1773. Count Items Matching a Rule

Easy Array String

Problem Description

In this problem, we are provided with an array named `items`, where each element is a list containing strings that represent the type, color, and name of an item in that order (i.e., `items[i] = [type_i, color_i, name_i]`). We are also given two strings, `ruleKey` and `ruleValue`, which represent a rule we should use to match items.

An item matches the rule if one of the following conditions is satisfied:

- If `ruleKey` is "type", the type of the item (`type_i`) should match `ruleValue`.
- If `ruleKey` is "color", the color of the item (`color_i`) should match `ruleValue`.
- If `ruleKey` is "name", the name of the item (`name_i`) should match `ruleValue`.

We need to return the count of items that match the provided rule.

Intuition

The problem can be approached by understanding that the `ruleKey` will always be one of three options: "type", "color", or "name". Given this, we can determine that each option corresponds to an index in the item's list. "type" corresponds to index 0, "color" to index 1, and "name" to index 2.

Knowing this, the solution approach is straightforward:

1. Determine the index that `ruleKey` corresponds to.
2. Iteratively count the number of items where the element at the determined index matches the `ruleValue`.

The provided solution translates this intuition into code by first using a conditional expression to find the index (`i`) that corresponds with the `ruleKey`. Notice the use of the first character (`'t'`, `'c'`, or any other) to determine the index. This works because in the context of this problem, the first character of each `ruleKey` option is unique.

Solution Approach

The implementation of the solution uses a simple array iteration and indexing technique. It leverages the built-in `sum()` function to count the number of matches for the given condition, using a generator expression to avoid creating an intermediary list.

Here's the breakdown of the implementation steps:

1. Determine the index for comparison based on `ruleKey`:
   - We need to compare the item's type, color, or name based on the `ruleKey`. Since these attributes are in a fixed order within each item (`[type_i, color_i, name_i]`), we map the `ruleKey` to an index:
     - "type" → `0`
     - "color" → `1`
     - "name" → `2`
   - This is accomplished using a simple conditional expression:

```
i = 0 if ruleKey[0] == 't' else (1 if ruleKey[0] == 'c' else 2)
```

2. Count the number of items that match `ruleValue` at the determined index:
   - We then iterate over each item in the `items` list and increase the count when the element at the index `i` is equal to `ruleValue`.
   - This is done using the `sum()` function and a generator expression, which succinctly combines iteration and condition checking:

```
return sum(v[i] == ruleValue for v in items)
```

- The generator expression `(v[i] == ruleValue for v in items)` iterates through each item `v` in `items`, checking whether the element at index `i` equals `ruleValue`. For each item where the condition is true, the expression yields `True`, which is numerically equivalent to `1`.
- The `sum()` function then adds up all the `1`s, effectively counting the number of `True` instances, thus giving us the total count of matching items.

In terms of algorithms and data structures, the solution uses linear search, which is appropriate since there are no constraints that would benefit from a more complex searching algorithm. The algorithm's time complexity is O(n), where n is the number of items in the `items` list, as every item must be checked against the rule. The space complexity is O(1) since the solution does not allocate additional space proportional to the input size; it only uses a few variables for counting and storing the index.

Example Walkthrough

Let's take the following example to illustrate the solution approach:

Suppose we have an array of items where each item is described by its type, color, and name:

```
items = [["phone", "blue", "pixel"], ["computer", "silver", "lenovo"], ["phone", "gold", "iphone"]]
```

And our task is to count how many items match the rule consisting of `ruleKey = "color"` and `ruleValue = "silver"`.

According to the approach:

1. First, we determine the comparison index from the `ruleKey`.
   - Since `ruleKey` is "color", we are interested in the second element of each item, which corresponds to index `1` (since indexing starts at 0).
   - Following the provided code snippet, we would get `i = 1` because `ruleKey[0]` is `'c'`, corresponding to "color".
2. Next, we iterate over each item and count the occurrences where the item's color matches "silver":
   - We check each item's element at index 1 to see whether it equals "silver".
   - From our example array `items`, only the second item (`["computer", "silver", "lenovo"]`) has "silver" as its color at index 1.

Using a generator expression within the `sum()` function, we effectively iterate with the condition:

```
count = sum(item[1] == "silver" for item in items)
```

This gives us:

```
count = sum(True for ["computer", "silver", "lenovo"])
```

- Since "silver" matches the only "silver" in our `items`, the `sum()` function calculates the sum of `1` instance (`True`), resulting in `count = 1`.

Therefore, in this example, the number of items that match the rule (`"color", "silver"`) is `1`.

