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 2 Specifications - BST

Release date: 04-03-2024 at 06:00
Due date: 08-03-2024 at 23:59
Total marks: 275

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:

- Recursion
- Standard Binary Search Trees (BST) including:
  - Inserting data into BSTs
  - Simple traversals of BSTs
  - Removing data from BSTs
  - Special/Niche traversals of BSTs

4 Introduction

Binary Search Trees (BSTs) are a fundamental data structure in computer science, which organizes data in a hierarchical structure. This data structure is pivotal for efficient data storage and retrieval, allowing for binary search capabilities within a tree-like structure. BSTs are defined by a set of rules that govern the arrangement of nodes within the tree, primarily that each node has up to two children, and for any given node, all the elements in the left subtree are less than the node, and all the elements in the right subtree are greater than the node. This property makes BSTs incredibly efficient for operations such as search, insertion, and deletion.

BSTs are crucial for understanding more complex data structures and algorithms. Their basic operations—insertion, deletion, and traversal—form the foundation for more advanced structures like Red-Black Trees, AVL Trees, and B-Trees. Moreover, BSTs are widely used in applications that require fast lookup, insertion, and deletion, such as database management systems and indexing services.

This practical will guide you through implementing a standard BST in Java, focusing on recursion, insertion, traversal, and deletion. By the end, you will gain hands-on experience with these operations, enhancing your understanding of both BSTs and their applications in computer science. You may, as usual, add your own helper methods but you may not add your own member variables or extra classes.

Note that the output formats have been colour-coded for clarity. The colour coding is as follows:

1. Blue: Blue is used to represent value that must come from within the code. This could be function output or variable values
2. Red: Red is used to represent literal characters. These can be safely copy-pasted from the spec.
3. Purple: Purple is used for non-obvious formatting and punctuation. These includes commas and spaces.

The COS212 team has also created a visualiser for this practical so that you can interact with a BST. This visualiser is closely aligned with the memo code and as such should help to clear up any ambiguities (Questions like: "What if I insert the same item twice?") The visualiser can be found here `https://cos212.online/BSTClient/page.html`

5 Warning

For this practical, you are **only allowed to use recursion**. The following rules must be adhered to for the entirety of this practical. Failure to comply with any of these rules **will result in a mark of 0**.

- The words "for","do" and "while" may not appear anywhere in any of the files you upload.
- You are not allowed to create any extra classes, and not allowed to upload any java files not in the following list:
  - BinaryNode.java
  - BST.java

- Main.java

- Note that you are also not allowed to add extra classes in the allowed files. If the word "class" appears twice in any file you will also receive 0.

- You are not allowed to add any global variables to the classes, or change the global variables given. The only global variables allowed are as follows:

  - BinaryNode
    - public BinaryNode<T> *left*
    - public BinaryNode<T> *right*
    - public T *data*
  - BST
    - public BinaryNode<T> *root*

6 Tasks

You are tasked with implementing a Binary Search Tree using only recursion. **No iterative control flow statements (while, for, do while) are permitted**.The `BinaryNode` class has been provided. Your implmentation must adhere to the given class diagram (Figure 1).

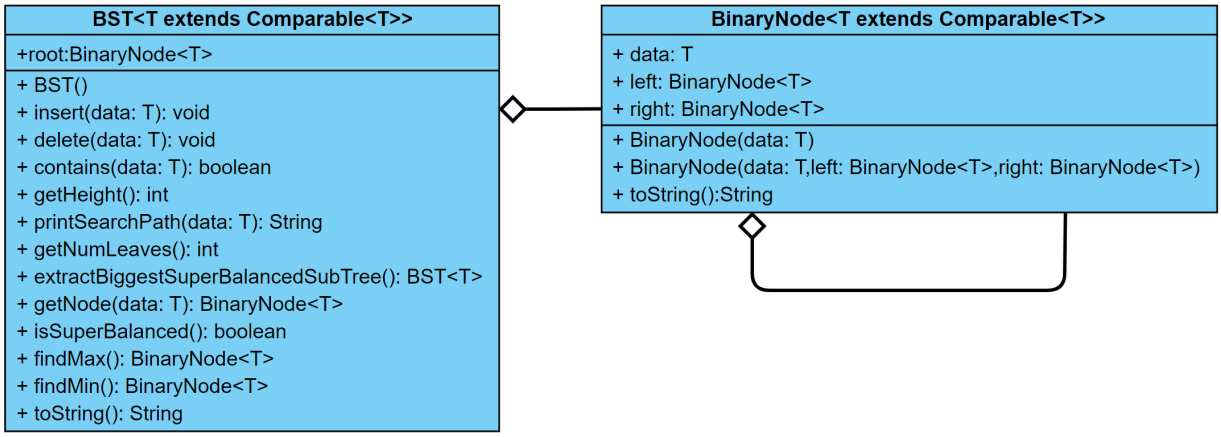

Figure 1: BST and BinaryNode UML class diagram

6.1 BST<T extends Comparable<T> >

- Members:

  - root: BinaryNode<T>
    - The root node of the BST.

