## CS 1632 - DELIVERABLE 4: Performance Testing

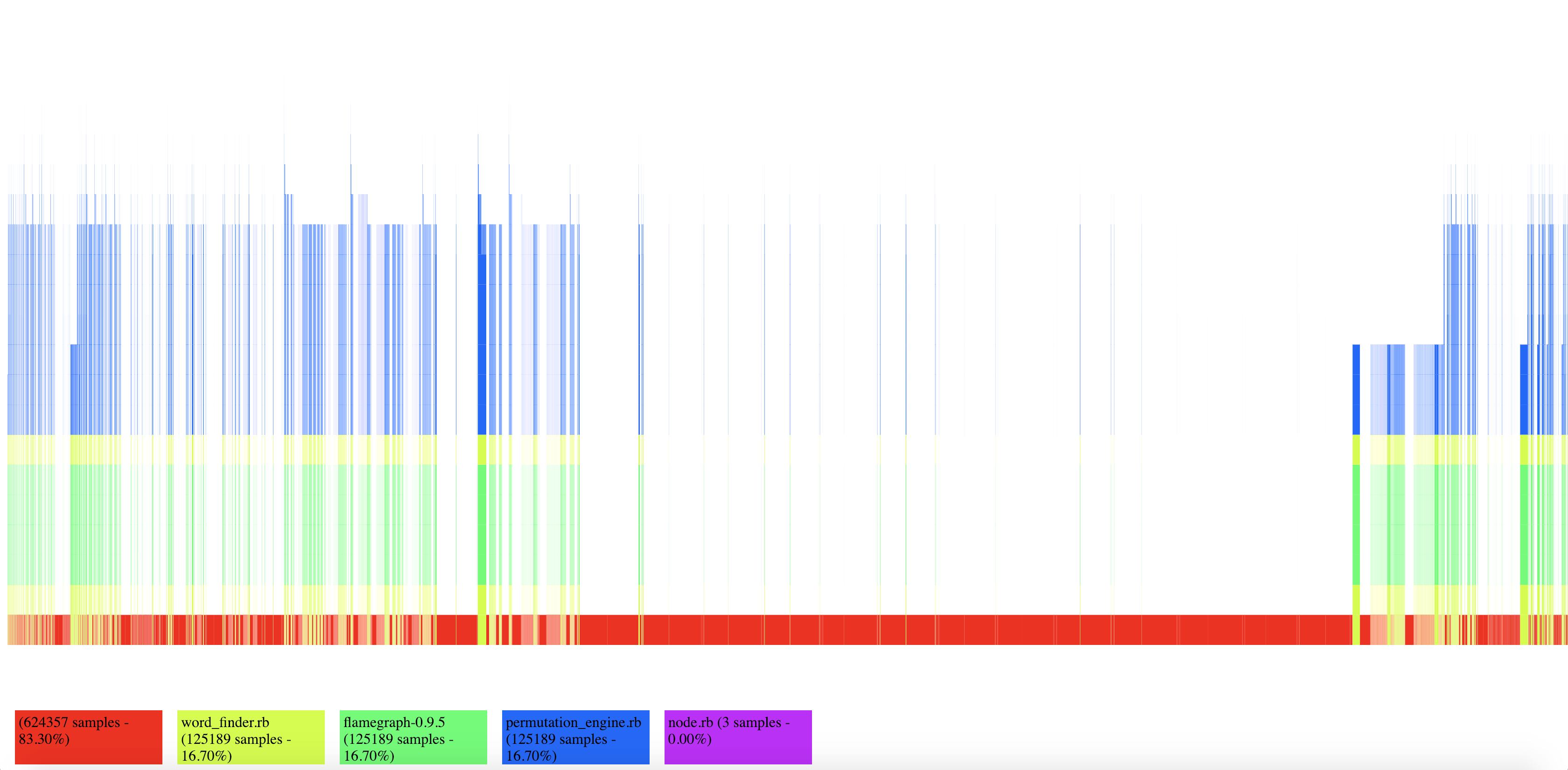
Nathan Harris (@Nathan-Harris) & Adam Richman (@adamrichman1)

<https://github.com/adamrichman1/D4>

**SUMMARY:**

In general, the actual coding of the functionality, and the following speed improvement proved to be quite difficult. Initially, the speed up appeared to be impossible based upon the flamegraph, since the only methods which took up most of the time on graphs other than the ultra big were all single iterations used to populate the dictionary. Once running the flamegraph on the ultra big graph, room for improvements became quite clear. The vast majority of the time according to the flame graph was actually being spent by the garbage collector. So, seeing as the machine it was being tested upon had RAM to spare, the first solution that came to mind was to disable Ruby’s garbage collector, which was consuming so much CPU time. This resulted in a crash of the entire machine. Instead, we attempted to improve the memory usage of the program as a whole, as we figured that less memory use would mandate less garbage collection, and therefore, a faster execution. We attempted to achieve this by taking advantage of Ruby’s lazy enumerators, based on the sheer number of enumerators we were initializing. Additionally, we fine-tuned the malloc settings of Ruby’s garbage collector to compensate for higher-performance systems. In regards to edge cases, the majority we had to deal with were regarding words that were too small or disjoint graphs, which we dealt with by having limitations on the minimum size of the words.

**FLAME GRAPH:**

****

**FINAL TIMES:**

* Small Graph:
  1. 0m0.362s
  2. 0m0.210s
  3. 0m0.197s
     + Average time: 0m0.256s
* Medium Graph:
  1. 0m0.336s
  2. 0m0.317s
  3. 0m0.292s
     + Average time: 0m0.315s
* Big Graph:
  1. 0m0.285s
  2. 0m0.204s
  3. 0m0.214s
     + Average time: 0m0.234s
* Ultra Big Graph:
  1. 6m30.295s
  2. 6m45.210s
  3. 7m0.783s
     + Average time: 6m45.256s