Alan Urbanek

Professors Brennan, Jamalian

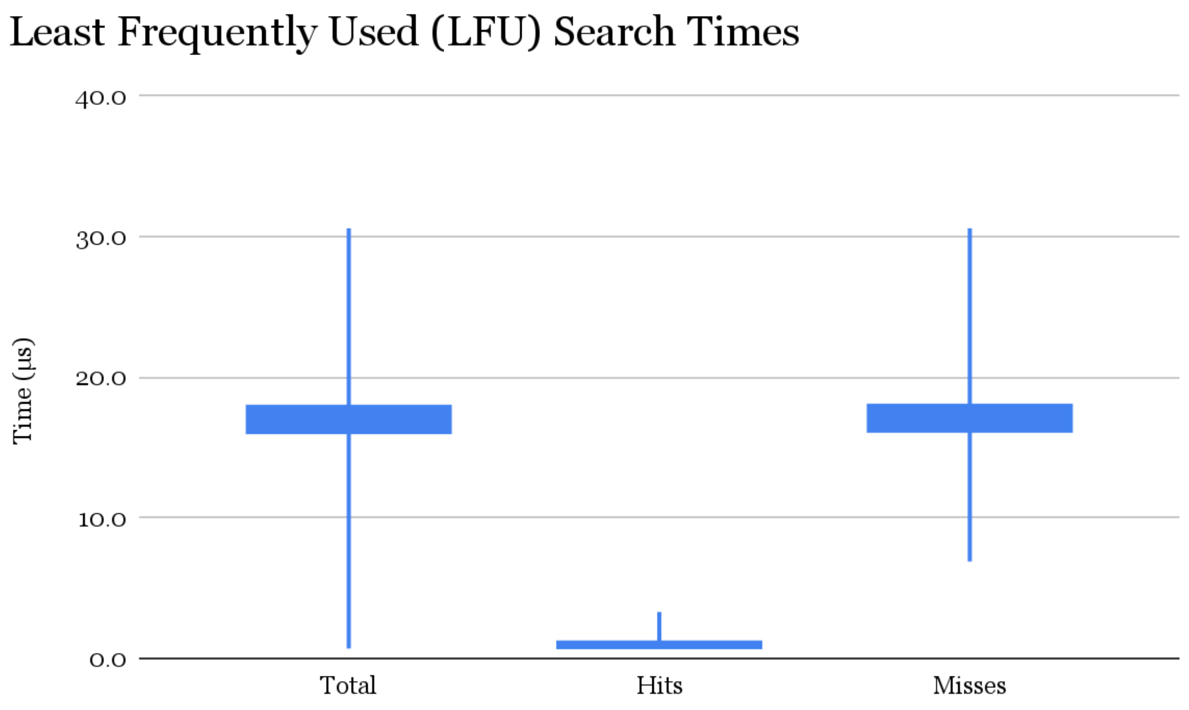
Advanced Data Structures and Algorithms

13 May 2025

Trie and Cache Access Performance Analysis

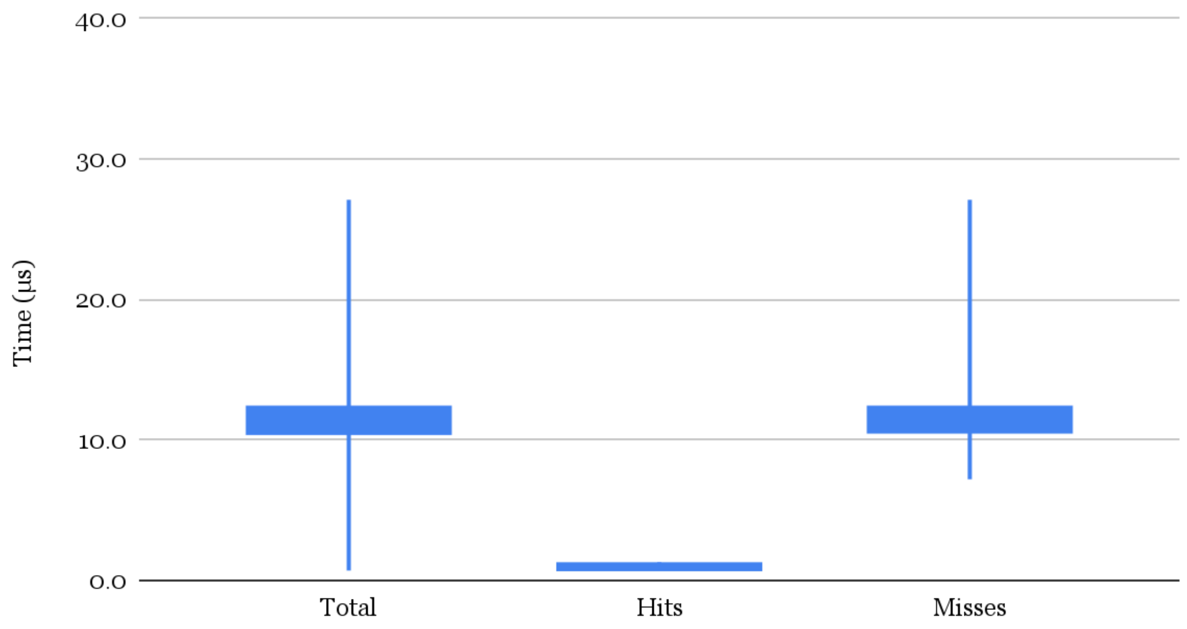
This semester's Advanced Data Structures and Algorithms course has culminated in this Final Project: a population lookup program that has evolved over the past four weeks. Initially, it was simply searching a database of cities, with a small cache to store recent searches for *much* faster repeated searches, replacing cache items with the First In, First Out (FIFO) method when it became full. Then, alternate cache deletion methods were implemented, to be selected by the user at the beginning of a program run: deleting the Least Frequently Used (LFU), or deleting randomly (the FIFO method was also kept). Next, we implemented a trie for the database to be loaded into at the start of the program; when the cache misses, the trie is searched instead of the original database, which is far faster. Now, finally, we have written a separate script to perform performance analyses of a high volume of searches (500) using each deletion scheme: LFU, FIFO, and Random. Like the previous Midterm Project, each search was timed with the `high_resolution_clock` from C++'s `<chrono>` library, with additional output to determine if the search was found in the cache or if searching the trie was required. The given world cities database contained 47,980 entries, and each test searched 500 entries randomly but with the same seed so each test used the same random searches.

First, let's discuss the Least Frequently Used deletion scheme, which ended up being the worst of the three. This relatively low performance is likely a result of the logic to determine which cache item is currently the least used, which requires iterating through the entire cache and updating the deletion selection whenever a less used result is found. Essentially, it needs to read the entire cache instead of the first item in the cache like FIFO or a specific random item like the Random scheme. Most of the searches fell between 16 and 18 microseconds, with the worst taking nearly 31 microseconds. On the lower end, the “outliers” are actually the times of the searches that resulted in a cache hit, showcasing how immensely faster searching the cache is compared to searching the whole trie. Most searches that hit the cache took around 1 microsecond, with the slowest taking 3. Interestingly, in this sample of searches, LFU also had the least amount of hits, with only 18 of the 500 searches (3.6%) resulting in a cache hit.

