Solution Review: Find All Words Stored in Trie

This review provides a detailed analysis of the solution to the Find All Words Stored in Trie Challenge.

We'll cover the following

Solution: Recursion

main.py

Trie.py

TrieNode.py

```python
from Trie import Trie
from TrieNode import TrieNode


# Create Trie => trie = Trie()
# TrieNode => {children, is_end_word, char,
# mark_as_leaf(), unmark_as_leaf()}
# get_root => trie.get_root()
# Insert a Word => trie.insert(key)
# Search a Word => trie.search(key) return true or false
# Delete a Word => trie.delete(key)
# Recursive Function to generate all words
def get_words(root, result, level, word):

    # Leaf denotes end of a word
    if root.is_end_word:
        # current word is stored till the 'level' in the chara
        temp = ""
        for x in range(level):
            temp += word[x]
        result.append(str(temp))

    for i in range(26):
        if root.children[i]:
            # Non-None child, so add that index to the charact
            word[level] = chr(i + ord('a'))  # Add character f
            get_words(root.children[i], result, level + 1, wor


def find_words(root):
    result = []
    word = [None] * 20  # assuming max level is 20
    get_words(root, result, 0, word)
    return result


keys = ["the", "a", "there", "answer", "any", "by", "bye", "th
t = Trie()
for i in range(len(keys)):
    t.insert(keys[i])
lst = find_words(t.root)
print(str(lst))
```

```
43
```

The `find_words(root)` function contains a `result` list which will contain all the words in the trie. `word` is a character array in which node characters are added one by one to keep track of all the letters in the same recursive call.

`get_words()` is our recursive function which begins from the root and traverses every node. Whenever a node is the end of a word, `temp` (containing the character array) is converted into a string and inserted into `result`.

Since `word` cannot be reset before recording every new word, we simply update the values at each index using `level`.

Time Complexity

As the algorithm traverses all the nodes, its run time is *O(n)* where **n** is the number of nodes in the trie.