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.

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. In C++ overload the == operator so that two objects of the Student class can be compared and overload the () operator so that the hash value of the object can be determined. 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.

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) Chaining
b) Linear Probing
c) Quadratic Probing
d) Double Hashing (you can use the hash functions discussed in the class)

For each hash techniques we need to create a separate class (class Chaining, class LinearProbing, class QuadraticProbing, and class DoubleHashing).

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, create a hash table object. Use 100 Student objects (load the first 100 objects from the file) and insert them to the hash table.

In addition, show a successful find operation (create a Student object that already exists in the hash table and search for this Student object) and an unsuccessful find operation (create a Student object that does not exist in the hash table and search for this Student object).

Provide two screenshots (insert, find) for each of the four hash algorithms. Also, attach the source program files (.cpp or .java) with your submission.

Deliverable 5:
For each of the four algorithms, use different input size as specified below and note down the average number of steps required to insert these elements in the hash table. Use the following table and report those numbers from your program.

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. 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 duplicates/collision in the hash table)}. Note, in the insert operation, we are using only half of the objects from the vector of Student objects variable.

Similarly, generate a random integer number between {0 and vector.size()} and store that in the index variable. Get the Student object from the vector of Student objects variable at the index position. Now, search for this Student object in the hash table. Please note, we are randomly choosing a Student object from all stored objects from the vector of Student objects variable, therefore, there will be many objects that will not be found in the hash table.

Hashtable size	Input size	insert steps	find steps	total steps
1100	1000			
2100	2000			
3100	3000			
4100	4000			
5100	5000			

Deliverable 6: By using the tabular values, plot a graph for each algorithm. Plot a graph showing the steps of the insertion algorithm for all the four algorithms. Similarly, plot a graph showing the steps of the find algorithm for all the four algorithms.

a) Which hash table algorithm requires the least number of steps to store the Student objects? Comment why that is the case.
b) Which hash table algorithm requires the most number of steps to store the Student objects? Comment why that is the case.
c) Compare the performance of the hash tables when the input size was 1000 compared to 5000. Why does the insert algorithm requires more steps to store the objects when the input size is 5000 compared to the input size 1000?

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 four program files (one program file for each of the four hash programs).

Submit the solution on the Brightspace website by 11:59pm, Tuesday, 28 November 2023. 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 has table class, insert and find methods have been called by using the given input/hash table size. The number of statements that execute 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 execution steps of the four hash algorithms	10 points