Problem 1418: Display Table of Food Orders in a Restaurant

Problem Information

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

Problem Description

Given the array

orders

, which represents the orders that customers have done in a restaurant. More specifically

orders[i]=[customerName

i

,tableNumber

i

,foodItem

i

]

where

customerName

i

is the name of the customer,

tableNumber

i

is the table customer sit at, and

foodItem

i

is the item customer orders.

Return the restaurant's “

display table

”

. The “

display table

” is a table whose row entries denote how many of each food item each table ordered. The first column is the table number and the remaining columns correspond to each food item in alphabetical order. The first row should be a header whose first column is “Table”, followed by the names of the food items. Note that the customer names are not part of the table. Additionally, the rows should be sorted in numerically increasing order.

Example 1:

Input:

orders = [["David","3","Ceviche"],["Corina","10","Beef Burrito"],["David","3","Fried Chicken"],["Carla","5","Water"],["Carla","5","Ceviche"],["Rous","3","Ceviche"]]

Output:

[["Table","Beef Burrito","Ceviche","Fried Chicken","Water"],["3","0","2","1","0"],["5","0","1","0","1"],["10","1","0","0","0"]]

Explanation:

The displaying table looks like:

Table,Beef Burrito,Ceviche,Fried Chicken,Water

3 ,0 ,2 ,1 ,0 5 ,0 ,1 ,0 ,1 10 ,1 ,0 ,0 ,0 For the table 3: David orders "Ceviche" and "Fried Chicken", and Rous orders "Ceviche". For the table 5: Carla orders "Water" and "Ceviche". For the table 10: Corina orders "Beef Burrito".

Example 2:

Input:

orders = [["James","12","Fried Chicken"],["Ratesh","12","Fried Chicken"],["Amadeus","12","Fried Chicken"],["Adam","1","Canadian Waffles"],["Brianna","1","Canadian Waffles"]]

Output:

[["Table","Canadian Waffles","Fried Chicken"],["1","2","0"],["12","0","3"]]

Explanation:

For the table 1: Adam and Brianna order "Canadian Waffles". For the table 12: James, Ratesh and Amadeus order "Fried Chicken".

Example 3:

Input:

orders = [["Laura","2","Bean Burrito"],["Jhon","2","Beef Burrito"],["Melissa","2","Soda"]]

Output:

[["Table","Bean Burrito","Beef Burrito","Soda"],["2","1","1","1"]]

Constraints:

1 <= orders.length <= 5 * 10^4

orders[i].length == 3

1 <= customerName

i

.length, foodItem

i

.length <= 20

customerName

i

and

foodItem

i

consist of lowercase and uppercase English letters and the space character.

tableNumber

i

is a valid integer between

1

and

500

.

Code Snippets

C++:

```
class Solution {
public:
vector<vector<string>> displayTable(vector<vector<string>>& orders) {

}
};
```

Java:

```
class Solution {
public List<List<String>> displayTable(List<List<String>> orders) {

}
}
```

Python3:

```
class Solution:
def displayTable(self, orders: List[List[str]]) -> List[List[str]]:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function displayTable(orders: string[][]): string[][] {

};
```

C#:

```
public class Solution {
public IList<IList<string>> DisplayTable(IList<IList<string>> orders) {

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
char *** displayTable(char *** orders, int ordersSize, int* ordersColSize,
int* returnSize, int** returnColumnSizes){

}
```

Go:

```
func displayTable(orders [][]string) [][]string {

}
```

Kotlin:

```
class Solution {
fun displayTable(orders: List<List<String>>): List<List<String>> {

}
}
```

Swift:

```
class Solution {
func displayTable(_ orders: [[String]]) -> [[String]] {

}
}
```

Rust:

```
impl Solution {
pub fn display_table(orders: Vec<Vec<String>>) -> Vec<Vec<String>> {

}
}
```

Ruby:

```
# @param {String[][]} orders
# @return {String[][]}
def display_table(orders)

end
```

PHP:

```
class Solution {

/**
* @param String[][] $orders
* @return String[][]
*/
function displayTable($orders) {

}
}
```

Scala:

```
object Solution {
def displayTable(orders: List[List[String]]): List[List[String]] = {

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Display Table of Food Orders in a Restaurant
* Difficulty: Medium
* Tags: array, string, 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<vector<string>> displayTable(vector<vector<string>>& orders) {

}
};
```

Java Solution:

```
/**
* Problem: Display Table of Food Orders in a Restaurant
* Difficulty: Medium
* Tags: array, string, 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<List<String>> displayTable(List<List<String>> orders) {

}
}
```

Python3 Solution:

```
"""
Problem: Display Table of Food Orders in a Restaurant
Difficulty: Medium
Tags: array, string, 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 displayTable(self, orders: List[List[str]]) -> List[List[str]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Display Table of Food Orders in a Restaurant
* Difficulty: Medium
* Tags: array, string, 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[][]} orders
* @return {string[][]}
*/
var displayTable = function(orders) {

};
```

TypeScript Solution:

```
/**
* Problem: Display Table of Food Orders in a Restaurant
* Difficulty: Medium
* Tags: array, string, 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 displayTable(orders: string[][]): string[][] {

};
```

C# Solution:

```
/*
* Problem: Display Table of Food Orders in a Restaurant
* Difficulty: Medium
* Tags: array, string, 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<IList<string>> DisplayTable(IList<IList<string>> orders) {
}
}
```

C Solution:

```
/*
* Problem: Display Table of Food Orders in a Restaurant
* Difficulty: Medium
* Tags: array, string, 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
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
char *** displayTable(char *** orders, int ordersSize, int* ordersColSize,
int* returnSize, int** returnColumnSizes){

}
```

Go Solution:

```
// Problem: Display Table of Food Orders in a Restaurant
// Difficulty: Medium
// Tags: array, string, 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 displayTable(orders [][]string) [][]string {

}
```

Kotlin Solution:

```
class Solution {
fun displayTable(orders: List<List<String>>): List<List<String>> {

}
}
```

Swift Solution:

```
class Solution {
func displayTable(_ orders: [[String]]) -> [[String]] {

}
}
```

Rust Solution:

```
// Problem: Display Table of Food Orders in a Restaurant
// Difficulty: Medium
// Tags: array, string, 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 display_table(orders: Vec<Vec<String>>) -> Vec<Vec<String>> {

}
}
```

Ruby Solution:

```
# @param {String[][]} orders
# @return {String[][]}
def display_table(orders)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[][] $orders
* @return String[][]
*/
function displayTable($orders) {

}
}
```

Scala Solution:

```
object Solution {
def displayTable(orders: List[List[String]]): List[List[String]] = {

}
}
```