Lab 3 Hash Table
Linear Probing, Quadratic Probing, Double Hashing Probing. Collisions

Assignment Overview

Write a program that reads numbers from an input file, creates hash tables, inserts the numbers into and retrieves them from the hash tables, and outputs counts of collisions for linear probing, quadratic probing, and double hashing probing. Calculate the counts of collisions separately for the input data, the same data sorted in ascending order, and the same data sorted in descending order.

Before doing the assignment, be sure to read Course Guidelines document in D2L Content.

Pseudocode

There is a partial pseudocode provided to help you with the assignment.

Function or Class	**Pseudocode File**
LinearProbingHash	LinearProbingHash-pseudocode.txt
HashInterface	HashInterface-pseudocode.txt
main (Driver)	Driver-pseudocode.txt

Hash Table Classes

Implement **LinearProbingHash** class in a language of your choice (the pseudocode is provided). Implement classes **QuadraticProbingHash** and **DoubleHashingProbingHash** similar to **LinearProbingHash** class.

If you are comfortable with it, write generic (template) classes. Otherwise just write classes that work with int or Integer keys and values.

If you are comfortable with it, write the classes that extend or implement **HashInterface** (note that the driver pseudocode uses **HashInterface**). **HashInterface** class can be implemented as an interface or an abstract base class, if the language requires it. All concrete hash classes may extend or implement **HashInterface** class.

If you prefer, to make the code less redundant, you may also write **HashBase** class (no pseudocode provided) and move all functions except lookup and constructor there.

Driver (main)

Write main driver class/function that uses all your hash classes, tests them, and calculates and prints number of collisions in various scenarios.

The driver has to do three passes. In the first pass, the driver has to read an input file and test all three hash classes using the input data. After it, in the second pass, the driver has to sort the data in ascending

order and run the tests again. And finally, in the third pass, the driver has to sort the data in descending order and run the tests one more time.

In each pass, the driver has to create three new hash tables and put all keys and their values into the hash tables, retrieve the values, check that the values are correct, and print results including collision counts.

When putting keys and values in a hash table, set *value* that is equal to *key**2. When getting the values by the keys, check that the retrieved *value* is equal to *key**2, and throw an exception otherwise.

Calculate the count of collisions separately for each pass and for Linear Probing, for Quadratic Probing and for Double Hashing Probing. The collision count shall be initialized to zero in the hash class constructor and incremented every time a collision happens in get() or put() functions.

Test keys and values

You can assume that the keys and values in the hash tables are integers. If the language allows it, use **int** type (or hint)**,** or **Integer,** or similar wrapper type.

If your language allows it and you feel comfortable with it, you may write a generic (template) class. Note that some languages do not allow **int** as generic parameter type, you will need the wrapper type in such a case.

Input and Output Files

Each input file contains the integer keys separated by whitespaces. All keys have to be read and processed until the end of file.

	Correspondent files	**Correspondent files**	**Correspondent files**	**Correspondent files**
Provided test input files	in150.txt	in160.txt	in170.txt	in180.txt
Provided sample output files	out150_collisions.txt out150_tables.txt	out160_collisions.txt out160_tables.txt	out170_collisions.txt out170_tables.txt	**none**
Output files to produce	out150_collisions.txt out150_tables.txt	out160_collisions.txt out160_tables.txt	out170_collisions.txt out170_tables.txt	out180_collisions.txt out180_tables.txt

Use the provided files to test your program. Run the program using all four test input files to produce eight output files and submit the input and output files together with the source code.

The format of the produced output files shall be similar to the provided sample output files. The collision numbers have to be the same. The tables have to be the same. If the numbers are different, you have a problem. Fix it before submitting the program.

hash() function

The hash() function has to return `(key>>8)|((key&0xff)<<16)`. While each language and platform have some default hash function, unfortunately, they vary between languages and platforms. If you use the default hash function, the number of collisions will differ from the counts in the provided test output files.

Algorithms

The hash table shall be a dynamic array (vector) of references (pointers) to hash records. Each element of the array is either a reference (pointer) to some record [key, value] or null (nullptr).

Use hash table size 191. In double hashing probing, use R=181 in hash2() function (it is a prime that less than the hash table size 191).

Inside each class, you have to use one probing algorithm in a systematic way. For example, in **LinearProbingHash**, you use only linear probing to insert and retrieve all data.

Put() and get() functions have similarities and differences. Both functions need to find a location in the hash table matching the given key. As it is a common part, it shall be written as a separate function called lookup().

The hashIndex() function adjusts hash() value so it can be used as the hash table index.

The lookup() function uses hashIndex() value as an index into the hash table. The location is found if the hash table record is empty or has the given value.

Otherwise, when the record value does not match, it is called a collision. The lookup() function does not give up but tries another location using specific probing for the hash table, and so on, so on. The lookup() function stops when the location is found or all table is checked.

Note that a collision happens every time the probed location has non-matching value. For example, if lookup() function has to try multiple locations and location 1-5 do not match, but location 6 matches – there are 5 collisions.

After the location is found, the get() function returns the found value or null (if the record is empty). The put() function updates the hash record with the given value (and create the record, if necessary) or throws an exception if the table is full (all table is checked but no location found).

Grading

In addition to Course Guidelines, there is the following substantial requirement

- In the produced output files, the counts of collisions have to be correct (see the provided sample output files).

Final thoughts

It is not easy assignment, but as soon as you understand how Linear Probing, Quadratic Probing, and Double Hash Probing work, and how to save records in the Hash Table and retrieve them, you will be able to use it in your work. And understanding how it works you can get only by thoroughly doing it.

Best Luck!