Data Structure and Algorithms Coursework 1

F28SD

23/02/2021

Varun Senthil Kumar

H00332328

BSc Computer Science(Artificial Intelligence)(Hons)

Dubai Campus

Program Design

This spell checker program implements Hash Table and Linked List algorithms to provide suggestions to misspelled words. The main objective of this coursework is to compare a linked list based and hash table-based set.

The Hash Table algorithm design is briefly explained below.

The Hash Table class contains two constructors. The first constructor is a basic constructor while the second is a parametric constructor, that is, it takes in an argument. The difference between the two constructors is that the basic constructor sets the maxLoad variable to 0.5f while the parametric constructor sets the maxLoad variable as the value of the argument passed to it.

public int giveHashCode(String s)

This function returns the hash code of a given string. I have implemented this function using polynomial accumulation. The design of polynomial accumulation is as below:

1. Declare a polynomial accumulator *'a'* and a variable *'pos'*, both initialized to 0 and 1 respectively.
2. A *'for'* loop is initialized to go till the length of the word.
3. Every time the loop goes through, *'a'* is added to itself and the integer value of the character at index *'i'*. *'pos'* is multiplied by 33.
4. When the loop ends, the function returns the absolute value of *'a'*.

public void insword(String word) throws SpellCheckException

This function adds the given string to the HashTable. For doing so, I have made use of double hashing.

To add a word, it first calculates the hashCode and the double hash value of the word.

The double hash is implemented in base of 5. It adds the word to an empty cell or a cell with a placeholder. An exception is raised if the word already exists.

Similar implementation is used in rmword() and wordExists().

private void resize()

This function increases the array size to the next prime number larger than the current array by two times. It is implemented by creating an iterator with every word which is then rehashed and stored in the new array.

private float getLoadFactor()

This function calculates and returns the load factor. The load factor is equal to the no. of words in the array divided by the size of the array.

The result is type casted to float.

Public float getAverageProbes()

This function calculates and returns the average no. of probes, which is equal to the total no. of probes divided by the total no. of operations done.

The result is type casted to float.

The linked list algorithm design is not as complex as the Hash Table implementation.

For adding a word into the linked list, a *'for'* loop first goes through the linked list to check if the word already exists in the linked list. An exception is thrown if the word is already present in the linked list. Otherwise, the word is added to the linked list.

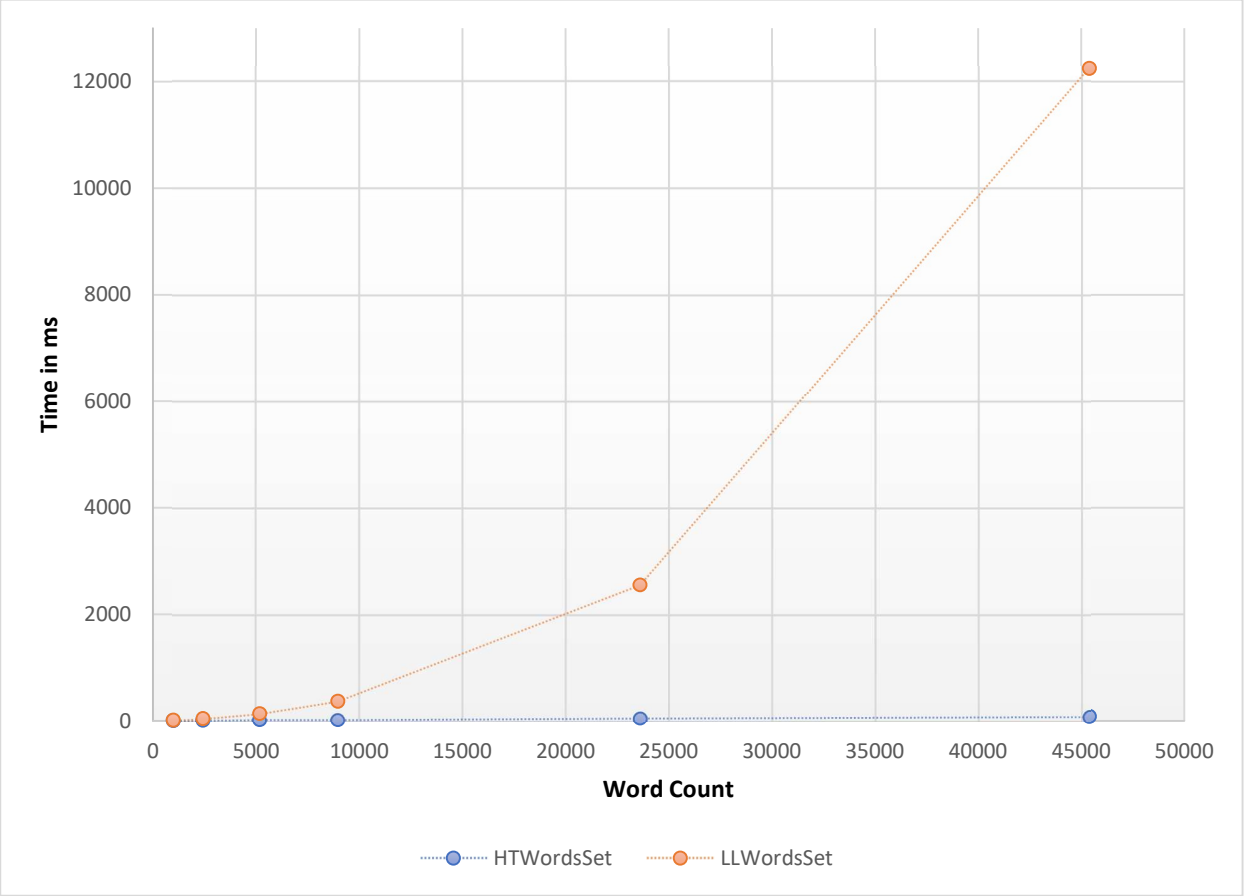
For removal of a word from the linked list, a *'for'* loop goes through the linked list to find the word. Then the word is removed. An exception is thrown if the word is not found.

Both the Linked List and Hash Table class contain an Iterator function, which is used to iterate through the data structure.

The main function is currently implemented using HashTable. The Linked List command is commented out.

There are currently no known limitations for this program.

Hash Table vs LinkedList



The chart shows that the WordCount vs Time chart for the Hash Table implementation is a linear equation, meaning that the time is increasing in a linear fashion.

For the Linked List implementation, the graph shown is similar to the Hash Table at the beginning, where the word count is comparatively less than later files. As the word count increases, there is an exceptional increase in run time. This resembles a quadratic function.