CS1632 – DELIVERABLE 4: Performance Testing

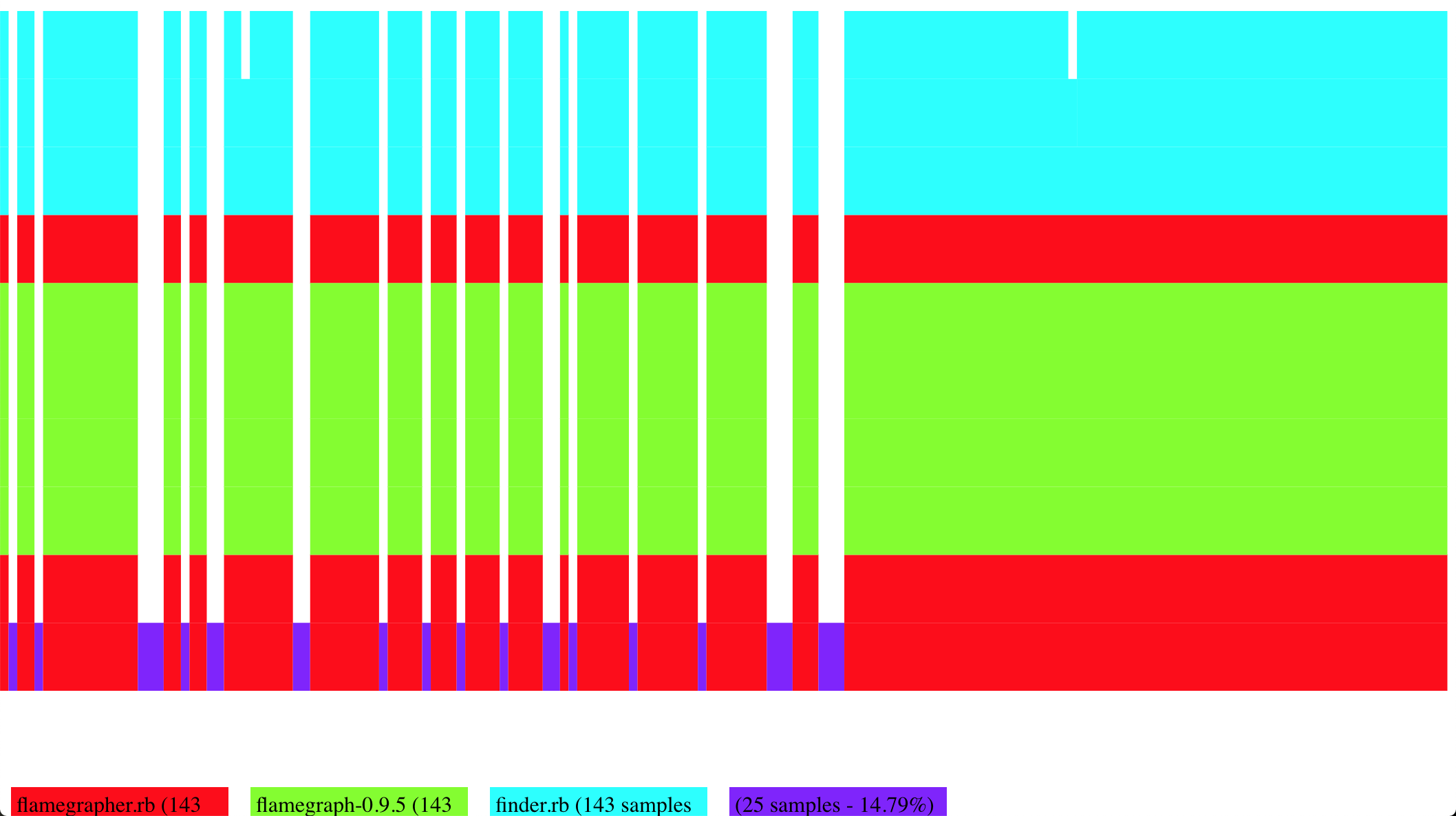
Jessica Chan

<https://github.com/jmc282/D4>

**Summary**

I found the most challenging aspect of this deliverable to be creating the program, specifically finding all of the possible paths between vertices. A significant amount of graph theory was required to be able to make the program work as expected. In the end, it didn’t work entirely. The program is not able to find all possible paths, but does find most paths on smaller graphs.

I created a separate file flamegrapher.rb which generates a flamegraph for the word finder program, given a file name argument. The flamegraph had different appearances for each file it was generated on. The larger graphs had a spike in the methods used to find all possible paths, but for the most part they were all generally flat. In my program I choose to store an array representation of the word list to check my permutations against. I could see from the flamegraph for small\_graph.txt that a large amount of time was spent in this array, called wordlist. This is where I choose to make optimizations. The optimization I made was simple, and caused significant improvement in running time. I have two arrays of strings, permutations and wordlist. permutations contained all permutations of paths, which is an array of string representations of all possible paths in the graph. Previously, for each element in permutations, I checked if wordlist.include? element. If it did, I appended the word to a new list called realwords. I found out that the .include? method for sets is a lot faster than the .include? method for arrays.before doing the same checks, I converted the wordlist to a set, so that wordlist.include? is now using the set .include? method. The change caused major improvement over the initial version in real time, user time, and on large graphs, system time.



Flamegraph for small\_graph.txt