STRINGS-2

Question: Write a function to find the longest common prefix string amongst an array of strings.

YOU HAVE 15 MINUTES

EXAMPLE:

INPUT: ["flower","flow","flight"], ["dog","racecar","car"]

OUTPUT: “fl”, ””

Link: https://leetcode.com/problems/longest-common-prefix/

Just take the *prefix* = “flower”

🟩 “flower”	“flow”	“flight”
	“flow”	“fl”

Horizontal Scanning

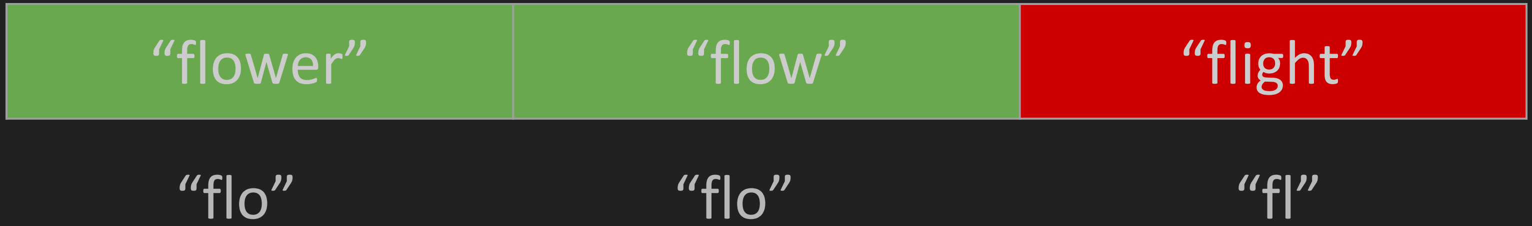
Just take the *prefix* = “flower”

“flower”	“flow”	“flight”
	“flow”	“fl”

“flower”	“flow”	“flight”

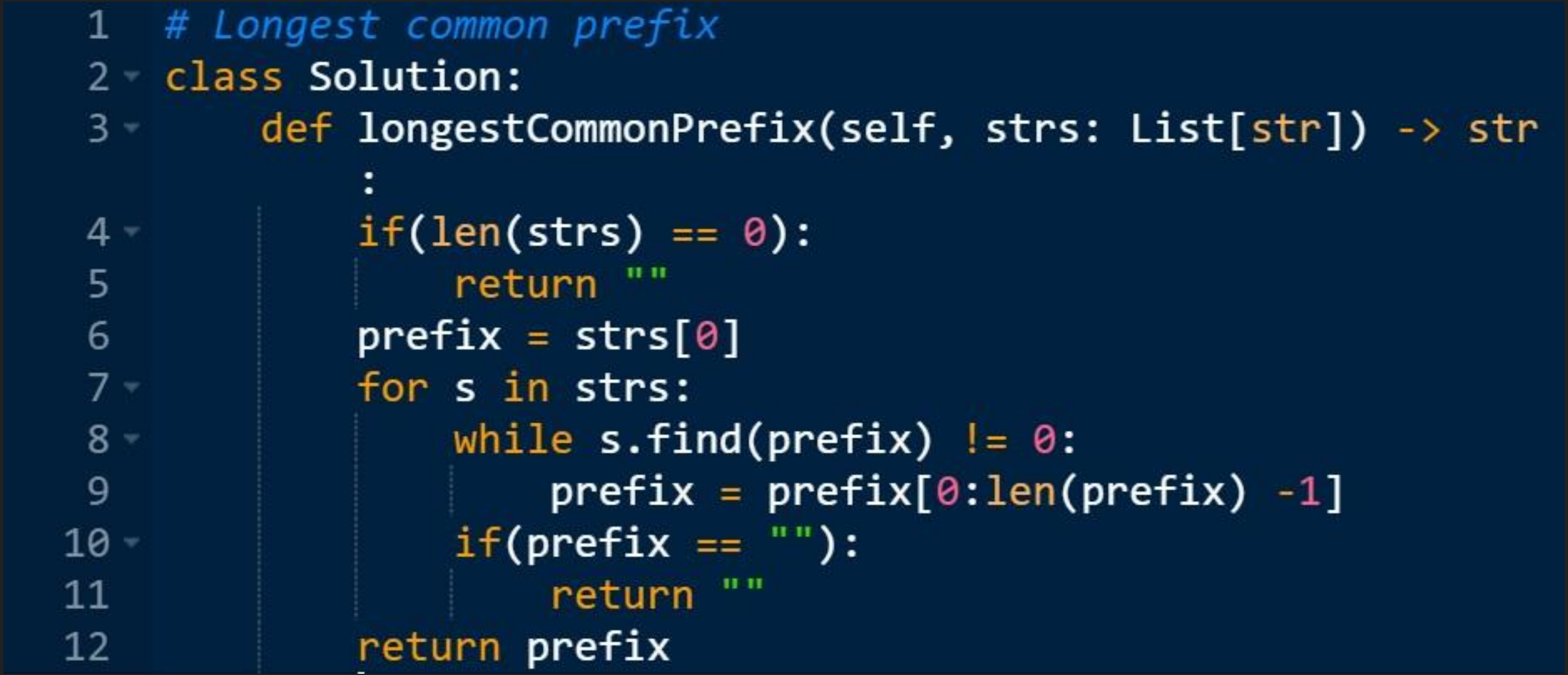
“f” “f” “f”

“flower”	“flow”	“flight”
“fl”	“fl”	“fl”


"flower"
"flow"
"flight"
"flo"
"flo"
"fl"

