University of Colorado Boulder

Data Structures

CSCI 2270

Final Project: Optimizing Insertion and Retrieval of Patient ID Numbers for a Contagion Tracking Application

Author:
Christopher Hanson

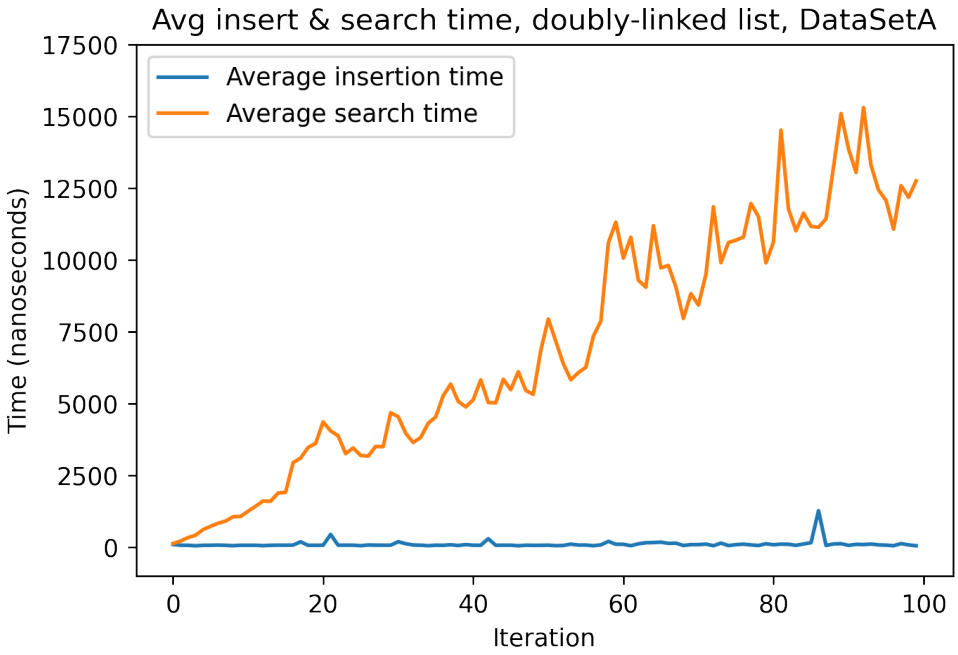
24 July 2020

Summary

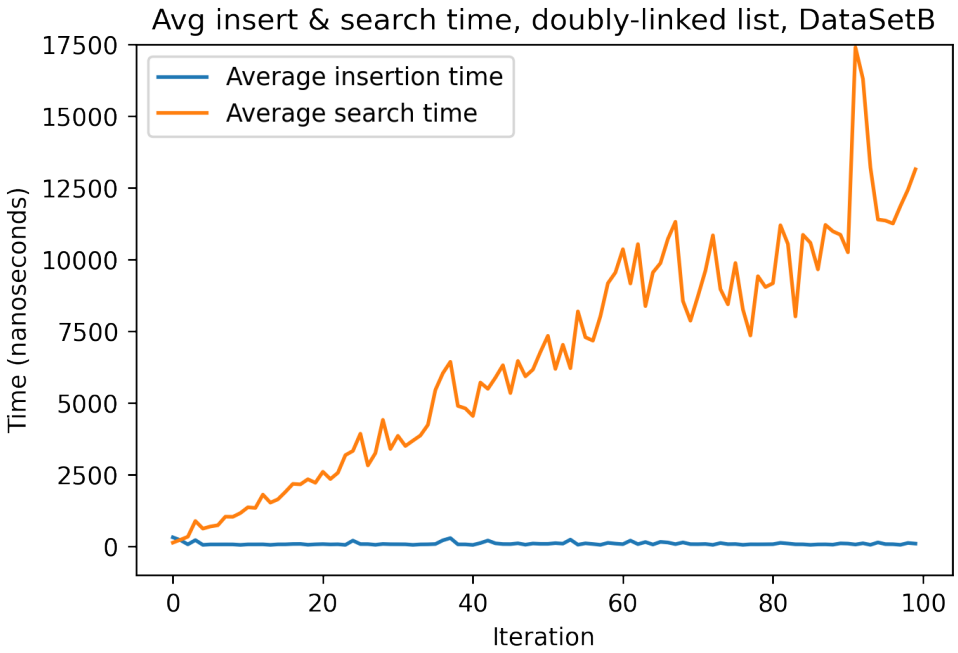
The goal of this project is to compare doubly-linked lists to multiple variants of a hash table to find a data structure that strikes the best balance in runtime performance for a fictitious contagion tracking application. A hash table with the quadratic probing open addressing scheme is identified as the best available data structure for this contagion tracking application. This finding should be expected, because hash tables are known to have a time complexity of $O(1)$ for both search and insert operations (Hoenigman, 2015).

