Problem 929: Unique Email Addresses

Problem Information

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

Problem Description

Every

valid email

consists of a

local name

and a

domain name

, separated by the

'@'

sign. Besides lowercase letters, the email may contain one or more

'.'

or

'+'

.

For example, in

"alice@leetcode.com"

,

"alice"

is the

local name

, and

"leetcode.com"

is the

domain name

.

If you add periods

'.'

between some characters in the

local name

part of an email address, mail sent there will be forwarded to the same address without dots in the local name. Note that this rule

does not apply

to

domain names

.

For example,

"alice.z@leetcode.com"

and

"alicez@leetcode.com"

forward to the same email address.

If you add a plus

'+'

in the

local name

, everything after the first plus sign

will be ignored

. This allows certain emails to be filtered. Note that this rule

does not apply

to

domain names

.

For example,

"m.y+name@email.com"

will be forwarded to

"my@email.com"

.

It is possible to use both of these rules at the same time.

Given an array of strings

emails

where we send one email to each

emails[i]

, return

the number of different addresses that actually receive mails

.

Example 1:

Input:

emails = ["test.email+alex@leetcode.com","test.e.mail+bob.cathy@leetcode.com","testemail+david@lee.tcode.com"]

Output:

2

Explanation:

"testemail@leetcode.com" and "testemail@lee.tcode.com" actually receive mails.

Example 2:

Input:

emails = ["a@leetcode.com","b@leetcode.com","c@leetcode.com"]

Output:

3

Constraints:

1 <= emails.length <= 100

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

emails[i]

consist of lowercase English letters,

'+'

,

'.'

and

'@'

.

Each

emails[i]

contains exactly one

'@'

character.

All local and domain names are non-empty.

Local names do not start with a

'+'

character.

Domain names end with the

".com"

suffix.

Domain names must contain at least one character before

".com"

suffix.

Code Snippets

C++:

```
class Solution {
public:
int numUniqueEmails(vector<string>& emails) {

}
};
```

Java:

```
class Solution {
public int numUniqueEmails(String[] emails) {

}
}
```

Python3:

```
class Solution:
def numUniqueEmails(self, emails: List[str]) -> int:
```

Python:

```
class Solution(object):
def numUniqueEmails(self, emails):
"""
:type emails: List[str]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string[]} emails
* @return {number}
*/
var numUniqueEmails = function(emails) {

};
```

TypeScript:

```
function numUniqueEmails(emails: string[]): number {

};
```

C#:

```
public class Solution {
public int NumUniqueEmails(string[] emails) {

}
}
```

C:

```
int numUniqueEmails(char** emails, int emailsSize) {

}
```

Go:

```
func numUniqueEmails(emails []string) int {

}
```

Kotlin:

```
class Solution {
fun numUniqueEmails(emails: Array<String>): Int {

}
}
```

Swift:

```
class Solution {
func numUniqueEmails(_ emails: [String]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn num_unique_emails(emails: Vec<String>) -> i32 {

}
}
```

Ruby:

```
# @param {String[]} emails
# @return {Integer}
def num_unique_emails(emails)

end
```

PHP:

```
class Solution {

/**
```

```
* @param String[] $emails
* @return Integer
*/
function numUniqueEmails($emails) {

}
}
```

Dart:

```
class Solution {
int numUniqueEmails(List<String> emails) {

}
}
```

Scala:

```
object Solution {
def numUniqueEmails(emails: Array[String]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec num_unique_emails(emails :: [String.t]) :: integer
def num_unique_emails(emails) do

end
end
```

Erlang:

```
-spec num_unique_emails(Emails :: [unicode:unicode_binary()]) -> integer().
num_unique_emails(Emails) ->
.
```

Racket:

```
(define/contract (num-unique-emails emails)
(-> (listof string?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Unique Email Addresses
* 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:
int numUniqueEmails(vector<string>& emails) {

}
};
```

Java Solution:

```
/**
* Problem: Unique Email Addresses
* 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 int numUniqueEmails(String[] emails) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Unique Email Addresses
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 numUniqueEmails(self, emails: List[str]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def numUniqueEmails(self, emails):
"""
:type emails: List[str]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Unique Email Addresses
* 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[]} emails
* @return {number}
*/
var numUniqueEmails = function(emails) {

};
```

TypeScript Solution:

```
/**
* Problem: Unique Email Addresses
* 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 numUniqueEmails(emails: string[]): number {

};
```

C# Solution:

```
/*
* Problem: Unique Email Addresses
* 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 int NumUniqueEmails(string[] emails) {

}
}
```

C Solution:

```
/*
* Problem: Unique Email Addresses
* 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
*/

int numUniqueEmails(char** emails, int emailsSize) {

}
```

Go Solution:

```
// Problem: Unique Email Addresses
// 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 numUniqueEmails(emails []string) int {

}
```

Kotlin Solution:

```
class Solution {
fun numUniqueEmails(emails: Array<String>): Int {

}
}
```

Swift Solution:

```
class Solution {
func numUniqueEmails(_ emails: [String]) -> Int {
```

```
}
}
```

Rust Solution:

```
// Problem: Unique Email Addresses
// 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 num_unique_emails(emails: Vec<String>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String[]} emails
# @return {Integer}
def num_unique_emails(emails)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $emails
* @return Integer
*/
function numUniqueEmails($emails) {

}
}
```

Dart Solution:

```
class Solution {
int numUniqueEmails(List<String> emails) {

}
}
```

Scala Solution:

```
object Solution {
def numUniqueEmails(emails: Array[String]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec num_unique_emails(emails :: [String.t]) :: integer
def num_unique_emails(emails) do

end
end
```

Erlang Solution:

```
-spec num_unique_emails(Emails :: [unicode:unicode_binary()]) -> integer().
num_unique_emails(Emails) ->
.
```

Racket Solution:

```
(define/contract (num-unique-emails emails)
(-> (listof string?) exact-integer?)
)
```