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

```
from Trie import Trie
from TrieNode import TrieNode


# Create Trie => trie = Trie()
# TrieNode => {children, is_end_word, char}
# 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
```

```
14      if root.is_end_word:
15          # current word is stored till the 'level' in t ar a
16          temp = ""
17          for x in range(level):
18              temp += word[x]
19          result.append(str(temp))
20
21      for i in range(26):
22          if root.children[i]:
23              # Non-None child, so add that index to the character arra
24              word[level] = chr(i + ord('a'))  # Add character for the
25              get_words(root.children[i], result, level + 1, word)
26
27
28  def find_words(root):
```

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.

Back

Challenge 2: Find All Words Stored in ...

Next

Challenge 3: List Sort Using Trie

Completed

Report an Issue