Problem 599: Minimum Index Sum of Two Lists

Problem Information

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

Problem Description

Given two arrays of strings

list1

and

list2

, find the

common strings with the least index sum

.

A

common string

is a string that appeared in both

list1

and

list2

.

A

common string with the least index sum

is a common string such that if it appeared at

list1[i]

and

list2[j]

then

i + j

should be the minimum value among all the other

common strings

.

Return

all the

common strings with the least index sum

. Return the answer in

any order

.

Example 1:

Input:

list1 = ["Shogun","Tapioca Express","Burger King","KFC"], list2 = ["Piatti","The Grill at Torrey Pines","Hungry Hunter Steakhouse","Shogun"]

Output:

["Shogun"]

Explanation:

The only common string is "Shogun".

Example 2:

Input:

list1 = ["Shogun","Tapioca Express","Burger King","KFC"], list2 = ["KFC","Shogun","Burger King"]

Output:

["Shogun"]

Explanation:

The common string with the least index sum is "Shogun" with index sum = (0 + 1) = 1.

Example 3:

Input:

list1 = ["happy","sad","good"], list2 = ["sad","happy","good"]

Output:

["sad","happy"]

Explanation:

There are three common strings: "happy" with index sum = (0 + 1) = 1. "sad" with index sum = (1 + 0) = 1. "good" with index sum = (2 + 2) = 4. The strings with the least index sum are "sad" and "happy".

Constraints:

1 <= list1.length, list2.length <= 1000

1 <= list1[i].length, list2[i].length <= 30

list1[i]

and

list2[i]

consist of spaces

' '

and English letters.

All the strings of

list1

are

unique

.

All the strings of

list2

are

unique

.

There is at least a common string between

list1

and

list2

.

Code Snippets

C++:

```
class Solution {
public:
vector<string> findRestaurant(vector<string>& list1, vector<string>& list2) {

}
};
```

Java:

```
class Solution {
public String[] findRestaurant(String[] list1, String[] list2) {

}
}
```

Python3:

```
class Solution:
def findRestaurant(self, list1: List[str], list2: List[str]) -> List[str]:
```

Python:

```
class Solution(object):
def findRestaurant(self, list1, list2):
```

```
"""
:type list1: List[str]
:type list2: List[str]
:rtype: List[str]
"""
```

JavaScript:

```
/**
* @param {string[]} list1
* @param {string[]} list2
* @return {string[]}
*/
var findRestaurant = function(list1, list2) {

};
```

TypeScript:

```
function findRestaurant(list1: string[], list2: string[]): string[] {

};
```

C#:

```
public class Solution {
public string[] FindRestaurant(string[] list1, string[] list2) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** findRestaurant(char** list1, int list1Size, char** list2, int
list2Size, int* returnSize) {

}
```

Go:

```
func findRestaurant(list1 []string, list2 []string) []string {

}
```

Kotlin:

```
class Solution {
fun findRestaurant(list1: Array<String>, list2: Array<String>): Array<String>
{

}
}
```

Swift:

```
class Solution {
func findRestaurant(_ list1: [String], _ list2: [String]) -> [String] {

}
}
```

Rust:

```
impl Solution {
pub fn find_restaurant(list1: Vec<String>, list2: Vec<String>) -> Vec<String>
{

}
}
```

Ruby:

```
# @param {String[]} list1
# @param {String[]} list2
# @return {String[]}
def find_restaurant(list1, list2)

end
```

PHP:

```
class Solution {
```

```
/**
* @param String[] $list1
* @param String[] $list2
* @return String[]
*/
function findRestaurant($list1, $list2) {

}
}
```

Dart:

```
class Solution {
List<String> findRestaurant(List<String> list1, List<String> list2) {

}
}
```

Scala:

```
object Solution {
def findRestaurant(list1: Array[String], list2: Array[String]): Array[String]
= {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_restaurant(list1 :: [String.t], list2 :: [String.t]) :: [String.t]
def find_restaurant(list1, list2) do

end
end
```

Erlang:

```
-spec find_restaurant(List1 :: [unicode:unicode_binary()], List2 ::
[unicode:unicode_binary()]) -> [unicode:unicode_binary()].
find_restaurant(List1, List2) ->
.
```

Racket:

```
(define/contract (find-restaurant list1 list2)
(-> (listof string?) (listof string?) (listof string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Index Sum of Two Lists
* Difficulty: Easy
* 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> findRestaurant(vector<string>& list1, vector<string>& list2) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Index Sum of Two Lists
* Difficulty: Easy
* 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 String[] findRestaurant(String[] list1, String[] list2) {
}
}
```

Python3 Solution:

```
"""
Problem: Minimum Index Sum of Two Lists
Difficulty: Easy
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 findRestaurant(self, list1: List[str], list2: List[str]) -> List[str]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findRestaurant(self, list1, list2):
"""
:type list1: List[str]
:type list2: List[str]
:rtype: List[str]
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Index Sum of Two Lists
* Difficulty: Easy
* 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[]} list1
* @param {string[]} list2
* @return {string[]}
*/
var findRestaurant = function(list1, list2) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Index Sum of Two Lists
* Difficulty: Easy
* 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 findRestaurant(list1: string[], list2: string[]): string[] {

};
```

C# Solution:

```
/*
* Problem: Minimum Index Sum of Two Lists
* Difficulty: Easy
* 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 string[] FindRestaurant(string[] list1, string[] list2) {
}
}
```

C Solution:

```
/*
* Problem: Minimum Index Sum of Two Lists
* Difficulty: Easy
* 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** findRestaurant(char** list1, int list1Size, char** list2, int
list2Size, int* returnSize) {

}
```

Go Solution:

```
// Problem: Minimum Index Sum of Two Lists
// Difficulty: Easy
// 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 findRestaurant(list1 []string, list2 []string) []string {

}
```

Kotlin Solution:

```
class Solution {
fun findRestaurant(list1: Array<String>, list2: Array<String>): Array<String>
{

}
}
```

Swift Solution:

```
class Solution {
func findRestaurant(_ list1: [String], _ list2: [String]) -> [String] {

}
}
```

Rust Solution:

```
// Problem: Minimum Index Sum of Two Lists
// Difficulty: Easy
// 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 find_restaurant(list1: Vec<String>, list2: Vec<String>) -> Vec<String>
{

}
}
```

Ruby Solution:

```
# @param {String[]} list1
# @param {String[]} list2
# @return {String[]}
def find_restaurant(list1, list2)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $list1
* @param String[] $list2
* @return String[]
*/
function findRestaurant($list1, $list2) {

}
}
```

Dart Solution:

```
class Solution {
List<String> findRestaurant(List<String> list1, List<String> list2) {

}
}
```

Scala Solution:

```
object Solution {
def findRestaurant(list1: Array[String], list2: Array[String]): Array[String]
= {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_restaurant(list1 :: [String.t], list2 :: [String.t]) :: [String.t]
def find_restaurant(list1, list2) do

end
end
```

Erlang Solution:

```
-spec find_restaurant(List1 :: [unicode:unicode_binary()], List2 ::
[unicode:unicode_binary()]) -> [unicode:unicode_binary()].
```

```


find_restaurant(List1, List2) ->
.
```

Racket Solution:

```
(define/contract (find-restaurant list1 list2)
(-> (listof string?) (listof string?) (listof string?))
)
```