Algorithms and Data Structures II Assignment 2

Raul Rodriguez Castro

2025-06-09

Consider the conjecture below. For small enough values of the load factor α, the performance of linear probing is equivalent to the performance of double hashing.

The hash Table

Consider a hash table T similar to the one presented in the lecture slides, $|T| = 13$ using open addressing. The hashing function $f(k)$ of choice is Murmur hash as it is the best function to hash 32-bit integers because it has the following properties: 1. The avalanche effect: Flipping one input bit should result in the output bit having each a 50% chance to flip. 2. Low bias: There is little to no correlation between which output bits are being flipped

For double hashing the function $h_2(k)$ was taken from the lecture slides. The hashing schema looks as follows: let $i \in \mathbb{N}$ Linear hashing $h_1(k) = (f(k) + i) \mod 13$ Double hashing $h(k) = (f(k) + i(10k + 6)) \mod 13$

To analyze if there exists a load factor such that linear and double hashing have an equivalent performance on T the following experiment was carried out.

The Sample Data

The sample set included over 100,000 strings of randomly generated customer data with the following schema: Customer Schema Index Customer Id First Name Last Name Company City Country Phone 1 Phone 2 Email Subscription Date Website

Let our data set described above be denoted as D, where each string $s \in D$ represents a record conforming to the customer schema.

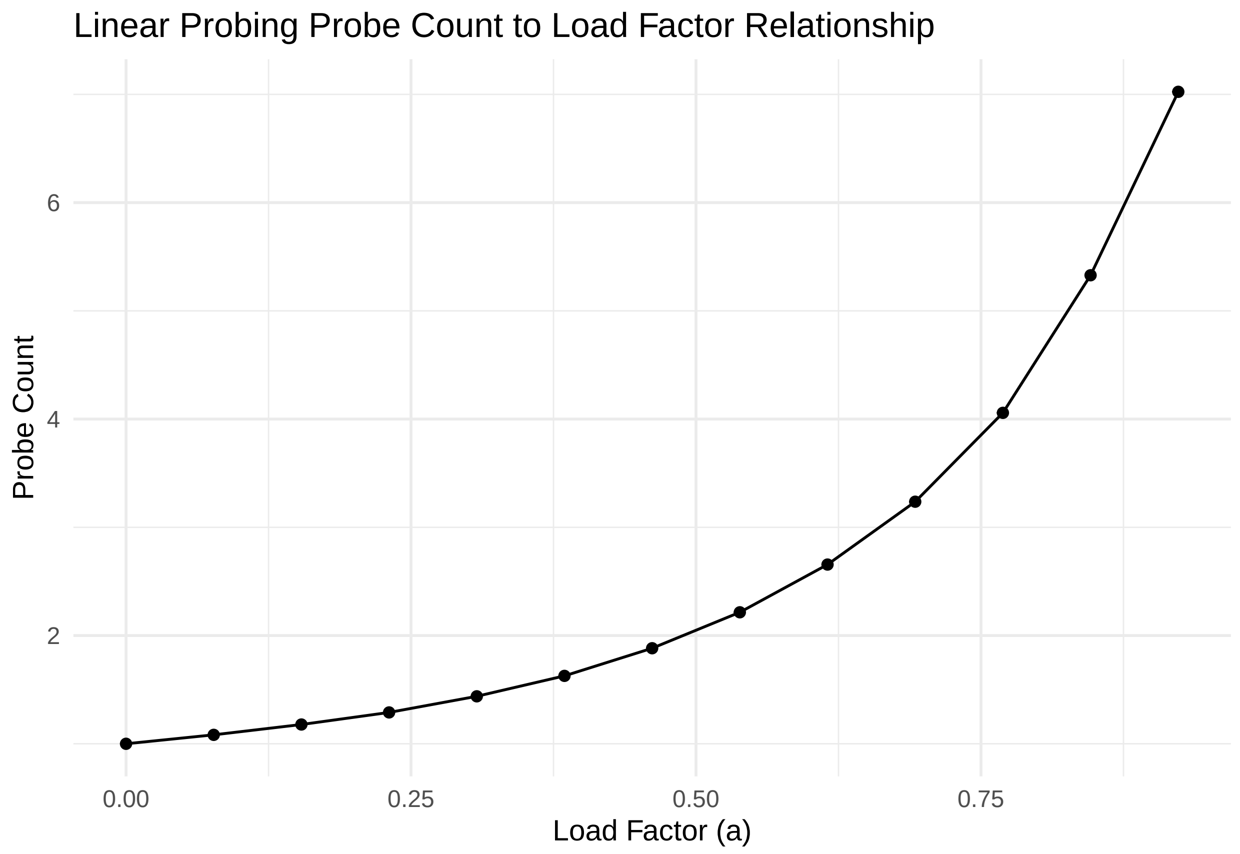
The strings were hashed by computing an integer k, $k = \Sigma s_i, 0 \leq i \leq |s|$. The resulting integer k was then fed into the hash function $h_1(k)$ (for linear probing) or $h_2(k)$ (for double hashing) to obtain the index for that string s.

