1455. Check If a Word Occurs As a Prefix of Any Word in a Sentence

Easy String String Matching

Leetcode Link

Problem Description

The challenge is to find out if a given `searchWord` is a prefix of any word in the `sentence`. The `sentence` is a string made up of words separated by single spaces. A prefix is defined as the initial part of a word. If the `searchWord` is found to be the prefix of one or more words in the sentence, we need to return the position of the first word (with the smallest index) where the `searchWord` is a prefix. Otherwise, we return `-1`. The words in the sentence are 1-indexed, meaning the first word is considered to be at position 1, the second word at position 2, and so on.

Intuition

The intuitive approach to solve this problem involves checking each word in the given `sentence` to see if it starts with the `searchWord`. We split the `sentence` into individual words and iterate over them. With each iteration, we check whether the current word begins with the `searchWord` using the `startswith()` method. If we find a match, we return the current index, which corresponds to the position of the word in the sentence. If we finish iterating over all the words without finding a match, we return `-1` as per the problem's requirement. The `enumerate()` function coupled with a starting index of 1 is used to keep track of the word positions in a 1-indexed fashion.

Solution Approach

The solution uses a simple linear scan algorithm to solve the problem, which is efficient considering the problem's constraints. Since the sentence is guaranteed to have words separated by a single space, we can directly use the `split()` function in Python, which, by default, splits the string by any whitespace, including spaces. This provides us with a list of words contained in the sentence.

With the list obtained, we proceed with the `enumerate()` function, which iterates over the list and provides both the index and value of each item. However, to maintain the 1-indexed requirement, we start the enumeration from 1 by passing `1` as the second argument to `enumerate()`.

During each iteration, we check for the prefix condition using the `startswith()` method, which is a string method in Python that returns `True` if the string starts with the specified prefix, `False` otherwise.

The loop iterates through all words in the sentence. If a word is found that starts with `searchWord`, the loop breaks, and the current index is returned. This index corresponds to the word's 1-indexed position in the original sentence. If no such word is found and the loop finishes, the function returns `-1` to indicate the absence of a valid prefix match.

Here's the specific implementation from the provided code:

```python
class Solution:
    def isPrefixOfWord(self, sentence: str, searchWord: str) -> int:
        # Split sentence into words and enumerate them starting from 1
        for i, word in enumerate(sentence.split(), 1):
            # Check if current word starts with searchWord
            if word.startswith(searchWord):
                return i  # Return the 1-index position of the word
        return -1  # Return -1 if no prefix match is found in any words
```

No additional data structures are used, and the `startswith()` method provides a clean and readable way to check the prefix condition, making this solution straightforward and easy to understand.

Example Walkthrough

Let's take the `sentence` "hello world hi hey" and the `searchWord` "he". We want to find out if "he" is a prefix of any word in the `sentence` and return the position of the first word where "he" is a prefix.

The solution approach is as follows:

1. The `sentence` is split into individual words, giving us a list `["hello", "world", "hi", "hey"]`.
2. We use `enumerate()` to iterate over this list, starting indexing at 1 to match the problem's 1-indexed word position requirement.
3. For each word, we check if the `searchWord` "he" is a prefix of the word using the `startswith()` method.
4. We start with the first word "hello":
   - "hello".startswith("he") is `True`.
   - Since this condition is true, and "hello" is the first word, we return its 1-index position, which is `1`.

Therefore, in this example, the `searchWord` "he" is a prefix of the word "hello", and the position returned is `1`.

If the `searchWord` was "hi" instead, the steps would be followed until we reached the word "hi":

1. "hello".startswith("hi") is `False`.
2. "world".startswith("hi") is `False`.
3. "hi".startswith("hi") is `True`, so we would return the 1-index position, which is `3` in this case.

And if our `searchWord` was something like "xyz", which isn't a prefix for any of the words:

1. "hello".startswith("xyz") is `False`.
2. "world".startswith("xyz") is `False`.
3. "hi".startswith("xyz") is `False`.
4. "hey".startswith("xyz") is `False`.
   - Since no words start with "xyz", we reach the end of the iteration and return `-1`.

