Theory: Quantifiers

16 minutes 0 / 5 problems solved

There is a type of character called **quantifiers** which defines how often another character can occur in a regex pattern. A quantifier can be written after a regular character as well as after a special one. In general, quantifiers are one of the most essential and important features of the regex language, since they allow a single pattern to match different strings varying in length.

§1. The list of quantifiers

There is a list of quantifiers to be remembered:

- `+` matches one or more repetitions of the preceding character;
- `*` matches zero or more repetitions of the preceding character;
- `{n}` matches exactly `n` repetitions of the preceding character;
- `{n,m}` matches at least `n` but not more than `m` repetitions of the preceding character;
- `{n,}` matches at least `n` repetitions of the preceding character;
- `{0,m}` matches no more than m repetitions of the preceding character.

Note, there is another quantifier `?` that makes the preceding character optional. It is short for `{0,1}`. We will not consider the quantifier here, because you should already know it.

§2. The plus quantifier

Here we demonstrate the **plus** character that matches one or more repetitions of the preceding character:

```
1  String regex = "ca+b";
2
3  "cab".matches(regex); // true
4  "caaaaab".matches(regex); // true
5
"cb".matches(regex); // false, because it does not have at least one repetition of
 'a'
```

As you can see, it matches only those strings which have one or more repetition of the `'a'` character.

§3. The star quantifier

The example below demonstrates the **star** character that matches zero or more repetitions of the preceding character:

```
1  String regex = "A[0-3]*";
2
3
"A".matches(regex);  // true, because the pattern matches zero or more repetitions
4  "A0".matches(regex); // true
5  "A000111222333".matches(regex); // true
```

As you can see, the star quantifier, unlike the plus quantifier, allows the pattern to match also the strings that do not have the "quantified" character at all.

In the following example, there is a pattern describing the string "John" located between the undefined number of undefined characters in the text.

Current topic:

Quantifiers

Topic depends on:

× Sets, ranges, alternations

Topic is required for:

Regexes in programs

Table of contents:

```
String johnRegex = ".*John.*"; // it matches all strings containing the substring "John"

  String textWithJohn = "My friend John is a computer programmer";

  textWithJohn.matches(johnRegex); // true

  String john = "John";

  john.matches(johnRegex); // true



  String textWithoutJohn = "My friend is a computer programmer";



  textWithoutJohn.matches(johnRegex); // false
```

So, the **star** quantifier can be used to check whether a substring of a string matches a pattern. Using it, we can skip spaces or any other characters we don't want to predict in our pattern.

§4. Specifying the number of repetitions

Both previous quantifiers have a wide range of applications, but they do not allow you to specify how many times a character can be repeated. Fortunately, there is a group of quantifiers that allow specifying the number of repetitions in the **curly braces**: `{n}`, `{n,m}`, `{n,}`.

> An important clarification: no spaces are supposed to be used inside curly braces. There can be only one or two numbers and, optionally, a comma. Putting spaces inside curly braces leads to the "deactivation" of the quantifier and, as a result, a totally different regular expression. For example, `"a{1, 2}"` will match only the exact string `"a{1, 2}"`, not `"a"` or `"aa"`.

Take a look at the example where we demonstrate how to match exactly `n` repetitions of the preceding character using the `{n}` quantifier:

```
String regex = "[0-9]{4}"; // four digits

"6342".matches(regex);  // true
"9034".matches(regex);  // true

"182".matches(regex);   // false
"54312".matches(regex); // false
```

Matching from `n` to `m` repetitions is possible thanks to `{n,m}` quantifier. Note that the range specified in the curly braces both starts and ends **inclusively**: `m` encountered repetitions also count as a match. This is standard for the regex language no matter the implementation, although it may sound a bit counter-intuitive for Java's users since Java's ranges end exclusively.

```
String regex = "1{2,3}";

"1".matches(regex);    // false
"11".matches(regex);   // true
"111".matches(regex);  // true
"1111".matches(regex); // false
```

The last example demonstrates how to match at least `n` repetitions using the `{n,}` quantifier.

```
String regex = "ab{4,}";

"abb".matches(regex); // false, not enough 'b'
"abbbb".matches(regex); // true
"abbbbbbb".matches(regex); // true
```

The quantifier that matches not more than `m` repetitions works similarly. Try it yourself.

§5. Conclusions

The key points of this topic are:

- in the regex language, the quantifiers allow us to match strings varying in length.
- the star quantifier matches zero or more repetitions of the preceding character.
- the plus quantifier is almost the same as the star, with the exception that it doesn't match the absence of the character. The minimal number of repetitions for it is one.
- curly braces allow more careful control of the number of repetitions: you can specify the minimal or the maximum number of repetitions, or both.

Report a typo

304 users liked this theory. **2** didn't like it. **What about you?**

Start practicing

Comments (9) Hints (0) Useful links (1) Show discussion