Due : Nov. 14 (11/14), 13:00
Late submission due: Nov. 15 (11/15), 13:00

Overview

This assignment contains one part.

General Notes

- *Read this homework guideline carefully.* If you do not follow the guidelines, you may receive a 0 regardless of whether your code works or not.
- Do not use any IDEs (Eclipse, IntelliJ IDEA, etc.)
  - We recommend Sublime Text (Linux/Mac/Windows), Atom (Linux /Mac/Windows), Notepad++ (Windows), or TextWrangler (Mac).
  - IDEs often create a "package" of your code, which breaks the autograder.
  - **If you know how to fix the package problem**, you can use any IDE you want. However, we will not answer any questions related to this problem since we have already recommended a solution.
- Do not change any method or class signatures. If your code changes any class or method names or signatures, you will receive an automatic 0.
- Make sure your code compiles. Non-compiling code will automatically receive a 0. If you have a problem that is causing you to not be able to compile, it may be better to just comment out the incorrect code and return a dummy value (something like null or -1) so the rest can compile.
- To ensure that your code will be accepted by the autograder, you should submit your code on YSCEC, download it again, recompile it and check the provided test suite. This way, you know that the file you are submitting is the correct one.
- You can use any course materials. However, if you do not cite the source you referred to, it might be checked as copied code. Please write the material (and page) you referred in comments.

1 Hash Table

In computing, a hash table (hash map) is a data structure that implements an associative array abstract data type, a structure that can map keys to values. A hash table uses a hash function to compute an index, also called a hash code, into an array of buckets or slots, from which the desired value can be found.

1.1 Hashing

In this assignment, you will build a hash table of an arbitrary type (given at the construction) in Hash.java file. For simplicity, we will only store the key in the hash table. For the hashing of the key, use the hash method interface in the IHashFunction.java file. If a collision occurs, use linear probing to avoid the collision. Also, you must resize the hash table if the table has too many items. When the resize happens, you must rehash the table. The rehashing order starts from index 0 to the last index ot the table at the time of the rehash.

You must implement the following methods as well as its constructor in the Hashing.java file:

`put`: Insert the key into the hash table. If a collision occurs, use linear probing to avoid collision.

`remove`: Find the key and delete the key from the hash table. If the key does not exist, raise an exception.

`exists`: If the key exists in the table, return **true**. Otherwise, return **false**.

`size`: Return the number of keys in the hash table.

`tablesize`: Return the size of the table.

`show`: Return the keys in the hash table as an array. The index of each key in the list must be the same index of the bucket index of the key in the hash table. If a bucket has no item, assign **null** on that list index.

1.2 Bounce Hashing

Bounce hashing is a scheme in computer programming for resolving hash collisions of values of hash functions in a table, with worst-case constant lookup time. Bounce Hashing insert a new key into a hash table that may push an older key to a different location in the table.

Bounce hashing uses two hash functions (h_1, h_2) to store the element based on the following algorithm:

1. To insert an element x, start by trying to put x into the hash table T.
2. If $h_1(x)$ is empty on T, place x there.
3. Otherwise, place x there, evict the old element y, and try placing y into the table with the other hash function. (If $h_1(x) = h_1(y)$ use $h_2(y)$. Otherwise, use $h_1(y)$)
4. Repeat this process, bouncing between the two hash functions until all elements stabilize.
5. If this process makes a cycle, you should resize the table and perform a rehash by inserting all elements back into the table using new hash functions h_1, h_2. The rehashing order starts from index 0 to the last index of the table at the time of the rehash.
6. After inserting x, if a resize is needed, resize the table and rehash.

e.g.)
After inserting the keys 5, 0 and 17 for the hash functions $h_1(x) = x \ mod \ (table \ size)$ and $h_2(x) = (x/(tablesize)) \ mod \ (table \ size)$, the hash table looks like this:

0	1	2	3	4
~~5~~ 0	5	17		

You must implement the following methods as well as its constructor in the BounceHashing.java file:

`put`: Insert the key into the hash table using the above procedure. If a collision occurs, use the bounce methodology to avoid a collision.

`remove`: Find the key and delete the key. If the key does not exist, raise an exception.

`exists`: If the key exists in the table, return **true**. Otherwise, return **false**.

`size`: Return the number of keys in the hash table.

`tablesize`: Return the size of the table.

`show`: Return the keys in the hash table as an array. The index of each key in the list must be the same index of the bucket index of the key in the hash table. If a bucket has no item, assign **null** on that list index.

General Directions

- Write your name and student ID number in the comment at the top of the files you submit.
- Implement all of the required methods.
- You should not import anything that is not already included in the file.
- Pay careful attention to the required return type and edge cases.
- All the codes we provide can be found in src/base directory. If you are unsure what a class/method exactly does, please refer to the code.
- You are free to implement any algorithm that you wish, but you must code it yourself. If you referred to any course materials, you must write the name of the material and page in comments. We will only be testing that your code produces a correct result and terminates in a reasonable amount of time.

Submission Procedure

You *must* make a zip file for submission using Gradle build tool (refer to Compiling section). For this assignment, the zip file will contain only the following three files:

- Hash.java
- BounceHash.java
- your_student_id_number.txt

You must rename 2020xxxxxx.txt to your actual student ID number. Inside of that text file, you must include the following text and write your name at the bottom. Please be sure to write all the following text including the last period.

In completing this assignment, I pledge that I have not given nor received any unauthorized assistance.

If this file is missing, you will get 0 on the assignment. It should be named *exactly* your student id, with no other text. For example, *2020123456.txt* is correct while something like *2020123456_pa5.txt* will receive 0.

Compiling

This assignment uses Gradle build tool to automate compiling and testing procedure. The following command will test your Java code against the provided testcases:

```
% ./gradlew -q runTestRunner
```

The following command will zip the files for your submission. The zip file will be named with your student id (the name of .txt file) and will lie in "build" directory. Be aware that the command will be interrupted if your pledge does not comply the guideline.

```
% ./gradlew -q zipSubmission
```

Since the testrunner blocks the standard output from printing, it is hard to test your code fragment while writing the code. For this purpose, we also provide an empty Main class. As this file is not for submission, you may use any features Java provides. The following command will run the Main class instead of the testcases.

```
% ./gradlew -q runMain
```

On Windows, try `gradlew.bat` instead of `./gradlew` if you met an error. Moreover, you may omit the '`-q`' option to review the compile log.

Testing

We have provided small testcases (src/test) for you to check your code. You can test your code by the means mentioned above.

Note that the testcases we will use to grade your code is `very much more` rigorous than the one provided here (and not necessarily a superset of the provided tests). You should consider making your own test cases to check your code more thoroughly.