Python Solution

```python
class Solution:
    def isPrefixOfWord(self, sentence: str, search_word: str) -> int:
        # Split the sentence into words
        words = sentence.split()

        # Enumerate over the words starting with an index of 1
        for index, word in enumerate(words, start=1):
            # Check if the current word starts with the search_word
            if word.startswith(search_word):
                # If search_word is a prefix, return the position of the word.
                return index

        # If no word starts with search_word, return -1
        return -1

# Example Usage:
# sol = Solution()
# result = sol.isPrefixOfWord("hello world", "wor")
# print(result)  # Outputs: 2 since "world" is the second word and has "wor" as a prefix

```

Java Solution

```java
class Solution {

    // Method that finds if the searchWord is a prefix of any word in the sentence.
    // If it is, returns the position (1-indexed) of the first occurrence. If not, returns -1.
    public int isPrefixOfWord(String sentence, String searchWord) {
        // Split the sentence into an array of individual words.
        String[] words = sentence.split(" ");

        // Iterate through each word in the array.
        for (int i = 0; i < words.length; i++) {
            // Check if the current word starts with the searchWord.
            if (words[i].startsWith(searchWord)) {
                // If it does, return the position of the word in the sentence, noting that index is 1-based.
                return i + 1;
            }
        }
        // If no word in the sentence is prefixed by searchWord, return -1.
        return -1;
    }
}

```

C++ Solution

```cpp
class Solution {
public:
    // Function to find if the searchWord is a prefix of any word in the sentence.
    // Returns the word's index if found, otherwise returns -1.
    int isPrefixOfWord(string sentence, string searchWord) {
        // Initialize a stringstream with the given sentence
        stringstream ss(sentence);

        // Variable to store each word while extracting from the sentence
        string currentWord;

        // Variable to keep track of the word's index
        int wordIndex = 1;

        // Extract words one by one from the stringstream
        while (ss >> currentWord) {
            // Check if the current word starts with the searchWord
            if (currentWord.find(searchWord) == 0) {
                // If searchWord is a prefix of currentWord,
                // return the current word's index
                return wordIndex;
            }
            // Move to the next word
            ++wordIndex;
        }
        // If the searchWord is not a prefix of any word, return -1
        return -1;
    }
};

```

Typescript Solution

```typescript
/**
 * Checks if the searchWord is a prefix of any word in a given sentence.
 * If it is, returns the 1-based index of the first word where it's a prefix.
 * Otherwise, returns -1.
 *
 * @param sentence - The sentence to search within.
 * @param searchWord - The word to check as a prefix.
 * @returns The 1-based index of the first word with the prefix or -1.
 */
function isPrefixOfWord(sentence: string, searchWord: string): number {
    // Split the sentence into an array of words using space as a separator
    const words = sentence.split(' ');

    // Get the number of words in the array
    const wordCount = words.length;

    // Loop through the array of words
    for (let index = 0; index < wordCount; index++) {
        // Check if the current word starts with the searchWord
        if (words[index].startsWith(searchWord)) {
            // If it does, return the current index plus one (1-based index)
            return index + 1;
        }
    }

    // If no word starts with the searchWord, return -1
    return -1;
}

```

Time and Space Complexity

Time Complexity

The time complexity of the given code is `O(n * k)` where `n` is the number of words in the sentence and `k` is the length of `searchWord`. This is because:

- The `split()` method is called on the sentence, which takes `O(m)` time, where `m` is the length of the sentence.
- Every word is compared with `searchWord` using `startswith()`, which in the worst case checks up to `k` characters for each of the `n` words.

Space Complexity

The space complexity of the given code is `O(n)` where `n` is the number of words in the sentence. This is due to:

- The `split()` method, which creates a list of words from the sentence, storing each word as a separate element in the list. In the worst case, the list will contain `n` elements, thus taking `O(n)` space.