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

Assignment 2 Specifications: Advanced Trees

Release date: 31-03-2025 at 06:00
Due date: 02-05-2025 at 23:59
Total marks: 300

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

Splay Trees are variations of Binary Search Trees which move accessed elements to the tree's root. This allows faster retrieval of data accessed often since these will be higher up in the tree. One of the downsides of Splay Trees is that during normal operations the tree is often traversed twice. This problem is solved using Top-Down Splay Trees. Top-Down Splay Trees is a variation of a Splay Tree where the tree is splayed on the way down while searching for the node. This is more efficient since the tree is traversed only once.

Databases are organized collections of data that allows users to store, manage and retrieve data efficiently. Databases usually include indexed columns which are columns which are optimized for faster searching and retrieval. This is implemented by storing the data of that column inside a Data Structure that allows efficient searching, like Trees.

3 Tasks

For this assignment, you will be implementing a simple Database with indexed columns. The database will only be allowed to store Strings for simplicity reasons (*Note: This causes some problems when storing integers since the string "9" is greater than "80" which doesn't make sense for numbers but makes sense for Strings. This is the intended behaviour, so just be aware of this while debugging.*). The indexed columns in your Database will use a custom Data Structure called a "Forest of Top-Down Splay Trees". The operations for this Data Structure will be explained in the relevant section. The UML for this Assignment is given below.

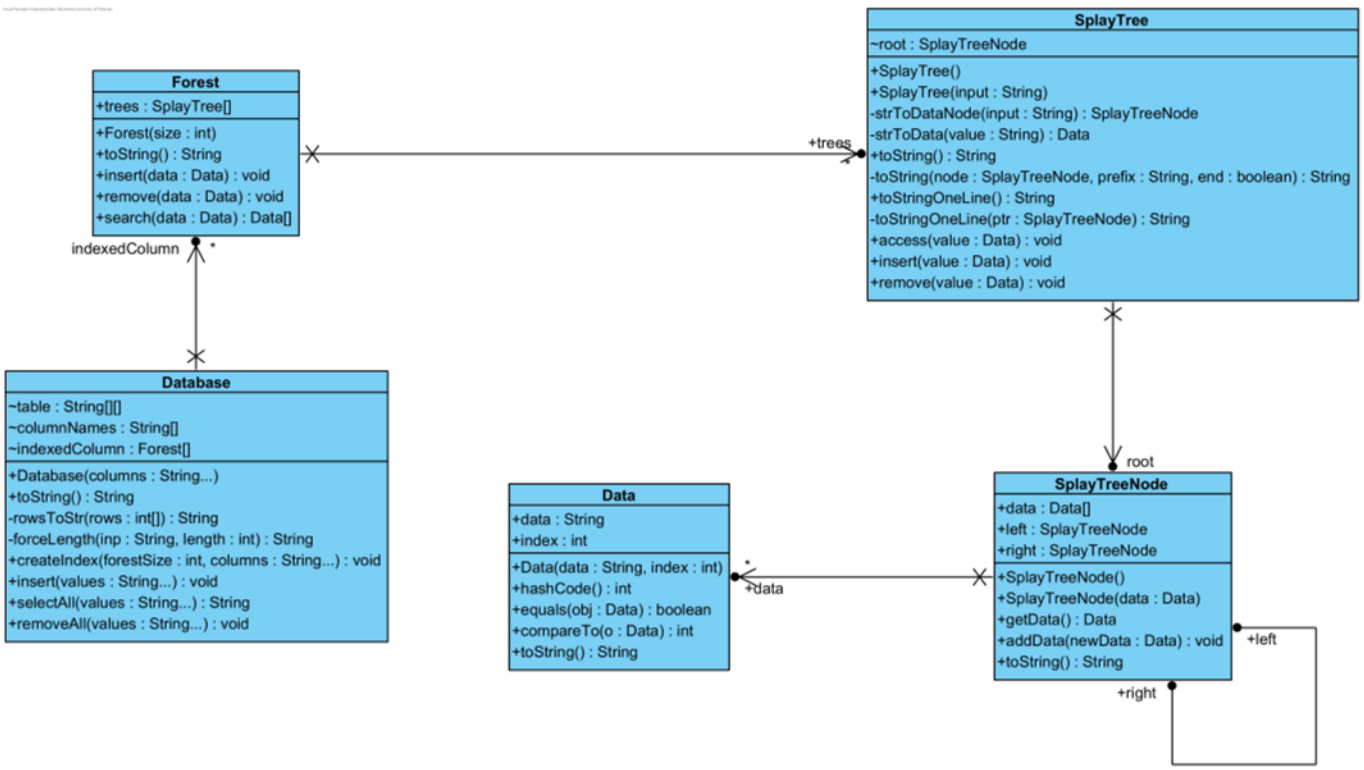

Figure 1: UML

3.1 Data

- This class is given to you. Do not make changes to this since it will be overwritten on FitchFork.
- An object of this class represents an entry in the final database.
- The data member of the class, represents the attribute of the entry.
- The index member shows which row in the database this data is in.
- This class has a `hashCode` function which will be used by the Forest class.
- The equals and `compareTo` functions have been overwritten. Both of these functions only look at the data. Thus, even when two data have different indices they will still be equal when they have the same data.

3.2 SplayTreeNode

- This class is given to you. Do not make changes to this since it will be overwritten on FitchFork.
- This represents a node in the Top-Down Splay Tree.
- Instead of storing a single Data variable, it stores an array of Data variables. This is to allow duplicates in the Splay Tree. When duplicate values are inserted, there will only be one node in the tree, but that node will have an array of Data variables.