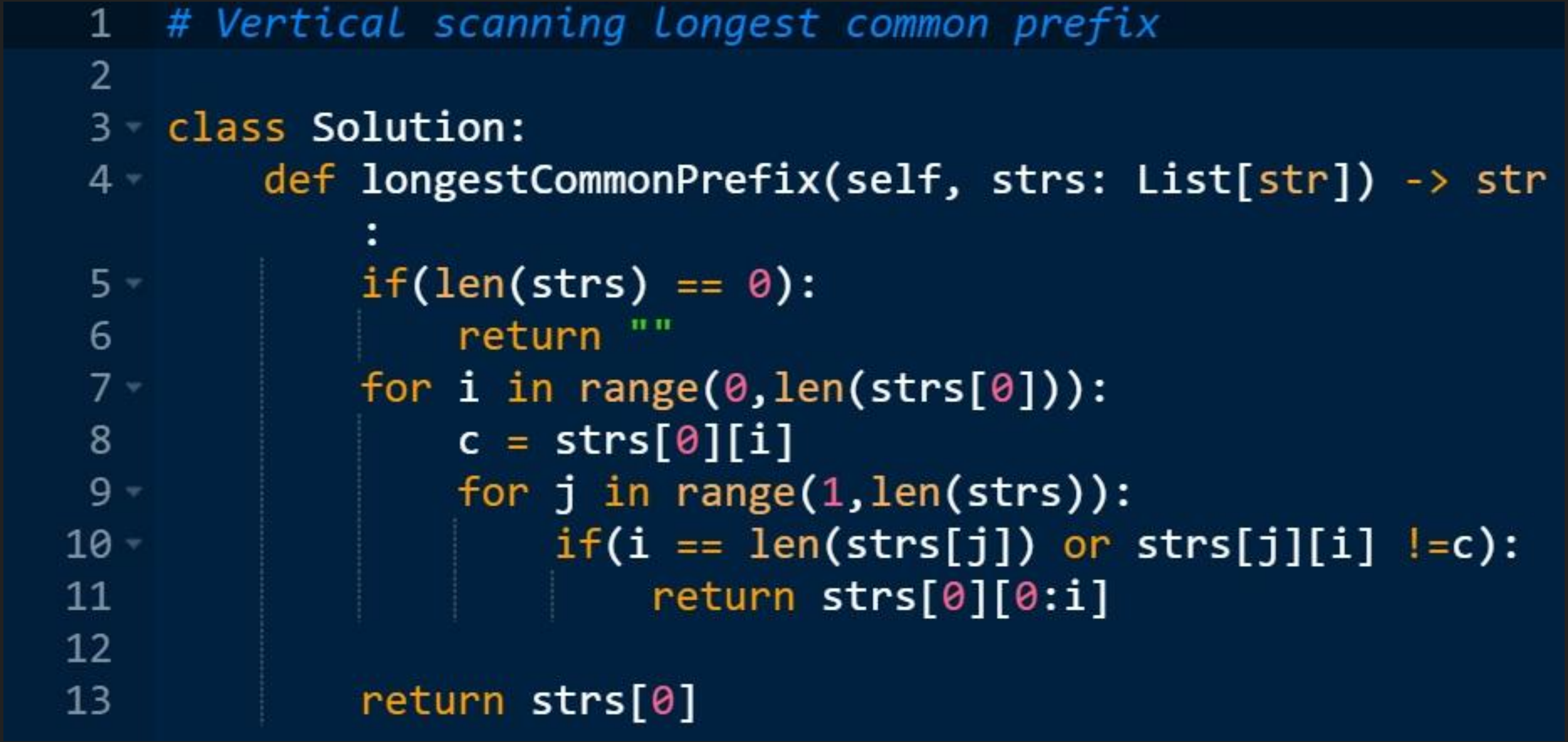
Vertical Scanning

“flower”	“flow”	“flight”
“fl”	“fl”	“fl”



```python
# Longest common prefix
class Solution:
    def longestCommonPrefix(self, strs: List[str]) -> str
        :
        if(len(strs) == 0):
            return ""
        prefix = strs[0]
        for s in strs:
            while s.find(prefix) != 0:
                prefix = prefix[0:len(prefix) -1]
            if(prefix == ""):
                return ""
        return prefix
```

TIME COMPLEXITY - O(N’), where N’ is the sum of all the characters of string in the list.
SPACE COMPLEXITY - O(1)



```python
# Vertical scanning longest common prefix

class Solution:
    def longestCommonPrefix(self, strs: List[str]) -> str
        :
        if(len(strs) == 0):
            return ""
        for i in range(0,len(strs[0])):
            c = strs[0][i]
            for j in range(1,len(strs)):
                if(i == len(strs[j]) or strs[j][i] !=c):
                    return strs[0][0:i]

        return strs[0]
```

TIME COMPLEXITY - O(N’), where N’ is the sum of all the characters of string in the list.
SPACE COMPLEXITY - O(1)

Question: For some given string S, a query word is stretchy if it can be made to be equal to S by any number of applications of the following extension operation: choose a group consisting of characters c, and add some number of characters c to the group so that the size of the group is 3 or more.Given a list of query words, return the number of words that are stretchy. YOU HAVE 15 MINUTES

LOOK UP :

Link: https://leetcode.com/problems/expressive-words/

Link: https://leetcode.com/problems/expressive-words/

EXAMPLE:

INPUT: S = "heeellooo", words = ["hello", "hi", "helo"]

OUTPUT: 1

Explanation: We can extend "e" and "o" in the word "hello" to get "heeellooo". We can't extend "helo" to get "heeellooo" because the group "ll" is not size 3 or more.

