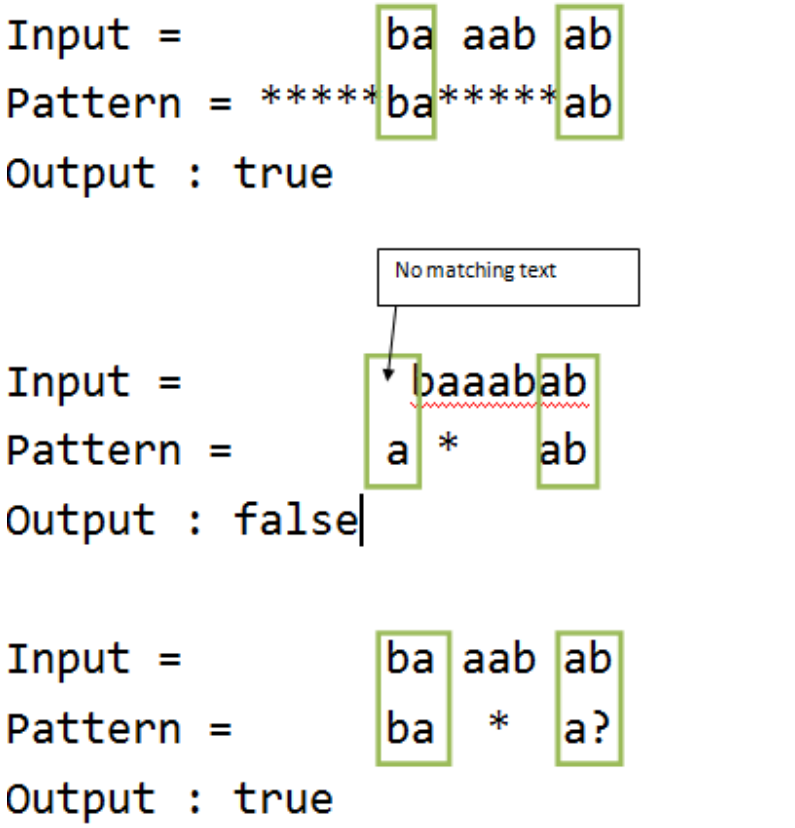
Wildcard Pattern Matching

Given a text and a wildcard pattern, implement wildcard pattern matching algorithm that finds if wildcard pattern is matched with text. The matching should cover the entire text (not partial text).

The wildcard pattern can include the characters '?' and '*'
'?' – matches any single character
'*' – Matches any sequence of characters (including the empty sequence)

For example,

Text = "baaabab",
Pattern = "*****ba*****ab", output : true
Pattern = "baaa?ab", output : true
Pattern = "ba*a?", output : true
Pattern = "a*ab", output : false



Each occurrence of '?' character in wildcard pattern can be replaced with any other character and each occurrence of '*' with a sequence of characters such that the wildcard pattern becomes identical to the input string after replacement.

Let's consider any character in the pattern.

Case 1: The character is '*'

Here two cases arise

1. We can ignore '*' character and move to next character in the Pattern.
2. '*' character matches with one or more characters in Text. Here we will move to next character in the string.

Case 2: The character is '?'

We can ignore current character in Text and move to next character in the Pattern and Text.

Case 3: The character is not a wildcard character

If current character in Text matches with current character in Pattern, we move to next character in the Pattern and Text. If they do not match, wildcard pattern and Text do not match.

We can use Dynamic Programming to solve this problem –
Let **T[i][j]** is true if first i characters in given string matches the first j characters of pattern.

Recommended: Please solve it on "*PRACTICE*" first, before moving on to the solution.

