[Spell Checker]

Proposal

**<Samuel Jessee>**

**CSC316: Data Structures for Computer Scientists**

**<sijessee@ncsu.edu>**

**North Carolina State University**

**Department of Computer Science**

**<April 11, 2016>**

Executive Summary

This project covers the creation process of a spell-checking program. In this project, a hash table will be implemented to make the process of searching a large dictionary much faster. This report covers the problem in more detail, as well as expanding on the proposed solution. It also explains the algorithms to be used in the program, shows a class diagram, and provides a set of black box tests. Lastly, a task plan provides a timeframe for the tasks of the project.

Table of Contents

[Problem Statement 1](#_Toc440806182)

[Project goals & Benefits 2](#_Toc440806183)

[Algorithm Design 3](#_Toc440806184)

[Proposed algorithm 3](#_Toc440806185)

[Algorithm Design Rationale 3](#_Toc440806186)

[Algorithm Analysis 4](#_Toc440806187)

[Software Design 5](#_Toc440806188)

[Design Pattern(s) 5](#_Toc440806189)

[Black Box Test Plan 6](#_Toc440806190)

[Task Plan 9](#_Toc440806191)

# Problem Statement

The problem calls for a program that can spell-check a document using a dictionary of 25,144 of the most common English words. The program needs to be capable of efficiently reading and storing a dictionary file, and reading through the provided document word by word, flagging each word not found in the dictionary as a possible misspelling. This program will allow users to easily find spelling errors in large documents without having to read the entire document word for word.

# Project goals & Benefits

The goal of this project is to implement a program that can efficiently spell-check documents using large dictionaries of words. To meet this goal, the program will implement a hash table. The point of using a hash table is to make it much more efficient to search for words in a large dictionary. With a dictionary of over 25,000 words, a binary search tree would take a lot of time to search through, and since a single word may call for multiple search operations, using a binary search tree to implement this program would be impractical. Therefore, the goal of this project is to successfully implement a hash table system that will allow search operations to be carried out much faster, regardless of the size of dictionary used. This system will not only benefit spell-checking, but also any other program that references a very large database of information, by allowing those programs to search for information faster than a binary search tree.

# Algorithm Design

## Proposed algorithm

--------------------------------

Algorithm hash(String w, int p)

Input a String to get hash code for, and an exponent used to get a constant.

Output the hash code for the String.

f <- 0

r <- 2^(p)

for each character c in w, do

add (c \* r) to f

return f

--------------------------------

Algorithm compress(int f, int m)

Input hash code to compress, and size of hash table

Output compressed code

gr <- inverse of golden ratio constant

h <- floor( m {f \* gr } )

return h

--------------------------------

//This algorithm uses my LinkedList class, and its addToFront method

//Hash table is implemented as an ArrayList

Algorithm resolve(String w, int h)

Input the word and its compressed hash code

ht <- the hash table containing linked lists at each hash code

if ht[h] is null then

ht[h] <- new LinkedList starting with w

else ht[h].addToFront(w)

--------------------------------

## Algorithm Design Rationale

The proposed algorithms use a polynomial hash code process to create hash codes. The reason this was chosen is because polynomial hash codes take character order into account when being created, so two keys won't have the same hash code just because they contain the same characters. This will minimize collisions in a dictionary of many words that will have the same letters. For compression, the golden ratio method is used, since it spreads out keys uniformly and avoids many collisions. To deal with the remaining collisions, the algorithms use a linked list structure at each position in the hash table. If a word's hash code puts it in a spot that is already taken, the word will be added to the front of the Linked List. This method will be simple to implement and less likely to cause problems during the debugging process. Using the addToFront method ensures that the process of creating the dictionary goes as fast as possible, since addToFront is the fastest adding method for linked lists. Because adding to the front of a linked list is fast, and search operations will have to read through lists one element at a time anyway, I believe linked lists are the best list structure to use for this scenario.

## Algorithm Analysis

The hash algorithm has a runtime complexity of 3 + n, which is O(n), where n is the length of the word. The runtime of the hash algorithm will be directly proportional to the length of the word being hashed.

