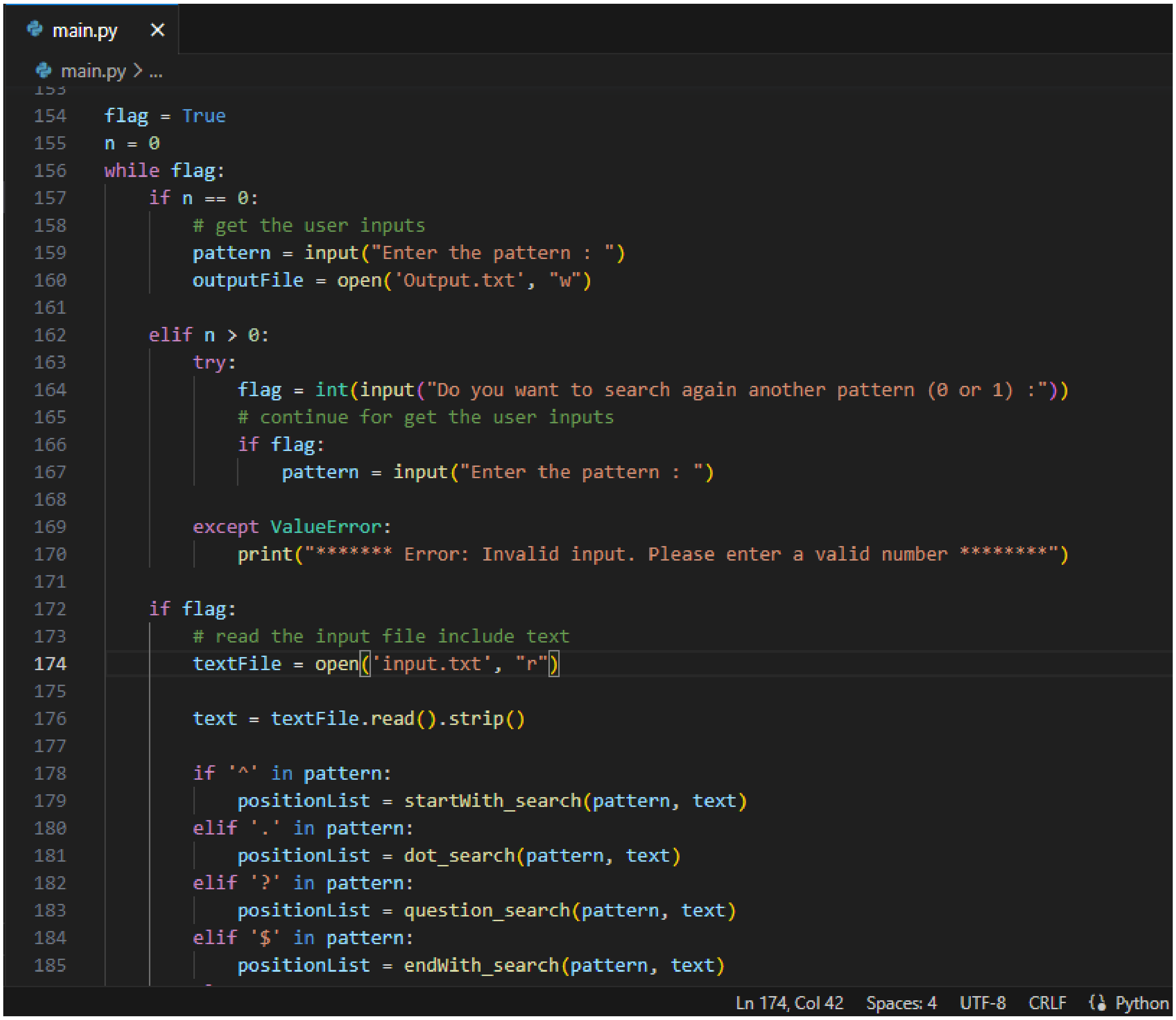
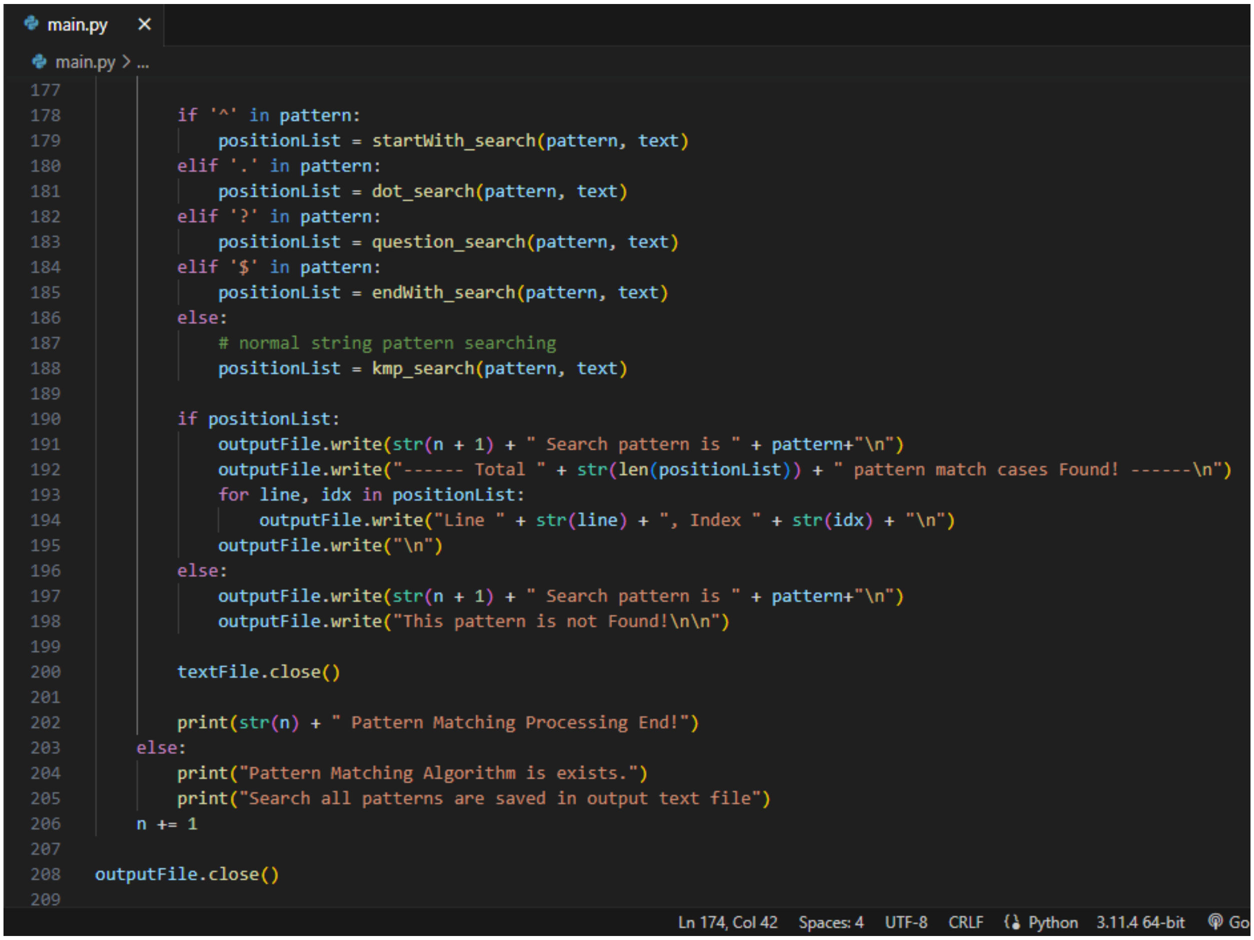
Pattern Matching Algorithm

