

# PA4: Hash Tables and Maps

Kevin Lei

March 31, 2024

## 1 Introduction

The purpose of this programming assignment is to implement a hash table using separate chaining, linear probing, and double hashing. In this report, we will discuss the theoretical background of the hash table, and how that compares to experimental results.

## 2 Theoretical Statement

In all three implementations, the insert function has a time complexity of  $O(1)$ , given that the hash function is good enough and the maximum load factor is reasonable. This is because the hash function is just a series of mathematical operations that can be done in constant time, which is then used to index into the hash table. The way that collisions are handled is what makes the difference between the three implementations, but still retains an average time complexity of  $O(1)$ . In the case of separate chaining, the element to be inserted is simply appended to the linked list at the index of the hash table. The size of this linked list can be assumed to be relatively small, so searching for a particular element before adding it is efficient, and done in near constant time. In the case of linear probing, the element is inserted at the next available index in the hash table. This is done by incrementing the index until an empty spot is found, which also does not require many steps proportional to the size of the hash table. However, linear probing can lead to clustering, and if the maximum load factor is chosen to be too high, the time complexity can degrade to  $O(n)$ . The issue of clustering is mostly mitigated by double hashing, which uses a second hash function to determine the step size which would otherwise be fixed in linear probing. This allows for a more even distribution of elements in the hash table, and reduces the likelihood of clustering. The time complexity of inserting an element in the hash table is also  $O(1)$ , since all that is required is to hash the key and check the index until the key is found or all possible indices are checked.

One thing to note is that the time complexity of the insert function is amortized  $O(1)$ , and the maximum load factor should be small enough to prevent the hash table from degrading to  $O(n)$ , but large enough to prevent frequent resizing of the hash table. When too many elements are present in the hash table

proportional to its capacity, the performance of the hash table degrades, and all elements must be rehashed into a larger hash table. Since this operation is costly and requires  $O(n)$  time, it is important to choose a maximum load factor that is reasonable for the application and that the insert function remains close to  $O(1)$ .

### **3 Experimental Analysis**

### **4 Conclusion**