Analyzing HashTable Efficiencies with Shakespeare Concordance Applications

HashTable Project

Elven Isaac Shum

Jan 13 2020

1 Overview

This project's goal is to analyze the "size vs. efficiency" trade off in Hash Tables; it consists of 3 major components:

HashTable

This HashTable utilizes the 1000WordDictionary as it's data set. It consists of two arrays which store key-value pairs, respectively. Given the data set, the "value" simply defaults to a placeholder number, as the interesting part is in the distribution of the Values (ie the words).

ConcordanceHashTable

This ConcordanceHashTable utilizes Shakespeare's Sonnets as it's data set. It consists of two arrays which store key-value pairs; one stores Strings, the other is a LinkedList of "Occurance" objects (essentially a Pair).

ConcordanceHashTable App: "WordSearcher"

This App essentially packages the data from the ConcordanceHashTable and makes an Application. This can be used for searching particular words.

2 Data and Analysis

2.1 Analysis Metrics and Methods

Average Number of Collisions This metric simply calculates the average number of collisions, given a DataSet and a HashTable size. An interesting correlation can be shown via HashTable size and the collision number.

"Goodness" Metric The "Goodness" Metric is essentially a measurement of Optimal Distribution. For HashTables, an Optimal distribution means the *key-value pairs are uniformly spaced out throughout the HashTable.* This Metric utilized a Chi-Squared test, where the "expected" is the optimal distribution, to yeild the following equation.

$$\frac{\sum_{i=1}^{m-1} \frac{(b_i)(b_i+1)}{2}}{\frac{n}{m}(n+2m-1)}$$

where

n: number of keys

m: size of the HashTable

b_i: number of items in HashTable Location i

For reference, to be considered an Optimal Uniform Distribution, the outputted value from that equation should be from 0.95-1.05

Size-Efficiency TradeOff

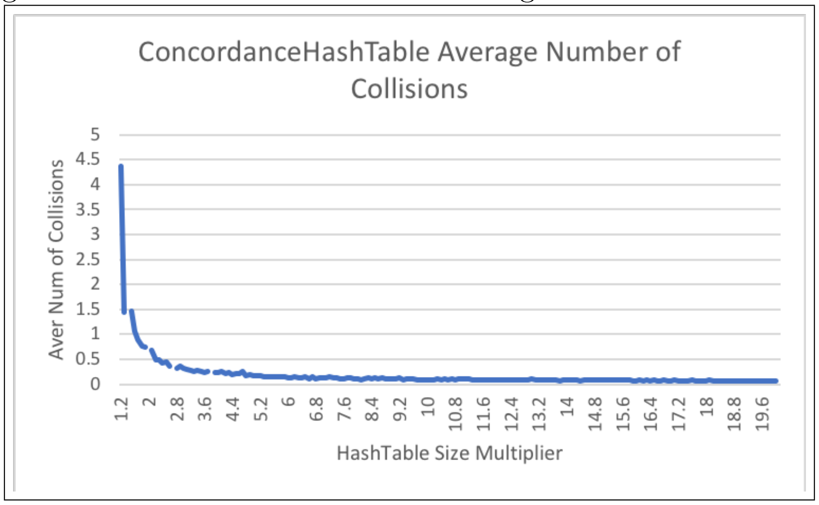
The optimal, yet simple, Hash Function I found was a take on the classic "add the ASCII values and divide by the HashTable size." Because I noticed values would clump up, since the ASCII values are quite close, I decided to square each ASCII value, then sum, then divide. This exaggerated the uniqueness of specific characters and resulted–as you will see–in a near optimal distribution.

We will demonstrate the "efficiency" based on the number of collisions–meaning, when inserting a key, how many times the Hash collide with already placed data and need to increment. If the collisions is 0, then the set is likely uniformely distributed. On the otherhand, for a hashfunction like "return 17," one would receive many collisions.

