Assignment 3: Analyzing Hashing with Chaining

Ashim Sedhain

University of the Cumberlands

MSCS-532-M20: Algorithms and Data Structures

June 14, 2025

Analyzing Hashing with Chaining

This analysis examines the time complexity and behavior of a hash table that uses chaining for collision resolution and a universal hash function to minimize clustering. The focus is on evaluating the efficiency of core operations—insertion, search, and deletion—under the assumption of simple uniform hashing. Further, the role of the load factor is discussed, along with strategies such as dynamic resizing and good hash function design to maintain optimal performance.

Expected Time Complexity

Under the simple uniform hashing assumption, each key is equally likely to be hashed into any of the m available buckets, independently of where other keys are placed. Given n total elements and m buckets, the load factor $\alpha = \frac{n}{m}$ represents the average number of elements per bucket.

For a hash table using chaining, the expected time complexities are as follows:

- Insertion: $O(1 + \alpha)$
- Search: $O(1 + \alpha)$
- Deletion: $O(1 + \alpha)$

The constant term accounts for the hash computation and bucket access, while the α term accounts for traversing a chain of items in the case of collisions. When the load factor is kept low (i.e., $\alpha < 1$), these operations approach constant time in practice.

Impact of Load Factor on Performance

The load factor α is critical in determining the efficiency of the hash table:

- When α is low (e.g., less than 1), most buckets contain at most one element, and the expected cost for search and deletion remains $O(1)$.
- When α grows (e.g., $\alpha > 1$), the likelihood of multiple elements being in the same bucket increases, causing longer chains and resulting in slower operations.

In the worst-case scenario, if all elements hash to the same bucket, the performance degenerates to linear time $O(n)$. Although this is rare with a good hash function, it illustrates the importance of managing the load factor.

Strategies to Maintain Low Load Factor and Minimize Collisions

To ensure that hash table operations remain efficient, the following strategies are commonly adopted:

1. Dynamic Resizing:

   The hash table should monitor its load factor and resize (typically doubling the number of buckets) when α exceeds a predetermined threshold (commonly 0.7–0.75). Resizing requires rehashing all existing keys, which is an $O(n)$ operation, but the amortized cost over many insertions remains $O(1)$.

2. Use of a Universal Hash Function:

   A universal hash function is a randomized family of hash functions designed to minimize the probability of collision between any two distinct keys. This reduces clustering and improves the uniform distribution of elements across buckets.

Conclusion

Hash tables with chaining and universal hashing offer efficient expected-time performance for dynamic data structures. By maintaining a low load factor and using strong hashing methods, one can achieve near-constant-time operations for insert, search, and delete. Dynamic resizing plays a key role in sustaining this performance as the number of elements grows. For critical applications, thoughtful implementation choices, such as bucket sizing and hash function selection, are essential to prevent performance degradation.