Spencer Jewett
11/11/16

Assignment 10 Analysis

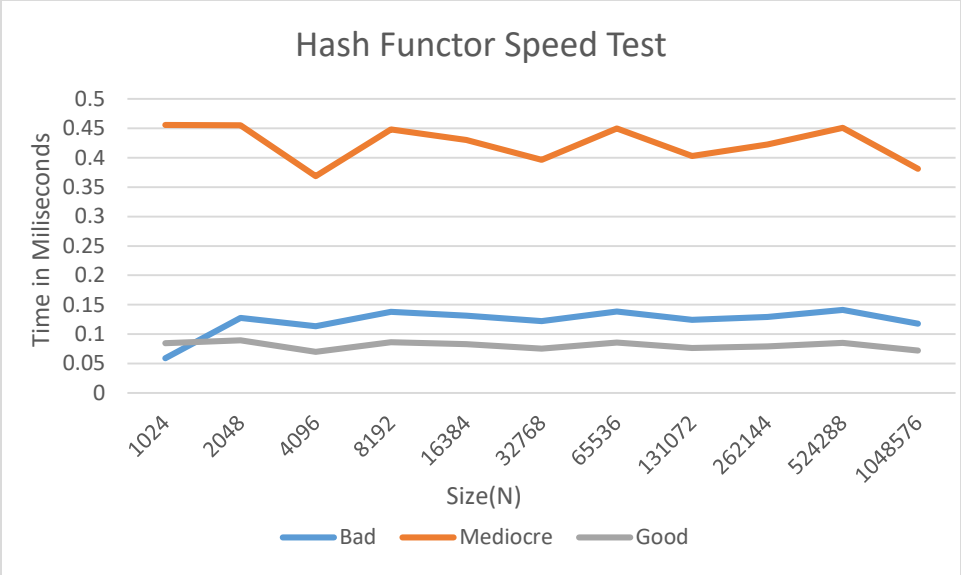
1. The Load factor for the quadratic probing strategy refers to the amount of space in the table that is free, if half or less is available to store items, then the table must be enlarged to allow for it. In the separate chaining instance, the load factor refers to the average length of the linked lists in the table. When the load factor reaches a critical mass, as determined by a chosen algorithm, the table is enlarged and rehashed. While beneficial this is unnecessary for this strategy

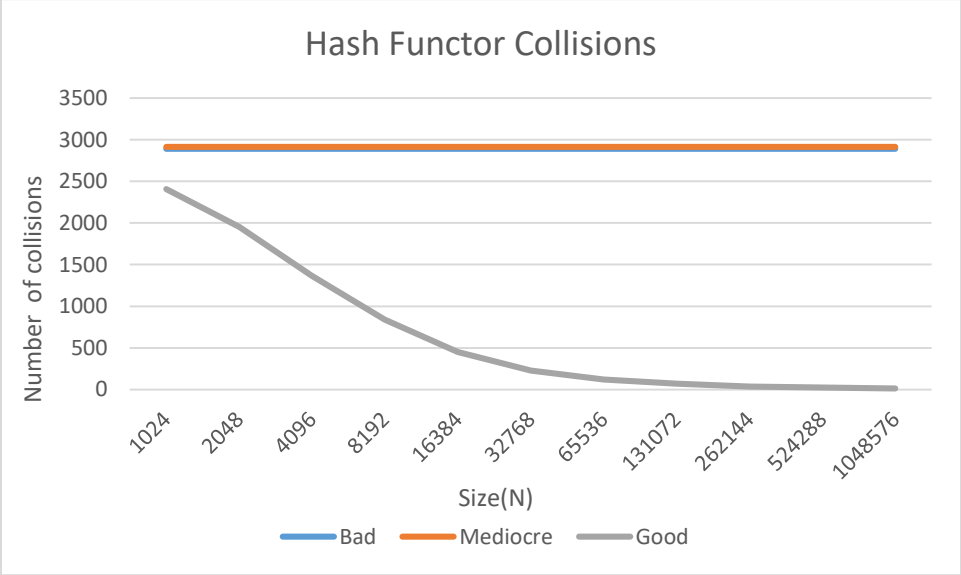
2. Take the item being inserted, pull the last character in the string, return it's ANSCII number. In the case of the empty string or "", 0 is returned. It is expected to perform badly because there are many words that end with the same letter and therefor will cause collisions

3. The moderate hash functor is a little more complex than the basic one, if the string inputted is empty it returns 0, if it has one character, that character's ANSCII number is multiplied by seven and returned if it has two characters the first is multiplied by seven then added to the value of the second character. If there are three or more, the value of the first character is added to the value of the second character. That sum is raised the the value of the third character, this number is multiplied by seven and then returned. For strings of 3 or more characters, this will allow for greater variety of hash numbers then the bad functor.

