20020130

CS-202

Assignment 4

Words chosen: 10,000 from secret1.txt (first 10,000 words)

Time without cache: 447 seconds

Time with cache: 334 seconds

**How the code ran:**

The part implemented without the cache ran slow. This was pretty obvious because of the fact that we performed a linear search from the start of the dictionary every time we read a word from the secret word file. In the worst case it would take O(N) time. The code however, was much much smaller than the one which ran the cache.

When I searched the dictionary using the cache, it ran much faster. There was an observed difference of more than a 100 seconds on the first 10,000 words. This was expected because words were repeated throughout the file. Whenever we read a word from secret1.txt, we checked if it was in the cache. Since finding a word in a hash-table is O(1), we saved time whenever the word was present in the cache. In case it wasn’t, then it took more time than the without cache version as we had to not only search for the word but also insert it into the cache (also delete a word from the cache). However, the LFU methodology ensured that we had the most repeated words inside the cache so it saved time. If the test had been run on say 100,000 words, then the cache would have been more refined and we’d observe a higher difference b/w the two implementations.

**Type of hashing chosen: Linear Probing**

There were multiple reasons for selecting linear probing. Firstly, it was better than the other two versions of hashing which we implemented in the assignment. I did not choose separate chaining because we only had a table of size 1000 and way more words than that in the dictionary. So chaining would have resulted in no deletions being made to the cache because every new word would simply be inserted to the head of the linked list at the index generated by the compression function. So over time, our hash table would become huge and would require a lot of memory which would destroy the whole idea of a cache which has fixed storage. Not to mention the fact that searching it would take longer as well.

I did not choose double-hash either because for most of the time, our cache is almost full and I suppose double-hashing would take longer to find an empty slot (if there is one) because the odds of landing on an empty slot would be pretty slim. Also, it would require extra computation (hashing twice) every time we have to remove/insert a word from the cache.

Therefore, linear probing was the better of the three. At max, we would have to go through the entire array (which would happen in the worst case i.e the empty slot is the slot behind the slot which was generated by the compression function). Also, a single hash function requires less computation.