Problem 2452: Words Within Two Edits of Dictionary

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

You are given two string arrays,

queries

and

dictionary

. All words in each array comprise of lowercase English letters and have the same length.

In one

edit

you can take a word from

queries

, and change any letter in it to any other letter. Find all words from

queries

that, after a

maximum

of two edits, equal some word from

dictionary

.

Return

a list of all words from

queries

,

that match with some word from

dictionary

after a maximum of

two edits

. Return the words in the

same order

they appear in

queries

.

Example 1:

Input:

queries = ["word","note","ants","wood"], dictionary = ["wood","joke","moat"]

Output:

["word","note","wood"]

Explanation:

- Changing the 'r' in "word" to 'o' allows it to equal the dictionary word "wood". - Changing the 'n' to 'j' and the 't' to 'k' in "note" changes it to "joke". - It would take more than 2 edits for "ants" to equal a dictionary word. - "wood" can remain unchanged (0 edits) and match the corresponding dictionary word. Thus, we return ["word","note","wood"].

Example 2:

Input:

queries = ["yes"], dictionary = ["not"]

Output:

[]

Explanation:

Applying any two edits to "yes" cannot make it equal to "not". Thus, we return an empty array.

Constraints:

1 <= queries.length, dictionary.length <= 100

n == queries[i].length == dictionary[j].length

1 <= n <= 100

All

queries[i]

and

dictionary[j]

are composed of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
vector<string> twoEditWords(vector<string>& queries, vector<string>&
dictionary) {

}
};
```

Java:

```
class Solution {
public List<String> twoEditWords(String[] queries, String[] dictionary) {

}
}
```

Python3:

```
class Solution:
def twoEditWords(self, queries: List[str], dictionary: List[str]) ->
List[str]:
```

Python:

```
class Solution(object):
def twoEditWords(self, queries, dictionary):
"""
:type queries: List[str]
:type dictionary: List[str]
:rtype: List[str]
"""
```

JavaScript:

```
/**
* @param {string[]} queries
* @param {string[]} dictionary
* @return {string[]}
*/
var twoEditWords = function(queries, dictionary) {

};
```

TypeScript:

```
function twoEditWords(queries: string[], dictionary: string[]): string[] {

};
```

C#:

```
public class Solution {
public IList<string> TwoEditWords(string[] queries, string[] dictionary) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** twoEditWords(char** queries, int queriesSize, char** dictionary, int
dictionarySize, int* returnSize) {

}
```

Go:

```
func twoEditWords(queries []string, dictionary []string) []string {

}
```

Kotlin:

```
class Solution {
fun twoEditWords(queries: Array<String>, dictionary: Array<String>):
```

```
List<String> {

}
}
```

Swift:

```
class Solution {
func twoEditWords(_ queries: [String], _ dictionary: [String]) -> [String] {

}
}
```

Rust:

```
impl Solution {
pub fn two_edit_words(queries: Vec<String>, dictionary: Vec<String>) ->
Vec<String> {

}
}
```

Ruby:

```
# @param {String[]} queries
# @param {String[]} dictionary
# @return {String[]}
def two_edit_words(queries, dictionary)

end
```

PHP:

```
class Solution {

/**
* @param String[] $queries
* @param String[] $dictionary
* @return String[]
*/
function twoEditWords($queries, $dictionary) {

}
```

```
}
```

Dart:

```
class Solution {
List<String> twoEditWords(List<String> queries, List<String> dictionary) {

}
}
```

Scala:

```
object Solution {
def twoEditWords(queries: Array[String], dictionary: Array[String]):
List[String] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec two_edit_words(queries :: [String.t], dictionary :: [String.t]) ::
[String.t]
def two_edit_words(queries, dictionary) do

end
end
```

Erlang:

```
-spec two_edit_words(Queries :: [unicode:unicode_binary()], Dictionary ::
[unicode:unicode_binary()]) -> [unicode:unicode_binary()].
two_edit_words(Queries, Dictionary) ->
.
```

Racket:

```
(define/contract (two-edit-words queries dictionary)
(-> (listof string?) (listof string?) (listof string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Words Within Two Edits of Dictionary
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
vector<string> twoEditWords(vector<string>& queries, vector<string>&
dictionary) {

}
};
```

Java Solution:

```
/**
* Problem: Words Within Two Edits of Dictionary
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public List<String> twoEditWords(String[] queries, String[] dictionary) {

}
}
```

Python3 Solution:

```
"""
Problem: Words Within Two Edits of Dictionary
Difficulty: Medium
Tags: array, string

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def twoEditWords(self, queries: List[str], dictionary: List[str]) ->
List[str]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def twoEditWords(self, queries, dictionary):
"""
:type queries: List[str]
:type dictionary: List[str]
:rtype: List[str]
"""
```

JavaScript Solution:

```
/**
* Problem: Words Within Two Edits of Dictionary
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {string[]} queries
* @param {string[]} dictionary
* @return {string[]}
```

```
*/
var twoEditWords = function(queries, dictionary) {

};
```

TypeScript Solution:

```
/**
* Problem: Words Within Two Edits of Dictionary
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function twoEditWords(queries: string[], dictionary: string[]): string[] {

};
```

C# Solution:

```
/*
* Problem: Words Within Two Edits of Dictionary
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public IList<string> TwoEditWords(string[] queries, string[] dictionary) {

}
}
```

C Solution:

```
/*
* Problem: Words Within Two Edits of Dictionary
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** twoEditWords(char** queries, int queriesSize, char** dictionary, int
dictionarySize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Words Within Two Edits of Dictionary
// Difficulty: Medium
// Tags: array, string
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func twoEditWords(queries []string, dictionary []string) []string {

}
```

Kotlin Solution:

```
class Solution {
fun twoEditWords(queries: Array<String>, dictionary: Array<String>):
List<String> {

}
}
```

Swift Solution:

```
class Solution {
func twoEditWords(_ queries: [String], _ dictionary: [String]) -> [String] {

}
}
```

Rust Solution:

```
// Problem: Words Within Two Edits of Dictionary
// Difficulty: Medium
// Tags: array, string
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn two_edit_words(queries: Vec<String>, dictionary: Vec<String>) ->
Vec<String> {

}
}
```

Ruby Solution:

```
# @param {String[]} queries
# @param {String[]} dictionary
# @return {String[]}
def two_edit_words(queries, dictionary)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $queries
* @param String[] $dictionary
* @return String[]
*/
function twoEditWords($queries, $dictionary) {
```

```
}
}
```

Dart Solution:

```
class Solution {
List<String> twoEditWords(List<String> queries, List<String> dictionary) {

}
}
```

Scala Solution:

```
object Solution {
def twoEditWords(queries: Array[String], dictionary: Array[String]):
List[String] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec two_edit_words(queries :: [String.t], dictionary :: [String.t]) ::
[String.t]
def two_edit_words(queries, dictionary) do

end
end
```

Erlang Solution:

```
-spec two_edit_words(Queries :: [unicode:unicode_binary()], Dictionary ::
[unicode:unicode_binary()]) -> [unicode:unicode_binary()].
two_edit_words(Queries, Dictionary) ->
.
```

Racket Solution:

```
(define/contract (two-edit-words queries dictionary)
(-> (listof string?) (listof string?) (listof string?))
)
```