Dan Kondratyuk

Data Structures I

Charles University

31 December 2017

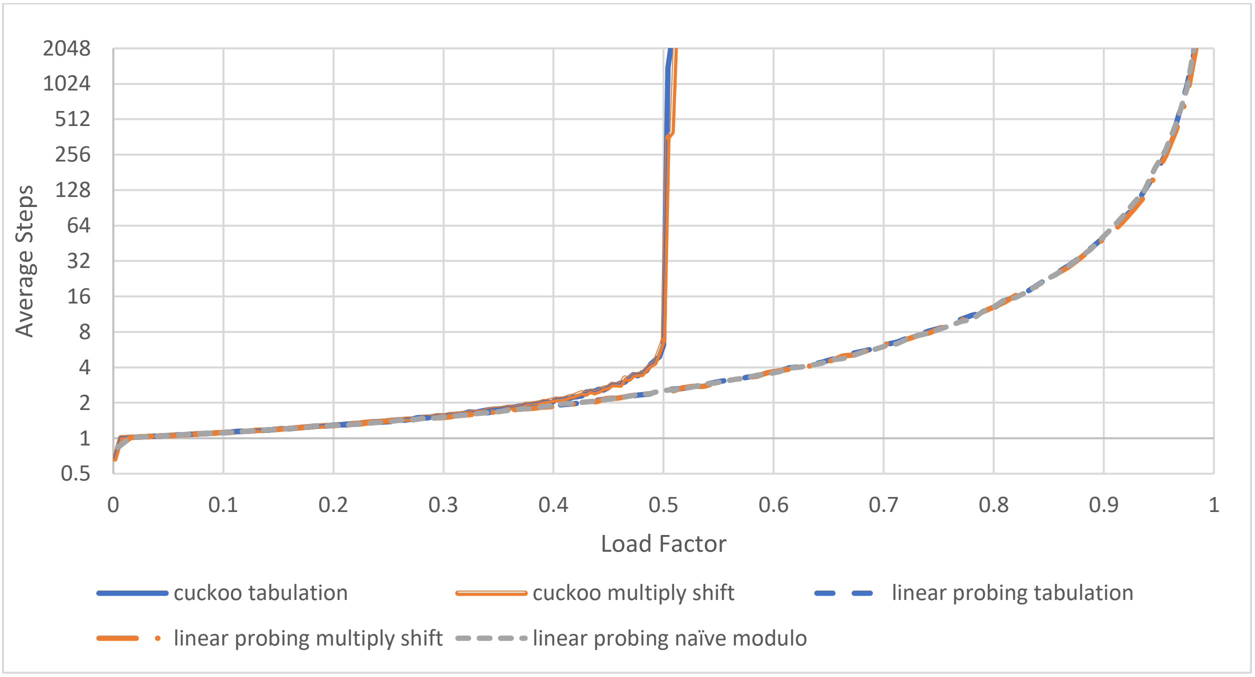
Assignment 4

This assignment analyzes the performance of various hashing strategies using 32-bit integers.

I. Random Case Study

The random case study studies how the running time and number of steps depend on the load factor for various combinations of hashing systems and insert functions. The first test plots the average number of steps needed to insert an element at a given table load factor (alpha). Both graphs use a table size of $m = 2^{20}$.

