<https://leetcode.com/problems/isomorphic-strings/description/>

Given two strings s and t, determine if they are isomorphic.

Two strings s and t are isomorphic if the characters in s can be replaced to get t.

All occurrences of a character must be replaced with another character while preserving the order of characters. No two characters may map to the same character, but a character may map to itself.

**Example 1:**

**Input:** s = "egg", t = "add"

**Output:** true

**Explanation:**

The strings s and t can be made identical by:

Mapping 'e' to 'a'.

Mapping 'g' to 'd'.

**Example 2:**

**Input:** s = "foo", t = "bar"

**Output:** false

**Explanation:**

The strings s and t can not be made identical as 'o' needs to be mapped to both 'a' and 'r'.

**Example 3:**

**Input:** s = "paper", t = "title"

**Output:** true

**Constraints:**

1 <= s.length <= 5 \* 10^4

t.length == s.length

s and t consist of any valid ascii character.

**Attempt 1: 2023-03-24**

**Solution 1: Hash Table (10 min)**

**Style 1: Hash Table**

class Solution {

    public boolean isIsomorphic(String s, String t) {

        Map<Character, Character> m1 = new HashMap<>();

        Map<Character, Character> m2 = new HashMap<>();

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

            char sChar = s.charAt(i);

            char tChar = t.charAt(i);

            if(!m1.containsKey(sChar) && !m2.containsKey(tChar)) {

                m1.put(sChar, tChar);

                m2.put(tChar, sChar);

            } else {

                if((m1.containsKey(sChar) && m1.get(sChar) != tChar)

                    || (m2.containsKey(tChar) && m2.get(tChar) != sChar)) {

                    return false;

                }

            }

        }

        return true;

    }

}

Time Complexity: O(n)

Space Complexity: O(n)

**Style 2: character array**

**Wrong Solution**

s = "13", t = "42"

java.lang.ArrayIndexOutOfBoundsException: Index -48 out of bounds for length 128

at line 10, Solution.isIsomorphic

at line 56, \_\_DriverSolution\_\_.\_\_helper\_\_

at line 89, \_\_Driver\_\_.main

class Solution {

    public boolean isIsomorphic(String s, String t) {

        char[] sToT = new char[128];

        char[] tToS = new char[128];

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

            char sChar = s.charAt(i);

            char tChar = t.charAt(i);

            int sIdx = sChar - 'a';

            int tIdx = tChar - 'a';

            if(sToT[sIdx] == 0 && tToS[tIdx] == 0) {

                sToT[sIdx] = tChar;

                tToS[tIdx] = sChar;

            } else {

                if(sToT[sIdx] != tChar || tToS[tIdx] != sChar) {

                    return false;

                }

            }

        }

        return true;

    }

}

**Refer to Deepseek**

To fix the ArrayIndexOutOfBoundsException in your solution, you need to correctly handle all ASCII characters by using their direct values as indices instead of adjusting with 'a'. Here's the corrected approach:

**Approach**

Direct Indexing: Use the ASCII value of each character directly as the index for the mapping arrays. This avoids negative indices caused by subtracting 'a' from non-letter characters.

Array Size: Increase the array size to 256 to cover all possible ASCII characters (0-255).

**Mapping Check: Track whether a character has been mapped using separate boolean arrays to handle all characters, including the null character ('\u0000').**

**Solution Code**

class Solution {

public boolean isIsomorphic(String s, String t) {

char[] sToT = new char[256];

boolean[] sMapped = new boolean[256];

char[] tToS = new char[256];

boolean[] tMapped = new boolean[256];

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

char sChar = s.charAt(i);

char tChar = t.charAt(i);

if (!sMapped[sChar] && !tMapped[tChar]) {

sToT[sChar] = tChar;

sMapped[sChar] = true;

tToS[tChar] = sChar;

tMapped[tChar] = true;

} else {

if (sToT[sChar] != tChar || tToS[tChar] != sChar) {

return false;

}

}

}

return true;

}

}

Time Complexity: O(n)

Space Complexity: O(n)

**Explanation**

Direct Indexing: Each character's ASCII value is used directly to index into the arrays, ensuring valid indices and avoiding negative values.

Boolean Tracking: Separate boolean arrays (sMapped and tMapped) track whether a character has been mapped, allowing correct handling of all characters, including '\u0000'.

Efficiency: Using arrays provides O(1) access time, ensuring the solution remains efficient even for large input sizes. The time complexity is O(n) where n is the length of the strings, and space complexity is O(1) due to the fixed-size arrays.

This approach correctly handles all ASCII characters and resolves the index error by using appropriate indices and tracking mappings accurately.

**Refer to**

[L290.Word Pattern](note://F76ECAEBA42D4F90B5BC0D78CA816BA7)

[L890.Find and Replace Pattern (Ref.L205,L290)](note://WEBfbeaed37e3a76872fd1d3ab511ca5882)