This Python script implements various pattern matching algorithms to search for patterns in a given text file. It supports different pattern matching techniques such as Knuth-Morris-Pratt (KMP), dot (.) matching, question mark (?) matching, start (^) matching, and end ($) matching.

Code Explanation



```python
flag = True
n = 0
while flag:
    if n == 0:
        # get the user inputs
        pattern = input("Enter the pattern : ")
        outputFile = open('Output.txt', "w")

    elif n > 0:
        try:
            flag = int(input("Do you want to search again another pattern (0 or 1) :"))
            # continue for get the user inputs
            if flag:
                pattern = input("Enter the pattern : ")

        except ValueError:
            print("******* Error: Invalid input. Please enter a valid number ********")

    if flag:
        # read the input file include text
        textFile = open('input.txt', "r")

        text = textFile.read().strip()

        if '^' in pattern:
            positionList = startWith_search(pattern, text)
        elif '.' in pattern:
            positionList = dot_search(pattern, text)
        elif '?' in pattern:
            positionList = question_search(pattern, text)
        elif '$' in pattern:
            positionList = endWith_search(pattern, text)
```



```python

        if '^' in pattern:
            positionList = startWith_search(pattern, text)
        elif '.' in pattern:
            positionList = dot_search(pattern, text)
        elif '?' in pattern:
            positionList = question_search(pattern, text)
        elif '$' in pattern:
            positionList = endWith_search(pattern, text)
        else:
            # normal string pattern searching
            positionList = kmp_search(pattern, text)

        if positionList:
            outputFile.write(str(n + 1) + " Search pattern is " + pattern+"\n")
            outputFile.write("------ Total " + str(len(positionList)) + " pattern match cases Found! ------\n")
            for line, idx in positionList:
                outputFile.write("Line " + str(line) + ", Index " + str(idx) + "\n")
            outputFile.write("\n")
        else:
            outputFile.write(str(n + 1) + " Search pattern is " + pattern+"\n")
            outputFile.write("This pattern is not Found!\n\n")

        textFile.close()

        print(str(n) + " Pattern Matching Processing End!")
    else:
        print("Pattern Matching Algorithm is exists.")
        print("Search all patterns are saved in output text file")
    n += 1

outputFile.close()

