1154. Day of the Year

Easy Math String

Leetcode Link

Problem Description

The problem requires us to determine the day number of the year for a given date formatted as `YYYY-MM-DD`. This means we need to calculate which day of the year the given date is. For example, January 1st would be day number 1, February 1st would be day number 32 (assuming it is not a leap year), and so on. To solve this, we need to consider the varying lengths of each month and also account for leap years, which affect the length of February.

Intuition

The solution takes a straightforward approach to determining the day number by summing the number of days in each month leading up to the given month and then adding the days elapsed in the current month. The first step in the solution is to parse the input string to extract the year, month, and day as integers.

Leap years are taken into account by checking if February should have 28 or 29 days. If the year is divisible by 400 or divisible by 4 but not by 100, it is a leap year, so February has 29 days; otherwise, it has 28 days. This rule comes from the definition of leap years in the Gregorian calendar.

An array named `days` is then constructed, which holds the number of days in each month, adjusted for leap years for February. The day number is calculated by summing the days in the months prior to the given month and then adding the day of the month itself. The index into the `days` array is one less than the month number since array indices in Python are zero-based, but months are one-based (i.e., January is month 1, not month 0).

Solution Approach

The implementation uses a simple algorithm with basic data structures, specifically a list to represent the number of days in each month.

Here's a step-by-step breakdown of the `dayOfYear` function:

1. **Parse the date**: First, the function splits the input date string into the year `y`, month `m`, and day `d` components by using the `split('-')` method and converting each to an integer.
2. **Determine if the year is a leap year**: Based on the extracted year, it calculates whether it is a leap year. The given year `y` is a leap year if it is evenly divisible by `400` or it is divisible by `4` but not by `100`. This leap year check is used to decide the number of days in February (29 days for a leap year, otherwise 28).
3. **Store days per month**: A list named `days` is created, holding the number of days in each month, and it is adjusted for leap years for February. The list is indexed such that `days[0]` corresponds to January, `days[1]` to February, and so on up to `days[11]` for December.
4. **Calculate the day number**: Finally, the function returns the sum of the days in the months before the given month (note that `days[: m - 1]` creates a slice of the list from the beginning up to but not including the index `m - 1`). We subtract `1` from `m` because the days in the current month (`m`) have not yet elapsed completely, so we only want to sum the days of the full months that have passed. Then it adds the days `d` from the current month to this sum.

The algorithm's efficiency comes from the constant time operations to check for a leap year and the simple summation of integers in a list, making the overall time complexity O(1), since the list is of fixed length (12 elements for the months of the year), and thus does not depend on the size of the input. The space complexity is also O(1) because the list of days per month does not grow with the input size.

Example Walkthrough

Let's walk through an example to illustrate the solution approach using the date "2021-03-01."

1. **Parse the date**: The input date string "2021-03-01" is split into year `y=2021`, month `m=3`, and day `d=1`.
2. **Determine if the year is a leap year**: The year 2021 is not a leap year because it is not divisible by 400 and, while it is divisible by 4, it is also divisible by 100. Therefore, February of 2021 has 28 days.
3. **Store days per month**: We create a list `days` that holds the number of days in each month for the year 2021: `[31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]`.
4. **Calculate the day number**: We sum the days in the months prior to March, which are January and February: `31 (January) + 28 (February) = 59`. Since we are calculating for March 1st, we add the day `d`, which is 1 in this case, to the sum of the previous months: `59 + 1 = 60`.

The day number for "2021-03-01" is therefore 60. It is the 60th day of the year 2021.

Python Solution

```python
class Solution:
    def dayOfYear(self, date: str) -> int:
        # Split the date string into year, month, and day components
        year, month, day = (int(part) for part in date.split("-"))

        # Determine if the year is a leap year for February day count
        # Leap year occurs every 4 years, except for years that are divisible by 100
        # unless they are divisible by 400 as well
        is_leap_year = year % 400 == 0 or (year % 4 == 0 and year % 100 != 0)
        february_days = 29 if is_leap_year else 28

        # List of days in each month, taking into account leap year for February
        days_in_month = [31, february_days, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]

        # Calculate the day of the year by summing the days in the previous months
        # and adding the day of the current month
        day_of_year = sum(days_in_month[:month - 1]) + day

        return day_of_year

