10. Regular Expression Matching 🗹 Sept. 13, 2017 | 325.4K views

Average Rating: 4.22 (203 votes)

'.' Matches any single character. '*' Matches zero or more of the preceding element.

Given an input string (s) and a pattern (p), implement regular expression matching with support for '.'

```
The matching should cover the entire input string (not partial).
```

Note:

• p could be empty and contains only lowercase letters a-z, and characters like . or *. Example 1:

and '*'.

• s could be empty and contains only lowercase letters a-z.

Input: s = "aa"

Output: false

p = "a"

```
Explanation: "a" does not match the entire string "aa".
Example 2:
  Input:
 s = "aa"
  p = "a*"
```

```
Example 3:
  Input:
  s = "ab"
  p = ".*"
```

Input:

```
Example 5:
 Input:
 s = "mississippi"
 p = "mis*is*p*."
 Output: false
```

Algorithm

Python

3

4

5

6 7 8

9

def match(text, pattern): 1 2 if not pattern: return not text

Without a Kleene star, our solution would look like this:

any of these operations, then the initial inputs matched.

if not pattern:

return not text

Сору Python Java class Solution(object): 2 def isMatch(self, text, pattern):

of the pattern, or delete a matching character in the text. If we have a match on the remaining strings after

```
Complexity Analysis
   ullet Time Complexity: Let T,P be the lengths of the text and the pattern respectively. In the worst case, a
```

call to ${\sf match(text[i:], pattern[2j:])}$ will be made $\binom{i+j}{i}$ times, and strings of the order O(T-1)

i) and O(P-2*j) will be made. Thus, the complexity has the order $\sum_{i=0}^T \sum_{j=0}^{P/2} {i+j \choose i} O(T+j)$

P-i-2j). With some effort outside the scope of this article, we can show this is bounded by

involving smaller strings. **Algorithm**

Top-Down Variation

Java

1

2

3 4

5

6

7

8

9

10

11

Python

class Solution(object):

 $memo = \{\}$

def dp(i, j):

def isMatch(self, text, pattern):

12 13 14 15

Bottom-Up Variation

Java

1 2

3

4

Python

class Solution(object):

def isMatch(self, text, pattern):

if (i, j) not in memo: if j == len(pattern): ans = i == len(text)else: first_match = i < len(text) and pattern[j] in {text[i], '.'}</pre> if j+1 < len(pattern) and pattern[j+1] == '*':</pre> ans = dp(i, j+2) or first_match and dp(i+1, j)else: ans = first_match and dp(i+1, j+1)

```
8
                      first_match = i < len(text) and pattern[j] in {text[i], '.'}
  9
                      if j+1 < len(pattern) and pattern[j+1] == '*':</pre>
  10
                          dp[i][j] = dp[i][j+2] or first_match and dp[i+1][j]
  11
                      else:
  12
                          dp[i][j] = first_match and <math>dp[i+1][j+1]
  13
              return dp[0][0]
  14
Complexity Analysis
     complexity is O(TP).
     space complexity is O(TP).
Rate this article: * * * * *
```

SHOW 14 REPLIES

O Previous

Comments: 180

Preview

agree.

Type comment here... (Markdown is supported)

buoy08 ★ 821 ② November 4, 2018 11:34 AM

Regular Expression ...

s961206 ★ 733 ② February 26, 2019 4:49 AM

53 A V C Share Reply

thwin 🖈 17 🗿 August 17, 2018 2:34 PM

Read More 214 A V C Share Reply **SHOW 18 REPLIES**

https://www.youtube.com/watch?v=l3hda49XcDE&list=PLrmLmBdmllpuE5GEMDXWf0PWbBD9Ga1lO

Read More

Read More

SHOW 4 REPLIES

SHOW 5 REPLIES

- var isMatch = function(s, p) { var regex = new RegFxn('^' + n + '\$'. 'm'): Read More
- Thanks for posting this. Has someone taken another crack at analyzing the runtime complexity of the recursive approach? As currently written, it's impossible to parse. 14 A V Share Share Reply

misled * 31 ② July 2, 2018 12:23 AM

SHOW 9 REPLIES

- combinations without repetitions here?? I dont understand what the mathematical formula means 12 A V 🗗 Share 🦘 Reply SHOW 1 REPLY
- In approach 2 (Top-Down Variation): As recursive call dp(i, j+2, text, pattern), dp(i+1, j, text, pattern), and dp(i+1, j+1, text, pattern), i and j
- has increased, so