TRANSFORM THE STRINGS TO AN ENCODING

Take S = "heeellooo" and make S=”h1e3l2o3”

Take words = ["hello", "hi", "helo"] and make words = ["h1e1l2o1", "h1i1", "h1e1l1o1"]

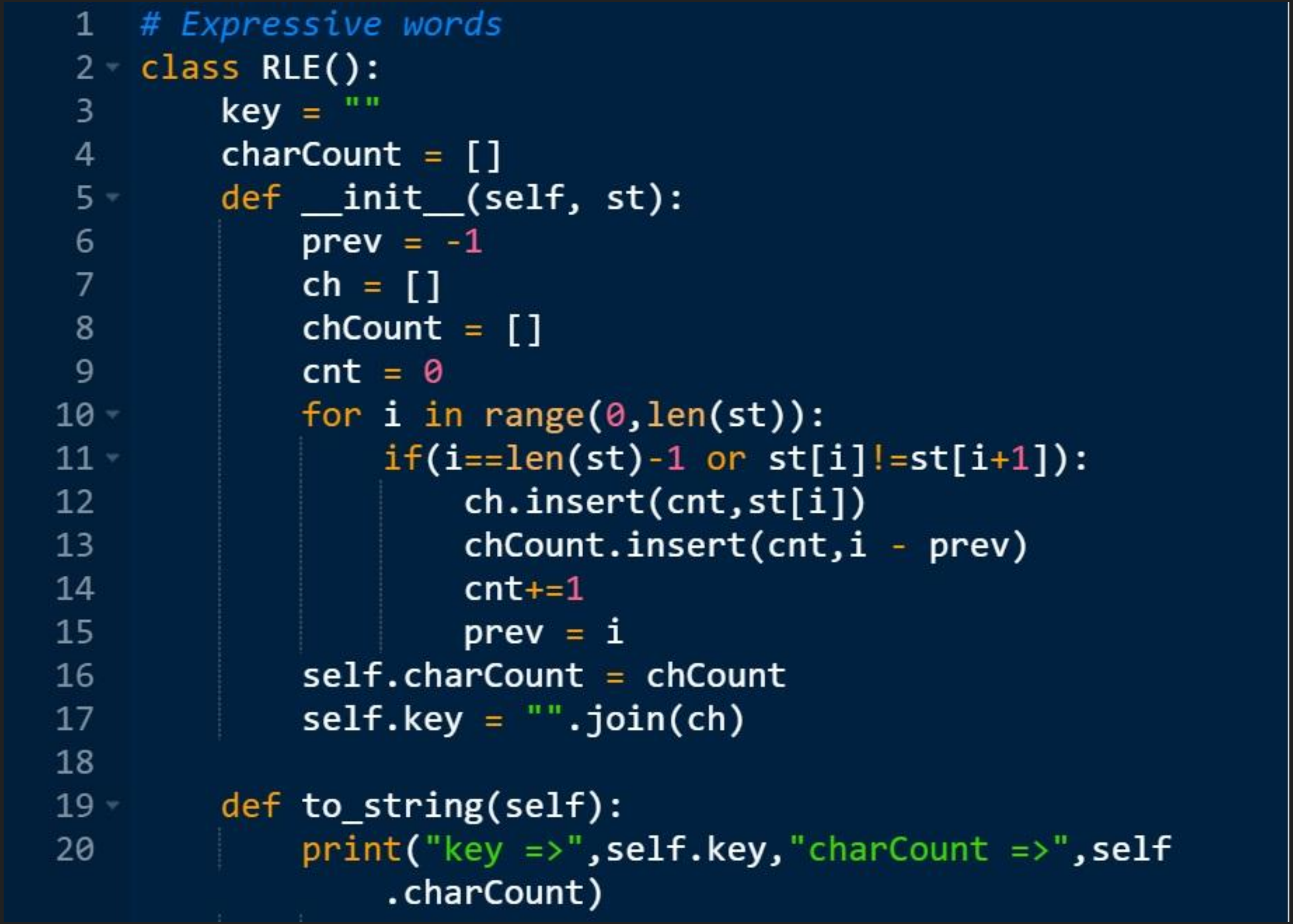
Take S = "heeellooo" and make S=”h1e3l2o3”

Take words = ["hello", "hi", "helo"] and make words = ["h1e1l2o1", "h1i1", "h1e1l1o1"]

