CSC 103 Programming Assignment #5 11/03/21

Due date – **Wednesday Dec 15th @ 11:59pm**

Each student must do the work on this project by themselves unless they have permission by the instructor to work with another. You are not allowed to use a previous student’s work, or find a solution online and use it. Doing so will have consequences far worse than just a zero on this assignment. Taking credit for work you have not done is unethical. Note: making minor changes to an existing solution is not considered doing the work. Any work that ends up in your projects that is not your own, must be cited as such. You will lose credit for parts of your projects that are not yours, but you will not be accused of plagiarism so long as you do not take credit for it. Any plagiarism will result in an automatic zero on this assignment, and possible withdrawal from this class.

**Project # 5 Analysis of Hashing, Double Hashing, and Chain Hashing**

Your task for this program is to fill up a Table with some sample data using hash techniques and count the number of **collisions** that happen when placing the elements in the table. You will report the average number of collisions that happened in placing all the elements and show a listing of numbers of collisions for each attempted placing in the table. See a sample output at end of project description.

You will then repeat this process using a different Table class that uses Double Hashing techniques. And a Third using Chain Hashing. You will report the averages of each and output a table of collisions. See sample output on end of this document.

Values to input-

You will be given an input file which contains 200 names and 9-digit numbers. The names will be your elements <String> and the numbers will be your keys <Integer>.

Summary of program:

Your program will start by reading each line in the file and placing the names into the table based on the key given. The table will be created from the given **Table** class (which you will be able to alter). As prime numbers work the best for hashing, and we want to force a fair number of collisions, you should use a table size of 241. [This will cause the finished table to be over 82% full. And because 239 is also a prime, we will be set up well to use double hashing.] As you place each element, you will need to keep track of how many times you attempted to place the element in a spot that was already used. Each of these is a collision. (**I am not interested in how many collisions each location of the array is having, just the number of collisions per element placed into the table**.) Repeat(or process at the same time) the test for double hashing and for chain hashing. Then report the results for all three tests in a table (see output example). With the same hash function and input, we should all get the same results.

Note: The **Table** class is already completed for you to use linear hashing without any changes, however you will still need to alter the **Table** class with just enough code to be able to count and report back on the number of collisions for each time you place an element. You are not allowed to alter the class in a way that no longer makes it a generic class.

You will need to create additional versions of the **Table** class, one with alterations to place data using a double hashing method and the other using a chain hashing method. Call these classes **TableDoubleHash** and **TableChainHash**. These classes will have almost all of the same code as the original **Table**, but you will add just enough alterations on how to handle the collisions for double hashing or chain hashing.

Chain hashing means you will need to make some changes on how data is stored within the table as well. But it turns out that it is not a lot of difficult updates. Note: You do not need to create a new Node class (ChainedHashNode) as discussed in the textbook. Instead – you will search for the proper location in the array as in linear hashing, and then when you are ready to put in the new element at an index, you will add a new Node (that contains the element data) to the spot in the array. Remember that with chained hashing the main array will be storing a linked list at each location. (also the keys will be stored this way in a separate array) It may sound confusing, but there is really not a lot of extra coding.

Note: All the methods that exist in the original **Table** class should work in your altered versions. So, make sure to update all of them, even if you don’t need all of them to get the output. (**get** and **remove** methods are a couple of examples)

What classes that you need to hand in:

* **Table** (template given)
* **TableDoubleHash** (table with a bunch of changes)
* **TableChainHash** (for chain hashing, uses Node class)
* **Node** (unaltered as given)
* **HashTesting**- main driver of program.

The sample data file given is called **names.txt** and will be used as a starting point to test your program. A copy of this file is given in with these program directions. I recommend that you do some testing, such as using a small subset of the input and placing by hand the next name and then checking that it was placed where you expected.

The input file used is data stored in text format (not binary, so that we can read it) with one name and 9-digit number on each line and spaces between fields. It is organized like this:

**Name 9DigitNumber**

Tips for good grades:

* Make sure you use comments where needed and use variable names that make sense, some of your grade will depend on program style as well as the use of your program.
* Update the comments in the class file, to include your names and any new information
* You will lose points for things like not indenting, or naming variables in non-descriptive ways. Do no leave in debugging code, or commented out code.
* I use jGrasp and the java version that is in the lab computers. So make sure that your programs work with this.
* Test your own projects thoroughly before you hand them in.
* Late projects will not be accepted so plan ahead.

The four classes you are using for this project should be in separate files. Name them **HashTesting.java, Table.java, TableDoubleHash.java** and **TableChainHash.java**. If you do not name these files correctly, you will lose points. Javadoc is not needed for submission.

All students must submit the assignment. Hand in electronically – (NOT E-mail!!!)

In Blackboard:

1. Make sure that you have created and tested all files to submit: **HashTesting.java, Table.java, TableDoubleHash.java, Node.java** and **TableChainHash.java**
2. Using the zip utility on your computer place all five files into a zipped folder called Project5\_lastname.zip (using your real lastname)
3. Open Blackboard for this course
4. Click on Programming Projects (left side of screen)
5. Click on correct Project to submit
6. Click in Attach files box – Browse my computer (Click “Start New” first if this is not your first submission)
7. Locate zipped folder and Attach
8. Click Submit on bottom right
9. You will get a confirmation e-mail
10. You may submit as often as you want, I will only grade the last submission that happens before the due date/time.

A sample output of my program is shown on next page….

Sample output: (used smaller subsample of data)

(First twenty-five names/numbers – into array size of 31)

Collisions per Attempted placement in Tables  
Attempt Linear Double Chain  
 1 0 0 0  
 2 0 0 0  
 3 0 0 0  
 4 0 0 0  
 5 0 0 0  
 6 0 0 0  
 7 0 0 0  
 8 0 0 0  
 9 1 1 1  
 10 0 0 0  
 11 0 0 0  
 12 0 0 0  
 13 1 1 1  
 14 0 0 0  
 15 1 3 1  
 16 4 2 2  
 17 1 2 1  
 18 0 0 0  
 19 2 3 1  
 20 9 6 1  
 21 7 0 0  
 22 3 0 0  
 23 10 5 1  
 24 9 2 0  
 25 0 0 0  
linear average = 1.92  
double average = 1.0  
chain average = 0.36