Caleb Kumar

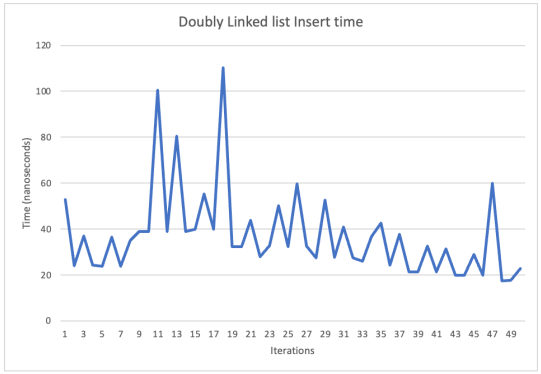
CSCI 2270

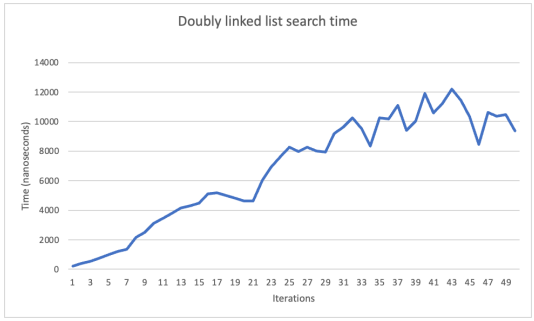
Abhidip B.

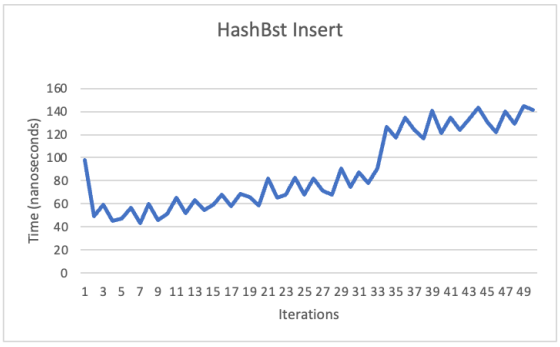
7/19/2022

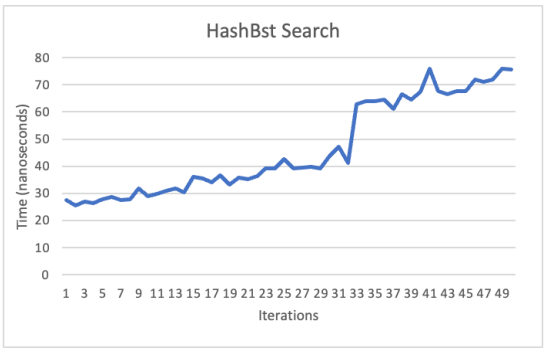
Insert/Search Times of Data Structures

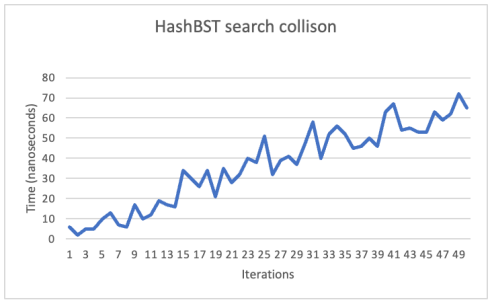
For this experiment I timed 3 different data structures, a doubly linked list, a hash table where the indexes of the table are binary search trees, and another hash table where I doubled hashed a key if the index was preoccupied. For the hash tables, I recorded the number of collisions for both, inserting, and searching. The doubly linked list I used had a tail pointer, new nodes were appended to the end through this tail pointer. The doubly linked insert graphs conformed to the O(1) constant insert time, the search time conformed with the expected O(n) time. Searching takes longer with a linked list as you must traverse the list from either the start or the end. The hash BST, using the BST to deal with collisions, if the keys are the same in placed them into the right tree. The hash BST produced graphs consistent with their respective times for insert, and search (O(log n)). Using a hash table with double hashing as a method of dealing with collisions was the worst data structure, both collisions and search times are large, and continue to get larger with more data. While the hash double hashing has a good insert time the trade off for overall usage is not worth it. For the best data structure for the CU system, I would choose the hash BST structure, while all operations conform to their respective time complexities, the graphs show the times all to be below 200 nanoseconds and collisions significantly less than the double hashing table. I hypothesize the reason that the hash BST would perform the best is because of its time complexity for all operations. The structure of BSTs make dealing with collisions and searching easy. Graphs available on the next pages.