```

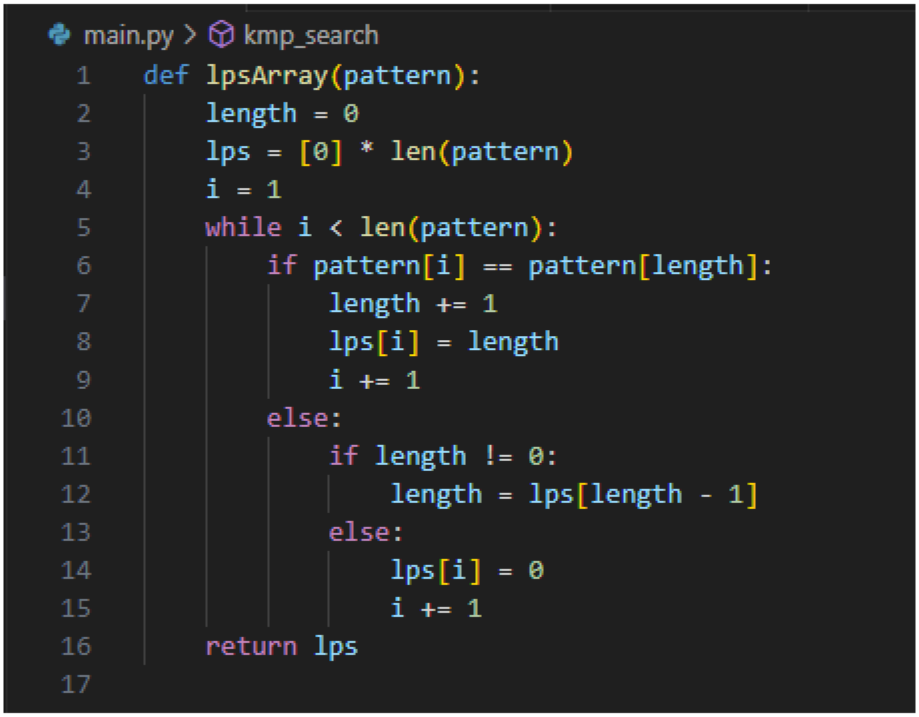
This part initializes a loop that interacts with the user. It starts by setting flag to True and n to 0. When n is 0, the program prompts the user to input a pattern and opens an output file ("Output.txt") for writing.

For n greater than zero, this part asks the user whether they want to search for another pattern. If “yes” then, the prompts the user to input another pattern. If the user enters a worng value a ValueError is caught, and an print the error message.
The code opens the "input.txt" file for reads it’s content, and stores it in the “text” variable after add strip method.

This section of code determines which pattern matching method to use based on the characters present in the entered pattern. Depending on the characters in the pattern (e.g., '^', '.', '?', '$'), condition true search the any function is called (startWith_search, dot_search, question_search, endWith_search). If none of these special characters are included, it defaults to using the kmp_search function for search to the normal string pattern.
If any matches are found, the output will save the search results, including line numbers; indices search pattern and total match found strings, in the Output.txt

file. If no matches are found, it prints the message indicating that the pattern was not found.
After processing a pattern, print the end of processing for that pattern.
If flag is false, The user chose to exit to the loop, the code prints messages indicating the end of the algorithm and informs the user that search results are saved in the ‘output.txt’ text file. Finally, after all iterations of the loop, the output file is closed.

