Advanced Programming

Samir Datta*

February 28, 2018

Instructions

---

1. Modify the attached file assignment.2.py as *yourusername.2.py*. For example, I would have used aalok.2.py.
2. The file that you submit should be a modification of the attached assignment.2.py file, i.e, it should contain the functions that have already been defined in this file. You may want to define more, but the given ones, including main must be there.
3. The functions provided will be treated as explained in the problem statements and the comments. You are free to change the names for the arguments, but the code will be tested by running the main function in the file (after we add some more test cases).
4. The *return* type of each function should be as specified in the assignment.2.py file.

Problem 1

Implement a Binary Search Tree that lets you keep a count of the elements inserted.

To do this, you would first implement a `Node` class that has the following fields:

- `.rightChild`: A reference to a node which is its rightChild. Should be set to `None`, if there is no right child.
- `.leftChild`: Similar to `rightChild`.
- `.key`: The corresponding `key` in that node.
- `.value`: The corresponding `value` in that node. This is supposed to keep track of the number of times a particular element has been inserted in the Binary Search Tree

*Teaching Assistants - Kishlaya Jaiswal, Agnishom Chattopadhyay

The node class should also have a constructor, a `.increment`, a `.setLeft`, and a `.setRight` method, as shown in the template.

The search tree is expected to have the Binary Search Tree Property, i.e, every key to the left of a node must be smaller than the key of the node and every key to the right of the node must be larger than the key of the node.

The search tree should have the following methods:

- `BinarySearchTree(initialList)` (The constructor): `initialList` would be a list of `(key, value)` pairs. This constructor would create a search tree whose keys are the given `key`s and the values are the corresponding `value`s. You are guaranteed that `initialList` would not repeat keys.
- `.count(key)`: Returns an `int`. The number of times the `key` was inserted into the BST. Also returns 0 if the `key` is not present.
- `.insert(key)`: Increments the count of the given `key`. If the given `key` is not there, it adds the `key` and sets its count to 1.

Problem 2

Two words are anagrams of each other if their letters can be rearranged to obtain each other.

This problem will proceed in two steps:

- You will be given a bunch of words which represents all the words in some language (not necessarily English!)
- Then, you will be asked a series of queries, each of which are strings, for which you will have to tell the number of anagrams each string has which are part of the word list described in step 1.

The code for the solution to this problem in this class will be organized in a class called the `AnagramProblemSolver`. It'd have the two following methods:

- `AnagramProblemSolver(wordList)` (The constructor): This would take in a list of strings `wordList`, preprocess it, and get ready for the queries.
- `.query(word)`: Returns an `int`. Calling this method would be tell the number of anagrams of `word` in `wordList`, where `wordList` is the list provided when constructing an object of this class.

You are expected to use the data structure from Problem 1 to solve this problem in an efficient way.

Everytime I climbed up the tree, I fell down.
Do you know why?