CptS 233 Micro Assignment #4

For this micro assignment, you must fully implement the addElement and removeElement functions inside LinearHashTable. All changes to the code must take place within "LinearHashTable.java". Example test cases are provided in MA_HashTable_Add_Remove_main.java.

Instructions

If you build and run MA_HashTable_Add_Remove_main.java you'll get a suite of tests on the LinearHashTable implemented in LinearHashTable.java. These tests exercise the interfaces, and if you successfully implement the MA TODO sections all tests should pass.

Just like in prior assignments, this assignment has been delivered to your Git repo. It is on a branch called MA4-HashTable. You MUST go to the gitlab server and merge in the merge request created by this assignment. After that, you can run 'git pull' on your computer to get the new code to start working.

NOTE: If you use an IDE (Eclipse, etc) to edit your project, you'll need to put the files back into the MA4-HashTable directory once you're done working. Various IDEs like to keep their files in various directory structures and I cannot build the testing system to detect and use them all. By putting the files back into the same places when you're done working. it allows my GitLab testing code to work properly. If you don't put the files back, it won't build and test properly. Feel free to make more sub directories to work in as you see fit, but these files are in the final place they need to run from.

There's a simple build script called "Makefile" in the MA directory. This is for a build system called 'make'. If you're on Linux or have XCode installed, you can run 'make build' and 'make test' to build and test the project. This isn't required for the project to work, but it's a shortcut. Windows people can use it too, but you'll need to figure out how to install GNU make.

Your code must be added, committed, and pushed to your Git repository to be turned in. Put a small file onto blackboard to show the TA that you're done working and need to be graded.

The three functions you'll need to implement are all related to detecting tree imbalances and then the key rotations to fix it. These functions are all found in LinearHashTable.java and clearly denoted at the top of the class implementation. I **highly** recommend that you carefully read through the rest of the code to see what might be useful in your implementation.

Another thing to pay attention to is how the test code is written. Please look at how I test the different functions/API of the class. The tests add, remove, and do add-remove-add kinds of tests. If your linear probing works properly it should all pass.

Linear Probing:

This is a *Linear* probing hash table. Don't implement separate chaining nor quadratic probing. If the bucket you want to insert into is full, then add one to your index, mod by the vector size, and try again.

Lazy Deletion:

Because this is a probing-based collision resolution hash table, you need to use the lazy deletion method. Look at what's actually stored in the _items vector. It's an array of HashItem objects. See how they're setup to be used for lazy deletion and make sure your bookkeeping is all done properly when you implement addElement and removeElement.

Grading

Your submission will be graded based on the following:

1. [8] Your solution builds, does not cause any runtime issues, correctly implements "addElement" and "removeElement", and passes all test cases
2. [2] Pity points. You're welcome ;)
    - You provide meaningful variable names
    - You provide sufficient and meaningful comments
    - Your code is well structured

Due Date

See instructions on Blackboard.