Auto Complete

Problem Level: Hard

Problem Description:

Given n number of words and an incomplete word w. You need to auto-complete that word w. That means, find and print all the possible words which can be formed using the incomplete word w.

Note : Order of words does not matter.

Sample Input 1:

```
7
do dont no not note notes den
no
```

Sample Output 1:

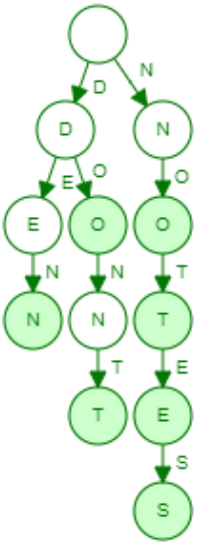
```
no
not
note
notes
```

Explanation:

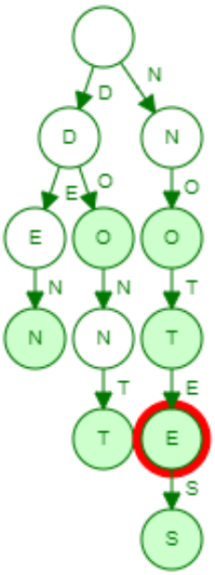
"abc" and "cba" forms a palindrome

Approach to be followed:

To solve this question, we use tries. Let us understand why. The trie formed using the input looks like



In this visualisation, the green nodes represent the end of the word. Next we will try to find a word, say, “note” in this TRIE Data Structure. It will jump from node to node and when the end of the leaf node approaches, it processes the string and displays the string.



"note" is processed by starting up from the root node and comparing each node as it moves down. Here the traversal can be viewed as N->O->T->E as the search terminates due to the end of the string. If we have taken another word "notebook" then the traversals would look like N->O->T->E->EndOfWord . The string could not be found in the trieas the end of the string is reached and the search is terminated.

As we insert keys in the trie, each input key is inserted as an individual node and performs the role of an index to further child nodes. If our input is a prefix of an existing string present in the trie, then we can mark the new node as a leaf node to the pre-defined key. Otherwise, the new keys are constructed and the last node is marked as the Leaf Node. Search Operation is relatively easy in trie as a Search Operation starts from the Root Node and moves ahead by comparing characters. If any character does not exist, then no such string is found in the trie.

Our autocomplete function will be an extension of the search feature. Once we have found the string in our trie, we need to do a depth first search or breadth first search from that node (as a head node) and keep a list of all complete strings encountered in the process. This list of strings is our list of autocomplete words.

Steps:

1. Using search function, find the last node of the incomplete word w, if at all it exists.
2. Using DFS/BFS, traverse to each possible branch considering this found node as the head node.
3. Store all the possible strings thus obtained inside a list.

Pseudocode for Auto complete feature:

Pseudo Code:

```
function autoComplete(root, pattern, output):

    if root equals null:

          return
```

```
          if pattern.length equals 0:
                if root.terminal is true:
                      print(output)

                for each char in root.children:
                      s = output + char
                      printS(root.children[char], s)
                return



          indexChar = pattern[0]
          output += pattern[0]
          if root.children.get(indexChar) not equal to null:
                autoComplete(root.children[indexChar], pattern[1:],
output)

     function printS(self, root, s) :
          if root equals Non :
                return



          if (root.terminal):
                print(s)



          for char in root.children:
                t = s + char
                printS(root.children[char],t)
```

Time Complexity: O(N * M), where **N** is the number of words in the Trie and **M** is the average word length