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: B-Trees

Release date: 03-03-2025 at 06:00

Due date: 07-03-2025 at 23:59

Total marks: 94

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 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.
- Note that plagiarism is considered a very serious offence. Plagiarism will not be tolerated, and disciplinary action will be taken against offending students. Please refer to the University of Pretoria's plagiarism page at `https://portal.cs.up.ac.za/files/departmental-guide/`.
- Please note that there is a late deadline which is 1 hour after the initial deadline, but you will lose 20% of your mark.

2 Overview

B-Trees are self balancing Trees that maintain sorted data and allows searches, sequential access, insertions, and deletions in logarithmic time. They allow each node to have more than one child.You will be creating a simple Database system using a BTree data structure to store information.

3 Tasks

You are required to implement a database system that implements a B-Tree structure. It stores container information for a freight logistics organization.

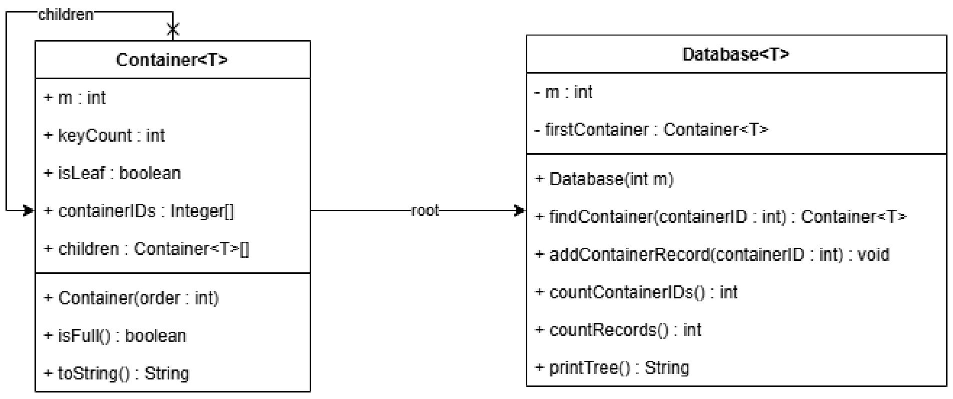

Figure 1: Binary-Tree Structure for Container Database

3.1 Container

The Container class represents a node in the B-Tree structure. Each container node can store multiple container IDs and maintains references to its child nodes.

3.1.1 Members

- **m**
  - The order of the B-Tree node
- **isLeaf**
  -Indicates whether this node is a leaf node
- **keyCount**
  -Number of keys currently stored in this node
- **containerIDs**
  -Array storing the container IDs in ascending order
- **children**
  -Array storing references to child nodes

3.1.2 Functions

- **Container**
  - Constructor that initializes a new Container node
  - Parameters: order - The order (M) of the B-Tree
  - Initializes containerIDs and children arrays of size M-1 and M respectively

- **isFull**
  - Checks if the node has reached its maximum capacity
  - Returns: true if the node contains M-1 keys, false otherwise

- **toString()**
  - Returns a string representation of the node
  - Pre-implemented, do not modify

3.2 Database

The Database class manages the B-Tree structure and provides operations for inserting and searching container records.

3.2.1 Members

- **firstContainer**
  - Reference to the root node of the B-Tree

- **m**
  - The order of the B-Tree

3.2.2 Functions

- **Database**
  - Constructor that initializes an empty B-Tree
  - Parameters: m - The order of the B-Tree

- **findContainer**
  - Searches for a container with the given ID
  - Parameters: containerID - The ID to search for
  - Returns: The Container node containing the ID, if the containerID isn't found, the node where it should be found is returned

- **addContainerRecord**
  - Inserts a new container ID into the B-Tree
  - Parameters: containerID - The ID to insert
  - Must maintain B-Tree properties after insertion
  - no duplicates allowed

- **countContainerIDs**
  - Counts the total number of container IDs in the tree
  - Returns: The total number of container IDs stored in the B-Tree

- **countRecords**
  - Counts the total number of Container nodes in the tree
  - Returns: The total number of Container nodes in the B-Tree

- **printTree**
  - Generates a string representation of the B-Tree
  - Pre-implemented, do not modify

4 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.

5 Upload checklist

The following files should be in the root of your archive

- Main.java
- Container.java
- Database.java
- Any textfiles needed by your Main

6 Allowed libraries

7 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!**