Word Pattern

Given a `pattern` and a string `str`, find if `str` follows the same pattern.

Here **follow** means a full match, such that there is a bijection between a letter in `pattern` and a **non-empty** word in `str`.

Examples:

1. pattern = `"abba"`, str = `"dog cat cat dog"` should return true.
2. pattern = `"abba"`, str = `"dog cat cat fish"` should return false.
3. pattern = `"aaaa"`, str = `"dog cat cat dog"` should return false.
4. pattern = `"abba"`, str = `"dog dog dog dog"` should return false.

Notes:
You may assume `pattern` contains only lowercase letters, and `str` contains lowercase letters separated by a single space.

Credits:
Special thanks to @minglotus6 for adding this problem and creating all test cases.

Solution 1

```
public boolean wordPattern(String pattern, String str) {
    String[] words = str.split(" ");
    if (words.length != pattern.length())
        return false;
    Map index = new HashMap();
    for (Integer i=0; i<words.length; ++i)
        if (index.put(pattern.charAt(i), i) != index.put(words[i], i))
            return false;
    return true;
}
```

I go through the pattern letters and words in parallel and compare the indexes where they last appeared.

Edit 1: Originally I compared the **first** indexes where they appeared, using `putIfAbsent` instead of `put`. That was based on mathsam's solution for the old Isomorphic Strings problem. But then czonzhu's answer below made me realize that `put` works as well and why.

Edit 2: Switched from

```
    for (int i=0; i<words.length; ++i)
        if (!Objects.equals(index.put(pattern.charAt(i), i),
                            index.put(words[i], i)))
            return false;
```

to the current version with `i` being an `Integer` object, which allows to compare with just `!=` because there's no autoboxing-same-value-to-different-objects-problem anymore. Thanks to lap_218 for somewhat pointing that out in the comments.

written by StefanPochmann original link here

Solution 2

I think all previous C++ solutions read all words into a vector at the start. Here I read them on the fly.

```
bool wordPattern(string pattern, string str) {
    map<char, int> p2i;
    map<string, int> w2i;
    istringstream in(str);
    int i = 0, n = pattern.size();
    for (string word; in >> word; ++i) {
        if (i == n || p2i[pattern[i]] != w2i[word])
            return false;
        p2i[pattern[i]] = w2i[word] = i + 1;
    }
    return i == n;
}
```

written by StefanPochmann original link here

Solution 3

This problem is pretty much equivalent to Isomorphic Strings. Let me reuse two old solutions.

From here:

```
def wordPattern(self, pattern, str):
    s = pattern
    t = str.split()
    return map(s.find, s) == map(t.index, t)
```

Improved version also from there:

```
def wordPattern(self, pattern, str):
    f = lambda s: map({}.setdefault, s, range(len(s)))
    return f(pattern) == f(str.split())
```

From here:

```
def wordPattern(self, pattern, str):
    s = pattern
    t = str.split()
    return len(set(zip(s, t))) == len(set(s)) == len(set(t)) and len(s) == len(t)
```

Thanks to zhang38 for pointing out the need to check len(s) == len(t) here.

written by StefanPochmann original link here