**[916. Word Subsets](https://leetcode.com/problems/word-subsets/)**

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.

* For example, "wrr" is a subset of "warrior" but is not a subset of "world".

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. You may return the answer in **any order**.

**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 = ["l","e"]

**Output:** ["apple","google","leetcode"]

**Constraints:**

* 1 <= words1.length, words2.length <= 104
* 1 <= words1[i].length, words2[i].length <= 10
* words1[i] and words2[i] consist only of lowercase English letters.
* All the strings of words1 are **unique**.

To solve this problem, we need to determine the "universal strings" in words1. A string is universal if it contains all the characters (including multiplicities) from every string in words2.

Here's how we can approach the problem:

**Steps to Solve:**

1. **Count Character Frequencies**:
   * Use a helper function to count the frequency of each character in a string.
   * For words2, compute the **maximum frequency** for each character across all strings in words2. This is because a universal string must satisfy the highest frequency requirement for each character from all strings in words2.
2. **Check Each String in**words1:
   * For each string in words1, check if it contains at least the maximum frequency of each character required (from the above step).
3. **Output Universal Strings**:
   * Collect and return the strings in words1 that satisfy the above condition.

**Implementation:**

Here’s the Python code:

from collections import Counter def wordSubsets(words1, words2): # Step 1: Compute the maximum frequency for each character in words2 max\_freq = Counter() for b in words2: b\_freq = Counter(b) for char in b\_freq: max\_freq[char] = max(max\_freq[char], b\_freq[char]) # Step 2: Check each word in words1 result = [] for a in words1: a\_freq = Counter(a) if all(a\_freq[char] >= max\_freq[char] for char in max\_freq): result.append(a) return result

**Explanation:**

1. Counter**to Count Frequencies**:
   * Counter is used to store the frequency of characters for each string. For example:

python

Copy code

Counter("hello") # Output: {'h': 1, 'e': 1, 'l': 2, 'o': 1}

1. **Aggregate Maximum Frequencies**:
   * For words2, compute the maximum frequency for each character:

python

Copy code

words2 = ["l", "e"] max\_freq = Counter() # Initially empty for b in words2: b\_freq = Counter(b) for char in b\_freq: max\_freq[char] = max(max\_freq[char], b\_freq[char]) # Result: max\_freq = {'l': 1, 'e': 1}

1. **Check Universal Condition**:
   * For each string in words1, ensure it satisfies the character frequency requirements in max\_freq. If so, add it to the result.

**Complexity:**

1. **Time Complexity**:
   * **Counting Frequencies**: O(n2×m2)*O*(*n*2​×*m*2​), where n2*n*2​ is the length of words2 and m2*m*2​ is the average length of strings in words2.
   * **Checking Words**: O(n1×m1)*O*(*n*1​×*m*1​), where n1*n*1​ is the length of words1 and m1*m*1​ is the average length of strings in words1.
   * Overall: O(n1×m1+n2×m2)*O*(*n*1​×*m*1​+*n*2​×*m*2​).
2. **Space Complexity**:
   * Storing character frequencies in max\_freq: O(26)*O*(26) (constant space for English letters).

**Example Walkthrough:**

**Input:**

python

Copy code

words1 = ["amazon", "apple", "facebook", "google", "leetcode"] words2 = ["e", "o"]

**Execution:**

1. **Build**max\_freq**for**words2:

python

Copy code

max\_freq = {'e': 1, 'o': 1}

1. **Check**words1:
   * "amazon": Doesn't have 'e'.
   * "apple": Doesn't have 'o'.
   * "facebook": Has 'e' and 'o' (both at least 1 occurrence).
   * "google": Has 'e' and 'o'.
   * "leetcode": Has 'e' and 'o'.
2. **Result**:

python

Copy code

["facebook", "google", "leetcode"]

This solution is efficient and adheres to the constraints. Let me know if you need further clarification!