Experiments and analysis

Doubly-linked list (`dll.cpp`) A doubly-linked list has $O(1)$ time complexity for insertion when each insertion is made at the head of the list using a head pointer or at the tail using a tail pointer, but $O(n)$ time complexity for search operations (Hoenigman, 2015), a fact which was affirmed by the current experiment. The figure below shows the search and insert performance on Data Set A of the attached implementation of a doubly-linked list, with time taken per insertion seen in blue, and time taken per search seen in orange. All figures in this paper show each experiment over 100 iterations, with each iteration itself being the average of 100 insertion or search operations, for a total of 10 000 operations per experimental criterion.

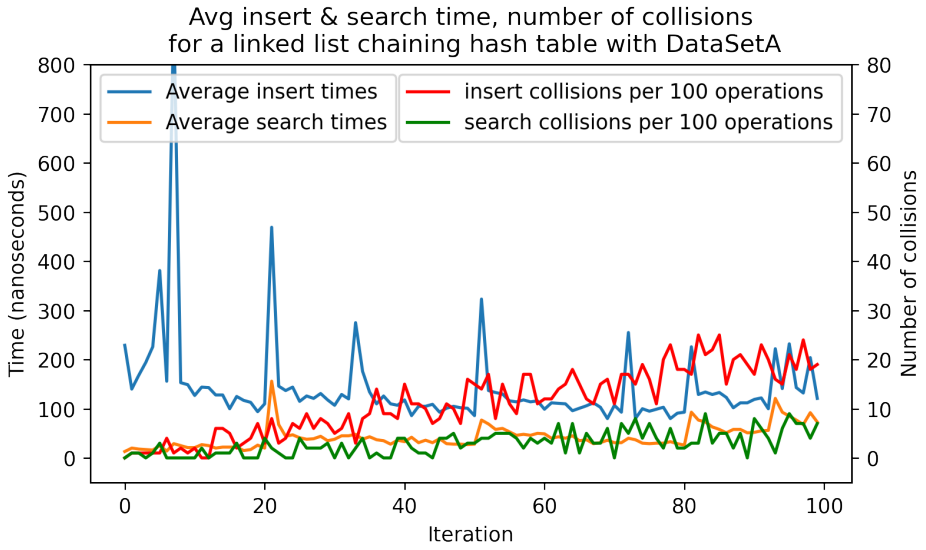


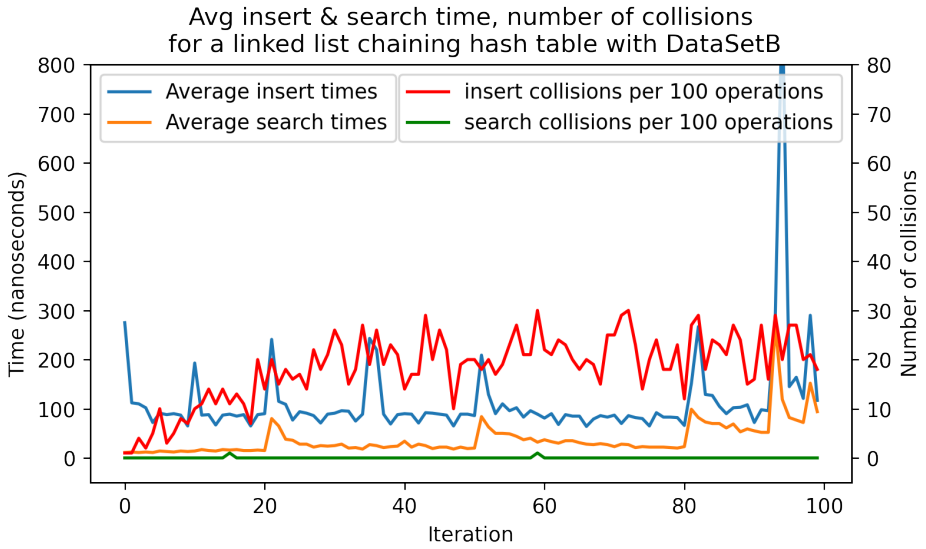
Effectively identical results were found using Data Set B, as it is still $O(1)$ for insertion but $O(n)$ for search:



