Wildcard Pattern Matching

Wildcard Pattern Matching: Given a string and a pattern containing wildcard characters, i.e., `*` and `?`, where `?` can match to any single character in the string and `*` can match to any number of characters including zero characters, design an efficient algorithm to check if the pattern matches with the complete string or not.

For example,

```
Input:  string = “xyxzzxy”, pattern = “x***y”
Output: Match

Input:  string = “xyxzzxy”, pattern = “x***x”
Output: No Match

Input:  string = “xyxzzxy”, pattern = “x***x?”
Output: Match

Input:  string = “xyxzzxy”, pattern = “*”
Output: Match
```

Practice this problem

The idea is to use dynamic programming to solve this problem. If we carefully analyze the problem, we can see that it can easily be further divided into subproblems. Let's take the top-bottom approach to solve this problem.

For a given `pattern[0…m]` and `word[0…n]`,

- If `pattern[m] == '*'`, if `*` matches the current character in the input string, move to the next character in the string; otherwise, ignore `*` and move to the next character in the pattern.
- If `pattern[m] == '?'`, ignore current characters of both string and pattern and check if `pattern[0…m-1]` matches `word[0…n-1]`.
- If the current character in the pattern is not a wildcard character, it should match the current character in the input string.

Special care has to be taken to handle base conditions:

- If both the input string and pattern reach their end, return true.
- If the only pattern reaches its end, return false.
- If only the input string reaches its end, return true only if the remaining characters in the pattern are all `*`.

Following is the top-down DP solution in C++, Java, and Python using memoization:

C++

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;

// Function that matches an input string with a given wildcard pattern
bool isMatch(string word, string pattern, int n, int m, auto &lookup)
{
```

```
    // If both the input string and pattern reach their end,
    // return true
    if (m < 0 && n < 0) {
        return true;
    }

    // If only the pattern reaches its end, return false
    else if (m < 0) {
        return false;
    }

    // If only the input string reaches its end, return true
    // if the remaining characters in the pattern are all '*'
    else if (n < 0)
    {
        for (int i = 0; i <= m; i++)
        {
            if (pattern[i] != '*') {
                return false;
            }
        }

        return true;
    }

    // If the subproblem is encountered for the first time
    if (!lookup[m][n])
    {
        if (pattern[m] == '*')
        {
            // 1. '*' matches with current characters in the input string.
            // Here, we will move to the next character in the string.

            // 2. Ignore '*' and move to the next character in the pattern
            lookup[m][n] = isMatch(word, pattern, n - 1, m, lookup) ||
                           isMatch(word, pattern, n, m - 1, lookup);
        }
        else {
            // If the current character is not a wildcard character, it
            // should match the current character in the input string
            if (pattern[m] != '?' && pattern[m] != word[n]) {
                lookup[m][n] = 0;
            }
            // check if pattern[0…m-1] matches word[0…n-1]
            else {
                lookup[m][n] = isMatch(word, pattern, n - 1, m - 1, lookup
            }
        }
    }

    return lookup[m][n];
}

// Wildcard Pattern Matching Implementation in C++
int main()
{
```

```
65      string word = "xyxzzxy";
66      string pattern = "x***x?";
67
68      int n = word.length();
69      int m = pattern.length();
70
71      // Create a DP lookup table
72      vector<vector<bool>> lookup(m + 1, vector<bool>(n + 1, false));
73
74      if (isMatch(word, pattern, n - 1, m - 1, lookup)) {
75          cout << "Match";
76      }
77      else {
78          cout << "No Match";
79      }
80
81      return 0;
82  }
```

Download Run Code

Output:

```
Match
```

Java

Python

The time complexity of the above top-down solution is `O(m.n)` and requires `O(m.n)` extra space, where `n` is the length of the text and `m` is the length of the pattern.

Following is an iterative C++, Java, and Python implementation of the above code:

C++

```
1   #include <iostream>
2   #include <string>
3   using namespace std;
4
5   // Function that matches the input string with a given wildcard pattern
6   bool isMatch(string word, string pattern)
7   {
```

```
8       // get the length of string and wildcard pattern
9       int n = word.length();
10      int m = pattern.length();
11
12      // create a DP lookup table
13      // all elements are initialized by false by default
14      bool T[n+1][m+1];
15
16      // if both pattern and string are empty: match
17      T[0][0] = true;
18
19      // handle empty string case (i == 0)
20      for (int j = 1; j <= m; j++)
21      {
22          if (pattern[j-1] == '*') {
23              T[0][j] = T[0][j-1];
24          }
25      }
26
27      // build a matrix in a bottom-up manner
28      for (int i = 1; i <= n; i++)
29      {
30          for (int j = 1; j <= m; j++)
31          {
32              if (pattern[j-1] == '*') {
33                  T[i][j] = T[i-1][j] || T[i][j-1];
34              }
35              else if (pattern[j-1] == '?' || word[i-1] == pattern[j-1]) {
36                  T[i][j] = T[i-1][j-1];
37              }
38          }
39      }
40
41      // last cell stores the answer
42      return T[n][m];
43  }
44
45  // Wildcard Pattern Matching Implementation in C
46  int main()
47  {
48      string word = "xyxzzxy";
49      string pattern = "x***x?";
50
51      if (isMatch(word, pattern)) {
52          cout << "Match" << endl;
53      }
54      else {
55          cout << "No Match" << endl;
56      }
57
58      return 0;
59  }
```

Download Run Code

```
Output:

Match
```

Java

Python

The time complexity of the above bottom-up solution is `O(m.n)` and requires `O(m.n)` extra space, where `n` is the length of the text and `m` is the length of the pattern.

Dynamic Programming, String

Algorithm, Amazon, Bottom-up, Hard, Recursive, Top-down