Problem 1152: Analyze User Website Visit Pattern

Problem Information

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

Problem Description

You are given two string arrays

username

and

website

and an integer array

timestamp

. All the given arrays are of the same length and the tuple

[username[i], website[i], timestamp[i]]

indicates that the user

username[i]

visited the website

website[i]

at time

timestamp[i]

.

A

pattern

is a list of three websites (not necessarily distinct).

For example,

["home", "away", "love"]

,

["leetcode", "love", "leetcode"]

, and

["luffy", "luffy", "luffy"]

are all patterns.

The

score

of a

pattern

is the number of users that visited all the websites in the pattern in the same order they appeared in the pattern.

For example, if the pattern is

["home", "away", "love"]

, the score is the number of users

x

such that

x

visited

"home"

then visited

"away"

and visited

"love"

after that.

Similarly, if the pattern is

["leetcode", "love", "leetcode"]

, the score is the number of users

x

such that

x

visited

"leetcode"

then visited

"love"

and visited

"leetcode"

one more time

after that.

Also, if the pattern is

["luffy", "luffy", "luffy"]

, the score is the number of users

x

such that

x

visited

"luffy"

three different times at different timestamps.

Return the

pattern

with the largest

score

. If there is more than one pattern with the same largest score, return the lexicographically smallest such pattern.

Note that the websites in a pattern

do not

need to be visited

contiguously

, they only need to be visited in the order they appeared in the pattern.

Example 1:

Input:

username = ["joe","joe","joe","james","james","james","james","mary","mary","mary"], timestamp = [1,2,3,4,5,6,7,8,9,10], website = ["home","about","career","home","cart","maps","home","home","about","career"]

Output:

["home","about","career"]

Explanation:

The tuples in this example are: ["joe","home",1],["joe","about",2],["joe","career",3],["james","home",4],["james","cart",5],["james","maps",6],["james","home",7],["mary","home",8],["mary","about",9], and ["mary","career",10]. The pattern ("home", "about", "career") has score 2 (joe and mary). The pattern ("home", "cart", "maps") has score 1 (james). The pattern ("home", "cart", "home") has score 1 (james). The pattern ("home", "maps", "home") has score 1 (james). The pattern ("cart", "maps", "home") has score 1 (james). The pattern ("home", "home", "home") has score 0 (no user visited home 3 times).

Example 2:

Input:

username = ["ua","ua","ua","ub","ub","ub"], timestamp = [1,2,3,4,5,6], website = ["a","b","a","a","b","c"]

Output:

["a","b","a"]

Constraints:

3 <= username.length <= 50

1 <= username[i].length <= 10

timestamp.length == username.length

1 <= timestamp[i] <= 10

9

website.length == username.length

1 <= website[i].length <= 10

username[i]

and

website[i]

consist of lowercase English letters.

It is guaranteed that there is at least one user who visited at least three websites.

All the tuples

[username[i], timestamp[i], website[i]]

are

unique

.

Code Snippets

C++:

```
class Solution {
public:
vector<string> mostVisitedPattern(vector<string>& username, vector<int>&
timestamp, vector<string>& website) {

}
};
```

Java:

```
class Solution {
public List<String> mostVisitedPattern(String[] username, int[] timestamp,
String[] website) {

}
}
```

Python3:

```
class Solution:
def mostVisitedPattern(self, username: List[str], timestamp: List[int],
website: List[str]) -> List[str]:
```

Python:

```
class Solution(object):
def mostVisitedPattern(self, username, timestamp, website):
"""
:type username: List[str]
:type timestamp: List[int]
:type website: List[str]
:rtype: List[str]
"""
```

JavaScript:

```
/**
* @param {string[]} username
* @param {number[]} timestamp
* @param {string[]} website
* @return {string[]}
*/
var mostVisitedPattern = function(username, timestamp, website) {

};
```

TypeScript:

```
function mostVisitedPattern(username: string[], timestamp: number[], website:
string[]): string[] {

};
```

C#:

```
public class Solution {
public IList<string> MostVisitedPattern(string[] username, int[] timestamp,
string[] website) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** mostVisitedPattern(char** username, int usernameSize, int* timestamp,
int timestampSize, char** website, int websiteSize, int* returnSize) {

}
```

Go:

```
func mostVisitedPattern(username []string, timestamp []int, website []string)
[]string {

}
```

Kotlin:

```
class Solution {
fun mostVisitedPattern(username: Array<String>, timestamp: IntArray, website:
Array<String>): List<String> {

}
}
```

Swift:

```
class Solution {
func mostVisitedPattern(_ username: [String], _ timestamp: [Int], _ website:
[String]) -> [String] {

}
}
```

Rust:

```
impl Solution {
pub fn most_visited_pattern(username: Vec<String>, timestamp: Vec<i32>,
website: Vec<String>) -> Vec<String> {

}
}
```

Ruby:

```
# @param {String[]} username
# @param {Integer[]} timestamp
# @param {String[]} website
# @return {String[]}
def most_visited_pattern(username, timestamp, website)

end
```

PHP:

```
class Solution {

/**
* @param String[] $username
* @param Integer[] $timestamp
* @param String[] $website
* @return String[]
*/
function mostVisitedPattern($username, $timestamp, $website) {

}
}
```

