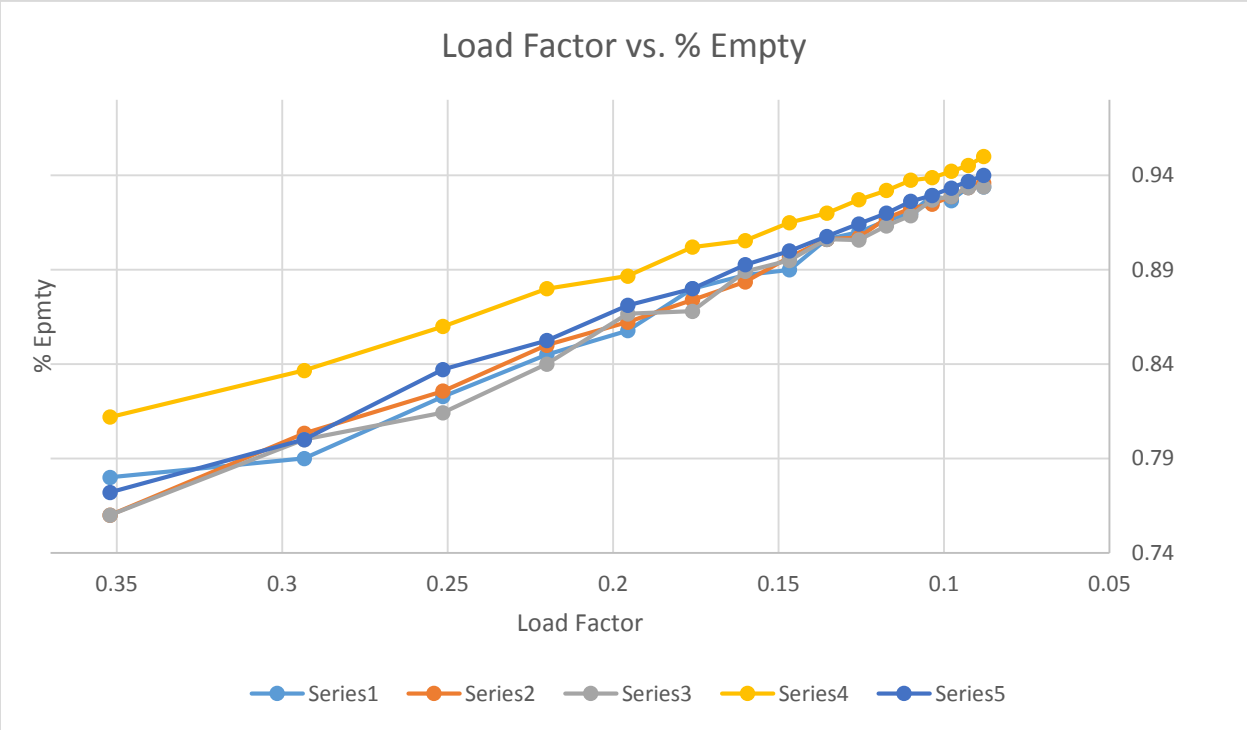
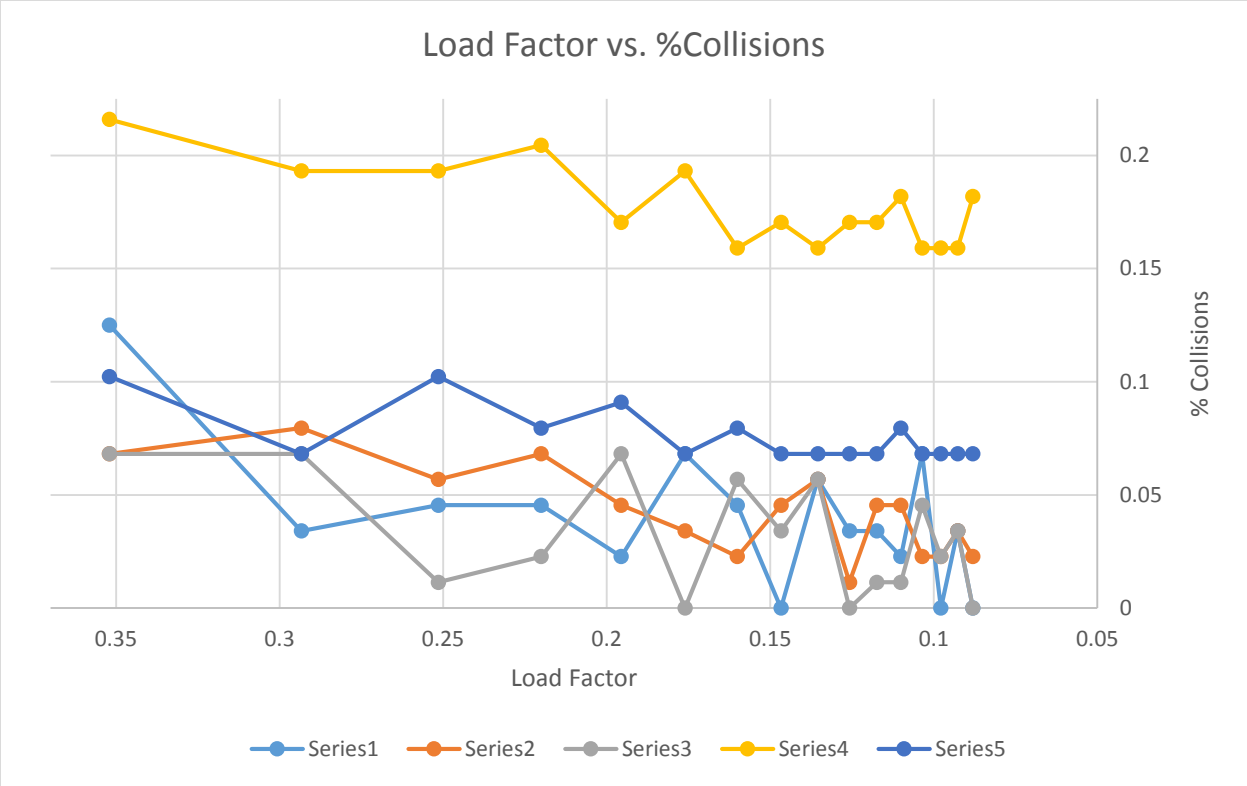
Hypothesis:

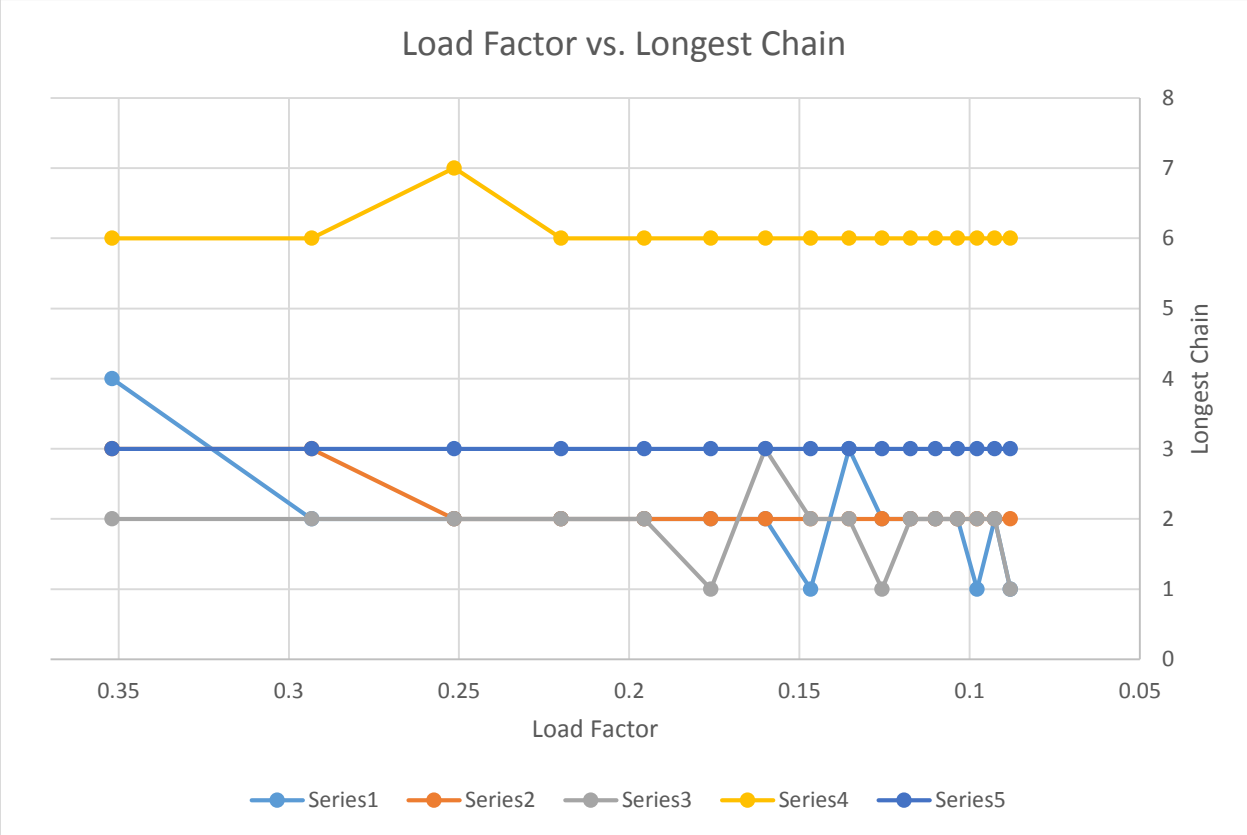
Hash functions that return the same value for different inputs are those that do comparatively little modification to the hash, i.e. functions 1 and 2. These only add the values of the letters, or multiply the hash value by each letter's ASCII value, leading to greater risk of identical hashes for small words and words with similar letters. Hash function 3 introduces an external variable that increases the chances of different hashes as the length of the word increases, meaning shorter words will still have a relatively large chance of being hashed together, but longer words should differentiate well. Hash function 4 starts large, ensuring that smaller words have a lesser chance to be hashed together, and seems to me to be the frontrunner in decreasing %empty and %collisions. Finally, hash 5 seems to be arranged using exponents to attempt to randomly distribute the hashes.

