Function Documentation: isIsomorphic(s: str, t: str) -> bool

Problem Overview

- The function isIsomorphic determines whether two strings s and t are isomorphic. Two strings are said to be isomorphic if there is a one-to-one mapping between every character in s and every character in t such that the characters in s can be replaced to form t, while maintaining the original character order. No two different characters in s should map to the same character in t, and each character should have a unique counterpart.

Key Definitions

- **Isomorphic Strings:** Two strings s and t are isomorphic if there exists a one-to-one mapping between the characters of s and the characters of t.
  - All occurrences of a character in s must map to the same character in t, and vice versa.
  - Characters must map uniquely, meaning no two characters from s can map to the same character in t, but a character can map to itself.

Input Parameters

- **s (string):** The first string to compare.
  - ***Constraints:*** $1 <= len(s) <= 5 * 10^4$
  - Consists of any valid ASCII characters.

- **t (string):** The second string to compare.
  - ***Constraints:*** len(t) == len(s)
  - Consists of any valid ASCII characters.

Output

- **Returns:** A boolean value.
  - ***True:*** If the two strings are isomorphic.
  - ***False:*** If the two strings are not isomorphic.

Methodology

The function determines whether s and t are isomorphic by ensuring that:

1. **Mapping from s to t:**
   - Each character in s consistently maps to one unique character in t.

2. **Reverse Mapping from t to s:**
   - Each character in t consistently maps to one unique character in s.

To achieve this, the function uses two hash maps (or dictionaries):

- ***s_to_t:*** Maps each character in s to its corresponding character in t.
- ***t_to_s:*** Maps each character in t to its corresponding character in s (reverse mapping).

Algorithm Steps

1. **Initialization:**
   - Two empty dictionaries are created: s_to_t (to map characters from s to t) and t_to_s (to map characters from t to s).

2. **Iteration through characters:**
   - Loop through each pair of characters from s and t at corresponding positions.

3. **Check Existing Mapping:**

- For each character in s, check if it has already been mapped to a character in t.
    - ➢ If the mapping exists, ensure it matches the current character in t. If not, return False.
    - ➢ If the mapping does not exist, create a new mapping from the character in s to the character in t.

4. **Check Reverse Mapping:**

- Simultaneously, for each character in t, check if it has already been mapped to a character in s.
    - ➢ If the reverse mapping exists, ensure it matches the current character in s. If not, return False.
    - ➢ If the reverse mapping does not exist, create a new reverse mapping from the character in t to the character in s.

5. **Completion:**

- If the entire string is processed without any mapping conflicts, return True.

Edge Cases

1. **Different lengths:** Since the function assumes s and t are of the same length (as per the problem constraints), no need to handle differing lengths explicitly.

2. **Self-mapping:** A character can map to itself (e.g., 'a' to 'a').

3. **Multiple mappings:** A character in s cannot map to more than one character in t and vice versa. The function handles this by checking existing mappings for each character.

4. **Empty strings:** Both s and t can be empty (i.e., s = "" and t = ""), in which case they are trivially isomorphic, and the function returns True.

Time Complexity

- **O(n):** The function processes each character of both strings once, where n is the length of the strings. Since both strings have the same length, the time complexity is linear in terms of the length of the input.

Space Complexity

- **O(n):** The function uses two dictionaries, each storing up to n mappings (where n is the length of the strings). Therefore, the space complexity is also linear.

Example 1:

- **Input:** s = "egg", t = "add"
- **Output:** True
- **Explanation:**
  - 'e' maps to 'a', and 'g' maps to 'd'.
  - Both mappings are consistent across the strings.

Example 2:

- **Input:** s = "foo", t = "bar"
- **Output:** False
- **Explanation:**
  - 'o' would need to map to both 'a' and 'r', which violates the one-to-one mapping condition.

Example 3:

- **Input:** s = "paper", t = "title"
- **Output:** True
- **Explanation:**
  - 'p' maps to 't', 'a' maps to 'i', 'e' maps to 'l', and 'r' maps to 'e'. All mappings are consistent.

Use Cases

- **Pattern Matching:** Checking if two patterns in different domains (e.g., strings, symbols, etc.) are equivalent by character substitution.

- **Cryptography:** Can be used to verify simple cryptographic substitutions where each character is uniquely replaced by another.