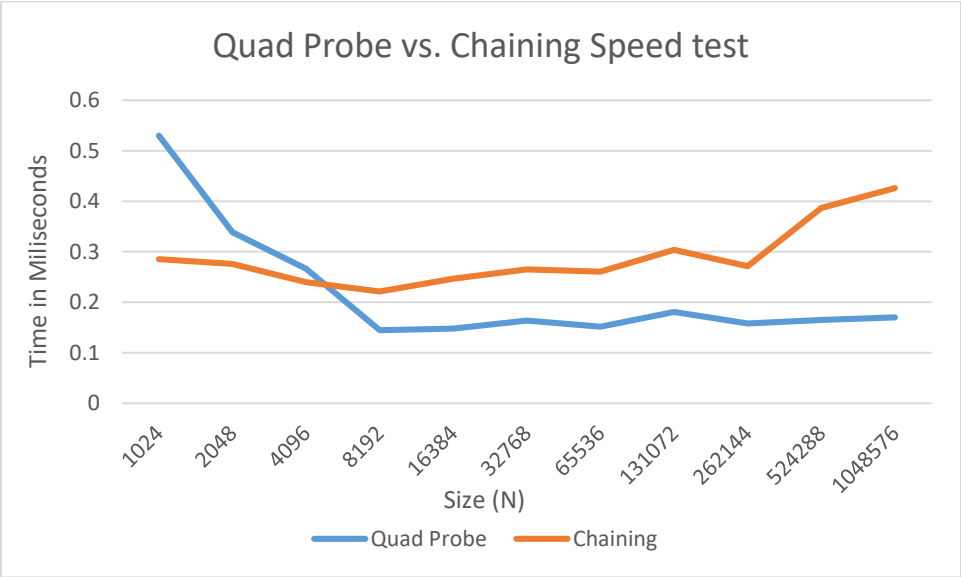
4. The hash is started with a number:5381, then if the string is empty, 0 is returned. If it is not empty, we dump it into a for loop that loops as many times as there are characters in the inputted string. We take the hash and shift the value to the left by 5 then minus the result by the original hash, then add a character at i where I is the number of times the for loop has run, after going through all the letters in the string, the solution is returned. It should work better as it's been given a prime seed and also resolves any int overflow issues. This is expected to be far better as it imitates a proven formula XXX

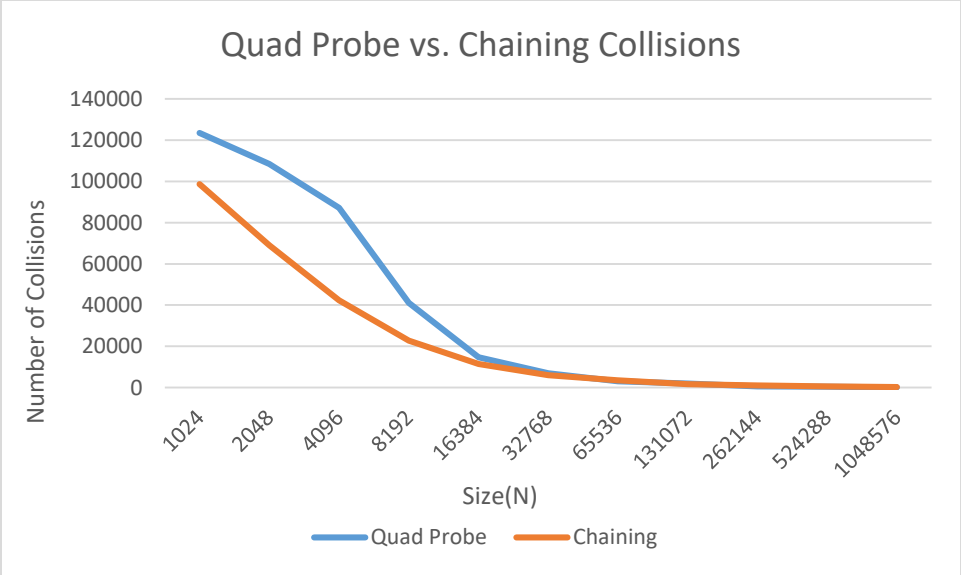
5. For the Hash Functor test, each functor is timed running a collection of hashes, these times are averaged out over 1000 tests for each set size which starts at 1024 and increases by 2^(10+i), i being the iteration we are on. To get the collisions we use the collision counter in the program while adding lists of increasing sizes, sizes the same as aforementioned speed test. Here are the results:


Hash Functor Speed Test
Time in Miliseconds
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
1024
2048
4096
8192
16384
32768
65536
131072
262144
524288
1048576
Size(N)
Bad
Mediocre
Good


Hash Functor Collisions
Number of collisions
3500
3000
2500
2000
1500
1000
500
0
1024
2048
4096
8192
16384
32768
65536
131072
262144
524288
1048576
Size(N)
Bad
Mediocre
Good

6. To test the two strategies, we inputted a list of strings (starting at size 1024) into each program and averaged out the time it took to do this 1000 times at the set size, the set size would be incremented by 2^(10+i) where i is the iteration we are on. We iterate this test ten times plot the results. For collisions we simply averaged out the number of collisions per grouping of tests. The results are as follows:





7. Complexity of BadHashFunctor is O(c), MediocreHashFunctor is O(c), GoodHashFunctor has a complexity of O(N)

8. If the Load factor for the quadratic strategy is not managed well it could lead to rehashing more often than is needed, and lower the performance of the table. If a massive quantity of items all end up in the same bucket in the chain strategy it can bog down performance for all items that end up with the same hash as that bucket.

9. To implement a remove method for the hash table, you would take the item you want removed, find it's hash, modulo that by the capacity to get its initial location, depending on which table you are using you then account for collision possibilities while searching for the item. If the item is found, you set the bucket at the specified index to null, minus one from the size and return true. If not found false would be returned

10. Yes, it is possible to make these generic, it would require going in and changing the functors to allow any input and modify the code as well, all the methods would have to be adapted for generic, doing type checks for contains while in the backing array the buckets could hold objects, allowing for all types of information to be stored within the table

11. I spent 22 amount of hours on this assignment