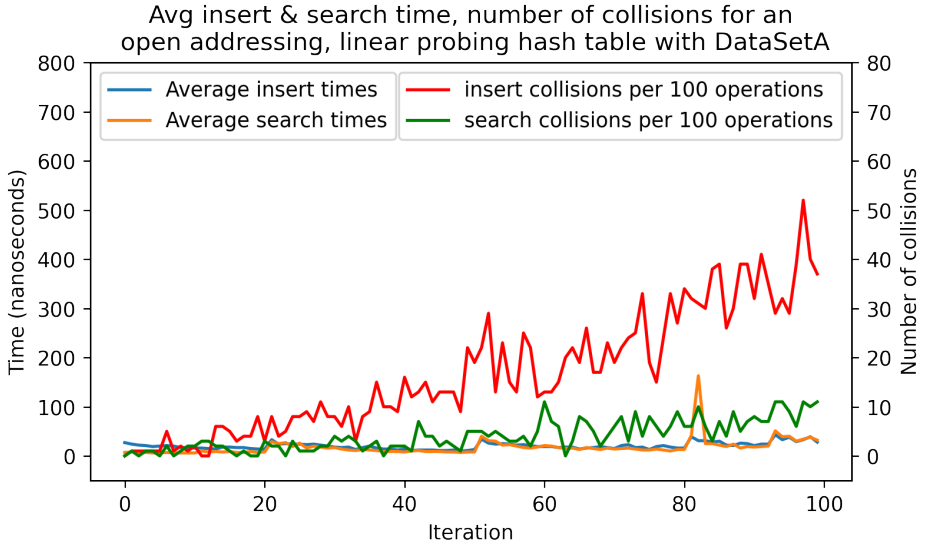
Doubly-linked list (`dll.cpp`)

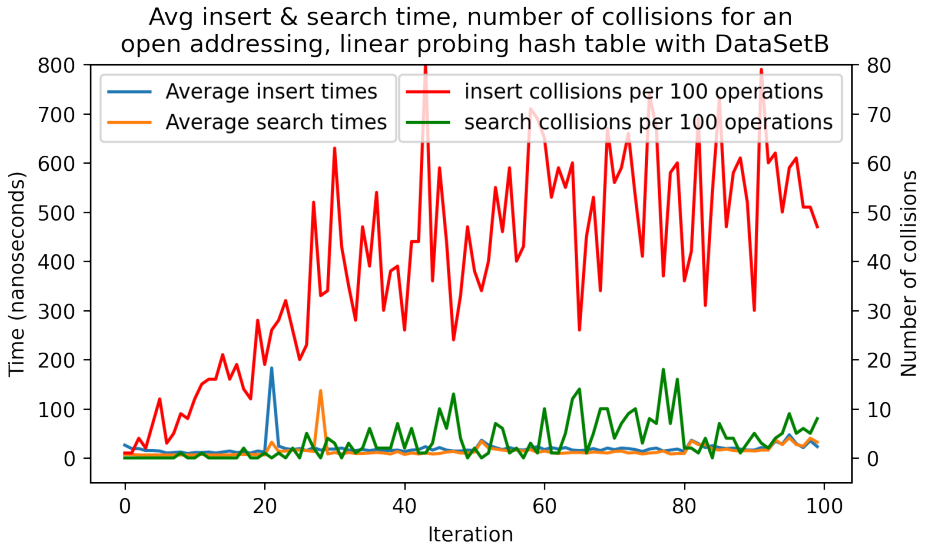
Three hash table implementations with varying means of collision resolution (`hashchain.cpp`, `hashlinear.cpp`, and `hashquad.cpp`)
The next six figures show search and insert performance for the attached implementations of hash tables (with two figures on each page, starting with linked list chaining on the next page (page 3), followed by open addressing with linear probing on page 4, and finally open addressing with quadratic probing on page 5). Across all six figures we can see that search is $O(1)$ for each hash table variant. The numbers of insert collisions in red can be seen to grow against the right vertical axis (and most dramatically for the linear probing variant) as the number of patient IDs in the table increases the hash tables' load factors, but this is of less concern because it is possible to mitigate this by increasing a hash table's size when necessary. The final two figures (page 6) compare the best-performing hash table variant with the doubly-linked list implementation.



