CSC 464 Assignment 2
Part 3

Sterling Laird - V00834995

November 26th 2018

1 Introduction

For my project, I have chosen to implement a peer to peer collaborative document editing system. For the third part of assignment 2 I decided to implement some of the basic functionality needed for my project and attempt to test it to show that it performs as desired. This report covers the portion of my implementation thus far, and outlines the steps I have taken towards testing it.

All code in this assignment was written by me, Sterling Laird with most of my knowledge of the algorithms implemented coming directly from *Logoot: A Scalable Optimistic Replication Algorithm for Collaborative Editing on P2P Networks* (Weiss, Urso, & Molli, 2009) and from various online sources outlining the algorithms used to solve these problems such as *stackoverflow.com*. All implementations can be found on my Github page (https://github.com/sterlinglaird/CSC-464).

2 Implementation

While researching approaches for my peer to peer document collaboration project, I discovered that the two main approaches are using Operational Transformations (OT) and Conflict-free Replicated Data Types (CRDT). While each approach has advantages and disadvantages, I decided on exploring CRDTs as I had previously been aware of OT methods, and the idea of working with a new concept (to me) was enticing. The CRDT that I decided to implement was the Logoot approach from the paper mentioned previously. I made this decision because the algorithms are not as complex as in other approaches and the paper appeared easier to understand than some of the other approaches, yet still promised competitive performance metrics.

For this portion of the assignment, I focused on implementing the underlying data structure that ensures total ordering of elements in the document when inserting and deleting elements. In Logoot, this total ordering helps ensure that peers can modify the document with arbitrary latency and all peers will reach eventual consistency of their own local copies. Each element in a Logoot CRDT has a positional index that is fixed regardless of surrounding insertions and deletions. When an insertion of an element I is done between any two elements A, B the resulting position of I is $pos(A) < pos(I) < pos(B)$. Similarly when deleting elements, the surrounding elements are undisturbed. The positional indexes are implemented with a list of positions representing digits in a arbitrary base so that all positions can be considered a real number that exists between between 0 and $maxdigit$. I chose to use base $2^8 = 256$ as larger numbers can make testing difficult as with a larger base, you need exponentially more elements to create a the same maximum length of the digits list. Using a higher base decreases the length of each positions digits on average, but increases the minimum memory of each element, in my final implementation I may experiment with larger bases but for the testing phase a lower base is ideal.

I chose to do my implementation in Go as I am trying to learn and improve my skills in that language. As of submission I have completed the basic functionality of the CRDT allowing for arbitrary insertions and deletions, I have not started working on the peer to peer components of this project and currently the portions of the Logoot algorithms that ensure eventual consistency between peers have not been fully developed.

3 Testing

As mentioned in the previous section, my implementation thus far has focused on the basic portions of the data-structure and hasn't specifically targeted the concurrency related portions of the problem, as such my testing has also been focused on the data-structure itself. The two main operations that you can perform on this data-structure are insert and delete. I also implemented a number of auxiliary operations that are needed for fully using these operations such as moving throughout the document. The single most important function is $GeneratePositionBetween()$ which generates a valid position between any two other positions. This function is used to compute new locations for insertions and contains many edge cases and difficult operations.

In order to test the functionality I implemented, I created two fuzz/unit testing functions, one of which tests the ability to insert new elements, and the other tests the ability to insert and delete elements. Both of these functions are currently set up to each run 1000 random operations and each are run 1000 times with a different random seed each time. As no failures are arising, I am cautiously optimistic that my implementation is correct thus far. Due to time constraints, parts of my code are not ideal in terms of code quality, so it is currently difficult to logically verify my results. Both of the tests I run are considered a success by maintaining a separate copy of what we would expect the document to look like after each operation, if a discrepancy between the Logoot document and the separate copy are found, an error will be reported. Keeping a separate copy is a feasible strategy for testing these features because we are not testing any concurrency related features in a simulated peer to peer network. Because of this we can fairly trivially implement a data structure which the keeps relative ordering consistent (for testing) with insertion and deletion operations, but fails to keep absolute positions as Logoot does. Once testing moves to the more concurrency related features, different strategies will need to be researched to ensure that the operations behave as expected.

4 Further Work

As mentioned previously, as of now I have solely focused on the single user case of this problem. I intend on starting work on the peer to peer network case of Logoot shortly as that will be the fundamental architecture of my project. After completing my implementation of Logoot I will start work on the server and application side of my project with the goal of having a working prototype which allows for a network of peers to collaborate on a single document without a central server in real or near real-time.

Additionally I hope to put more work into the basic algorithms of my implementation to logically verify their correctness. I also would like to experiment for formal verification methods to ensure the correctness of these important components.