CPS305 Lab 5

Topics:

- Tries

Files:

- Lab5.py
- Incorrectly named files will receive a mark of zero.

Submit Files:

- Submit file through D2L
- Your file should not have any print statements in it.
- You will lose a mark for every print statements.
- There is a getName() function in the Lab5.py file.
- Change the output string to your last and first name.
- Your name should be EXACTLY as written on D2L.
- Incorrectly filed out getName function result in a mark of zero.

Lab Description:

In this lab you will implement a trie abstract data structure ADT that includes an autoComplete() method. The trie node should be implement using a sequential method. The autocomplete method should recursively explore the trie and return a list of all words in the trie that match that prefix. The returned list must be in alphabetical order. The implementation details are stated below.

Requirements:

- First, create a method called getName() that returns your name. If this does not work, you will receive a mark of zero.
- You are given a Python Class template MyTrie, and a list of words in the file 'american-english-no-accents.txt'.
- Write a recursive trie data structure, where each node is implemented sequentially.
- The trie should be implemented in a way where it can support the insertion of every word in text file given. The list of characters in that list include lower alphabet letters “a” to “z”, upper alphabet letters “A” to “Z”, and apostrophe character “’”
- The trie class must have an autoComplete(x) method where x is the prefix and the method should return a list of all words in the trie that matches that prefix.
- The returned list must be in alphabetical order where upper case letters are lower in order than lower case letters and shorter length words are lower in order than longer words.
- The MyTrie class, the following methods must be implemented:
    - insert(word, position=0) – this method has two inputs, ‘word’ a string containing, and an integer “position”. This should be recursive method that insert “word” into the trie. The input “position” has a default value of 0 and can be used to keep track of index/position of “word”.
    - remove(word, position=0) – this method has two inputs, ‘word’ a string containing, and an integer “position”. The method should recursively explore the trie to find the “word” and remove it from the trie if found. The input “position” has a default value of 0 and can be used to keep track of index/position of “word”.

- exists(word, position=0) – this method has two inputs, ‘word’ a string containing, and an integer “position”. The method should recursively explore the trie to find the “word” and return turn True if it is found and False otherwise. The input “position” has a default value of 0 and can be used to keep track of index/position of “word”.
- autoComplete(word, position=0) – the requirements for this method is described above.
- depth_of_word(word, position=0) – this method has two inputs, ‘word’ a string containing, and an integer “position”. The method should recursively explore the trie to find the “word” and return turn the depth of the “word” and -1 if “word” is not in the trie. The input “position” has a default value of 0 and can be used to keep track of index/position of “word”.
- char_to_position(c) – since the trie internal nodes are implemented sequentially, this method should return the index that mathes the input character “c”. This method should have the time complexity of O(1).

Example:

- Given the words 'dad', 'daddy', 'daddio', 'danny', 'mum', and 'mummy', the trie should look like this:

```
                     |d|m|
                    /      \
                 |a|        |u|
               /              \
             |d | n|           |m|
            /      \            \
          |d | #|  danny      |m | #|
        /       \              /    \
     |i | y|    dad        mummy    mum
   /       \
daddio   daddy
```

- For brevity, the nodes in the diagram above only show the letters used but of course each node’s array should be large enough to contain every possible characters.
- When the prefix 'da' is autocompleted, the list returned should be: ['dad', 'daddio', 'daddy', 'danny']. When the prefix '' is given, every word in the trie should be returned, in alphabetical order. When the prefix 'uncl' is given, an empty list should be returned.
- The depth of the word “dad” is 4 and the word “danny” is 3.

Notes/Hints:

- Ensure that duplicate words do not get added to the trie twice. Both lower and upper case letters will be used.
- The file 'american-english-no-accents.txt' is used by the tester but you can write your own test dictionary and tester program.
- Although the characters include lower and upper case letters and apostrophe, it is suggested that an extra space should be added as indicated in the class lecture.

Testing Your Code:

You can test your code with the tester. The tester is not a be all and end all. You should rely on your own testing before using the tester. Note that the lab should be done on Python 3 and the tester only works on Python 3. The tester is named Lab5Tester.py. Your file must be named correctly and be in the same folder of the tester. Here is an example of executing it on the command line (note that the $ is representative of the beginning of the new line).

```
$ python3 Lab5Teser.py
```