**class Solution { public List<String> wordSubsets(String[] universalSet, String[] subsetWords) { // This array will keep the max frequency of each letter required by subsetWords int[] maxSubsetFreq = new int[26]; // Calculate the max frequency of each character across all words in subsetWords for (String subsetWord : subsetWords) { // Temporary array to store frequency of each character in the current word int[] tempFreq = new int[26]; for (char ch : subsetWord.toCharArray()) { // Increment character frequency tempFreq[ch - 'a']++; // Update the maxSubsetFreq array with the maximum frequency needed for this character maxSubsetFreq[ch - 'a'] = Math.max(maxSubsetFreq[ch - 'a'], tempFreq[ch - 'a']); } } // This will store our final result List<String> result = new ArrayList<>(); // Loop through each word in universalSet for (String word : universalSet) { // Temporary array to store frequency of each character in the current word int[] wordFreq = new int[26]; for (char ch : word.toCharArray()) { // Increment character frequency wordFreq[ch - 'a']++; } // Check if the current word contains all the required characters in proper frequency boolean isUniversal = true; for (int i = 0; i < 26; ++i) { if (maxSubsetFreq[i] > wordFreq[i]) { // If any character is found in less frequency than required, // mark word as non-universal, and break the loop isUniversal = false; break; } } // If the word is universal, add it to the result list if (isUniversal) { result.add(word); } } // Return the list of all universal words return result; } }**

Here's an explanation of the code you provided for the **Word Subsets** problem. It is implemented in Java and solves the problem efficiently:

**Code Walkthrough:**

**Key Concepts:**

1. maxSubsetFreq**Array**:
   * This array stores the maximum frequency of each character required by all words in subsetWords.
   * The size is 26, representing the 26 lowercase English letters ('a' to 'z').
2. **Frequency Calculation for**subsetWords:
   * For each word in subsetWords, a temporary frequency array tempFreq is created.
   * The maximum frequency of each character across all subsetWords is stored in maxSubsetFreq.
3. **Checking Universality of**universalSet**Words**:
   * For each word in universalSet, the frequency of characters is calculated using wordFreq.
   * The word is checked against maxSubsetFreq to ensure it meets or exceeds the required character frequencies for all characters.

**Detailed Steps:**

1. **Calculate**maxSubsetFreq:

java

Copy code

for (String subsetWord : subsetWords) { int[] tempFreq = new int[26]; for (char ch : subsetWord.toCharArray()) { tempFreq[ch - 'a']++; maxSubsetFreq[ch - 'a'] = Math.max(maxSubsetFreq[ch - 'a'], tempFreq[ch - 'a']); } }

* + For each word in subsetWords, a temporary frequency array tempFreq is used to count character occurrences.
  + The maxSubsetFreq array is updated to ensure it captures the highest frequency of each character required by all subsetWords.

1. **Check Universality of Words in**universalSet:

java

Copy code

for (String word : universalSet) { int[] wordFreq = new int[26]; for (char ch : word.toCharArray()) { wordFreq[ch - 'a']++; } boolean isUniversal = true; for (int i = 0; i < 26; ++i) { if (maxSubsetFreq[i] > wordFreq[i]) { isUniversal = false; break; } } if (isUniversal) { result.add(word); } }

* + For each word in universalSet, the wordFreq array counts the character occurrences.
  + The word is compared against maxSubsetFreq for each character:
    - If any character frequency in wordFreq is less than that in maxSubsetFreq, the word is not universal, and the loop breaks.
  + If all frequencies meet the requirements, the word is added to the result list.

1. **Return the Result**:

java

Copy code

return result;

* + After processing all words in universalSet, the list of universal words is returned.

**Complexity Analysis:**

1. **Time Complexity**:
   * Building maxSubsetFreq: O(n⋅m)*O*(*n*⋅*m*), where n*n* is the length of subsetWords and m*m* is the average length of the words in subsetWords.
   * Checking universality: O(p⋅q)*O*(*p*⋅*q*), where p*p* is the length of universalSet and q*q* is the average length of the words in universalSet.
   * Total: O(n⋅m+p⋅q)*O*(*n*⋅*m*+*p*⋅*q*).
2. **Space Complexity**:
   * maxSubsetFreq and wordFreq arrays: O(26)*O*(26) each (constant space for English letters).
   * Total: O(26)*O*(26) + space for the result list.