- Methods:

  - BST()
    - Constructor to initialize the binary search tree with no nodes.
  - insert(data: T)
    - Inserts a new node with the specified data into the BST.
    - If the specified data can aleady be found within a node in the tree then the insert does nothing.

- delete(data: T)
  - Deletes the node with the specified data from the BST.
  - Ensure that you re-link nodes correctly after deletion.
  - Take note of the possible cases (leaf, one child, etc)
- contains(data: T):boolean
  - Returns true if a node with the specified data exists in the BST; otherwise, false,
- printSearchPath(data: T):String
  - Returns a string representation of the path to the node with the specified data.
  - The output should be the shortest path to the given data
  - The output should be in the following format:

```
<root_data><space>-><space><step1_data><space>-><space><step2_data><space>
    -><space>...<space>-><space><stepN_data><space>-><space><goal_data>
```

  - For example with the given tree (Figure 2) the path to 60 is:

```
100 -> 50 -> 75 -> 60
```

  - If the given data is not in the tree then go as far as possible and use `Null` as the `goal_data`. For example with the given tree (Figure 2) the path to 110 is:

```
100 -> 150 -> 125 -> Null
```

- findMax():BinaryNode<T>
  - Returns the node with the maximum value in the BST.
- findMin():BinaryNode<T>
  - Returns the node with the minimum value in the BST.
- getNode(data: T):BinaryNode<T>
  - Returns the BinaryNode containing the specified data, if it exists in the BST, otherwise returns null.
- getNumLeaves():int
  - Returns the number of leaf nodes in the BST.
- getHeight():int
  - Returns the height of the BST.
  - For this practical we will define the height of the BST as the maximum number of nodes that can be traversed to get to a leaf. An empty tree will have a height of 0, a tree with only a root node will have a height 1.
- isSuperficiallyBalanced():boolean
  - Returns true if the BST is superficially balanced; otherwise, false. For this practical a BST is considered superficially balanced if the number of nodes in the left subtree is exactly equal to the number of nodes in the right subtree.
- extractBiggestSuperficiallyBalancedSubTree():BST<T>
  - Extracts and returns the largest superficially balanced subtree (in terms of node count) within the BST as a new BST object.
  - If there are multiple superficially balanced subtrees with the same node count take the left-most one.
  - The biggest superficially balanced tree in figure 2 has 150 as the root node
- toString():String
  - Returns a string representation of the BST.
  - This has been provided for you. Pretty please don't change it.

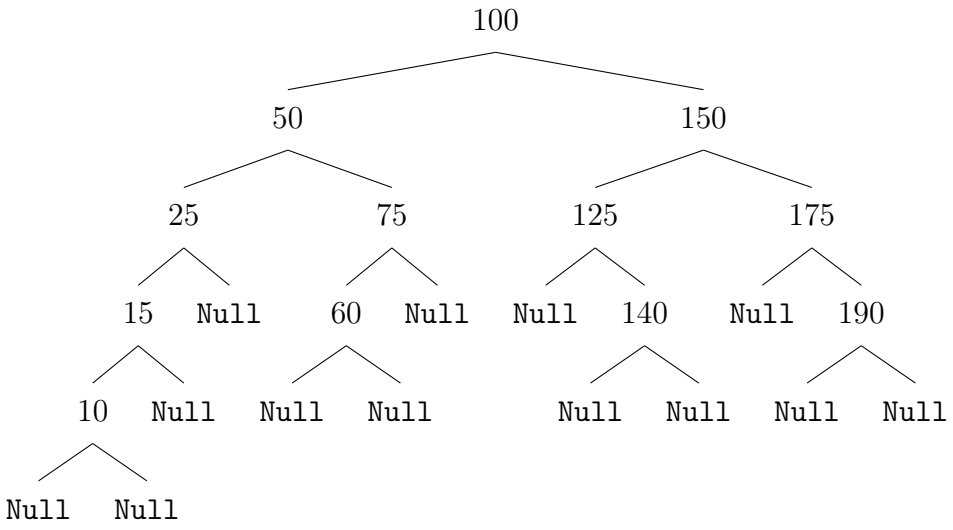

Figure 2: Example of a Binary Search Tree

7 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.cpp 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.

8 Upload checklist

The following files should be in the root of your archive

- Main.java
- BST.java
- BinaryNode.java
- Any .txt files you use for testing

9 Allowed libraries

- None, sorry :(

10 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!**