Problem 1773: Count Items Matching a Rule

Problem Information

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

Problem Description

You are given an array

items

, where each

items[i] = [type

i

, color

i

, name

i

]

describes the type, color, and name of the

i

th

item. You are also given a rule represented by two strings,

ruleKey

and

ruleValue

.

The

i

th

item is said to match the rule if

one

of the following is true:

ruleKey == "type"

and

ruleValue == type

i

.

ruleKey == "color"

and

ruleValue == color

i

.

ruleKey == "name"

and

ruleValue == name

i

.

Return

the number of items that match the given rule

.

Example 1:

Input:

items = [["phone","blue","pixel"],["computer","silver","lenovo"],["phone","gold","iphone"]], ruleKey = "color", ruleValue = "silver"

Output:

1

Explanation:

There is only one item matching the given rule, which is ["computer","silver","lenovo"].

Example 2:

Input:

items = [["phone","blue","pixel"],["computer","silver","phone"],["phone","gold","iphone"]], ruleKey = "type", ruleValue = "phone"

Output:

2

Explanation:

There are only two items matching the given rule, which are ["phone","blue","pixel"] and ["phone","gold","iphone"]. Note that the item ["computer","silver","phone"] does not match.

Constraints:

1 <= items.length <= 10

4

1 <= type

i

.length, color

i

.length, name

i

.length, ruleValue.length <= 10

ruleKey

is equal to either

"type"

,

"color"

, or

"name"

.

All strings consist only of lowercase letters.

Code Snippets

C++:

```
class Solution {
public:
int countMatches(vector<vector<string>>& items, string ruleKey, string
ruleValue) {

}
};
```

Java:

```
class Solution {
public int countMatches(List<List<String>> items, String ruleKey, String
ruleValue) {

}
}
```

Python3:

```
class Solution:
def countMatches(self, items: List[List[str]], ruleKey: str, ruleValue: str)
-> int:
```

Python:

```
class Solution(object):
def countMatches(self, items, ruleKey, ruleValue):
"""
```

```
:type items: List[List[str]]
:type ruleKey: str
:type ruleValue: str
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string[][]} items
* @param {string} ruleKey
* @param {string} ruleValue
* @return {number}
*/
var countMatches = function(items, ruleKey, ruleValue) {

};
```

TypeScript:

```
function countMatches(items: string[][], ruleKey: string, ruleValue: string):
number {

};
```

C#:

```
public class Solution {
public int CountMatches(IList<IList<string>> items, string ruleKey, string
ruleValue) {

}
}
```

C:

```
int countMatches(char*** items, int itemsSize, int* itemsColSize, char*
ruleKey, char* ruleValue) {

}
```

Go:

```
func countMatches(items [][]string, ruleKey string, ruleValue string) int {

}
```

Kotlin:

```
class Solution {
fun countMatches(items: List<List<String>>, ruleKey: String, ruleValue:
String): Int {

}
}
```

Swift:

```
class Solution {
func countMatches(_ items: [[String]], _ ruleKey: String, _ ruleValue:
String) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_matches(items: Vec<Vec<String>>, rule_key: String, rule_value:
String) -> i32 {

}
}
```

Ruby:

```
# @param {String[][]} items
# @param {String} rule_key
# @param {String} rule_value
# @return {Integer}
def count_matches(items, rule_key, rule_value)

end
```

PHP:

```
class Solution {

/**
* @param String[][] $items
* @param String $ruleKey
* @param String $ruleValue
* @return Integer
*/
function countMatches($items, $ruleKey, $ruleValue) {

}
}
```

Dart:

```
class Solution {
int countMatches(List<List<String>> items, String ruleKey, String ruleValue)
{

}
}
```

Scala:

```
object Solution {
def countMatches(items: List[List[String]], ruleKey: String, ruleValue:
String): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_matches(items :: [[String.t]], rule_key :: String.t, rule_value
:: String.t) :: integer
def count_matches(items, rule_key, rule_value) do

end
end
```

Erlang:

```
-spec count_matches(Items :: [[unicode:unicode_binary()]], RuleKey ::
unicode:unicode_binary(), RuleValue :: unicode:unicode_binary()) ->
integer().
count_matches(Items, RuleKey, RuleValue) ->
.
```

Racket:

```
(define/contract (count-matches items ruleKey ruleValue)
(-> (listof (listof string?)) string? string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Items Matching a Rule
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int countMatches(vector<vector<string>>& items, string ruleKey, string
ruleValue) {

}
};
```

Java Solution:

```
/**
* Problem: Count Items Matching a Rule
* Difficulty: Easy
* Tags: array, string
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int countMatches(List<List<String>> items, String ruleKey, String
ruleValue) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Items Matching a Rule
Difficulty: Easy
Tags: array, string

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def countMatches(self, items: List[List[str]], ruleKey: str, ruleValue: str)
-> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def countMatches(self, items, ruleKey, ruleValue):
"""
:type items: List[List[str]]
:type ruleKey: str
:type ruleValue: str
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Count Items Matching a Rule
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {string[][]} items
* @param {string} ruleKey
* @param {string} ruleValue
* @return {number}
*/
var countMatches = function(items, ruleKey, ruleValue) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Items Matching a Rule
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function countMatches(items: string[][], ruleKey: string, ruleValue: string):
number {

};
```

C# Solution:

```
/*
* Problem: Count Items Matching a Rule
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int CountMatches(IList<IList<string>> items, string ruleKey, string
ruleValue) {

}
}
```

C Solution:

```
/*
* Problem: Count Items Matching a Rule
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int countMatches(char*** items, int itemsSize, int* itemsColSize, char*
ruleKey, char* ruleValue) {

}
```

Go Solution:

```
// Problem: Count Items Matching a Rule
// Difficulty: Easy
// Tags: array, string
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach
```

```
func countMatches(items [][]string, ruleKey string, ruleValue string) int {

}
```

Kotlin Solution:

```
class Solution {
fun countMatches(items: List<List<String>>, ruleKey: String, ruleValue:
String): Int {

}
}
```

Swift Solution:

```
class Solution {
func countMatches(_ items: [[String]], _ ruleKey: String, _ ruleValue:
String) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count Items Matching a Rule
// Difficulty: Easy
// Tags: array, string
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn count_matches(items: Vec<Vec<String>>, rule_key: String, rule_value:
String) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String[][]} items
# @param {String} rule_key
# @param {String} rule_value
# @return {Integer}
def count_matches(items, rule_key, rule_value)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[][] $items
* @param String $ruleKey
* @param String $ruleValue
* @return Integer
*/
function countMatches($items, $ruleKey, $ruleValue) {

}
}
```

Dart Solution:

```
class Solution {
int countMatches(List<List<String>> items, String ruleKey, String ruleValue)
{

}
}
```

Scala Solution:

```
object Solution {
def countMatches(items: List[List[String]], ruleKey: String, ruleValue:
String): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_matches(items :: [[String.t]], rule_key :: String.t, rule_value
:: String.t) :: integer
def count_matches(items, rule_key, rule_value) do

end
end
```

Erlang Solution:

```
-spec count_matches(Items :: [[unicode:unicode_binary()]], RuleKey ::
unicode:unicode_binary(), RuleValue :: unicode:unicode_binary()) ->
integer().
count_matches(Items, RuleKey, RuleValue) ->
.
```

Racket Solution:

```
(define/contract (count-matches items ruleKey ruleValue)
(-> (listof (listof string?)) string? string? exact-integer?)
)
```