2942. Find Words Containing Character

Easy Array String

Problem Description

You are given an array of strings, where each string is indexed starting from 0 - this is what is meant by a "0-indexed" array. Your task is to find and return the indices of all strings in this array that contain a specific character `x`. The character `x` is provided to you, and you need to search through each string in the array to check if `x` is included in that string. If it is, you keep track of that string's index. Once you're done checking all the strings, you produce an array of these indices. It's important to note that the order of these indices in the final array does not matter.

Intuition

The intuition behind the solution is pretty straightforward. Since we only need to know whether a character exists in the words, we can simply iterate through each word in the list and check for the presence of the character `x`. This can be done easily with the help of Python's `in` keyword, which checks for the presence of a substring in a string.

For efficiency, we use a list comprehension which allows us to iterate through the list of words and at the same time, keep track of the index of each word. We leverage `enumerate()` here, which returns both the index and the corresponding word. The condition `if x in w` inside the list comprehension checks if the character `x` is in the current word `w`. If it is, the index `i` gets added to the output list. This process continues for each word in the list `words`.

Solution Approach

For the given problem, the solution uses a simple yet effective approach that does not require complex algorithms or data structures. The approach follows a sequential procedure using basic operations provided by Python:

1. **Iteration with Enumeration**:
   - Python's built-in `enumerate()` function is used, which is a common pattern when you need to traverse a list and also keep track of indices.
   - `enumerate(words)` gives us a neat way to loop over `words` while having access to both the index `i` and word `w` at the same time.
2. **Membership Testing**:
   - Inside the loop, we use the membership operator `in` which checks whether the character `x` is present in the word `w`. This operation is performed for every word in the list.
   - The condition `x in w` evaluates to either `True` if `x` is found inside the string `w`, or `False` otherwise.
3. **List Comprehension**:
   - A list comprehension is utilized to build the resulting list of indices in a clean and efficient one-liner.
   - This eliminates the need for initializing an empty list beforehand and appending indices one by one inside a for-loop.
4. **Conditional Expression in List Comprehension**:
   - The list comprehension has a form of `[expression for item in iterable if condition]`, where the `expression` is simply the index `i`.
   - The condition in our case is `if x in w`, and it ensures that the index `i` is included in the final list only if the word `w` contains the character `x`.

The function `findWordsContaining` returns the list that is generated by the list comprehension, which includes all indices `i` where the condition `x in w` holds true.

In summary, the solution achieves the goal by using iteration to go through each word, using the membership test to find words containing the character `x`, and a list comprehension to build the list of indices efficiently. This is a direct and practical approach that leverages Python's language features to solve the problem succinctly.

Example Walkthrough

Let's walk through an example to illustrate the solution approach described above.

Imagine we are given an array of strings `words = ["apple", "banana", "cherry", "date"]` and the character `x = 'a'`. We want to find all the indices of strings containing the character `'a'`.

Following the solution approach:

1. **Iteration with Enumeration**: We start by enumerating the list `["apple", "banana", "cherry", "date"]`. This means we get pairs of indices and words like `(0, "apple")`, `(1, "banana")`, `(2, "cherry")`, and `(3, "date")`.
2. **Membership Testing**:
   - For `(0, "apple")`, we check if `'a'` is in `"apple"`. It is, so this index (0) could be part of our resulting list.
   - Then for `(1, "banana")`, `'a'` is also found in `"banana"`. Index (1) should also be part of our resulting list.
   - For `(2, "cherry")`, `'a'` is not found, so we skip this index.
   - Finally, for `(3, "date")`, `'a'` is again not found, so we also skip this index.
3. **List Comprehension**: Using list comprehension, we combine the steps of enumeration and membership testing into one line. It would look like this:

   ```
   [i for i, w in enumerate(words) if x in w]
   ```

   Executing this line with our example list would produce `[0, 1]` since those are the indices of the words containing the letter `'a'`.
4. **Conditional Expression in List Comprehension**: The conditional (`if x in w`) ensures that only indices of words containing `'a'` are included in the final list.