More clearly, create two attributes key and counts

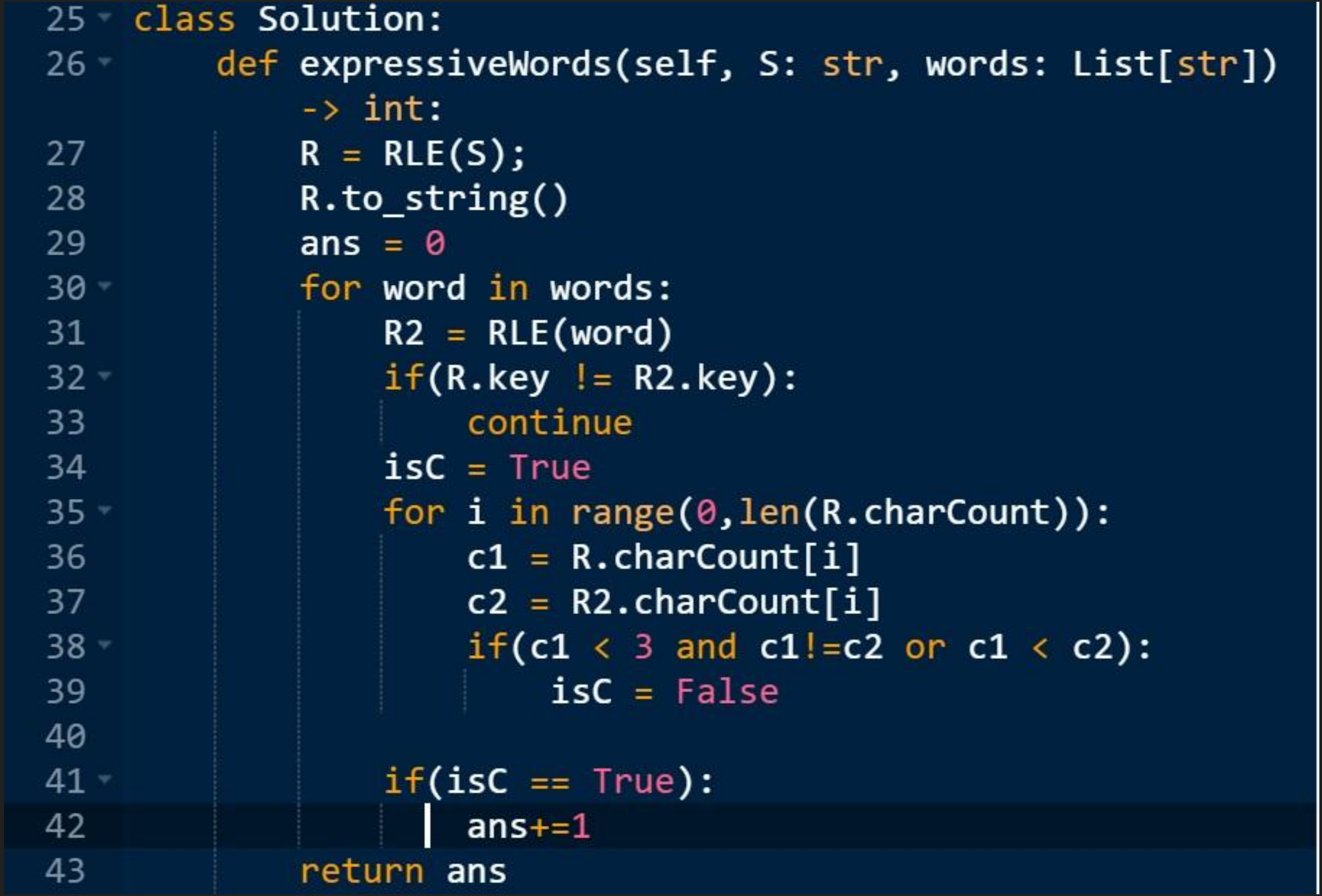
S = "heeellooo", key = helo and counts = [1,3,2,3]

So, words = ["hello", "hi", "helo"] become [“helo”,”hi”,”helo”] and [[1,1,2,1] , [1,1] , [1,1,1,1]]



```python
# Expressive words
class RLE():
    key = ""
    charCount = []
    def __init__(self, st):
        prev = -1
        ch = []
        chCount = []
        cnt = 0
        for i in range(0,len(st)):
            if(i==len(st)-1 or st[i]!=st[i+1]):
                ch.insert(cnt,st[i])
                chCount.insert(cnt,i - prev)
                cnt+=1
                prev = i
        self.charCount = chCount
        self.key = "".join(ch)

    def to_string(self):
        print("key =>",self.key,"charCount =>",self
            .charCount)
```

Create a class that decomposes the ‘expressive’ nature of the strings in input to run length encoding. Use the encoding for problem.



```
class Solution:
    def expressiveWords(self, S: str, words: List[str])
        -> int:
        R = RLE(S);
        R.to_string()
        ans = 0
        for word in words:
            R2 = RLE(word)
            if(R.key != R2.key):
                continue
            isC = True
            for i in range(0,len(R.charCount)):
                c1 = R.charCount[i]
                c2 = R2.charCount[i]
                if(c1 < 3 and c1!=c2 or c1 < c2):
                    isC = False

            if(isC == True):
                ans+=1
        return ans
```

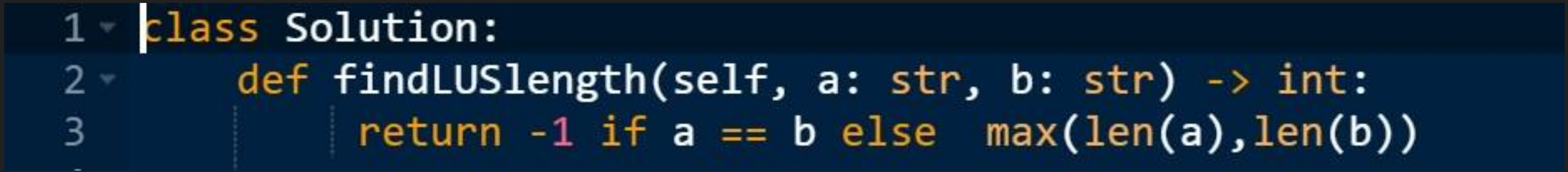
TIME COMPLEXITY: O(NK), N is number of words and K is the length of the largest word in words.
SPACE COMPLEXITY: O(K)

Question: Given a group of two strings, you need to find the longest uncommon subsequence of this group of two strings. The longest uncommon subsequence is defined as the longest subsequence of one of these strings and this subsequence should not be any subsequence of the other strings. YOU HAVE 15 MINUTES

INPUT: “aba”, ”cdc”

OUTPUT: 3

https://leetcode.com/problems/longest-uncommon-subsequence-i/



```python
class Solution:
    def findLUSlength(self, a: str, b: str) -> int:
        return -1 if a == b else  max(len(a),len(b))
```

TRIE

Because it’ an efficient answer to many string based problems.

