Programming Assignment 3

Hash Tables

By

Kishlay Kumar and K Pramod Krishna

E-A 008: Data Structures and Algorithms

Prof. Aakash Tyagi

TA : Raghav Awasty

Introduction

We will implement three different versions of a hash table and study their performance.
Namely:
(1) Chaining
(2) Linear Probing
(3) Double Hashing

Theoretical Analysis

Hash tables: They are a type of data structure in which the address or the index value of the data element is generated from a hash function. That makes accessing the data faster as the index value behaves as a key for the data value. In other words, Hash table stores key-value pairs but the key is generated through a hashing function.

<u>Chaining in Hash tables</u>

Chaining is a technique that avoids collisions in hash tables. A collision occurs when two keys are hashed to the same index in a hash table. Collisions are a problem because every slot in a hash table is supposed to store a single element.

Insertion:

Amortized time complexity = O(1)

Total time complexity for N elements = O(N)

Removal:

Amortized time complexity = O(N)

Total time complexity for N elements = $O(N^2)$

Linear probing in Hash tables

It is a scheme for resolving collisions in hash tables, data structures for maintaining a collection of key–value pairs and looking up the value associated with a given key. A hash table maintains one key-value pair in each cell. When the hash function creates a collision by mapping a new key to a hash table cell that is currently filled by another key, linear probing looks for the next available space in the table and inserts the new key there.

Insertion

Amortized time complexity = O(N)

Total time complexity for N elements = $O(N^2)$

Removal

Amortized time complexity = O(N)

Total time complexity for N elements = $O(N^2)$

Hash Table - Double Hashing

Open Addressed Hash tables use the collision-resolution method of double hashing. When a collision happens, the idea behind double hashing is to apply a second hash function to the key. The benefit of double hashing is that it produces a uniform distribution of records across a hash table, making it one of the greatest methods of probing. Clusters are not produced by this method. It is one of the most efficient ways to deal with collisions.

Insertion

Amortized time complexity = O(N)

Total time complexity for N elements = $O(N^2)$

Removal

Amortized time complexity = O(N)

Total time complexity for N elements = $O(N^2)$

Experimental Setup

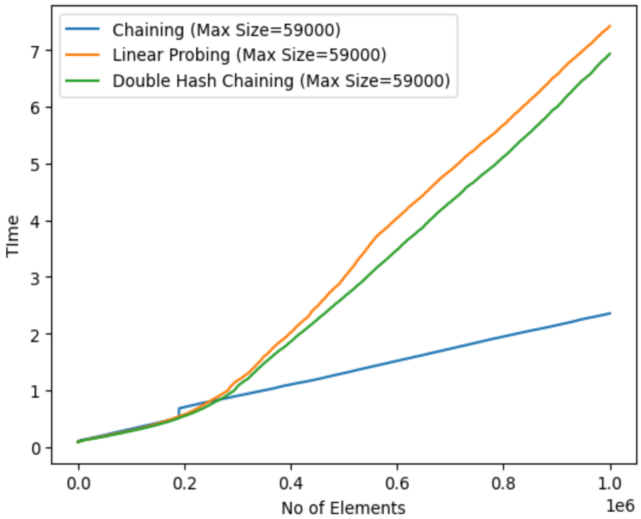
We used a machine running windows 11 OS with the following specs:

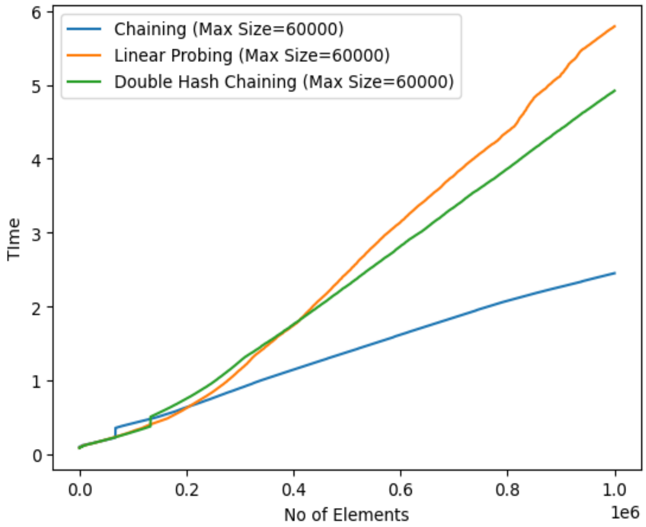
System Model	Pulse GL66 11UEK
System Type	x64-based PC
System SKU	1581.3
Processor	11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz, 2301 Mhz, 8 Core(s), 16 Logical Processor(s)
BIOS Version/Date	American Megatrends International, LLC. E1581IMS.30F, 07-12-2021
SMBIOS Version	3.3
Embedded Controller Version	255.255
BIOS Mode	UEFI
BaseBoard Manufacturer	Micro-Star International Co., Ltd.
BaseBoard Product	MS-1581
BaseBoard Version	REV:1.0
Platform Role	Mobile
Secure Boot State	On
PCR7 Configuration	Elevation Required to View
Windows Directory	C:\WINDOWS
System Directory	C:\WINDOWS\system32
Boot Device	\Device\HarddiskVolume1
Locale	United States
Hardware Abstraction Layer	Version = "10.0.22621.819"
User Name	MSI\kishl
Time Zone	India Standard Time
Installed Physical Memory (RA...	16.0 GB
Total Physical Memory	15.7 GB
Available Physical Memory	4.68 GB
Total Virtual Memory	30.0 GB
Available Virtual Memory	13.1 GB

