☰ Articles ›

❮ Previous (/articles/merge-k-sorted-list/) Next ❯ (/articles/non-decreasing-array/)

10. Regular Expression Matching (/problems/regular-expression-matching/)

Sept. 13, 2017 | 99.8K views

Average Rating: 4.53 (32 votes)

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

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

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

Note:

- `s` could be empty and contains only lowercase letters `a-z` .
- `p` could be empty and contains only lowercase letters `a-z` , and characters like `.` or `*` .

Example 1:

```
Input:
s = "aa"
p = "a"
Output: false
Explanation: "a" does not match the entire string "aa".
```

Example 2:

```
Input:
s = "aa"
p = "a*"
Output: true
Explanation: '*' means zero or more of the precedeng element, 'a'. Therefore, by repeating 'a' on
```

Example 3:

```
Input:
s = "ab"
p = ".*"
Output: true
Explanation: ".*" means "zero or more (*) of any character (.)".
```

Example 4:

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

Example 5:

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

Solution

Approach 1: Recursion

Intuition

Articles ›

If there were no Kleene stars (the `*` wildcard character for regular expressions), the problem would be easier - we simply check from left to right if each character of the text matches the pattern.

When a star is present, we may need to check many different suffixes of the text and see if they match the rest of the pattern. A recursive solution is a straightforward way to represent this relationship.

Algorithm

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

Python Copy

```python
def match(text, pattern):
    if not pattern: return not text
    first_match = bool(text) and pattern[0] in {text[0], '.'}
    return first_match and match(text[1:], pattern[1:])
```

If a star is present in the pattern, it will be in the second position $\text{pattern}[1]$. Then, we may ignore this part of the pattern, or delete a matching character in the text. If we have a match on the remaining strings after any of these operations, then the initial inputs matched.

Java Python Copy

```java
class Solution {
    public boolean isMatch(String text, String pattern) {
        if (pattern.isEmpty()) return text.isEmpty();
        boolean first_match = (!text.isEmpty() &&
                               (pattern.charAt(0) == text.charAt(0) || pattern.charAt(0) == '.'));

        if (pattern.length() >= 2 && pattern.charAt(1) == '*'){
            return (isMatch(text, pattern.substring(2)) ||
                    (first_match && isMatch(text.substring(1), pattern)));
        } else {
            return first_match && isMatch(text.substring(1), pattern.substring(1));
        }
    }
}
```

Complexity Analysis

- Time Complexity: Let T, P be the lengths of the text and the pattern respectively. In the worst case, a call to `match(text[i:], pattern[2j:])` will be made $\binom{i+j}{i}$ times, and strings of the order $O(T - i)$ and $O(P - 2 * j)$ will be made. Thus, the complexity has the order $\sum_{i=0}^{T} \sum_{j=0}^{P/2} \binom{i+j}{i} O(T + P - i - 2j)$. With some effort outside the scope of this article, we can show this is bounded by $O\big((T+P)2^{T+\frac{P}{2}}\big)$.
- 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 $O\big((T+P)2^{T+\frac{P}{2}}\big)$ 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 $\mathrm{dp(i, j)}$: does $\mathrm{text[i:]}$ and $\mathrm{pattern[j:]}$ match? We can describe our answer in terms of answers to questions involving smaller strings.

Algorithm

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 $\mathrm{dp(i, j)}$ to handle those calls instead, saving us expensive string-building operations and allowing us to cache the intermediate results.

Top-Down Variation

Java Python Copy

```
        memo = new Result[text.length() + 1][pattern.length() + 1];
        return dp(0, 0, text, pattern);
    }

    public boolean dp(int i, int j, String text, String pattern) {
        if (memo[i][j] != null) {
            return memo[i][j] == Result.TRUE;
        }
        boolean ans;
        if (j == pattern.length()){
            ans = i == text.length();
        } else{
            boolean first_match = (i < text.length() &&
                                   (pattern.charAt(j) == text.charAt(i) ||
                                    pattern.charAt(j) == '.'));

            if (j + 1 < pattern.length() && pattern.charAt(j+1) == '*'){
                ans = (dp(i, j+2, text, pattern) ||
                       first_match && dp(i+1, j, text, pattern));
            } else {
                ans = first_match && dp(i+1, j+1, text, pattern);
            }
        }
        memo[i][j] = ans ? Result.TRUE : Result.FALSE;
        return ans;
    }
}
```

