Project 4 Report: Data Structures and Algorithms

Document Author(s): Louis Warner
Date: 7-29-2015

Overview

This report will discuss the hash table dictionary implementation used in Project 4 for the spelling checker program. The project was designed to store all of the words from a given dictionary text file in a hash table and then used to check a subsequent text file for misspelled words. The program generates an output file that contains any misspelled words along with a statistical analysis of how the hash table implementation performed for the given text files.

Table Size

In choosing the table size for this implementation, I based my decision off of the number of words expected to be found in the dictionary file (25,144 words). The first prime number after this is 25,147 so this was the initial size tried for the project. I found that by doubling the size and moving to the next prime number, I was able to reduce the amount of collisions by a little under 50%. Similarly by doubling the size again, I was able to reduce the amount of collisions by a few more percentage points, but not enough to justify such a large amount of memory being allocated to the project. I settled on 50,291 for the size of the table, which is the next prime number after 25,147 * 2.

The Hash Function

The hash function used was :

```
public int hash(String key, int tableSize){
	int hashVal = 0;
	for( int i = 0; i < key.length( ); i++ ){
		hashVal = 37 * hashVal + key.charAt( i );
	}
	hashVal %= tableSize;
	if( hashVal < 0 ){
		hashVal += tableSize;	}
	return hashVal;
}
```

I noticed a similar function being used in many hash tables of similar size in many hash table projects across the web, and so I modified my hash function to implement a version of Horner's method for evaluating polynomials. Starting from 0, it multiples the current hash value by 37 and adds the value of a character in the string. It does this until the string has no more characters, and then does a modulo operation based on the table size. If at this point, the hash value is negative, the method adds the table size to the value to make it positive.

Collision Resolution

The collision resolution method used in the hash table was a separate chaining method. Each hash entry was given the capability of becoming a part of a linked list, so any conflicts caused when adding to the table led to the entry becoming linked to the entry already residing in the hash table. This method left some "memory gaps" in the hash table array, but resulted in speedy insert and lookup operations for the file sizes used in this project.