3.3 SplayTree

3.3.1 Members

- root
  - This is the root of the Splay Tree. If this is null, then the tree is empty.

3.3.2 Functions

- Constructors
  - This class has two constructors both of which are given to you. The one creates an empty tree, while the other takes in the output of `toStringOneLine` and then reconstructs that tree.
  - Do not change the constructors since this will be used on FitchFork.
- String functions
  - You are provided with multiple string functions which should be used for printing and debugging reasons.
  - Do not change the constructors since this will be used on FitchFork.
- access
  - This should perform Top-Down splaying on the tree to bring the accessed node to the top.
  - **Warning:** This function will be called by other functions and classes. Make sure you only call this function when you are supposed to since calling this will modify the tree.

- insert
  - This will add the passed-in data to the tree. Use the Top-Down insertion algorithm to perform this action.
  - As was explained in the Data section, duplicate values are allowed and is handled by the Data class.
- remove
  - This will remove all occurrences of the passed-in data from the tree using the Top-Down deletion algorithm.
  - Note: Since this is Top-Down deletion, when deleting a value not in the tree, the tree can still change.

3.4 Forest

3.4.1 Members

- trees
  - This is an array of Top-Down Splay trees. The array will never contain null, but it is possible that some of the trees are empty.
  - When trying to add, remove or access data you should only look in one tree. The index of this tree is calculated using the following formula, where h is the hashCode of the data, and L is the number of trees in the forest:

$$\text{index} = |h| \% L$$

3.4.2 Functions

- Constructor
  - You are provided with a constructor that creates an array of Splay Trees using the passed-in size to allocate size for the array. All trees are initialised to empty trees.
  - Do not change this since it will be used by FitchFork.
- toString
  - You are provided with a toString function for debugging and printing reasons.
  - Do not change this since it will be used by FitchFork.
- insert
  - Insert the passed in data to the correct tree in the forest.
- remove
  - Remove the passed in data from the correct tree in the forest.
- search
  - This returns an array of Data matching the passed-in argument. If the data was not found in the correct tree, then return null.

3.5 Database

3.5.1 Members

- table
  - This is a 2D array which will be used to store the data.
  - The array can contain null rows. These are rows which were deleted but to prevent needing to update all indices in the trees, the deleted rows are shown as null. These rows will have null as each column value.
  - A row will either contain all nulls, or no nulls. It is not possible for the rows to contain some nulls and some normal values.
- columnNames
  - These are the names of the columns for the database.
- indexedColumn
  - This is an array of forests used to store the indexed columns.
  - If a column is not indexed, then it will be null. If the column is indexed, then there will be forest at that index.

3.5.2 Functions

- Constructor
  - This function is given to you. Do not change it as it will be used on FitchFork.
  - The constructor takes in a variable number of String arguments. This is treated as an array in the code.
  - The table is created with 0 rows, the `columnNames` are initialised and the `indexedColumns` are initialised such that no columns are indexed.
- String functions
  - You are provided with two string functions. Do not change this as it will be used on FitchFork.
  - The `toString` function can be used to print and debug your Database.
  - The `rowsToStr` function will be used by some other functions. It takes in an array of integers which represents the rows that need to be printed. It will add some formatting to the rows so that it is easier to read.
  - The `forceLength` function is a helper function for the `rowsToStr` function.

- createIndex
  - This should convert the passed-in columns to indexed columns. Use the passed-in forestSize to intialise the forest.
  - The passed-in columns variable is an array of Strings and can thus be used to index multiple columns simultaneously.
  - If some of the passed-in columns do not match any column names then ignore that value. The order of the passed-in parameter does not need to match the columnNames order.
  - If the column is already indexed, then do nothing.
  - *Hint: This function can be called at any time. If there are already values in the table, then these need to be added to the forest.*
- insert
  - **You may assume that the passed-in parameter contains the correct number of items. Your code should not check for this since only valid input will be used.**
  - If there are null rows, then insert the new row at the first null row. If there are no null rows left, then increase the size of the table by 1, and add then add the new row at the bottom.
  - Remember to also update the indexed columns.
- selectAll
  - This should return the result of calling the `rowsToStr` function with the rows matching the passed-in parameters.
  - The passed-in parameter is an array of conditions. The format will always be the name of a column, followed by `==`, followed by the value you are looking for. **You may assume only valid column names will be used and that the format is followed.**
  - If a column is indexed, then you have to use the forest to search for the value. If the column is not indexed, then you have to linearly search in that column.
- removeAll
  - This should remove all columns matching the passed-in parameter. The parameter follows the same format as `selectAll`.
  - When deleting a row, turn this index in the table to a null row, by making all column values null for this row. Remember to update the indexed columns.
  - The algorithm is as follows
    - Get an array of all row indices that needs to be removed.
    - Loop through that array and do the following for each row:
      - For all indexed columns, start by searching for the deleted value. Store this result.
      - Call remove on the forest.
      - The result from the search might contain results that need to be re-added to the forest, since these rows might not have been deleted. Thus loop through the search result and re-add all values to the forest, which does not match the row number of the row you are deleting.
      - Update this row to a null row.

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

- SplayTree.java
- Forest.java
- Database.java
- Main.java
- Any textfiles needed by your Main

6 Allowed libraries

- No imports allowed.

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 assignment, you will have 3 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!**