Putting it all together, the function `findWordsContaining(['apple', 'banana', 'cherry', 'date'], 'a')` returns the list `[0, 1]`, successfully identifying the indices of words that contain the character `'a'`.

Solution Implementation

Python

```python
class Solution:
    def find_words_containing(self, words: List[str], char_sequence: str) -> List[int]:
        """
        Find and return the indices of words that contain the specified character sequence.

        Parameters:
        words: List of strings to search within.
        char_sequence: String sequence to find in the words list.

        Returns:
        List of indices of words that contain the char_sequence.
        """

        # Use list comprehension to find indices of words containing the specified character sequence
        indices = [index for index, word in enumerate(words) if char_sequence in word]

        return indices
```

Java

```java
import java.util.List;
import java.util.ArrayList;

class Solution {

    // This method finds and returns the indices of the words containing a specific character.
    public List<Integer> findWordsContaining(String[] words, char targetChar) {
        // List to store indices of words containing the target character.
        List<Integer> indicesWithTargetChar = new ArrayList<>();

        // Iterate over the array of words.
        for (int index = 0; index < words.length; ++index) {
            // Check if the current word contains the target character.
            if (words[index].indexOf(targetChar) != -1) {
                // If it does, add the index of this word to the list.
                indicesWithTargetChar.add(index);
            }
        }
        // Return the list of indices.
        return indicesWithTargetChar;
    }
}
```

C++

```cpp
#include <vector>
#include <string>

class Solution {
public:
    // Function to find and return the indices of words containing a specific character 'x'
    std::vector<int> findWordsContaining(std::vector<std::string>& words, char x) {
        std::vector<int> indices; // Vector to store indices of words containing the character 'x'

        // Iterate through all the words in the vector
        for (int i = 0; i < words.size(); ++i) {
            // Check if the current word contains the character 'x'
            if (words[i].find(x) != std::string::npos) {
                indices.push_back(i); // If 'x' is found, add the index to the results vector
            }
        }

        return indices; // Return the vector of indices
    }
};
```

TypeScript

```typescript
// This function finds the indices of all words containing a specified substring.
// words: Array of strings to be searched.
// query: Substring to search for within the array elements.
// Returns an array of indices where the substring is found.
function findWordsContaining(words: string[], query: string): number[] {
    // Initialize an empty array to store the indices.
    const indicesOfWordsContainingQuery: number[] = [];

    // Loop through each word in the 'words' array.
    for (let index = 0; index < words.length; ++index) {
        // Check if the current word contains the substring 'query'.
        if (words[index].includes(query)) {
            // If so, add the index of this word to our result array.
            indicesOfWordsContainingQuery.push(index);
        }
    }

    // Return the array of indices where the substring is found.
    return indicesOfWordsContainingQuery;
}


class Solution:
    def find_words_containing(self, words: List[str], char_sequence: str) -> List[int]:
        """
        Find and return the indices of words that contain the specified character sequence.

        Parameters:
        words: List of strings to search within.
        char_sequence: String sequence to find in the words list.

        Returns:
        List of indices of words that contain the char_sequence.
        """

        # Use list comprehension to find indices of words containing the specified character sequence
        indices = [index for index, word in enumerate(words) if char_sequence in word]

        return indices
```

Time and Space Complexity

The time complexity of the given code snippet can be understood by considering each operation in the list comprehension. The function iterates through each word in the list `words` and checks if the substring `x` is present in each word `w`. This operation involves checking each character in each word against the characters in `x` until a match is found or the word is exhausted. Thus, if we let `L` be the sum of the lengths of all the strings in the list `words`, the worst-case scenario is when `x` is not present in any word, resulting in a time complexity of `O(L)`.

The space complexity of the function, aside from the output list, is `O(1)`. This is because the only extra space used is for the loop iteration variable and the index `i`, which do not depend on the input size. Therefore, the space used for the function's operations other than the storage for the return list is constant.