Testing The Same Part of a String for More Than One Requirement

Lookaround, which was introduced in detail in the previous topic, is a very powerful concept. Unfortunately, it is often underused by people new to regular expressions, because lookaround is a bit confusing. The confusing part is that the lookaround is zero-length. So if you have a regex in which a lookahead is followed by another piece of regex, or a lookbehind is preceded by another piece of regex, then the regex traverses part of the string twice.

A more practical example makes this clear. Let's say we want to find a word that is six letters long and contains the three consecutive letters `cat`. Actually, we can match this without lookaround. We just specify all the options and lump them together using alternation: `cat\w{3}|\wcat\w{2}|\w{2}cat\w|\w{3}cat`. Easy enough. But this method gets unwieldy if you want to find any word between 6 and 12 letters long containing either "cat", "dog" or "mouse".

Lookaround to The Rescue

In this example, we basically have two requirements for a successful match. First, we want a word that is 6 letters long. Second, the word we found must contain the word "cat".

Matching a 6-letter word is easy with `\b\w{6}\b`. Matching a word containing "cat" is equally easy: `\b\w*cat\w*\b`.

Combining the two, we get: `(?=\b\w{6}\b)\b\w*cat\w*\b`. Easy! Here's how this works. At each character position in the string where the regex is attempted, the engine first attempts the regex inside the positive lookahead. This sub-regex, and therefore the lookahead, matches only when the current character position in the string is at the start of a 6-letter word in the string. If not, the lookahead fails and the engine continues trying the regex from the start at the next character position in the string.

The lookahead is zero-length. So when the regex inside the lookahead has found the 6-letter word, the current position in the string is still at the beginning of the 6-letter word. The regex engine attempts the remainder of the regex at this position. Because we already know that a 6-letter word can be matched at the current position, we know that `\b` matches and that the first `\w*` matches 6 times. The engine then backtracks, reducing the number of characters matched by `\w*`, until `cat` can be matched. If `cat` cannot be matched, the engine has no other choice but to restart at the beginning of the regex, at the next character position in the string. This is at the second letter in the 6-letter word we just found, where the lookahead will fail, causing the engine to advance character by character until the next 6-letter word.

If `cat` can be successfully matched, the second `\w*` consumes the remaining letters, if any, in the 6-letter word. After that, the last `\b` in the regex is guaranteed to match where the second `\b` inside the lookahead matched. Our double-requirement-regex has matched successfully.

Optimizing Our Solution

While the above regex works just fine, it is not the most optimal solution. This is not a problem if you are just doing a search in a text editor. But optimizing things is a good idea if this regex will be used repeatedly and/or on large chunks of data in an application you are developing.

You can discover these optimizations by yourself if you carefully examine the regex and follow how the regex engine applies it, as we did above. The third and last `\b` are guaranteed to match. Since word boundaries are zero-length, and therefore do not change the result returned by the regex engine, we can remove them, leaving: `(?=\b\w{6}\b)\w*cat\w*`. Though the last `\w*` is also guaranteed to match, we cannot remove it because it

adds characters to the regex match. Remember that the lookahead discards its match, so it does not contribute to the match returned by the regex engine. If we omitted the `\w*`, the resulting match would be the start of a 6-letter word containing “cat”, up to and including “cat”, instead of the entire word.

But we can optimize the first `\w*`. As it stands, it will match 6 letters and then backtrack. But we know that in a successful match, there can never be more than 3 letters before “cat”. So we can optimize this to `\w{0,3}`. Note that making the asterisk lazy would not have optimized this sufficiently. The lazy asterisk would find a successful match sooner, but if a 6-letter word does not contain “cat”, it would still cause the regex engine to try matching “cat” at the last two letters, at the last single letter, and even at one character beyond the 6-letter word.

So we have `(?=\b\w{6}\b)\w{0,3}cat\w*`. One last, minor, optimization involves the first `\b`. Since it is zero-length itself, there’s no need to put it inside the lookahead. So the final regex is: `\b(?=\w{6}\b)\w{0,3}cat\w*`.

You could replace the final `\w*` with `\w{0,3}` too. But it wouldn’t make any difference. The lookahead has already checked that we’re at a 6-letter word, and `\w{0,3}cat` has already matched 3 to 6 letters of that word. Whether we end the regex with `\w*` or `\w{0,3}` doesn’t matter, because either way, we’ll be matching all the remaining word characters. Because the resulting match and the speed at which it is found are the same, we may just as well use the version that is easier to type.

A More Complex Problem

So, what would you use to find any word between 6 and 12 letters long containing either “cat”, “dog” or “mouse”? Again we have two requirements, which we can easily combine using a lookahead: `\b(?=\w{6,12}\b)\w{0,9}(cat|dog|mouse)\w*`. Very easy, once you get the hang of it. This regex will also put “cat”, “dog” or “mouse” into the first backreference.

Make a Donation

Page URL: https://www.regular-expressions.info/lookaround2.html
Page last updated: 22 November 2019
Site last updated: 08 April 2020