GovDataMiner: Scraping through Data using Skip Lists

Anjali Fernandes, Camille Rekhson, Raul Jordan

Link to our Demo Video
https://www.youtube.com/watch?v=SsU9879ZgaQ&feature=youtu.be

Overview

We implemented the Skip List data structure as a way of storing big data from data.gov and making an interface for searching through and filtering those large data sets for the information that a user needs, and using our code to output the data as an easy to read CSV file for users to download.

Planning

We had planned to create an implementation of skip lists that allowed us to store 3 different government data sets, and that contained the most common functions associated with working with such a data structure (such as find, insert, remove, contains, etc.), and planned to take in some sort of user input. First, the user would choose which data set to filter through, and then he/she would provide information on how to filter the data, and we wanted to return a filtered skip list that the user could use in some way.

Here is our draft specification and our final specification

We had also planned to create a user interface that would allow us to take in this user input and make it easy to navigate for our users and not make them worry about the implementation, but rather make it streamlined and efficient for them to obtain their desired output.

Overall, our abstractions allowed us to easily work with the three data sets and create functions that filter through them easily and our code structure allowed us to be efficient and change small parts of the code while keeping the rest of the functionality as expected. In the end, we came up with a nice web interface running on our local servers to use as the user front end, and it works quickly and correctly.

All of the milestones turned out to work as we expected, and we ended up designing a clean interface, and implemented all of our functionality in as few modules as we could, while still writing scalable code that was testable and efficient.

Raul worked on a lot of the skiplist class interface and the builder and filter implementations, Anjali developed the structure of the project and testing interfaces, Camille helped on the details of the user interface and the Flask framework.

Design & Implementation

Our experiences with building this project were much different from the previous work we have done in the course. Our entire group had little experience with Python aside from small standalone scripts for text parsing or other similar programs we had made in the past, and had no idea of how to start a large project in the language. We had it clear that we were going to implement the skiplist interface as a class with methods defining each of its useful functions, and our experience with modules and functors in OCaml came in handy, and it was not too difficult to code the abstract skiplist class.

However, we had no idea of how to build the skiplists for all three of our data files at first, so we implemented three classes that inherited from the skiplist class and added their own specific functions, but we noted that this was repeating a lot of our old code and didn't seem as efficient or scalable. In the end, we ended up structuring our project around two modules called **Builder** and **Filter** which are given a set of parameters and build and filter skiplists regardless of the data file they're given. This allowed us to fix some of the more annoying bugs we encountered, such as incorrect indexing or incorrect filtering of the skiplists, in a way that was easy to understand and without having to go through many files and change code for all of them.

In the end, we tested the project as a standalone command line module with a main.py function and all of the skiplists were built correctly and efficiently. We had initially done tests using the python nosetests package, but seeing that our source code was getting bigger and bigger, we decided to eliminate the use of the package altogether and use python's if __name__==’__main__’ feature to write tests in our main files and make sure that the tests passed if the files were run as standalone programs. This allowed easier control of our tests.

One of the other great challenges was creating the user interface, and we managed to learn the Flask microframework for this. Putting together the html/css was easy, but integrating our algorithms with the user input was hard. We had trouble managing the POST requests from the html and ended up coming up with a good solution. We made a helper functions file that allowed us to manage these problems through a lot of functions that worked around the annoying details and helped us figure the rest of the project out. All in all, our levels of abstraction and testing helped us to keep things much simpler than they would have been otherwise, and our project ended up working sooner than we expected.

Reflection

We were surprised at how much we could accomplish with a simple idea and little knowledge of a new programming language in a few weeks. We chose this idea knowing in advance that creating software always takes twice as long as expected, and pacing ourselves allowed us to create good tests, and document our code more elegantly for other programmers to understand better and make our code maintainable.

We were really surprised at the amount of tools that are available in python and how many complicated details that get in the way of implementation in other languages are not that great of a concern in python.

If we had more time, we would have implemented our additional features and explored other python libraries or other ways to make our functionality simpler, and perhaps implemented some sort of concurrency into our skiplist class.

Advice for Future Students

We would advise to come up with an idea that is interesting enough by itself, without having to implement a lot of complex or additional functionalities. This allowed us to work on making our project more efficient and comprehensive by not having to worry a lot on many additional features.

We would also advise to choose a language that is right for the tool. Even though python is great for our project because of its many text-parsing capabilities, it may not be the best for asynchronous programming or

concurrency (use javascript!). However, we met the goals we set for ourselves and are satisfied with our results!

If we would start from scratch, we would definitely pay more attention to building more efficient interfaces and not repeating ourselves too much, but we would definitely use python for our project and the Flask framework, which made things really simple in the end.