Ryan Wallace CS51 April 1, 2016

Assignment 5: Moogle Writeup

After implementing the crawler and the DictSet module, it was possible to test the relative performance of implementations of sets as lists and dictionaries.

The space complexity of the implementations is one factor to consider. Generally, it appears that the list implementation is more space efficient, as the dictionary implementation redundantly stores each link as both the key and value in the dictionary. Thus, the dictionary implementation requires at least twice as much space as the dictionary implementation. Nevertheless, the asymptotic space complexity is the same for both implementations as the space required by the dictionary is approximately a constant factor of two greater than that needed by the list implementation.

The other factor to consider is the difference in time complexity of the two implementations. The two relevant time performance metrics are time to crawl a corpus of pages, and the time to query a search term. In order to measure these items, two functions were written in moogle.ml, time_query and time_crawls. Because these functions are time consuming and print clutter to standard out, they should not necessarily be called every time the moogle program is run. Thus, the flag “time” as a boolean variable was included at the top of moogle.ml, which when set to true, runs the timing functions, and prints the timing results to

standard out, and when set to false, does not.

The function time_query queries six commonly found words, (a, in, the, and, moo, set) using the do_query function. Since it is more scalable to not need to manually average the time required by each of these queries, the do_query function was modified to return a tuple of the html string it returned originally, but also a float containing the time taken by the query. Using the list set implementation, average querying time for the simple-html corpus over 7 pages was 0.000005 seconds, 0.000008 seconds for the html corpus over 20 pages, and 0.000963 seconds for the wiki corpus over 42 pages. Using the dict set implementation, average querying time for the simple-html corpus over 7 pages was 0.000004 seconds seconds, 0.000004 seconds for the html corpus over 20 pages, and 0.001007 seconds for the wiki corpus over 42 pages. Clearly, as the corpus size increases, the time savings of the dictionary implementation become clear. These time savings are due to the fact that querying is done by calling D.lookup, which in the case of the list implementation is linear time in the size of the index, but log time in the case of the dictionary (as it is implemented as a BST.)

The function time_crawls crawls makes six calls to crawl with each of the three corpora and varying limits on the number of pages to crawl, and prints the time used to crawl each set. An instance of the data outputted using the list implementation is:

Crawling 7 pages from simple-html/index.html took 0.000954 seconds

Crawling 7 pages from html/index.html took 0.126255 seconds

Crawling 7 pages from wiki/Teenage_Mutant_Ninja_Turtles took 1.111464 seconds

Crawling 20 pages from html/index.html took 0.412838 seconds

Crawling 20 pages from wiki/Teenage_Mutant_Ninja_Turtles took 7.248927 seconds

Crawling 224 pages from wiki/Teenage_Mutant_Ninja_Turtles took 257.479494 seconds

And an instance of the data outputted using the dictionary implementation is:

Crawling 7 pages from simple-html/index.html took 0.000937 seconds

Crawling 7 pages from html/index.html took 0.095832 seconds

Crawling 7 pages from wiki/Teenage_Mutant_Ninja_Turtles took 0.716036 seconds

Crawling 20 pages from html/index.html took 0.295958 seconds

Crawling 20 pages from wiki/Teenage_Mutant_Ninja_Turtles took 2.085174 seconds

Crawling 224 pages from wiki/Teenage_Mutant_Ninja_Turtles took

51.008401 seconds

As with for querying, we see that the dictionary implementation is marginally faster for small corpora, but substantially and meaningfully faster for large corpora (interestingly over five times faster for the full wiki corpus). This again is due to the underlying implementation of dictionaries as binary search trees, which allows for faster, log time lookup and insertion than the linear time taken by the list approach.