Homework 2: Routy

Patrik Zhong

September 22, 2021

1 Introduction

In this homework assignment we implemented our own router by implementing 5 different files, namely map.erl, dijksta.erl, hist.erl, intf.erl and routy.erl. These modules interact with eachother by using the dijkstra shortest path algorithm to find the least amount of "hops" needed to get from point A to point B and a mapping of the nodes, and the implemented router utilizes the interface and history of the nodes to come together in our final router.

1.1 Implementing map

This was a rather straightforward part that eased us actually coding and building something in erlang. After having freshed up a bit on functional programming and the correct syntax it went on quite well. It wasn't until foldl/3 showed up when implementing the allnodes function that I got a bit stuck. By realizing that the foldl basically was a loop that just accumulated things in a list, I solved it by basically making every tuple in the tuplelist into a list in itself, and then I made sure that each element in the list was unique relative to things I had in the accumulated list.

1.2 Implementing dijkstra

This was the without a doubt, hardest part of the entire assignment. I was stuck on this part for days, not realizing how I would have to structure my pattern-matches and what types of cases I would deal with. Even then, I kept getting stuck with seemingly trivial issues, such as hitting infinity when I really wasn't supposed to, or not going deep enough in the "graph". I honestly would've been stuck here for days and not completed the assignment if it wasn't for Klas Segeljakt, who showed us in an extra lecture how dijkstra could be implemented.

1.3 Implementing Interface and History

Both of these were a very relaxing and slow cruise relative to the previous dijkstra module and were implemented without any real issues. I did have

to rectify my history module because I didn't process it correctly, but this was something I noticed later on when implementing the router. Common faults like not having brackets around a variable you want to return were way more common in erlang than in Java I felt, because Java is so much stricter with typesafety. Parts of me realized that statically type languages are both a luxury in the sense that they remove a lot of uncertainty of what functions actually output, but if you're a good programmer, which I am not, then I can understand that these restrictions are quite limiting.

1.4 Implementing Routy

This part was maybe not the hardest when it came down to code, as we very little own code. However, it was still the hardest part when it came to actually understanding what we were implementing. Just implementing the status fetch was a bit tricky, as one really had to think how the different "cities" would respond to one another. It was also the part where I looked the most at external sources, such as googling for link state and thus understanding the code. The testing setups were a bit hard to do unless you checked examples from further down and understanding how the pid's interacted with eachother. When implementing the adding of roads between cities, it was easiest to check what happened by just printing out how each city's table changed depending on how we added roads between them and what each node broadcasted. Also building a test-case was a bit tricky in the end, because it required coming up with many different scenarios and painting your own graphs of how you wanted your cities to connect and/or if a message would be able to come through. I also encountered a very silly bug in this module that traced all the way back to my interface implementation.

2 Conclusions

This was a considerably harder assignment than the last one, with us having to implement our own code and all. This was however a very good learning experience, where I not only got a good actual introduction to Erlang, but also how routing in a smaller scale works and how Erlang basically was built for this. This really helped me understand the next part of how we routed the messages between them, depending on what information they had.