Izmir Dokuz Eylul University

Department of Computer Engineering

CME 2201 - Assignment 1

DEVELOPING A SUPERMARKET MANAGEMENT SYSTEM

BY USING HASH TABLES

2021510010 - Çağrı AYDIN

Data Structure

The focus of this project is learning and getting experienced about hashing algorithms and ADT types and this project expected to see the fastest search time possible. For this purpose, we used HashDictionary and for holding the transactions we used ArrayList in the value part of HashDictionary. We used ArrayList instead of LinkedList because if we want to be fast for getting a value ArrayList is superior to LinkedList because of it is array-based structure but it will take much more time in adding or deleting a value, so we used ArrayList for better search time.

Our main values store in a HasDictionary which we use UUID as a key and a custom class we crated named CustomerInfo as a value part. In CustomerInfo class we store our customers UUID, name and their transaction. For storing their transactions, we opened a new class named Transactions which stores the date of the transaction, product name and the total transaction which that customer made. Back in Customer info class we created a ArrayList for Transactions which stores all the new transactions that customer make.

How code/methods work

->In HashDictionary class

In the constructure we want to be informed by the hash method, indexing method and load factor for building our first dictionary.

In **add** method we divided this method into two pieces one of them works only in Linear Probing and other one is only working for Double Hashing, but we get our first index at the begging of the add method because it will start at that index whenever method is chosen. In linear probing if our first index is null in HashDictionary we create a new entry, if our first index is not equal to null than we add 1 to our index and mode by table size and check again until it finds an empty space. In double hashing we first check the first index if it is null than we enter our new entry if it is not, we calculate our second function for adding into index, then we add our second function output into our index until it finds an empty space or same key value.

In **getValue** method we also divided this method into two sections for double hashing and linear probing because we don't want to iterate all through the Dictionary. In linear probing section we find our first index by using getHashIndex method if our first index's key equal to our key we want, it will return the index if it is not, it will iterate all through the dictionary until it finds the key or returns null which means that key does not exist. In double hashing every part is the same as linear probing but in double hashing we add our second function instead of 1 so we are not iterating every element in dictionary.

In **search** method we call getValue method if our return value is not equal to null it will return the value part which in this case our CustomerInfo class.

In **getHashIndex** method we also divided this method into two parts. One of them is SSF and the other one is PAF. In SSF part we sum all the ASCII values of the char at our key value and returns the sum of the values. We did a -96 because we want the alphabetical

number of the char instead of the ASII value. In PAF we decided z as 33 and calculate the sum of every char at our key value then returned it.

In **enlargeHashTable** method we created a new dictionary which is at least double the size of the previous dictionary and rounded it to the nearest prime number by using getNextPrime method. Then we add all our elements into the new dictionary by their new hash indexes with the help of our getHashIndex method.

In **isPrime and getNextPrime** methods we want to see next prime number for our new dictionary, so we add 1 if the size is even than add 2 until we find a value which is not able to divide by our divisor.

->In CustomerInfo class

In **addTransaction** method is designed to add the customers' shopping date and product in the ArrayList which is inside the CustomerInfo class. When a new shopping information is occurred for a new customer, the customer will be added in our hash table, and the transaction will be added afterwards, if a new shopping information for a customer, which is already inside the hash table, has occurred, only the transaction will be added to the Array List.

Load Factor	Hash Function	Collision Handling	Collision Count	Indexing Time	Avg. Search Time	Min. Search Time	Max. Search Time
α=50%	SSF	LP	1.2294600846E10	238.983s	0.784s	0.697s	0.856s
		DH	8.82492971E8	5.664s	0.041s	0.039s	0.045s
	PAF	LP	1.0738538E7	1.882s	0.036s	0.035s	0.042s
		DH	1966507	1.399s	0.018s	0.018s	0.023s
α=80%	SSF	LP	1.2294600846E10	240.371s	0.714s	0.678s	0.884s
		DH	8.82492971E8	5.83s	0.039s	0.034s	0.054s
	PAF	LP	1.0738538E7	1.851s	0.034s	0.032s	0.042s
		DH	1966507.0	1.46s	0.017s	0.017s	0.019s