811. Subdomain Visit Count

String)

Counting

Hash Table

Problem Description

<u>Array</u>

Medium

subdomain implies that all of its parent domains were visited the same number of times. A "count-paired domain" will be specified in the format rep d1.d2.d3, where rep represents the number of times the domain d1.d2.d3 was visited.

In this problem, we're working with domain visit counts. A domain may have multiple subdomains, and the count given for a

The task is to analyze an array of these count-paired domains, and output an array of count-paired domains that represent the counts of all possible subdomains. Let's take the example given 9001 discuss leetcode com. The count 9001 applies not only to

discuss.leetcode.com, but also to its parent domains leetcode.com and com. Therefore, the output should reflect the count for all of these domains.

The intuition behind the solution is to break down the given domains into all possible subdomains while keeping track of their visit counts. We'll iterate through each count-paired domain given in the array and parse it to extract the visit count and the domain

Intuition

itself. Once we have those, we'll increment the count for that domain and all of its subdomains. To achieve this, we can use a data structure like a Python Counter to map domains to their visit counts. As we encounter each domain and its subdomains, we add the visit count to the total already stored in the Counter for that domain or subdomain.

Let's break this down further: 1. We traverse the given list of count-paired domains.

3. We iterate through the characters in the full domain and whenever we encounter a dot('.') or a space (' '), we recognize a subdomain.

- 4. We update the Counter with the visit count for the domain or subdomain found.
- 5. After processing all count-paired domains, we'll have a Counter that contains all the domains and their respective total visit counts. 6. The final step is to format the output as requested, which would be to convert the domain and counts back into the rep d1.d2.d3 format for the

Here's the step-by-step approach of the algorithm using the Counter:

2. For each count-paired domain, we identify the visit count and the full domain.

- result array.
- **Solution Approach**
- The solution uses the Counter data structure from Python's collections module, which is a subclass of dict. It's specifically designed to count objects and is an ideal choice for this problem since we need to count visits for each domain and its subdomains.

1. Initialize a Counter instance (named cnt in the code) to keep track of the count of domains.

paired domain".

Example Walkthrough

2. Iterate over the list of cpdomains: For each domain string, find the first space character which delimits the visit count and the domain. Extract the visit count and convert it to an integer (v).

■ When a dot '.' or space ' ' is encountered, it marks the beginning of a (sub)domain. Update the Counter by adding the visit count v to the current (sub)domain. ■ This is done using the notation cnt[s[i + 1 :]] += v, which incrementally adds the count to the existing value for the subdomain

Iterate through each character of the domain:

- string s[i + 1 :]. The slicing s[i + 1 :] creates substrings representing the current domain and its subsequent subdomains each time a dot or space is encountered.
- 3. After the loop, cnt contains all the (sub)domains and their respective total counts. 4. Finally, the solution prepares the output by formatting the Counter items into a list of strings (f'{v} {s}'), each corresponding to a "count-
- dictionaries in Python have an average-case time complexity of 0(1) for updates. Lastly, the Counter nicely organizes our counts and domains, making the final output generation step simple.

number of domains and subdomains. Second, updating the Counter takes constant time 0(1) for each subdomain because

The algorithm is efficient for a couple of reasons. First, iterating through the cpdomains array takes O(N) time, where N is the total

Let's consider a simple example with the following input array: ["900 google.mail.com", "50 yahoo.com", "1 intel.mail.com", "5 wiki.org"]. We need to output the counts for all domains and subdomains. Follow these steps: 1. Initialize a Counter instance to keep track of domain counts. 2. Start with the first domain string "900 google.mail.com". Split the string at the space to get the count 900 and the domain google.mail.com.

• Encounter the first dot, yielding the subdomain mail.com. Add "mail.com": 900 to the Counter. Encounter the second dot, yielding the subdomain com. Add "com": 900 to the Counter. 4. Move to the second domain string "50 yahoo.com":

3. Begin parsing the domain:

5. Process "1 intel.mail.com" similarly, resulting in:

"intel.mail.com": 1 added to the Counter.

• "com": 1 is added to existing com count, now com has 951.

 Split and obtain count 50 and domain yahoo.com. Add "yahoo.com": 50 to the Counter. Add "com": 50 to the existing count of com in the Counter, now com has a total of 950.

