Master algorithms together on Binary Search! Create a room, invite your friends, and race to finish the problems.

Daily Coding Problem #25

Problem

This problem was asked by Facebook.

Implement regular expression matching with the following special characters:

- . (period) which matches any single character
- * (asterisk) which matches zero or more of the preceding element

That is, implement a function that takes in a string and a valid regular expression and returns whether or not the string matches the regular expression.

For example, given the regular expression "ra." and the string "ray", your function should return true. The same regular expression on the string "raymond" should return false.

Given the regular expression ".*at" and the string "chat", your function should return true. The same regular expression on the string "chats" should return false.

Solution

This problem should strike you as recursive. The string should match the regex if we can match the head of the string with the head of the regex and the rest of the string with the rest of the regex. The special characters `.` and * make implementing this a bit trickier, however, since the * means we can match 0 or any number of characters in the beginning.

The basic idea then is to do the following. Let's call the string we want to match `s` and the regex `r`.

- Base case: if `r` is empty, then return whether `s` is empty or not.
- Otherwise, if the first thing in `r` is not proceeded by a *, then match the first character of both `r` and `s`, and if they match, return `match(r[1:], s[1:])`. If they don't, then return false.
- If the first thing in `r` _is_ proceeded by a *, then try every suffix substring of `s` on `r[2:]` and return true if any suffix substring works.

The code should look something like this:

```
def matches_first_char(s, r):
    return s[0] == r[0] or (r[0] == '.' and len(s) > 0)

def matches(s, r):
    if r == '':
        return s == ''

    if len(r) == 1 or r[1] != '*':
        # The first character in the regex is not proceeded by a *.
        if matches_first_char(s, r):
```

```
            return matches(s[1:], r[1:])
        else:
            return False
    else:
        # The first character is proceeded by a *.
        # First, try zero length.
        if matches(s, r[2:]):
            return True
        # If that doesn't match straight away, then try globbing more prefixes
        # until the first character of the string doesn't match anymore.
        i = 0
        while matches_first_char(s[i:], r):
            if matches(s[i+1:], r[2:]):
                return True
            i += 1
```

This takes O(len(s) * len(r)) time and space, since we potentially need to iterate over each suffix substring again for each character.

Fun fact: Stephen Kleene introduced the * operator in regular expressions and as such, it is sometimes referred to as the Kleene star.

© Daily Coding Problem 2019 Privacy Policy Terms of Service Press