Lab109

Steven Glasford

4-5-2019

1 Client.java

```
/**
 * A main controller class manipulating the fuck out of this bitching place.
 * @author  Steven Glasford
 * @version 4-8-2019
 */
//used for manipulating files and shit
import java.io.File;
//incase the file is not found
import java.io.FileNotFoundException;
//good for looking at data imported from a file...bitch.
import java.util.Scanner;
//used for changing the number formating
import java.text.NumberFormat;
import java.util.Locale;

public class Client {

    /**
     * @param args No command line arguments; bitch.
     */
    public static void main(String[] args) throws FileNotFoundException {
        //make a 2d array to store the data in so you can just shit your
        //data into an array table or whatever the fuck.
        //the first slot will contain the alpha value, the second slot will
        //contain the total number of collisions, and the third slot
        //will contain the max number of collisions at any particular point.
        //and we will run the program between 2 and 21 for each hash method.
        //the motherfucking will contain the information produced by the
        //polynomialHashCode, and the array shitass will contain the
        //information produced by madCompression
        int[][] motherfucking = new int[15][3];
        int[][] shitass = new int[15][3];

        //a list of prime numbers to use for the madCompression method,
        //this will make the program much faster than determining a new
        //prime number
        //the first 15 prime numbers after 45402 (the number of items in the
        //file)
        int[] primes = new int[shitass.length];


        //open the motherfucking file containing the fucking words
        File queef = new File("/home/steven/NetBeansProjects/"
                + "Lab109-GlasfordSR/src/words.txt");

        //kill the program if the file does not exist, put something more
        //interesting later perhaps
        if (!queef.isFile()){
```

```
    System.out.println("I am so sorry but the file you provided"
            + "does not exist bitch face, enter something else.");
    return;
}
//create a singlyLinkedList that will contain all of the words
SinglyLinkedList vagina = new SinglyLinkedList();

//create a scanner class so it is easier to save a the data into the
//SinglyLinkedList, will throw a file not found exception if the file
//does not exist
Scanner penis = new Scanner(queef);

//read in every word in penis and save them into the vagina.
while (penis.hasNext()){
    //add the injected matter at the end
    vagina.addLast(penis.next());
}

//use the int alpha, because why not, this loop will go through each
//of the tests and save the pertinent data into the motherfucking array
for (int alpha = 0; alpha < motherfucking.length; alpha++){
    //create a new table containing all of the hash values
    SinglyLinkedList table = new SinglyLinkedList();
    //this will help to determine if a value is unique
    boolean tripWire = false;
    //this is the total size of unique entrants
    int size = 0;

    //this for loop will go through the vagina table and calculate if a
    //hash value is unique for every part of the entrants in vagina,
    //if it is unique it will add it to a new list of table, and if not
    //it will find the repeated hash and add 1 to its value in the
    //entrants key value pair.
    for (int i = 0; i < vagina.size(); i++){
        //save the data entry temporarily after figuring out the
        //polynomialHashCode
        MapEntry shitHead = new MapEntry(polynomialHashCode((String)
                //increase the value of alpha by two since it cannot be
                //0 or 1
                vagina.first(),alpha + 30),0);
        //rotate the vagina list after getting the hashValue
        vagina.addLast(vagina.removeFirst());
        //rotate the vagina so you can keep using it over and
        //over again

        //go through the table to see if the entry is contained in the
        //table, if it is unique add it to the end of the table.
        for (int j = 0; j < table.size(); j++){
            //create a new temporary MapEntry surface so you can
            //alter the piece of pissing garbage
            MapEntry wrist = (MapEntry) table.removeFirst();
            //add the number of foundances to the value key if
            //encountered
            if (wrist.getKey() == shitHead.getKey()){
                //increase the value by 1 if the same key is found
                wrist.setValue(wrist.getValue() + 1);
                //add the piece of shit to the end of the table if
                //it is found
                table.addLast(wrist);
                //set the tripWire to true, so you know to not add the
                //fucker to the list
```

```
                tripWire = true;
            }
            //rotate and check the next entry in the list
            else
                table.addLast(wrist);
        }

        //if the tripWire is not tripped then you can be assured that
        //the entry is uniquer and you can add it to the end of
        //the table.