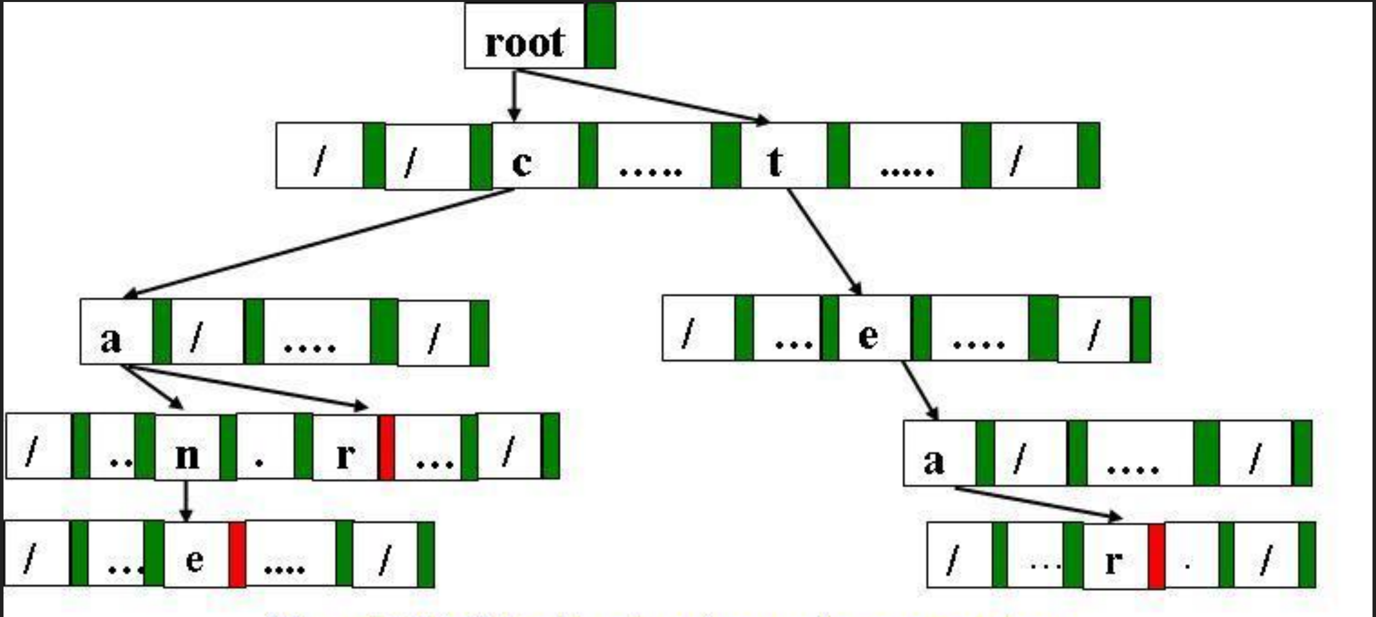


Figure 1: Trie Data Structure for words: cane, car, tear

TRIE

TRIE AS A DATA STRUCTURE WILL BE DISCUSSED AGAIN IN BACKTRACKING SESSIONS. FYI

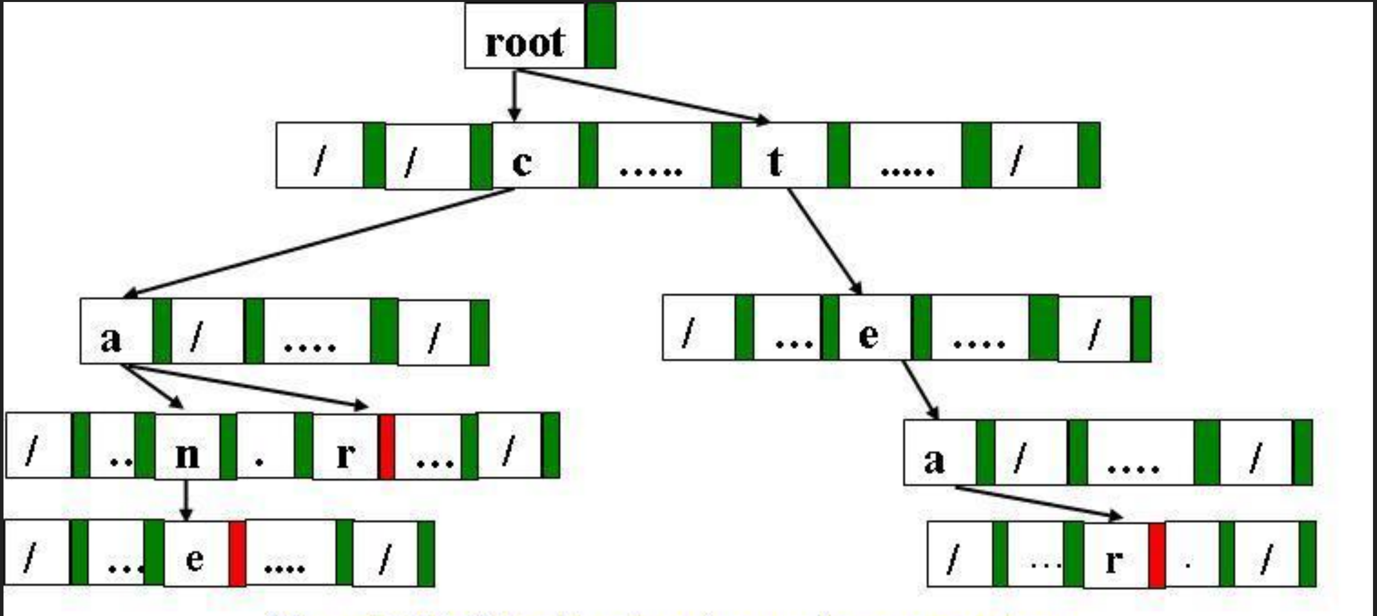


Figure 1: Trie Data Structure for words: cane, car, tear

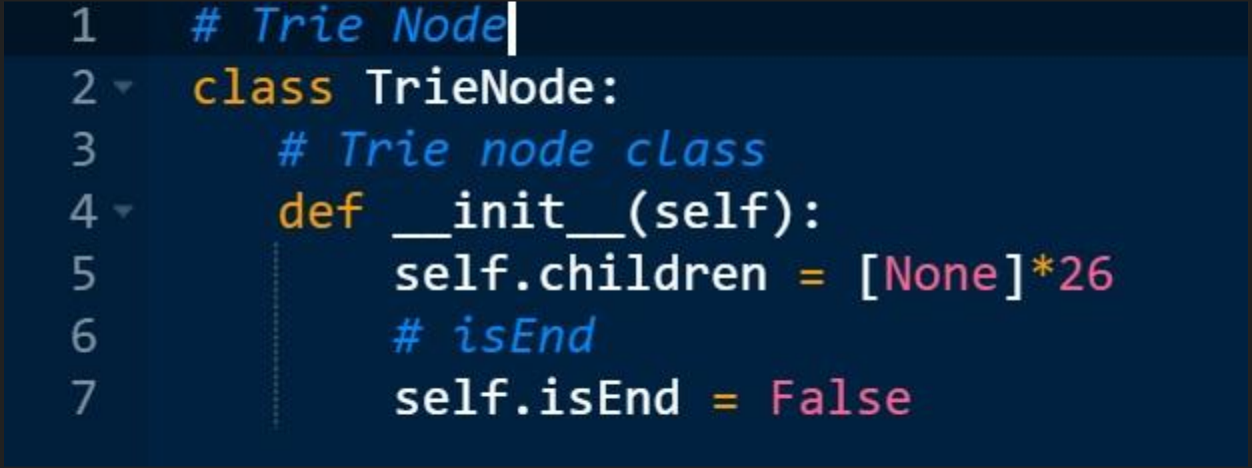
1. CHARACTER
2. 26 CHILDREN
