Problem 721: Accounts Merge

Problem Information

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

Problem Description

Given a list of

accounts

where each element

accounts[i]

is a list of strings, where the first element

accounts[i][0]

is a name, and the rest of the elements are

emails

representing emails of the account.

Now, we would like to merge these accounts. Two accounts definitely belong to the same person if there is some common email to both accounts. Note that even if two accounts have the same name, they may belong to different people as people could have the same name. A person can have any number of accounts initially, but all of their accounts definitely have the same name.

After merging the accounts, return the accounts in the following format: the first element of each account is the name, and the rest of the elements are emails in sorted order

. The accounts themselves can be returned in

any order

.

Example 1:

Input:

accounts = [["John","johnsmith@mail.com","john_newyork@mail.com"],["John","johnsmith@mail.com","john00@mail.com"],["Mary","mary@mail.com"],["John","johnnybravo@mail.com"]]

Output:

[["John","john00@mail.com","john_newyork@mail.com","johnsmith@mail.com"],["Mary","mary@mail.com"],["John","johnnybravo@mail.com"]]

Explanation:

The first and second John's are the same person as they have the common email "johnsmith@mail.com". The third John and Mary are different people as none of their email addresses are used by other accounts. We could return these lists in any order, for example the answer [['Mary', 'mary@mail.com'], ['John', 'johnnybravo@mail.com'], ['John', 'john00@mail.com', 'john_newyork@mail.com', 'johnsmith@mail.com']] would still be accepted.

Example 2:

Input:

accounts = [["Gabe","Gabe0@m.co","Gabe3@m.co","Gabe1@m.co"],["Kevin","Kevin3@m.co","Kevin5@m.co","Kevin0@m.co"],["Ethan","Ethan5@m.co","Ethan4@m.co","Ethan0@m.co"],["Hanzo","Hanzo3@m.co","Hanzo1@m.co","Hanzo0@m.co"],["Fern","Fern5@m.co","Fern1@m.co","Fern0@m.co"]]

Output:

[["Ethan","Ethan0@m.co","Ethan4@m.co","Ethan5@m.co"],["Gabe","Gabe0@m.co","Gabe1@m.co","Gabe3@m.co"],["Hanzo","Hanzo0@m.co","Hanzo1@m.co","Hanzo3@m.co"],["Kevin","Kevin0@m.co","Kevin3@m.co","Kevin5@m.co"],["Fern","Fern0@m.co","Fern1@m.co","Fern5@m.co"]]

Constraints:

1 <= accounts.length <= 1000

2 <= accounts[i].length <= 10

1 <= accounts[i][j].length <= 30

accounts[i][0]

consists of English letters.

accounts[i][j] (for j > 0)

is a valid email.

Code Snippets

C++:

```
class Solution {
public:
vector<vector<string>> accountsMerge(vector<vector<string>>& accounts) {

}
};
```

Java:

```
class Solution {
public List<List<String>> accountsMerge(List<List<String>> accounts) {

}
}
```

Python3:

```
class Solution:
def accountsMerge(self, accounts: List[List[str]]) -> List[List[str]]:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function accountsMerge(accounts: string[][]): string[][] {

};
```

C#:

```
public class Solution {
public IList<IList<string>> AccountsMerge(IList<IList<string>> accounts) {

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
char*** accountsMerge(char*** accounts, int accountsSize, int*
accountsColSize, int* returnSize, int** returnColumnSizes) {

}
```

Go:

```
func accountsMerge(accounts [][]string) [][]string {

}
```

Kotlin:

```
class Solution {
fun accountsMerge(accounts: List<List<String>>): List<List<String>> {

}
}
```

Swift:

```
class Solution {
func accountsMerge(_ accounts: [[String]]) -> [[String]] {

}
}
```

Rust:

```
impl Solution {
pub fn accounts_merge(accounts: Vec<Vec<String>>) -> Vec<Vec<String>> {

}
}
```