For Figures 1 and 2, we show both HashTable and ConcordanceHashTable's Average Number of collisions. The values on the x axis represent the "HashTable size multiplier." To normalize the data, it's important to base the HashTables size off the estimated number of Keys. So, the plotted is simply the "size multiplier", which is ultimately multiplied by the NumberOfUniqueKeys to achieve the HashTableSize.

$$SizeMultiplier * NumOfUniqueKeys = HashTableSize$$

Figure 1: ConcordanceHashTable Average Number of Collisions



As you can see from Figures 1 and 2, as the size increases the the average number of collisions drops down dramatically–ultimately plateauing quickly at the end. Note that in Figure 2, there's a quick spike downwards at around x=5.2. If one were creating an optimal sizeMultiplier, slightly after then would likely be considered Ideal. For more data, please run the code and select CalculateEfficiencies

Finally, as a test of the HashFunction, we also calculate the Goodness Measurement of the HashTable. Genearlly, the values very quickly approach our target of the optimal 0.95, which indicates to us that our HashFunction is incredibly efficient–especially at SizeMultipliers at 3 or higher.

Figure 2: HashTable Average Number of Collisions

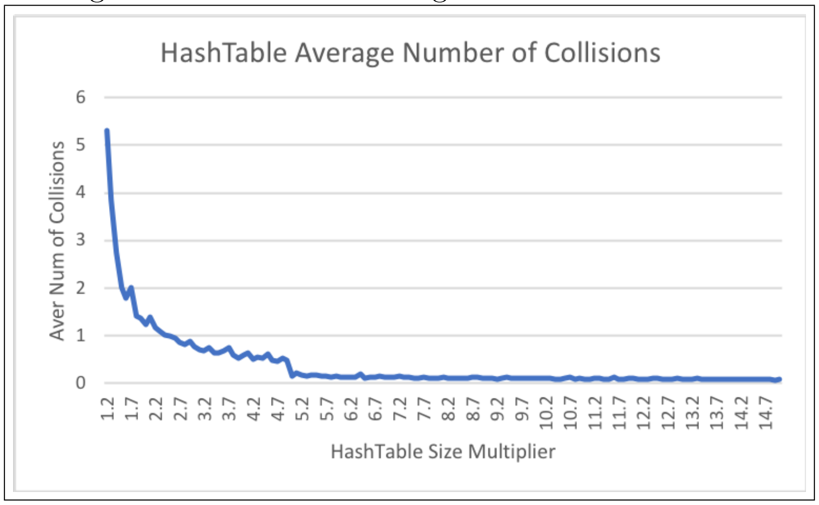
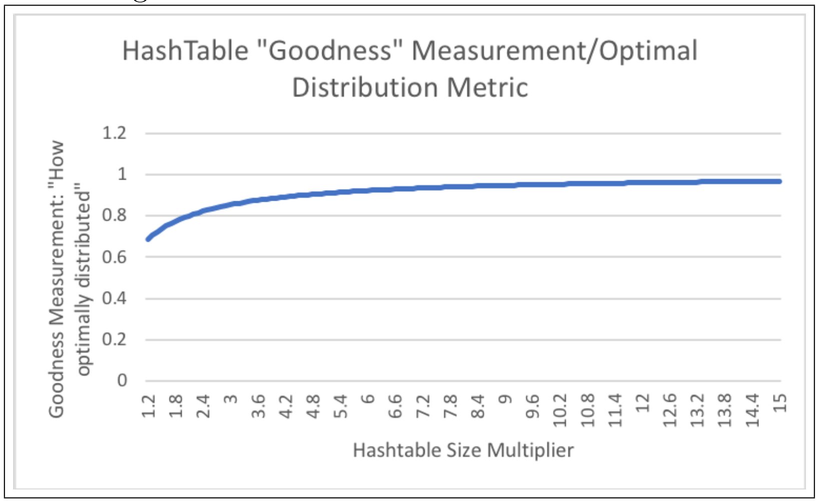


Figure 3: HashTable Goodness measurement


HashTable "Goodness" Measurement/Optimal Distribution Metric
Goodness Measurement: "How optimally distributed"
Hashtable Size Multiplier