DP Initialization:

```
// both text and pattern are null
T[0][0] = true;

// pattern is null
T[i][0] = false;

// text is null
T[0][j] = T[0][j - 1] if pattern[j – 1] is '*'
```

DP relation :

```
// If current characters match, result is same as
// result for lengths minus one. Characters match
// in two cases:
```

```
// a) If pattern character is '?' then it matches
//    with any character of text.
// b) If current characters in both match
if ( pattern[j – 1] == ‘?’) ||
     (pattern[j – 1] == text[i - 1])
    T[i][j] = T[i-1][j-1]

// If we encounter ‘*’, two choices are possible-
// a) We ignore ‘*’ character and move to next
//    character in the pattern, i.e., ‘*’
//    indicates an empty sequence.
// b) '*' character matches with ith character in
//     input
else if (pattern[j – 1] == ‘*’)
    T[i][j] = T[i][j-1] || T[i-1][j]

else // if (pattern[j – 1] != text[i - 1])
    T[i][j]  = false
```

Below is the implementation of above Dynamic Programming approach.

C++

```
// C++ program to implement wildcard
// pattern matching algorithm
#include <bits/stdc++.h>
using namespace std;

// Function that matches input str with
// given wildcard pattern
bool strmatch(char str[], char pattern[],
```

```
                int n, int m)
{
    // empty pattern can only match with
    // empty string
    if (m == 0)
        return (n == 0);

    // lookup table for storing results of
    // subproblems
    bool lookup[n + 1][m + 1];

    // initailze lookup table to false
    memset(lookup, false, sizeof(lookup));

    // empty pattern can match with empty string
    lookup[0][0] = true;

    // Only '*' can match with empty string
    for (int j = 1; j <= m; j++)
        if (pattern[j - 1] == '*')
            lookup[0][j] = lookup[0][j - 1];

    // fill the table in bottom-up fashion
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= m; j++)
        {
            // Two cases if we see a '*'
            // a) We ignore ‘*’ character and move
            //    to next  character in the pattern,
            //     i.e., ‘*’ indicates an empty sequence.
            // b) '*' character matches with ith
            //     character in input
            if (pattern[j - 1] == '*')
                lookup[i][j] = lookup[i][j - 1] ||
                               lookup[i - 1][j];

            // Current characters are considered as
            // matching in two cases
            // (a) current character of pattern is '?'
            // (b) characters actually match
            else if (pattern[j - 1] == '?' ||
```

```
                    str[i - 1] == pattern[j - 1])
                lookup[i][j] = lookup[i - 1][j - 1];

            // If characters don't match
            else lookup[i][j] = false;
        }
    }

    return lookup[n][m];
}

int main()
{
    char str[] = "baaabab";
    char pattern[] = "*****ba*****ab";
    // char pattern[] = "ba*****ab";
    // char pattern[] = "ba*ab";
    // char pattern[] = "a*ab";
    // char pattern[] = "a*****ab";
    // char pattern[] = "*a*****ab";
    // char pattern[] = "ba*ab****";
    // char pattern[] = "****";
    // char pattern[] = "*";
    // char pattern[] = "aa?ab";
    // char pattern[] = "b*b";
    // char pattern[] = "a*a";
    // char pattern[] = "baaabab";
    // char pattern[] = "?baaabab";
    // char pattern[] = "*baaaba*";

    if (strmatch(str, pattern, strlen(str),
                         strlen(pattern)))
        cout <<    "Yes" << endl;
    else
        cout << "No" << endl;

    return 0;
}
```

