<https://leetcode.com/problems/word-pattern/>

Given a pattern and a string s, find if s 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 s.

**Example 1:**

Input: pattern = "abba", s = "dog cat cat dog"

Output: true

**Example 2:**

Input: pattern = "abba", s = "dog cat cat fish"

Output: false

**Example 3:**

Input: pattern = "aaaa", s = "dog cat cat dog"

Output: false

**Constraints:**

1 <= pattern.length <= 300

pattern contains only lower-case English letters.

1 <= s.length <= 3000

s contains only lowercase English letters and spaces ' '.

s **does not contain** any leading or trailing spaces.

All the words in s are separated by a **single space**.

**Attempt 1: 2022-12-24**

**Wrong Solution**

class Solution {

    public boolean wordPattern(String pattern, String s) {

        String[] strs = s.split("\\s+");

        Map<Character, String> map = new HashMap<Character, String>();

        for(int i = 0; i < pattern.length(); i++) {

            char c = pattern.charAt(i);

            // If no current {key, value} mapping, the {key, value} mapping should be the first

            if(!map.containsKey(c)) {

                // The wrong condition is here, if pattern = "abba",

                // str = "dog dog dog dog", when put ('b', "dog") on

                // map, it will surely return null, but "dog" is

                // occupied by another projection ('a', "dog"), the

                // right expression should reflect miss projection

                // such as map.containsValue(a[i])

                if(map.put(c, strs[i]) != null) {

                    return false;

                }

            // If already have current {key, value} mapping, the {key, value} mapping must has the same value

            } else {

                if(!map.put(c, strs[i]).equals(strs[i])) {

                    return false;

                }

            }

        }

        return true;

    }

}

**Solution 1:  HashTable (10 min)**

class Solution {

    public boolean wordPattern(String pattern, String s) {

        String[] strs = s.split("\\s+");

        if(strs.length != pattern.length()) {

            return false;

        }

        Map<Character, String> map = new HashMap<Character, String>();

        for(int i = 0; i < pattern.length(); i++) {

            char c = pattern.charAt(i);

            // If no current {key, value} mapping, the {key, value} mapping should be the first

            if(!map.containsKey(c)) {

                // But if already contains the projection value of this key,

                // which means the value match another key not relate as

                // one on one projection.

                // E.g pattern = "abba", str = "dog dog dog dog",

                // First putting ('a', "dog") on map, key = 'b' not contain

                // in map now, when we try to put ('b', "dog") on map, we

                // find "dog" already on map and is projection of 'a', this

                // violate one(key) on one(value) projection, so return false

                if(map.containsValue(strs[i])) {

                    return false;

                }

                map.put(c, strs[i]);

            // If already have current {key, value} mapping, the {key, value} mapping must has the same value

            } else {

// Don't write condition as 'if(map.get(c).equals(strs[i]))'

// Test out by:

// pattern = "abba", s = "dog cat cat dog"

// Output = false, Expected = true

// Why need map.put(key, value) method ?

// Because map.put(key, value) will return the previous value associated with key,

// which means previous value for this key equals current value, then confirm the

// {key, value} pair is unique

                if(!map.put(c, strs[i]).equals(strs[i])) {

                    return false;

                }

            }

        }

        return true;

    }

}

Time Complexity : O(N)

Space Complexity : O(N)