3. isEnd

Every node has multiple (26) branches. Each branch is a possible character of keys (strings). We need to mark the last node of every key (string) as end of word node. A Trie node field isEnd marks end of word node.

IMPLEMENT A TRIE

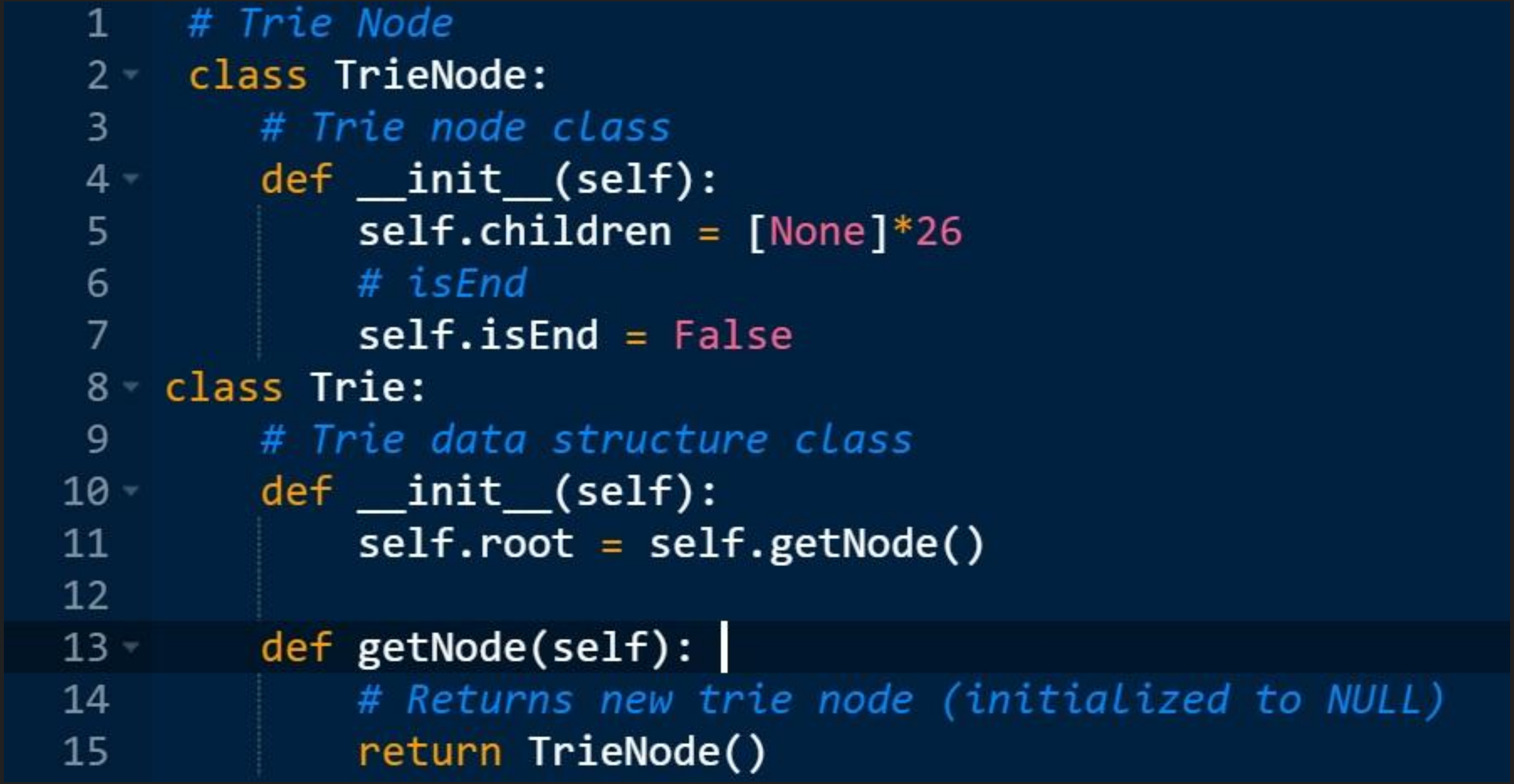
https://leetcode.com/problems/implement-trie-prefix-tree/

