Applicant: Matt Fergoda
Course: Harvard's CS50: Introduction to Computer Science via EdX
Dates taken: May - Sept. 2019
Link to full assignment description:
https://docs.cs50.net/2019/x/psets/4/speller/hashtable/speller.html

Assignment description
We were asked to implement a program that spell checks a text. The program must load a text file containing correctly spelled words into a hash table (the "dictionary"), then check whether words in a separate text are in the dictionary regardless of capitalization. It must also print to the terminal all words in the text that are not in the dictionary. We were asked to make the fastest spell checker we could according to wall-clock time.

We were specifically asked to implement `dictionary.c`, which contains the functions `hash`, `load`, `check`, `size`, and `unload`. This write-up is in reference to `dictionary.c` which reflects my original work. `speller.c`, `speller.o`, `dictionary.h`, `dictionary.o`, and the `Makefile` were all included as part of the assignment's distribution code and do not reflect my original work.

Implementation and methodology
I chose to implement the dictionary as a hash table consisting of an array of linked lists. In theory, a well-designed hash table has constant lookup time, which is especially desirable when storing a large number of entries. To approach constant lookup time, as a rule of thumb it is important to keep the load factor (number of entries / number of buckets) less than 1.

First, I implemented my own hash function that summed the ascii values of a word and divided them into around 10,000 bins. Though this led to a 3s overall program runtime, the load factor was about 14 indicating speed could be improved (the dictionary contains 143,091 words). I decided to investigate even more efficient hash functions and settled on a polynomial rolling hash (as described here: https://cp-algorithms.com/string/string-hashing.html, last retrieved 2/12/20):

$$H(s) = \sum_{i=0}^{n-1} s[i]\, p^i \,\%\, m$$

where s is a string, n is the length of the string, p is a prime number roughly equal to the characters in the input alphabet, and m is the number of bins in the hash table. This hash function has the advantage of spreading out values across bins to minimize collisions without being too computationally costly.

In order to adequately spread values across the hash table, the source suggests that m be a large prime number. I chose m = 524,287 (discovered by Piertro Cataldi: https://en.wikipedia.org/wiki/Largest_known_prime_number, last retrieved 2/12/20) which yields a load factor of 0.27. Although such a low load factor indicates wasted memory, the next smallest prime (131,071) would lead to a load factor greater than 1 and speed would suffer. I chose to prioritize speed. With this implementation, the program performs the spell check in about 0.75s on the Holmes text, taking less than one third of the lookup time of the previous hash function.