CS 345 (Spring 18): Computer Organization

Project #5

Red-Black Trees, Hash Tables

due at 5pm, Tue 1 Mar 2018

1 Introduction

It's time for the last project! We'll be implementing two more dictionaries: a red-black tree (insertion only) and a hash table. Again, I encourage you to use your solutions for Project 3 and 4 as the starting point for the red-black tree; of course, a hash table is very different.

Most of the structure of this project will be familiar; we are using the same Dictionary interface as Project 4 (I haven't even renamed the file); I'll provide a `TestDriver` class, and the testcase format will be the same.

However, you'll notice that this class does not require you to write a `GenDotFile` class (and also, I haven't provided a `TestDotFile` class). This is because I'm not requiring you to produce `.dot` files. I **strongly encourage** you to do so for your red-black tree, but I won't grade you on it.

1.1 `x=change(x)`

Once again, I'll be requiring that you use `x=change(x)` - although (obviously) this only applies to the red-black tree. As with the AVL tree, you'll need to do some "fixup" of the tree after each insertion. (In a red-black tree, you will also need to do some special fixup which only applies to the root.)

1.2 Skippable Functions

Your red-black tree must implement the `Dictionary` interface - meaning that it must contain every method. However, the following functions may be stub functions (that is, they may do nothing and simply return):

- `delete()`

  Since we haven't looked at how to perform deletion on a red-black tree, it won't be required for your program.[1]

- `genDebugDot()`

  My test driver **will** call this function if you run a testcase with `debug` turned on, so your implementation should do something reasonable. However, if you want, this can be an empty function.

[1] My grading script won't test your red-black tree on any testcase that includes delete, so if you want to experiement with `delete()`, it won't break anything.

NOTE: I think that skipping this function would be a **terrible idea!** Even I needed my `.dot` file to debug my code - so you should expect the same for your code. But I won't grade you on this.

Your hash table must implement the `Dictionary` interface. However, the following functions may be stub functions:

- `genDebugDot()`

  It would be kind of strange to draw a hash table using `.dot`, and I didn't choose to implement this in my own code. However, you may do so if you want. I'd be interested to see what you come up with!

While you **are** required to implement all three of the `print*Order()` functions, they should all print the exact same information (probably, you should have two of them simply call the third). I'll detail what is required for this later.

1.3 Required Filenames to Turn in

You must turn in the following files:

```
Proj05_RedBlack_student.java
Proj05_HashTable_student.java
```

2 Style Requirements

2.1 No Global Variables

You've seen this before.

2.2 Red-Black Tree: Must use `x=change(x)`

You **must** use the `x=change(x)` style in the red-black tree for `insert`.

In addition, you must write rotate helper functions (both right and left) which use the `x=change(x)` style[2]. They'll be useful to you!

Finally, you must write a helper function in the `x=change(x)` style which looks for red-red problems (after an insert), and fixes them. See below for details.

3 Design Details

3.1 Red-Black

Your red-black tree must:

- Support (key,value) pairs, just like Project 3. Duplicates are still illegal.

[2] You can choose whether or not these functions recolor the nodes. I choose not to do so, and instead to recolor the nodes from the caller. It worked well.

- Use the bottom-up style for fixing red-red problems.
- `insert,search` must both run in $O(\lg n)$ time. **Never perform any global, brute-force search or scan of all of the nodes.** (Except for a traversal, of course.)
- The various traversals must run in $O(n)$ time.

3.2 Hash Table

Your hash table must:

- Support the same (key,value) pairs. It also must not allow duplicates.
- Must support `delete()` (unlike the red-black tree).
- Must initialize the hash table with exactly 4 slots[3].
- Must maintain a count of the number of (key,value) pairs in the table.
- Must double the size of the table any time that the number of pairs is **half** the number of slots in the table. (You are allowed, but not required, to shrink the table back down in `delete()`.)
- Include a helper function[4] which implements the hash function:

```
hash = (key * 3721) % lengthOfTable;
```

- Use the linked-list style of hashing; I have provided the `HashTableEntry` class for your use.

3.3 Traversals of the Hash Table

As we've discussed in class, hash tables are **terrible** for traversals! They take time $O(n + C)$, where C is the **capacity** of the table (that is, the number of slots) instead of the number of elements n. Moreover, if you want a sorted traversal, there's no real alternative to simply extracting them all and sorting.