Testing The Hash Table

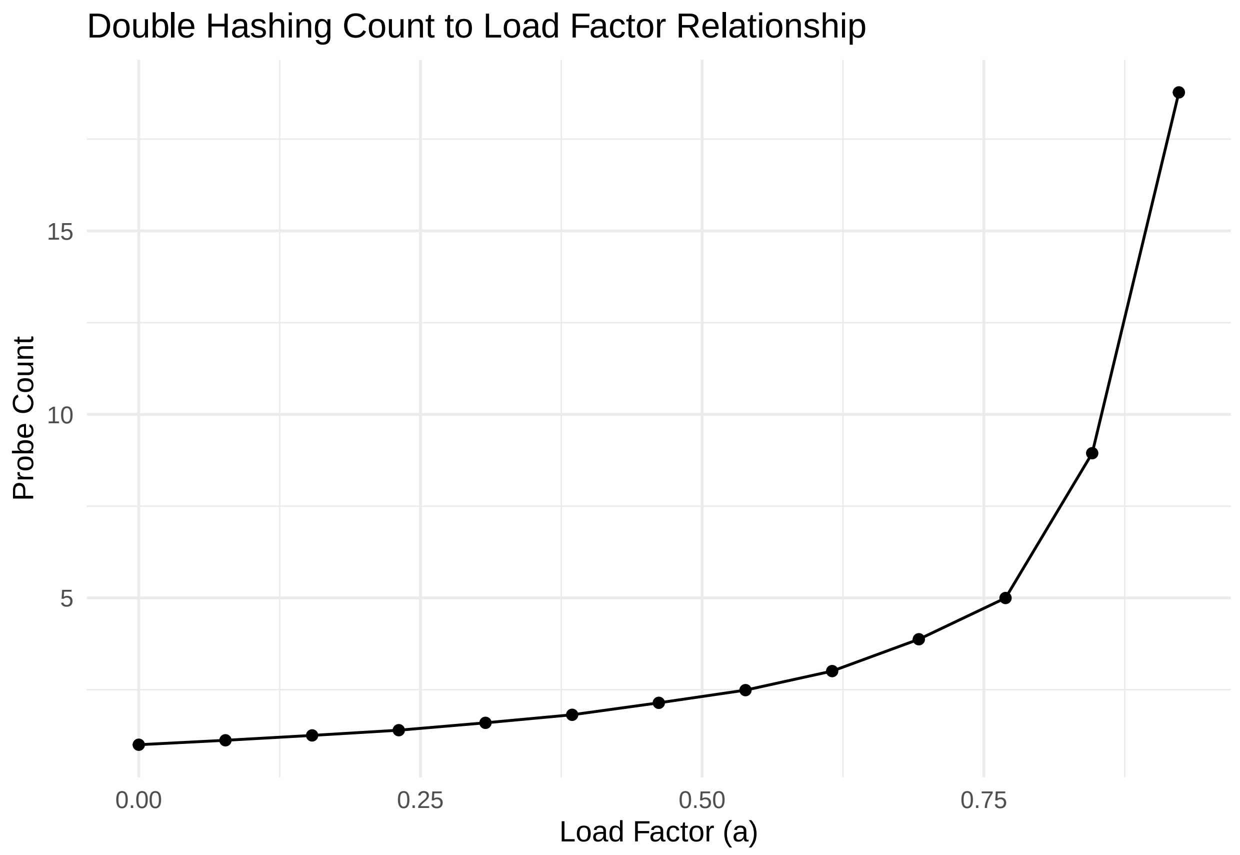
Different load factors were achieved by inserting elements into the table until it was full. This was performed with 100,000 different strings taken from the customers csv. The strings were partitioned into blocks of 13 strings each. For each of these blocks insertion was recorded along with the number of probes it took. Then for each of the α values, the average number of probes required to insert an element was computed. This calculation includes all blocks across the 100,000 strings. For every string insertion attempt into T, a record was created with the following schema: Item number: represents the i-th item being inserted into the table. Key: Contains the value that s was hashed to. Status: Either a hit (successful insertion) or miss (collision with an element at that index) Load factor (α): indicates the item to slot ratio.

Analyzing The Data

Figure 1



# Figure 2



Figures 1 and 2 suggest that the number of probes required to insert an element has an exponential growth as α approaches 1. Furthermore, linear and double hashing seem to require around 2.5 probes when α is around 0.50.

Conclusion

In conclusion, the data collected seems to show that for the particular hash table outlined in the experiment above, linear and double probing seem to be approximately equivalent in terms of probes needed to successfully insert an element to T.

Reflection

Out of 8 marks I would give question I 6-7 marks. I will first cover the strong points of part I. The hash table and its logic are clearly outlined. The experiment also focuses on a key point of testing the conjecture which is testing a wide range of alpha factors on a wide range of inputs to get a more accurate approximation of linear and double hashing. Due to time and size constraints I was unable to dive deeply into the hash functions and clustering. From 'Hash functions and the avalanche effect' I wanted to use the author's method to test whether the double hashing function was 'good' that is; whether it has a low bias and satisfies the avalanche effect property. As this could have had a big impact on the results for the required number of probes. Second, I would have liked to explore clustering distribution across both linear and double hashing to gain insight into performance degradation as T becomes more full.

References

Skeeto. (2018, July 31). Hash functions and the avalanche effect [Blog post]. Null Program. https://nullprogram.com/blog/2018/07/31/

Skeeto. (n.d.). hash-prospector: Issue #19 – Better avalanche test. GitHub. https://github.com/skeeto/hash-prospector/issues/19

Fwojcik. (n.d.). AvalancheTest.cpp. GitLab. https://gitlab.com/fwojcik/smhasher3/-/blob/main/tests/AvalancheTest.cpp?ref_type=heads