Lab 6

Mouna Kacimi, Werner Nutt, Camilo Thorne

Lists and Binary Search Trees

The purpose of this lab is to apply the list and binary search tree data structures to manipulate files (.txt files to be more precise). Dynamic data structures are more convenient that static data structures (such as n-dimensional arrays) in this case, since at any point in time, files might be (i) added, (ii) deleted, (iii) modified or (iv) accessed. Lists and binary search trees allow for such operations to be efficient w.r.t. both the time and space resources they require.

In what follows we will represent files as pairs $(\text{name}, \text{path})$ where name is a string containing the file's name or identifier and path its absolute system path.

1. In this exercise we will implement a file index structure based on lists:

   (a) Modify the `List` and `Node` datatypes and structures so that they deal with files. Notice that Node keys are not anymore integers, but strings (filenames).

   (b) Modify the methods `void addSorted(Node n)` and `void insertsort()` to sort files in *lexicographic order*.

   Hint: Implement a comparator for your new `List` structure; use the comparator native to the Java `String` class.

2. In this exercise we will implement a file index structure based on binary search trees. Modify the `BinTree` and `Node` datatypes and structures so that they deal with files.

3. Modify again nodes, lists and binary trees into *generic* datatypes and structures, viz., implement (via Java generics) the classes: `Node<E>`, `List<E>`, and `BinTree<E>`, where `E` denotes an arbitrary class. Which methods will give problems, and how can we deal with them?