CS532: Homework 9 (version 5)

Version 2: added in a new initial question to print tree.

Version 3: For split node question, I added suggestion to make the newly created node have the remainder of the string.

For add_word, I added the new constraint that you have to keep track of which nodes can be word endings. I also added in a testcase.

Added the final question, about tokenization.

Version 4: For add_word, I added the requirement that it should return the node corresponding to the end of the word.

Version 5: added an example to show how the code should work.

Trie Tree

These questions will continue the trie tree that we did in Homework 7. You must start with the solution given in homework 7 for the methods for __init__, add_child, find and walk.

Question 1: Print Tree

To help with debugging your trie tree code, make a method print_tree that prints the trie tree showing all of the values at each node. Each node should follow its parent, but indented the length of its parent’s value. Below is a sample for the tree in the figure.
Question 2: Split Node

To add a new word to a trie tree, you might need to split a prefix. For example, in the figure, if you want to add the word ‘robust’, the ‘om’ node needs to be split into a ‘o’ and ‘m’ node so that ‘bust’ can be added to ‘o’. Here you will make the routine to take the first step, which is to split a node.

Make a method called split_node that will take a node in a trie tree and a proper prefix of the node’s character sequence, it will split the node into a parent and a child node with the parent having the given prefix, and the child having the remainder.

For the subsequent questions, it will be easier if you make created node to have the remainder of the string, rather than the prefix. For the example above, in which we are splitting ‘om’, change the ‘om’ node to be just ‘o’, and have the new node have value ‘m’. This way, the node that the method is called on does not change with respect to nodes closer to the root.

Here is some sample code to show how split node should work, and its output.

```python
def test_split_node():
    top = Node('')
    top.add_child(chars('roman'))
    top.find('roman').add_child(chars('e'))
    top.find('roman').add_child(chars('us'))
    top.print_tree()
    top.find('roman').split_node('rom')
    top.print_tree()
    top.find('rom').add_child(chars('e'))
    top.print_tree()
```

```
roman
  e
 us
rom
  an
   e
 us
rom
  an
   e
 us
 e
```

Hand in the code.

Question 3: Find Partial

Make a method called find_partial that, when given the top node of a trie tree and a word, will search for the word in the trie tree. If it does not find the word in the tree, it should return (a) the node that covers
as much of the prefix as is in the trie tree, and (b) as well as the suffix of the word that is not in the tree. If
the word is in the tree, it should return the find node, just like find, but also the empty string (so we know
it was successful). Make this routine as similar to find as possible.

For example, in the sample tree, if you give it the word ‘robust’ should return the ‘r’ node, and the string
‘obust’. If you give it ‘romain’, it should return the ‘om’ node along with ‘ain’.

Hand in the code.

Question 4: Add Word

Write a method add_word that adds a word to a trie tree. If the word is already in the tree, it should not
change the tree. The method should return the node that accounts for the last part of the word.

Your method must use the two methods you just added along with add_child.

For the example tree, your routine should be able to successfully add a word like ‘rombus’, where it does not
need to split a node, and a word like ‘romaine’ where it first needs to split a node before adding in the new
suffix.

Note: for the words ‘rose’ and ‘roses’, the node that ‘rose’ ends at should have a child with a value of ‘s’. In
other words, nodes that have children can be valid word endings.

Since some nodes that have children might be words and some might not be words, add an extra field word
to the object to indicate whether a valid word ends at this node. You need to change split_node so that it
updates this value correctly, and change print_tree so that it indicates if a node is the ending for a word.

You should test your code on a variety of testcases to make sure it is correct. In particular, it should give
the following result for the following sequence of commands.

top = chars(""")
top.add_word("roses")
top.add_word("romane")
top.add_word("romanus")
top.add_word("rome")
top.add_word("rose")
top.add_word("roment")
top.add_word("roses")
top.print_tree()

Output:

ro
 m
 an
 e W
 us W
 e W
 nt W
 se W
 s W

Hand in the code for make_word, and the code any other routines that you hanged, which should be
print_tree, split_node, and __init__. 
Question 5: Tokenizing

Many languages use spaces to mark the beginning and end of words in the language. Some languages do not mark the end of words, and rely on people to determine what the boundaries are. Build a tokenizer to find all possible word segmentations.

Use the lexicon from homework 2, and build a Trie tree for it, using your prior code to retrieve the next 5 words at a time from your ‘web resource’. Call this routine `create_trie`. It should return the top node of the trie tree.