6. Lastly, "5 wiki.org": Split into count 5 and domain wiki.org. Add "wiki.org": 5 to the Counter.

Add "org": 5 to the Counter, as it's the first time we encounter org.

def subdomainVisits(self, cpdomains: List[str]) -> List[str]:

visit_count = int(cpdomain.split(' ')[0])

Return the formatted list of domain visit counts

public List<String> subdomainVisits(String[] cpdomains) {

int spaceIndex = domainInfo.indexOf(" ");

Map<String, Integer> visitCounts = new HashMap<>();

// Iterate over each combined domain visit count entry.

int visitCount = stoi(cpdomain.substr(0, spaceIndex));

for (size_t i = spaceIndex; i < cpdomain.size(); ++i) {</pre>

if (cpdomain[i] == ' ' || cpdomain[i] == '.') {

// Check for a space or a dot character to identify subdomains

domainVisitCounts[cpdomain.substr(i + 1)] += visitCount;

// Prepare the answer vector to hold the results in the required format

// Concatenate the count and the domain with a space separator

// Import statements are not required in TypeScript for basic types like Array and String

// A function to process an array of cpdomains to count the visits and split them by subdomains

// Add the visit count to the corresponding subdomain in the hashmap

// Loop through the domain portion of the string

// Traverse the hashmap and format the output accordingly

results.push_back(to_string(count) + " " + domain);

// Return the formatted list of visit counts per subdomain

// Loop through each combined domain visit count and domain string

// Convert the count portion of the string to an integer

// Find the space character to separate the count from the domain

const visitCount = parseInt(cpdomain.substring(0, spaceIndex), 10);

// Add the visit count to the corresponding subdomain in the map

// Find the next dot character to move to the higher-level domain

domainVisitCounts[domain] = (domainVisitCounts[domain] || 0) + visitCount;

// Need to add 1 to the space index to skip the space character

// Process all subdomains, including the full domain itself

// Prepare the answer array to hold the results in the required format

// Concatenate the count and the domain with a space separator

function subdomainVisits(cpdomains: string[]): string[] {

const spaceIndex = cpdomain.indexOf(' ');

// Get the full domain string from the cpdomain

let domain = cpdomain.substring(spaceIndex + 1);

const dotIndex = domain.index0f('.');

domain = domain.substring(dotIndex + 1);

// Traverse the map and format the output accordingly

const count = domainVisitCounts[domain];

for (const domain in domainVisitCounts) {

results.push(`\${count} \${domain}`);

// Create a map to store the counts of domain visits

const domainVisitCounts: Record<string, number> = {};

for (const auto& [domain, count] : domainVisitCounts) {

vector<string> results;

cpdomains.forEach(cpdomain => {

while (domain) {

const results: string[] = [];

return results;

};

TypeScript

// Create a map to store subdomain visit counts.

for (String domainInfo : cpdomains) {

Split the domain on '.' to find all subdomains

domain = cpdomain.split(' ')[1]

subdomains = domain.split('.')

for i in range(len(subdomains)):

Create a Counter object to keep track of the domain visit counts

For each subdomain, build the subdomain string and update the counter

Join the subdomain parts to form the subdomain to count visits

result = [f"{count} {domain}" for domain, count in domain_visits_count.items()]

// Find the index of the first space to separate number of visits from the domain.

// Parse the number of visits using the substring before the space.

"mail.com": 1 is added to existing mail.com count, now mail.com has 901.

Since google.mail.com has no spaces, add "google.mail.com": 900 to the Counter.

• "900 google.mail.com" • "901 mail.com" • "951 com"

The Counter now has the correct total counts for all domains and subdomains. Here's what the final Counter looks like:

The next step is to format this output according to the rep d1.d2.d3 format. Iterate through the Counter and construct the result

• {"google.mail.com": 900, "mail.com": 901, "com": 951, "yahoo.com": 50, "wiki.org": 5, "org": 5, "intel.mail.com": 1}

• "1 intel.mail.com" The final output is the array: ["900 google.mail.com", "901 mail.com", "951 com", "50 yahoo.com", "5 wiki.org", "5 org",

"1 intel.mail.com"]. This array represents the visit counts for each domain and subdomain.