In summary, functions 3, 4 and 5 seem to be good, and functions 1 and 2 seem to be worse.

Graphs:




Load Factor vs. %Collisions
0.2
0.15
0.1
0.05
0
% Collisions
0.35
0.3
0.25
0.2
0.15
0.1
0.05
Load Factor
Series1
Series2
Series3
Series4
Series5


Load Factor vs. Longest Chain
8
7
6
5
4
3
2
1
0
Longest Chain
0.35
0.3
0.25
0.2
0.15
0.1
0.05
Load Factor
Series1
Series2
Series3
Series4
Series5

Questions:

1. In what ways do the graphs match/not match your hypotheses about which hash functions would be better than others?

Hash functions 1, 2, 3, and 5 seem to behave somewhat similarly, with function 5 behaving somewhat less efficiently than the others. This means my hypotheses were incorrect; functions 4 and 5 are the “worst” while 1, 2 and 3 are the “best”.

2. Which is the worst hash function and why?

Hash function 4 results in the longest chains, largest percentage of collisions and largest percentage of empty buckets. Essentially, it places many elements, most of which can be assumed to be non-identical, in the same buckets, making it less efficient to search/manipulate and giving it the title of worst hash function.

3. Which is the best hash function and why?

Function 3 has slightly less % collisions and % empty buckets than others functions a majority of the time, meaning it is the most efficient. Honorable mention goes to function 5, which is incredibly consistent and never makes chains longer than 3 items, but has higher %collisions and %empty buckets because of this.

4. Are operations on a Hash Table really O(1)?

Not exactly. It depends on how you handle collisions. In a perfect list with no collisions the runtime would be O(1). However if you have any collisions in the table that stops being true because you must perform some sort of iteration.

5. How would the graphs be different if we had used Open Addressing instead of Chaining?

Because of the nature of Open Addressing the Load Factor would be significantly different. As mentioned question 2, hash function had a considerable number of empty buckets, this would not be true will Open Addressing as open addressing priority would be to fill empty buckets.

6. Suppose we were hashing something with a fixed length (int, pointer types, etc). Could we devise a better hash function for these cases? Why or why not?

If you didn’t need to care about memory use you could create a hashtable of tableSize INTMAX. It would have a perfect O(1) runtime as there would be no need for collisions as there would be a spot for every potential item. You would be allocating extortionate amount of space though.