CS2420 Assignment010 Analysis

ANDREW WORLEY / U0651238
University of Utah

1. The load factor for quadratic probing determines when the backing array will grow. Also, it is required that the backing array grows to avoid second clustering resulting in quadratic probing never finding a location to insert**.** A load factor no greater than .5 will guarantee that quadratic probing will function properly.

For separating chaining implementation, the load factor will be determined by adding up the sizes of the linked lists storing items in the backing array and finding an average. The lower the load factor the more the backing array will require to grow and rehash previously entered items. The higher the load factor the cost to access items increases as items begin to cluster at the same hash index. Also, the size of the backing array is important, as a backing array of size 1 will work but all values would be stored in one linked list. The size of the backing array will determine the load factor threshold, and a good threshold would be 1 divided by the backing array's size. This will account for large backing arrays requiring less rehashing as they supply more unique indices.

2-4 A hash tester class was used to compare hash functors. The backing array had a size of 23, and the same strings were used. Those strings were "jello", "zello","kello","vello","fellow", "hello","hollow", "cello", "mellow", "bello", and "bellows." No resizing or rehashing was involved in this test.

2. The bad hash functor employed simply used the strings length as the hash index. This produced many collisions for same length strings, but is very fast if no similar sized strings would be stored.

The bad hash functor had 24 collisions when tested using the hash tester mentioned.

3. The mediocre hash functor employed uses recursion to divide every string input into halves and swapping them on the return out. Once the string is scrambled then a result integer is created by adding the character values in the string to the result. This produced significantly less collisions, but suffers from the work required on each string.

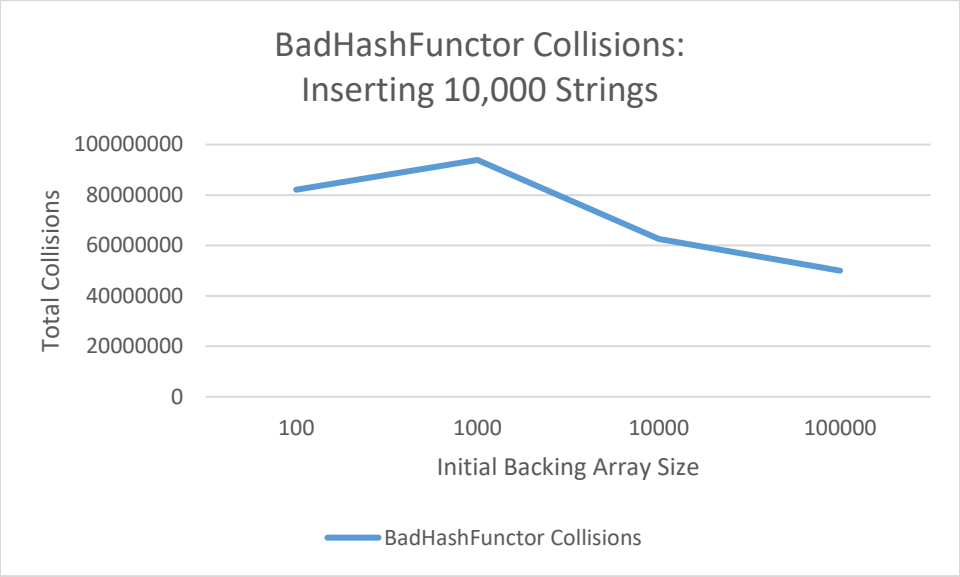
The mediocre hash functor had 3 collisions when tested using the hash test mentioned.

4. The good hash functor multiples the first and last character value then again by a hash value to create a unique hash index. This method is simple and created the least collisions.

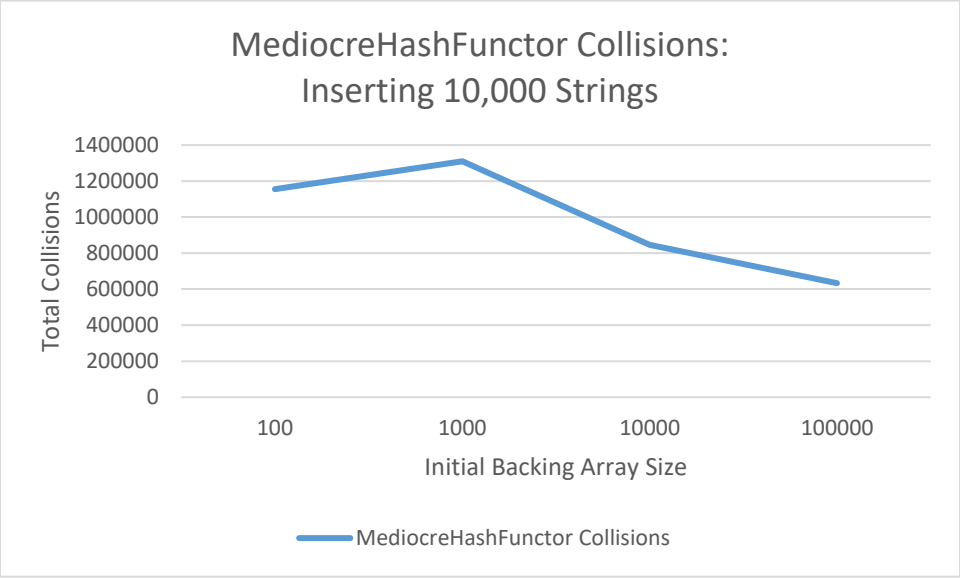
The good hash functor had 1 collision when tested using the hash test mentioned.

