Introduction to Data Structures (cs106)

Lab *: Dictionary Representations

In this lab, we will explore two representations for dictionaries. Begin by checking out the dictionaries project; it includes one partial implementation that is based on recording a list of all the pairs in the dictionary. Your job is to complete this implementation (and its test suite), and then to create a second representation that is based on a binary tree representation.

First, complete the implementation of Dictionary1:

- add an __eq__ method and a __repr__ method according to the usual rules for these operations
- add a "merge" method to combine two dictionaries into one that contains all the keys from either of the original two
- add tests for the above to the test suite, and test your algorithms
- provide axioms for merge (axioms are not required for __eq__ or __repr__)
- do *not* change the original axioms or tests that came with the starter files

If you wish, you may change the algorithms for the existing methods of Dictionary1, as long as they still obey the axioms and tests that came with the starter files. You may also add other methods if you wish to do so; you should give tests for any that can be used outside of the class, but don't have to provide axioms unless the axioms for some other method (lookup, put, or merge) refer to one of the new operations.

Then, implement class Dictionary2:

- in Dictionary2, each dictionary must be represented by a binary tree. There are a number of ways this can be done; you are free to choose any approach that both
    a. has the same axioms, and passes the same tests, as Dictionary1 (though it is fine if the repr method produces output in a different order … just put a comment by the test if this happens), and
    b. uses the BinaryTree class (from your 20Questions lab or Miller & Ranum) as the representation
- As different representations of the same abstraction, Dictionary1 and Dictionary2 should provide the same operations and produce the same abstract results --- in other words, any correct program that uses dictionaries according to their axioms should work with either class (though the two classes may require different resources, e.g. one might be slower and the other faster).

In the next lab, we will explore performance issues in detail, but this week's work will be focused entirely on correctness. If you wish to do so, you are welcome to run the Dictionary-speedtest.py program to compare the relative speeds of the two implementations, but this will not become important until the next lab.