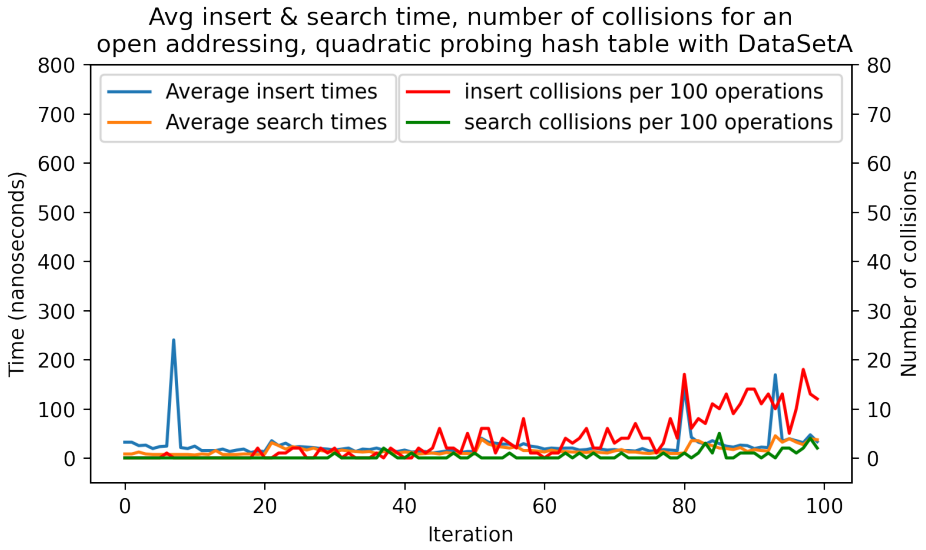


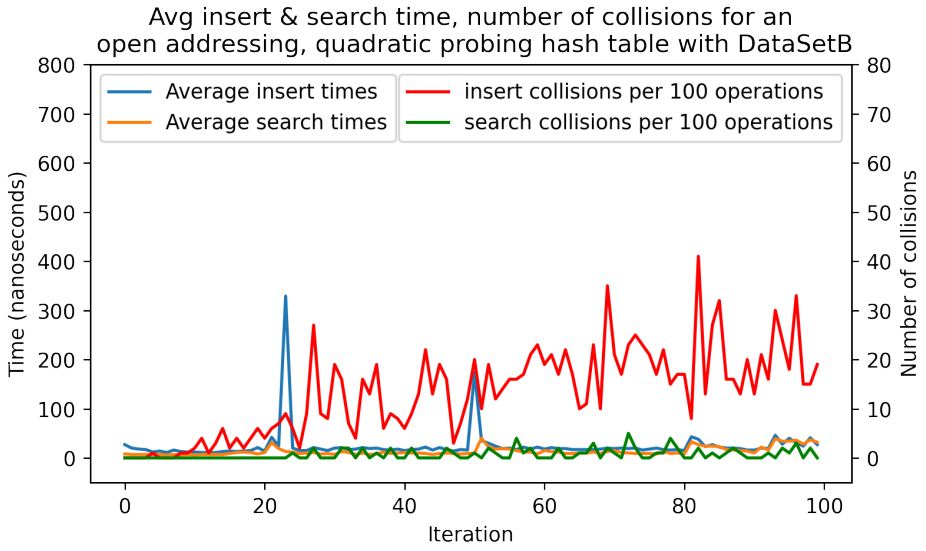
Hash table with linked list chaining (`hashchain.cpp`)