Create a LPS Array function



```python
def lpsArray(pattern):
    length = 0
    lps = [0] * len(pattern)
    i = 1
    while i < len(pattern):
        if pattern[i] == pattern[length]:
            length += 1
            lps[i] = length
            i += 1
        else:
            if length != 0:
                length = lps[length - 1]
            else:
                lps[i] = 0
                i += 1
    return lps
```

1. Initialize “length” to track the length of matching prefix and suffix.

2. Create an array “lps” with zeros, where each index corresponds to a position in the pattern.

3. Loop through the pattern starting from the second character - at index one.

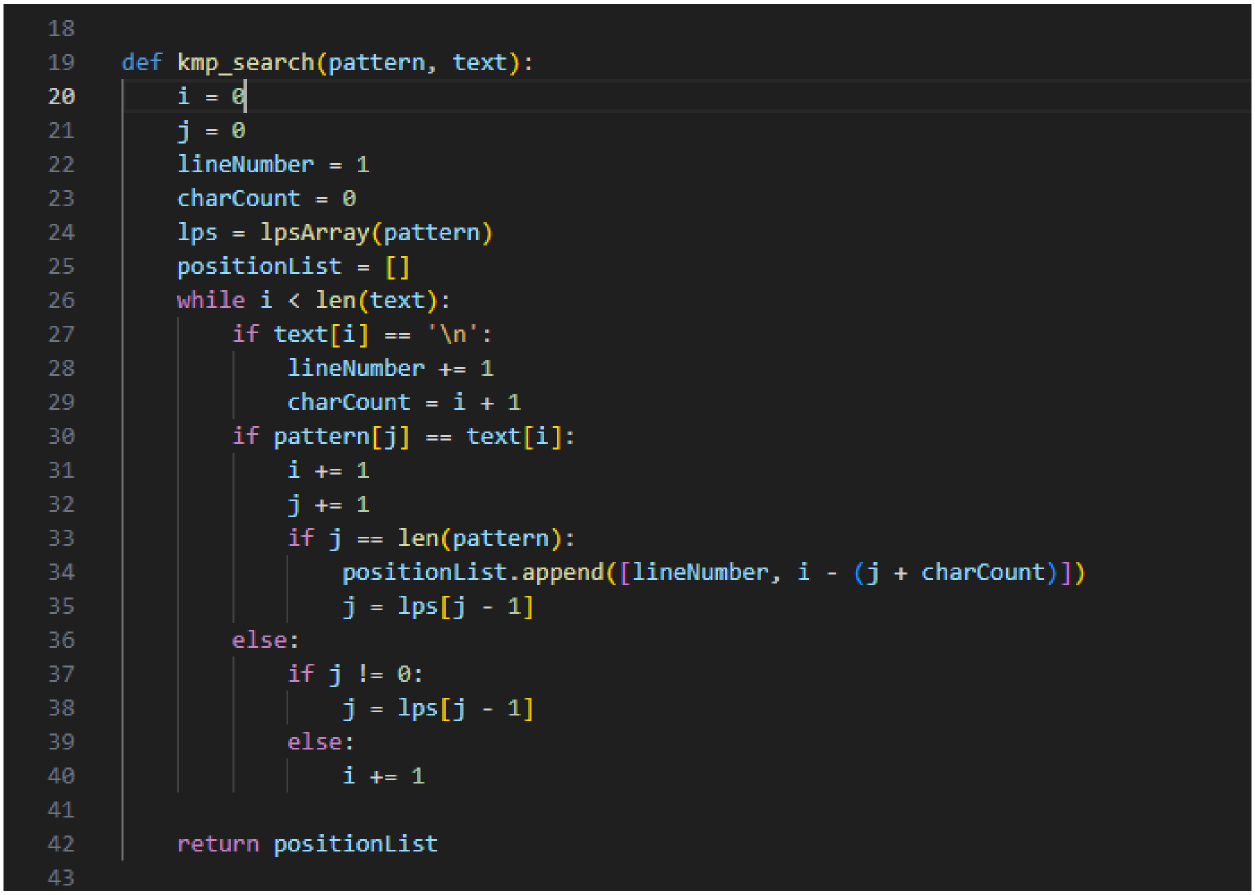
4. If the character at the current position matches the character at “length”, increment “length” and store it in “lps”.

