Problem 1181: Before and After Puzzle

Problem Information

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

Problem Description

Given a list of

phrases

, generate a list of Before and After puzzles.

A

phrase

is a string that consists of lowercase English letters and spaces only. No space appears in the start or the end of a phrase. There are no consecutive spaces in a phrase.

Before and After puzzles

are phrases that are formed by merging two phrases where the

last word of the first phrase

is the same as the

first word of the second phrase

. Note that only the

last word of the first phrase

and the

first word of the second phrase

are merged in this process.

Return the Before and After puzzles that can be formed by every two phrases

phrases[i]

and

phrases[j]

where

i != j

. Note that the order of matching two phrases matters, we want to consider both orders.

You should return a list of

distinct

strings

sorted lexicographically

, after removing all

duplicate

phrases in the generated Before and After puzzles.

Example 1:

Input:

phrases = ["writing code","code rocks"]

Output:

["writing code rocks"]

Example 2:

Input:

phrases = ["mission statement","a quick bite to eat","a chip off the old block","chocolate bar","mission impossible","a man on a mission","block party","eat my words","bar of soap"]

Output:

["a chip off the old block party","a man on a mission impossible","a man on a mission statement","a quick bite to eat my words","chocolate bar of soap"]

Example 3:

Input:

phrases = ["a","b","a"]

Output:

["a"]

Example 4:

Input:

phrases = ["ab ba","ba ab","ab ba"]

Output:

["ab ba ab","ba ab ba"]

Constraints:

1 <= phrases.length <= 100

1 <= phrases[i].length <= 100

Code Snippets

C++:

```
class Solution {
public:
vector<string> beforeAndAfterPuzzles(vector<string>& phrases) {

}
};
```

Java:

```
class Solution {
public List<String> beforeAndAfterPuzzles(String[] phrases) {

}
}
```

Python3:

```
class Solution:
def beforeAndAfterPuzzles(self, phrases: List[str]) -> List[str]:
```

Python:

```
class Solution(object):
def beforeAndAfterPuzzles(self, phrases):
"""
:type phrases: List[str]
:rtype: List[str]
"""
```

JavaScript:

```
/**
* @param {string[]} phrases
* @return {string[]}
*/
var beforeAndAfterPuzzles = function(phrases) {

};
```

TypeScript:

```
function beforeAndAfterPuzzles(phrases: string[]): string[] {

};
```

C#:

```
public class Solution {
public IList<string> BeforeAndAfterPuzzles(string[] phrases) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** beforeAndAfterPuzzles(char** phrases, int phrasesSize, int*
returnSize) {

}
```

Go:

```
func beforeAndAfterPuzzles(phrases []string) []string {

}
```

Kotlin:

```
class Solution {
fun beforeAndAfterPuzzles(phrases: Array<String>): List<String> {

}

}
```

Swift:

```
class Solution {
func beforeAndAfterPuzzles(_ phrases: [String]) -> [String] {

}
}
```

Rust:

```
impl Solution {
pub fn before_and_after_puzzles(phrases: Vec<String>) -> Vec<String> {

}
}
```

Ruby:

```
# @param {String[]} phrases
# @return {String[]}
def before_and_after_puzzles(phrases)

end
```

PHP:

```
class Solution {

/**
* @param String[] $phrases
* @return String[]
*/
function beforeAndAfterPuzzles($phrases) {

}
}
```

Dart:

```
class Solution {
List<String> beforeAndAfterPuzzles(List<String> phrases) {

}
}
```

Scala:

```
object Solution {
def beforeAndAfterPuzzles(phrases: Array[String]): List[String] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec before_and_after_puzzles(phrases :: [String.t]) :: [String.t]
def before_and_after_puzzles(phrases) do

end
end
```

Erlang:

```
-spec before_and_after_puzzles(Phrases :: [unicode:unicode_binary()]) ->
[unicode:unicode_binary()].
before_and_after_puzzles(Phrases) ->
.
```

Racket:

```
(define/contract (before-and-after-puzzles phrases)
(-> (listof string?) (listof string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Before and After Puzzle
* Difficulty: Medium
* Tags: array, string, graph, hash, sort
*
* 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> beforeAndAfterPuzzles(vector<string>& phrases) {

}
};
```

Java Solution:

```
/**
* Problem: Before and After Puzzle
* Difficulty: Medium
* Tags: array, string, graph, hash, sort
*
* 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> beforeAndAfterPuzzles(String[] phrases) {

}
}
```

Python3 Solution:

```
"""
Problem: Before and After Puzzle
Difficulty: Medium
Tags: array, string, graph, hash, sort
```

```
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 beforeAndAfterPuzzles(self, phrases: List[str]) -> List[str]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def beforeAndAfterPuzzles(self, phrases):
"""
:type phrases: List[str]
:rtype: List[str]
"""
```

JavaScript Solution:

```
/**
* Problem: Before and After Puzzle
* Difficulty: Medium
* Tags: array, string, graph, hash, sort
*
* 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[]} phrases
* @return {string[]}
*/
var beforeAndAfterPuzzles = function(phrases) {

};
```

TypeScript Solution:

```
/**
* Problem: Before and After Puzzle
* Difficulty: Medium
* Tags: array, string, graph, hash, sort
*
* 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 beforeAndAfterPuzzles(phrases: string[]): string[] {

};
```

C# Solution:

```
/*
* Problem: Before and After Puzzle
* Difficulty: Medium
* Tags: array, string, graph, hash, sort
*
* 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> BeforeAndAfterPuzzles(string[] phrases) {

}
}
```

C Solution:

```
/*
* Problem: Before and After Puzzle
* Difficulty: Medium
* Tags: array, string, graph, hash, sort
*
* 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** beforeAndAfterPuzzles(char** phrases, int phrasesSize, int*
returnSize) {

}
```

Go Solution:

```
// Problem: Before and After Puzzle
// Difficulty: Medium
// Tags: array, string, graph, hash, sort
//
// 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 beforeAndAfterPuzzles(phrases []string) []string {

}
```

Kotlin Solution:

```
class Solution {
fun beforeAndAfterPuzzles(phrases: Array<String>): List<String> {

}
}
```

Swift Solution:

```
class Solution {
func beforeAndAfterPuzzles(_ phrases: [String]) -> [String] {

}
}
```

Rust Solution:

```
// Problem: Before and After Puzzle
// Difficulty: Medium
// Tags: array, string, graph, hash, sort
//
// 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 before_and_after_puzzles(phrases: Vec<String>) -> Vec<String> {

}
}
```

Ruby Solution:

```
# @param {String[]} phrases
# @return {String[]}
def before_and_after_puzzles(phrases)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $phrases
* @return String[]
*/
function beforeAndAfterPuzzles($phrases) {

}
}
```

Dart Solution:

```
class Solution {
List<String> beforeAndAfterPuzzles(List<String> phrases) {

}
}
```

Scala Solution:

```
object Solution {
def beforeAndAfterPuzzles(phrases: Array[String]): List[String] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec before_and_after_puzzles(phrases :: [String.t]) :: [String.t]
def before_and_after_puzzles(phrases) do

end
end
```

Erlang Solution:

```
-spec before_and_after_puzzles(Phrases :: [unicode:unicode_binary()]) ->
[unicode:unicode_binary()].
before_and_after_puzzles(Phrases) ->
.
```

Racket Solution:

```
(define/contract (before-and-after-puzzles phrases)
(-> (listof string?) (listof string?))
)
```