CMPT-435-Assignment-3

Ahmed Handulle

November 6, 2022

In this document, I will be explaining my code from Assignment 3 in detail.

1 External Java Packages

Below is a list of external Java packages that I have used to create my program.

```java
import java.io.File; // importing file utility package to manage our file
import java.util.Scanner; // Importing the Scanner class to read text files
import java.util.ArrayList; // importing the ArrayList class to store elements
import java.util.Random; // For Random Number Calculations
import java.text.DecimalFormat; // Keeping a double number to two decimals
```

2 Main Method

```java
public class Main {
    //Converting large doubles to two decimal places
    private static final DecimalFormat twoDecimals = new DecimalFormat("0.00");
    //Initializing the size of the HashTable
    public static final int HASH_TABLE_SIZE = 250;

// Calculate the index where the item will be stored in the linked list
public static int makeHashCode(String str) {
     str = str.toUpperCase();
     int length = str.length();
     int letterTotal = 0;

     // Iterate over all letters in the string, totalling their ASCII values.
     for (int i = 0; i < length; i++) {
        char thisLetter = str.charAt(i);
        int thisValue = (int)thisLetter;
        letterTotal = letterTotal + thisValue;
     }
     // Scale letterTotal to fit in HASH_TABLE_SIZE.
     int hashCode = (letterTotal * 1) % HASH_TABLE_SIZE;  // % is the "mod" operator

     return hashCode;
    }

// Linear search function that returns the number of comparisons used to search each single item
public static int linearSearch(String targetString, String[] stringArray){
    // Initialize counting variable to store comparisons used for each item
    int comparCount = 0;
    int n = stringArray.length;
    for(int i=0; i<n; i++){
        comparCount++;
        //Compare the input string with every item in the original array and return the count if it is found other wise 0
        if(stringArray[i]==targetString)
            return comparCount;
    }
    return comparCount;
}

// Binary search function that returns the number of comparisons
public static int binarySearch(String targetString, String[] stringArray){
    // Initialize a variable for counting the comparisons
    int comparCount = 0;
    int leftIndex = 0;
    int rightIndex = stringArray.length - 1;
    //Check if there are any elements in the array that has not been checked yet
    while(leftIndex<=rightIndex){
        comparCount++;
        //Calculate the middle index of the array
        int midVal = leftIndex + (rightIndex - leftIndex) / 2;
        //Check if the element at the middle index is that same as the input element
        if (stringArray[midVal] == targetString)
            //If true, just return the number of comparisons used to find that element
            return comparCount;
        comparCount++;
        // Check if the element in the middle of the array is the less that or greater than the input element
        if (stringArray[midVal].compareTo(targetString)>0)
            rightIndex = midVal - 1;
        else
            leftIndex = midVal + 1;
        }
    return comparCount;
}

// This method calculates the average number of comparisons used for each search
public static double average(int[] array){
```

```java
    int sum = 0;
    //Iterate over all the comparison values in the array and add them together
    for (Integer i : array){
        sum += i;
    }
    // Save the result in double format
    double avg = sum/array.length;
    return avg;
}

// Driver Main method to drive the above code

public static void main(String[] args) {
    // Creating an ArrayList object to store lines of strings
    ArrayList<String> strArry = new ArrayList<String>();

    try {
        // creating a new file object
        File f = new File("magicitems.txt");
        // creating a Scanner Object
        Scanner myReader = new Scanner(f);
        // reading each line and removing the spaces and making it all lowercase
        while (myReader.hasNextLine()) {
            String linee = myReader.nextLine();
            strArry.add(linee.replaceAll("[^A-Za-z]", "").toLowerCase());
        }
        // closing scanner
        myReader.close();
        // Now catching to see if any errors exist in processing this program file
    } catch (Exception e) {
        e.getStackTrace();
    }

    // Converting the String ArrayList to String Array of the same length
    String[] arr = new String[strArry.size()];
    for (int i = 0; i < strArry.size(); i++) {
        arr[i] = strArry.get(i);
    }

    // Sorting the String Array using MergeSort by Calling the Mergesort class
    Mergesort.mergeSort(arr);

    // Picking 42 items randomly from the array of magic items and then storing them in a new array
    int n = 42;
    String[] pickeItems = new String[n];
    // Revisit Fisher-Yates shuffling technique
    Random ran = new Random();
    for(int i=0; i<n; i++){
        int randomIndex = i + ran.nextInt(arr.length-i);
        //Swap the item at position (random index) with the item at
```

```
         the ith index in the array (arr)
            String temp = arr[randomIndex];
            arr[randomIndex]=arr[i];
            arr[i]=temp;
            //Store the randomly picked item in the new array
            pickeItems[i]=temp;
        }

        //resorting the original array (arr) to use it for other operations in the program
        Mergesort.mergeSort(arr);
        // Storing the number of comparisons used for each search item in the following 2 arrays respectively
        int[] linearSearchCompArray = new int[pickeItems.length];
        int[] binarySearchCompArray = new int[pickeItems.length];

        // Searchin and storing the comparison results in above defined the arrays
        for (int i=0; i<pickeItems.length; i++){
            //Computer the number of comparisons used for each element that was randomly picked
            int linearResult = linearSearch(pickeItems[i], arr);
            int binaryResult = binarySearch(pickeItems[i], arr);
            linearSearchCompArray[i]=linearResult;
            binarySearchCompArray[i]=binaryResult;
        }

        // Print the number of comparisons used for each linear search and the average
        for(Integer i: linearSearchCompArray)
            System.out.print(i+" ");
        System.out.println(" ");
        //Print the average comparison used by Linear searching for 42 item
