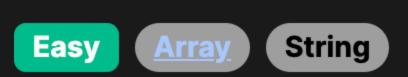
1662. Check If Two String Arrays are Equivalent



Problem Description

This problem presents two arrays, word1 and word2, each containing strings. The task is to determine if these two arrays represent the same string when their contents are concatenated together. In other words, if we join all the elements of word1 end-to-end to make a single string and do the same with word2, and those two resulting strings are identical, we should return true. Otherwise, we will return false. It is essential to concatenate the elements in the order they appear in their respective arrays.

Intuition

To solve this problem, we rely on the simple property that strings are equal if they contain the same sequence of characters in the same order. Therefore, the solution approach is straightforward:

- Concatenate all elements in word1 to form a single string.
 Concatenate all elements in word2 to form another single string.
- Concatenate all elements in word2 to form another single string
- Compare these two strings for equality.

Solution Approach

The implementation of the solution is straightforward and elegant, thanks to Python's high-level string handling capabilities. The algorithm does not rely on complex data structures or patterns; it primarily uses the built-in string functionality provided by Python.

1. The join method is called on an empty string ('') with word1 as the argument. The join method takes all elements in word1,

Here's a step-by-step walk-through of the arrayStringsAreEqual function within the Solution class:

which are strings themselves, and concatenates them in the order they appear in the array. This results in a single string that represents the concatenation of all individual strings in word1.

joined_word1 = ''.join(word1)

```
The same process is applied to word2:
```

```
joined_word2 = ''.join(word2)
```

strings are identical.

result = joined_word1 == joined_word2

Now, we have two strings represented by joined_word1 and joined_word2. All that remains is to check whether these two

```
The result of this comparison is a boolean (True or False). The function directly returns this result, completing the check with a single line of code:
```

return joined_word1 == joined_word2

This approach takes full advantage of Python's ability to handle strings and perform operations on lists. It effectively reduces

what could be a more complex algorithm involving manual iteration and concatenation to a simple one-liner that is easy to understand and maintain.

Let's consider an example where word1 = ["ab", "c"] and word2 = ["a", "bc"]. We need to follow the described solution

Example Walkthrough

approach to determine if these two arrays represent the same string when concatenated.

1. First, we concatenate the elements of word1 using the join method on an empty string.

```
After applying the join method, we end up with the string "abc".
```

joined_word1 = ''.join(["ab", "c"]) # This evaluates to "abc"

joined_word2 = ''.join(["a", "bc"]) # This evaluates to "abc"

```
Similarly, the join method results in the string "abc" for word2.
```

Next, we concatenate the elements of word2.

they are identical:
result = joined_word1 == joined_word2 # This evaluates to True

Now we have both strings obtained from word1 and word2 respectively. We will now compare these two strings to check if

```
In this case, joined_word1 is "abc" and joined_word2 is also "abc". The comparison yields True.
```

from typing import List # Import the List type from typing module for type annotations

Checks if the strings from two lists are equal when concatenated.

* @return true if the concatenated strings are equal, false otherwise.

// Join the elements of the first array into a single string

// Convert the second vector of strings into a single string

return concatenatedWord1 == concatenatedWord2;

const firstCombinedString = firstWordArray.join('');

// Compare the two concatenated strings and return whether they are equal

// This function checks if two arrays of strings are equivalent after combining their elements.

@param {string[]} firstWordArray - The first array of strings to be compared.

String concatenatedWord1 = String.join("", wordArray1);

public boolean arrayStringsAreEqual(String[] wordArray1, String[] wordArray2) {

return result # This returns True

Since result was True, the function would return True, indicating that word1 and word2 do indeed represent the same string

as the concatenated strings from both arrays are identical.

* @param wordArray1 The first array of strings.

* @param wordArray2 The second array of strings.

Finally, we will directly return the result of the comparison:

```
Therefore, given the inputs word1 = ["ab", "c"] and word2 = ["a", "bc"], the function arrayStringsAreEqual will return True,
```

Solution Implementation

Python

when concatenated.

```
class Solution:
   def array_strings_are_equal(self, word1: List[str], word2: List[str]) -> bool:
```

*/

```
Parameters:
    word1 (List[str]): First list of strings.
    word2 (List[str]): Second list of strings.

Returns:
    bool: True if the concatenated strings are equal, False otherwise.

"""

# Concatenate all strings in the first list and compare with the concatenation of the second list return ''.join(word1) == ''.join(word2)

Java

class Solution {

    /**

    * Checks if two string arrays are equal when their elements are concatenated.
```

```
// Join the elements of the second array into a single string
String concatenatedWord2 = String.join("", wordArray2);

// Compare the two concatenated strings for equality
return concatenatedWord1.equals(concatenatedWord2);
}

C++

#include <vector>
#include <string>
#include <string>
#include <numeric> // Required for std::accumulate

class Solution {
public:
    bool arrayStringsAreEqual(vector<string>& word1, vector<string>& word2) {
        // Convert the first vector of strings into a single string
        std::string concatenatedWord1 = std::accumulate(word1.begin(), word1.end(), std::string(""));
```

std::string concatenatedWord2 = std::accumulate(word2.begin(), word2.end(), std::string(""));

// @param {string[]} secondWordArray - The second array of strings to be compared. // @return {boolean} - Returns true if the concatenated strings are equal, otherwise false. function arrayStringsAreEqual(firstWordArray: string[], secondWordArray: string[]): boolean { // Concatenate the elements of the first array into a single string

TypeScript

Time and Space Complexity

The time complexity of the code is 0(m + n) where m is the total number of characters in word1 and n is the total number of characters in word2. This is because the join operations for word1 and word2 each iterate over their respective arrays to build a single string, which takes linear time relative to the number of characters in each array.

The space complexity of the code is 0(m + n) as well, since it needs to allocate space for the new strings generated by "".join(word1) and "".join(word2). No additional space is required beyond the strings themselves.