Solution Review: Word Formation From a Dictionary Using Trie

This review provides a detailed analysis of the solution to the Word Formation From a Dictionary Using a Trie Challenge.

We'll cover the following

Solution: Iterative Word Matching

main.py

Trie.py

TrieNode.py

```
from Trie import Trie
from TrieNode import TrieNode


def is_formation_possible(dct, word):

    # Create Trie and insert dctionary elements in it
    trie = Trie()
    for elem in dct:
        trie.insert(elem)

    # Get Root
    current = trie.root
```

```
14
15      # Iterate all the letters of the word
16      for i in range(len(word)):
17          # get index of the character from Trie
18          char = trie.get_index(word[i])
19
20          # if the prefix of word does not exist, word would not either
21          if current.children[char] is None:
22              return False
23
24          # if the substring of the word exists as a word in trie,
25          # check whether rest of the word also exists,
26          # if it does return true
27          elif current.children[char].is_end_word:
28              if trie.search(word[i+1:]):
```

The algorithm can be divided into three parts. The first and simplest part is making a trie for the words in the dictionary.

The second part is to check if there is a `word` in the trie which can become a prefix for the query word. In the case of `“helloworld”`, we can find `“hello”` in the trie. Since there can be multiple prefixes of a word, we have to check for every such prefix. As we iterate through the trie, looking for prefix, whenever we find a prefix that exists as a word in the trie, we lookup the remaining word in the trie using the search function. If this substring exists we have found a solution

Time Complexity#

We perform the insert operation **m** times for a dictionary of size **m**. After that, the search operation runs on the `word` in the sequence:

```
"h", "he", "hel", "hell"...
```

If the length of the average word in the dictionary is **h**, then [illegible] for trie construction is $O(m \times h)$. Let the length of the word being searched be **n**. Then, the complexity for this turns out to be n^2. Hence, the total time complexity is $O(mh + n^2)$. We could argue that in some applications, the trie is constructed only once, and then many many lookups are performed. So, the cost of trie creation is amortized over all the lookups. In that case, the complexity reduces to $O(n^2)$.

We will solve this challenge again in the hashing chapter.