Code Documentation for HashMapPractice Project

By Rachel Hussmann

Overview

The purpose of the HashMapPractice project was to create our own hashmap and hashing algorithms from scratch. An additional goal was to experiment with real data to see where collisions occurred, how long the program took and the total memory that the Java Virtual Machine (JVM) needed for the program. The dataset I used for this part of the project is the Kaggle dataset called "Emergency – 911 calls". This data lists 911 calls from Mongomery County, PA between 2015 and 2020. The link to the dataset is here: https://www.kaggle.com/datasets/mchirico/montcoalert?resource=download

This project has nine classes.

How It Works

Book

The Book class is a simple class responsible for holding information about a Book object. The main goal for this class was to test if the generic variable for the LinkedList and the RachelSimpleHashMap classes. It has one constructor and one method:

- Book
    - Parameter: String name – The name of the book
    - Functionality – Constructor for the Book class
    - Returns: Nothing
- toString
    - Parameters: None
    - Functionality – Returns a string representative of the object. For this class, it is the name of the book.
    - Returns: String – The name of the book

EmergencyCallEntry

The EmergencyCallEntry class is responsible for taking in data from the Kaggle dataset "Emergency – 911 Calls". This class works alongside the Importer class to accept data from

a .csv file and organizes the data into multiple EmergencyCallEntry objects. It has four variables:

- String titleOfEmergency
- String timeStamp
- String twp
- String address

The EmergencyCallEntry class has one constructor and one method:

- EmergencyCallEntry
    - Parameter: String titleOfEmergency – The title of the emergency, String timeStamp – The date and time of the emergency, String twp – The township that the emergency took place in, String address – The street address of the emergency
    - Functionality – Constructor for the EmergencyCallEntry class
    - Returns: Nothing
- toString
    - Parameters: None
    - Functionality – Returns a string representative of the object. For this class, it is the timeStamp of the emergency and the title of the emergency.
    - Returns: String – The timeStamp and the title of the emergency

The class also contains four sets of getters and setters for each of the variables.

Exporter

The Exporter class is responsible for saving data into a .txt file. It has one method:

- createFile
    - Parameters: String nameOfFile – The desired name for the file, String data – The data to be saved in the file
    - Functionality – Saves inputted data to a .txt file.
    - Returns: Nothing, but prints statements

Importer

The Importer class contains methods that accept a file and extract the data from the file. Specifically for this project, it accepts .csv data of 911 calls and create the appropriate list of EmergencyCallEntry objects. The .csv file is partially processed before being given to the Importer object (unnecessary columns have been removed). The class contains two methods:

- importFile
    - Parameter: String filepath – The filepath of the file that needs to be imported
    - Functionality – Imports the file and places each line of data in an ArrayList of strings. Sends the data to trimAndProcess and returns its result.
    - Returns: ArrayList<EmergencyCallEntry> - An ArrayList of EmergencyCallEntry objects that contain data from the .csv file
- trimAndProcess
    - Parameter: ArrayList<String> data – The ArrayList of strings that need to be processed
    - Functionality – Trims the titles off of the data from the .csv file and places the data into EmergencyCallEntry objects.
    - Returns: ArrayList<EmergencyCallEntry> - An ArrayList of EmergencyCallEntry objects that holds the data from the .csv file

LinkedList<E>

The LinkedList class contains methods and inner classes that assist in the creation and management of a linked list of items. This class has the ability to handle generic cases and has an inner private class called Node. The parameter for this class is E, which is the representation of a generic object of any class. This class has seven methods:

- add
    - Parameter: E item – The generic object to be added to the list
    - Functionality – Accepts a new item and adds it to the linked list.
    - Returns: Nothing
- add
    - Parameters: E item – The generic object to be added to the list, int index – The index to add the new item at
    - Functionality – Accepts a new item and adds a new node at the given index.
    - Returns: Nothing
- remove
    - Parameters: int index – The index of the Node to be deleted
    - Functionality – Deletes the Node at the specified index.
    - Returns: Nothing
- get
    - Parameter: int index – The index of the object the user wants returned
    - Functionality – Finds the Node in the list and returns the data of the Node at that index.
    - Returns: E - The generic object at the specified index
- set