        double averageLinearResult = average(linearSearchCompArray);
        System.out.println("Average number of Linear Search Comparisons (42 items): " + twoDecimals.format(averageLinearResult));
        System.out.println(" ");

        // Print the number of comparisons used for each binary search and the average
        for(Integer i: binarySearchCompArray)
            System.out.print(i+" ");
        System.out.println(" ");
        //Print the average comparison used by Binary searching for 42 item
        double averageBinaryResult = average(binarySearchCompArray);
        System.out.println("Average number of Binary Search Comparisons (42 items): " + twoDecimals.format(averageBinaryResult));
        System.out.println(" ");


        // Creating the Hashtable with chaining using the Linked list class I have already created some changes
        Linkedlist[] hashtable = new Linkedlist[HASH_TABLE_SIZE];

        //Representing each HashTable index as a linked list object
```

```java
    for(int i=0; i<HASH_TABLE_SIZE; i++){
        hashtable[i] = new Linkedlist();
    }

    // Loading magicitems into the hashtable
    for(int i=0; i<arr.length; i++){
        String key = arr[i];
        int index = makeHashCode(key);
        //Put fuction is implemented in the linked link class and it inserts elements in their corresponding positions in the hashtable
        hashtable[index].put(key);
    }

    // Retreiving the 42 picked items from the hash table and
    //the number of comparisons used for each item will be stored in the below array
    int[] hashComparisonCountArray = new int[pickeItems.length];

    System.out.println("Retrieving the 42 items from the HashTable: ");
    System.out.println(" ");
    for (int i=0; i<pickeItems.length; i++){
        int hashCodeIndex = makeHashCode(pickeItems[i]);
        //The get function is implemented in the linkedlist class and it retrieves the input element from the HashTable
        int comparisonsUsed = hashtable[hashCodeIndex].get(pickeItems[i]);
        hashComparisonCountArray[i]=comparisonsUsed;
        System.out.println(pickeItems[i] + ": " + comparisonsUsed);
    }
    System.out.println(" ");

    // Calculating the Average comparions used to search the 42 items in hashgtable
    double averageHashtableComparisons = average(hashComparisonCountArray);
    System.out.println("Average number of Binary Search Comparisons (42 items): " + twoDecimals.format(averageHashtableComparisons));

}
}

//----------------------The End of the Assignment-----------


// Reference to the LinkedLIst Class that I am using

public class Linkedlist {

    //Node class for storing single elements
    static class Node {
        String data;
        Node next;
        Node(String val){
            this.data = val;
            this.next = null;
        }
    }
//instance variables of the linkedlist

    Node head;
    Node tail;
    int size = 0;
/**
 * this Function returns the length of the linked List
 */
    public int len(){
        return this.size;
    }
/**
 * This function returns a boolean value
 * and checks whether the the Linkedlist is empty or not
 */
    public boolean isEmpty(){
        if (len()==0){
            return true;
        } else {
            return false;
        }
    }
/**
 * Now we are creating a Funtion to put elements into the hashtable using linkedlist object
 */
    public void put(String key){
        Node newNode = new Node(key);

        if (isEmpty()){
            this.head = newNode;
        } else {
            this.tail.next= newNode;
        }
        this.tail = newNode;
        this.size++;
    }
    // Performing search by returning the number of comparison of the searched element
    public int get(String findKey){
        Node currentNode = this.head;
        int comparisonCount = 0;
        while(currentNode!=null){
            comparisonCount++;
            if(currentNode.data == findKey){
                return comparisonCount;
            }
            currentNode = currentNode.next;
        }
        return comparisonCount;
    }
}

// Reference to The Mergesort Class and method I am using
```

```java

public class Mergesort {
    //This method gets called when sorting
    public static void mergeSort(String[] str) {
        mergesort(str, 0, str.length-1);
    }
    //Actual merge sort method
    public static void mergesort(String[] str, int leftEndIndex, int rightEndIndex) {

        if(leftEndIndex<rightEndIndex){
            //Calcute the mid index of the input array
            int mid = (leftEndIndex+rightEndIndex)/2;
            //Recursively calling by itself to break the arry into smaller sub array until size of 1
            mergesort(str, leftEndIndex, mid);
            mergesort(str, mid+1, rightEndIndex);
            //Call the merge method
            merge(str, leftEndIndex, mid, rightEndIndex);
        }
    }
    // This method merges the sub arrays by comparing and sorting
    private static void merge(String[] str, int leftEndIndex, int mid, int rightEndIndex) {

        //Using temprorary array to perfom the merging
        String[] tempArray = new String[rightEndIndex+1];
        int i = leftEndIndex;
        int j = mid+1;
        int tempArrayIndex = leftEndIndex;

        //Sorting two sub arrays in ascending by comparing the elements of each crosponding indicis
        while((i<= mid) && (j<=rightEndIndex)){
            if(str[i].compareTo(str[j])<0){
                tempArray[tempArrayIndex]= str[i];
                i++;
            } else {
                tempArray[tempArrayIndex]= str[j];
                j++;
            }
            tempArrayIndex++;
        }
        //Check if there are elements left in the left sub array, if so compare them with one another and sort in ascending order
        while (i<= mid){
            tempArray[tempArrayIndex]= str[i];
            i++;
            tempArrayIndex++;
        }
        // Check if there are elements left int he right sub array, if so compare them with one another and sort in ascending order
        while (j<=rightEndIndex) {
            tempArray[tempArrayIndex]= str[j];
            j++;
            tempArrayIndex++;
        }
        // Copy the elemets back to their original array
        for(int x=leftEndIndex; x <rightEndIndex+1; x++){
            str[x]=tempArray[x];
        }
    }
}

//-------------------Go to the Next Page for Results---------------
```

