CSE 12 Getting Started HW 6 --- HashTable

Overview

In this PA you are going to implement a HashTable that `insert` and `lookup` elements.

Why Hash Table? Or Why Hash?

- The question is this, how much does it take to look up an element in an array?
  - $O(n)$.
- How much does it take to maybe look up in a well-structured array (maybe sorted)?
  - $O(log(n))$
  - It is okay, but that still cause a lot.
- However, what about hash table.
  - If there is no collision, then it is $O(1)$ other than the time that spent in hashing.
  - So if you have a huge space to utilize, a tons of things to store, and everything is relative easy to hash, then Hash table is very very efficient.
- To achieve that, you need to 2 methods to Insert and lookup.
  - However, they both need to locate the element.
    - Because when you have a hash value, you need to do something with it.
- These are the main functional functions you need to do for this PA.

Functions to write

Driver.java

- Appropriate methods for UCSDStudent
- Compare it to the `Variable` class that we provided in the code page.

HashTable.java

- `debugOn()`
  - Very easy, check what we have done before in the `List` (not `ListEngine`) of your homework 5
  - One line of code, don't overthink
- `debugOff()`
  - Same
- `HashTable(int sz, String caller)`

    - Correctly initialize all the member fields.
    - You need to keep track of the static `counter` to see how many tables you have.
- `jettison()`
    - You should already be pretty familiar with it
    - Easier than `hw5`
    - But you do need to jettison each element in your table.
- `getOccupancy()`:
    - Return occupancy
- `insert(Base element, boolean recursiveCall)`
    - This function is a recursive function.
    - `recursieCall` starts from `False` when being called from `Driver.java`
    - You shall check whether you insertion can be done
        - When is it not a legal insertion?
    - You shall call `locate`
    - Before calling `locate`, if it is not a recursive call, you should set `index = -1` or `count = 0` depends on which approach you choose.
        - You should use what `locate` returns to check whether it is duplicate insertion
        - You need to remove the old item and put a new one there
        - Your old item will never be used, so what?
        - Also calling `locate` will help you setup your `index` variable and/or `count` variable.
    - After your `locate` find you a place
        - If it is an empty spot, you just put your `element` in there, and you done!
            - So this is the base case of your recursion.
        - The recursive step happens if you found a spot, but it is already taken by an element.
            - You need to bump that old element out and reinsert the old element back to the table.
                - In your recursive call before calling `locate`, you should setup `index` or `count` so that you can catch up.
- `locate()`
    - This function should determine where to `lookup` or `insert` given an `element` in the `hashtable`.
    - You shall use the formula that you learned from class to determine what is the original hash value and what is the increment.
    - **NOTE:** If the element that you are `locating` is `bumped` from its old spot, your initial index need to "catch up".
        - Example:
            - Preobsequence = 2, 4, 1, 3, 0
            - If index = -1:
                - Index = original

        - `index = (index + increment) % table size`
        - Index = Hashvalue % table size
        - 4 = `index = (index + increment) % table size`
        - Start my prob sequence at `count = 0` `index = 2`
      - If index == 4:
        - When I was bumped, my prob sequence says I was bumped from 4
        - `index = (4 + increment) % table_size`
      - That is you should not start from the original hash value if the `locate` is called recursively in `insert`.
    - **Hint:**
      - You can use `increment` and `index` or `count` to determine what is the hash value that you should start with.
      - Your `insert` and `lookup` should setup the `index` or `count` up before calling `locate`.
  - Then just looping through until you find a place to place the current element.
    - Because it is a bully algorithm, you should compare the two elements by calling the `isLessThan`.
    - You should think of both `insert` and `lookup` when writing this function.
      - What is the condition that you encounter implies your table don't have the element that you are looking up?
      - But what does the same condition mean if you are inserting?
      - This should help you determine what is the return value of your `lookup`.
- `lookup()`
  - You shall call `locate` as well.
  - This is a super easy comparing to the `insert`, so don't overthink.
    - But you should set `index = -1` or `count = 0` before locate, because you should think of it as a base case.