Summary

Challenges

We found the most challenging parts to be insert and remove withing the 2-3 tree implementation. This was because the way we wrote it, we had to understand how the 2-3 tree worked, and then had to exhaustively pattern match against all cases, while simultaneously try to group cases together in order to reduce the amount of code to write. Additionally, we struggled a bit getting our different testing files up and running, as we were not completely familiar with the module system, and how to instantiate different modules to run tests on them.

Design Decisions

One interesting design decision that we made was to implement insert and remove separately. While we could have written insert by first calling remove to make sure the item we were inserting in the tree was not there, and then writing an insert function that inserted items that we knew wasn't in the tree, we decided to make them completely separate. This was done in order, to first make sure that if remove had any errors in it, the insert function would still work, and second, to make insert more efficient, as if the key was still in the tree, we could just replace the value. We also worked hard to make sure that we were able to simultaneously work on the project, without the need to be using the same computer. This meant that we worked on separate files, so that when it was time to push changed to git, we did not need to worry about merge conflicts or the like. A trade-off to this that neither of us has a 100% knowledge of the others implementation.

Major Issues

We did not run into any major issues while working on this project.

Known Bugs

We are not aware of any bugs in our solution.

Specification

We added comments above any code that was not included in the .mli files in order to complete the specification for our project. We tried our best to mirror the comment specs given with this project and past projects, and to be clear, yet concise with our descriptions of our code.

Design and Implementation

While most of the design of the project was given to us, in order to implement it, we made some key choices. Firstly, whenever we could, with the exception of not using delete in insert for the 2-3 tree implementation, we made sure that if a part of of one function could be used to implement another, we did so in order to reduce the amount of duplicated code. We did a top-down approach, where we first defined what the data structure was, and then made sure that, given the tree was given in the correct structure, each function did what it was supposed to do. This comes with numerous advantages and disadvantages. Specifically, each function is very short and clear – it takes in a certain input and returns the correct output. However, it is very sensitive to bad input because of this – besides the functions rep ok to test whether an input is correct, each function does not test the structure of inputs, and therefore if given bad input, there are no guarantees of what happens. For example, in data.ml, I implemented an exception called Unreachable. This was done so that all pattern matching was complete, but given good input, the function would never reach that part of code. Thus, this gave a good cue that if ever the exception Unreachable came up in testing, I either gave the function bad input, or the function was wrong. For engine, time was taken to plan out how helper functions would be implemented, and how the Dictionary and Set modules would be used to simplify code within the Engine. Additionally, we made sure to communicate with each other problems we faced, or bugs we had, so we could each help troubleshoot each others code.

Testing

For testing, We decided to not using huge files in the interest of time to run code - but still tested on them, mostly in utop, to ensure that our implementations worked for large inputs. We worked to ensure that testing was complete – especially on functions that were more complicated, like insert or remove, while is empty and empty did not need quite so thorough testing. We tested exceptions in utop to make sure they were thrown. We did attempt to use assert_raise with exceptions, but in the interest of time, did not add all of these cases to our test files. We worked to catch edge cases and corner cases of every function we wrote, and tried to include these in our tests. For Engine, we tested in a manner similar to we have in past projects, with many assert statements called with expected inputs and outputs, generated by running functions we implemented. For Data, we decided to use the test harness to test our own code, in addition to the fact that it will be used for some buggy Dictionary implementations. For the test harness, we tried to include some string based tests, and also some integer based tests, knowing that a Dictionary can be used on any type and it needs to be certain that it works on multiple types. Overall, we tried to use a methodical and repeatable approach to testing our code, so that we know where test failures come from and can conclude with confidence that our code works once it can pass all of our tests. We felt our tests were successful in showing that we had implemented our solution correctly.

Work Plan

In order to divide the work, we decided that George worked on all the functions in data.ml, the basic list implementation, the set dictionary, and then the dictionary implementation using 2-3 trees, while Eric worked on implementing engine.ml, as well as doing the majority of the testing. This minimized the risk of redesign, as each person had full control of the modules that they implemented, thus design decisions would not clash as any sort of specifications that each person decided on for each part would not affect the other. Some key dependencies of doing it this way was that Eric could not test parts of engine until the set dictionary was implemented, and this could have created a problem where bugs and issues with that would not have been revealed until later. Additionally, this was useful in that we could work simultaneously and commit code to git without worrying about merge conflicts in our source control. Also, because we each had very specific roles within the project, we were able to set reachable deadlines for work and able to make sure we consistently made progress towards finishing the assignment.

Known Bugs

We are not aware of any bugs in our solution.

Comments

We spent approximately 20 hours total working on this project. Within data.ml, approximately half was dedicated to designing (specifically designing our implementation of insert and remove from a 2-3 tree so that it correctly models the behavior of a 2-3 tree) and then half was dedicated to coding and testing now that the behavior of these functions were clearly understood. Some advice that you should have given before we started would have been how to access and deal with the modules we needed in our code. Once the functions were written, we had trouble instantiating some of them, which was a bit of an annoyance to understand. Working with a partner was a new challenge we encountered in this assignment. Making sure that we split up the work in such a way that using git was never a problem, making sure that we coordinated such that each file was only being edited by one person at once, and ensuring that we were not waiting on each other for certain functions to be done were all minor obstacles that we successfully overcame during working on the project. In addition, if one person had a bug in their implementation, and it was giving the other person errors, this was hard to resolve. It was surprising how easy to divide the work while working on the project. Due to the modules used, as long as each person knew what had to be done, because of how segmented it was, we successfully got it done, and were able to use each other's work in our code without worrying about how each other had implemented the functions. This is a pretty cool features of modules and we definitely understand now why they are used for developing projects that are used by many people. I liked the finished working assignment, and being able to see how effective it is for searching through the documents

in a directory. I think I would make the "onboarding" process for this assignment a bit simpler, as it was definitely hard to get up and running with the implementations needed until you had a good understanding of the module system and how everything works together in the code.