        if (!tripWire){
            //add shitHead to the end of the table
            table.addLast(shitHead);
            //increase the size by one
            size++;
        }
        //reset the tripWire after you add it to the fucker
        tripWire = false;
        //delete shitHead after you are finished with it.
        shitHead = null;
    }
    //temporarily store the size of the table
    int jizz = table.size();

    //store the size of the alpha value in the motherfucking array
    motherfucking[alpha][0] = alpha + 30;

    //go through the table to get valuable information
    for (int i = 0; i < jizz; i++){
        //temporarily store the data of the first entrant in the
        //table into a manipulated variable, as well as reduce the
        //size of the table by one by using removeFirst()
        MapEntry dildo = (MapEntry) table.removeFirst();
        //get the value stored in the temporary variable and add it to
        //the total number of collisions variable, remember the
        //second entrant in the motherfucking array contains the total
        //number of collisions.
        motherfucking[alpha][1] += dildo.getValue();

        //if the value at the temporary variable is greater than
        //the variable in the max collision part of the
        //motherfucking array
        if (dildo.getValue() > motherfucking[alpha][2])
            motherfucking[alpha][2] = dildo.getValue();

    }
}

//add a quotation mark at the very begining of the run, as well as the
//end so the latex will understand the output as a text and not code
System.out.println("/*");

System.out.println("This table contains the information about"
        + "\nthe number of collisions and the number used for alpha.");

System.out.println(asciiPenis(motherfucking,"Polynomial Hash Code",
        "Alpha Number", "Total collisions","Max Collisons"));

//quickly find all of the primes you are going to test
//the following is a temporary int, that stores the size of the
//primes or something, needs to be at least 5 more than number of
```

```
//imported words
int testicles = vagina.size() + 5;
//find the next 15 or something primes after the the size of
//your vagina
for (int i = 0; i < primes.length; i++){
    //save the primes into the array, and find the next
    primes[i] = findNextPrime(testicles);
    //find the next prime that is at least 5 more than the last
    testicles = primes[i] + 5;
}

//use the int alpha, because why not, this loop will go through each
//of the tests and save the pertinent data into the motherfucking array
for (int alpha = 0; alpha < shitass.length; alpha++){
    //create a new table containing all of the hash values
    SinglyLinkedList table = new SinglyLinkedList();
    //this will help to determine if a value is unique
    boolean tripWire = false;
    //this is the total size of unique entrants
    int size = 0;

    //this for loop will go through the vagina table and calculate if a
    //hash value is unique for every part of the entrants in vagina,
    //if it is unique it will add it to a new list of table, and if not
    //it will find the repeated hash and add 1 to its value in the
    //entrants key value pair.
    for (int i = 0; i < vagina.size(); i++){
        //save the data entry temporarily after figuring out the
        //polynomialHashCode
        MapEntry shitHead = new MapEntry(
                madCompression(polynomialHashCode((String)
                //use an alpha value of 41, since it doesn't give any
                //collisions, change the number for p, using the
                //prime array, use 69 for a (because it needs to
                vagina.first(),41),vagina.size(), primes[alpha],
                69, 420), 0);
        //rotate the vagina list after getting the hashValue
        vagina.addLast(vagina.removeFirst());
        //rotate the vagina so you can keep using it over
        //and over again

        //go through the table to see if the entry is contained in the
        //table, if it is unique add it to the end of the table.
        for (int j = 0; j < table.size(); j++){
            //create a new temporary MapEntry surface so you can
            //alter the piece of pissing garbage
            MapEntry wrist = (MapEntry) table.removeFirst();
            //add the number of foundances to the value key
            //if encountered
            if (wrist.getKey() == shitHead.getKey()){
                //increase the value by 1 if the same key is found
                wrist.setValue(wrist.getValue() + 1);
                //add the piece of shit to the end of the table
                //if it is found
                table.addLast(wrist);
                //set the tripWire to true, so you know to not add the
                //fucker to the list
                tripWire = true;
            }
            //rotate and check the next entry in the list
            else
```

```
                    table.addLast(wrist);
            }

            //if the tripWire is not tripped then you can be assured that
            //the entry is uniquer and you can add it to the end of
            //the table.