**Example Walkthrough:**

**Input:**

java

Copy code

String[] universalSet = {"amazon", "apple", "facebook", "google", "leetcode"}; String[] subsetWords = {"e", "o"};

**Execution:**

1. **Build**maxSubsetFreq:
   * For "e": maxSubsetFreq = [0, 0, 0, ..., 1, ..., 0] (1 occurrence of 'e').
   * For "o": maxSubsetFreq = [0, 0, 0, ..., 1, ..., 1] (1 occurrence each of 'e' and 'o').
2. **Check Universality**:
   * "amazon": Doesn't contain 'e'. **Not universal**.
   * "apple": Doesn't contain 'o'. **Not universal**.
   * "facebook": Contains 'e' and 'o'. **Universal**.
   * "google": Contains 'e' and 'o'. **Universal**.
   * "leetcode": Contains 'e' and 'o'. **Universal**.
3. **Result**:

java

Copy code

["facebook", "google", "leetcode"]

This implementation is efficient and adheres to the constraints.

**more optimized solution**

Here's an optimized solution to the **Word Subsets** problem. The primary focus is on reducing the number of iterations and making the code as efficient as possible.

**Optimized Solution:**

Instead of recalculating character frequencies for each word repeatedly, we precompute the maximum frequency of each character required from all subsetWords and directly compare it against the character frequencies of words in universalSet.

**Java Code:**

java

Copy code

import java.util.ArrayList; import java.util.List; class Solution { public List<String> wordSubsets(String[] universalSet, String[] subsetWords) { // Calculate the maximum frequency required for each character across all subsetWords int[] maxSubsetFreq = new int[26]; for (String subsetWord : subsetWords) { int[] tempFreq = getFrequency(subsetWord); for (int i = 0; i < 26; i++) { maxSubsetFreq[i] = Math.max(maxSubsetFreq[i], tempFreq[i]); } } // List to store universal words List<String> result = new ArrayList<>(); // Check each word in universalSet for (String word : universalSet) { int[] wordFreq = getFrequency(word); // Check if the word meets the required frequency for each character if (isUniversal(wordFreq, maxSubsetFreq)) { result.add(word); } } return result; } // Helper function to calculate the frequency of each character in a string private int[] getFrequency(String word) { int[] freq = new int[26]; for (char ch : word.toCharArray()) { freq[ch - 'a']++; } return freq; } // Helper function to check if a word meets the universal condition private boolean isUniversal(int[] wordFreq, int[] maxSubsetFreq) { for (int i = 0; i < 26; i++) { if (wordFreq[i] < maxSubsetFreq[i]) { return false; } } return true; } }

**Explanation of the Optimized Code:**

**Key Improvements:**

1. **Precompute Frequencies**:
   * Instead of calculating frequencies multiple times, the getFrequency function is used to compute the frequency of characters in a word in O(L)*O*(*L*), where L*L* is the length of the word.
   * The maxSubsetFreq array ensures that we process subsetWords only once.
2. **Direct Comparison**:
   * The isUniversal function efficiently checks if the frequency of each character in a word from universalSet is at least as much as required by maxSubsetFreq.
3. **Compact Helper Functions**:
   * The helper functions getFrequency and isUniversal modularize the code and improve readability while keeping operations efficient.

**Complexity Analysis:**

**Time Complexity:**

1. **Processing**subsetWords:
   * For each word of average length M*M*, calculating frequency takes O(M)*O*(*M*).
   * Let N2*N*2 be the length of subsetWords.
   * Total: O(N2⋅M)*O*(*N*2⋅*M*).
2. **Processing**universalSet:
   * For each word of average length L*L*, calculating frequency takes O(L)*O*(*L*).
   * Let N1*N*1 be the length of universalSet.
   * Total: O(N1⋅L)*O*(*N*1⋅*L*).
3. **Comparison**:
   * Checking if a word is universal takes O(26)=O(1)*O*(26)=*O*(1) for each word.

Overall Time Complexity:

O(N2⋅M+N1⋅L)*O*(*N*2⋅*M*+*N*1⋅*L*)

**Space Complexity:**

* Frequency arrays (maxSubsetFreq, wordFreq, tempFreq) require O(26)=O(1)*O*(26)=*O*(1) space.

