Problem 2115: Find All Possible Recipes from Given Supplies

Problem Information

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

Problem Description

You have information about

n

different recipes. You are given a string array

recipes

and a 2D string array

ingredients

. The

i

th

recipe has the name

recipes[i]

, and you can

create

it if you have

all

the needed ingredients from

ingredients[i]

. A recipe can also be an ingredient for

other

recipes, i.e.,

ingredients[i]

may contain a string that is in

recipes

.

You are also given a string array

supplies

containing all the ingredients that you initially have, and you have an infinite supply of all of them.

Return

a list of all the recipes that you can create.

You may return the answer in

any order

.

Note that two recipes may contain each other in their ingredients.

Example 1:

Input:

recipes = ["bread"], ingredients = [["yeast","flour"]], supplies = ["yeast","flour","corn"]

Output:

["bread"]

Explanation:

We can create "bread" since we have the ingredients "yeast" and "flour".

Example 2:

Input:

recipes = ["bread","sandwich"], ingredients = [["yeast","flour"],["bread","meat"]], supplies = ["yeast","flour","meat"]

Output:

["bread","sandwich"]

Explanation:

We can create "bread" since we have the ingredients "yeast" and "flour". We can create "sandwich" since we have the ingredient "meat" and can create the ingredient "bread".

Example 3:

Input:

recipes = ["bread","sandwich","burger"], ingredients = [["yeast","flour"],["bread","meat"],["sandwich","meat","bread"]], supplies = ["yeast","flour","meat"]

Output:

["bread","sandwich","burger"]

Explanation:

We can create "bread" since we have the ingredients "yeast" and "flour". We can create "sandwich" since we have the ingredient "meat" and can create the ingredient "bread". We can create "burger" since we have the ingredient "meat" and can create the ingredients "bread" and "sandwich".

Constraints:

n == recipes.length == ingredients.length

1 <= n <= 100

1 <= ingredients[i].length, supplies.length <= 100

1 <= recipes[i].length, ingredients[i][j].length, supplies[k].length <= 10

recipes[i], ingredients[i][j]

, and

supplies[k]

consist only of lowercase English letters.

All the values of

recipes

and

supplies

combined are unique.

Each

ingredients[i]

does not contain any duplicate values.

Code Snippets

C++:

```
class Solution {
public:
vector<string> findAllRecipes(vector<string>& recipes,
vector<vector<string>>& ingredients, vector<string>& supplies) {

}
};
```

Java:

```
class Solution {
public List<String> findAllRecipes(String[] recipes, List<List<String>>
ingredients, String[] supplies) {

}
}
```

Python3:

```
class Solution:
def findAllRecipes(self, recipes: List[str], ingredients: List[List[str]],
supplies: List[str]) -> List[str]:
```

Python:

```
class Solution(object):
def findAllRecipes(self, recipes, ingredients, supplies):
"""
:type recipes: List[str]
:type ingredients: List[List[str]]
:type supplies: List[str]
```

```
:rtype: List[str]
"""
```

JavaScript:

```
/**
* @param {string[]} recipes
* @param {string[][]} ingredients
* @param {string[]} supplies
* @return {string[]}
*/
var findAllRecipes = function(recipes, ingredients, supplies) {

};
```

TypeScript:

```
function findAllRecipes(recipes: string[], ingredients: string[][], supplies:
string[]): string[] {

};
```

C#:

```
public class Solution {
public IList<string> FindAllRecipes(string[] recipes, IList<IList<string>>
ingredients, string[] supplies) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** findAllRecipes(char** recipes, int recipesSize, char*** ingredients,
int ingredientsSize, int* ingredientsColSize, char** supplies, int
suppliesSize, int* returnSize) {

}
```

Go:

```
func findAllRecipes(recipes []string, ingredients [][]string, supplies
[]string) []string {

}
```

Kotlin:

```
class Solution {
fun findAllRecipes(recipes: Array<String>, ingredients: List<List<String>>,
supplies: Array<String>): List<String> {

}
}
```

Swift:

```
class Solution {
func findAllRecipes(_ recipes: [String], _ ingredients: [[String]], _
supplies: [String]) -> [String] {

}
}
```

Rust:

```
impl Solution {
pub fn find_all_recipes(recipes: Vec<String>, ingredients: Vec<Vec<String>>,
supplies: Vec<String>) -> Vec<String> {

}
}
```

Ruby:

```
# @param {String[]} recipes
# @param {String[][]} ingredients
# @param {String[]} supplies
# @return {String[]}
def find_all_recipes(recipes, ingredients, supplies)

end
```

PHP:

```
class Solution {

/**
* @param String[] $recipes
* @param String[][] $ingredients
* @param String[] $supplies
* @return String[]
*/
function findAllRecipes($recipes, $ingredients, $supplies) {

}
}
```

Dart:

```
class Solution {
List<String> findAllRecipes(List<String> recipes, List<List<String>>
ingredients, List<String> supplies) {

}
}
```

