Resit Assignments Sorting & Searching

What to do?

If you failed one assignment you have to repair this assignment. It will be graded using the same criteria as before.

If you failed two or more assignments you have to repair those assignments and have to do an extra assignment which combines the subjects from the whole course. They will be graded using the same criteria as before and the extra assignment will be graded as described in the following sections.

When is the deadline?

The deadline for the resit is in week 8 block 3, Friday March 29th 23:59.

Where to hand in the assignments?

On Moodle there will be four dedicated assignments, named Assignment-1-Resit, Assignment-2-Resit, Assignment-3-Resit and Assignment-Extra-Resit.

Extra assignment

HuffmanCompression

For the extra assignment you need to determine the compression-ratio for a Java source file using the Huffman compression method. On the VLO there is a project called Assignment-`Assignment-Resit` that contains the files and code to get you started. The class `HuffmanCompression` has three methods that you must implement. You are allowed to add extra methods, please be careful with the access-modifiers. Don't make every method `public` just because you are used to. There is also a `Node` class which you must use when constructing the Huffman tree. Also the class `Node` contains three methods that needs a proper implementation.
For both classes there are unit-test classes that only contain some very basic tests. It is of course also your task to add extra tests that will ensure that your code works properly.

For calculating the compression-ratio you need to know many occurrences there are for every unique character in the text and how long (measured in bits) the code is for that character. Please keep in mind that everything IS a character when compressing text. Not only the characters form the alphabet, but also slashes, exclamation points, carriage-return, line-feeds, and horizontal tabs are characters. You don't have to consider the translation table itself!
Assume that 1 character needs 8 bits in the original text!

Once you have determined the number of occurrences for every character you can build the Huffman-tree. For this you must use the provided `Node` class.
With the Huffman-tree at hand you should be able to determine the compressed code that will be used when compressing the file.

Knowing the number of occurrences of each character and the code that will used when compressing the text you can determine the compression-ratio.

Node

This class must be able to write itself and its children to an `ObjectOutputStream` and reconstruct itself using from an `ObjectInputStream` in ***pre-order***. In other words, implement the `write` and `read` methods so the tests in `NodeTest` don't fail anymore. Please don't change the test method itself. If it keeps failing you have not implemented the code correctly!

Report

The report contains a brief description of the written code in plain English[1] or Dutch[2] (avoid technical mambo-yambo).

Tips

- There are some methods in the class `HuffmanCompression` that have the default-scope (package private), you can use these methods to test parts of your code during development.
- When implementing the methods in the `Node` class there is a tricky difference between a leaf and a node. When writing a leaf you can use `output.writeObject(character)` and when writing a node you can use `output.writeObject(null)`. When reading from the inputstream you then can distinguish the difference between a node and a leaf by checking for a `null` value returned by `input.readObject()`.
- Try to use recursion since this tends to make your code very elegant and keeps the methods short.
- While grading your report and your code only the classes `Node`, `HuffmanCompression`, `ExtendedHuffmanCompressionTest`, and `ExtendedNodeTest` will be inspected and used, so if you have more classes state that clearly in your report.

Grading

Report must be in PDF format.
The report adheres to the requirements stated in the study manual.
All code is present in the uploaded ZIP-file.
The nodes are written and read in pre-order.
All the tests in the supplied (Assingment-Resit.zip from the VLO) code pass.
Extra tests that proof that the methods are implemented correctly.
The extra tests all reside in the `ExtendedXxxTest` classes.
Any extra classes are clearly stated in the report.

[1] If your lecturer doesn't speak Dutch fluently.
[2] If your lecturer does speak Dutch fluently.