Lets first understand how to approach ths and solve this, as this is not regular general question (because of "\*" ) we need to try and hit few example to understand what really happen

whenever we try to match any two character of pattern and text.

Lets take very simple example first;

Example 1: text = "" and pattern = "" [ Both are empty or null ].

Approach: Its clear that if pattern is null or empty then text has to be either null or empty, then only they match.

Hence

Deduction 1: if ( p == null || p.isEmpty() ) return (text == null || text.isEmpty() )

Now onwards i'll make the example more complex but will do bit by bit so that we can find out the pattern

Example 2: text = "a" and pattern ="" [ text is not empty but patter is ]. We hit the above condition and we result to "False"

Example 3: text = "a" and pattern = "a" [ both have single character ]

Approach: as we can simply see both have single character and they "match" so our result is True

Deduction 2: if both character in pattern and text match, then we say they “match Probably” [ I’ll explain probably in a while ]

Example 4: text=”ab” and pattern = “a”

Approach: We see at index 0 in both they match, but there still some character left in pattern ( not ‘.’ And ‘\*’ ) so they don’t match.

Deduction 3: if there is some character remaining in pattern after matching complete text then they don’t match= false ; provided that remaining characters are not ‘.’ And ‘\*’

I think, now you have pretty good idea about character vs character

Lets introduce some ‘.’.

Example 5: text=”ab” and pattern = “a.” [ patter has 1 ‘.’ At the last ]

Approach: As both first character are match we left with text=”b” and pattern=”.” , Since “.” Says it can match to any character, hence we left with text = “” and pattern=”” and we have the solution for this already. Case 1; Hence True

Deduction 4: Both character and “.” Behaves in same way, either both are character in pattern and text and they match or pattern is a “.” Also that match provided that previous text and pattern match.

Let make it more complex

Example 5: text=”ab” and pattern = “a..” [ patter has 2 ‘.’ At the last ]

Approach: We saw ‘a’ and ‘a’ match, we left with “b” and “..”. Now we hit a dot, then we’ll see does “” and “.” Match, which says No as text is empty and pattern is not. Hence False.

Deduction 5: Just same as Deduction 4 with minor tweak.

Lets introduce the killer of this question “\*”

Example 6: text=”ab” and pattern = “a\*” [ patter has 1 ‘\*’ At the last ]

Approach: We know that “\*” means that Either zero or at least 1 character match with “just previous to \*”. [ That’s important point ]

Which means that we can assume that there is no character before “\*” and there is at least 1 character same as previous of “\*”.

Hence

In our example;

“ab” -> “a\*” => “b” -> “\*” => Don’t match Hence False

Example 7: text=”ab” and pattern = “ab\*” [ patter has 1 ‘\*’ At the last ]

“ab” -> “ab\*” => “b” -> “b\*”

Now we are at the critical point, what happens here?

There is two path possible ,

Either we say

“b” -> “” [ zero occurrence of b ] => False

“b” -> “b” [ 1 occurrence of b ] => True

“b” -> “bb” [ We don’t need to move ahead as that enough to test, in this case ]

Deduction 6: Whenever we hit star “\*” **we need to look back 1 or 2 character behind**.

2 because we are assuming that there no occurrence of previous character (of “\*” ) or 1 because there at least 1 occurrence of previous character (of “\*”).

Deduction: If you observe carefully, each time we have to look back what was the solution if don’t have the current pattern character ( either character or “.” OR ‘\*” ].

Which tells us to cache the solution Hence DP [ as this tells if you know previous solution you can build this solution and they are overlap too ].

Let’s taka a final example to prove our point

Example: text = “xaabyc” pattern = “xa\*b.c” [ has 1 \* and 1 “.” ]

Lets match

“xaabyc” -> “xa\*b.c” => “aabyc” -> “a\*b.c”

=> “abyc” -> “\*b.c” OR

=> “abyc” -> “ab.c”

=> “byc” -> “b.c” => “yc” -> “.c” => “c” -> “c” => “” -> “” Hence Matched.

[Please note, for simplicity I did not wrote all the paths, when ever we hit “\*” we have two path.]

Now lets come to more difficult path; where you have like this “\*\*…\*” or “.\*” Or “..\*” or “.\*\*” … like this

We can assume it same as “.\*” -> “$\*” where $ represent any character, now again your problem reduce to same, isn’t it ☺

Last but not least, there is one special case like

Text =”abc” and pattern =”\*df”

In this case we need to loop before what was the solution as there is no character previous to “\*”.

Let’s write a recurrence relation and base case.

Base case:

If both string are empty = True

If one of them is empty but other is not = False.

Let’s Say

M[i][j] Defines the our solution dp.

**‘true’** when we can match 0 to i character of text are tested against 0 to j character of pattern and they match

Otherwise **‘False’**

**Text[i-1] == Pattern[j-1] ; Then we need to look back what was state when current character don’t exist => M[i-1][j-1]**

**Text[i-1] != Pattern[j-1]; Now there could be following reason that they are not equal**

1. The character at both are different like Text[i-1] = ‘A’ and Pattern[j-1] = ‘B’ => False
2. The character at pattern is ‘.’ => True
3. The character at pattern is “\*” => Look back either 2 character before (assume previous character don’t exist Or 1 character before assume at least 1 occurrence is there of previous character. Now previous character could be “.” Or a character

If “.” Then it can match to any one Or if both character match then What was the last state of text matching pattern, which is M[i-1][j]

M[i][j] = {

**Text[i-1] == Pattern[j-1] =>** M[i-1][j-1]

**Text[i-1] != Pattern[j-1] and** **Pattern[j-1]!=’.’ and Pattern[j-1]!=’\* =>** False

**Pattern[j-1] == ‘.’ =>** M[i-1][j-1]

**Pattern[j-1] == ‘\*’ =>** M[i][j-2] | (if **Pattern[j-2] == ‘.’ Or Pattern[j-2] == Text[i-1]**) M[i-1][j]

1 if p.charAt(j-1) != s.charAt(i) : dp[i][j] = dp[i][j-2] //in this case, a\* only counts as empty

2 if p.charAt(i-1) == s.charAt(i) or p.charAt(i-1) == '.':

dp[i][j] = dp[i-1][j] //in this case, a\* counts as multiple a

or dp[i][j] = dp[i][j-1] // in this case, a\* counts as single a

or dp[i][j] = dp[i][j-2] // in this case, a\* counts as empty