**Approach, Performance & Complexity Analysis**

Assumptions:

As per the problem statement I am considering only string types.

In the command line arguments, I am passing list of string as comma separated strings. Later, I split these input string in list of strings as input to the class but you want to pass list of string, I would appreciate you manually provide list of string in the file itself and run.

I have not added requirement.txt as there is no external library is used. You can run with your existing python setup on server or desktop.

I wanted to work on directly structure but keeping in mind the simple solution I have put the main python file and test file in the same folder. I could create structure like below. So, pls accept it.

| -- init.py

| -- src

| -- search.py

| -- tests

| -- test\_search.py

| -- requirement.txt

| -- README.me

Python Version:

The python version used for this task is 3.10.0.

External Libs:

There is no external library is used in the code.

Approach:

The approach used for this solution is written below. [find() function in the class]

*Let’s initialize:*

input\_strings\_list = [“seo”, “hello”, ”world”], match\_string = “eos”, match\_count=0, matched\_output\_strings = []

1. Taking one by one input string from list of strings. i.e., “seo”
2. Then iterative over the string taken from Step-1, char by char i.e., ‘s’, ’e’, ’o’
3. Checking if the char is present in match\_string i.e., in “seo”
4. If the char is present then increase the match\_count (to hold the consecutiveness of char in string) else reset the match\_count to zero to start the match from current char
5. Similarly, if next char is matched then match\_count is incremented with 1 else Step-4
6. If the match\_count gets equal to the length of match\_string then stop because you have found the string and you don’t to search anymore in the string and add to the output list i.e., in matched\_output\_strings and continue with Step-1.
7. If all the strings from input list are iterated then return the output as matched\_output\_strings

Note: match\_count temp variable is used to count the number of consecutive matches with match\_string in this

Code Optimization:

The code is already optimized and will perform best with any inputs. The test cases are also added in the code under test\_search.py file.

Pros/Cons:

Pros: If you perform search with small strings then it will be optimum. I think this will be more performant over sliding window approach.

Cons: If you perform search with large string then there might be slowness.

Other Questions:

Q.1. What do time complexity and space complexity look like in your approach? Can they be optimized?

Answer: The time space and space complexity is calculated as below.

k – list of input strings, n – match\_string

*Time Complexity:*

O(k) + O(k-n) + O(k-n) = ***O(kn)***

*Space Complexity:*

O(k) = ***O(K)***

Q.2. If the size of the initial string list is very large, would that influence the efficiency of the approach? And what if the number of “find” requests gets extremely large? Do you need to restructure/rethink the approach?

Answer: Yes, if the size is very large then we can split the input string in chunks and using multiprocessing we can optimize the performance of solution.