Python

class Solution:

Solution Implementation

from collections import Counter

strings:

• "50 yahoo.com"

• "5 wiki.org"

• "5 org"

domain_visits_count = Counter() # Iterate through the list of domain visit counts for cpdomain in cpdomains: # Extract the visit count and the domain from the current string

```
subdomain = '.'.join(subdomains[i:])
        domain_visits_count[subdomain] += visit_count
# Format the results as specified in the problem statement
```

return result

Java

class Solution {

```
int visitCount = Integer.parseInt(domainInfo.substring(0, spaceIndex));
            // Start iterating characters at the space to check for subdomains.
            for (int i = spaceIndex; i < domainInfo.length(); ++i) {</pre>
                // Check for boundaries of subdomains.
                if (domainInfo.charAt(i) == ' ' || domainInfo.charAt(i) == '.') {
                    // Extract the subdomain using substring.
                    String subdomain = domainInfo.substring(i + 1);
                    // Update the visit count for the current subdomain.
                    visitCounts.put(subdomain, visitCounts.getOrDefault(subdomain, 0) + visitCount);
       // Prepare the answer list containing formatted results.
       List<String> results = new ArrayList<>();
       // Iterate through the entry set of the visitCounts map.
        for (Map.Entry<String, Integer> entry : visitCounts.entrySet()) {
            // Format the entry as "visit_count domain" and add it to the results list.
            results.add(entry.getValue() + " " + entry.getKey());
        return results;
#include <vector>
#include <string>
#include <unordered_map>
class Solution {
public:
    // Function to process a vector of cpdomains to count the visits and split them by subdomains
    vector<string> subdomainVisits(vector<string>& cpdomains) {
       // Create a hashmap to store the counts of domain visits
       unordered_map<string, int> domainVisitCounts;
       // Loop through each combined domain visit count and domain string
        for (const auto& cpdomain : cpdomains) {
           // Find the space character to separate the count from the domain
            size_t spaceIndex = cpdomain.find(' ');
            // Convert the count portion of the string to an integer
```

// If no dot is found, or it is the last position, we stop if (dotIndex < 0) break;</pre> // Update the domain to be the string after the dot

});

```
// Return the formatted list of visit counts per subdomain
      return results;
from collections import Counter
class Solution:
   def subdomainVisits(self, cpdomains: List[str]) -> List[str]:
       # Create a Counter object to keep track of the domain visit counts
       domain_visits_count = Counter()
       # Iterate through the list of domain visit counts
        for cpdomain in cpdomains:
            # Extract the visit count and the domain from the current string
            visit_count = int(cpdomain.split(' ')[0])
            domain = cpdomain.split(' ')[1]
```

For each subdomain, build the subdomain string and update the counter

Join the subdomain parts to form the subdomain to count visits

result = [f"{count} {domain}" for domain, count in domain_visits_count.items()]

The time complexity of the code is determined by several factors: 1. Iterating through each domain in cpdomains, which gives us O(n) where n is the number of domains.

Split the domain on '.' to find all subdomains

subdomain = '.'.join(subdomains[i:])

Return the formatted list of domain visit counts

domain_visits_count[subdomain] += visit_count

Format the results as specified in the problem statement

subdomains = domain.split('.')

return result

Time Complexity

Space Complexity

Time and Space Complexity

for i in range(len(subdomains)):

3. The inner loop enumerates over each character in the domain string s which is 0(m) in the worst case. 4. The slicing of strings s[i + 1 :] is done in each loop iteration, which, in Python, is also 0(m).

For space complexity, we consider the memory allocations:

The second to the fourth steps occur within the loop of the first step, so the total time complexity is 0(n * m) since we have to assume that each domain could be of variable length, and we are bound by the length of both the domains and the list itself.

2. Inside the loop, the index() method is called to find the first space, which operates in O(m), where m is the length of the string s.

1. The Counter object cnt storage grows with the number of distinct subdomains found, in the worst case this may contain all the subdomains

which would be 0(d) where d is the total number of distinct subdomains. 2. The created list for the final output will have as many elements as there are entries in cnt which is also 0(d).

The space complexity, therefore, is O(d), which is determined by the number of unique subdomains.