Programming assignment 2

Getting started

All your submissions should be implemented in Java. If you do not have a Java development environment, we recommend that you either use the Visual Studio Code Java extensions or Jetbrains IntelliJ. If you use IntelliJ, do not forget to register as a student, so you get access to the full versions. If you run Linux, macOS, or WSL, and want more control over your JDK, we recommend SDKMAN!

Note that you must solve all of the programming problems in Java. You are allowed to use Python (or any other suitable tool) to analyse the results, plot graphs, and so on. If you have discussed an alternative language with Morgan, it's fine to ignore the Java requirement and use, e.g., C++ instead.

Problems

Problem 1

Implement a generic deque (double-ended queue). In a deque you can add and remove from either end so that it can work either as a stack or a queue. Your implementation should at least support the following operations: `size`, `isEmpty`, `addFirst`, `addLast`, `removeFirst`, and `removeLast`. You should also implement an iterator for your deque.

Include reasonable error handling, e.g., throw exceptions when trying to remove something from an empty list.

You should use a linked list to implement your deque. Note that you are not allowed to use any of the existing implementations in Java; you should implement your own linked list with the appropriate operations.

Problem 2

Implement a randomized queue where each dequeue operation returns a random element from the queue. Your implementation should support the following operations: `size`, `isEmpty`, `enqueue`, `dequeue`. Note that dequeue should pick one of the inserted elements randomly, remove it and return it. You should also implement an iterator for your randomized queue. Note that the iterator should iterate over the elements in random order, returning each element of the list once.

You should use an array to implement your randomized queue. Note that the random iterator must be able to iterate over all elements in linear time. You are allowed to use additional memory.

Problem 3

Write a program that, given a starting directory/folder, finds all the files and folders of that directory recursively and represents them using a tree. Start by implementing a left-most child, right sibling tree. You can decide the API, but it should at least support finding a node by path/name and adding a child to an existing node. Then use the methods in `java.io.File` to find/list the files contained in the provided folder.

Problem 4

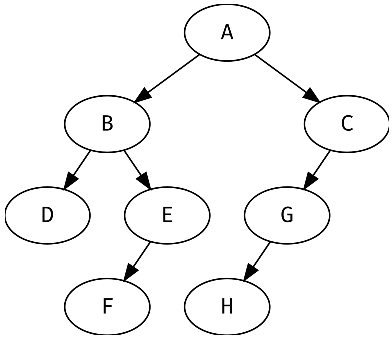
Implement a binary search tree that supports at least the following operations: `height`, `size`, `add`, `remove`, and `contains`. You should also implement in-, pre-, and post-order iterators. Once you are convinced that your operations work as expected, add an operation to remove the kth largest value in the tree, where k is a parameter.

For example, if you have a tree that contains the values `1` to `10` and we want to remove the third largest value (k=3), then eight would be removed.

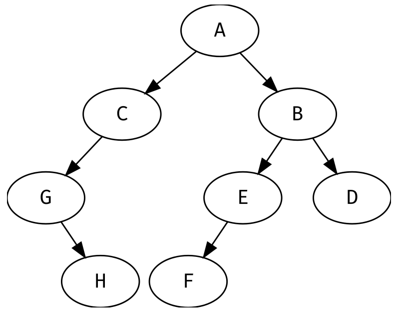
Your implementation should throw an exception if there is no kth largest value.

Problem 5

Two binary trees, T_1 and T_2, are isomorphic if T_1 can be transformed into T_2 by swapping left and right children of (some of the) nodes in T_1. Note that the structure and values in the tree should be considered. The two trees in Figure 1 are isomorpic since A can be transformed into B if you swap the children of A, B, and G. Implement a method that checks if two trees are isomorphic. It is ok to use a limited implementation of a binary tree, e.g., not containing general methods to add, delete, etc. What is the complexity of your algorithm? Provide an analysis that supports your answer in the report.



A



B

Figure 1: Two isomorphic trees

Problem 6

Implement one of the "balanced" trees discussed (AVL, splay, or red-black) and compare it to a binary search tree without effort to balance it. You should devise a reasonable way to create realistic average case trees (e.g., combinations of inserts and deletes) and then compare in terms of operation cost/time, height, etc. Describe and reflect upon your experiment and findings in the report.

Problem 7 (optional, for higher grade)

Huffman coding is commonly used for lossless data compression. Write a program that reads a text file, computes the frequency for each character, and creates a Huffman tree. You do not need to produce the compressed output, but your tree should provide a method to get the Huffman code for a character.

Submission guidelines

Submit your solutions as a single zip-file via Moodle no later than 17:00 on October 6, 2023 (cutoff 08:00 October 9). This is a group assignment that can be done in groups of one or two students. Your submission should contain well-structured and organized Java code for the problems with a README.txt (or .md) file that describes how to compile and run the Java programs and a report in PDF format that describes the findings from problems 5 and 6. It is sufficient if one person in the group submits the work to Moodle, but make sure that the submissions contains all your names.