    - Parameters: int index – The index of the Node to be changed, E item – The new item to be set at the specified index
    - Functionality – Changes the data within the Node at the specified index.
    - Returns: E – The data that has been replaced
- size
    - Parameters: None
    - Functionality – Returns the current number of nodes in the linked list.
    - Returns: int – The current number of nodes in the linked list
- Iterator
    - This method is a special case because while it acts as a method, it also declares some methods within it, which will be discussed here.
    - Parameters: None
    - Functionality – Creates a way for the class to use foreach loops and other iteration-based methods.
    - Returns: Iterator<E> - The Iterator object for the LinkedList class
        - Within the return statement, some methods are declared for the iterator object.
        - hasNext
            - Parameters: None
            - Functionality – Checks to see if the current Node has an object attached to the .next pointer by checking to see if the .next object is null.
            - Returns: boolean - True if the list does have a next element, False if the list does not have a next element
        - next
            - Parameters: None
            - Functionality – Captures the data from a Node, returns it and moves down the list to the next Node.
            - Returns: E - The object within the current Node

Node<E>

The Node class is a private inner class within the LinkedList class. It is responsible for creating each collection of data for each part of the linked list. The class contains two variables and one constructor:

- E data
- Node<E> next
- Node
    - Parameter: E dataItem – The data to be held within the node

    - Functionality – Constructor for the Node class
    - Returns: Nothing

RachelSimpleHashMap<E>

The RachelSimpleHashMap contains methods that create hashes from keys and creates an array of linked lists that hold the key and data value pairs. This class has the ability to handle generic cases and has an inner private class called KeyValuePair. The parameter for this class is E, which is the representation of a generic object of any class. The class contains ten methods:

- add
    - Parameters: String key – The key value that will be hashed and used to find the item, E item – The generic object or data to be stored
    - Functionality – Adds a key and an item to the hashmap.
    - Returns: Nothing
- get
    - Parameter: String key – The key value for the desired object
    - Functionality – Returns the data associated with the key.
    - Returns: E – The generic object associated with the given key
- remove
    - Parameter: String key – The key values that will be hashed and used to find the item
    - Functionality – Removes a key value pair from the hashmap.
    - Returns: Nothing
- size
    - Parameters: None
    - Functionality – Returns the number of key value pairs added to the hashmap.
    - Returns: int – The total number of key value pairs in the hashmap
- isEmpty
    - Parameters: None
    - Functionality – Checks to see if the hashmap is empty.
    - Returns: boolean – True if the hashmap is empty, False if the hashmap is not empty
- checkNumberOfCollisions
    - Parameters: None
    - Functionality – Prints out the structure of the hashmap with the number of collisions next to it.
    - Returns: String – The structures of the hashmap with the number of collisions as a string value
- dumbHash

    - Parameter: String key – The key to be hashed
    - Functionality – Takes a key value and hashes it by returning the number of characters in the string.
    - Returns: int – The hashed key (The number of characters in the key)
- contains
    - Parameter: String key – The key value to find in the hashmap
    - Functionality – Checks to see if a key exists in the hashmap.
    - Returns: boolean – True if the key was found in the hashmap, False if the key was not found
- checkBounds
    - Parameter: int hash – The hashed key value
    - Functionality – Checks to see if the hash is larger than the size of the hashmap.
    - Returns: Nothing
- resize
    - Parameters: None
    - Functionality – Dynamically resizes the array when it needs to be expanded.
    - Returns: Nothing

KeyValuePair

The KeyValuePair class is a private inner class that is used to store the key and value in one place for the RachelSimpleHashMap class. It has two variables and one method:

- String key
- E data
- KeyValuePair
    - Parameters: String key – The key for the pair, E data – The data associated with the pair
    - Functionality – Constructor for the KeyValuePair class
    - Returns: Nothing

StopWatch

The Node class contains methods that keep track of how long it takes code to run. This class has one constructor and four methods:

- StopWatch
    - Parameters: None
    - Functionality – Creates a StopWatch object.
    - Returns: Nothing