5. If the characters don't match, update “length” based on previous matches stored in “lps”.

6. Return the computed LPS array.

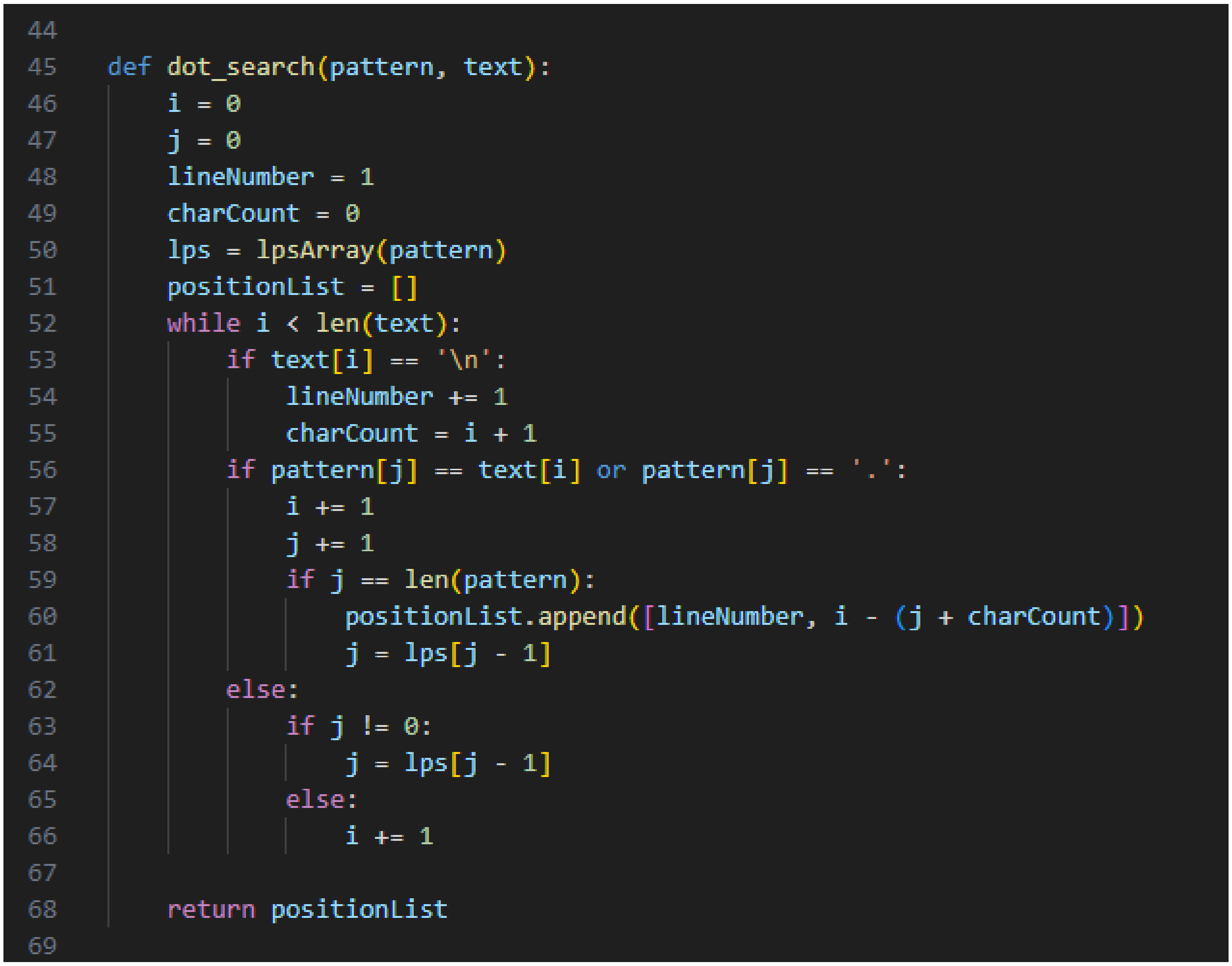
The LPS array helps the Knuth-Morris-Pratt (KMP) algorithm efficiently skip unnecessary comparisons. it can safely jump in the pattern when a mismatch is encountered. This leads to faster pattern matching.

