929. Unique Email Addresses

Easy Array Hash Table String   Leetcode Link

Problem Description

The problem relates to the processing of email addresses in order to identify unique recipients. An email address consists of two parts: a local name and a domain name, separated by '@'. The local part can include periods '.' and plus signs '+'. The periods within the local part have no effect on the email delivery, meaning 'alice.z' and 'alicez' in local parts are effectively the same. Additionally, everything after a plus sign in the local part of the email is disregarded. The domain name, however, is unaffected by these rules.

The goal is to count how many unique email addresses are actually being used for email delivery after applying the mentioned rules. For example, "alice.z@leetcode.com" and "alicez@leetcode.com" count as one unique email address since the period in the local part is ignored.

Given an array of email strings, the task is to return the count of distinct addresses that will actually receive emails, after normalizing them according to the rules.

Intuition

To solve this problem, we need to normalize each email address by removing extra periods and any characters following a plus sign in the local name.

Here's how we can approach it:

1. We initialize an empty set to store the unique email addresses.
2. We iterate over each email in the list.
3. For each email, we split it into local and domain parts using the '@' symbol.
4. We remove all periods from the local part.
5. If there is a plus sign in the local part, we truncate the local part up to the plus sign, ignoring everything after it.
6. We recombine the normalized local part with the domain part, and add the resulting email address to the set.
7. Since a set automatically ensures that only unique items are stored, after processing all emails, the size of the set will be equal to the number of unique email addresses that actually receive mails.
8. We return this count as the final answer.

By using a set, we avoid having to manually check for duplicates, which simplifies the logic and improves efficiency.

Solution Approach

The implementation in Python uses a set to store unique email addresses that are normalized according to the problem's rules.

1. **Set Data Structure**: A set is chosen because it only stores unique values, which automatically helps us to keep track of unique emails without any duplicates.

2. **String Manipulation**: The algorithm uses basic string operations such as `split`, `replace`, and slicing to manipulate the local names as per the rules provided.

3. **Processing Emails**:

   - **Splitting**: Each email is split into a `local` and `domain` part using `email.split('@')`. This separates the local name from the domain name.

   - **Removing Periods**: All periods are removed from the local part with `local.replace('.', '')`, since periods do not change the destination of the email.

   - **Handling Plus Sign**: If a plus sign is present in the local part, it finds the index of the first plus sign using `local.find('+')`. The local part is then truncated up to that index (not including the plus and everything after it) with `local[:i]`, where `i` is the index of the plus sign.

   - **Reassembly**: The cleaned local part is concatenated with the domain part using `'@'` to form the normalized email.

   - **Adding to Set**: The resulting email address is added to the set. Python takes care of checking for uniqueness.

4. **Return Unique Count**: The length of the set, `len(s)`, reflects the number of unique normalized emails, which is the desired output.

This algorithm effectively streamlines the email normalization while respecting the defined rules and ensures that we are able to calculate the unique recipient emails in an efficient manner.

Example Walkthrough

Let's take a small example to illustrate the solution approach. Suppose we have the following array of email strings:

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

We want to find out how many unique email addresses there are after normalizing these emails according to the rules described.

1. We declare a set to keep track of unique normalized emails: `unique_emails = set()`.

2. Start iterating over each email:

   - For the first email, `test.email+alex@leetcode.com`:

     - Split into `local = "test.email+alex"` and `domain = "leetcode.com"`.
     - Remove periods in the local part: `local = local.replace('.', '')` resulting in `local = "testemail+alex"`.
     - Look for a plus sign and truncate everything after it: `local = local[:local.find('+')]` resulting in `local = "testemail"`.
     - Combine the modified local part with the domain part: `normalized_email = "testemail@leetcode.com"`.
     - Add the normalized email to the set: `unique_emails.add(normalized_email)`.
   - Repeat the steps for the other emails:

     - Second email `test.e.mail+bob.cathy@leetcode.com` becomes `testemail@leetcode.com` after normalization.
     - Third email `testemail+david@lee.tcode.com` remains as `testemail@lee.tcode.com` because the domain is different.
3. After processing all emails, we have the set `unique_emails = {"testemail@leetcode.com", "testemail@lee.tcode.com"}`.

4. The length of the set is 2, meaning there are two unique email addresses.

So the function would return `2` for this given example, illustrating that after normalizing the given email addresses, there are two unique emails that would receive messages.