- start
    - Parameters: None
    - Functionality – Starts the StopWatch. If the StopWatch object has already been started, it ignores the command.
    - Returns: Nothing
- stop
    - Parameters: None
    - Functionality – Stops the StopWatch. If the StopWatch object has already been stopped, it ignores the command.
    - Returns: Nothing
- getElapsedTime
    - Parameters: None
    - Functionality – Returns the amount of time the code took.
    - Returns: long – The amount of time the code took in milliseconds
- reset
    - Parameters: None
    - Functionality – Sets the StopWatch object back to the default state.
    - Returns: Nothing

HashMapTester

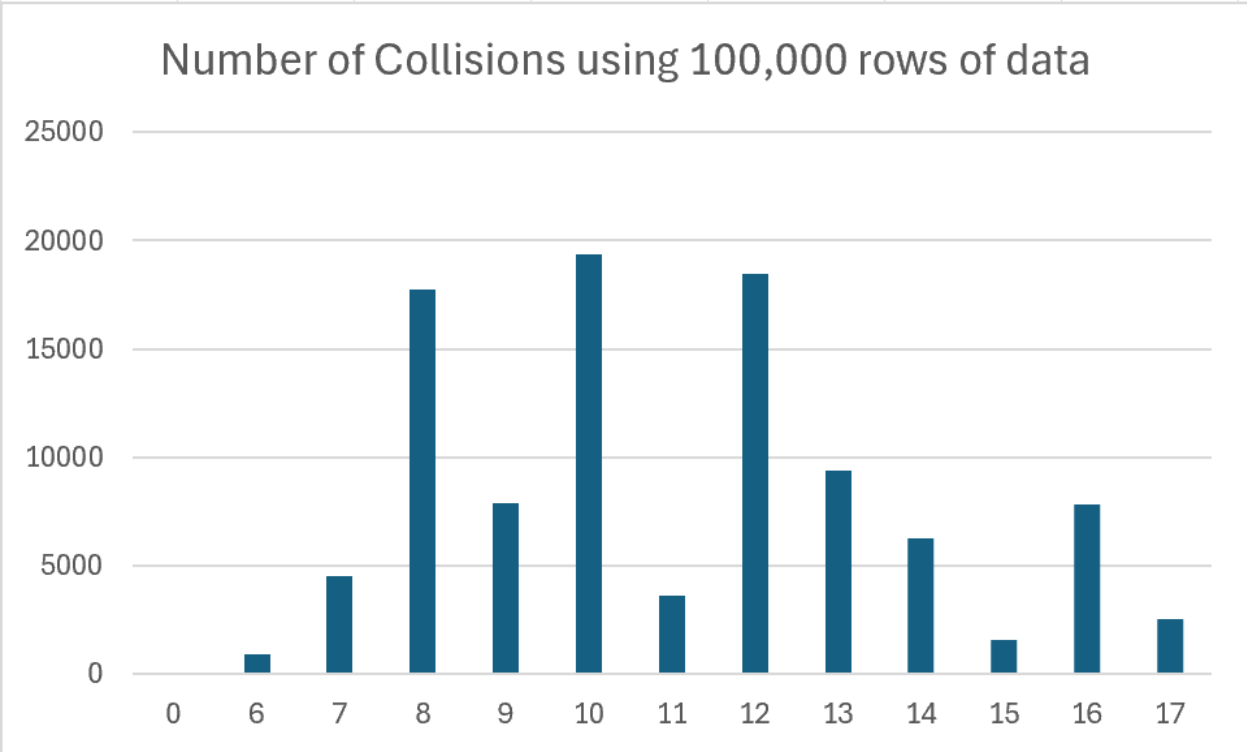
The HashMapTester class contains the main method and is used to test the methods from the RachelSimpleHashMap class.