Ruby:

```
# @param {String[][]} accounts
# @return {String[][]}
```

```
def accounts_merge(accounts)

end
```

PHP:

```
class Solution {

/**
* @param String[][] $accounts
* @return String[][]
*/
function accountsMerge($accounts) {

}
}
```

Dart:

```
class Solution {
List<List<String>> accountsMerge(List<List<String>> accounts) {

}
}
```

Scala:

```
object Solution {
def accountsMerge(accounts: List[List[String]]): List[List[String]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec accounts_merge(accounts :: [[String.t]]) :: [[String.t]]
def accounts_merge(accounts) do

end
end
```

Erlang:

```
-spec accounts_merge(Accounts :: [[unicode:unicode_binary()]]) ->
[[unicode:unicode_binary()]].
accounts_merge(Accounts) ->
.
```

Racket:

```
(define/contract (accounts-merge accounts)
(-> (listof (listof string?)) (listof (listof string?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Accounts Merge
* Difficulty: Medium
* Tags: array, string, graph, hash, sort, search
*
* 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<vector<string>> accountsMerge(vector<vector<string>>& accounts) {

}
};
```

Java Solution:

```
/**
* Problem: Accounts Merge
* Difficulty: Medium
* Tags: array, string, graph, hash, sort, search
*
```

```
* 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<List<String>> accountsMerge(List<List<String>> accounts) {

}
}
```

Python3 Solution:

```
"""
Problem: Accounts Merge
Difficulty: Medium
Tags: array, string, graph, hash, sort, search

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Accounts Merge
* Difficulty: Medium
* Tags: array, string, graph, hash, sort, search
*
* 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[][]} accounts
* @return {string[][]}
*/
var accountsMerge = function(accounts) {

};
```

TypeScript Solution:

```
/**
* Problem: Accounts Merge
* Difficulty: Medium
* Tags: array, string, graph, hash, sort, search
*
* 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 accountsMerge(accounts: string[][]): string[][] {

};
```

C# Solution:

```
/*
* Problem: Accounts Merge
* Difficulty: Medium
* Tags: array, string, graph, hash, sort, search
*
* 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<IList<string>> AccountsMerge(IList<IList<string>> accounts) {

}
}
```

C Solution:

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

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
char*** accountsMerge(char*** accounts, int accountsSize, int*
accountsColSize, int* returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Accounts Merge
// Difficulty: Medium
// Tags: array, string, graph, hash, sort, search
//
// 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 accountsMerge(accounts [][]string) [][]string {

}
```

Kotlin Solution:

```
class Solution {
fun accountsMerge(accounts: List<List<String>>): List<List<String>> {

}
}
```

Swift Solution:

```
class Solution {
func accountsMerge(_ accounts: [[String]]) -> [[String]] {

}
}
```

Rust Solution:

```
// Problem: Accounts Merge
// Difficulty: Medium
// Tags: array, string, graph, hash, sort, search
//
// 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 accounts_merge(accounts: Vec<Vec<String>>) -> Vec<Vec<String>> {

}
}
```

Ruby Solution:

```
# @param {String[][]} accounts
# @return {String[][]}
```

```
def accounts_merge(accounts)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[][] $accounts
* @return String[][]
*/
function accountsMerge($accounts) {

}
}
```

Dart Solution:

```
class Solution {
List<List<String>> accountsMerge(List<List<String>> accounts) {

}
}
```

Scala Solution:

```
object Solution {
def accountsMerge(accounts: List[List[String]]): List[List[String]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec accounts_merge(accounts :: [[String.t]]) :: [[String.t]]
def accounts_merge(accounts) do

end
end
```

Erlang Solution:

```
-spec accounts_merge(Accounts :: [[unicode:unicode_binary()]]) ->
[[unicode:unicode_binary()]].
accounts_merge(Accounts) ->
.
```

Racket Solution:

```
(define/contract (accounts-merge accounts)
(-> (listof (listof string?)) (listof (listof string?)))
)
```