Data Structures & Algorithms III

SCS 2201 String Matching Assignment

Index No: 21002177

This code aims to use various wildcard characters like ".," "*," "+," and "?"to provide a variety of pattern-finding strategies inside a given text. Using wildcards, the method allows for flexible pattern matching and is designed to find instances of a user-defined pattern within a given input text. To get accurate and reliable pattern searching results, however, the code needs to have several flaws and unfinished areas fixed.

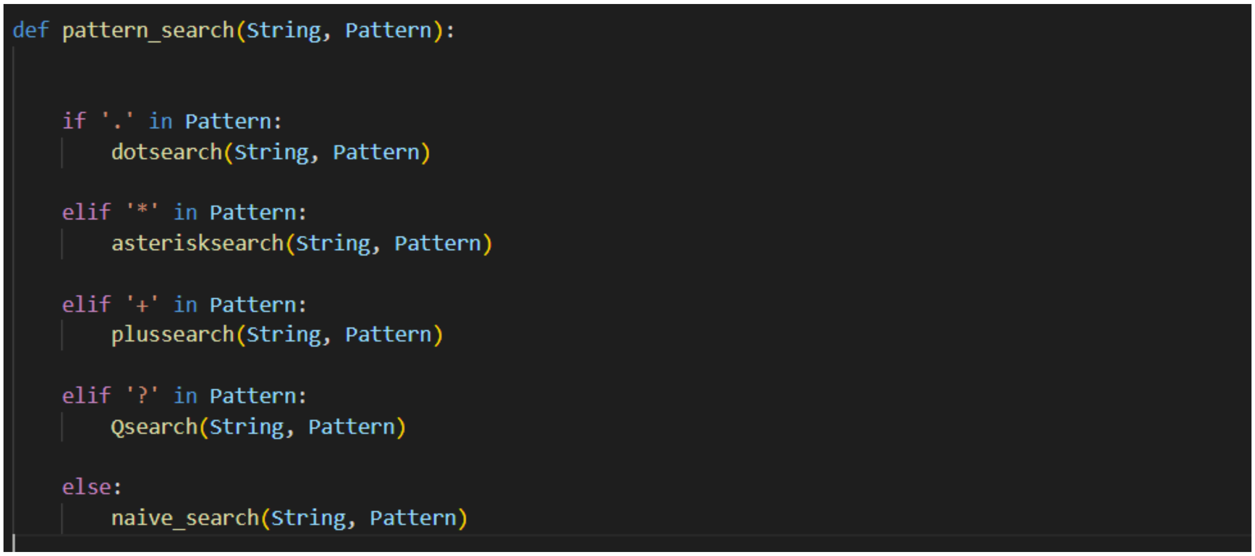
General explanation of the solution

Algorithm Modules:

- Pattern matching for specific patterns containing '.', '*', '+', and '?' is handled by the **dotsearch, asterisksearch, plussearch,** and **Qsearch** functions, respectively.
- These routines are made to search the given text for instances of the specified patterns and write the indexes to an output file.

Identifying Pattern:

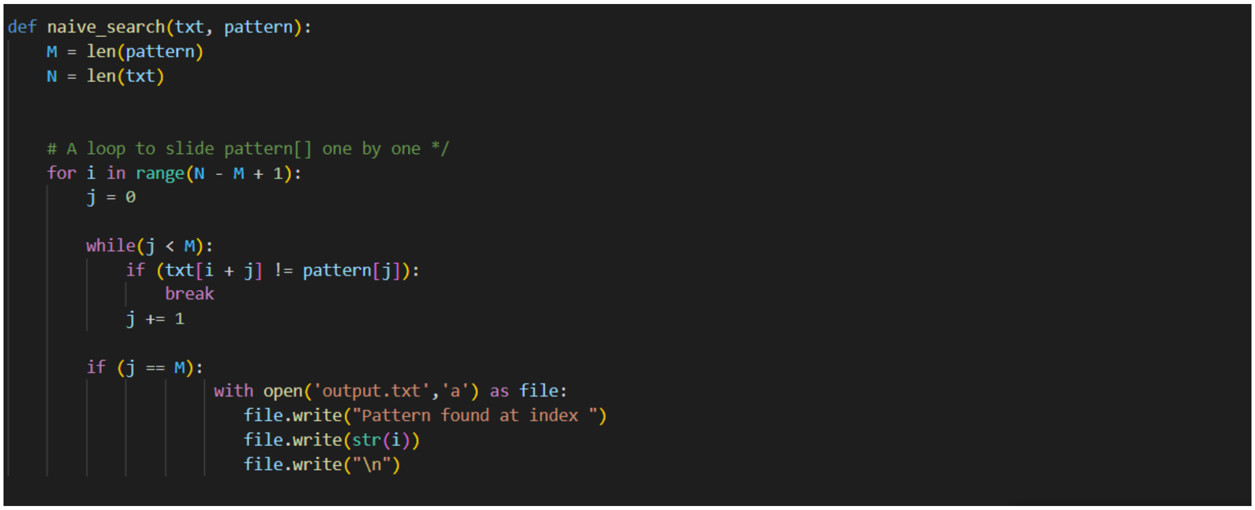
- The **pattern_search** function identifies the type of pattern based on special characters present and calls the corresponding algorithm function.



```
def pattern_search(String, Pattern):


    if '.' in Pattern:
        dotsearch(String, Pattern)

    elif '*' in Pattern:
        asterisksearch(String, Pattern)

    elif '+' in Pattern:
        plussearch(String, Pattern)

    elif '?' in Pattern:
        Qsearch(String, Pattern)

    else:
        naive_search(String, Pattern)
```