However, when you're implementing a standard interface, you have no choice - the method must exist. As the implementer of the class, you have a couple of options: (a) throw an "Unsupported Operation" exception[5]; or (b) provide a (slow) implementation. In this project, you must do the latter: collect all of the keys, sort them, and then print out each (key,value) pair.

[3] This is **much smaller** than you'd use in the real world. Realistic values might be 16, 256, or even 1024. However, I want to exercise your "grow the table" code a lot - so I'm making you start with a crazy-small table.

[4] Your function may be `static` or non-`static`, your choice. However, choose a style that matches your design: if you declare it non-`static`, then it **must** use read one of the fields of the object. If you don't use any of those fields, then mark it `static`.

[5] `https://docs.oracle.com/javase/7/docs/api/java/lang/UnsupportedOperationException.html`

How should you sort the keys? It's up to you, with one requirement **do not use an $O(n^2)$ algorithm!** I chose to use the `Arrays.sort()` class, provided by Java; it's simple and (I presume) a reasonably fast implementation.

The output from the traversals (for the hash table) should look exactly like the traversals from Project 4.

3.4 Red-Black Traversal Format

The output from the red-black tree traversal should look just like Project 4, except that each entry should include the color of the node. (See the output from the example class to see exactly how this should look.)

4 Other Notes

4.1 Red-Black Hints

The following are hints (not requirements) about how to implement the red-black tree. I'd encourage you to learn from my experience here, but I won't grade you on them.

- Write a helper function which checks to see if a node is red or black; make it static, so that it can handle `null` pointers.
- In your function that deals with red-red problems, return immediately (that is, make it a NOP) if the node you're checking is red; instead, only do this work when you're at a black node.

  It's a **lot** easier to fix up the tree if you can assume that you see the entire widget!
- Write a helper function which recolors a node, and both of its children. This turns out to be useful in several different cases.
- Don't try to handle the root node as a special case in your fixup function; instead, handle this **after** insert has completed, in the non-recursive wrapper function.

4.2 Hash Table Notes

I've chosen **not** to have you print out a lot of internal information about the structure of the hash table - meaning that my testcases won't be able to check a lot of the internal details I required above (such as the hash function, and the resize requirements).

I've done this to make things a little easier on you - but beware that the TAs will still go and inspect your code for (some) of these requirements. So while it might be tempting, don't try to skip them!

But, on the other hand, if I don't specify how to handle some case, you are free to do as you wish. For instance, I don't tell you whether `insert()` should insert at the head or tail of the linked list; either is acceptable.

5 Base Code

Download all of the files from the project directory

`http://lecturer-russ.appspot.com/classes/cs345/spring18/projects/proj05/`

If you want to access any of the files from Lectura, you can also find a mirror of the class website (on any department computer) at:

`/home/russelll/cs345_website/`

6 A Note About Grading

Your code will be tested automatically. Therefore, your code must:

- Use exactly the filenames that we specify (remember that names are case sensitive).
- **Not** use any other files (unless allowed by the project spec) - since our grading script won't know to use them.
- Follow the spec precisely (don't change any names, or edit the files I give you, unless the spec says to do so).
- (In projects that require output) match the required output **exactly!** Any extra spaces, blank lines misspelled words, etc. will cause the testcase to fail.

To make it easy to check, I have provided the grading script. I **strongly recommend** that you download the grading script and all of the testcases, and use them to test your code from the beginning. You want to detect any problems early on!

6.1 Testcases

You can find a set of testcases for this project at
`http://lecturer-russ.appspot.com/classes/cs345/spring18/projects/proj05/`

See the descriptions above for the precise testcase format, and also for information about how to run the two test driver classes.

6.2 Other Testcases

For many projects, we will have "secret testcases," which are additional testcases that we do not publish until after the solutions have been posted. These may cover corner cases not covered by the basic testcase, or may simply provide additional testing. **You are encouraged to write testcaes of your own, in order to better test your code.** You are also encouraged to share your testcases on Piazza!

6.3 Automatic Testing

We have provided a testing script (in the same directory), named `grade_proj05`. Place this script, all of the testcase files, and your program files in the same directory. (I recommend that you do this on Lectura, or a similar department machine. It **might** also work on your Mac, but no promises!)