Report on Hash Tables and Hash Functions

Yu Sheng(Julia)

2018/3/6

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Codes of Task#1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import java.io.IOException;

import java.net.URL;

import java.util.HashMap;

import java.util.Map;

import java.util.Scanner;

public class zip {

private static Map<String, String> places = new HashMap<String, String>();

private static int N=50000;

private static int [] counts=new int[N];

public static void main(String[] args) {

readData();

System.out.println("the size of HashMap is:" + places.size());

Scanner cw = new Scanner(System.in);

boolean more = true;

String query = null;

do {

System.out.print("\nQuery: ");

query = cw.nextLine();

String result = places.get(query);

if (result != null) {

System.out.println("The zip code " + query + " belongs to " + result);

} else {

System.out.print("\nYour search for: " + query);

System.out.print(" is not in the database. Sorry!\n");

}

System.out.print("\nWould you like to ask another query(Y/N)?");

String response = cw.nextLine();

more = response.equalsIgnoreCase("y");

} while (more);

cw.close();

}

public static void readData() {

Scanner inS;

try {

for(int i=0;i<N;i++)

counts[i]=0;

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

inS = new Scanner(webFile.openStream());

String line = inS.nextLine();

while (inS.hasNextLine()) {

line = inS.nextLine();

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

String town, state, zip;

zip = tokens[0];

town = tokens[1];

state = tokens[2];

String ts=town+", "+state;

String key = zip;

counts[Math.abs(zip.hashCode()) % N]++;

places.put(key.toLowerCase(), ts);

}

inS.close();

int sum=0;

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

if(counts[i]>0)

sum=sum+(counts[i]-1);

}

System.out.println(N+" "+sum);

} catch (IOException e) {

e.printStackTrace();

System.exit(1);

} // catch

}// read data

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Result for Task#1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

the size of HashMap is:42613

Query: 19010

The zip code 19010 belongs to Bryn Mawr, PA

Would you like to ask another query(Y/N)?y

Query: 99999

Your search for: 99999 is not in the database. Sorry!

Would you like to ask another query(Y/N)?y

Query: 09079

The zip code 09079 belongs to Apo, AE

Would you like to ask another query(Y/N)?y

Query: 49087

The zip code 49087 belongs to Schoolcraft, MI

Would you like to ask another query(Y/N)?y

Query: 2763728638

Your search for: 2763728638 is not in the database. Sorry!

Would you like to ask another query(Y/N)?n

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Discussion & Plot \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

I use Java to implement the program for this lab, thus use hashmap function to implement the hash table. The input of program is a zip code and the output of the program is the city and state of that zip belongs to. I use the hash table to store the information from database. Once a new line from data file is read, set zip code(first word in the line) in type string to be a new key and value will be the corresponding city and state(second and third words), which is also in type string. I will keep doing until the entire database is loaded, then ask a person to enter a zip code and see whether it is a key in hash table, if it is, print out the corresponding city and state names, if not, then the zip code is not in the database. I will ask whether there is new query for zip code, if answer is ‘y’, then we will search for this new zip code using the process described above, until answer is ‘n’ where the whole program ends.

For Task #2, we generate a new array called counts which records the number of collision in each entry. The size of array is N, and at the beginning N is 50000. During building the hash table, every time a new line is read and we have a new zip, we calculate an index which is the hash code of that zip code mod N. The index represents the position of zip in the counts array, thus we add 1 to counts[index]. When counts[i]=2, then two zip are both mapped into this position I, which also means there is one collision happened in this position. And if there is a new zip being mapped into this position, counts[i]=2+1=3, then there is one more collision, thus we have 2 collisions in this position now. Therefore, number of collisions at each index of array equals to the number of zip codes mapped minus one at that index, except when no zip code is mapped to this index, then the number of collision would be zero. Summing up the number of collisions at each index will give us the total number of collisions at that array size N. We will repeat the same process for N increased by 50000 each time until total collision numbers becomes 0.

**Fig 1 Array size vs. Total Collision numbers**

Thus we find that in Fig 1 above, total collision numbers are generally decreasing as array size increases. And total collision number approaches to zero several times when array size is large (bigger than 50000).

**Fig 2 Array size vs. Load Factor**

Load factor is calculated by number of entries in database divided by array size. Since the number of entries in database remains the same as 42613, the load factor is decreasing and approaching to zero when array size increases, which is also shown in Fig 2. What’s more, we see in Fig 2, load factor drops in a rapid rate at the beginning of the increment of array size, since load factor is 0.85 when array size is 50000 but reduced a half becomes 0.42 when array size is 100000.

Combining two graphs, we find when array size increases, total collision number is decreasing, searching in the hash table would be quicker since there won’t be many keys all mapped to same bucket. However, load factor is approaching to 0 at the same time, which means the proportion of unused section of hash table increases and it is a huge waste of memory. Therefore, we should balance between making hash table quicker and not wasting too much memory usage when determining the most suitable array size.