CS2420 Assignment010 Analysis

5. The bad hash functor collisions are high as expected.

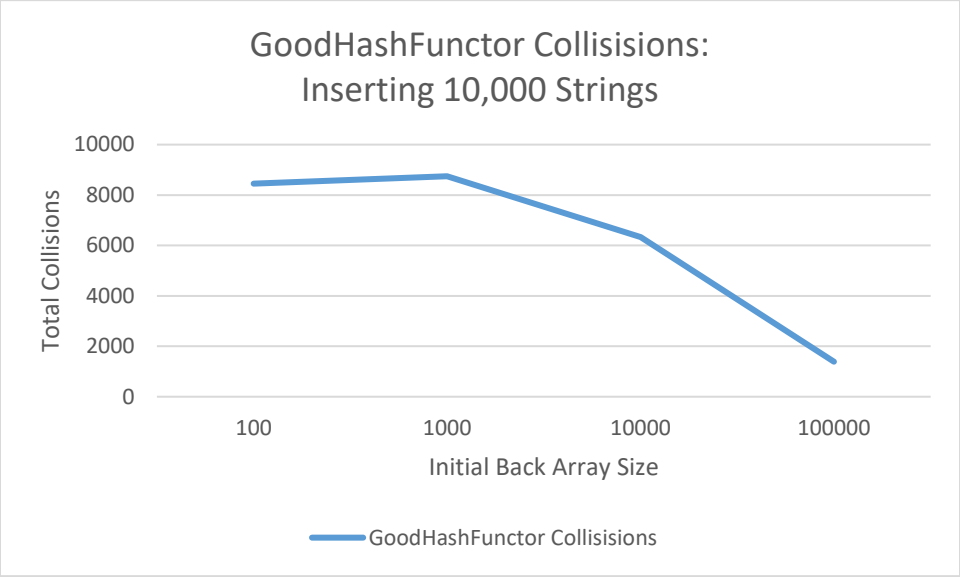


5. The mediocre hash functor shows significant reduction in collisions.

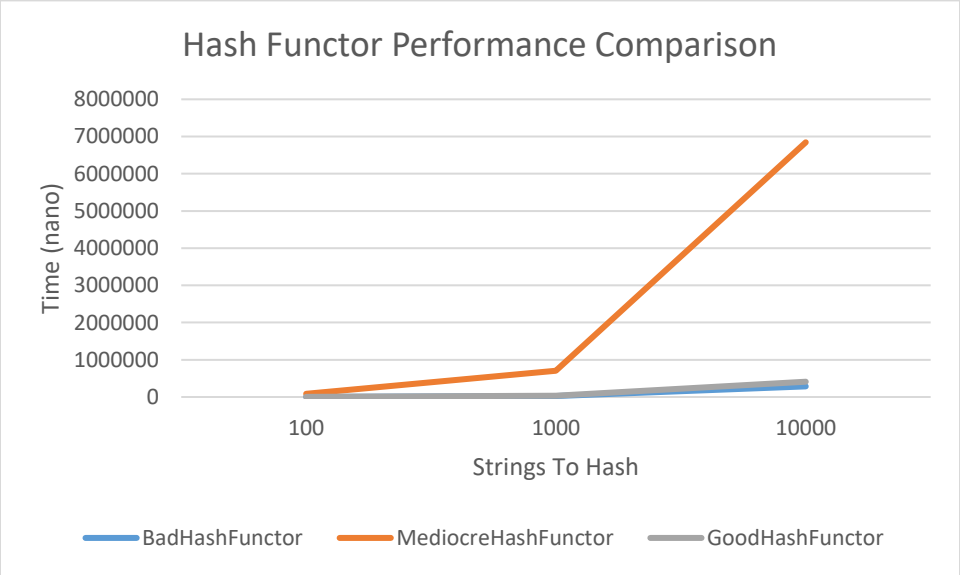


CS2420 Assignment010 Analysis

5. The good hash function sees the best reduction in collisions.



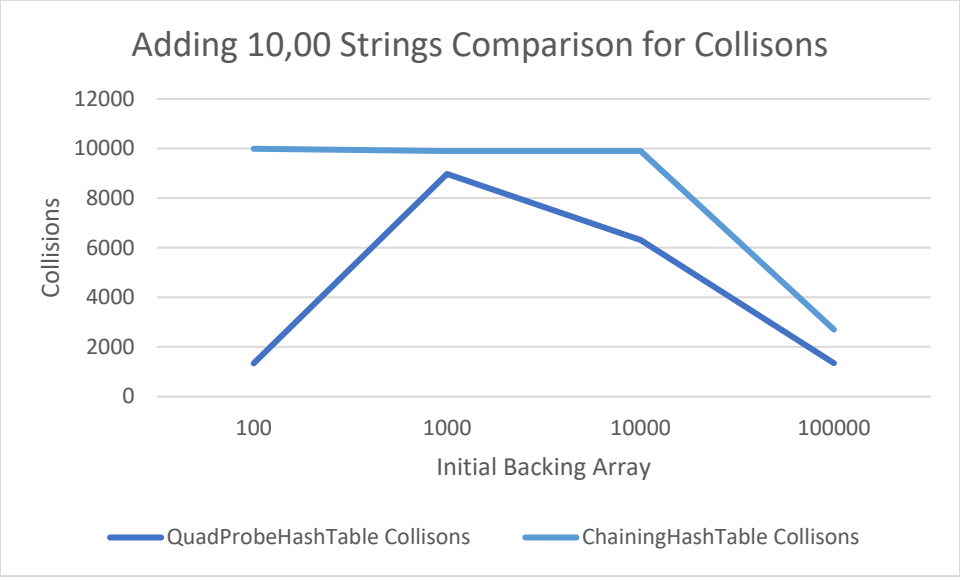
5. The run times on the graph display the issue with the mediocre hash functor being too complicated. The bad and good hash functors display constant time by comparison.



The tests to determine the collisions tracked the total collisions to create each table with 10,000 strings with different starting backing arrays.

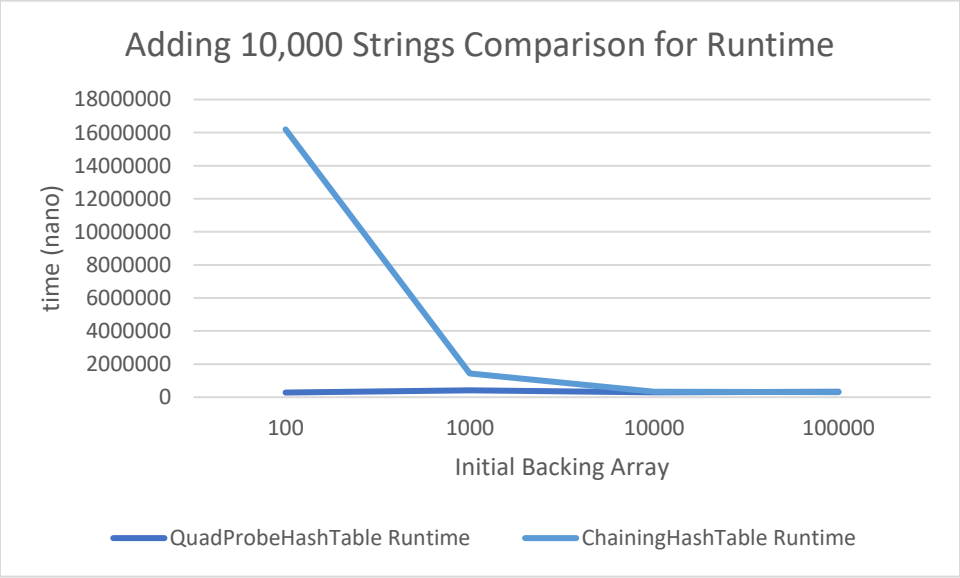
6. The comparison between the quadratic probe hash table and the chaining hash table show that more collisions occur with the chaining table for the circumstances stated in the graphs.

CS2420 Assignment010 Analysis



6. The chaining table shows a higher runtime because it does not grow, but for larger sets

It quickly matches the quadratic runtime.



Conclusion, the chaining hash table is better suited for larger hash sets.

7. The good and bad hash functors demonstrate O(N) complexity, due to that every character in the string is used to create the hash. By comparison the good and bad appear O (C) in the graph displayed earlier. The mediocre hash functor shows O(N^2) complexity as seen in the same graph, which is expected due to its recursive complexity.

8. The load factor determines the frequency to iterate through the set and rehash previously entered items. Poorly chosen load factors will decrease performance, and for quadratic probing it is possible to enter an infinite loop searching for an insert location. Separate chaining will always work as long as the backing array is at least one but performance will suffer if no load threshold is employed or if the load threshold is too low.

CS2420 Assignment010 Analysis

9. One implementation for the quadratic probing hash table would be to copy and modify the contains method to set the found index back to null and reduce the size of the list. For the separate chaining hash table the same mentality would be employed to find the linked list containing the item and calling remove on the list, and reducing the overall size of the hash table.

10. The quadratic probe and separate chaining classes would need to allow generic data. The quadratic probe hash table backing array would use the class generic value, and the separate chaining linked lists would as well. The hashing functor implementation would be fine if only integer hash indices were returned.

11. 10-12 hours.