2678. Number of Senior Citizens

Easy Array String

Leetcode Link

Problem Description

You are given an array `details`, where each element represents information about an individual passenger. The format of this information is a string exactly 15 characters long. Let's break down the content of this string:

- The first ten characters are the phone number of the passenger.
- The eleventh character (index 10) represents the passenger's gender.
- The twelfth and thirteenth characters (indexes 11 and 12) denote the passenger's age.
- The last two characters (indexes 13 and 14) signify the seat that has been allotted to the passenger.

Your task is to calculate and return the number of passengers who are older than 60 years, purely based on the age data provided within each string.

Intuition

The intuitive approach to solving this problem relies on understanding the structure of each string in the `details` array. Since the age of each passenger is located at a specific position within the string (indexes 11 and 12), we can focus entirely on those two characters.

By simply iterating over each string in the `details` array, we can extract the age part of the string, convert it to an integer, and then check if this age is greater than 60. For each passenger meeting this criterion (age > 60), we would increment our count.

Therefore, the solution consists of traversing each string, extracting the age, converting it to an integer, and applying the comparison operator to tally up all the seniors in the list.

Solution Approach

The solution implementation relies on a simple iteration and counting pattern, a common tactic in problems requiring the counting of elements that satisfy a certain condition. Here, Python's concise list comprehension and the built-in `sum` function make this task even more straightforward.

The algorithm can be described as follows:

1. Use a list comprehension to iterate through each string `x` in the given `details` array.
2. For each `x`, extract the characters that represent the age: `x[11:13]`.
3. Convert that substring to an integer with `int(x[11:13])`.
4. Use a comparison operator `>` to check if the extracted age is greater than `60`.
5. The result of this comparison will be a boolean value (`True` if the age is greater than 60, `False` otherwise).
6. The list comprehension returns a list of boolean values, which when passed to the `sum` function, counts the `True` values since `True` is considered as `1` and `False` is `0` in Python.
7. The final sum represents the total number of passengers over the age of 60.

Here's the list comprehension explained:

```
sum(int(x[11:13]) > 60 for x in details)
```

- `for x in details` means we are looping over every element in the `details` array.
- `int(x[11:13])` is converting the age portion of the string to an integer.
- `int(x[11:13]) > 60` performs the check to see if the age is above 60.

The beauty of this approach lies in its simplicity and Python's ability to handle such operations in one line, demonstrating the power of list comprehensions and the `sum` function in Python.

Example Walkthrough

Let's consider the following small example to illustrate the solution approach.

Suppose the `details` array is as follows:

```
details = [
    "123456789OM658C1",
    "0987654321F578C2",
    "5678901234M521C3",
    "2345678901F411C4",
    "3456789012M629C5"
]
```

Following the solution approach:

1. We iterate over each string in the `details` array.
2. For the first string `details[0]`, we extract the age `details[0][11:13]` which gives us '65'. We convert this to an integer using `int('65')`, which is `65`.
3. We check if `65` is greater than `60`. It is, so it contributes to our count.
4. For the second string `details[1]`, the extracted age is '57', which is not greater than `60`. Therefore, it doesn't add to the count.
5. We repeat this process for all elements in the list. Here are the ages:
   - "123456789OM**65**8C1" → Age is `65`, count incremented.
   - "0987654321F**57**8C2" → Age is `57`, not incremented.
   - "5678901234M**52**1C3" → Age is `52`, not incremented.
   - "2345678901F**41**1C4" → Age is `41`, not incremented.
   - "3456789012M**62**9C5" → Age is `62`, count incremented.
6. Now we have 2 passengers over the age of 60 from our list.
7. The `sum` function would add up the `True` values (each `True` representing a passenger above 60) resulting in a final count of `2`.

Therefore, when we run this code:

```
sum(int(x[11:13]) > 60 for x in details)
```

It will iterate over the list of `details`, check ages, and return the sum of `True` values which corresponds to the number of passengers above the age of 60. In our example, the final count is `2`.

Python Solution

```python
class Solution:
    def countSeniors(self, details: List[str]) -> int:
        # This method takes a list of strings where each string contains details that include age.
        # It returns the count of how many details strings represent people older than 60 years.

        # Initialize a count variable to keep track of the number of seniors
        senior_count = 0

        # Iterate over each detail string in the provided list
        for detail in details:
            # Extract the age part from the detail string, assuming age is always at the 11th to 13th characters
            age = int(detail[11:13])

            # Check if the extracted age is greater than 60
            if age > 60:
                # If true, increment the senior count
                senior_count += 1

        # Return the total count of seniors
        return senior_count

# Note: You'd still need to import List from the typing module
# for the type annotation to work: from typing import List
```

Java Solution

```java
class Solution {
    // Method to count the number of seniors based on their age details.
    public int countSeniors(String[] details) {
        // Initialize a counter for seniors.
        int seniorCount = 0;

        // Loop through all the provided age details.
        for (String detail : details) {
            // Extract the age from the current detail string.
            // Assuming the age is always in the same position (index 11 to 13).
            int age = Integer.parseInt(detail.substring(11, 13));

            // Check if the age is greater than 60 to determine if the person is a senior.
            if (age > 60) {
                // Increment the count for seniors.
                seniorCount++;
            }
        }

        // Return the final count of seniors.
        return seniorCount;
    }
}
```

C++ Solution

```cpp
#include <vector>
#include <string>

class Solution {
public:
    // Counts the number of seniors based on their age found in details.
    // Assumes that details contain an age starting at the 12th character of each string.
    int countSeniors(vector<string>& details) {
        int seniorCount = 0; // Initialize a count of seniors

        // Iterate over each string in the details vector
        for (const auto& detail : details) {
            // Extract the age from the string. Assuming that the age
            // starts at the 12th character (index 11) and is 2 digits long.
            int age = stoi(detail.substr(11, 2));

            // Increment the senior count if the age is greater than 60
            seniorCount += (age > 60) ? 1 : 0;
        }

        // Return the total count of seniors
        return seniorCount;
    }
};

// Note: This code assumes that the 'details' vector and each string within it are properly formatted,
// with the age starting at the 12th character and being two characters long.
```

Typescript Solution

```typescript
// Function to count the number of seniors in a list of details
function countSeniors(details: string[]): number {
    // Initialize the count of seniors
    let seniorCount = 0;

    // Loop through each detail string in the details array
    for (const detail of details) {
        // Extract the age part of the detail string by slicing from
        // the 12th character to the 14th (zero indexed)
        const age = parseInt(detail.slice(11, 13));

        // If the age is greater than 60, increment the senior count
        if (age > 60) {
            seniorCount++;
        }
    }

    // Return the total count of seniors
    return seniorCount;
}
```

Time and Space Complexity

The time complexity of the code is `O(n)`, where `n` is the length of the `details` list. This is because the code iterates over each element of the list exactly once when performing the sum, and the string slicing and integer comparison for each element are constant time operations.

The space complexity of the code is `O(1)`, indicating that the amount of additional memory used does not depend on the size of the input list. The only extra space used is for the cumulative sum and temporary variables for the string slicing and the integer comparison, all of which require a constant amount of space.