Linear Probing

M. Danish

Computer Science, CSU Global

CSC506, Design and Analysis of Algorithms

Dr. Lori Farr

01/22/2023

For this assignment, I researched the linear probing method in hashing. First, I will describe the linear probing method and the problem it solves. Then, I will discuss the performance, both in ideal conditions and in less-than-ideal conditions. Finally, I will discuss the drawbacks of linear probing and explain how they are overcome through rehashing.

Linear Probing Method in Hashing

Linear probing addresses the problem of collisions in hash tables. Values are mapped to buckets in hash tables by dividing a key by the number of elements in the hash table; then, the remainder is the bucket number. When a bucket is already occupied, however, a collision occurs. Linear probing then seeks the next empty bucket and stores the value there. It distinguishes between "empty-since-start" and "empty-after-removal," and only stops when it finds a bucket that has been empty since the start. If it can't find an empty-since-start bucket between the mapped bucket and the end of the array, it then starts over at bucket 0. Searching stops completely when it encounters an empty-since-start bucket (Lysecky, 2019; Quanrud, 2021). This is because if the key existed in the array, then logically, it would be somewhere between the mapped bucket and the first empty-since-start bucket, so if the search hits an empty-since-start bucket without finding the key, then it's safe to assume the key *isn't* in the array.

Performance Analysis

Now, let's look at the performance of linear probing. Flajolet et. al (1998, p. 2) described linear probing as, "an excellent trade-off between algorithmic simplicity and efficiency, as long as the filling ratio [of the table] is not too large, say less than ⅔ or ¾." Specifically, it has an average runtime complexity of $O(1)$ for search and insertion operations. Linear probing does not perform quite as well in full and almost-full tables, having an average runtime complexity of $O(n^{3/2})$ (Flajolet et. al, 1998).

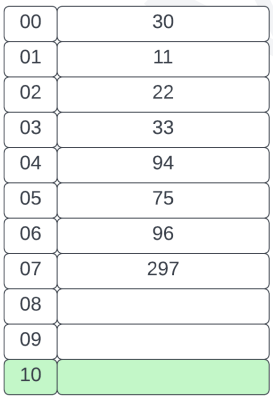
Overcoming the Drawbacks of Linear Probing Through Rehashing

Lastly, we will discuss the process of rehashing and its effect on linear probing. The main drawback of linear probing is that it causes the runtime complexity to increase as the size of the array increases. The traditional method for overcoming this drawback is rehashing, which involves creating a new, larger hash table and re-mapping all existing records to the new table. Although rehashing methods vary, spiral storage is considered the most efficient when coupled with linear probing. This is because the spiral storage method adds one bucket to the hash table at a time as the size of the array increases. Therefore, the entire table never has to be rehashed at once, and only local reorganization must occur with each addition of a new bucket (Larson, 1988).

The spiral storage method has been observed to eliminate lengthy delays in rehashing time. Total rehashing, for instance, has been observed to take 0.00055 seconds of CPU time per record. In a large array, this adds up—total rehashing of a table consisting of 20,000 records was observed at a full 11 seconds of CPU time (Larson, 1988).

Figure 1

A New Bucket Being Added to a Hash Table via the Spiral Storage Method



Note: Performance will start to degrade when the hash table reaches 75% full. A new bucket is added to the table so as to maintain a runtime complexity of $O(1)$ as new values are added and rehashing occurs (Larson, 1988).

Conclusion

The linear probing method is used to resolve collisions as new values are added to hash tables. The average runtime complexity of linear probing is $O(1)$. However, linear probing becomes less efficient when the fullness of a hash table is 75% or greater. Rehashing solves this problem by adding more buckets to the hash table and re-mapping the items in it. Although total rehashing of a hash table can also cause performance to degrade, the spiral storage method can be used to add one bucket to the hash table at a time and rehash only a percentage of the table at each iteration. When combined with spiral storage in this way, linear probing allows hash tables to maintain a runtime complexity of $O(1)$.

References

Flajolet, P., Poblete, P., & Viola, A. (1998). On the analysis of linear probing hashing. *Algorithmica. An International Journal in Computer Science*, *22*(4), 490–515. https://doi.org/10.1007/pl00009236

Larson, P.-A. (1988). Dynamic hash tables. *Communications of the ACM*, *31*(4), 446–457. https://doi.org/10.1145/42404.42410

Lysecky, R., & Vahid, F. (2019, August). Design and analysis of algorithms. In R. Lysecky, & F. Vahid, *Data structures essential: Pseudocode with python examples.* Zybooks. ISBN: 9781394012268

Quanrud, K. (2021). *Linear probing*. Purdue University. https://www.fundamentalalgorithms.com/s21/notes/linear-probing.pdf