290. Word Pattern

Previous Next

June 29, 2020 | 3.6K views

Average Rating: 4.82 (11 votes)

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`.

Example 1:

```
Input: pattern = "abba", str = "dog cat cat dog"
Output: true
```

Example 2:

```
Input:pattern = "abba", str = "dog cat cat fish"
Output: false
```

Example 3:

```
Input: pattern = "aaaa", str = "dog cat cat dog"
Output: false
```

Example 4:

```
Input: pattern = "abba", str = "dog dog dog dog"
Output: false
```

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

Solution

This problem is similar to Isomorphic Strings.

Approach 1: Two Hash Maps

Intuition

The most naive way to start thinking about this problem is to have a single hash map, tracking which character (in `pattern`) maps to what word (in `str`). As you scan each character-word pair, update this hash map for characters which are not in the mapping. If you see a character which already is one of the keys in mapping, check whether the current word matches with the word the character maps to. If they do not match, you can immediately return `False`, otherwise, just keep on scanning until the end.

This type of check will work well for cases such as:

- "abba" and "dog cat cat dog" -> Returns `True`.
- "abba" and "dog cat cat fish" -> Returns `False`.

But it will fail for:

- "abba" and "dog dog dog dog" -> Returns `True` (Expected `False`).

A fix for this is to have two hash maps, one for mapping characters to words and the other for mapping words to characters. While scanning each character-word pair,

- If the character is **NOT** in the character to word mapping, you additionally check whether that word is also in the word to character mapping.
  - If that word is already in the word to character mapping, then you can return `False` immediately since it has been mapped with some other character before.
  - Else, update both mappings.
- If the character **IS IN** in the character to word mapping, you just need to check whether the current word matches with the word which the character maps to in the character to word mapping. If not, you can return `False` immediately.

Implementation

Java | Python3 — Copy

```
class Solution:
    def wordPattern(self, pattern: str, str: str) -> bool:
        map_char = {}
        map_word = {}

        words = str.split(' ')
        if len(words) != len(pattern):
            return False

        for c, w in zip(pattern, words):
            if c not in map_char:
                if w in map_word:
                    return False
                else:
                    map_char[c] = w
                    map_word[w] = c
            else:
                if map_char[c] != w:
                    return False
        return True
```

Complexity Analysis

- Time complexity : $O(N)$ where N represents the number of words in `str` or the number of characters in `pattern`.
- Space complexity : $O(M)$ where M represents the number of unique words in `str`. Even though we have two hash maps, the character to word hash map has space complexity of $O(1)$ since there can at most be 26 keys.

Addendum: Rather than keeping two hash maps, we can only keep character to word mapping and whenever we find a character that is not in the mapping, you can check whether the word in current character-word pair is already **one of the values** in the character to word mapping. However, this is trading time off for better space since checking for values in a hash map is a $O(M)$ operation where M is the number of key value pairs in the hash map. Thus, if we decide to go this way, our time complexity will be $O(NM)$ where N is the number of unique characters in `pattern`.

Another similar approach to Approach 1 would be using hash set to keep track of words which have been encountered. Instead of checking whether the word is already in the word to character mapping, you just need to check whether the word is in the encountered word hash set. And, rather than updating the word to character mapping, you just need to add the word to the encountered word hash set. Hash set would have a better practical space complexity even though the big-O space complexity for hash set and hash map is the same.

Approach 2: Single Index Hash Map

Intuition

Rather than having two hash maps, we can have a single index hash map which keeps track of the first occurrences of each character in `pattern` and each word in `str`. As we go through each character-word pair, we insert unseen characters from `pattern` and unseen words from `str`.

The goal is to make sure that the indices of each character and word match up. As soon as we find a mismatch, we can return `False`.

Let's go through some examples.

