





## Solution Review: Find All Words Stored in Trie

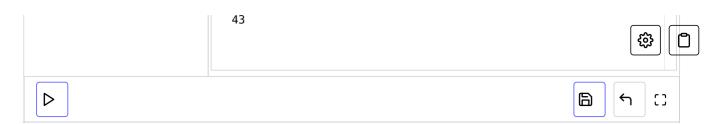
This review provides a detailed analysis of the solution to the Find All Words Stored in Trie Challenge.

We'll cover the following ^

- Solution: Recursion
  - Time Complexity

## Solution: Recursion #

```
1 from Trie import Trie
main.py
                                from TrieNode import TrieNode
                              3
Trie.py
                              4
                              5 # Create Trie => trie = Trie()
TrieNode.py
                              6 # TrieNode => {children, is_end_word, char,
                              7
                                # mark_as_leaf(), unmark_as_leaf()}
                                # get root => trie.get root()
                              9 # Insert a Word => trie.insert(key)
                             10 # Search a Word => trie.search(key) return true or false
                             11 # Delete a Word => trie.delete(key)
                             12 # Recursive Function to generate all words
                             13 def get_words(root, result, level, word):
                             14
                             15
                                     # Leaf denotes end of a word
                                     if root.is_end_word:
                                         # current word is stored till the 'level' in the chara
                             17
                             18
                                         temp = ""
                             19
                                         for x in range(level):
                             20
                                             temp += word[x]
                             21
                                         result.append(str(temp))
                             22
                                     for i in range(26):
                             23
                             24
                                         if root.children[i]:
                             25
                                             # Non-None child, so add that index to the charact
                             26
                                             word[level] = chr(i + ord('a')) # Add character f
                             27
                                             get_words(root.children[i], result, level + 1, wor
                             28
                             29
                             30
                                def find_words(root):
                                     result = []
                             31
                             32
                                     word = [None] * 20 # assuming max level is 20
                                     get_words(root, result, 0, word)
                             33
                             34
                                     return result
                             35
                             36
                                keys = ["the", "a", "there", "answer", "any", "by", "bye", "th
                             37
                             38 t = Trie()
                                for i in range(len(keys)):
                             39
                             40
                                     t.insert(keys[i])
                             41
                                lst = find_words(t.root)
                                 print(str(lst))
```



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.