TRIE NODE



```python
# Trie Node
class TrieNode:
    # Trie node class
    def __init__(self):
        self.children = [None]*26
        # isEnd
        self.isEnd = False
```

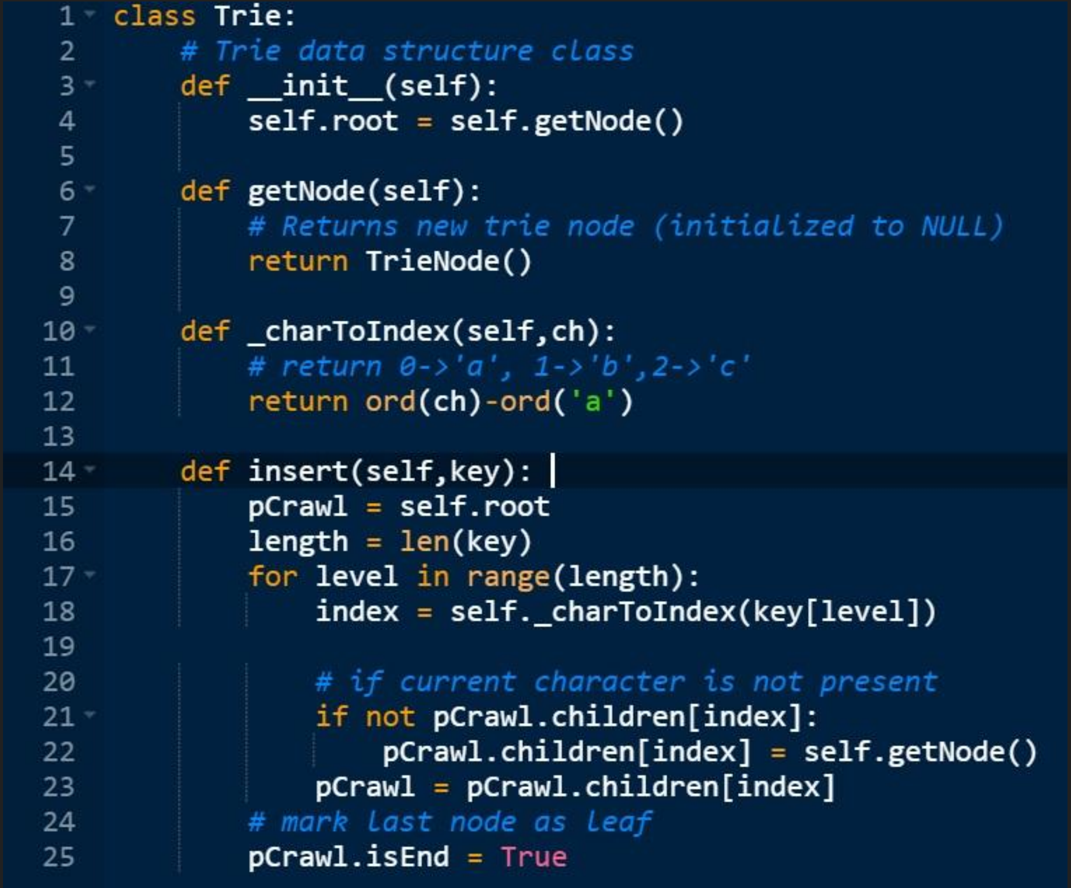
TRIE DATA STRUCTURE



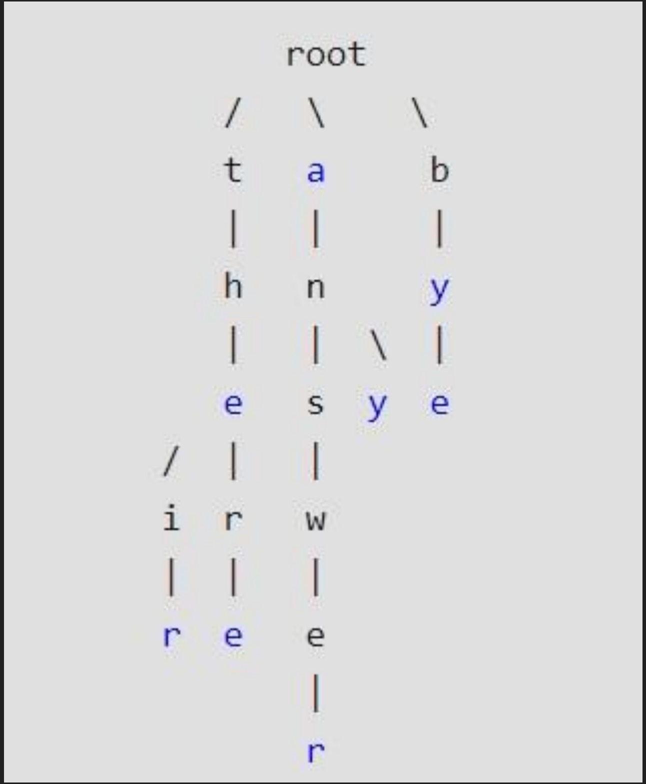
```python
 # Trie Node
 class TrieNode:
     # Trie node class
     def __init__(self):
         self.children = [None]*26
         # isEnd
         self.isEnd = False
class Trie:
     # Trie data structure class
     def __init__(self):
         self.root = self.getNode()

     def getNode(self):
         # Returns new trie node (initialized to NULL)
         return TrieNode()
```