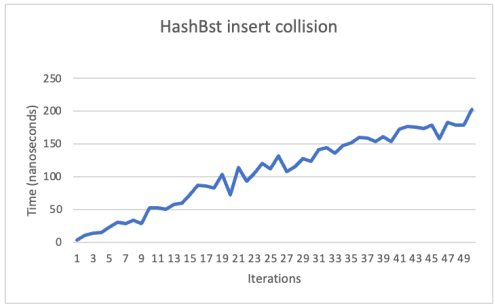

Doubly Linked list Insert time
Time (nanoseconds)
Iterations

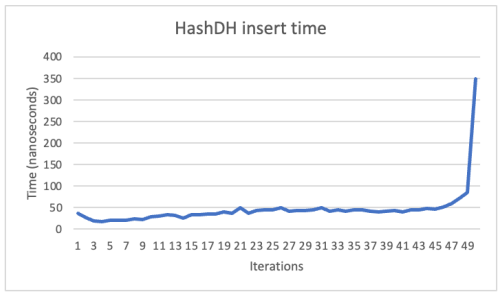

Doubly linked list search time
Time (nanoseconds)
Iterations

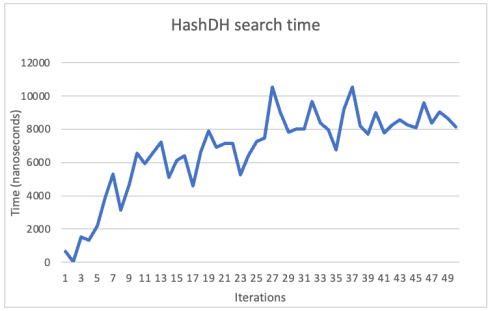

HashBst Insert
Time (nanoseconds)
Iterations

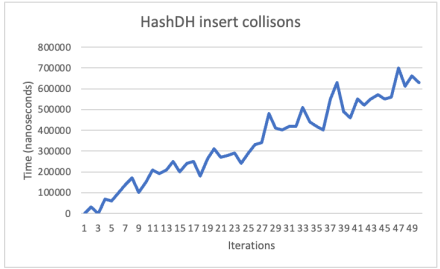

HashBst Search
Time (nanoseconds)
Iterations

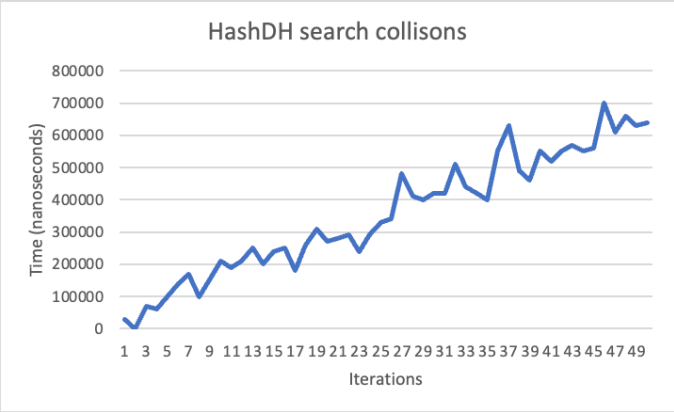

HashBST search collison
Time (nanoseconds)
0 10 20 30 40 50 60 70 80
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
Iterations

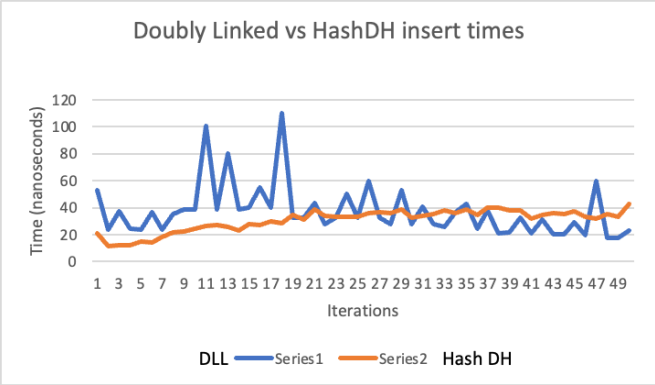

HashBst insert collision
Time (nanoseconds)
0 50 100 150 200 250
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
Iterations

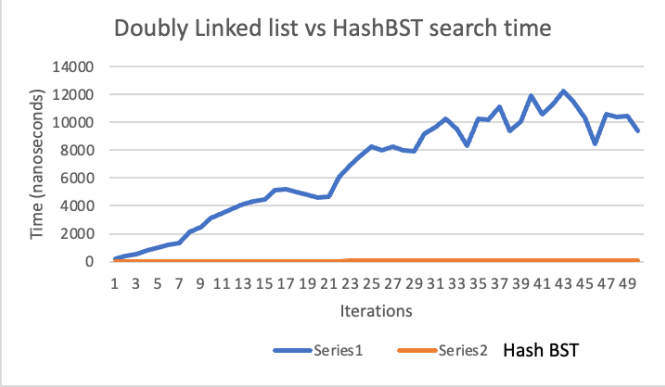

HashDH insert time
Time (nanoseconds)
0 50 100 150 200 250 300 350 400
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
Iterations


HashDH search time
Time (nanoseconds)
0 2000 4000 6000 8000 10000 12000
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
Iterations


HashDH insert collisons
Time (nanoseconds)
0 100000 200000 300000 400000 500000 600000 700000 800000
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
Iterations


HashDH search collisons
800000
700000
600000
500000
400000
300000
200000
100000
0
Time (nanoseconds)
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
Iterations


Doubly Linked vs HashDH insert times
120
100
80
60
40
20
0
Time (nanoseconds)
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
Iterations
DLL
Series1
Series2
Hash DH


Doubly Linked list vs HashBST search time
14000
12000
10000
8000
6000
4000
2000
0
Time (nanoseconds)
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
Iterations
Series1
Series2
Hash BST