**Example Walkthrough:**

**Input:**

java

Copy code

String[] universalSet = {"amazon", "apple", "facebook", "google", "leetcode"}; String[] subsetWords = {"e", "o"};

**Execution:**

1. **Build**maxSubsetFreq:
   * For "e": maxSubsetFreq = [0, 0, ..., 1, ..., 0] (1 occurrence of 'e').
   * For "o": maxSubsetFreq = [0, 0, ..., 1, ..., 1] (1 occurrence each of 'e' and 'o').
2. **Check Universality**:
   * "amazon": Fails for 'e'. **Not universal**.
   * "apple": Fails for 'o'. **Not universal**.
   * "facebook": Passes. **Universal**.
   * "google": Passes. **Universal**.
   * "leetcode": Passes. **Universal**.
3. **Result**:

java

Copy code

["facebook", "google", "leetcode"]

This approach minimizes redundant calculations and ensures an efficient solution!

**from collections import Counter class Solution: def wordSubsets(self, words1: List[str], words2: List[str]) -> List[str]: # Create a counter to store the maximum frequency of each character # across all words in words2 max\_freq\_counter = Counter() for word in words2: word\_freq\_counter = Counter(word) for char, freq in word\_freq\_counter.items(): # Update the counter for each character to the maximum frequency max\_freq\_counter[char] = max(max\_freq\_counter[char], freq) # Initialize a list to keep all words from words1 that meet the criteria universal\_words = [] # Iterate through each word in words1 to check if it is a universal word for word in words1: word\_freq\_counter = Counter(word) # Check if word has at least as many of each character as needed is\_universal = all(freq <= word\_freq\_counter[char] for char, freq in max\_freq\_counter.items()) # If the word meets the criteria, add it to the universal words list if is\_universal: universal\_words.append(word) # Return the list of universal words return universal\_words**

This Python solution efficiently solves the **Word Subsets** problem using the Counter class from the collections module.

Here's an explanation and analysis of the code:

**Code Explanation:**

**1. Maximum Frequency Calculation for**words2**:**

* A Counter named max\_freq\_counter is used to store the **maximum required frequency** of each character across all strings in words2.
* For each word in words2, the frequency of characters is calculated using Counter(word).
* The max function is used to update max\_freq\_counter, ensuring it always holds the highest frequency for each character.

**2. Checking Words in**words1**:**

* For each word in words1, its character frequency is calculated using Counter(word).
* A word is considered **universal** if it contains **at least** as many of each character as specified in max\_freq\_counter.
* The all() function checks this condition for every character in max\_freq\_counter.

**3. Adding Universal Words:**

* Words that meet the universal condition are added to the universal\_words list.

**4. Returning the Result:**

* Finally, the list of universal words is returned.

**Complexity Analysis:**

**Time Complexity:**

1. **Building**max\_freq\_counter**:**
   * Let N2*N*2 be the number of words in words2 and M*M* be the average length of these words.
   * Constructing Counter for each word takes O(M)*O*(*M*), and updating max\_freq\_counter takes O(26)=O(1)*O*(26)=*O*(1) per character.
   * Total: O(N2⋅M)*O*(*N*2⋅*M*).
2. **Checking Universality of Words in**words1**:**
   * Let N1*N*1 be the number of words in words1 and L*L* be the average length of these words.
   * Constructing Counter for each word takes O(L)*O*(*L*).
   * Checking if a word is universal involves O(26)=O(1)*O*(26)=*O*(1) comparisons per word.
   * Total: O(N1⋅L)*O*(*N*1⋅*L*).

**Overall Time Complexity:**

O(N2⋅M+N1⋅L)*O*(*N*2⋅*M*+*N*1⋅*L*)

**Space Complexity:**

* max\_freq\_counter uses O(26)=O(1)*O*(26)=*O*(1) space.
* Temporary Counter objects for each word require O(26)*O*(26) space.
* Total: O(1)*O*(1) additional space (not counting the input and output).

**Example Walkthrough:**

**Input:**

python

Copy code

words1 = ["amazon", "apple", "facebook", "google", "leetcode"] words2 = ["e", "o"]

**Execution:**

1. **Build**max\_freq\_counter**:**
   * For "e": {'e': 1}.
   * For "o": {'e': 1, 'o': 1}.