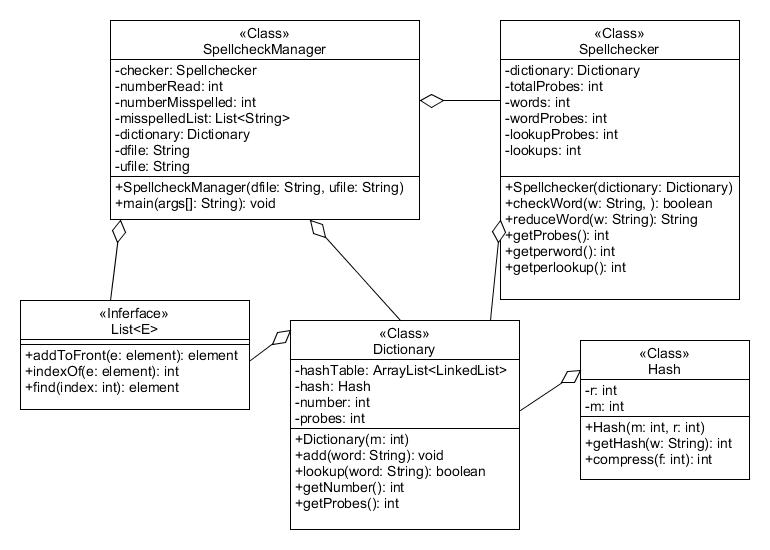
The compress algorithm has a constant runtime O(1) since it only performs mathematical operations, and does not have to loop.

The resolve algorithm also has a constant runtime O(1). The algorithm calls the addToFront method of Linked List, which also has a constant runtime, since it does not have to iterate through the list to add an element to the front.

These runtimes were all pretty obvious just by looking at the algorithms.

# Software Design

------------------------------------



------------------------------------

## Design Pattern(s)

This design utilizes the recursion design pattern. Recursion is the pattern of making a method call itself to solve a smaller version of a problem. Recursion makes it easy to think about and solve large problems, such as having to spell check a word multiple times with small variations in spelling. This is why recursion is used in Spellchecker, specifically in the checkWord method. Each time checkWord calls reduceWord and gets a different word, it calls itself again with the new word. This process makes it easy to repeat the process of checking and reducing an arbitrary number of times for a word.

# Black Box Test Plan

This black box test plan uses multiple files, testfile-\*.txt, and testdictionary-\*txt. In each test case, use the files named in the description, entered in the order given.

|  |  |  |  |
| --- | --- | --- | --- |
| Test ID | Description | Expected Results | Actual Results |
| No errors | Use files testfile-1.txt and testdictionary-1.txt | Misspelled words:  Words in dictionary:1  Words in text:1  Misspelled words:0  Total probes:1  Average probes per word:1  Average probes per lookup:1 |  |
| Capitalized | Use testfile-2.txt and testdictionary-1.txt | Misspelled words:  Words in dictionary:1  Words in text:1  Misspelled words:0  Total probes:2  Average probes per word:2  Average probes per lookup:1 |  |
| Multiple changes needed | Use testfile-3.txt and testdictionary-1.txt | Misspelled words:  Words in dictionary:1  Words in text:1  Misspelled words:0  Total probes:4  Average probes per word:4  Average probes per lookup:1 |  |
| Misspelled word with multiple changes | Use testfile-4.txt and testdictionary-1.txt | Misspelled words: Acbedededs's  Words in dictionary:1  Words in text:1  Misspelled words:1  Total probes:7  Average probes per word:7  Average probes per lookup:1 |  |
| Multiple words | Use testfile-5.txt and testdictionary-2.txt | Misspelled words:Baaker's, cak  Words in dictionary:2  Words in text:6  Misspelled words:2  Total probes:12  Average probes per word:2  Average probes per lookup:2 |  |

# Task Plan

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Task Description | Owner | Planned  Start Date | Planned  End Date | Status |
| Write proposal and algorithm | Samuel | 4/8 | 4/10 | Completed 4/11 |
| Write out classes and javadoc | Samuel | 4/12 | 4/13 |  |
| Implement methods | Samuel | 4/13 | 4/16 |  |
| Write test cases and debug | Samuel | 4/17 | 4/19 |  |
| Final report | Samuel | 4/20 | 4/21 |  |

**Team Contact Information**

Samuel Jessee, sijessee@ncsu.edu, 919-610-1546