String Matching Algorithm

This file contains a Python implementation of a string matching algorithm that incorporates several advanced features, including wildcards, optional characters, caret symbol, and escape symbols. The algorithm can determine if a given pattern matches substrings within a target string.

Features

1. **Wildcard (.):** The algorithm allows the use of a dot **'.'** in the pattern, which matches any single character in the target string.
2. **Optional Character (?):** The algorithm supports the **'?'** symbol in the pattern, indicating that the preceding character is optional. It can appear either once or not at all in the target string.
3. **Caret Symbol (^):** If the pattern starts with **'^'** the algorithm ensures that the pattern matches only at the beginning of the target string.
4. **Escape Symbol (\\):** The backslash '\\' serves as an escape symbol. When followed by a character, it matches that character literally within the target string.

The Code

```python
# String matching naive algorithm
# add Wildcard ( . )
# add Optional character ( ? )
# add caret ( ^ ) symbol
# add The Escape Symbol (  \  )

string = "abcd"
pattern = "ab?cd"


def matched(string, pattern):
    s = len(string)
    p = len(pattern)

    # if pattern is Empty
    if not p:
        return True

    i = 0
    while True:
        # Escape Symbol (  \  )
        if i < p and i < s and pattern[i] == "\\":
            if string[i] == pattern[i+1]:
                if matched(string[i+1:], pattern[i+2:]):
                    return True
        # Optional character ( ? )
        if i+1 < p and pattern[i+1] == '?':
            # with this character
            option1 = matched(string[i:], pattern[i]+pattern[i+2:])
            # without this character
            option2 = matched(string[i:], pattern[i+2:])
            if option1 or option2:
                return True
            else:
                return False

        # Wildcard ( . )
        if i < p and pattern[i] == '.':
            i += 1
            continue

        if p <= i:
            return True
        if s <= i:
            return False
        if string[i] != pattern[i]:
            return False
        i += 1


def find(string, pattern):
    s = len(string)
    p = len(pattern)
    if pattern[0] == '^':
        if matched(string, pattern[1:]):
            print(0)
    else:
        for i in range(s):
            if matched(string[i:], pattern):
                print(i)


find(string, pattern)
```

Explanation

The code implements a string matching algorithm using various symbols to enhance the pattern matching capabilities. The algorithm supports wildcards, optional characters, escape symbols, and the caret symbol for string matching. Let's break down the code step by step:

```
def matched(string, pattern):
    s = len(string)
    p = len(pattern)

    # if pattern is Empty
    if not p:
        return True
```

The **matched** function is a recursive function that checks whether the given string matches the pattern. It takes the input string and pattern and initializes the lengths of both.

If the pattern is empty, it means that the pattern has been fully matched against the string, so the function returns **True**.

```
    i = 0
    while True:
        # Escape Symbol (  \  )
        if i < p and i < s and pattern[i] == "\\":
            if string[i] == pattern[i+1]:
                if matched(string[i+1:], pattern[i+2:]):
                    return True
```

This part of the code handles the escape symbol (\ ). If the current character in the pattern is a backslash and the next character in both the string and pattern matches, the algorithm recursively calls **matched** with the remaining string and pattern, effectively skipping these two characters.

```
    # Optional character ( ? )
    if i+1 < p and pattern[i+1] == '?':
        # with this character
        option1 = matched(string[i:], pattern[i]+pattern[i+2:])
        # without this character
        option2 = matched(string[i:], pattern[i+2:])
        if option1 or option2:
            return True
        else:
            return False
```

This section handles the optional character symbol (?). If the next character in the pattern is a question mark, it considers two options: matching the string with and without the current character. It calls **matched** recursively for both cases and returns **True** if either option matches.

```
    # Wildcard ( . )
    if i < p and pattern[i] == '.':
        i += 1
        continue
```

The code deals with the wildcard symbol (.) which matches any single character. It increments i to skip the wildcard character in the pattern and continues the loop to the next iteration.

```
    if p <= i:
        return True
    if s <= i:
        return False
    if string[i] != pattern[i]:
        return False
    i += 1
```

If none of the special symbols are found, the algorithm checks for regular character matching. If either the pattern or the string is exhausted, the function returns **True** if both have been fully matched, and **False** if either of them remains unmatched. The **i** index is incremented to move to the next character.

```
def find(string, pattern):
    s = len(string)
    p = len(pattern)
    if pattern[0] == '^':
        if matched(string, pattern[1:]):
            print(0)
    else:
        for i in range(s):
            if matched(string[i:], pattern):
                print(i)
```

The **find** function starts the process of finding matches. If the pattern starts with the caret symbol (^), it means that the pattern should match at the beginning of the string. It calls the **matched** function with the remaining string and the pattern excluding the caret. If a match is found, it prints **0**.

If the pattern doesn't start with the caret, the function iterates through each position of the string and calls the **matched** function with the remaining string and the entire pattern. If a match is found, it prints the index at which the match starts.

```
find(string, pattern)
```

Finally, the **find** function is called with the provided string and pattern, initiating the matching process.

Algorithms

1. **Wildcard (.):** The naive algorithm is well-suited for handling wildcards. Since the naive approach iterates through each character in the pattern and the string, it can easily skip over wildcard characters (.) and continue matching the subsequent characters.
2. **Optional Character (?):** The naive algorithm can handle optional characters as well. The implementation of checking two options, one with the character and one without, aligns with the iterative nature of the naive algorithm. It simply tests each option sequentially, allowing the algorithm to consider all possibilities.
3. **Caret (^) Symbol:** The caret symbol indicates that the pattern should match at the beginning of the string. The naive algorithm is able to accommodate this requirement by iterating through the string from the beginning and checking if the remaining pattern matches at each position.
4. **Escape Symbol (\ ):** The escape symbol is used to match a specific character literally, even if it's a special character in regex syntax. The implementation of checking for the backslash character and the subsequent character in the pattern closely aligns with the way the naive algorithm progresses through each character.

In this implementation, the functionalities incorporated are handled sequentially, character by character, which aligns well with the basic iteration approach of the naive algorithm. Therefore, using the naive algorithm in this context is justified and can effectively handle the given functionalities.

Test Cases

Test Cases	Expected Output	Actual Output
String the sun dipped below the horizon Patten the	0 21	0 21
String the sun dipped below the horizon Patten tel		
String the sun dipped below the horizon Patten sun	4	4
String the sun dipped below the horizon Patten ^sun		
String the sun dipped below the horizon Patten ^the sun	0	0
String the sun dipped below the horizon Patten d.pped		
String the sun dipped below the horizon Patten bew?low	15	15

Test Cases	Expected Output	Actual Output
String the sun dipped below the horizon Patten h.riz?on	25	25
String the sun dipped below the horizon. hi Patten horizon\.	25	25

** If not found, nothing will printed.