Sarah Depew

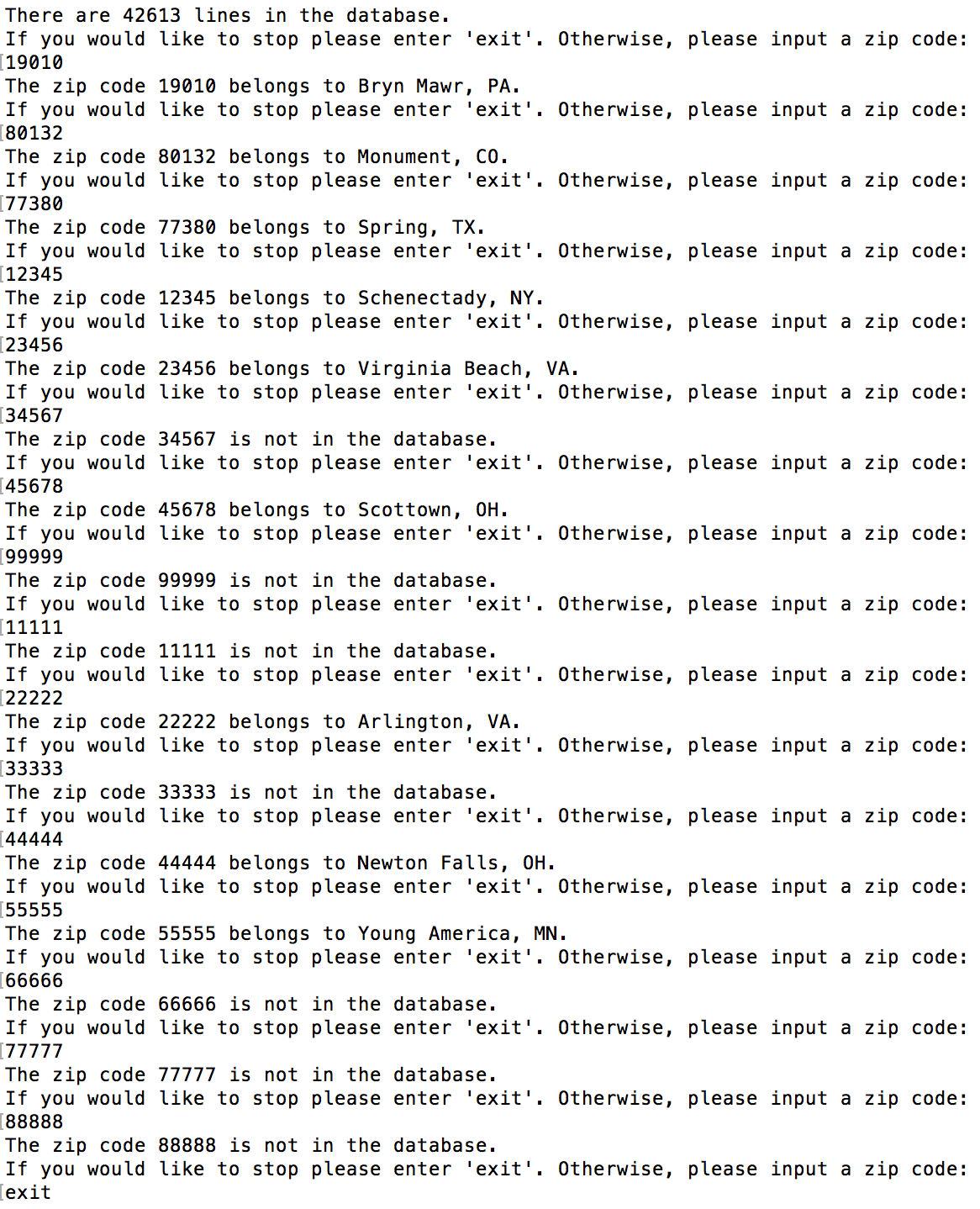
Professor Kumar

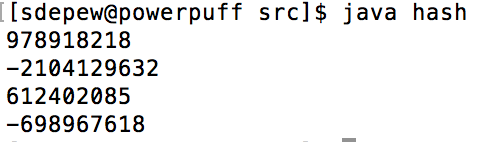
CS330 Algorithms: Design & Practice

8 March 2018

**Lab 6: Hash Tables and Hash Functions**

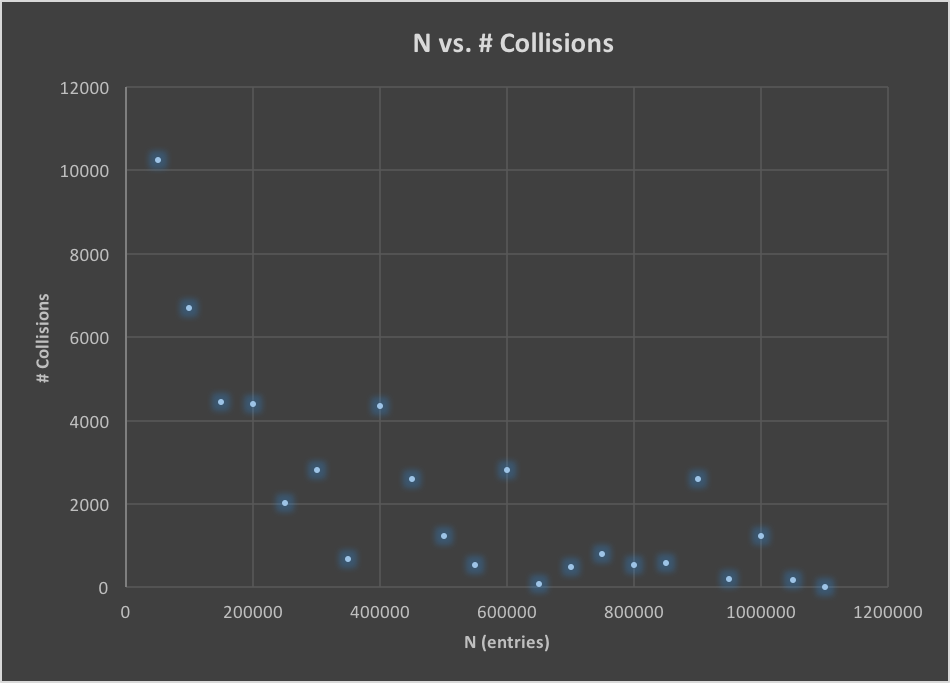
**Sample Runs and Screenshots from Task #1:**

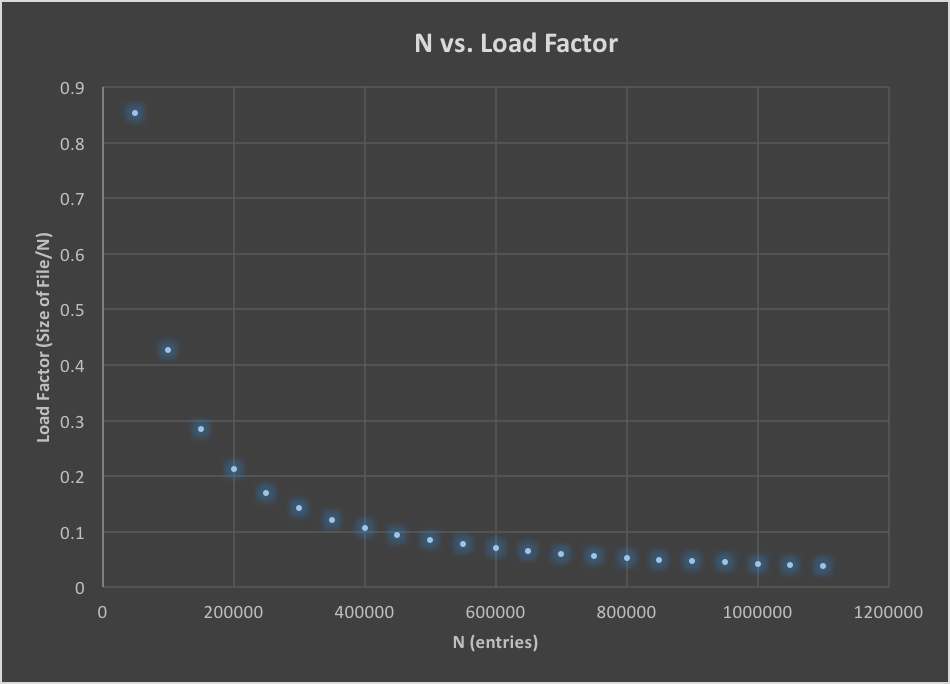
****





**Plots:**

****

****

plot the maximum number of collisions vs. *N*.

(how many entries actually had a collision)

**Discussion:** (A report containing a discussion on this lab: implementation and use of hash tables, the performance of hash functions, based on data you collected.)

*N* is number of hash codes you have…

In this lab, the first task was to read in, parse, and organize the contents of a file in a manner that made it easily accessible. In this lab, a HashTable was used as the main data structure to hold the information in the file. As a means of accessing the file’s information about locations in the United States, the town and state of a location was indexed by its zip code. Since zip codes are unique to a single geographical location, this guaranteed that there would be no data overlaps or issues with obtaining the desired outcome for the data.

After writing a short, yet effective, program to allow a user to type in a zip code and access data from the hash table, it came time to investigate hashing further. In the second task of this lab, the impetus was to learn more about hashing in Python and in Java. Each language has methods that allow you to compute and, using additional methods, print out the hash codes of various strings. In this portion of the lab, a variety of Bryn Mawr-oriented strings had their hash codes computed and printed to the user.