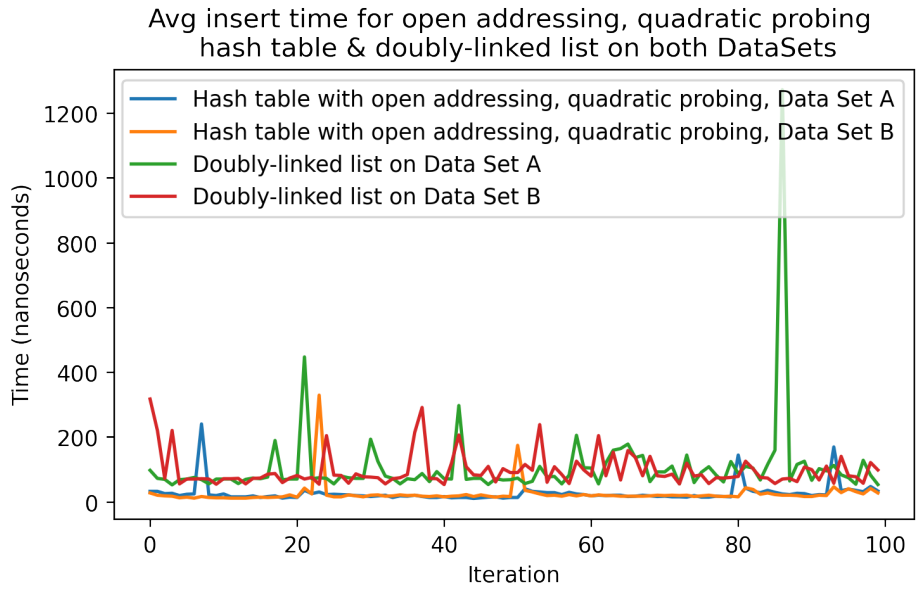


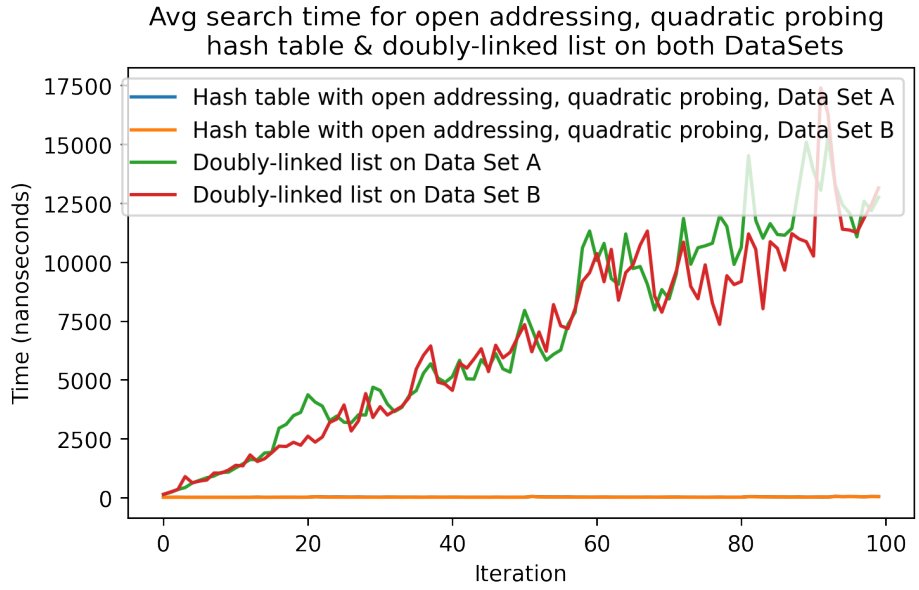
Hash table with open addressing, linear probing (`hashlinear.cpp`)





Hash table with open addressing, quadratic probing (`hashquad.cpp`)





Comparison of insert times (top figure) and search times (bottom) between the doubly-linked list and the hash table with the quadratic probing open addressing scheme

Discussion and conclusion

While the three variants of hash table—linked list chaining, linear probing, and quadratic probing—all showed $O(1)$ search time complexity, and all but the linear probing hash table showed $O(1)$ insert time complexity, the quadratic probing variant performed best among them and among all data structures covered in this paper. Hash tables clearly perform much better than doubly-linked lists on search (see the bottom figure on page 6), and that is because just one computation can usually index exactly where the desired value should be in the table, provided the hash table has an appropriate prime number as the modulo and the table size. The more difficult comparison to make is between hash table variants, but even there the choice is fairly clear in the current case.

Quadratic probing vs linear probing When compared to the linear probing variant, the quadratic probing variant had 4.5 times fewer insert collisions (369 vs 1671) and ten times fewer search collisions (41 vs 404) on Data Set A. On Data Set B, it had three times fewer insert collisions (1322 vs 4023) and five times fewer search collisions (69 vs 354).

Quadratic probing vs linked list chaining The quadratic probing variant similarly outperformed the linked list chaining variant hash table. In this comparison, the quadratic probing variant had three times fewer insert collisions (369 vs 1130) and 6.5 times fewer search collisions (41 vs 274) on Data Set A, and on Data Set B it had 28% fewer insert collisions (1322 vs 1842), although far more search collisions (69 vs 2), but this is likely because of the required table size/modulo which, at a value of 40 009, is a better fit for Data Set A (whose average value was 40 526.4) than for Data Set B (whose average value was 55 233.2), where the resultingly more frequent collisions are resolved through the more expensive process of quadratic probing than through simple linked list chaining.

In conclusion, a hash table with quadratic probing is the recommended data structure for patient ID insertion and retrieval in this contagion tracking application, with the recommendation that some attention is also given to choosing an appropriate table size/modulo.

References

Hoenigman, R. (2015). *Visualizing data structures.* Lulu Press.