

Problem 916: Word Subsets

Problem Information

Difficulty: **Medium**

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

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 = ["lc","eo"]

Output:

["leetcode"]

Example 3:

Input:

words1 = ["acaac","cccbb","aacbb","caacc","bcbabb"], words2 = ["c","cc","b"]

Output:

["cccbb"]

Constraints:

$1 \leq \text{words1.length}, \text{words2.length} \leq 10$

4

$1 \leq \text{words1}[i].\text{length}, \text{words2}[i].\text{length} \leq 10$

words1[i]

and

words2[i]

consist only of lowercase English letters.

All the strings of

words1

are

unique

Code Snippets

C++:

```
class Solution {
public:
    vector<string> wordSubsets(vector<string>& words1, vector<string>& words2) {

    }
};
```

Java:

```
class Solution {
    public List<String> wordSubsets(String[] words1, String[] words2) {

    }
}
```

Python3:

```
class Solution:
    def wordSubsets(self, words1: List[str], words2: List[str]) -> List[str]:
```

Python:

```
class Solution(object):
    def wordSubsets(self, words1, words2):
        """
        :type words1: List[str]
        :type words2: List[str]
        :rtype: List[str]
        """
```

JavaScript:

```
/**
 * @param {string[]} words1
 * @param {string[]} words2
```

```

* @return {string[]}
*/
var wordSubsets = function(words1, words2) {

};

```

TypeScript:

```

function wordSubsets(words1: string[], words2: string[]): string[] {

};

```

C#:

```

public class Solution {
    public IList<string> WordSubsets(string[] words1, string[] words2) {

    }
}

```

C:

```

/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
char** wordSubsets(char** words1, int words1Size, char** words2, int
words2Size, int* returnSize) {

}

```

Go:

```

func wordSubsets(words1 []string, words2 []string) []string {

}

```

Kotlin:

```

class Solution {
    fun wordSubsets(words1: Array<String>, words2: Array<String>): List<String> {

    }
}

```

```
}
```

Swift:

```
class Solution {  
    func wordSubsets(_ words1: [String], _ words2: [String]) -> [String] {  
  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn word_subsets(words1: Vec<String>, words2: Vec<String>) -> Vec<String>  
    {  
  
    }  
}
```

Ruby:

```
# @param {String[]} words1  
# @param {String[]} words2  
# @return {String[]}  
def word_subsets(words1, words2)  
  
end
```

PHP:

```
class Solution {  
  
    /**  
     * @param String[] $words1  
     * @param String[] $words2  
     * @return String[]  
     */  
    function wordSubsets($words1, $words2) {  
  
    }  
}
```

Dart:

```
class Solution {  
  List<String> wordSubsets(List<String> words1, List<String> words2) {  
  
  }  
}
```

Scala:

```
object Solution {  
  def wordSubsets(words1: Array[String], words2: Array[String]): List[String] =  
  {  
  
  }  
}
```

Elixir:

```
defmodule Solution do  
  @spec word_subsets(words1 :: [String.t], words2 :: [String.t]) :: [String.t]  
  def word_subsets(words1, words2) do  
  
  end  
end
```

Erlang:

```
-spec word_subsets(Words1 :: [unicode:unicode_binary()], Words2 ::  
[unicode:unicode_binary()]) -> [unicode:unicode_binary()].  
word_subsets(Words1, Words2) ->  
.
```

Racket:

```
(define/contract (word-subsets words1 words2)  
  (-> (listof string?) (listof string?) (listof string?))  
  )
```

Solutions

C++ Solution:

```
/*
 * Problem: Word Subsets
 * Difficulty: Medium
 * Tags: array, string, hash
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map
 */

class Solution {
public:
    vector<string> wordSubsets(vector<string>& words1, vector<string>& words2) {

    }
};
```

Java Solution:

```
/**
 * Problem: Word Subsets
 * Difficulty: Medium
 * Tags: array, string, hash
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map
 */

class Solution {
    public List<String> wordSubsets(String[] words1, String[] words2) {

    }
}
```

Python3 Solution:

```
"""
Problem: Word Subsets
Difficulty: Medium
Tags: array, string, hash
```

```

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
    def wordSubsets(self, words1: List[str], words2: List[str]) -> List[str]:
        # TODO: Implement optimized solution
        pass

```

Python Solution:

```

class Solution(object):
    def wordSubsets(self, words1, words2):
        """
        :type words1: List[str]
        :type words2: List[str]
        :rtype: List[str]
        """

```

JavaScript Solution:

```

/**
 * Problem: Word Subsets
 * Difficulty: Medium
 * Tags: array, string, hash
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map
 */

/**
 * @param {string[]} words1
 * @param {string[]} words2
 * @return {string[]}
 */
var wordSubsets = function(words1, words2) {

};

```

TypeScript Solution:

```
/**
 * Problem: Word Subsets
 * Difficulty: Medium
 * Tags: array, string, hash
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map
 */

function wordSubsets(words1: string[], words2: string[]): string[] {

};
```

C# Solution:

```
/*
 * Problem: Word Subsets
 * Difficulty: Medium
 * Tags: array, string, hash
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map
 */

public class Solution {
    public IList<string> WordSubsets(string[] words1, string[] words2) {

    }
}
```

C Solution:

```
/*
 * Problem: Word Subsets
 * Difficulty: Medium
 * Tags: array, string, hash
 *
 * Approach: Use two pointers or sliding window technique
```

```

* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** wordSubsets(char** words1, int words1Size, char** words2, int
words2Size, int* returnSize) {

}

```

Go Solution:

```

// Problem: Word Subsets
// Difficulty: Medium
// Tags: array, string, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func wordSubsets(words1 []string, words2 []string) []string {

}

```

Kotlin Solution:

```

class Solution {
    fun wordSubsets(words1: Array<String>, words2: Array<String>): List<String> {

    }
}

```

Swift Solution:

```

class Solution {
    func wordSubsets(_ words1: [String], _ words2: [String]) -> [String] {

    }
}

```

Rust Solution:

```
// Problem: Word Subsets
// Difficulty: Medium
// Tags: array, string, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
    pub fn word_subsets(words1: Vec<String>, words2: Vec<String>) -> Vec<String>
    {

    }
}
```

Ruby Solution:

```
# @param {String[]} words1
# @param {String[]} words2
# @return {String[]}
def word_subsets(words1, words2)

end
```

PHP Solution:

```
class Solution {

    /**
     * @param String[] $words1
     * @param String[] $words2
     * @return String[]
     */
    function wordSubsets($words1, $words2) {

    }
}
```

Dart Solution:

```

class Solution {
  List<String> wordSubsets(List<String> words1, List<String> words2) {

  }
}

```

Scala Solution:

```

object Solution {
  def wordSubsets(words1: Array[String], words2: Array[String]): List[String] =
  {

  }
}

```

Elixir Solution:

```

defmodule Solution do
  @spec word_subsets(words1 :: [String.t], words2 :: [String.t]) :: [String.t]
  def word_subsets(words1, words2) do

  end
end

```

Erlang Solution:

```

-spec word_subsets(Words1 :: [unicode:unicode_binary()], Words2 ::
[unicode:unicode_binary()]) -> [unicode:unicode_binary()].
word_subsets(Words1, Words2) ->
.

```

Racket Solution:

```

(define/contract (word-subsets words1 words2)
  (-> (listof string?) (listof string?) (listof string?))
  )

```