Create a KMP Search algorithm function



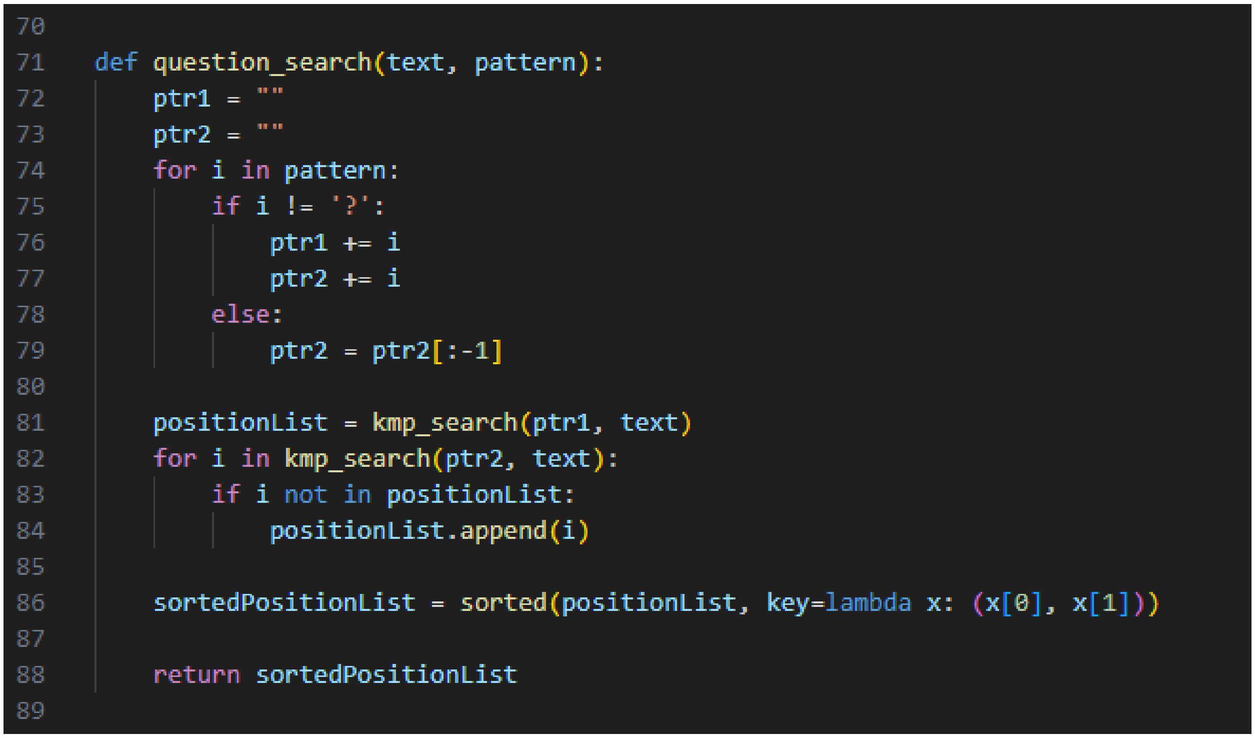
```python

def kmp_search(pattern, text):
    i = 0
    j = 0
    lineNumber = 1
    charCount = 0
    lps = lpsArray(pattern)
    positionList = []
    while i < len(text):
        if text[i] == '\n':
            lineNumber += 1
            charCount = i + 1
        if pattern[j] == text[i]:
            i += 1
            j += 1
            if j == len(pattern):
                positionList.append([lineNumber, i - (j + charCount)])
                j = lps[j - 1]
        else:
            if j != 0:
                j = lps[j - 1]
            else:
                i += 1

    return positionList