- Output: true Explanation: '*' means zero or more of the preceding element, 'a'. Therefore, by repeat Output: true
- Explanation: ".*" means "zero or more (*) of any character (.)". Example 4:

s = "aab"p = "c*a*b"Output: true Explanation: c can be repeated 0 times, a can be repeated 1 time. Therefore, it matche

Solution Approach 1: Recursion Intuition

If there were no Kleene stars (the * wildcard character for regular expressions), the problem would be easier

When a star is present, we may need to check many different suffixes of the text and see if they match the

Сору

Сору

Сору

Next 👀

Sort By ▼

Post

A Report

- we simply check from left to right if each character of the text matches the pattern.

rest of the pattern. A recursive solution is a straightforward way to represent this relationship.

3 first_match = bool(text) and pattern[0] in {text[0], '.'} 4 return first_match and match(text[1:], pattern[1:]) If a star is present in the pattern, it will be in the second position pattern[1]. Then, we may ignore this part

10 first_match and self.isMatch(text[1:], pattern)) else: 11 12 return first_match and self.isMatch(text[1:], pattern[1:])

first_match = bool(text) and pattern[0] in {text[0], '.'}

if len(pattern) >= 2 and pattern[1] == '*':

return (self.isMatch(text, pattern[2:]) or

```
O((T+P)2^{T+\frac{P}{2}}).
   • Space Complexity: For every call to match, we will create those strings as described above, possibly
     creating duplicates. If memory is not freed, this will also take a total of Oig((T+P)2^{T+rac{P}{2}}ig) space, even
     though there are only order O(T^2+P^2) unique suffixes of P and T that are actually required.
Approach 2: Dynamic Programming
Intuition
As the problem has an optimal substructure, it is natural to cache intermediate results. We ask the question
```

dp(i, j): does text[i:] and pattern[j:] match? We can describe our answer in terms of answers to questions

We proceed with the same recursion as in Approach 1, except because calls will only ever be made to

match(text[i:], pattern[j:]), we use dp(i, j) to handle those calls instead, saving us expensive

memo[i, j] = ansreturn memo[i, j] 16 17 18 return dp(0, 0)

dp = [[False] * (len(pattern) + 1) for _ in range(len(text) + 1)]

string-building operations and allowing us to cache the intermediate results.

```
dp[-1][-1] = True
5
           for i in range(len(text), -1, -1):
6
7
               for j in range(len(pattern) - 1, -1, -1):
ullet Time Complexity: Let T,P be the lengths of the text and the pattern respectively. The work for every
  call to dp(i, j) for i = 0, ..., T; j = 0, ..., P is done once, and it is O(1) work. Hence, the time
ullet Space Complexity: The only memory we use is the O(TP) boolean entries in our cache. Hence, the
```

lanrengufeng ★ 292 ② July 12, 2018 12:48 PM 214 A V C Share Reply SHOW 1 REPLY Reputation SapeginRenat ★ 214 ② March 19, 2019 6:28 PM A Report My java funny solution class Solution { public boolean isMatch(String s, String p) { return s.matches(n): A Report buddhiboyzz ★ 98 ② July 16, 2018 2:12 AM

How intuitive is dp solution during 45 min of interview. Only if somebody has crammed it. How many

bettertobeinpeace * 37 O December 31, 2018 7:52 AM as import Since no one has proposed the NFA solution, I have implemented it: class Solution { //build a NEA

Could anybody teach me why solution 1's complexity? Where does C(i+j, i) come from?

17 A V C Share Reply

I think the contest should ban some keywords for specific language.

I use javascript and this code solved Iol. Did I cheat? I felt like I did anyway.

- leonhartsberger 🛊 37 🗿 December 26, 2018 3:33 PM In the runtime complexity calculation of the recursion: what does the (i+j over i) mean? do they mean
- woodyjc ★ 14 ② January 29, 2019 8:35 AM A Report
- memo[i][j] will be always null . Why do the solutions to the sub-problems need to be saved ? Is that Read More
- **SHOW 5 REPLIES**

(1 2 3 4 5 6 ... 17 18 >