            if (!tripWire){
                //add shitHead to the end of the table
                table.addLast(shitHead);
                //increase the size by one
                size++;
            }
            //reset the tripWire after you add it to the fucker
            tripWire = false;
            //delete shitHead after you are finished with it.
            shitHead = null;
        }
        //temporarily store the size of the table
        int jizz = table.size();

        //store the prime number used in the first slot in the shitass
        //array
        shitass[alpha][0] = primes[alpha];

        //go through the table to get valuable information
        for (int i = 0; i < jizz; i++){
            //temporarily store the data of the first entrant in the
            //table into a manipulated variable, as well as reduce the
            //size of the table by one by using removeFirst()
            MapEntry bukkake = (MapEntry) table.removeFirst();
            //get the value stored in the temporary variable and add it to
            //the total number of collisions variable, remember the
            //second entrant in the shitass array contains the total
            //number of collisions.
            shitass[alpha][1] += bukkake.getValue();

            //if the value at the temporary variable is greater than
            //the variable in the max collision part of the
            //motherfucking array
            if (bukkake.getValue() > shitass[alpha][2])
                shitass[alpha][2] = bukkake.getValue();
        }
    }

    System.out.println("The following table contains data from the "
            + "\nrunning of madCompression method, and the number"
            + "\nused for the prime variable.");

    //print out the madCompression table
    System.out.println(asciiPenis(shitass,"MAD Compression","Prime "
            + "Number", "Total collisions","Max Collisons"));

    //this final part is for the latex compiler, so it will produce a
    //comment region for the output, instead of trying to convert it to
    //code format, I took a dump on my neighbors lawn when I was five,
    //and the neighbor thought his dog had canine AIDS.
    System.out.println("*/");
}

/**
 * Produces a hash code using the polynomial hashing function as
```

```
 * described in the book on page 413.
 * @param keyhole   The key you want to hash.
 * @param a         The number to use for the polynomial value, bitch.
 * @return          The hashed value...bitch.
 */
public static int polynomialHashCode(String keyhole, int a){
    //this will eventually become the hashcode
    long clitoris = 0;
    for (int i = 0; i < keyhole.length(); i++){
        //this is the variant given in class
        //clitoris += ((keyhole.charAt(i) * Math.pow(a, i)));

        //this is the variant given in the book, this gives much less
        //
        clitoris = (keyhole.charAt(i) + a * clitoris);
    }

    //cast to an int, we don't care if there is loss of extended data,
    //we just care that its pretty unique
    return  Math.abs((int) clitoris);
}

/**
 * Compress a hash code using a neatness from the fucking book, MAD stands
 * for MadMax, just kidding, it stands for Multiply-Add-and-Divide,
 * this is to try to get to a perfect hash or something.
 * @param hashCode  The hash you want to compress like a piece of fucking
 *                  dog shit on your shoe pancake dreams.
 * @param N         The size of the bucket.
 * @param p         The first prime number after the size of the
 *                  array thing.
 * @param a         An unspecific integer value
 * @param b         Another fucking unspecific integer value, bitch.
 * @return          to Thunderdome.
 */
public static int madCompression(int hashCode, int N, int p, int a,
        int b) throws IllegalArgumentException {

    //check the information contained in the variable a
    if (a > (p-1)) throw new IllegalArgumentException("a needs to be"
            + " less than p-1 not greater");
    //check the lower limit contained in the variable a
    if (a < 0) throw new IllegalArgumentException("a needs to be greater"
            + " than 0, not less than");
    //check the upper limit of contained in the variable b
    if (b > (p-1)) throw new IllegalArgumentException("b needs to be"
            + " less than p-1 not greater");
    //check the lower limit contained in the variable b
    if (b < 0) throw new IllegalArgumentException("b needs to be greater"
            + "than 0, not less");
    //check to see if the number for p is a prime number
    return Math.abs(((a * hashCode + b) % p) % N);
}

/**
 * Prints an ASCII table of a width of 79 characters to keep with the upper
 * limit of 80 characters in latex output, this table is only really
 * designed for an nx3 matrix, which is the most applicable for this
 * assignment; diarrhea in her vagina.
