as such it can only be assumed that the functions outlined in this specification are defined and implemented.

7 Upload checklist

The following files should be in the root of your archive

- Main.java
- SplayTree.java
- Node.java
- Any textfiles needed by your Main

8 Allowed libraries

- None

9 Submission

You need to submit your source files on the FitchFork website (https://ff.cs.up.ac.za/). All methods need to be implemented (or at least stubbed) before submission. Place the above-mentioned files in a zip named uXXXXXXXX.zip where XXXXXXXX is your student number. Your code must be able to be compiled with the Java 8 standard.

For this practical, you will have 5 upload opportunities and your best mark will be your final mark. Upload your archive to the appropriate slot on the FitchFork website well before the deadline. **No late submissions will be accepted!**