Assignment 7 Analysis

Ray Ding

1. Explain the hashing function you used for BadHashFunctor. Be sure to discuss why you expected it to perform badly (i.e., result in many collisions).

   This functor simply returns the length of the string as the hash code. It is expected to have a high number of collisions, especially for strings of the same length, as it does not consider the content of the strings.

2. Explain the hashing function you used for MediocreHashFunctor. Be sure to discuss why you expected it to perform moderately (i.e., result in some collisions).

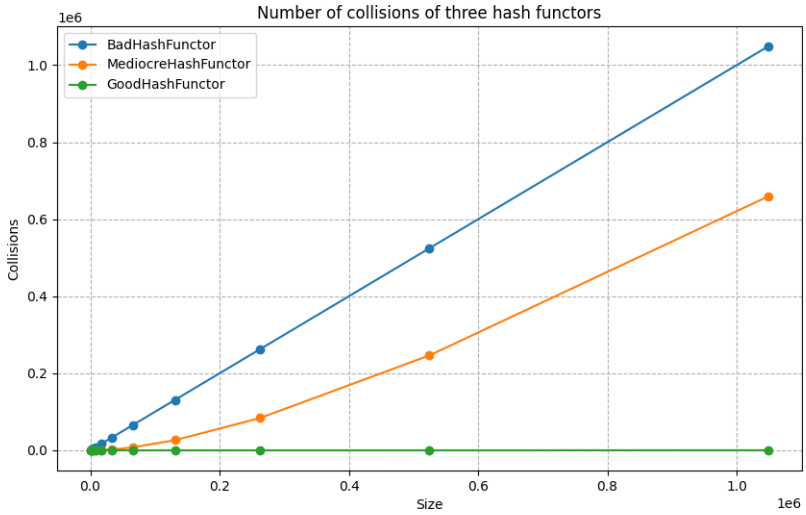
   This functor iterates over each character in the string and performs a simple arithmetic operation (adding the ASCII value of the character multiplied by the square of its position in the string). It is likely to perform better than BadHashFunctor but may still have a significant number of collisions. It incorporates both character values and their positions, but this method is still relatively basic and might not distribute hash codes evenly for different strings, especially if the strings share common patterns.

3. Explain the hashing function you used for GoodHashFunctor. Be sure to discuss why you expected it to perform well (i.e., result in few or no collisions).

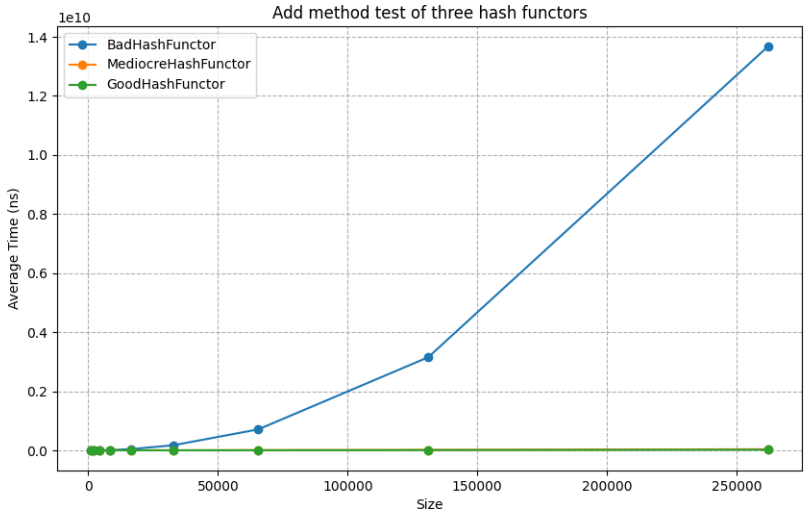
   This functor also iterates over each character in the string. It performs slightly more complex arithmetic operations involving multiplication and addition with a constant prime number. It is expected to have the fewest collisions among the three. It uses a more sophisticated approach (polynomial rolling hash) that considers both the character values and their order. This typically results in a better distribution of hash codes, reducing the likelihood of collisions.

4. Design and conduct an experiment to assess the quality and efficiency of each of your three hash functions. Briefly explain the design of your experiment. Plot the results of your experiment. Since the organization of your plot(s) is not specified here, the labels and titles of your plot(s), as well as, your interpretation of the plots is important. A recommendation for this experiment is to create two plots: one that shows the number of collisions incurred by each hash function for a variety of hash table sizes, and one that shows the actual running time required by various operations using each hash function for a variety of hash table sizes.

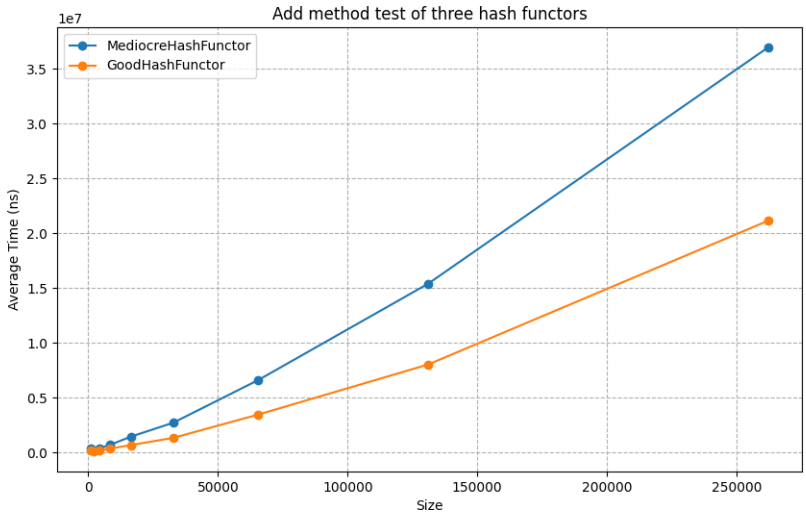
   The figure below shows the number of collisions of three different hash functors as a function of size. The three curves show behaviors as expected. The blue curve displays the linear behavior for worst case (bad hash functor). For good hash functor, the collision number is very small.



The following two figures display the average time it takes to run the addAll() method for those three different hash functors. The Big O is $O(N^2)$ for bad hash functor. It is consistent with the worst-case scenario, since it produces a large number of collisions, which could severely degrade the performance of the hash table operations.



The figure below is an enlarge view of the data of mediocre and good hash functors of the above figure. The Big O of good hash functor is close to O(N), while that of mediocre hash functor slightly deviates from O(N).



5. What is the cost of each of your three hash functions (in Big-O notation)? Note that the problem size (N) for your hash functions is the length of the String, and has nothing to do with the hash table itself. Did each of your hash functions perform as you expected (i.e., do they result in the expected number of collisions)?

   Yes, my hash functions perform as I expected.