CS532: Homework 7

Web Resource: Trie Trees

We eventually want to retrieve all of the data using the prefix5 command. To get there, it is advantageous that when you are retrieving data, you build a trie tree. A trie tree is a rooted tree allowing for an unbounded number of child, unlike a binary tree that can have at most 2. The figure below (taken from wikipedia) gives an example. Note that the tree is built so that each unique prefix is kept as a separate node. Since all words start with ‘r’ but split after that, ‘r’ is a node value. After that, all words continue with ‘om’ or ‘ub’, so that is split as well.

Our Trie trees will be a bit different from the one in the picture: nodes will have the string (rather than the arcs). The trie tree should always have a root whose string is “”. Siblings will be in sorted order (e.g., node ‘an’ will be before node ‘ulus’).

Below, I have included code that you must use. You will use a class for the nodes. I have given you the initializer. A node has a parent, first child pointer, and a sibling pointer. So, all children of a parent node are kept as a singly-linked list through the sibling link. The parent accesses the children through the child pointer, which points to the first child. All children point to their parent.

The code also includes some testing code, which your code should work with. The test code also helps demonstrate how the new routines should work.

In the next 3 questions you will add some routines that give basic support for trie-trees: addchild, find and get_word. These routines will be used in a future homework to build more user-friendly routines, which will be part of a future homework.

class Node:
    def __init__(self, value):
        self.parent = None
        self.child = None
        self.sibling = None
        self.value = value

    def add_child(self, other):
        raise NotImplementedError
def find(self, string):
    raise NotImplementedError

def walk(self, prefix=""):
    prefix += self.value
    if self.child is None:
        return [prefix]

    results = []
    ch = self.child
    while ch is not None:
        results.extend(ch.walk(prefix))
        ch = ch.sibling
    return results

def test():
    top = Node("")
top.add_child(Node("r"))
top.find("r").add_child(Node("ub"))
top.find("r").add_child(Node("om"))
top.find("rom").add_child(Node("ulus"))
top.find("rom").add_child(Node("an"))
top.find("roman").add_child(Node("e"))
print(top.walk())
print(top.find("roman").get_word())

if __name__ == "__main__":
    test()

Question 1: Add Child

addchild is used to add a new child to a parent node already in the tree for a given character string. For example, for the figure above, if there is already a node for ‘r’ and ‘om’ (but ‘om’ does not yet point to ‘r’), addchild can be used to add the ‘om’ node as a child of ‘r’. If there is a node for ‘ulus’, addchild can be used to add that node as a child of ‘om’. Note: this function does not need to determine how to split up a word, such as splitting ‘romans’ into ‘r’, ‘om’, and ‘ans’. We will address that in a later homework assignment.

addchild should keep the children of a node in sorted order. So when adding a new child to a parent that already has children, addchild needs to insert the child into the right spot in the single-linked list. So if ‘ub’ has been added to parent ‘r’, and then ‘om’ is added, ‘om’ needs to be inserted before ‘uh’. This is different than in the figure: in our version, ‘an’ should come before ‘ulus’ under the r-om path.

Hand in your code.

Question 2: Find

find is used to walk the tree to find the node that the string ends with. For the tree in the figure above, and if ‘top’ is a pointer to the top node, top.find(‘rube’) will walk down from the ‘r’ node, to the ‘ub’ node, and then to the ‘e’ node, and return that ‘e’ node. If the string does not exactly correspond to a path in the tree, find should return None.

Note that this find method find is specific to objects in our Node class. It is different than the find method that works for strings. Python knows which method to use based on the type of the object.
Hand in your code.

**Question 3: Get Word**

The method `get_word` is used to create the word that ends at the node. Thus, it should trace its way to the root of the Trie Tree in creating its output.

Hand your code.