3 Results

Randomly picked 42 items from the array of magicitems and Calculated the number of comparison used to search each item linearly. The values are Stored in the following array:

[352 139 532 514 271 344 219 321 561 539 374 427 132 399 58 589 110 258 566 178 484 203 490 97 377 597 241 665 487 645 90 147 633 280 263 55 118 125 180 470 234 84]

The Average of the above array is : [329.00]

The asymptotic running time of the Linear search is big O(n) because you might have to iterate over all the elements in the input until (n-1) to find the match

---

Binary search was then performed on all 42 randoly picked elements and their comparisons were calculated and stored in the below array:

[19 13 15 19 17 17 15 19 19 17 7 17 17 19 19 17 15 19 19 15 13 19 17 19 17 19 17 17 17 19 15 15 17 11 19 19 13 17 19 15 17 17]

The Average of the above array is : [16.00]

The asymptotic running time of the Binary search is big O(logn) because the array is split in half each time we make a comparison until we reach an array of size one and the element is not there.

Printing the number of (get + comparisons) for each item and computing the overall average to two decimal places:

mace: 1 cloakofelvenkind: 1 skullcap: 4 shadowskillarmor: 6 gosherwhipoflifeessencetransferral: 3 lightningtotem: 4 eyestalk: 2 incenseofmeditation: 2 stoneofcontrollingearthelementals: 3 snappingpurse: 2 manualofquicknessofaction: 5 pipesofthesewers: 2 clevershot: 1 necklaceoffireballstypev: 2 beltofgiantstrength: 1 thecircletofzahnlok: 4 caneofevocation: 3 glovesofarrowsnaring: 2 sustainingspoon: 3 diordroid: 2 robottobor: 2 elixirofhiding: 1 rodoftherustmonster: 4 bracersofarmor: 6 marvelouspigments: 2 thethainsoulring: 2 foldingcatapultorballista: 3 yerobeofuselessthings: 4 rodofgolemcasting: 5 wardingstakes: 2 bracersofarcherygreater: 1 cloakoftheundead: 1 universalsolvent: 2 hammerofpowerdampening: 3 glovesofswimmingandclimbing: 1 bedrollofcomfort: 1 censerofcontrollingairelementals: 1 circleofblastingmajor: 3 discofillumination: 1 ringoftheeightskills: 2 firejavelin: 4 bottleofair: 1

The overall average of searching all 42 items in the HashTable is: [2.00]

The asymptotic running time of the HashTable is as follows:

Searching (get()) = O(1)+ Load Factor (number of items/size of the list)

Insertion (put()) = O(1) because we are using a linked list to insert elements at the front of the list

Handling collision = O(1) since we are adding elements at the front of the chain (linkedlist) when a collision happens