Naive Search:

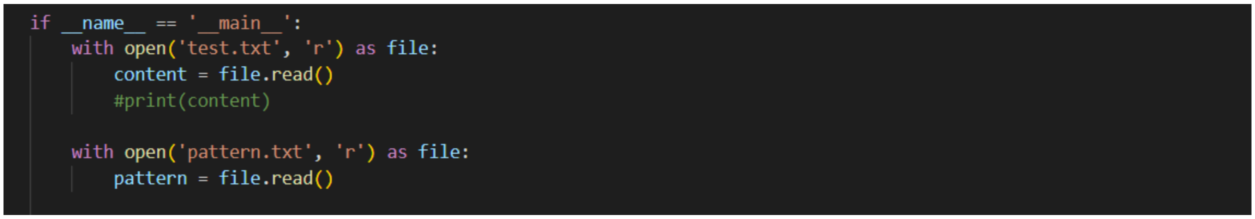
- The **naive_search** function implements a basic string-matching algorithm and is used as a default approach when no special characters are detected in the pattern.



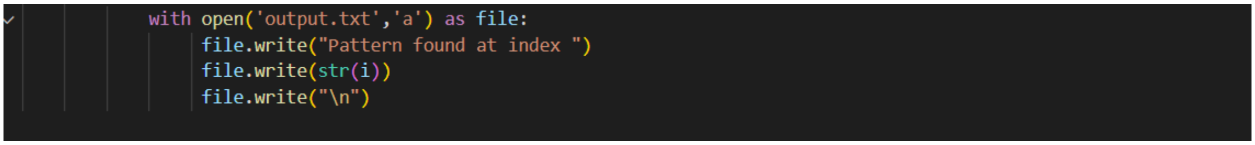
```
def naive_search(txt, pattern):
    M = len(pattern)
    N = len(txt)


    # A loop to slide pattern[] one by one */
    for i in range(N - M + 1):
        j = 0

        while(j < M):
            if (txt[i + j] != pattern[j]):
                break
            j += 1

        if (j == M):
                    with open('output.txt','a') as file:
                        file.write("Pattern found at index ")
                        file.write(str(i))
                        file.write("\n")
```

Input and Output Handling:

- **'test.txt'** is read by the program as the text, and **'pattern.txt'** is read as the pattern to be searched.
- The 'output.txt' file receives the identified pattern occurrences and their indices as an addition.



```
if __name__ == '__main__':
    with open('test.txt', 'r') as file:
        content = file.read()
        #print(content)

    with open('pattern.txt', 'r') as file:
        pattern = file.read()
```



```
            with open('output.txt','a') as file:
                file.write("Pattern found at index ")
                file.write(str(i))
                file.write("\n")
```

Why naive string-matching method?

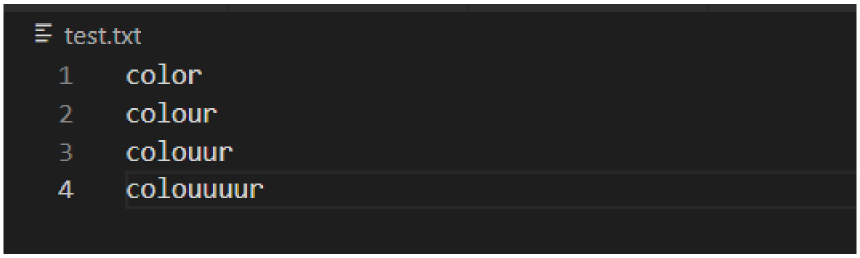
Because it is straightforward and simple to use, I went with the naive string-matching approach. Even though it can slow down the process when it comes to large amount of data, because of this program built for checking each character individually, I assume that small patterns are considered in this implementation.

