11.01.2025

Problem Name: word subsets

Problem Statement:
You are given two string arrays `words1` and `words2`.
A string `b` is a subset of string `a` if every letter in `b` occurs in `a` including multiplicity.
A string `a` from `words1` is **universal** if for every string `b` in `words2`, `b` is a subset of `a`.

Return an array of all the **universal strings** in `words1`.

Example 1:

```
Input: words1 = ["amazon","apple","facebook","google","leetcode"], words2 =
["e","o"]
Output: ["facebook","google","leetcode"]
```

Example 2:

```
Input: words1 = ["amazon","apple","facebook","google","leetcode"], words2 =
["lc","eo"]
Output: ["leetcode"]
```

Example 3:

```
Input: words1 = ["acaac","cccbb","aacbb","caacc","bcbbb"], words2 =
["c","cc","b"]
Output: ["cccbb"]
```

Approach: Using Frequency Count (Optimized)

Key Idea:

- To determine if a word is **universal**, it must satisfy the condition of being a **superstring** for all strings in `words2`.
- Convert the problem into **frequency counting**.

Steps:

1. **Calculate the Maximum Requirement for words2:**
   a. Compute the **maximum frequency** of each character across all words in `words2`.
2. **Check each word in words1:**
   a. Count the frequency of characters for each word in `words1`.
   b. Compare with the **maximum required frequency** from `words2`.

C++ Code:

```cpp
class Solution {
public:
    // Helper function to check if word1 can cover the requirements
    bool isSubSet(vector<int>& charRequired, vector<int>& temp) {
        for (int i = 0; i < 26; i++) {
            if (temp[i] < charRequired[i])
                return false;
        }
        return true;
    }

    vector<string> wordSubsets(vector<string>& words1, vector<string>& words2) {
        vector<string> res;
        vector<int> charRequired(26, 0);  // Store the maximum frequency needed
for each character

        // Step 1: Calculate the maximum frequency requirement across words2
        for (string& word : words2) {
            vector<int> temp(26, 0);
            for (char ch : word) {
                temp[ch - 'a']++;
                charRequired[ch - 'a'] = max(charRequired[ch - 'a'], temp[ch -
'a']);
            }
        }

        // Step 2: Check each word in words1 against the required frequency
        for (string& word : words1) {
            vector<int> temp(26, 0);
            for (char ch : word) {
                temp[ch - 'a']++;
            }
            if (isSubSet(charRequired, temp)) {
                res.push_back(word);
            }
        }
        return res;
    }
};
```

Explanation:

1. **Input:**
   a. `words1 = ["amazon","apple","facebook","google","leetcode"]`
   b. `words2 = ["e", "o"]`
2. **Step 1:** Calculate the maximum frequency requirement from `words2`.
   a. `e -> 1`, `o -> 1` → `charRequired = [0,0,0,...1,0,...1,...0]`
3. **Step 2:** Check each word in `words1`.
   a. `"facebook"` has both `"e"` and `"o"` → valid.
   b. `"amazon"` → doesn't satisfy `"o"` requirement.

Complexity Analysis:

- **Time Complexity:**
  a. `O(n * l + m * k)`
  b. `n` = number of words in `words1`.
  c. `l` = average length of words in `words1`.
  d. `m` = number of words in `words2`.
  e. `k` = average length of words in `words2`.
- **Space Complexity:**
  a. `O(26)` for storing the frequency count (constant space).

✅ Edge Cases Handled:

- If `words1` or `words2` is empty.
- If all words in `words1` are subsets of `words2`.

Test Cases:

```
Input: words1 = ["a","b","c"], words2 = ["a"]
Output: ["a"]

Input: words1 = ["abc","def","ghi"], words2 = ["z"]
Output: []
```

Summary:

- Use a **frequency count** approach for efficiency.

- Count the **maximum required frequency** for each character in `words2`.
- Validate each word in `words1` using the **subset** condition.

Would you like me to provide more test cases or further optimize this? 😊