```

Dot Search Algorithm function



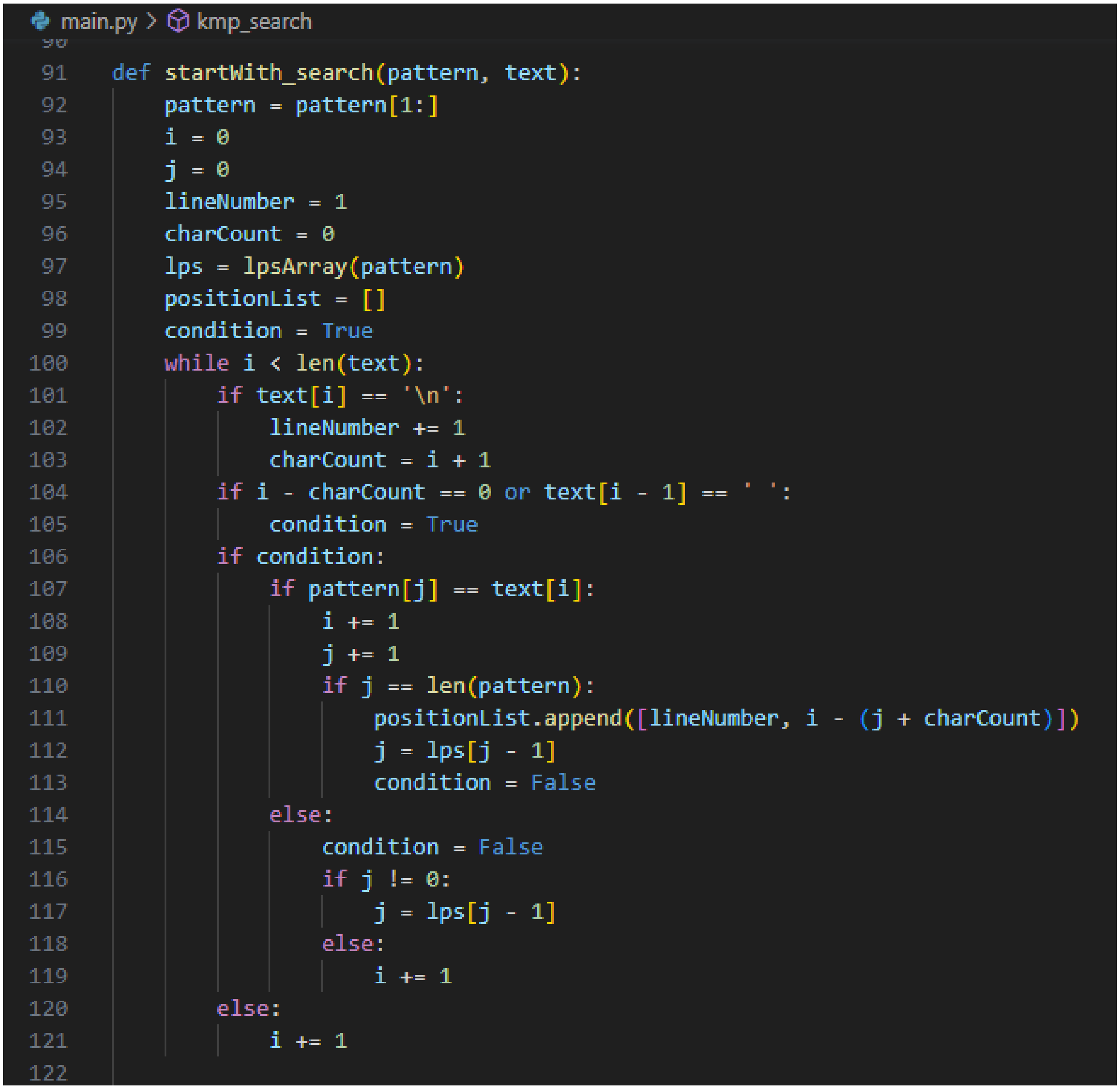
```python
def dot_search(pattern, text):
    i = 0
    j = 0
    lineNumber = 1
    charCount = 0
    lps = lpsArray(pattern)
    positionList = []
    while i < len(text):
        if text[i] == '\n':
            lineNumber += 1
            charCount = i + 1
        if pattern[j] == text[i] or pattern[j] == '.':
            i += 1
            j += 1
            if j == len(pattern):
                positionList.append([lineNumber, i - (j + charCount)])
                j = lps[j - 1]
        else:
            if j != 0:
                j = lps[j - 1]
            else:
                i += 1

    return positionList
```

