Due : November 26th, 2017

Overview

This assignment consists of two parts: implementing a Cuckoo Hash and implementing an AVL Tree.

General Notes

- Do not change any method or class signatures. You should only edit inside of the functions. If your code changes any class or method names or signatures, you will receive an automatic 0. You should not implement any other functions or instance variables besides the ones that are provided, unless explicitly allowed.
- If you are using Eclipse, be sure to remove the line declaring your code a package when you submit your code. Failure to do so could result in you receiving a 0 if the autograder fails.
- 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.
- Make sure that your code does not print out anything (there should be no System.out.println in your code). You will receive an automatic 0 if your code outputs something to STDOUT during the tests.

Cuckoo Hashing

Cuckoo Hashing is a well known hashing scheme that has some desirable properties. Its name comes from the egg laying procedure of the Cuckoo bird. The bird sometimes lays its eggs in other nests. When the eggs hatch, they push out the eggs that were there. In a Cuckoo Hash, element containment and deletion are both worst-case $O(1)$ operations, while insertion can be done in *amortized* $O(1)$ time.

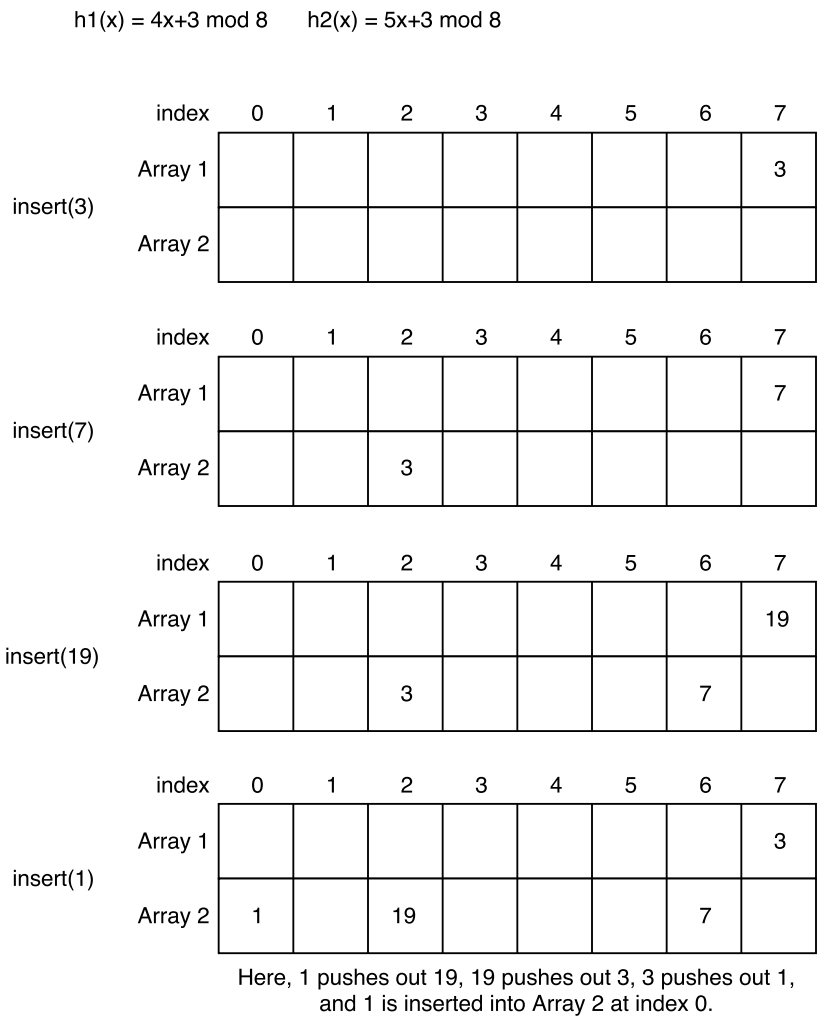
Definition

For this assignment, we will use the notation (a, b, N) to denote the function $ax + b \mod N$. In Java, the mod symbol is %.