 * @param dataTable The nx3 matrix that stores the data.
 * @param title     The title you want to give your table.
```

```
     * @param subTitle1 The first sub-title for the first column in the table.
     * @param subTitle2 The second sub-title for the second column in
     *                  the table.
     * @param subTitle3 The third sub-title for the third column in the table.
     * @return          An ASCII table containing your motherfucking, drip
     *                  drip cumming table.
     */
    public static String asciiPenis(int[][] dataTable, String title,
            String subTitle1, String subTitle2, String subTitle3){
        //this will eventually be the table that is returned
        StringBuilder foreskin = new StringBuilder("");

        //used for adding commas to the numbers in the table
        NumberFormat numberFormat = NumberFormat.getNumberInstance(Locale.US);

        //add the top part of the table, 79 is the max size of the
        //table so it will look better on ascii
        foreskin.append("|");
        foreskin.append(repeatedBitches('-',77));
        foreskin.append("|");

        //add the title to the table
        foreskin.append("\n|");
        foreskin.append(centerLabia(title,77));
        foreskin.append("|\n");

        //separate the title and the subtitles
        foreskin.append("|");
        foreskin.append(repeatedBitches('-',77));
        foreskin.append("|");

        //add the subtitles to the the table
        foreskin.append("\n|");
        //use 25 space because of latex requirement
        foreskin.append(centerLabia(subTitle1,25));
        foreskin.append("|");
        foreskin.append(centerLabia(subTitle2,25));
        foreskin.append("|");
        foreskin.append(centerLabia(subTitle3,25));
        //add the final pipe to the line;
        foreskin.append("|\n");

        //complete the bottom.
        foreskin.append("|");
        foreskin.append(repeatedBitches('-',77));
        foreskin.append("|\n");

        //add the data to the data
        for (int[] dataTable1 : dataTable) {
            foreskin.append("|");
            for (int j = 0; j < dataTable1.length; j++) {
                //add each number to the table, and add commas to the number
                foreskin.append(centerLabia(numberFormat.format(
                        dataTable1[j]), 25));
                foreskin.append("|");
            }
            foreskin.append("\n|");
            //separate the lines
            foreskin.append(repeatedBitches('-',77));
            foreskin.append("|\n");
        }
```

```
    //return to thunderdome
    return foreskin.toString();
}

/**
 * Returns a string of specified number of repeated characters.
 * @param bloodyAnal    The character you want repeated
 * @param num           The number of repeated characters.
 * @return              A string full of a bunch of repeated characters.
 */
public static String repeatedBitches(char bloodyAnal, int num){
    //a temporary StringBuilder thing
    StringBuilder cumShot = new StringBuilder("");
    //add the repeated character
    for (int i = 0; i < num; i++){
        cumShot.append(bloodyAnal);
    }
    //return this motherfucker
    return cumShot.toString();
}

/**
 * This centers text in a given amount of space; I pissed in her maggot
 * filled asshole.
 * @param text      The words you want to center.
 * @param width     The number of character you want to use up,
 *                  for example if the space you want to fill is 75
 *                  characters, it will center the text within
 *                  the 75 characters
 * @return          A StringBuilder for making it faster and easier to
 *                  build a table or other massive string.
 */
public static StringBuilder centerLabia(String text, int width){
    //the number of spaces for the left spaces
    int leftNipple = (width - text.length()) / 2;
    //the number of spaces for the right space
    int rightNipple = (width - leftNipple - text.length());

    //This will be used to build the string thingy
    StringBuilder volva = new StringBuilder("");

    //add the spaces to the string thing
    for (int i = 0; i < leftNipple; i++)
        volva.append(" ");

    //add the normal text to the stringbuilder
    volva.append(text);

    //add the rest of the spaces to the StringBuilder
    for (int i = 0; i < rightNipple; i++)
        volva.append(" ");

    //return this bitch
    return volva;
}

/**
 * Recursively finds the next prime number after a given value.
