Lab Assignment: Hash algorithm analysis

In this assignment, we are going to analyze the probing processing times of a number of hashing algorithms that we have learned in the class. Use Java programming language to complete this programming assignment. Do not use algorithms that we have not taught in the class.

We would like to determine how the hashing algorithm would behave when trying to add a new object of a custom class in the hash table (how many statements are needed to be processed to find a suitable position in the table) or search an object from the hash table (how many statements are needed to be executed in order to locate the object in the hash table (or conclude the search was unsuccessful).

Deliverable 1:: First, create a Student class (phone, name, email, course) and overload necessary operators or methods. Follow the steps suggested in the lecture slides {for example, the one expert suggestion slide} to compute the hash value of a Student object (string, string, string, string). In Java, overload the hash() and the equals() function in order to create/return a hash value and compare two objects of the Movie class respectively.

Deliverable 2: Load the Student information from the supplied file (or create your own mock dataset for the objects by visiting https://www.onlinedatagenerator.com/home/demo) and store the Student objects in a vector (or ArrayList) of Student objects. All file read related exceptions were handled in the program file.

Each line of the file contains the phone number, name, email, and the course information. In a line, each of these values are separated by a comma (you can open the file in any text editor and review the format before using it in your program). After reading a line of text from the file, you should be able to split these values by using a comma.

Deliverable 3: We would like to store the Student objects in the hash tables so that we can quickly insert them and find them. For this purpose, implement the following hashing algorithms:

a) Quadratic Probing. Here, h(k, i) = (h(k) + 3*i + i^2) % TableSize. Where i = 0, 1, 2, 3, ... N-1.
b) Double Hashing (you can use the hash functions discussed in the class).
    (i) Here, h(key, i) = (h(key) + g(key)) % p. Where i = 0, 1, 2, 3, ... N-1.
    (ii) h(key) = key % p. {p = TableSize}
    (iii) g(key) = q – (key % q), {2 < q < p; you can assume q = p - 13}

For each hash techniques we need to create a separate class (class QuadraticHashTable, and class DoubleHashTable).

Each class will have a constructor to initialize the needed instance variables. In addition, each class should have the insert (Student newStudent) method that will allow a Student object to be added in the hash table. In addition, the class will have the find (Student findStudent) method that will return true/false depending on whether the Student object has been found or not in the hash table.

Deliverable 4: In order to test each hash table class, provide a main method. Inside the main method, for each hash table object (QuadraticHashTable object or DoubleHashTable object).

a) Use 100 Student objects (load the first 100 objects from the file) and insert them to the hash table one at a time.
b) In addition, show a successful find operation (create a Student object that already exists in the hash table and search for this Student object) in the hash table and determine whether the search was successful.
c) Show an unsuccessful find operation (create a Student object that does not exist in the hash table and search for this Student object) in the hash table and determine whether the search was unsuccessful.
d) Provide two screenshots (insert, find) for each of the two hash algorithms.
e) Attach the source program files (QuadraticHashTable.java and DoubleHashTable.java) with your submission.

Deliverable 5: (QuadraticSimul.java and DoubleSimul.java)

For each algorithms, we create a separate program file to conduct the following simulation. For each of the two algorithms, use different input size as specified below and note down the average number of steps required to insert these objects in the hash table. Use the following table and report those numbers from your program.

a) Insert Operation
   (i) Input size: Use the first N random Student objects from the vector (or ArrayList). {Note: generate a random integer number between {0 and vector.size()/2} and store that in the index variable.
   (ii) Now get the Student object from the vector of Students variable at the index position (in this way, the same Student object would be selected more than once and there will be collision in the hash table)}.
   (iii) Calculate the sum of all probing sequences and divide it by the Input Size to determine the average. Report it in the table.

b) Find Operation
   (i) Repeat the following process (ii, iii, iv) 1000 times and calculate the average of all the probing counts for all the find operations.
   (ii) Generate a random integer number between {0 and vector.size()/2} and store that in the index variable.
   (iii) Get the Student object from the vector of Student objects variable at the index position.
   (iv) Now, search for this Student object in the hash table. Count the number of probing sequences (it is possible that the probing sequence will be 0) and add it in a sum variable.
   (v) Divide the calculated sum by 1000 to get the average probing sequence. Report the calculated average in the table.

Algorithm	Hashtable size	Input size	insert probing count	find probing count	average step count
Quadratic Hash Table	1000	1000			
	1500	1000			
	2000	2000			
	3000	2000			
Double Hash Table	1000	1000			
	1500	1000			
	2000	2000			
	3000	2000			

Deliverable 6: By using the tabular values do the following:

a) Plot a graph showing the average probing sequences for the insert operation for the two algorithms.
b) Similarly, plot a graph showing the average probing sequences of the find operation for the two algorithms.
c) Which hash table algorithm requires the least number of probing sequences to store the Student objects? Comment why that is the case.
d) Compare the performance of both of the two hash table algorithms when the input size was 1000 vs Hashtable size 1000 and 1500.

Use a report to include all the deliverables and export the document as a pdf file. Your submission would be the pdf file containing the deliverables and the two program files (one program file for each of the two hash programs).

Submit the solution on the BrightSpace website by 11:59pm, Friday, 3 May 2024. Let me know if you have any questions. Thank you.

Assignment rubrics | Total points: 80	Points
Deliverable 1: Student class has been created with the required instance variables. Method has been added to compute the hash value of the student object. Method has been provided to compare two Student objects.	10 points
Deliverable 2: lines from the file have been processed correctly to create a vector (or ArrayList) of Students objects	8 points
Deliverable 3: hash table classes have been created with constructor, insert, and find methods (you can add other methods as needed)	24 points
Deliverable 4: for each hash table class, driver main method has been provided to test the hash functions. (screenshots + program files have been provided)	8 points
Deliverable 5: for each hash table class, insert and find methods have been called by using the given input/hash table size in two simulation programs. The number of probing sequences to complete these operations have been recorded in the table	20 points
Deliverable 6: plots have been drawn to show the the comparison of the two hash algorithms	10 points

Assignment rubrics | Total points: 80	Points
Miscellaneous	-10x: Uses algorithms that has not been taught in the class -10: Does not use JavaDoc comments throughout the program code -10x: the program file has debugging/ compiler error