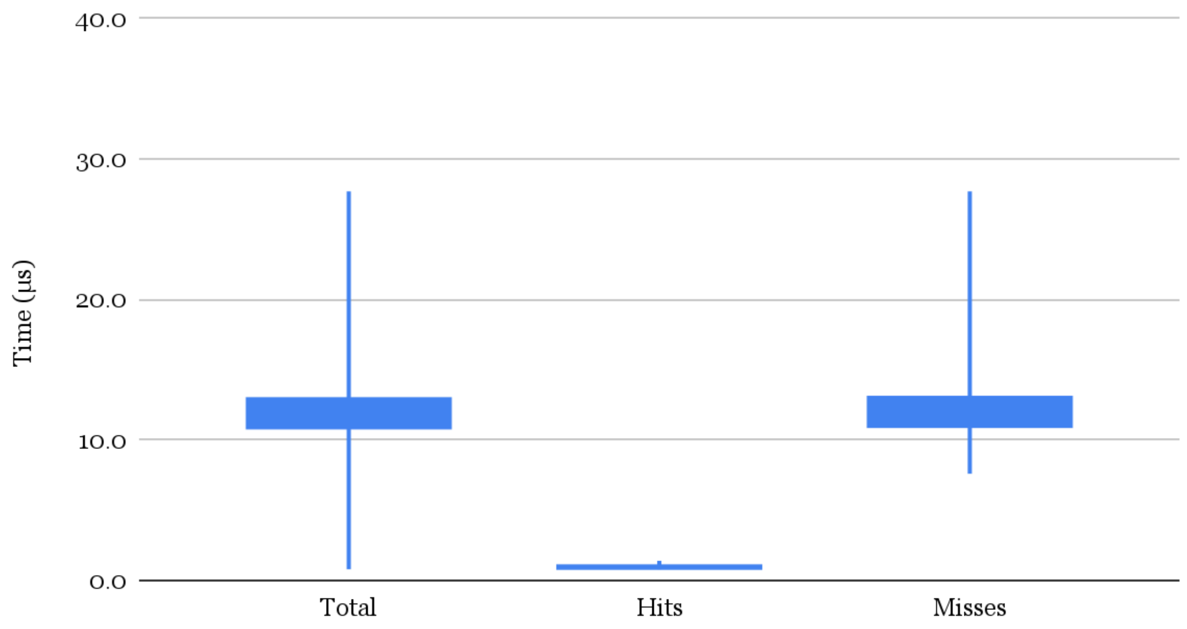


Next, the FIFO and Random deletion methods were actually very similar in their performance. The range of times for their hit searches was 0.7-1.3 microseconds and 0.8-1.4 microseconds, respectively, and the range of their missed searches was 7.2-27.1 microseconds and 7.6-27.7 microseconds, respectively. The less-than-1 microsecond increase on the Random method’s part is likely just due to the time it takes to pick the random item to delete, instead of automatically going for the oldest item in the cache. Plus, if it selects an empty cache bucket to delete from, it keeps picking buckets until a non-empty bucket is found, which can make selection times vary depending on how exactly the cache is filled.

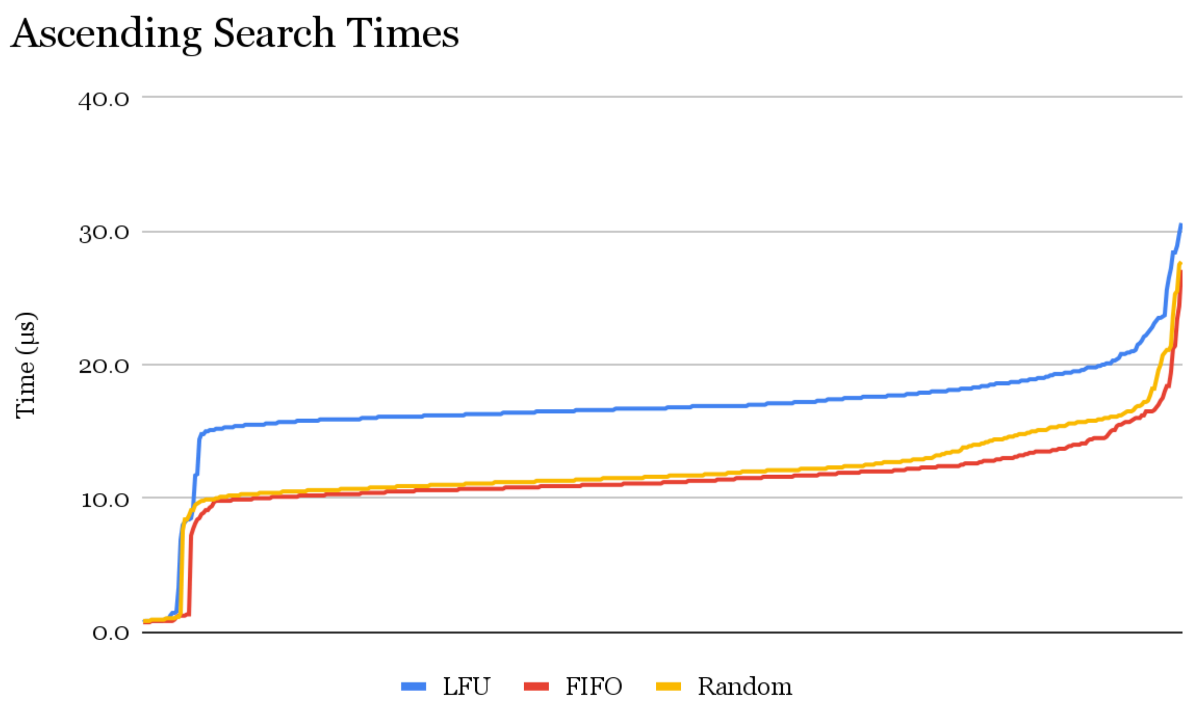
First In, First Out (FIFO) Search Times



Random Search Times



To truly determine which method was the best in this test (although it was clearly between FIFO and Random), I plotted the time for every search under each deletion scheme, in ascending order:



Each deletion scheme follows the same pattern: a sharp jump after the first 20 or so searches (the difference between hit searches and missed searches), and then slowly increasing search times as the test goes on, with larger increases towards the end—it is not immediately clear to me why this occurs, other than perhaps the trie taking exponentially longer to search under the worst cases. Either way, this makes the disparity between LFU and the other two deletion schemes even clearer, but more importantly, it shows that FIFO performed visibly better than Random selection at nearly all points in the chart. Thus, I come to the conclusion that that First In, First Out is the best cache deletion scheme to use in a scenario like this, with Random selection being a close second.