 * @param numbDick  The number you want to find the next prime after.
 * @return          An int that is the next prime number after the
 *                  given number.
```

```
     */
    public static int findNextPrime(int numbDick){
        if (isPrime(numbDick,2))
            return numbDick;
        else
            return findNextPrime(numbDick + 1);

    }

    /**
     * Recursively determines if a number is a prime
     * @param numTit    The number you want to determine if it is a prime.
     * @param divisor   The current divisor you are checking.
     * @return          True if it is a prime number, false otherwise.
     */
    public static boolean isPrime(int numTit, int divisor){
        //base cases
        if (numTit <= 2)
            return (numTit == 2);
        if (numTit % divisor == 0)
            return false;
        if (divisor * divisor > numTit)
            return true;

        //for the next divisor bitch ass.
        return isPrime(numTit, divisor + 1);
    }
}
```

2 MapEntry.java

```
/**
 * An alteration of the MapEntry from the UnsortedMap thing from the book,
 * very much altered, but the book it came from was Data Structures
 * And Algorithms.
 * @author Steven Glasford, Michael T Goodrich, Roberto Tamassia,
 * Michael H Goldwasser.
 */
public class MapEntry implements Entry {
        //key
        private int k;
        //value
        private int v;
        public MapEntry(int key, int value){
            k = key;
            v = value;
        }
        //public methods of the Entry interface
        @Override
        public int getKey() {return k;}
        @Override
        public int getValue() {return v;}
        public void createEntrant(int key, int value){
            k = key;
            v = value;
        }
        //utilities not exposed as part of the Entry interface
        public void setKey(int key) {k = key;}
        public int setValue(int value) {
            int old = v;
            v = value;
            return old;
        }
    }
```

3 Entry.java

```java
/**
 *An Interface for a key-value pair, diarrhea queef, altered to only contain
 * ints.
 * @author Michael T Goodrich, Roberto Tamassia, Michael H Goldwasser,
 * Steven Glasford
 * @version 4-3-2019
 */
public interface Entry {
    //returns the key stored in this entry.
    int getKey();
    //returns the value stored in this entry, bitch.
    int getValue();
}
```

4 List.java

```java
/**
 * A simplified version of the "java.util.List" interface
 *
 * @author Michael T. Goodrich
 * @author Roberto Tamassia
 * @author Michael H. Goldwater
 * @author Steven Glasford
 * @version 2-21-2019
 * @param <E>
 */

public interface List<E> {
    /**
     * Returns the number of elements in this list.
     * @return
     */
    int size();

    /**
     * Returns whether the list is empty
     * @return
     */
    boolean isEmpty();

    /**
     * Returns (but does not remove) the element at index i.
     * @param i
     * @return
     */
    E get(int i) throws IndexOutOfBoundsException;

    /**
     * Replaces the element at index i with e, and returns the replaced
     * element.
     * @param i
     * @param e
     * @return
     */
    E set(int i, E e) throws IndexOutOfBoundsException;

    /**
     * Inserts element e to be at index i, shifting all subsequent
     * elements later.
     * @param i
     * @param e
     */
    void add(int i, E e) throws IndexOutOfBoundsException;

    /**
     * Removes/returns the element at index i, shifting subsequent
     * elements earlier.