Scala:

```
object Solution {
def findAllRecipes(recipes: Array[String], ingredients: List[List[String]],
supplies: Array[String]): List[String] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_all_recipes(recipes :: [String.t], ingredients :: [[String.t]],
supplies :: [String.t]) :: [String.t]
def find_all_recipes(recipes, ingredients, supplies) do

end
end
```

Erlang:

```
-spec find_all_recipes(Recipes :: [unicode:unicode_binary()], Ingredients ::
[[unicode:unicode_binary()]], Supplies :: [unicode:unicode_binary()]) ->
[unicode:unicode_binary()].
find_all_recipes(Recipes, Ingredients, Supplies) ->
.
```

Racket:

```
(define/contract (find-all-recipes recipes ingredients supplies)
(-> (listof string?) (listof (listof string?)) (listof string?) (listof
string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find All Possible Recipes from Given Supplies
* 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> findAllRecipes(vector<string>& recipes,
vector<vector<string>>& ingredients, vector<string>& supplies) {

}
};
```

Java Solution:

```
/**
* Problem: Find All Possible Recipes from Given Supplies
```

```
* 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> findAllRecipes(String[] recipes, List<List<String>>
ingredients, String[] supplies) {

}
}
```

Python3 Solution:

```
"""
Problem: Find All Possible Recipes from Given Supplies
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 findAllRecipes(self, recipes: List[str], ingredients: List[List[str]],
supplies: List[str]) -> List[str]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findAllRecipes(self, recipes, ingredients, supplies):
"""
:type recipes: List[str]
:type ingredients: List[List[str]]
:type supplies: List[str]
```

```
:rtype: List[str]
"""
```

JavaScript Solution:

```
/**
* Problem: Find All Possible Recipes from Given Supplies
* 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[]} recipes
* @param {string[][]} ingredients
* @param {string[]} supplies
* @return {string[]}
*/
var findAllRecipes = function(recipes, ingredients, supplies) {

};
```

TypeScript Solution:

```
/**
* Problem: Find All Possible Recipes from Given Supplies
* 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 findAllRecipes(recipes: string[], ingredients: string[][], supplies:
string[]): string[] {

};
```

C# Solution:

```
/*
* Problem: Find All Possible Recipes from Given Supplies
* 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> FindAllRecipes(string[] recipes, IList<IList<string>>
ingredients, string[] supplies) {

}
}
```

C Solution:

```
/*
* Problem: Find All Possible Recipes from Given Supplies
* 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** findAllRecipes(char** recipes, int recipesSize, char*** ingredients,
int ingredientsSize, int* ingredientsColSize, char** supplies, int
suppliesSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Find All Possible Recipes from Given Supplies
// 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 findAllRecipes(recipes []string, ingredients [][]string, supplies
[]string) []string {

}
```

Kotlin Solution:

```
class Solution {
fun findAllRecipes(recipes: Array<String>, ingredients: List<List<String>>,
supplies: Array<String>): List<String> {

}
}
```

Swift Solution:

```
class Solution {
func findAllRecipes(_ recipes: [String], _ ingredients: [[String]], _
supplies: [String]) -> [String] {

}
}
```

Rust Solution:

```
// Problem: Find All Possible Recipes from Given Supplies
// 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 find_all_recipes(recipes: Vec<String>, ingredients: Vec<Vec<String>>,
supplies: Vec<String>) -> Vec<String> {

}
}
```

Ruby Solution:

```
# @param {String[]} recipes
# @param {String[][]} ingredients
# @param {String[]} supplies
# @return {String[]}
def find_all_recipes(recipes, ingredients, supplies)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $recipes
* @param String[][] $ingredients
* @param String[] $supplies
* @return String[]
*/
function findAllRecipes($recipes, $ingredients, $supplies) {

}
}
```

Dart Solution:

```
class Solution {
List<String> findAllRecipes(List<String> recipes, List<List<String>>
ingredients, List<String> supplies) {

}
}
```

Scala Solution:

```
object Solution {
def findAllRecipes(recipes: Array[String], ingredients: List[List[String]],
supplies: Array[String]): List[String] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_all_recipes(recipes :: [String.t], ingredients :: [[String.t]],
supplies :: [String.t]) :: [String.t]
def find_all_recipes(recipes, ingredients, supplies) do

end
end
```

Erlang Solution:

```
-spec find_all_recipes(Recipes :: [unicode:unicode_binary()], Ingredients ::
[[unicode:unicode_binary()]], Supplies :: [unicode:unicode_binary()]) ->
[unicode:unicode_binary()].
find_all_recipes(Recipes, Ingredients, Supplies) ->
.
```

Racket Solution:

```
(define/contract (find-all-recipes recipes ingredients supplies)
(-> (listof string?) (listof (listof string?)) (listof string?) (listof
string?))
)
```