Arnav Shirodkar

CMSC 201

[as9086@bard.edu](mailto:as9086@bard.edu)

23/4/20

Assignment: Build a frequency counter and test Zipf’s law using test files. Also test the efficieny of a Sequential Search Symbol Table with a Self-Organizing Symbol Table that puts the most recently accessed entry at the front of the queue.

Collaboration Statement: I worked on this myself apart from the implementation of mergesort for linked lists. For that I went online to coding simplified and watched some videos to break down how the various functions to accomplish mergesort on a linked list would work before implementing it into my own code for both the SequentialsearchST and the Selforganizing ST.

**Zipf’s law**

To test Zipf’s law I used two big texts: Herman Meville’s “Moby Dick” (moby.txt.) and Franz Kafka’s “Metamorphosis” (kafka.txt.). The graphs for each book are on the subsequent pages. (I took this graphs as screenshots from google sheets)

Based off each log-log plot, we can largely conclude that Zipf’s law generally holds for these texts, as the resultant lines on the log-log plot are mostly straight. As we dip into the region of words with a lower rank, the line seems to curve downward so it is possibe that Zipf’s law holds more strongly for words of a higher rank where the discrepancy in their frequency is easy to distinguish.

**SequentialSearchST vs SelfOrganizingST**

To measure efficiency between the two different symbol table implementations, I created a stopwatch at the beginning of the main function that printed the time elapsed after the linked list’s construction was complete. I did not factor in the time taken for mergesort as that is independent from the efficiency of each of these implementations.

|  |  |  |
| --- | --- | --- |
|  | SequentialSearchST/s | SelfOrganizingST/s |
| Kafka.txt | 4.245 | 1.898 |
| Moby.txt | 266.485 | 69.15 |

From one glance, it is apparent that the heuristic employed in the in the Self-Organizing table makes it far faster than the SequentialSearch Implementation. This is expected since freqeunt words are far more likely to appear at the very front of the list than at the back, reducing the total amount of “distance” traversed over the linked list.