Special points:

- Because of each string-matching algorithm is encapsulated in its own function its promoting,
    - Code reusability
    - Separation of concerns
- Clear output handling

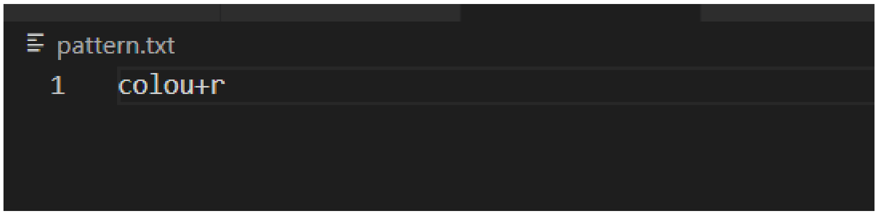
Test cases:

01.) “+”



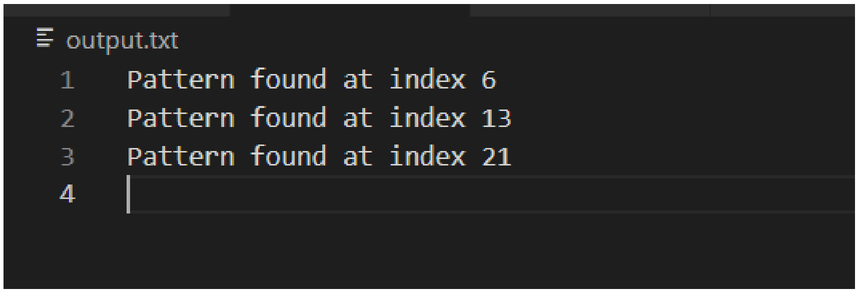
test.txt

```
color
colour
colouur
colouuuur
```



pattern.txt

```
colou+r
```

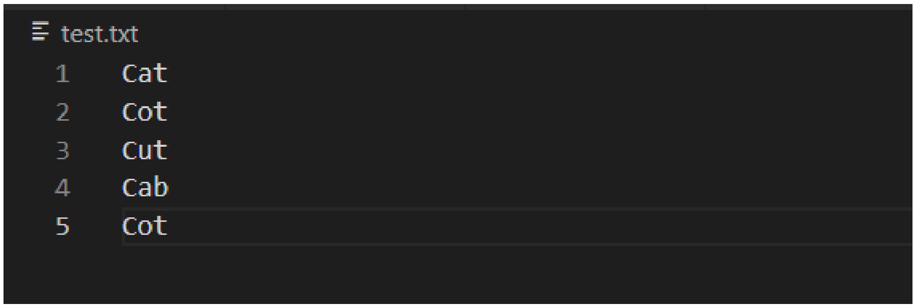


output.txt

```
Pattern found at index 6
Pattern found at index 13
Pattern found at index 21
```

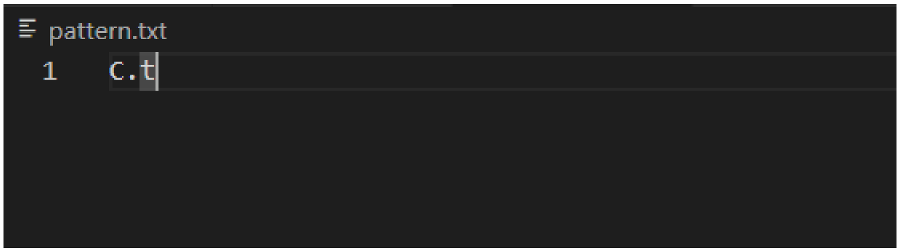
..

02.) “.”