Figure 1: Load Factor vs. Number of Steps



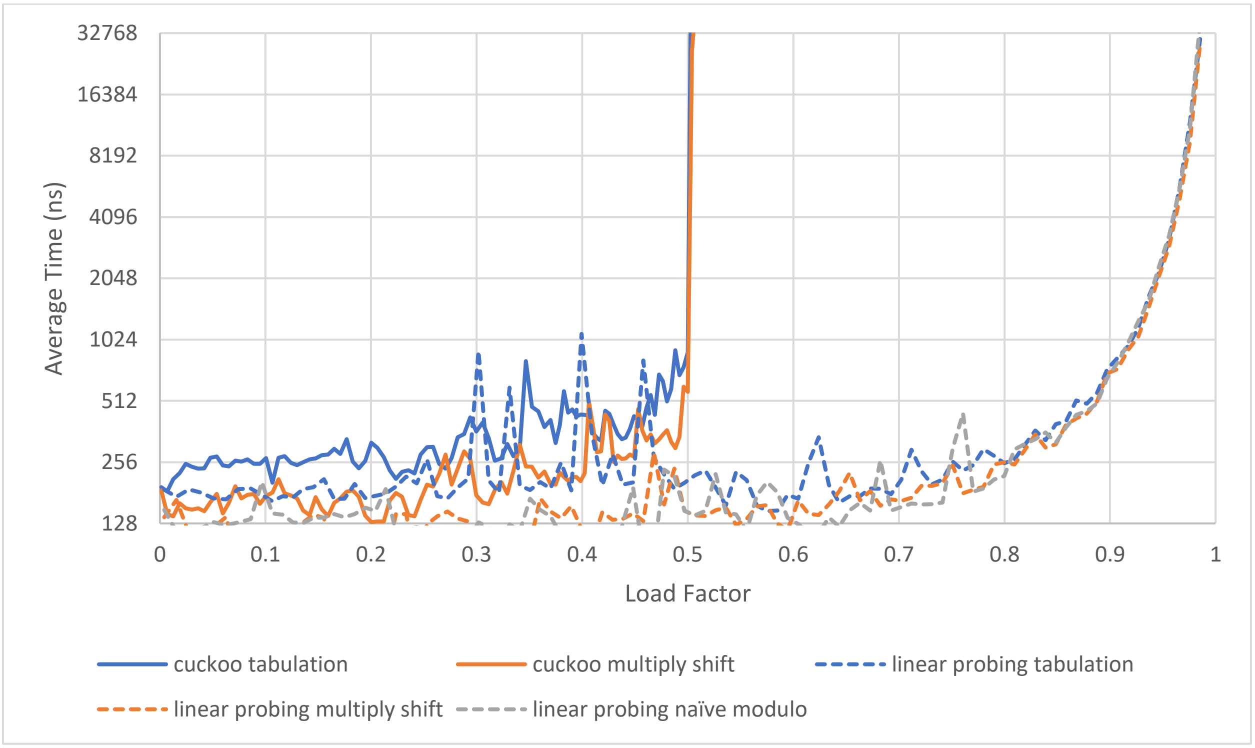
Because all elements are generated uniformly at random, the complexity of insert is not dependent on the type of hash function as they will all behave as if they are totally random.

There would only be differences among hash systems if the elements have some dependence to one another. Therefore, the complexity in the random test is only dependent on the type of hash scheme, i.e., cuckoo hashing or linear probing.

As expected, cuckoo hashing performs poorly as the load factor surpasses 0.5. Expected amortized complexity is $O(1)$ for load factors below 0.5, but as it gets larger the probability a cycle will occur increases exponentially, and so too does the number of necessary table rebuilds. A single rebuild increases the number of steps by n, which is an expensive operation.

Linear probing increases more steadily, following a hyperbolic curve as the load factor approaches 1. This can be viewed simply from the fact that the probability that a random element is at least k steps from the nearest open slot increases substantially as slots fill up with elements. It is expected that an insert operation with one free slot will need to perform $\frac{m}{2}$ steps. By extension, the fewer slots there are, the more steps are necessary per insert.

