CME2201 Data Structures Assignment

A Supermarket Management System

2022510127, Kerem Kalıntaş

2021510089, Berkay Karakuş

December 7, 2022

1. Introduction

In this assignment, we are expected to implement a simple supermarket management system using hash tables. But of course, the management system implementation is just a blanket for the main goal of this assignment.
The main goal is to measure and monitor the performances of different hashing algorithms, collision resolution strategies and load factors.
We are going to explain the logic behind the hash table we use and the relevance measure. Then we are going to interpret the results in the performance table.

2. Explanation

2.1 HashTable

Dictionary Interface

Dictionary Interface is a interface for dictionaries and has the following functions:

```
void put(KeyType key, ValueType value);

ValueType get(KeyType key);

ValueType remove(KeyType key);

void resize(int capacity);
```

HashTable implements this interface. Also the HashTable class takes two arguments in it's constructor. That being a *CollisionResolver* and a *HashFunction.* We will explain what these data types are and their importance in the HashTable performance.

HashFunction Interface

This interface has one function, and has the following signature:

```
int getIndex(String key, int capacity);
```

This interface is implemented in *HashFunctionPAF* and *HashFunctionSSF* classes. *getIndex* result will change according to implementing class.

CollisionResolver Interface

This interface has two functions, they have the following signature:

```
int locate(int index, KeyType key, HashEntry<KeyType, ValueType>[] entries);
```

```
void handleCollision(int index, HashEntry<KeyType, ValueType> newEntry,
HashEntry<KeyType, ValueType>[] entries) throws IllegalStateException;
```

locate function retrieves the keys index from the entries. Method of retrieving changes in different resolvers such as Linear Probing and Double Hashing.

handleCollision function resolve the hash collision occurred at the given index. It will put the entry in an empty location. This function will throw *IllegalStateException* on situations where hash collision cannot be solved.

2.2 Customer

This is a basic data type to store customer information and market transactions.
It has a subclass called *MarketTransaction* that implements *Comparable<MarketTransaction>.* This class holds all market transactions sorted based on their date in this subclass.

2.3 SortedList

SortedList class is a merely extension above the javas ArrayList. It has a function called *add* that adds the given element in it's sorted place.

3. Benchmark results

Load Factor	Hash Function	Collision Handling	Collision Count	Indexing Time	Avg. Search Time	Min. Search Time	Max. Search Time
α=50%	SSF	LP	98804	390431ms	764496ns	300ns	4358599ns
		DH	99419	195441ms	367331ns	300ns	2812700ns
	PAF	LP	12558	98437ms	172412ns	400ns	5439000ns
		DH	12529	139563ms	232092ns	400ns	6376800ns
α=80%	SSF	LP	129804	400880ms	737644ns	500ns	3342700ns
		DH	129877	197966ms	476368ns	300ns	5878800ns
	PAF	LP	22634	80967ms	154359ns	400ns	4473900ns
		DH	22603	98772ms	203999ns	400ns	6434300ns

4. Conclusion

We can conclude to three key points from the benchmarks

1. PAF is generally better than SSF, both in terms of number of collisions

and indexing time.

2. For SSF, double hashing is way better in terms of collisions and indexing time in contrast to linear probing. But for PAF, linear probing is better in terms of collisions, and nearly has the same indexing time performance as double hashing.