Valid Word Abbreviation

Given a **non-empty** string `s` and an abbreviation `abbr`, return whether the string matches with the given abbreviation.

A string such as `"word"` contains only the following valid abbreviations:

```
["word", "1ord", "w1rd", "wo1d", "wor1", "2rd", "w2d", "wo2", "1o1d", "1or1", "w1r1
", "1o2", "2r1", "3d", "w3", "4"]
```

Notice that only the above abbreviations are valid abbreviations of the string `"word"`. Any other string is not a valid abbreviation of `"word"`.

Note:
Assume `s` contains only lowercase letters and `abbr` contains only lowercase letters and digits.

Example 1:

```
Given s = "internationalization", abbr = "i12iz4n":

Return true.
```

Example 2:

```
Given s = "apple", abbr = "a2e":

Return false.
```

Solution 1

```
    public boolean validWordAbbreviation(String word, String abbr) {
        int i = 0, j = 0;
        while (i < word.length() && j < abbr.length()) {
            if (word.charAt(i) == abbr.charAt(j)) {
                ++i;++j;
                continue;
            }
            if (abbr.charAt(j) <= '0' || abbr.charAt(j) > '9') {
                return false;
            }
            int start = j;
            while (j < abbr.length() && abbr.charAt(j) >= '0' && abbr.charAt(j) <
= '9') {
                ++j;
            }
            int num = Integer.valueOf(abbr.substring(start, j));
            i += num;
        }
        return i == word.length() && j == abbr.length();
    }
```

written by lzmshiwo original link here

Solution 2

Update

Much nicer, I just turn an abbreviation like `"i12iz4n"` into a regular expression like `"i.{12}iz.{4}n"`. Duh.

Java:

```
public boolean validWordAbbreviation(String word, String abbr) {
    return word.matches(abbr.replaceAll("[1-9]\\d*", ".{$0}"));
}
```

Python:

```
def validWordAbbreviation(self, word, abbr):
    return bool(re.match(re.sub('([1-9]\d*)', r'.{\1}', abbr) + '$', word))
```

Obsolete original

(This now gets a memory error, since the exploding testcase I suggested at the end has been added to the test suite.)

"Clean":

```
def validWordAbbreviation(self, word, abbr):
    regex = re.sub('\d+', lambda m: '.' * int(m.group()), abbr)
    return bool(re.match(regex + '$', word)) and not re.search('(^|\D)0', abbr)
```

"Dirty" (abusing how Python handles the `>` there):

```
def validWordAbbreviation(self, word, abbr):
    regex = re.sub('\d+', lambda m: '.' * int(m.group()), abbr)
    return re.match(regex + '$', word) > re.search('(^|\D)0', abbr)
```

I turn each number into that many dots to get a regular expression. For example, when asked whether `"3t2de"` is a valid abbreviation for word `"leetcode"`, I turn `"3t2de"` into `"...t..de"` and check whether that regular expression matches `"leetcode"`, which it does. I also need to rule out the number `"0"` and leading zeros, which I do with another regular expression.

@1337c0d3r I suggest adding test case `"bignumberhahaha", "999999999"`, as that gets me a fully deserved MemoryError :-)

written by StefanPochmann original link here

Solution 3

```
class Solution {
public:
    bool validWordAbbreviation(string word, string abbr) {
        int i = 0, j = 0;
        while (i < word.size() && j < abbr.size()) {
            if (isdigit(abbr[j])) {
                if (abbr[j] == '0') return false;
                int val = 0;
                while (j < abbr.size() && isdigit(abbr[j]))
                    val = val * 10 + abbr[j++] - '0';
                i += val;
            }
            else if (word[i++] != abbr[j++]) return false;
        }
        return i == word.size() && j == abbr.size();
    }
};
```

written by zyoppyoo8 original link here