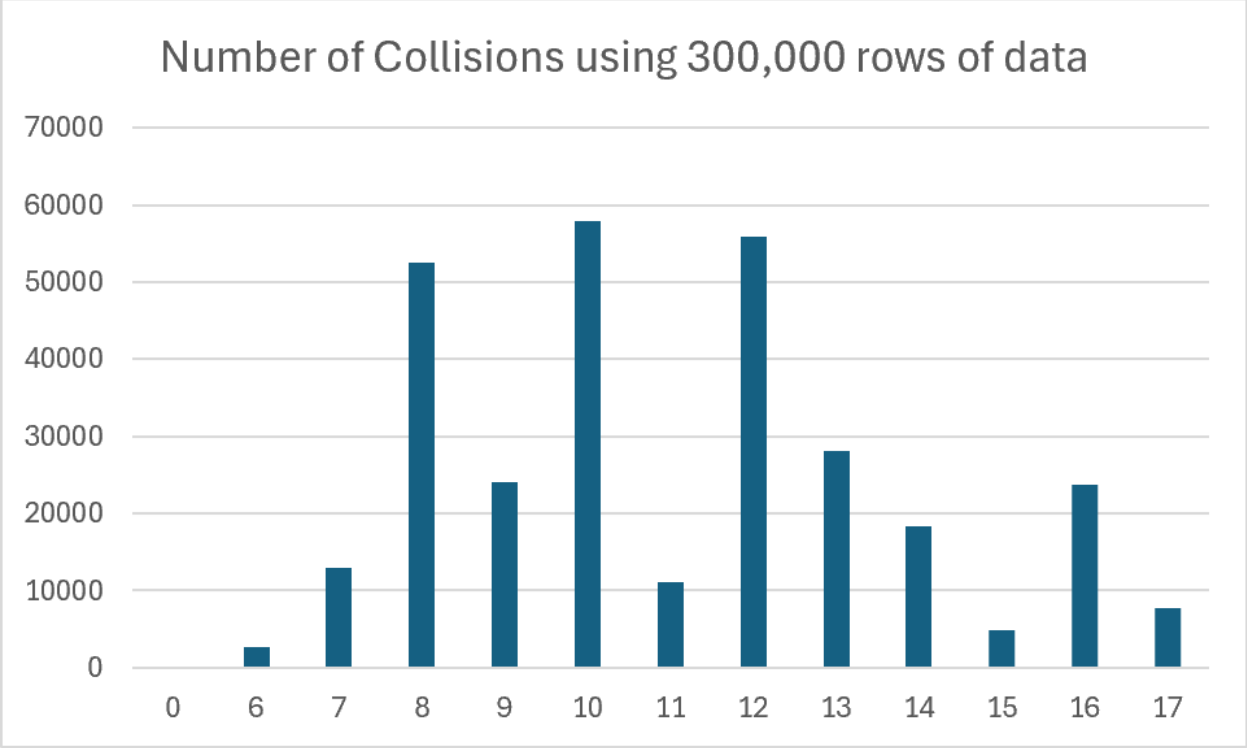
MemorySpeedTester

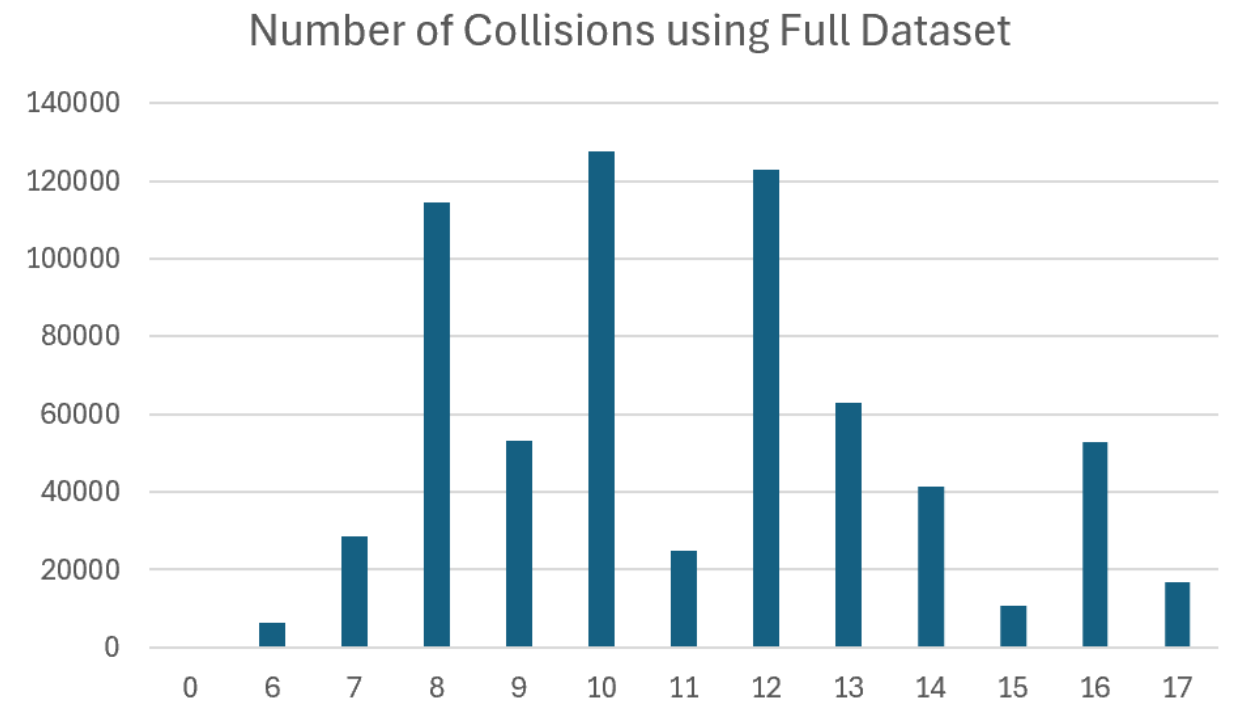
The MemorySpeedTester class contains another main method and is used to experiment with the RachelSimpleHashMap class by importing .csv data of differing sizes and exporting a .txt file containing the number of collisions, the memory used and how long the program took.

Analysis of the RachelSimpleHashMap and the 911 dataset

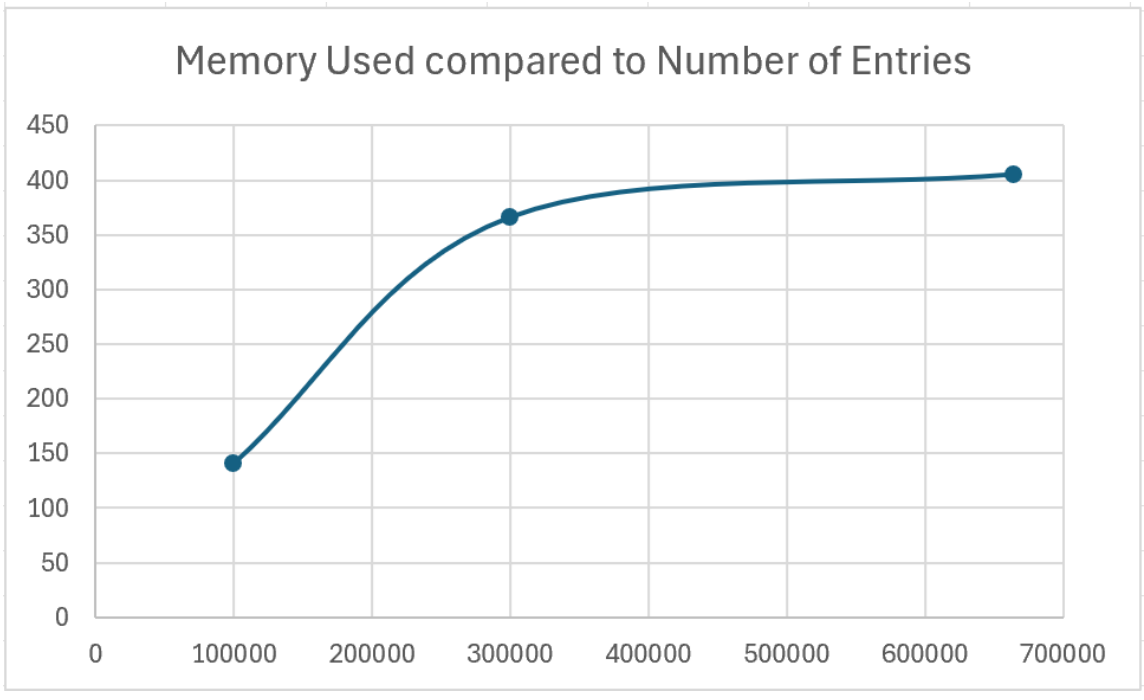
Three different versions of the same 911 data were used for this experiment. Each .csv file contained a different number of rows of data. Experiments were done on 100,000 rows of data, 300,000 rows and data and the full .csv file of data, which contained over 600,000 rows of data. The graphs below will show the collisions for each experiment, comparing the amount of memory used for each experiment and comparing the amount of time that elapsed for each experiment.