Dart:

```
class Solution {
List<String> mostVisitedPattern(List<String> username, List<int> timestamp,
List<String> website) {

}
}
```

Scala:

```
object Solution {
def mostVisitedPattern(username: Array[String], timestamp: Array[Int],
website: Array[String]): List[String] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec most_visited_pattern(username :: [String.t], timestamp :: [integer],
website :: [String.t]) :: [String.t]
def most_visited_pattern(username, timestamp, website) do

end
end
```

Erlang:

```
-spec most_visited_pattern(Username :: [unicode:unicode_binary()], Timestamp
:: [integer()], Website :: [unicode:unicode_binary()]) ->
[unicode:unicode_binary()].
most_visited_pattern(Username, Timestamp, Website) ->
.
```

Racket:

```
(define/contract (most-visited-pattern username timestamp website)
(-> (listof string?) (listof exact-integer?) (listof string?) (listof
string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Analyze User Website Visit Pattern
* Difficulty: Medium
* Tags: array, string, graph, hash, sort
*
* 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> mostVisitedPattern(vector<string>& username, vector<int>&
timestamp, vector<string>& website) {

}
};
```

Java Solution:

```
/**
* Problem: Analyze User Website Visit Pattern
* Difficulty: Medium
* Tags: array, string, graph, hash, sort
```

```
*
* 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> mostVisitedPattern(String[] username, int[] timestamp,
String[] website) {

}
}
```

Python3 Solution:

```
"""
Problem: Analyze User Website Visit Pattern
Difficulty: Medium
Tags: array, string, graph, hash, sort

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 mostVisitedPattern(self, username: List[str], timestamp: List[int],
website: List[str]) -> List[str]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def mostVisitedPattern(self, username, timestamp, website):
"""
:type username: List[str]
:type timestamp: List[int]
:type website: List[str]
:rtype: List[str]
"""
```

JavaScript Solution:

```
/**
* Problem: Analyze User Website Visit Pattern
* Difficulty: Medium
* Tags: array, string, graph, hash, sort
*
* 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[]} username
* @param {number[]} timestamp
* @param {string[]} website
* @return {string[]}
*/
var mostVisitedPattern = function(username, timestamp, website) {

};
```

TypeScript Solution:

```
/**
* Problem: Analyze User Website Visit Pattern
* Difficulty: Medium
* Tags: array, string, graph, hash, sort
*
* 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 mostVisitedPattern(username: string[], timestamp: number[], website:
string[]): string[] {

};
```

C# Solution:

```
/*
* Problem: Analyze User Website Visit Pattern
* Difficulty: Medium
* Tags: array, string, graph, hash, sort
*
* 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> MostVisitedPattern(string[] username, int[] timestamp,
string[] website) {

}
}
```

C Solution:

```
/*
* Problem: Analyze User Website Visit Pattern
* Difficulty: Medium
* Tags: array, string, graph, hash, sort
*
* 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** mostVisitedPattern(char** username, int usernameSize, int* timestamp,
int timestampSize, char** website, int websiteSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Analyze User Website Visit Pattern
// Difficulty: Medium
// Tags: array, string, graph, hash, sort
```

```
//
// 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 mostVisitedPattern(username []string, timestamp []int, website []string)
[]string {

}
```

Kotlin Solution:

```
class Solution {
fun mostVisitedPattern(username: Array<String>, timestamp: IntArray, website:
Array<String>): List<String> {

}
}
```

Swift Solution:

```
class Solution {
func mostVisitedPattern(_ username: [String], _ timestamp: [Int], _ website:
[String]) -> [String] {

}
}
```

Rust Solution:

```
// Problem: Analyze User Website Visit Pattern
// Difficulty: Medium
// Tags: array, string, graph, hash, sort
//
// 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 most_visited_pattern(username: Vec<String>, timestamp: Vec<i32>,
website: Vec<String>) -> Vec<String> {
}
}
```

Ruby Solution:

```
# @param {String[]} username
# @param {Integer[]} timestamp
# @param {String[]} website
# @return {String[]}
def most_visited_pattern(username, timestamp, website)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $username
* @param Integer[] $timestamp
* @param String[] $website
* @return String[]
*/
function mostVisitedPattern($username, $timestamp, $website) {

}
}
```

Dart Solution:

```
class Solution {
List<String> mostVisitedPattern(List<String> username, List<int> timestamp,
List<String> website) {

}
}
```

Scala Solution:

```
object Solution {
def mostVisitedPattern(username: Array[String], timestamp: Array[Int],
website: Array[String]): List[String] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec most_visited_pattern(username :: [String.t], timestamp :: [integer],
website :: [String.t]) :: [String.t]
def most_visited_pattern(username, timestamp, website) do

end
end
```

Erlang Solution:

```
-spec most_visited_pattern(Username :: [unicode:unicode_binary()], Timestamp
:: [integer()], Website :: [unicode:unicode_binary()]) ->
[unicode:unicode_binary()].
most_visited_pattern(Username, Timestamp, Website) ->
.
```

Racket Solution:

```
(define/contract (most-visited-pattern username timestamp website)
(-> (listof string?) (listof exact-integer?) (listof string?) (listof
string?))
)
```