Solution Implementation

Python

```python
class Solution:
    def countMatches(self, items: List[List[str]], ruleKey: str, ruleValue: str) -> int:
        # Determine index based on the first character of ruleKey
        # 0 for 'type', 1 for 'color', and 2 for 'name'
        index = 0 if ruleKey[0] == 't' else (1 if ruleKey[0] == 'c' else 2)

        # Count and return how many items match the given rule
        # Loop through each item in items list and check if the
        # element at the determined index matches the rule value
        return sum(item[index] == ruleValue for item in items)
```

Java

```java
class Solution {
    // Counts the number of matching items based on the given rule key and rule value.
    // @param items List of items where each item is represented as a List of Strings with a specific order [type, color, name].
    // @param ruleKey The rule key representing the attribute by which we want to match (type, color, or name).
    // @param ruleValue The value we want to match against the selected attribute.
    // @return The count of items that match the specified rule.
    public int countMatches(List<List<String>> items, String ruleKey, String ruleValue) {
        // Determine which attribute (0 for type, 1 for color, 2 for name) we need to check based on the ruleKey.
        int attributeIndex = "type".equals(ruleKey) ? 0 : ("color".equals(ruleKey) ? 1 : 2);

        // Initialize a counter for the number of matches.
        int matchCount = 0;

        // Iterate through all the items in the list.
        for (List<String> item : items) {
            // Check if the current item's attribute matches the ruleValue.
            if (item.get(attributeIndex).equals(ruleValue)) {
                // If it matches, increment the count of matches.
                matchCount++;
            }
        }

        // Return the final count of matches.
        return matchCount;
    }
}
```

C++

```cpp
#include <vector> // Include necessary header for vector
#include <algorithm> // Include necessary header for count_if function

class Solution {
public:
    // Function to count matches of items based on ruleKey and ruleValue.
    // @param items: a 2D vector containing item information
    // @param ruleKey: the key to match (type, color, or name)
    // @param ruleValue: the value to match with the ruleKey
    // @return: count of items that match the criterion
    int countMatches(vector<vector<string>>& items, string ruleKey, string ruleValue) {
        // Determine the index based on the ruleKey. "type" corresponds to index 0, "color" to 1, "name" to 2.
        int index;
        if (ruleKey == "type") {
            index = 0; // Index for type
        } else if (ruleKey == "color") {
            index = 1; // Index for color
        } else {
            index = 2; // Index for name
        }

        // Use count if algorithm to count elements satisfying a condition
        // The condition is that the specified field of an item matches the ruleValue
        int matchCount = count_if(items.begin(), items.end(), [&](const auto& item) {
            return item[index] == ruleValue;
        });

        return matchCount; // Return the total count of matches
    }
};
```

TypeScript

```typescript
// Define the type for the item list which is an array of arrays of strings.
type ItemList = string[][];

/**
 * Counts the matches of items based on a given rule.
 *
 * @param {ItemList} items - The array of items. Each item is an array containing properties like type, color, and name.
 * @param {string} ruleKey - The key of the rule which can be 'type', 'color', or 'name'.
 * @param {string} ruleValue - The value of the rule to match against the items' properties.
 * @returns {number} The count of items that match the rule.
 */
function countMatches(items: ItemList, ruleKey: string, ruleValue: string): number {
    // Determine the index associated with the ruleKey.
    // 0 for 'type', 1 for 'color', and 2 for 'name'.
    const ruleIndex: number = ruleKey === 'type' ? 0 : ruleKey === 'color' ? 1 : 2;

    // Use the reduce function to iterate over the items and increment the count
    // based on whether the item property at ruleIndex matches the ruleValue.
    return items.reduce((count: number, item: string[]) => {
        return count + (item[ruleIndex] === ruleValue ? 1 : 0)
    }, 0); // Initialize the count as 0.
}
```

```
class Solution:
    def countMatches(self, items: List[List[str]], ruleKey: str, ruleValue: str) -> int:
        # Determine index based on the first character of ruleKey
        # 0 for 'type', 1 for 'color', and 2 for 'name'
        index = 0 if ruleKey[0] == 't' else (1 if ruleKey[0] == 'c' else 2)

        # Count and return how many items match the given rule
        # Loop through each item in items list and check if the
        # element at the determined index matches the rule value
        return sum(item[index] == ruleValue for item in items)
```

Time and Space Complexity

The time complexity of the provided code is `O(n)`, where `n` is the number of items in the input list. This is because the code iterates once over all the items, checking whether the value at a certain index matches the `ruleValue`. The index `i` is determined based on the initial character of the `ruleKey`, which is a constant-time operation.

The space complexity is `O(1)` (constant space) because aside from the space taken by the input, the code uses a fixed amount of extra space - the variable `i` and the space for the generator expression inside the `sum` function. No additional space that scales with the input size is used.