Finally, a third file was written to compute the number of collisions of \_\_\_.\_\_\_\_.\_\_\_\_.\_\_\_.

What this lab helped me realize was that, while hash maps are *on average* very efficient with O(1) access to information, the runtime is definitely relative to and dependent upon the number of collisions that occur and the number of (chaining don’t need to be too big, since collisions…)

**You can have a load factor of 80% and still have almost constant access still**

**Code:**

**Task #1:**

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.net.MalformedURLException;

import java.net.URL;

import java.util.HashMap;

import java.util.Scanner;

public class Lab06 {

public static boolean EXIT = false;

private static String line = new String();

private static int zipCode;

private static HashMap<Integer, String> data = new HashMap<>();

public static void main(String args[]) {

//read in that

int numberLines = readInData();

if (numberLines == -1) {

System.out.println("I am sorry, but there was an error reading in the data.");

} else {

System.out.println("\n\nThere are " + numberLines + " lines in the database.");

}

//allow the user to query the newly read in information

while (!EXIT) {

Scanner scanner = new Scanner(System.in);

System.out.println("If you would like to stop please enter 'exit'. Otherwise, please input a zip code: ");

line = scanner.nextLine().trim();

if (line.toUpperCase().equals("EXIT")) {

System.out.println("Thank you! Have a good day.");

EXIT = true;

} else {

zipCode = Integer.parseInt(line);

if (data.containsKey(zipCode)) {

System.out.println("The zip code " + line + " belongs to " + data.get(zipCode) + ".");

} else {

System.out.println("The zip code " + line + " is not in the database.");

}

}

}

}

//used code/tutorial found on https://docs.oracle.com/javase/tutorial/networking/urls/readingURL.html

private static int readInData() {

int numLines = -1; //default return value

try {

// URL url = new URL("https://cs.brynmawr.edu/Courses/cs330/spring2018/testZip.txt");

URL url = new URL("https://cs.brynmawr.edu/Courses/cs330/spring2018/uszipcodes.csv");

BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()));

String inputLine = in.readLine();

String[] tokens = inputLine.split(",");

numLines = Integer.parseInt(tokens[0]);

while ((inputLine = in.readLine()) != null) {

tokens = inputLine.split(",");

String stateAndTown = tokens[1] + ", " + tokens[2];

data.put(Integer.parseInt(tokens[0]), stateAndTown);

}

in.close();

} catch (MalformedURLException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

} catch (NumberFormatException e) {

e.printStackTrace();

}

return numLines;

}

}

**Task #2:**

public class hash {

public static void main(String args[]) {

String s = "Bryn Mawr";

System.out.println(s.hashCode());

s = "K-Cass";

System.out.println(s.hashCode());

s = "Haverford";

System.out.println(s.hashCode());

s = "Swarthmore";

System.out.println(s.hashCode());

}

}

import hashlib

print(">>> import hashlib")

print(">>> hash(\"Bryn Mawr\")")

print(hash("Bryn Mawr"))

print("\n>>> hash(\"K-Cass\")")

print(hash("K-Cass"))

print("\n>>> hash(\"Haverford\")")

print(hash("Haverford"))

print("\n>>> hash(\"Swarthmore\")")

print(hash("Swarthmore"))

**Task #3:**

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.net.MalformedURLException;

import java.net.URL;

public class graph {

private static float N = 50000; //starting value

private static float numLines = 42613;

public static void main(String args[]) {

//read in that

int numberLines;

while ((numberLines = readInData(N)) != 0) {

System.out.println("At N = " + (int) N + " we get " + numberLines + " collisions and a load factor of " + (numLines / N));

N += 50000;

}

System.out.println("At N = " + (int) N + " we get " + numberLines + " collisions and a load factor of " + (numLines / N));

}

//used similar code to tutorial found on https://docs.oracle.com/javase/tutorial/networking/urls/readingURL.html to understand reading from URL's

private static int readInData(float N) {

int[] collisions = new int[(int) N];

try {

URL url = new URL("https://cs.brynmawr.edu/Courses/cs330/spring2018/uszipcodes.csv");

BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()));

String inputLine = in.readLine(); //read in and remove the first line

String[] tokens;

while ((inputLine = in.readLine()) != null) {

tokens = inputLine.split(",");

collisions[(Math.abs(tokens[0].hashCode()) % (int) N)]++;

}

in.close();

} catch (MalformedURLException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

} catch (NumberFormatException e) {

e.printStackTrace();

}

return collisionsNumber(collisions);

}

private static int collisionsNumber(int[] collisions) {

int totalCollisions = 0;

for (int i = 0; i < collisions.length; i++) {

if (collisions[i] > 1) {

totalCollisions += (collisions[i] - 1);

}

}

return totalCollisions;

}

}