Java

```java
// Java program to implement wildcard
// pattern matching algorithm
import java.util.Arrays;
public class GFG{

    // Function that matches input str with
    // given wildcard pattern
    static boolean strmatch(String str, String pattern,
                                int n, int m)
    {
        // empty pattern can only match with
        // empty string
        if (m == 0)
            return (n == 0);

        // lookup table for storing results of
        // subproblems
        boolean[][] lookup = new boolean[n + 1][m + 1];

        // initailze lookup table to false
        for(int i = 0; i < n + 1; i++)
            Arrays.fill(lookup[i], false);


        // empty pattern can match with empty string
        lookup[0][0] = true;

        // Only '*' can match with empty string
        for (int j = 1; j <= m; j++)
            if (pattern.charAt(j - 1) == '*')
                lookup[0][j] = lookup[0][j - 1];

        // fill the table in bottom-up fashion
        for (int i = 1; i <= n; i++)
        {
            for (int j = 1; j <= m; j++)
            {
                // Two cases if we see a '*'
                // a) We ignore '*'' character and move
                //    to next  character in the pattern,
                //     i.e., '*' indicates an empty sequence.
```

```
                // b) '*' character matches with ith
                //     character in input
                if (pattern.charAt(j - 1) == '*')
                    lookup[i][j] = lookup[i][j - 1] ||
                                   lookup[i - 1][j];

                // Current characters are considered as
                // matching in two cases
                // (a) current character of pattern is '?'
                // (b) characters actually match
                else if (pattern.charAt(j - 1) == '?' ||
                    str.charAt(i - 1) == pattern.charAt(j - 1))
                    lookup[i][j] = lookup[i - 1][j - 1];

                // If characters don't match
                else lookup[i][j] = false;
            }
        }

        return lookup[n][m];
    }

    public static void main(String args[])
    {
        String str = "baaabab";
        String pattern = "*****ba*****ab";
        // String pattern = "ba*****ab";
        // String pattern = "ba*ab";
        // String pattern = "a*ab";
        // String pattern = "a*****ab";
        // String pattern = "*a*****ab";
        // String pattern = "ba*ab****";
        // String pattern = "****";
        // String pattern = "*";
        // String pattern = "aa?ab";
        // String pattern = "b*b";
        // String pattern = "a*a";
        // String pattern = "baaabab";
        // String pattern = "?baaabab";
        // String pattern = "*baaaba*";

        if (strmatch(str, pattern, str.length(),
```

```
                                pattern.length()))
                System.out.println("Yes");
            else
                System.out.println("No");

        }
    }
    // This code is contributed by Sumit Ghosh
```

Output :

```
Yes
```

Time complexity of above solution is O(m x n). Auxiliary space used is also O(m x n).

Further Improvements:

We can improve space complexity by making use of the fact that we only uses the result from last row.

One more improvement is yo merge consecutive '*' in the pattern to single '*' as they mean the same thing. For example for pattern "*****ba*****ab", if we merge consecutive stars, the resultant string will be "*ba*ab". So, value of m is reduced from 14 to 6.

This article is contributed by **Aditya Goel**. If you like GeeksforGeeks and would like to contribute, you can also write an article and mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Recommended Posts:

WildCard pattern matching having three symbols (* , + , ?)

Dynamic Programming | Wildcard Pattern Matching | Linear Time and Constant Space

String matching where one string contains wildcard characters

Print all words matching a pattern in CamelCase Notation Dictonary

String matching with * (that matches with any) in any of the two strings

Longest Common Prefix Matching | Set-6

Prefix matching in Python using pytrie module

Longest Common Prefix using Word by Word Matching

Longest Common Prefix using Character by Character Matching

Pattern Searching using C++ library

Check if a string follows a^nb^n pattern or not

Print a pattern without using any loop

KMP Algorithm for Pattern Searching

Pattern Searching using a Trie of all Suffixes

Pattern Searching using Suffix Tree

Article Tags : Dynamic Programming | Pattern Searching | Strings | Amazon | InMobi | Microsoft | Ola Cabs | United Health Group | Walmart

Practice Tags : Microsoft | Amazon | Ola Cabs | Walmart | InMobi | United Health Group | Strings | Dynamic Programming | Pattern Searching

2

4

To-do Done

Based on **122** vote(s)

Feedback Add Notes Improve Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments Share this post!

A computer science portal for geeks

710-B, Advant Navis Business Park,
Sector-142, Noida, Uttar Pradesh - 201305
feedback@geeksforgeeks.org

COMPANY

About Us
Careers
Privacy Policy
Contact Us

LEARN

Algorithms
Data Structures
Languages
CS Subjects
Video Tutorials

PRACTICE

Company-wise
Topic-wise
Contests
Subjective Questions

CONTRIBUTE

Write an Article
Write Interview Experience
Internships
Videos