```

Java Solution

```java
class Solution {

    // Method to calculate the day of the year given a date
    public int dayOfYear(String date) {
        // Parse the year, month, and day from the input string
        int year = Integer.parseInt(date.substring(0, 4));
        int month = Integer.parseInt(date.substring(5, 7));
        int day = Integer.parseInt(date.substring(8));

        // Calculate if the year is a leap year
        boolean isLeapYear = (year % 400 == 0) || (year % 4 == 0 && year % 100 != 0);
        // Set the number of days in February depending on the leap year
        int februaryDays = isLeapYear ? 29 : 28;
        // Array containing the number of days in each month
        int[] daysOfMonth = {31, februaryDays, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

        // Initialize the count of days with the days of the given month
        int dayOfYearCount = day;

        // Add the days of all the months before the given month
        for (int i = 0; i < month - 1; ++i) {
            dayOfYearCount += daysOfMonth[i];
        }

        // Return the computed day of the year
        return dayOfYearCount;
    }
}

```

C++ Solution

```cpp
#include <string>
using std::string;  // Including string and using the standard namespace for string

class Solution {
public:
    // This function calculates the day of the year given a date in the format YYYY-MM-DD
    int dayOfYear(string date) {
        // Extract the year, month, and day as integers from the input string
        int year = stoi(date.substr(0, 4));
        int month = stoi(date.substr(5, 2));
        int day = stoi(date.substr(8));

        // Check if the year is a leap year
        int febDays = (year % 400 == 0 || (year % 4 == 0 && year % 100 != 0)) ? 29 : 28;

        // Initialize the array with the number of days for each month
        int daysPerMonth[] = {31, febDays, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

        // Start with the given day
        int totalDays = day;

        // Add the days of the months preceding the given month
        for (int i = 0; i < month - 1; ++i) {
            totalDays += daysPerMonth[i];
        }

        // Return the total count of days
        return totalDays;
    }
};

```

Typescript Solution

```typescript
/**
 * Calculates the day of the year based on the given date string.
 *
 * @param {string} date - The input date in the format "YYYY-MM-DD".
 * @return {number} - The day of the year.
 */
function dayOfYear(date: string): number {
    // Extract year, month, and day from the date string.
    const year: number = +date.slice(0, 4);
    const month: number = +date.slice(5, 7);
    const day: number = +date.slice(8);

    // Determine if the year is a leap year.
    const isLeapYear: boolean = year % 400 === 0 || (year % 4 === 0 && year % 100 !== 0);

    // Define the number of days in February depending on the leap year.
    const februaryDays: number = isLeapYear ? 29 : 28;

    // Create an array representing the days in each month.
    const monthDays: number[] = [31, februaryDays, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31];

    // Sum the days from the preceding months and add the days from the current month.
    return monthDays.slice(0, month - 1).reduce((accumulator, current) => accumulator + current, day);
}

// Example usage:
// const result: number = dayOfYear("2022-12-31");
// console.log(result); // Outputs the day of the year for December 31, 2022

```

Time and Space Complexity

The given Python code computes the day of the year for a given date in the format YYYY-MM-DD. Upon analyzing the code, we can derive its time complexity and space complexity.

Time Complexity

The time complexity of the code is `O(1)`.

- The date string splitting and integer conversion happens in constant time, since the date format is fixed.
- The logic to determine if it's a leap year for setting the value of February (`d`) is done in constant time.
- The creation of the `days` array, which contains the number of days for each month, is a constant operation because the array size does not change dynamically; it is always size 12.
- The `sum(days[: m - 1])` operation also has a constant time complexity since it will never sum more than 11 elements (the months before December).

Since none of these operations depends on the size of an input, they all occur in constant time.

Space Complexity

The space complexity of the code is also `O(1)`.

- The additional space used by the variables `y`, `m`, `d`, and `v` is constant and does not depend on the input size.
- The `days` array is also constant in size; it always contains 12 integers.
- The summation operation does not consume additional space that scales with input.

Therefore, both the additional memory used for the calculation and the output size are fixed, resulting in a constant space complexity.