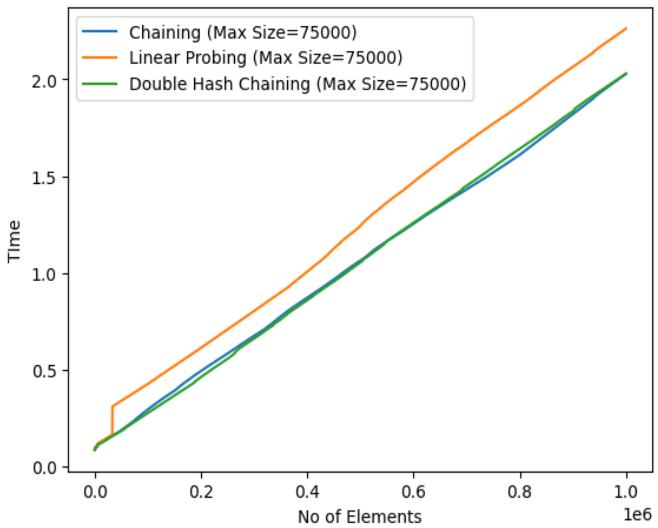
Test Inputs & Process:

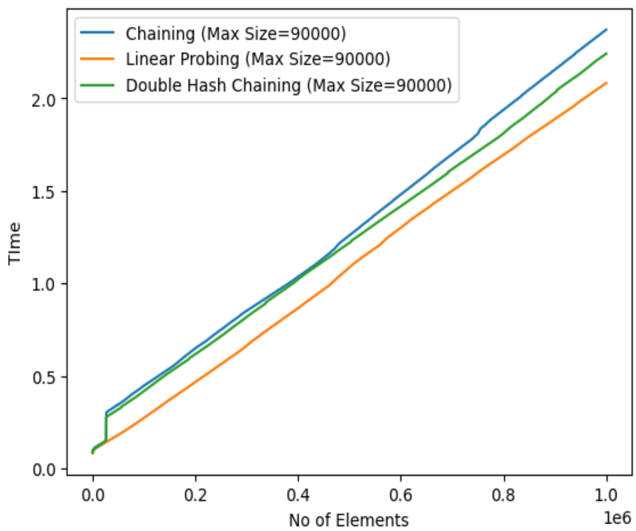
We tested the insertion times for various size of hash tables like 59000, 60000, 75000 and 90000 using the provided dictionary.txt as the input. Recorded the time for every 10 insertion operations, put the data in an array, and then visualized it using matplotlib. Following are the results of the experiment.

Experimental Results









Observations:

- The chaining implementation consistently outperformed the other two performances, as predicted by the theoretical data.
- The chaining implementation performs at O(N), as predicted by the theoretical study. However, when the capacity size was initially smaller and comparable to the number of unique key values, linear probing and double hashing were initially performed at $O(N^2)$. Still, when the capacity expanded, they also performed at O(N).
- The hash table's starting size has an impact on performance as well. The graphs above show that as capacity rises, there is less of a performance gap between linear probing and double hashing, and after a capacity of 70000, linear probing begins to outperform double hashing. This is due to the fact that when capacity rises, the number of collisions begins to decline.
- Linear probing is preferable when the hash table's capacity is high and extra space cannot be used. If not, we can employ double hashing.
- Using the chaining implementation is always preferable if space is not an issue.
- Using the chaining approach is preferable when the hash table's capacity is quite low.