TRIE INSERT

[“their”, ”the”, ”their”, ”answer”, ”any”, ”bye”]

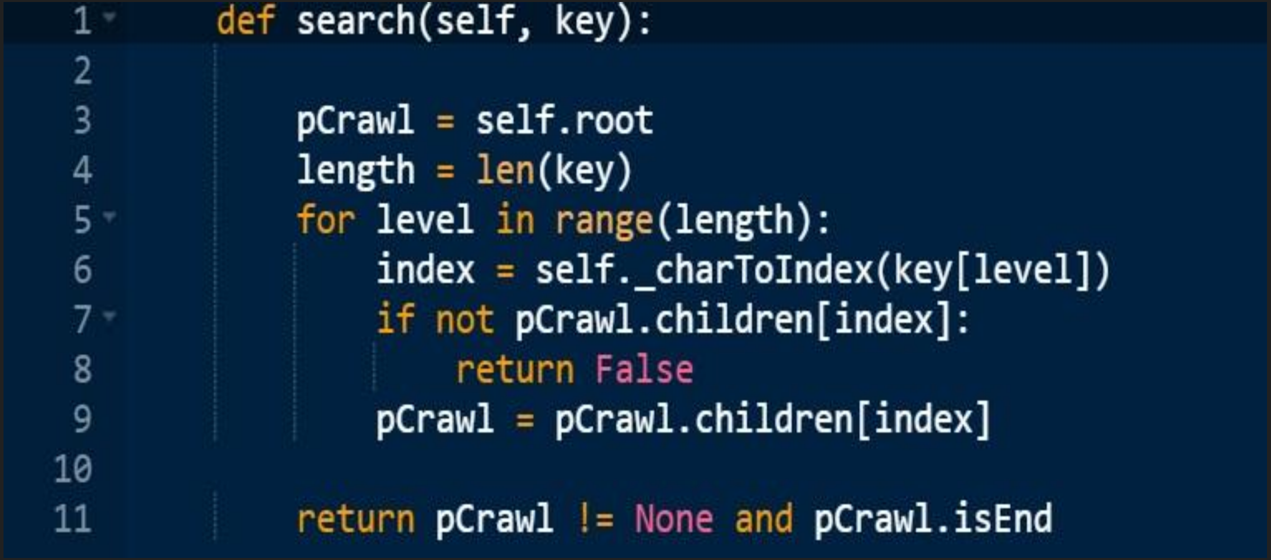


```python
class Trie:
    # Trie data structure class
    def __init__(self):
        self.root = self.getNode()

    def getNode(self):
        # Returns new trie node (initialized to NULL)
        return TrieNode()

    def _charToIndex(self,ch):
        # return 0->'a', 1->'b',2->'c'
        return ord(ch)-ord('a')

    def insert(self,key):
        pCrawl = self.root
        length = len(key)
        for level in range(length):
            index = self._charToIndex(key[level])

            # if current character is not present
            if not pCrawl.children[index]:
                pCrawl.children[index] = self.getNode()
            pCrawl = pCrawl.children[index]
        # mark last node as leaf
        pCrawl.isEnd = True
```

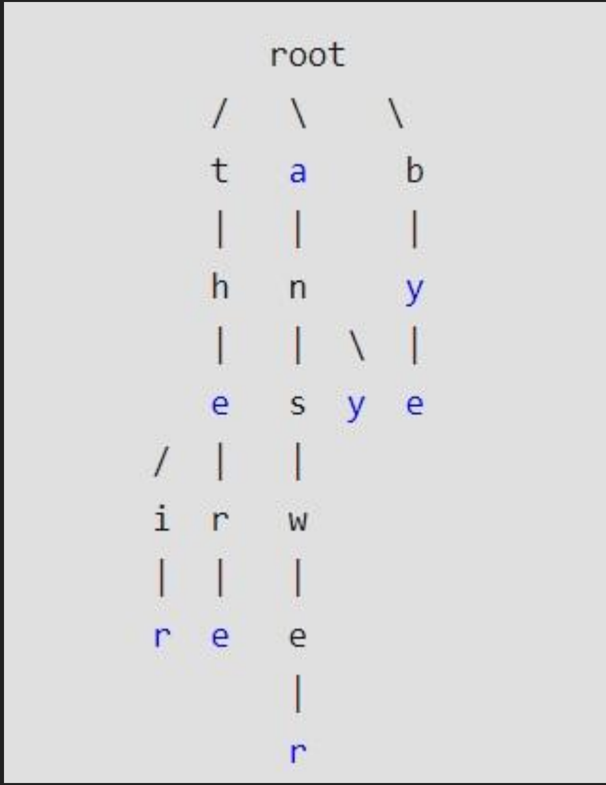


TRIE SEARCH

[“their”, ”the”, ”th”, ”ans”, ”anyother”, ”bye”]



```python
def search(self, key):

    pCrawl = self.root
    length = len(key)
    for level in range(length):
        index = self._charToIndex(key[level])
        if not pCrawl.children[index]:
            return False
        pCrawl = pCrawl.children[index]

    return pCrawl != None and pCrawl.isEnd
```



```
            root
         /    \     \
         t    a      b
         |    |      |
         h    n      y
         |    |  \   |
         e    s   y  e
      /  |    |
      i  r    w
      |  |    |
      r  e    e
              |
              r
```

TIME COMPLEXITY for insert and search - O(string_length)

SUMMARY

1. ANAGRAMS
2. PALINDROMES
3. PREFIXES
4. ENCODING
5. SLIDING WINDOW BASED APPROACH
6. TRIES