Create a routine called `tokenize` that takes as input the top node of a trie tree, and a string of words (with no spaces). It should return an array of different tokenizations. Each tokenization should be a list of words, where each word is in lexicon of hw7.txt.

The routine `tokenize` will do a breadth-first search in which the frontier will be a list of pairs: (a) an index into the input string indicating how much of the input has been processed so far, and (b) a list of words that have been tokenized up to that point. Your code should take the top element from the frontier and find all possible words to extend it by using the `find_partial` routine. That routine will find the longest match, and then you just need to add an entry into the frontier for each ancestor of it in the trie tree that is a valid word ending. Any time your code has a solution, you should add it to a list of answers, which you will output at the end. The routine should be iterative rather than recursive.

For the string ‘uponthewarmeat’, I get 5 tokenizations using hw7.txt. These are:

```
[‘upon’, ‘the’, ‘warm’, ‘eat’]
[‘upon’, ‘the’, ‘war’, ‘meat’]
[‘upon’, ‘the’, ‘war’, ‘me’, ‘at’]
[‘up’, ‘on’, ‘the’, ‘warm’, ‘eat’]
[‘up’, ‘on’, ‘the’, ‘war’, ‘meat’]
[‘up’, ‘on’, ‘the’, ‘war’, ‘me’, ‘at’]
```

To further help you, here are my internal print statements that I use for debugging: The non-indented lines are each item I get from the top of the frontier. The first indented line is the result I found from `find_partial`. The next lines are walking up the trie tree and checking if the prefix is marked as a word or not. The first word found is ‘upon’. ‘upon’ is a valid word, the parent represents the word ‘up, which is also a valid word, but the next two parents of ‘u’ and ” are not valid words. So, an entry corresponding to ‘upon’ and ‘up’ is added to the frontier.

```
0 []
  Found word ‘upon’
  Adding to frontier ‘upon’
  Adding to frontier ‘up’
  Node ‘u’ is not a word
  Node ‘’ is not a word
4 [‘upon’]
  Found word ‘the’
  Adding to frontier ‘the’
  Node ‘th’ is not a word
  Node ‘t’ is not a word
  Node ‘’ is not a word
2 [‘up’]
  Found word ‘on’
  Adding to frontier ‘on’
  Node ‘o’ is not a word
  Node ‘’ is not a word
7 [‘upon’, ‘the’]
```
Found word 'warm'
Adding to frontier 'warm'
Adding to frontier 'war'
Node 'wa' is not a word
Node 'w' is not a word
Node '' is not a word
4 ['up', 'on']

Found word 'the'
Adding to frontier 'the'
Node 'th' is not a word
Node 't' is not a word
Node '' is not a word
11 ['upon', 'the', 'warm']

Found word 'eat'
Adding to frontier 'eat'
Node 'ea' is not a word
Node 'e' is not a word
Node '' is not a word
10 ['upon', 'the', 'war']

Found word 'meat'
Adding to frontier 'meat'
Node 'mea' is not a word
Adding to frontier 'me'
Node 'm' is not a word
Node '' is not a word
7 ['up', 'on', 'the']

Found word 'warm'
Adding to frontier 'warm'
Adding to frontier 'war'
Node 'wa' is not a word
Node 'w' is not a word
Node '' is not a word
14 ['upon', 'the', 'warm', 'eat']
14 ['upon', 'the', 'war', 'meat']
12 ['upon', 'the', 'war', 'me']

Found word 'at'
Adding to frontier 'at'
Adding to frontier 'a'
Node '' is not a word
11 ['up', 'on', 'the', 'warm']

Found word 'eat'
Adding to frontier 'eat'
Node 'ea' is not a word
Node 'e' is not a word
Node '' is not a word
10 ['up', 'on', 'the', 'war']

Found word 'meat'
Adding to frontier 'meat'
Node 'mea' is not a word
Adding to frontier 'me'
Node 'm' is not a word
Node '' is not a word
14 ['upon', 'the', 'war', 'me', 'at']
13 ['upon', 'the', 'war', 'me', 'a']
Found word 't'
Node 't' is not a word
Node '' is not a word
14 ['up', 'on', 'the', 'warm', 'eat']
14 ['up', 'on', 'the', 'war', 'meat']
12 ['up', 'on', 'the', 'war', 'me']
   Found word 'at'
   Adding to frontier 'at'
   Adding to frontier 'a'
   Node '' is not a word
14 ['up', 'on', 'the', 'war', 'me', 'at']
13 ['up', 'on', 'the', 'war', 'me', 'a']
   Found word 't'
   Node 't' is not a word
   Node '' is not a word

Hand in your code for create_trie and tokenize.