Data Structures	Assignment 5
CCO 2103-01, Spring 2024	Due at 5:00pm, Wednesday, June 19

Hand in your solutions electronically using LearnUs. As was announced in class, you are not allowed to use any data structure libraries, including the linked lists, stacks, queues, trees, binary search trees, priority queues, heaps, graphs, union-find data structures, hash functions, hash tables, and other data structures provided by JDK. When in doubt, contact the course staff.

This assignment is *not* a completion points assignment.

Submit your source code(s), zipped as `yourStudentID.zip`. For example, if your student ID is `2023000000`, then you must zip all your source code(s) into `2023000000.zip` and submit this file. Each class should have its own `.java` file, of which the filename is the same as the class name. Do *not* include your student ID as part of the class names.

This assignment consists of one programming task.

(1) (100 points) Write a class that provides a good hash function.

Your code must have a class named `HashFunction`. It must have the following constructor and method:

- `public HashFunction(int m);`
  Constructs a new instance of `HashFunction` that provides a hash function whose return value is a nonnegative integer ***strictly*** smaller than `m`. You can assume that `m` is a prime number no smaller than `269`.

- `public int hash(BigInteger k);`
  Returns the hash value of the key `k`. The key value `k` is a big integer, represented by `java.math.BigInteger`. The return value must be a nonnegative integer that is strictly less than the `m` that was given when the object was constructed. That is, the smallest hash value allowed is `0` and the largest allowed is `m-1`. You can assume that `k` is nonnegative.

Note that, if we repeatedly call `hash()` on the same instance of `HashFunction` with the same key value, the return values must be equal. This must be the case even when `hash()` is called with *distinct* instances of `java.math.BigInteger` holding the same value.

You must implement your own hash function. This, in particular, means that you are ***not*** allowed to use the `hashCode()` method of any class. You are, however, allowed to use the other methods of the class `BigInteger`.

Your code must run within a reasonable amount of time on the TA's computer.

You do not need to provide an entry point to your code. Our grading program will use your class to run (possibly multiple) hash tables and check if the hash tables work correctly. To ease your testing, we will provide a sample hash table that uses your class. Do not include this sample code in your submission.

The sample code implements a hash table, each entry of which has a `BigInteger` key value that is associated with a `String` "payload." The sample code has two classes, `HEntry` and `HTable`. The class `HEntry` represents each entry of the hash table, and has the following fields.

- `public BigInteger key;`
  Holds the key.
- `public String payload;`
  Holds the payload string associated with the key.

The class `HTable` implements a hash table. It has the following methods.

- `public HTable(int size);`
  Constructs an empty hash table. The parameter `size` must be a prime number no smaller than `269`, specifying the size of the hash table.
- `public boolean insert(BigInteger k, String p);`
  If the table is full or `k` already exists in the table, does nothing and returns `false`. Otherwise, `k`, along with the payload `p`, is inserted to the table and `true` is returned. The parameters `k` and `p` must ***not*** be `null`.
- `public String query(BigInteger k);`
  If `k` exists in the table, returns the associated payload. Otherwise, returns `null`. The parameter `k` must ***not*** be `null`.

The hash tables used by the grading program are similar to this sample but not exactly the same.

Submit all your source files, including `HashFunction.java` and any other source files you created.

(Extra credit) (20 points)

Try to make your hash function as "good" as possible.

For each test case, your code (in conjunction with the grading program's hash tables) will be run three times and the smallest running time will be considered as your code's running time for that test case. Only the solutions that passed all test cases are eligible for an extra credit, determined as follows:

- For each test case, let $t_{\min}$ be the smallest running time in milliseconds of an eligible solution. (We may add the instructor's solution to the pool of eligible solutions.) Your score for that test case is $(t_{min} + 10)/t$, where t is your running time in milliseconds.
- Your overall combined score is determined by your worst score for a test case. That is, among your scores for the test cases, the smallest one becomes your overall combined score.
- Solutions with overall combined score .99 or greater are given an extra credit of 20 points. We may lower this threshold so that at least 5% of the eligible solutions receive 20 points.

- Solutions with overall combined score .85 or greater are given an extra credit of 10 points. We may lower this threshold so that at least 15% of the eligible solutions receive 10 points or higher.
- Solutions with overall combined score .60 or greater are given an extra credit of 5 points.