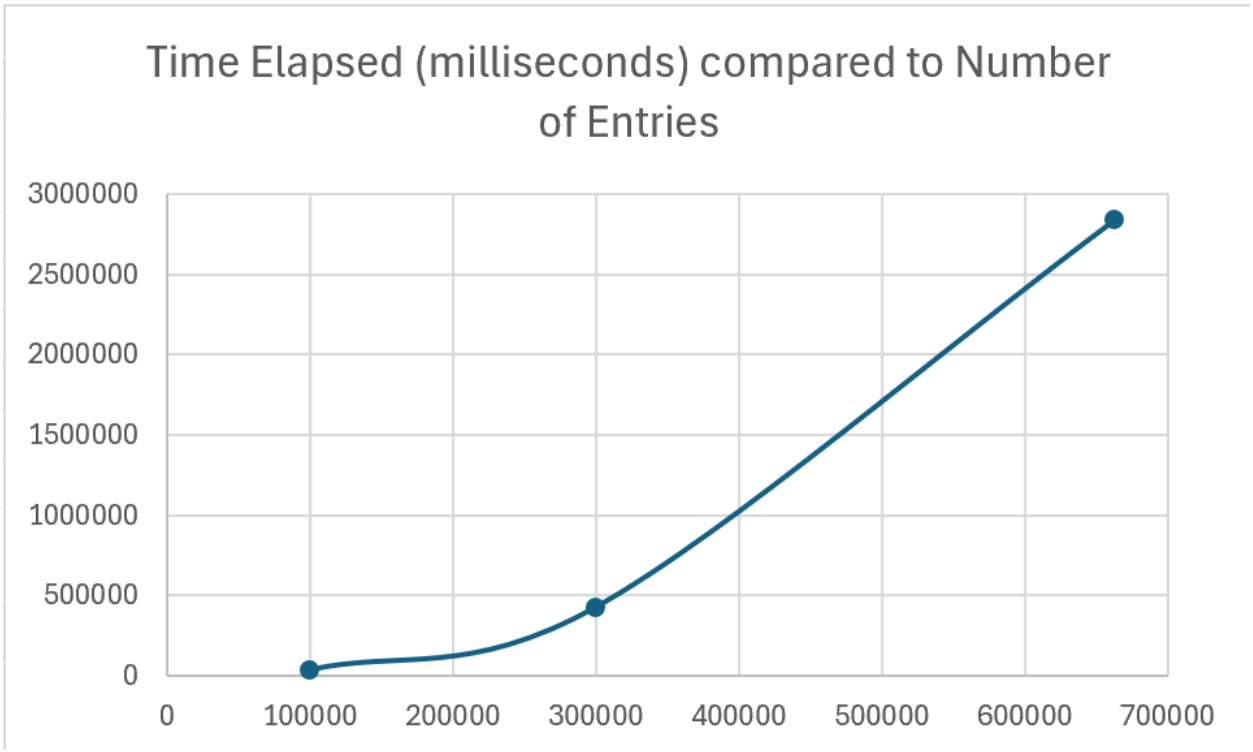

Number of Collisions using 100,000 rows of data
25000
20000
15000
10000
5000
0
0
6
7
8
9
10
11
12
13
14
15
16
17


Number of Collisions using 300,000 rows of data
70000
60000
50000
40000
30000
20000
10000
0
0
6
7
8
9
10
11
12
13
14
15
16
17



We can see from this data that the frequency of collisions across all the indices is pretty much the same, regardless of the number of entries. This tells us that collisions in large datasets are quite common, and to prevent this from happening, we would need a better hashing algorithm to prevent the collisions from happening.