     * @param i
     * @return
     */
    E remove(int i) throws IndexOutOfBoundsException;
}
```

5 SinglyLinkedList.java

```
/**
 *
 * SinglyLinkedList Class
 * Code Fragments 3.14, 3.15
 * from
 * Data Structures & Algorithms, 6th edition
 * by Michael T. Goodrich, Roberto Tamassia & Michael H. Goldwasser
 * Wiley 2014
 * Transcribed by
 * @author Steven Glasford
 * @version January 31, 2019
 * @param <E> a generic placeholder name
 */
public class SinglyLinkedList<E> {
    /**
     *
     * @param <E> a generic placeholder name
     *
     * A subclass creating the Node
     */
    private static class Node<E>{
        //reference to the element stored at this node
        private final E element;
        //reference to the subsequent node in the list
        private Node<E> next;
        public Node(E e, Node<E> n){
            element = e;
            next = n;
        }

        /**
         *
         * @return Return the current element
         */
        public E getElement(){return element;}

        /**
         *
         * @return return the address of the next item in the linked list
         */
        public Node<E> getNext() {return next;}

        /**
         *
         * @param n the next item in the list
         */
        public void setNext(Node<E> n) {next = n;}
    }

    //head node of the list (or null if empty)
    private Node<E> head = null;
    //last node of the list (or null if empty)
    private Node<E> tail = null;
    //number of nodes in the list
    private int count = 0;

    /**
     * constructs an initially empty list
     */
```

```
public SinglyLinkedList(){}

//access methods
/**
 *
 * @return Return the size of the linked list
 */
public int size() {return count;}

/**
 *
 * @return Determine if the linked list is empty
 */
public boolean isEmpty() {return count == 0;}

/**
 *
 * @return return the first element in the list
 *
 * returns (but does not remove) the first element
 */
public E first(){
    if (isEmpty()) return null;
    return head.getElement();
}

/**
 *
 * @return the last element in the linked list
 *
 * returns (but does not remove the last element
 */
public E last(){
    if (isEmpty()) return null;
    return tail.getElement();
}

//update methods

/**
 *
 * @param e A generic element
 *
 * adds element e to the front of the list
 */
public void addFirst(E e){
    //create and link a new node
    head = new Node<>(e, head);
    //special case: new node becomes tail also
    if (count == 0)
        tail = head;
    count++;
}

/**
 *
 * @param e A generic item
 *
 * adds element e to the end of the list
 */
public void addLast(E e) {
```

```
        //node will eventually be the tail
        Node<E> newest = new Node<>(e,null);
        //special case: previously empty list
        if (isEmpty())
            head = newest;
        else
            tail.setNext(newest);
        tail = newest;
        count++;
    }

    /**
     *
     * @return return the item that was removed
     *
     * removes and returns the first element
     */
    public E removeFirst(){
        //nothing to remove
        if (isEmpty()) return null;
        E answer = head.getElement();
        //will become null if list had only one node
        head = head.getNext();
        count--;
        //special case as list is now empty
        if(count == 0)
            tail = null;
        return answer;
    }
}
```

6 output.txt

```
/*
This table contains the information about
the number of collisions and the number used for alpha.
```

Polynomial Hash Code		
Alpha Number	Total collisions	Max Collisons
30	3	1
31	0	0
32	12,135	152
33	1	1
34	0	0
35	0	0
36	12	1
37	0	0
38	0	0
39	0	0
40	765	14
41	0	0
42	1	1
43	1	1
44	4	1

```
The following table contains data from the
running of madCompression method, and the number
used for the prime variable.
```

MAD Compression		
Prime Number	Total collisions	Max Collisons
45,413	16,811	7
45,427	16,733	6
45,433	16,754	6
45,439	16,757	7
45,481	16,737	7
45,491	16,752	7

```
|          45,497           |          16,662           |             6             |
|---------------------------------------------------------------------------------|
|          45,503           |          16,781           |             6             |
|---------------------------------------------------------------------------------|
|          45,523           |          16,656           |             7             |
|---------------------------------------------------------------------------------|
|          45,533           |          16,592           |             6             |
|---------------------------------------------------------------------------------|
|          45,541           |          16,758           |             7             |
|---------------------------------------------------------------------------------|
|          45,553           |          16,748           |             6             |
|---------------------------------------------------------------------------------|
|          45,569           |          16,811           |             6             |
|---------------------------------------------------------------------------------|
|          45,587           |          16,830           |             6             |
|---------------------------------------------------------------------------------|
|          45,599           |          16,785           |             8             |
|---------------------------------------------------------------------------------|

*/
```