Investigating collisions in hash tables

Collected average number of collisions over 10^5 trials:

N = 10^3
Collisions = 1.005170

N = 10^4
Collisions = 1.003880

N = 10^5
Collisions = 0.99482

N = 10^6
Collisions = 1.00210

N = 10^7
Collisions = 1.006390

Judging from the results, the average cost of a search miss in the resulting table is ~1 collisions.

Concordance Analysis

Process:
First, I removed punctuation, converted all words to lower-case, and separated the sonnet text file into an array of WordNodes, each contain a single word string and its location in the text. Then, I calculated the hash for the string of each word Node and used either linear or quadratic probing (Both implementations are included) to resolve collisions and determine the word's index in the concordance. Duplicates of a word which has been previously inserted in the concordance are not added as new entries, instead, the locations of the duplicates will be stored in the array of locations of the pre-existing word.

Experiment 1 - Verifying Accuracy of Concordance

To verify the accuracy of the concordance, I randomly selected words from the sonnet text and searched for it using the contains method. For words with a large number of occurrences, I compared the algorithm derived number of total occurrences with an online unique word calculator (screenshot provided). For words with lesser number of occurrences, I verified by checking the individual logs.

Case 1:

when(106)

```
when appears a total of 106 times in Shakespeare's
    Sonnets.
```

Case 2:

And(490)

```
and appears a total of 490 times in Shakespeare's Sonnets.
```

Case 3:

```
deceive appears at the position 9 on line 10 of Sonnet 4
deceive appears at the position 8 on line 12 of Sonnet 39
deceive appears a total of 2 times in Shakespeare's
    Sonnets.
```

Correct.

Case 4:

```
lose appears a total of 9 times in Shakespeare's Sonnets.
```

lose(9)

Case 5:

to(409)

```
to appears a total of 409 times in Shakespeare's Sonnets.
```

Case 6:

```
equipage appears at the position 7 on line 12 of Sonnet 32
equipage appears a total of 1 times in Shakespeare's
    Sonnets.
```

Correct.

Experiment 2 - Determining Table Size Effect on Collisions
As suggested by an online reference (http://cseweb.ucsd.edu/~kube/cls/100/Lectures/lec16/lec16-8.html), I used a concordance/hashing table size of 1.3*number of words.

Word count: 17536 (verified by Microsoft Word word counter)
Unique word count: 3201

Initially, 22807 was used for table size. 22807 is the nearest prime number to 1.3 * 17536, the word count of the sonnets.

However, I realized that there are only 3201 unique words, so in reality only 3201 array elements will contain a WordNode. 4417 is the nearest prime number to 1.3 * 3201.

I used these two numbers to compare the effect of table size on the hashing resistance towards collisions. I used the djb hash function for both sizes.

Results:

Table Size	**Collisions**
4417	15549
22807	14566
10^6	14341

The results suggest that hashing under larger table sizes results in less collisions, but the effect seems to diminish as size increases further.

Experiment 3: Comparing Effectiveness of Hashing Functions

I implemented four different hash functions: DJB by Professor Dan Bernstein, SDBM, Polynomial Rolling hash, a simple hashing function defined by the sum of the integer values of each character, and Robert Sedgewick's algorithm.

Results:

Hash Function	**Collisions in Table Size of 4417**	**Collisions in Table Size of 22807**
djb	15549	14566
sdbm	15507	14563
poly roll	16673	16326
simple	17259	17259
rs	15551	14545

As expected, the simple hash function had the worst performance because of its simplicity: since the ASCII codes for the lower case characters are bunched together, their sums will also be bunched together in a smaller range, which increases chance of collisions.

The djb, sdbm, and rs hash functions performed much better because of their distinctiveness. The constants that are used as multipliers in the functions are selected not only because they are prime but they have also been determined as optimal for hashing. Thus, these special numbers would generally output more distinct hashes and generate less collisions than algorithms which are more general.