Solution: Isomorphic Strings

Let's solve the Isomorphic Strings problem using the Hash Map pattern.

We'll cover the following

Statement

Given two strings, check whether two strings are isomorphic to each other or not. Two strings are isomorphic if a fixed mapping exists from the characters of one string to the characters of the other string. For example, if there are two instances of the character "a" in the first string, both these instances should be converted to another character (which could also remain the same character if "a" is mapped to itself) in the second string. This converted character should remain the same in both positions of the second string since there is a fixed mapping from the character "a" in the first string to the converted character in the second string.

> **Note:** Two different characters cannot map to the same character. Furthermore, all the instances of a character must be replaced with another character while protecting the order of characters.

Constraints:

- Both the strings consist of valid ASCII characters.
- The length of the string is $0 \leq$ `length` $\leq 5 \times 10^4$.
- The length of both strings is the same.

Solution

We can use the concept of hash maps to solve the problem more efficiently. The idea is to use a suitable data structure to store the mapping of characters of string1 to characters of string2 and vice versa.

We use the following two hash maps:

- `mapStr1Str2`: This stores the mapping from string1 to string2.
- `mapStr2Str1`: This stores the mapping of the characters from string2 to string 1.

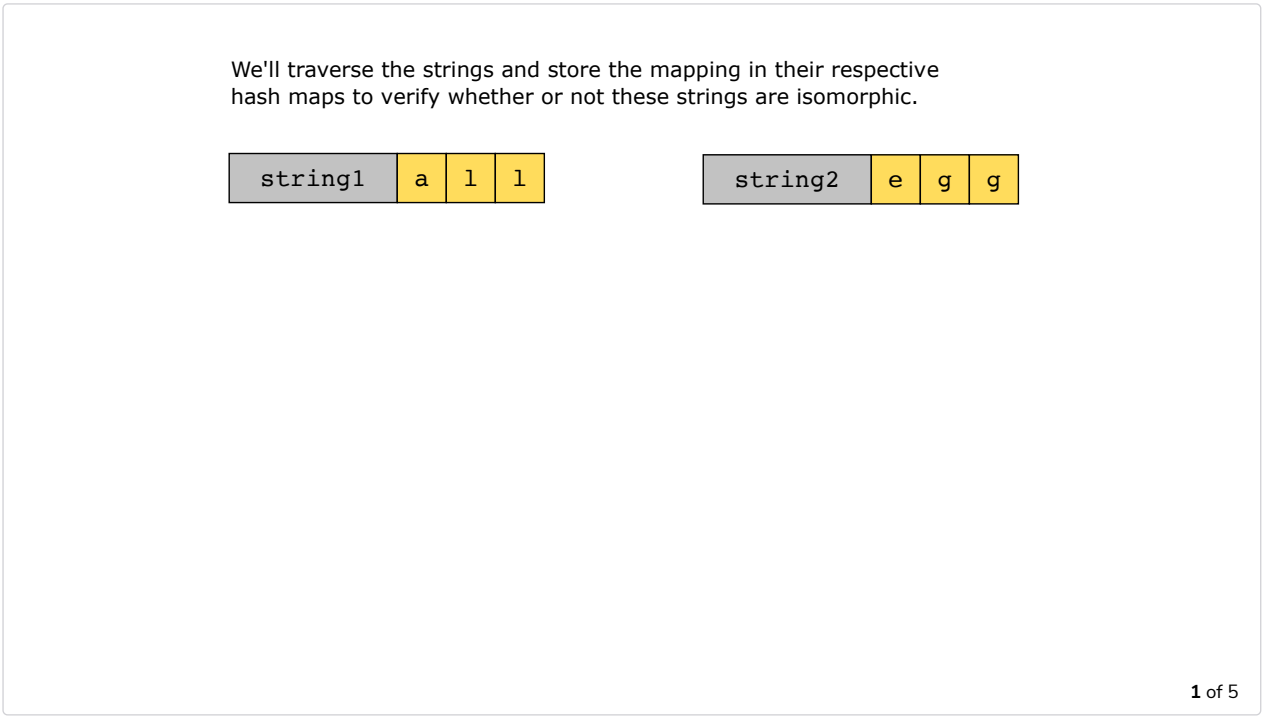
Since the length of both strings is the same, we can traverse the indexes of either one of the two strings in a loop.

Within the loop, we do the following:

> **Note:** `i` represents the i^{th} iteration of the loop which traverses the string.

- We check if `string1[i]` is present as a key in `mapStr1Str2` and that the value corresponding to this key is not `string2[i]`. If both these conditions are satisfied, we return FALSE since a fixed mapping does not exist for `string1[i]` to `string2[i]`. Otherwise, we skip this condition and move forward.
- We check if `string2[i]` is present as a key in `mapStr2Str1` and that the value corresponding to this key is not `string1[i]`. If both these conditions are satisfied, we return FALSE since a fixed mapping does not exist for `string2[i]` to `string1[i]`. Otherwise, we skip this condition and move forward.
- If both the above conditions do not return FALSE, it means the mapping is not defined in either hash map. So we add the key-value pairs of the mapping from the `string1[i]` to `string2[i]` and vice versa in the respective hash maps.
- If the loop ends successfully, it means both the strings are isomorphic since no condition that invalidates the isomorphic properties of both strings was met in the loop.

The slides below illustrate how we'd like the algorithm to run:



We can see the code of this solution below:

Java

```java
class CheckIsomorphic {

    public static boolean isIsomorphic(String string1, String string2) {
        // Initliazing the hashmaps
        Map<Character, Character> mapStr1Str2 = new HashMap <Character, Character> ();
        Map<Character, Character> mapStr2Str1 = new HashMap <Character, Character> ();

        int i = 0, j = 0;

        while (i < string1.length()) {
            char char1 = string1.charAt(i++);
            char char2 = string2.charAt(j++);
            // returning false if char1 in string1 exist in hashmap
            // and the char1 has different mapping in hashmap
            if (mapStr1Str2.containsKey(char1) && mapStr1Str2.get(char1) != char2)

            // returning false if char2 in string2 exist in hashmap
            // and the char2 has different mapping in hashmap
```

```
            if (mapStr2Str1.containsKey(char2) && mapStr2Str1.get(char2) != char1)
                return false;

            // mapping of char of one string to another and vice versa
            mapStr1Str2.put(char1, char2);
            mapStr2Str1.put(char2, char1);

        }
```

Isomorphic Strings

Time complexity

The time complexity for the solution is going to be $O(n)$, where n is the length of the string.

Space complexity

The space complexity for this solution will be$O(1)$ as the set of ASCII characters are fixed and in the dictionary we store valid ASCII characters.

Back
Isomorphic Strings

Next
Longest Palindrome

Mark as Completed