Articles ›

10. Regular Expression Matching ▾

Bottom-Up Variation

☰ Articles ›

Java Python Copy

```
class Solution {
    public boolean isMatch(String text, String pattern) {
        boolean[][] dp = new boolean[text.length() + 1][pattern.length() + 1];
        dp[text.length()][pattern.length()] = true;

        for (int i = text.length(); i >= 0; i--){
            for (int j = pattern.length() - 1; j >= 0; j--){
                boolean first_match = (i < text.length() &&
                                       (pattern.charAt(j) == text.charAt(i) ||
                                        pattern.charAt(j) == '.'));
                if (j + 1 < pattern.length() && pattern.charAt(j+1) == '*'){
                    dp[i][j] = dp[i][j+2] || first_match && dp[i+1][j];
                } else {
                    dp[i][j] = first_match && dp[i+1][j+1];
                }
            }
        }
        return dp[0][0];
    }
}
```

Complexity Analysis

- 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 complexity is $O(TP)$.
- Space Complexity: The only memory we use is the $O(TP)$ boolean entries in our cache. Hence, the space complexity is $O(TP)$.

Rate this article:

Previous (/articles/merge-k-sorted-list/)

Next (/articles/non-decreasing-array/)

Comments: 83 Sort By ▾

Login to Comment (/accounts/login/?next=/articles/regular-expression-matching/)

zhengzhicong (/zhengzhicong) ★ 47 November 8, 2018 4:14 AM

python3:

Articles ›

```
class Solution:
    def isMatch(self, s, p):
        """
```

Read More

0 Share

buoy08 (/buoy08) ★ 59 November 3, 2018 11:04 PM

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

38 Share

akkk33 (/akkk33) ★ 4 November 3, 2018 7:19 PM

So this is the best attempt I made with python 3 using built-in module `re`

```
class Solution:
    def isMatch(self, s, p):
        """
```

Read More

0 Share

SHOW 1 REPLY

WeiGrand (/weigrand) ★ 7 October 29, 2018 9:08 PM

```
var isMatch = function(s, p) {
    return new RegExp(`^${p}$`).test(s);
};
```

0 Share

fwanggg (/fwanggg) ★ 11 October 21, 2018 10:07 AM

dp-topdown approach seems incorrect to me. i,j should start from text.length and pattern.length just like the dp-bottomup approach. Otherwise, dp[i+1][j+1] gets set first before dp[i][j] does.

0 Share

xytjcxy (/xytjcxy) ★ 0 October 16, 2018 6:00 PM

if I cin s="abc"，p="ab*abc"，the result I expect is true，but the result I get from running code is false. So I think maybe there are some misunderstanding.

0 Share

SHOW 1 REPLY

nkeng (/nkeng) ★ 19 October 14, 2018 9:04 AM

for button up why does j start at pattern.length - 1 while i starts at text.length?

0 Share

SHOW 2 REPLIES

h11129 (/h11129) ★ 5 October 5, 2018 12:59 PM

The problem is misleading because it doesn't say "*" match only one element before it, which make the problem much more easier

0 Share

sxy1993sxy2018 (/sxy1993sxy2018) ★ 0 September 28, 2018 6:04 PM

I can't understand why the ouput of Example 3 is "true". '.' just represent one or zero charactor, but "ab" has two.

0 Share

SHOW 4 REPLIES

slz250 (/slz250) ★ 21 September 27, 2018 7:48 AM

Could someone explain the following about the bottom-up DP solution?

1. why is the outer for loop starting at len(text)? An unnecessary check b.c none of the conditionals wyd be fulfilled.
2. why do we even have dp[-1][-1] and why are the dimensions of our dp matrix len(text)+1 by len(pattern) + 1. The additional row and col seem unnecessary.

Read More

2 Share

SHOW 1 REPLY

‹ 1 2 3 4 5 6 ... 8 9 ›

☰ Articles ›

Contact Us (/support/) | FAQ (/faq/) | Terms (/terms/) | Privacy Policy (/privacy/)

United States (/region/)