Final max\_freq\_counter: {'e': 1, 'o': 1}.

1. **Check Universality for**words1**:**
   * "amazon": Fails for 'e'. **Not universal**.
   * "apple": Fails for 'o'. **Not universal**.
   * "facebook": Passes. **Universal**.
   * "google": Passes. **Universal**.
   * "leetcode": Passes. **Universal**.
2. **Result:**

python

Copy code

["facebook", "google", "leetcode"]

**Advantages of This Solution:**

1. **Readability:** The use of Counter simplifies frequency calculations and comparisons.
2. **Efficiency:** The max() operation and all() check are efficient and concise.
3. **Modular Design:** The code is easy to adapt or extend for similar problems.

**more optimized python solution**

To optimize the solution further, we can reduce the number of intermediate calculations by combining the frequency calculations for words2 and using array indexing for character frequencies instead of Counter. This avoids the overhead of dictionary operations and makes the solution more efficient.

Here's the optimized solution:

python

Copy code

class Solution: def wordSubsets(self, words1: List[str], words2: List[str]) -> List[str]: # Initialize an array to store max frequency of each letter in words2 max\_freq = [0] \* 26 # Compute the maximum required frequency for each character across words2 for word in words2: char\_freq = [0] \* 26 for char in word: char\_freq[ord(char) - ord('a')] += 1 for i in range(26): max\_freq[i] = max(max\_freq[i], char\_freq[i]) # List to store universal words universal\_words = [] # Check each word in words1 for word in words1: char\_freq = [0] \* 26 for char in word: char\_freq[ord(char) - ord('a')] += 1 # Check if word meets the universal criteria if all(char\_freq[i] >= max\_freq[i] for i in range(26)): universal\_words.append(word) return universal\_words

**Optimizations and Advantages:**

1. **Array Instead of Dictionary:**
   * Using an array of size 26 (max\_freq and char\_freq) instead of Counter significantly improves performance because array indexing is faster than dictionary lookups.
2. **Avoid Repeated Calculations:**
   * The frequency comparison between words1 and max\_freq is done in one pass using all().
3. **Preprocessing**words2**Efficiently:**
   * The maximum frequency for each character is calculated only once for all words in words2.

**Complexity Analysis:**

**Time Complexity:**

1. **Building**max\_freq**:**
   * Processing words2 requires O(N2⋅M)*O*(*N*2⋅*M*), where N2*N*2 is the number of words in words2, and M*M* is the average length of these words.
2. **Checking Universality of**words1**:**
   * Processing words1 requires O(N1⋅L)*O*(*N*1⋅*L*), where N1*N*1 is the number of words in words1, and L*L* is the average length of these words.

**Overall Time Complexity:**

O(N2⋅M+N1⋅L)*O*(*N*2⋅*M*+*N*1⋅*L*)

**Space Complexity:**

* Arrays max\_freq and char\_freq use O(26)=O(1)*O*(26)=*O*(1) space.

**Example Walkthrough:**

**Input:**

python

Copy code

words1 = ["amazon", "apple", "facebook", "google", "leetcode"] words2 = ["e", "o"]

**Execution:**

1. **Build**max\_freq**:**
   * For "e": max\_freq = [0, 0, ..., 1, ..., 0] (1 at index of 'e').
   * For "o": max\_freq = [0, 0, ..., 1, ..., 1, ..., 0] (1 at indices of 'e' and 'o').
2. **Check Universality for**words1**:**
   * "amazon": char\_freq = [2, 0, ..., 0]. Fails for 'e'. **Not universal**.
   * "apple": char\_freq = [1, 0, ..., 1]. Fails for 'o'. **Not universal**.
   * "facebook": char\_freq = [1, 0, ..., 1]. Passes. **Universal**.
   * "google": char\_freq = [0, 0, ..., 1, ..., 2, ..., 1]. Passes. **Universal**.
   * "leetcode": char\_freq = [0, 0, ..., 3]. Passes. **Universal**.
3. **Result:**

python

Copy code

["facebook", "google", "leetcode"]

This solution is both efficient and easy to understand

Top of Form

Bottom of Form