Implementation and comparison of 4 different data structures: AVL Trees, Red Black Trees, Splay Trees, 2-3 Trees, Binary Heap

Antonio Recalde Russo, Rafael Flores Gonzalez
Our demonstration video : https://www.youtube.com/watch?v=gzpxBLTHQbw

Overview
We implemented four different data structures, and performance tests for each. This project. Here is a summary of the implementation details, results and conclusion.

Instructions
In order to run our implementation, follow these steps:

1) unzip files
2) compile with "make"
3) for performance comparison: run "./time_test.native n op"
   where n is the number of elements to be inserted in the tree, and op could be a string "o" for ordered elements, "r" for random elements, and "ns" for elements that can only range from 0 to (n/1000 + 2)
4) you may also run "./testing.native" for the structural tests

Planning
We consider that our original planning was efficient. We stuck to the instructions and advice from our TF. This helped us manage the complexity and time required for this project. Here are our initial draft specification and final specification.

How it all worked out overall:
We have successfully implemented the most fundamental milestones. Our TF provided excellent advice on the magnitude of the project. Thanks to that, we have successfully implemented the most fundamental parts of the project. We realized that the things we initially intended to do were easier said than done.

Milestones:
- Create the data structures: AVL tree, Red Black tree, Splay tree (completed)
- Add other trees: Binary Heap, 2-3 trees. These are largely based on what we did in class (completed)
- Enable performance testing -time_test.ml- (completed)
- Add graphic user interface (not completed)

Design and implementation

Our experience with the design aspect of this project helped us understand the true value of this course. The transition from psets, where we would in the most basic sense just complete the missing code, to creating our own project form scratch was not particularly easy.

For this project, we used the OCaml programming language. We find that out of all languages available, the "type system" in OCaml made our program less bug-prone than implementations in loosely typed languages. In fact, our opinion about the amount of time spent on debugging errors was considerably less than our experience debugging in other languages.

Perhaps the most important aspect of our implementation was the use of interfaces. Working as a team, we do not always have full knowledge of the code written by a teammate. The use of interfaces, in this sense, allowed us to always stay on the same page.

Testing was the hardest part of the program. I think that testing using values hidden behind the abstraction barrier is very time consuming. It would have been nice if OCaml had a testing mode (maybe it does) that simplifies everything.

Each group member's contribution to the project:

- Antonio: AVL Trees, Splay Trees, 2-3 Tree.
- Rafael : time_test.ml, Red Black Trees.
- The remaining parts of the project were worked on conjunctly.

If there were more time: We would implement a GUI to allow easy visualization of our data structures.

Reflection

Most important thing we learned: Soon after the start of the project we realized that the use of abstraction is does not only make the code easier to read, but it also makes the work easier whenever major structural modifications need to be introduced. We realized that creating an abstract skeleton of the project translated an economy in terms of time and effort.

Things that worked great: We were able to compile. We did not have too many bugs. We implemented a divide and conquer strategy.

Things that worked badly: We were not able to create a GUI. The abstraction barrier did not give as much freedom when it came to testing. We wasted a lot of time in addons that we ended up not implementing due to lack of time.

How we would do things next time: We would plan our timeline and time management more efficiently.

Advice for future students
Make use of all the concepts and ideas we learned in this course (especially abstraction). They will make the work so much easier. Basically, they are granting you more life, as you’ll have more time to do other things.