As expected, as the number of entries goes up, the amount of memory needed for the JVM and the time it takes the program to run increases. Personally, the most surprising thing about these graphs is that for the full data set, it took over 45 minutes to process. In comparison, the smallest experiment only took less than a minute.

Output

Output for creating and acting on an integer RachelSimpleHashMap

```
Adding new items to the hashmap . . .

What value is associated with the key Zero: 0
What value is associated with the key One: 1
What value is associated with the key Three: 3

What is the size of the HashMap: 3

Adding more new items to the hashmap . . .

Linked List at index 3:
[One, 1]
Number of collisions: 0

Linked List at index 4:
[Zero, 0]
[Four, 4]
Number of collisions: 1

Linked List at index 5:
[Three, 3]
[Seven, 7]
Number of collisions: 1
```

```
Does the hashmap contain the key Zero before deleting: true

Deleting key Zero . . .

Does the hashmap contain the key Zero after deleting: false
```

Output for creating and acting on a Book RachelSimpleHashMap

```
Creating new hashmap . . .

Linked List at index 6:
[Austin, Pride and Prejudice]
Number of collisions: 0

Linked List at index 11:
[Shakespeare, Hamlet]
[Shakespeare, Romeo and Juliet]
Number of collisions: 1
```

Output of one of the .txt files

```
Linked List at index 0:
[, 2015-12-14 21:36:52, Fire: VEHICLE ACCIDENT]
[, 2015-12-15 11:31:36, EMS: UNKNOWN MEDICAL EMERGENCY]
[, 2015-12-24 17:30:07, Fire: VEHICLE ACCIDENT]
[, 2015-12-30 03:32:49, EMS: VEHICLE ACCIDENT]
[, 2015-12-30 03:32:28, Fire: VEHICLE ACCIDENT]
[, 2016-01-02 13:01:30, EMS: UNKNOWN MEDICAL EMERGENCY]
[, 2016-01-06 23:11:54, EMS: OVERDOSE]
[, 2016-02-20 22:17:11, EMS: SUBJECT IN PAIN]
[, 2016-03-11 14:11:44, EMS: LACERATIONS]
[, 2016-03-12 21:11:33, EMS: OVERDOSE]
[, 2016-03-13 19:55:43, EMS: UNCONSCIOUS SUBJECT]
[, 2016-03-18 18:13:07, EMS: FALL VICTIM]
[, 2016-03-19 20:46:38, EMS: NAUSEA/VOMITING]
[, 2016-03-27 12:41:42, EMS: MEDICAL ALERT ALARM]
[, 2016-03-28 01:36:42, EMS: RESPIRATORY EMERGENCY]
[, 2016-04-02 00:47:51, EMS: VEHICLE ACCIDENT]
[, 2016-04-02 00:46:53, Fire: VEHICLE ACCIDENT]
[, 2016-04-04 16:04:49, EMS: UNKNOWN MEDICAL EMERGENCY]
[, 2016-05-01 02:44:51, EMS: DIZZINESS]
[, 2016-05-10 09:11:23, EMS: UNKNOWN MEDICAL EMERGENCY]
[, 2016-05-29 07:22:13, EMS: FALL VICTIM]
[, 2016-06-01 12:37:35, EMS: SUBJECT IN PAIN]
[, 2016-06-01 16:34:27, EMS: SYNCOPAL EPISODE]
[, 2016-06-05 01:35:47, EMS: VEHICLE ACCIDENT]
[, 2016-06-14 08:56:30, Fire: WOODS/FIELD FIRE]
[, 2016-06-14 21:14:59, EMS: ALTERED MENTAL STATUS]
[, 2016-06-29 00:10:35, Traffic: HAZARDOUS ROAD CONDITIONS -]
[, 2016-07-20 06:42:37, EMS: UNCONSCIOUS SUBJECT]
[, 2016-07-20 21:26:23, Fire: GAS-ODOR/LEAK]
[, 2016-08-04 12:49:32, EMS: VEHICLE FIRE]
```

Memory and Time Strings at the Bottom of the .txt files

```
Amount of memory used: 405.12599182128906 MB
Amount of time elapsed: 2840144 milliseconds
```