Question Search Algorithm function



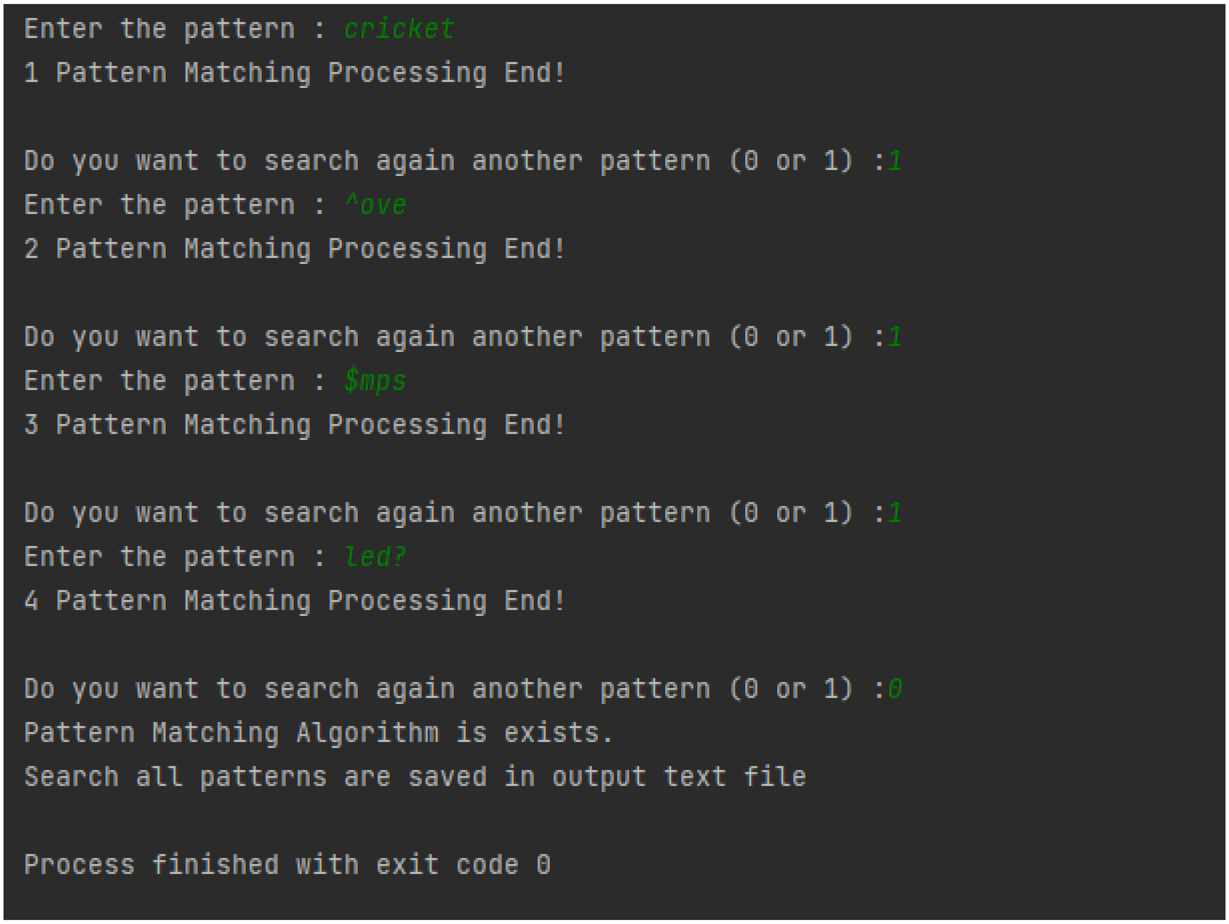
```python
def question_search(text, pattern):
    ptr1 = ""
    ptr2 = ""
    for i in pattern:
        if i != '?':
            ptr1 += i
            ptr2 += i
        else:
            ptr2 = ptr2[:-1]

    positionList = kmp_search(ptr1, text)
    for i in kmp_search(ptr2, text):
        if i not in positionList:
            positionList.append(i)

    sortedPositionList = sorted(positionList, key=lambda x: (x[0], x[1]))

    return sortedPositionList
```

Start with Search Algorithm



```python
def startWith_search(pattern, text):
    pattern = pattern[1:]
    i = 0
    j = 0
    lineNumber = 1
    charCount = 0
    lps = lpsArray(pattern)
    positionList = []
    condition = True
    while i < len(text):
        if text[i] == '\n':
            lineNumber += 1
            charCount = i + 1
        if i - charCount == 0 or text[i - 1] == ' ':
            condition = True
        if condition:
            if pattern[j] == text[i]:
                i += 1
                j += 1
                if j == len(pattern):
                    positionList.append([lineNumber, i - (j + charCount)])
                    j = lps[j - 1]
                    condition = False
            else:
                condition = False
                if j != 0:
                    j = lps[j - 1]
                else:
                    i += 1
        else:
            i += 1

```

User Inputs for Text cases



```
Enter the pattern : cricket
1 Pattern Matching Processing End!

Do you want to search again another pattern (0 or 1) :1
Enter the pattern : ^ove
2 Pattern Matching Processing End!

Do you want to search again another pattern (0 or 1) :1
Enter the pattern : $mps
3 Pattern Matching Processing End!

Do you want to search again another pattern (0 or 1) :1
Enter the pattern : led?
4 Pattern Matching Processing End!

Do you want to search again another pattern (0 or 1) :0
Pattern Matching Algorithm is exists.
Search all patterns are saved in output text file

Process finished with exit code 0
```

Output file

1 Search pattern is cricket

------ Total 6 pattern match cases Found! ------

Line 13, Index 9

Line 17, Index 14

Line 17, Index 69

Line 22, Index 26

Line 30, Index 8

Line 33, Index 47

```
2 Search pattern is ^ove
------ Total 4 pattern match cases Found! ------
Line 14, Index 50
Line 16, Index 67
Line 17, Index 63
Line 25, Index 10

3 Search pattern is led?
------ Total 10 pattern match cases Found! ------
Line 1, Index 60
Line 3, Index 70
Line 4, Index 42
Line 5, Index 29
Line 6, Index 57
Line 9, Index 59
Line 14, Index 33
Line 19, Index 35
Line 26, Index 97
Line 27, Index 22

4 Search pattern is $mps
------ Total 2 pattern match cases Found! ------
Line 3, Index 9
Line 6, Index 85
```