# Problem 811: Subdomain Visit Count

## Problem Information

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

## Problem Description

A website domain

"discuss.leetcode.com"

consists of various subdomains. At the top level, we have

"com"

, at the next level, we have

"leetcode.com"

and at the lowest level,

"discuss.leetcode.com"

. When we visit a domain like

"discuss.leetcode.com"

, we will also visit the parent domains

"leetcode.com"

and

"com"

implicitly.

A

count-paired domain

is a domain that has one of the two formats

"rep d1.d2.d3"

or

"rep d1.d2"

where

rep

is the number of visits to the domain and

d1.d2.d3

is the domain itself.

For example,

"9001 discuss.leetcode.com"

is a

count-paired domain

that indicates that

discuss.leetcode.com

was visited

9001

times.

Given an array of

count-paired domains

cpdomains

, return

an array of the

count-paired domains

of each subdomain in the input

. You may return the answer in

any order

.

Example 1:

Input:

cpdomains = ["9001 discuss.leetcode.com"]

Output:

["9001 leetcode.com","9001 discuss.leetcode.com","9001 com"]

Explanation:

We only have one website domain: "discuss.leetcode.com". As discussed above, the subdomain "leetcode.com" and "com" will also be visited. So they will all be visited 9001 times.

Example 2:

Input:

cpdomains = ["900 google.mail.com", "50 yahoo.com", "1 intel.mail.com", "5 wiki.org"]

Output:

["901 mail.com","50 yahoo.com","900 google.mail.com","5 wiki.org","5 org","1 intel.mail.com","951 com"]

Explanation:

We will visit "google.mail.com" 900 times, "yahoo.com" 50 times, "intel.mail.com" once and "wiki.org" 5 times. For the subdomains, we will visit "mail.com" 900 + 1 = 901 times, "com" 900 + 50 + 1 = 951 times, and "org" 5 times.

Constraints:

1 <= cpdomain.length <= 100

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

cpdomain[i]

follows either the

"rep

i

d1

i

.d2

i

.d3

i

"

format or the

"rep

i

d1

i

.d2

i

"

format.

rep

i

is an integer in the range

[1, 10

4

]

.

d1

i

,

d2

i

, and

d3

i

consist of lowercase English letters.

## Code Snippets

**C++:**

```
class Solution {
public:
vector<string> subdomainVisits(vector<string>& cpdomains) {

}
};
```

**Java:**

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

}
}
```

**Python3:**

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

**Python:**

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

**JavaScript:**

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

};
```

**TypeScript:**

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

};
```

**C#:**

```
public class Solution {
public IList<string> SubdomainVisits(string[] cpdomains) {

}
}
```

**C:**

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** subdomainVisits(char** cpdomains, int cpdomainsSize, int* returnSize)
{

}
```

**Go:**

```
func subdomainVisits(cpdomains []string) []string {

}
```

**Kotlin:**

```
class Solution {
fun subdomainVisits(cpdomains: Array<String>): List<String> {

}
}
```

**Swift:**

```
class Solution {
func subdomainVisits(_ cpdomains: [String]) -> [String] {

}
}
```

**Rust:**

```
impl Solution {
pub fn subdomain_visits(cpdomains: Vec<String>) -> Vec<String> {

}
}
```

**Ruby:**

```
# @param {String[]} cpdomains
# @return {String[]}
def subdomain_visits(cpdomains)

end
```

**PHP:**

```
class Solution {

/**
```

```
* @param String[] $cpdomains
* @return String[]
*/
function subdomainVisits($cpdomains) {

}
}
```

**Dart:**

```
class Solution {
List<String> subdomainVisits(List<String> cpdomains) {

}
}
```

**Scala:**

```
object Solution {
def subdomainVisits(cpdomains: Array[String]): List[String] = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec subdomain_visits(cpdomains :: [String.t]) :: [String.t]
def subdomain_visits(cpdomains) do

end
end
```

**Erlang:**

```
-spec subdomain_visits(Cpdomains :: [unicode:unicode_binary()]) ->
[unicode:unicode_binary()].
subdomain_visits(Cpdomains) ->
.
```

**Racket:**

```
(define/contract (subdomain-visits cpdomains)
(-> (listof string?) (listof string?))
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Subdomain Visit Count
* Difficulty: Medium
* 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:
vector<string> subdomainVisits(vector<string>& cpdomains) {

}
};
```

### Java Solution:

```
/**
* Problem: Subdomain Visit Count
* Difficulty: Medium
* 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 List<String> subdomainVisits(String[] cpdomains) {

}
```

```
}
```

### Python3 Solution:

```
"""
Problem: Subdomain Visit Count
Difficulty: Medium
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 subdomainVisits(self, cpdomains: List[str]) -> List[str]:
# TODO: Implement optimized solution
pass
```

### Python Solution:

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

### JavaScript Solution:

```
/**
* Problem: Subdomain Visit Count
* Difficulty: Medium
* 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[]} cpdomains
* @return {string[]}
*/
var subdomainVisits = function(cpdomains) {

};
```

### TypeScript Solution:

```
/**
* Problem: Subdomain Visit Count
* Difficulty: Medium
* 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 subdomainVisits(cpdomains: string[]): string[] {

};
```

### C# Solution:

```
/*
* Problem: Subdomain Visit Count
* Difficulty: Medium
* 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 IList<string> SubdomainVisits(string[] cpdomains) {

}
}
```

### C Solution:

```
/*
* Problem: Subdomain Visit Count
* Difficulty: Medium
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** subdomainVisits(char** cpdomains, int cpdomainsSize, int* returnSize)
{

}
```

### Go Solution:

```
// Problem: Subdomain Visit Count
// Difficulty: Medium
// 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 subdomainVisits(cpdomains []string) []string {

}
```

### Kotlin Solution:

```
class Solution {
fun subdomainVisits(cpdomains: Array<String>): List<String> {

}
}
```

### Swift Solution:

```
class Solution {
func subdomainVisits(_ cpdomains: [String]) -> [String] {

}
}
```

### Rust Solution:

```
// Problem: Subdomain Visit Count
// Difficulty: Medium
// 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 subdomain_visits(cpdomains: Vec<String>) -> Vec<String> {

}
}
```

### Ruby Solution:

```
# @param {String[]} cpdomains
# @return {String[]}
def subdomain_visits(cpdomains)

end
```

### PHP Solution:

```
class Solution {

/**
* @param String[] $cpdomains
* @return String[]
*/
function subdomainVisits($cpdomains) {
```

```
}
}
```

### Dart Solution:

```
class Solution {
List<String> subdomainVisits(List<String> cpdomains) {

}
}
```

### Scala Solution:

```
object Solution {
def subdomainVisits(cpdomains: Array[String]): List[String] = {

}
}
```

### Elixir Solution:

```
defmodule Solution do
@spec subdomain_visits(cpdomains :: [String.t]) :: [String.t]
def subdomain_visits(cpdomains) do

end
end
```

### Erlang Solution:

```
-spec subdomain_visits(Cpdomains :: [unicode:unicode_binary()]) ->
[unicode:unicode_binary()].
subdomain_visits(Cpdomains) ->
.
```

### Racket Solution:

```
(define/contract (subdomain-visits cpdomains)
(-> (listof string?) (listof string?))
)
```