import java.io.File;

import java.io.FileNotFoundException;

import java.util.HashSet;

import java.util.Scanner;

import java.util.Set;

/\*\*

\* In-class lab for learning about hash code functions and collisions.

\* @author arafferty

\* @author Yucong Jiang

\*/

public class HashCodeCalculationsSolution {

/\*\*

\* Always returns 0.

\*/

public static int hashCode0(String s) {

return 0;

}

/\*\*

\* Returns the ASCII code of the first character in the string.

\*/

public static int hashCode1(String s) {

if(s.isEmpty()) {

return 0;

} else {

return (int) s.charAt(0); // (int) casts a value to an integer

}

}

/\*\*

\* Returns the sum of the ASCII codes of the characters in the string.

\*/

public static int hashCode2(String s) {

int hashCode = 0;

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

hashCode += (int) s.charAt(i);

}

return hashCode;

}

/\*\*

\* Returns the sum of the values in this form:

\* sum = a\_1\*129^(n-1) + a\_2\*129^(n-2) + ... + a\_{n-1}\*129^(1) + a\_n\*129^(0)

\* where a\_{i} is the ASCII code of the ith character in the string, and n is

\* the length of the string.

\*/

public static int hashCode3(String s) {

int hashCode = 0;

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

hashCode = 129\*hashCode + (int) s.charAt(i);

}

return hashCode;

}

/\*\*

\* Implement this function so it works the way we talked about in class.

\* Compression function that takes a hash code (positive or negative) and

\* the number of buckets we have to use in our hash table, and compresses

\* the hash code into the range [0, numberOfBuckets).

\*/

public static int compressToSize(int hashCode, int numberOfBuckets) {

int hashIndex = hashCode % numberOfBuckets;

if (hashIndex < 0) {

hashIndex = hashIndex + numberOfBuckets;

}

return hashIndex;

}

/\*\*

\* Counts the number of buckets that have no words stored at them - i.e.,

\* they have value 0 - and calculates what proportion of the total buckets

\* that is.

\*/

public static double proportionOfBucketsWithNoWords(int[] buckets) {

int emptyBucketCount = 0;

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

if(buckets[i] == 0) {

emptyBucketCount++;

}

}

return emptyBucketCount\*1.0/buckets.length;

}

/\*\*

\* Returns the maximum value in a single bucket

\*/

public static int getMaxBucketValue(int[] buckets) {

int max = -1;//Safe starting value since all buckets[i] should be >= 0

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

if(buckets[i] > max) {

max = buckets[i];

}

}

return max;

}

/\*\*

\* Returns the average number of words in each non-empty bucket

\*/

public static double getAverageInNonEmptyBuckets(int[] buckets) {

int totalCount = 0;

int totalNonEmpty = 0;

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

totalCount += buckets[i];

if(buckets[i] != 0) {

totalNonEmpty++;

}

}

return totalCount\*1.0/totalNonEmpty;

}

/\*\*

\* Implement this method so that it calculates how many words would be placed

\* in each bucket in the array.

\* Each individual word should be counted only once (i.e., if "the" occurs

\* 501 times in the file, you should only hash it once, rather than thinking

\* of it as causing 500 collisions).

\* @param numBuckets number of spots to include in the array

\* @param file file to read from

\* @param hashCodeFunctionToUse which of the hash functions to use; see lab

\* description for more details

\* @return an array that indicates how many different words are place in index 0, 1, etc.

\*/

public static int[] collisionCounter(int numBuckets, String file, int hashCodeFunctionToUse) {

//Initialize the variables you'll need to count collisions (an array, a set)

int[] counts = new int[numBuckets];

Set<String> seenWords = new HashSet<>();

try {

Scanner scanner = new Scanner(new File(file));

//Write your code for counting collisions here.

while (scanner.hasNext()) {

String word = scanner.next().toLowerCase();

if (!seenWords.contains(word)) { // only consider words not seen yet.

seenWords.add(word);

int hashIndex = compressToSize(getHashCode(word, hashCodeFunctionToUse), numBuckets);

counts[hashIndex]++;

}

}

scanner.close();

} catch(FileNotFoundException e) {

e.printStackTrace();

}

return counts;//Change this line to return your count of collisions

}

private static int getHashCode(String word, int hashCodeFunctionToUse) {

int hashCode = -1;

if (hashCodeFunctionToUse == -1) {

hashCode = word.hashCode();

} else if (hashCodeFunctionToUse == 0) {

hashCode = hashCode0(word);

} else if (hashCodeFunctionToUse == 1) {

hashCode = hashCode1(word);

} else if (hashCodeFunctionToUse == 2) {

hashCode = hashCode2(word);

} else if (hashCodeFunctionToUse == 3) {

hashCode = hashCode3(word);

} else {

System.err.println("Incorrect hashCodeFunctionToUse in collisionCounter!");

System.exit(0);

}

return hashCode;

}

public static void main(String[] args) {

if (args.length == 0) { // Calcuating hash codes for "ant", "tan", "mop", and "tiger"

System.out.println("hashcode0: " + hashCode0("ant") + ", " +

hashCode0("tan") + ", " + hashCode0("mop") + ", " +

hashCode0("tiger"));

System.out.println("hashcode1: " + hashCode1("ant") + ", " +

hashCode1("tan") + ", " + hashCode1("mop") + ", " +

hashCode1("tiger"));

System.out.println("hashcode2: " + hashCode2("ant") + ", " +

hashCode2("tan") + ", " + hashCode2("mop") + ", " +

hashCode2("tiger"));

System.out.println("hashcode3: " + hashCode3("ant") + ", " +

hashCode3("tan") + ", " + hashCode3("mop") + ", " +

hashCode3("tiger"));

} else if (args.length == 1) { // Measuring Collisions: args[0] would be the file path.

int[] sizes = {1048, 1543, 2048, 196613, 200000};

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

for (int h = -1; h < 4; h++) {

int[] counts = collisionCounter(sizes[i], args[0], h);

System.out.println("\nSize = " + sizes[i] + ", Hash function #" + h);

System.out.println("proportionOfBucketsWithNoWords: " + proportionOfBucketsWithNoWords(counts));

System.out.println("getMaxBucketValue: " + getMaxBucketValue(counts));

System.out.println("getAverageInNonEmptyBuckets: " + getAverageInNonEmptyBuckets(counts));

}

}

} else if (args.length == 3) { // Measuring Collisions

int[] buckets = collisionCounter(Integer.parseInt(args[0]), args[1], Integer.parseInt(args[2]));

System.out.println("proportionOfBucketsWithNoWords: " + proportionOfBucketsWithNoWords(buckets));

System.out.println("getMaxBucketValue: " + getMaxBucketValue(buckets));

System.out.println("getAverageInNonEmptyBuckets: " + getAverageInNonEmptyBuckets(buckets));

System.out.println();

} else {

System.out.println("Please provide either zero, one (filePath), or three arguments: [numBuckets] [filePath] [hashFunctionToUse].");

}

}

}