A Cuckoo Hash is defined by an integer $N > 1$, two hash functions h_1 and h_2 both in the form (a, b, N) (but not necessarily the same), and two constants, $0 < threshold \leq 1$ and $chainLength > 0$. The Cuckoo Hash contains two arrays of size N, A_1 and A_2, which are initially filled with some default value (in this homework we will use 0). When an element, x, is inserted, we insert it into position $h_1(x)$ in array A_1. If there is an element, y, already in this position, we bump y out, insert x, then insert y into position $h_2(y)$ of array A_2. If there is already an element, z, at *that* position, we bump z out, insert y, then insert z it at position $h_1(z)$ into array A_1. This repeats until one of two situations happens:

- An element is placed in a previously unoccupied spot.
- An insertion causes a chain of bumped elements of length at least $chainLength$.

If we encounter a sufficiently long chain or we meet the threshold $\frac{e}{2N} \geq threshold$ (where e is the number of non-zero elements currently in the hash), we perform a resizing operation. To resize, we set $N' = 2N$ and choose two new hash functions, h_1 and h_2 both in the form (a', b', N'), where a' and b' are not necessarily the same as the previous a and b. We then reinsert (with the new hash functions) every element into two new tables, A'_1, A'_2, both of size N'.



Implementation Notes

- You can use `Random().nextInt(int bound)` to generate random integers between 0 and $bound - 1$.
- For this assignment, we will only be inserting positive integers into the Cuckoo Hash.
- The chain length is defined by the number of bumped out elements during an insertion. So, if you insert an element and it immediately finds an open spot, that chain is length 0. If you insert an element and it bumps an element out

which then bumps another out before finding an open spot, that chain is length 2. We activate the resize as soon as we reach *chainLength* bumps.

- If after inserting an element the threshold $\frac{e}{2N} \geq threshold$ would be met, resize the Cuckoo Hash and then insert the new element. For example, suppose $N = 2$, $threshold = .5$ and e is currently 1. Since if we inserted a new element we would have $\frac{e}{2N} = \frac{2}{4} = .5 \geq threshold$, we should resize the Cuckoo Hash and then insert the new element.
- Your hash functions must satisfy $0 < a < N$, $0 \leq b < N$.

Directions

- Write your name and student ID number in the comment at the top of the CuckooHash.java file.
- Implement all of the required functions in the CuckooHash class.
- You are not allowed to change the format of this class. You must only use the methods and instance variables provided.
- Pay careful attention to the required return types.

AVL Tree

The second part of this assignment is implementing an AVL Tree, which was discussed in class.

We will be using a slightly different version of the AVL Tree than what you saw in class. We will be using the convention that the balance factor of a node is defined as $height(right) - height(left)$ where left and right are the node's left and right children. Additionally, in class you discussed trees that have a key and a value associated with each node. In this, the key and value are the same and are the same as the value stored in the *data* variable of a node. So, you should determine where to place nodes based on the data stored in the *data* variable.

Implementation Notes

- The height of a leaf is 0. The height of a null child is -1.
- The Balance Factor should always be one of $\{0, 1, -1\}$.
- You should implement the operations as defined in the lectures. Be sure your AVL Tree maintains balance after insertions and deletions.

Directions

- Write your name and student ID number in the comment at the top of the AVLTree.java file.
- Implement all of the required functions in the AVLTree class.
- You are allowed to implement your own helper functions, but you **may not** add any instance variables.
- Pay careful attention to the required return values.

General Directions

- Write your name and student ID number in the comment at the top of the java files.
- Implement all of the required functions in the java files.
- You should not import anything that is not already included in the file.
- Pay careful attention to the required return types and edge cases.

Submission Procedure

To submit your homework, simply drag and drop each file individually into the attachment box on the YSCEC assignment page. You should **NOT** zip them or place them in any folder. For this assignment, you should submit only the following files:

- CuckooHash.java
- AVLTree.java
- ⟨student_id⟩.txt

Inside of the ⟨student_id⟩.txt file, you should include the following text, and write your name at the bottom.

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

If this file is missing, you will get a 0 on the assignment. It should be named *exactly* your student id, with no other text. For example, `2017123456.txt` is correct while something like `2017123456_hw1.txt` will receive a 0.

Testing

We have provided a small test suite for you to check your code. You can test your code by compiling and running the tester program. You will always get a notification if you fail any of the tests, but you may not get a notification if you passed.

Note that the test suite we use to grade your code will be 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.