Python Solution

```python
class Solution:
    def numUniqueEmails(self, emails: List[str]) -> int:
        # Initialize a set to keep track of unique emails
        unique_emails = set()

        # Iterate over each email in the list
        for email in emails:
            # Split the email into local and domain parts
            local, domain = email.split('@')
            # Remove all occurrences of '.'
            local = local.replace('.', '')
            # Find the index of '+' to ignore everything after it in the local part
            plus_index = local.find('+')
            # If '+' is present, keep only the part before it
            if plus_index != -1:
                local = local[:plus_index]
            # Combine the processed local part with the domain and add to the set
            unique_emails.add(local + '@' + domain)

        # The length of the set represents the count of unique emails
        return len(unique_emails)

```

Java Solution

```java
class Solution {

    public int numUniqueEmails(String[] emails) {
        // Create a hash set to store unique emails
        Set<String> uniqueEmails = new HashSet<>();

        // Iterate over each email in the provided array
        for (String email : emails) {
            // Split the email into local and domain parts
            String[] parts = email.split("@");
            String localPart = parts[0];
            String domainPart = parts[1];

            // Remove all periods from the local part
            localPart = localPart.replace(".", "");

            // Check for a '+' character and cut off any substring after it in the local part
            int plusIndex = localPart.indexOf('+');
            if (plusIndex != -1) {
                localPart = localPart.substring(0, plusIndex);
            }

            // Combine the processed local part with the domain part
            String cleanEmail = localPart + "@" + domainPart;

            // Add the clean email to the set (duplicates are ignored)
            uniqueEmails.add(cleanEmail);
        }

        // Return the number of unique emails found, which is the size of the set
        return uniqueEmails.size();
    }
}

```

C++ Solution

```cpp
#include <string>
#include <vector>
#include <unordered_set>
using namespace std;

class Solution {
public:
    int numUniqueEmails(vector<string>& emails) {
        // Use an unordered set to store unique emails.
        unordered_set<string> uniqueEmails;

        // Process each email in the input vector.
        for (auto& email : emails) {
            // Find the position of '@' which separates local and domain parts.
            size_t atPosition = email.find('@');

            // Extract the local part and domain part based on the position of '@'.
            string localPart = email.substr(0, atPosition);
            string domainPart = email.substr(atPosition);

            // Find the position of '+' which indicates the end of the relevant local part.
            size_t plusPosition = localPart.find('+');
            if (plusPosition != string::npos) {
                // Keep only the relevant part of the local part, before '+'.
                localPart = localPart.substr(0, plusPosition);
            }

            // Remove all the occurrences of '.' from the local part.
            size_t dotPosition = localPart.find('.');
            while (dotPosition != string::npos) {
                localPart.erase(dotPosition, 1);
                dotPosition = localPart.find('.');
            }

            // Combine the sanitized local part with the domain part and insert into set.
            uniqueEmails.insert(localPart + domainPart);
        }

        // Return the size of the set, which represents the number of unique emails.
        return uniqueEmails.size();
    }
};

```

Typescript Solution

```typescript
function numUniqueEmails(emails: string[]): number {
    // Create a new Set to store the unique emails
    return new Set(
        emails
            .map(email => {
                // For each email, split it into two parts [local part, domain part]
                const parts = email.split('@');
                const localPart = parts[0];
                const domain = parts[1];
                // Remove any occurrences of '.' and anything following '+'
                const sanitizedLocalPart = localPart.replace(/\+.*|\./g, '');
                // Concatenate the sanitized local part with domain
                return sanitizedLocalPart + '@' + domain;
            }),
    ).size; // Return the size of the Set, which is the count of unique emails
}

// The function numUniqueEmails takes an array of email strings,
// sanitizes them by removing the local part's dots and ignoring
// any characters after a '+', and then counts unique email addresses.

```

Time and Space Complexity

The above Python code processes a list of email addresses to find the number of unique email addresses after normalizing them based on the specific rules provided.

Time Complexity:

Let `n` be the number of email addresses and `k` be the average length of these email addresses.

- Splitting the email into local and domain strings takes `O(k)` time for each email.
- Replacing dots in the local part is `O(k)` as it requires traversing the local part.
- The find operation for '+' is also in `O(k)` because finding a character in a string takes linear time in the worst case.
- Slicing the local part up to '+' is at worst `O(k)` since in the worst case the '+' could be at the end of the local part.
- Adding the processed email to set `s` can be considered `O(1)` on average due to hash set operation, but the worst case (though rare due to proper hashing) for insertion is `O(k)` when there's a hash collision and it needs to traverse a linked list in the set bucket.

Therefore, each email address takes `O(k)` time to process, and hence the total time for processing all email addresses is `O(n*k)`.

Space Complexity:

- The set `s` will contain at most `n` elements if all normalized email addresses are unique. Since each email address is `O(k)`, the total space taken by the set is `O(n*k)` in the worst case.
- Auxiliary space is used for storing the local and domain parts of each email address along with storing the sliced local part when '+' is found. These do not depend on the number of emails, just on the length of the emails, so they are `O(k)` each.

Hence, the overall space complexity is `O(n*k)` if the space taken by the input list is not considered. If considering the space for input, the total space complexity remains `O(n*k)` since this is the dominating term.