Figure 2: Load Factor vs. Time

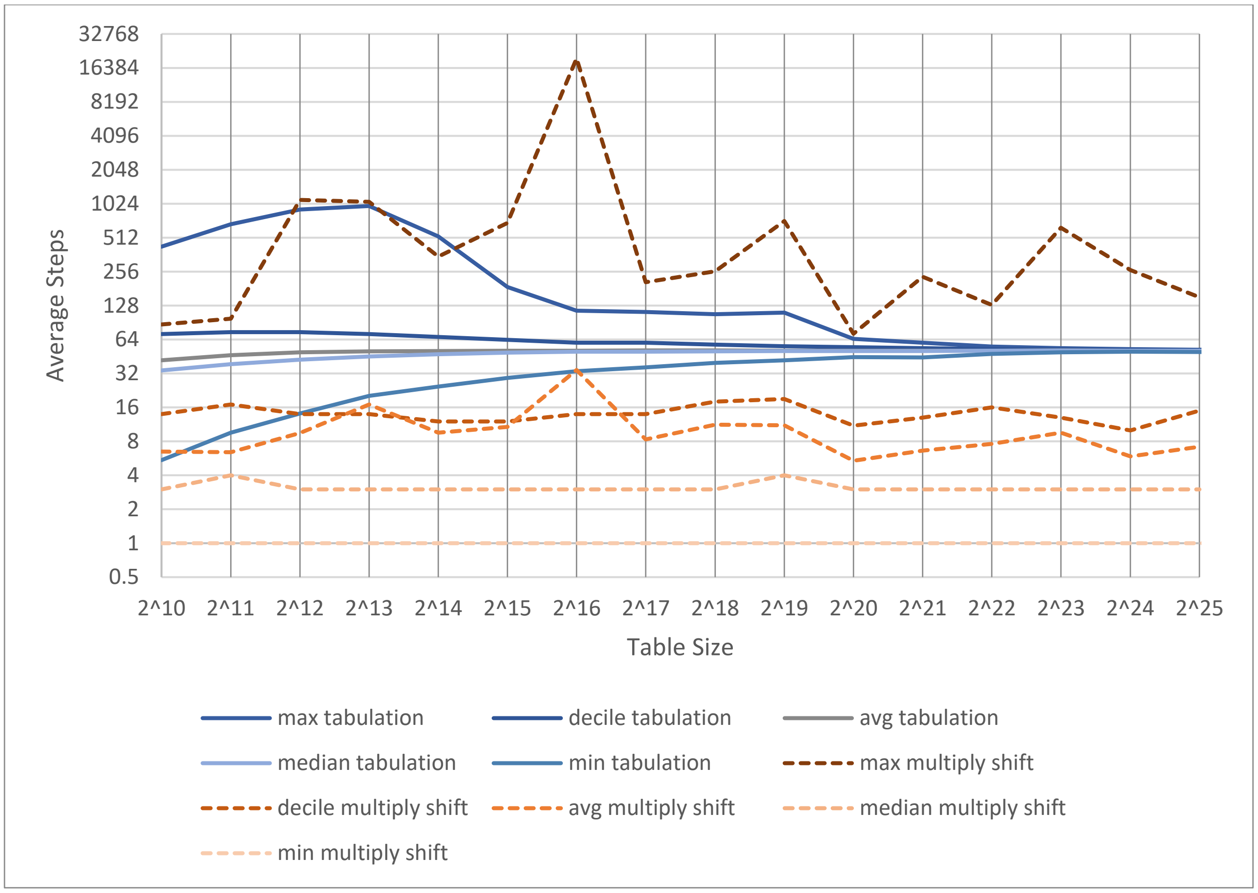


Like the previous graph, we see that the complexity of insert is not dependent on the type of hash function, but on the insert hash scheme. For the same reasons as stated above, the amount of time spent on cuckoo hashing increases substantially for load factors above 0.5, and the amount of time spent on linear probing likewise increases substantially as the load factor approaches 1. On average, tabulation hashing takes more time than multiply-shift or naïve-modulo hashing. This is because it requires more operations (read from table and xor bits) as opposed to performing one or two operations. In addition, linear probing requires less time than cuckoo hashing on average because linear probing calculates a single hash function while cuckoo hashing can require multiple hashes.

II. Sequential Case Study

The sequential test compares tabulation hashing and multiply-shift hashing systems by inserting elements sequentially (1 to n) and plotting the number of steps taken with respect to table size. Multiple test runs were averaged over each table size. Several statistical values are considered in the graph below. Solid lines indicate tabulation hashing, and dashed lines indicate multiply-shift hashing.

Figure 3: Table Size vs. Number of Steps



As the table size increases, tabulation hashing approaches an average constant value of around 50 steps per insert. It has been proven that tabulation hashing has constant complexity, which confirms the results of the graph above. In addition, the minimum and maximum values converge to the average value. This is due to there being more open slots increasing the likelihood that they are distributed uniformly across the hash table (tabulation hashing behaves just as well as totally random hashing for linear probing), and in turn this reduces the variance between inserts.

As the table size increases, multiply-shift hashing seems to perform better per insert. The average and median values are very low with ~2-8 steps per insert. However, we can see from the graph that the maximum value can get very large. It has been shown that a 2-independent hashing system like multiply-shift can perform very poorly. The probability of this happening is small (hence the averages being small) but as the load factor increases there’s a nonzero likelihood that there will be an element that will take a very large number of steps to find its slot.