`pattern`: 'abba'

`str`: 'dog cat cat dog'

1. 'a' and 'dog' -> map_index = `{'a': 0, 'dog': 0}`
   - Index of 'a' and index of 'dog' are the same.
2. 'b' and 'cat' -> map_index = `{'a': 0, 'dog': 0, 'b': 1, 'cat': 1}`
   - Index of 'b' and index of 'cat' are the same.
3. 'b' and 'cat' -> map_index = `{'a': 0, 'dog': 0, 'b': 1, 'cat': 1}`
   - 'b' is already in the mapping, no need to update.
   - 'cat' is already in the mapping, no need to update.
   - Index of 'b' and index of 'cat' are the same.
4. 'a' and 'dog' -> map_index = `{'a': 0, 'dog': 0, 'b': 1, 'cat': 1}`
   - 'a' is already in the mapping, no need to update.
   - 'dog' is already in the mapping, no need to update.
   - Index of 'a' and index of 'dog' are the same.

`pattern`: 'abba'

`str`: 'dog cat fish dog'

1. 'a' and 'dog' -> map_index = `{'a': 0, 'dog': 0}`
   - Index of 'a' and index of 'dog' are the same.
2. 'b' and 'cat' -> map_index = `{'a': 0, 'dog': 0, 'b': 1, 'cat': 1}`
   - Index of 'b' and index of 'cat' are the same.
3. 'b' and 'fish' -> map_index = `{'a': 0, 'dog': 0, 'b': 1, 'cat': 1, 'fish': 2}`
   - 'b' is already in the mapping, no need to update.
   - Index of 'b' and index of 'fish' are NOT the same. Returns `False`.

Implementation

Differentiating between character and string: In Python there is no separate `char` type. And for cases such as:

pattern: 'abba' str: 'b a a b'

Using the same hash map will not work properly. A workaround is to prefix each character in `pattern` with "char_" and each word in `str` with "word_".

Java | Python3 — Copy

```
class Solution:
    def wordPattern(self, pattern: str, str: str) -> bool:
        map_index = {}
        words = str.split()

        if len(pattern) != len(words):
            return False

        for i in range(len(words)):
            c = pattern[i]
            w = words[i]

            char_key = 'char_{}'.format(c)
            char_word = 'word_{}'.format(w)

            if char_key not in map_index:
                map_index[char_key] = i

            if char_word not in map_index:
                map_index[char_word] = i

            if map_index[char_key] != map_index[char_word]:
                return False

        return True
```

Complexity Analysis

- Time complexity : $O(N)$ where N represents the number of words in the `str` or the number of characters in the `pattern`.
- Space complexity : $O(M)$ where M is the number of unique characters in `pattern` and words in `str`.

Rate this article:

Previous Next

Comments: 5 — Sort By

Type comment here... (Markdown is supported)

Preview — Post

lokhman ★ 3 June 30, 2020 5:10 PM

```
class Solution(object):
    def wordPattern(self, pattern, s):
        s, pattern = s.split(), list(pattern)
        return map(s.index, s) == map(pattern.index, pattern)
```

Read More

3 Share Reply

SHOW 2 REPLIES

ahmedou ★ 7 July 1, 2020 5:00 PM

Thank you so much, it was very helpful. But I just wonder why the last does not work if we use int instead of integer in the for loop.

1 Share Reply

SHOW 1 REPLY

user2442ZX ★ 0 2 days ago

//my c++ implementation of the same logic of approach 1 .hopefully it will be helpful for beginners//

```
class Solution {
public:
```

Read More

0 Share Reply

SHOW 1 REPLY

bharaniabishshek123 ★ 0 3 days ago — Report

```
    def wordPattern(self, pattern: str, str: str) -> bool:

        list_str = str.split()
```

Read More

0 Share Reply

user9276im ★ 1 July 3, 2020 10:20 PM

```
class Solution:
def wordPattern(self, pattern: str, str: str) -> bool:
s = str.split()
z = list(zip(pattern, s))
if len(z) == len(pattern):
```

Read More

0 Share Reply