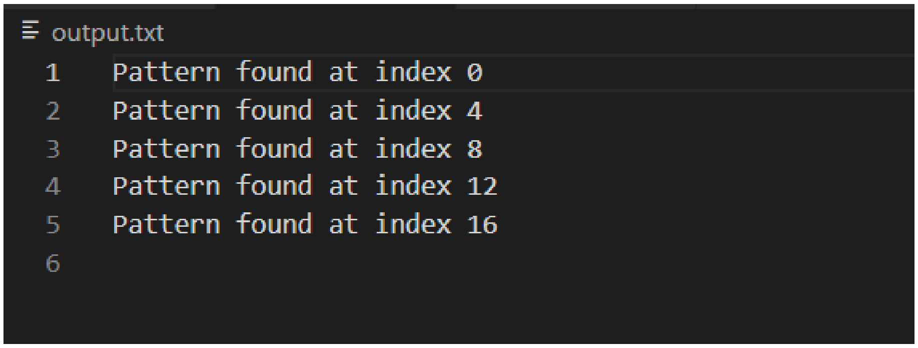
test.txt

```
Cat
Cot
Cut
Cab
Cot
```



pattern.txt

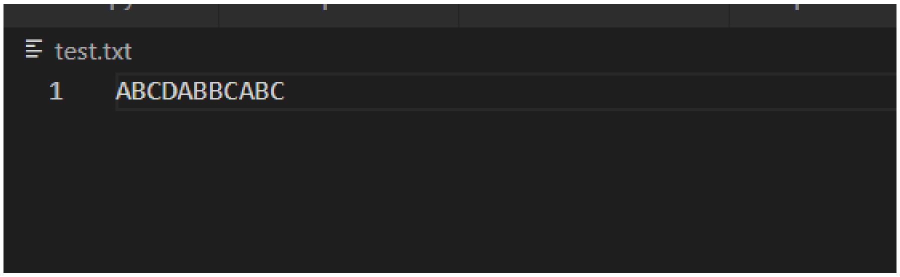
```
C.t
```



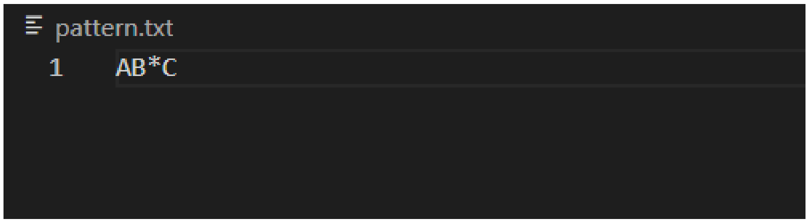
output.txt

```
Pattern found at index 0
Pattern found at index 4
Pattern found at index 8
Pattern found at index 12
Pattern found at index 16

```

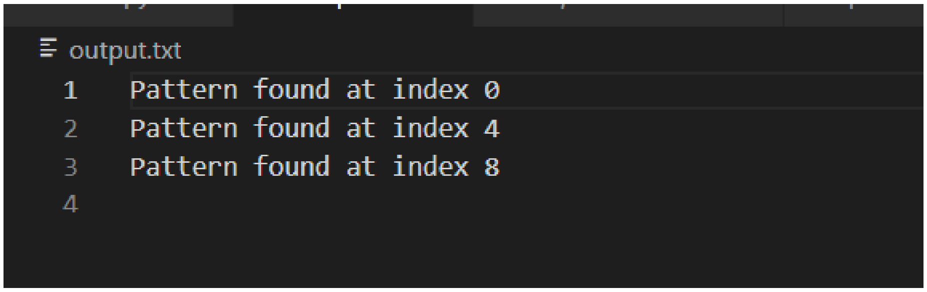
03.) “*”



```
ABCDABBCABC
```

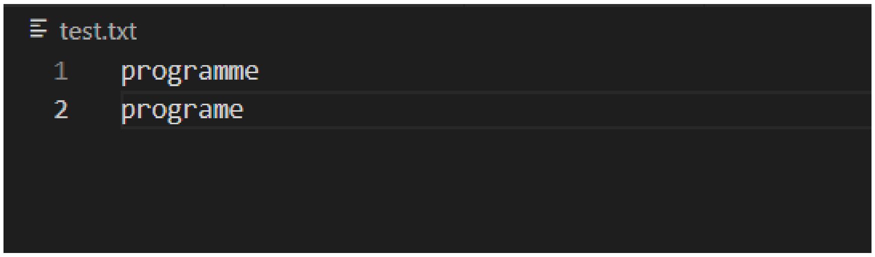


```
AB*C
```

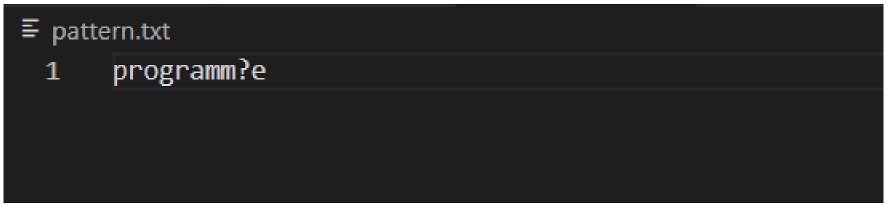


```
Pattern found at index 0
Pattern found at index 4
Pattern found at index 8
```

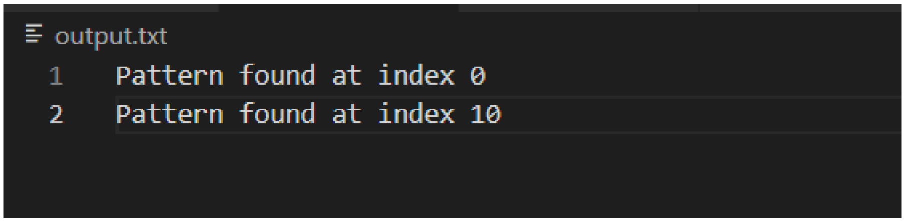
04.) “?”



```
programme
programe
```



```
programm?e
```



```
Pattern found at index 0
Pattern found at index 10
```