1118. Number of Days in a Month

Easy Math

Leetcode Link

Problem Description

The problem requires us to create a function that, given a year (`year`) and a month (`month`), returns the number of days in that given month. The challenge involves accounting for the different number of days in each month and determining whether the given year is a leap year since February has 29 days instead of 28 in leap years.

Intuition

The intuition behind the solution is to use a list to map each month to its respective number of days. We know that:

- January, March, May, July, August, October, and December all have 31 days.
- April, June, September, and November have 30 days.
- February has 28 days in a normal year and 29 days in a leap year.

To handle the special case of February in a leap year, we first need to determine whether the given year is a leap year. The determination can be made using the following rules:

- A year is a leap year if it is divisible by 4, except for end-of-century years, which must be divisible by 400.
- This means that if a year is divisible by 100 and not divisible by 400, it is NOT a leap year.

From these rules, we construct a logical condition that ensures the year is a leap year if it is either divisible by 400 or divisible by 4 but not by 100.

After this, we construct a list of days where February has 29 days if it's a leap year and 28 days otherwise. Then, the function simply returns the number of days corresponding to the given month by looking up the value in the pre-constructed list based on the `month` index.

Solution Approach

The solution follows a simple and direct approach using a combination of condition checking and list indexing, which are fundamental constructs in programming - especially useful for this type of calendar-related computations.

We start by determining if the provided `year` is a leap year. The code does this with a single line of Boolean logic:

```
leap = (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0)
```

This line employs the modulus operator `%` to check for divisibility. The condition checks if the `year` is divisible by 4 but not by 100, unless it is also divisible by 400, in which case the `year` is indeed a leap year.

With the leap year status determined, we proceed to construct a list that maps each month (from index `1` to `12`) to its corresponding number of days:

```
days = [0, 31, 29 if leap else 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
```

Notice that February (the second element of this list) has two possible values: `29` if `leap` is `True`, otherwise `28`. The list starts with a placeholder `0` at index `0` since there is no month `0`, and this alignment allows us to directly use the `month` value as an index to the list.

Finally, the function returns the number of days corresponding to the input `month` by accessing the days' list using the `month` as an index:

```
return days[month]
```

The use of list indexing here provides an efficient and clean solution, avoiding multiple conditional statements or explicit date handling libraries, which are unnecessary for the problem at hand.

Example Walkthrough

Let's walk through a small example to illustrate the solution approach. We'll create a function named `get_days_in_month` that implements the described approach. Suppose we are given the year `2020` which is a leap year and the month `February`, and we want to find out how many days February has in this year.

Given in 2020:

```
year = 2020
month = 2  # Since February is the second month
```

We first determine if `2020` is a leap year. The code:

```
leap = (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0)
```

evaluates to:

```
leap = (2020 % 4 == 0 and 2020 % 100 != 0) or (2020 % 400 == 0)
```

As `2020` is divisible by `4` and not divisible by `100`, `leap` becomes `True`. Since `2020` is a leap year, February will have `29` days.

Next, we create a list that maps each month to its number of days, accounting for leap years:

```
days = [0, 31, 29 if leap else 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
```

Which now translates to:

```
days = [0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
```

because `leap` is `True`, index `2` (which corresponds to February) is assigned the value `29`.

Finally, to find out the number of days in February 2020, the function performs a simple list indexing operation:

```
return days[month]
```

Which gives us:

```
return days[2]
```

The function returns `29`, which is the correct number of days in February during a leap year.

So by following these steps, our function `get_days_in_month` would correctly determine that there are `29` days in February `2020`. A straightforward sequence of logical checks and array indexing provides us with a concise and effective solution.

Python Solution

```python
class Solution:
    def numberOfDays(self, year: int, month: int) -> int:
        # Determine if the given year is a leap year. A year is a leap year if it is
        # divisible by 4, but not by 100, unless it is also divisible by 400.
        is_leap_year = (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0)

        # A list where the index corresponds to the month (1-12),
        # and the value is the number of days in that month. February has
        # 29 days if it's a leap year, or 28 days if it's not.
        days_in_month = [
            0,  # Index 0 - not used, for easier matching of month to index
            31,  # January
            29 if is_leap_year else 28,  # February
            31,  # March
            30,  # April
            31,  # May
            30,  # June
            31,  # July
            31,  # August
            30,  # September
            31,  # October
            30,  # November
            31  # December
        ]

        # Return the number of days in the specified month.
        return days_in_month[month]
```

Java Solution

```java
class Solution {

    /**
     * Calculates the number of days in a given month for a specific year
     *
     * @param year The year as an integer value
     * @param month The month as an integer value
     * @return The number of days in the given month of the year
     */
    public int numberOfDays(int year, int month) {
        // Determine if the year is a leap year
        boolean isLeapYear = (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);

        // Array holding the number of days in each month; for February, use leap year value if applicable
        int[] daysPerMonth = new int[] {
            0, // Placeholder for indexing purposes; there is no month 0
            31, // January
            isLeapYear ? 29 : 28, // February
            31, // March
            30, // April
            31, // May
            30, // June
            31, // July
            31, // August
            30, // September
            31, // October
            30, // November
            31  // December
        };

        // Return the number of days in the specified month
        return daysPerMonth[month];
    }
}
```

C++ Solution

```cpp
#include <vector> // Include the vector header for using the vector container

class Solution {
public:
    // Function to determine the number of days in a given month of a given year
    int numberOfDays(int year, int month) {
        // Check if the year is a leap year; a leap year is divisible by 4, not divisible by 100 unless also divisible by 400
        bool isLeapYear = (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);

        // Initialize a vector with the number of days in each month; February has 29 days if it is a leap year, otherwise 28
        std::vector<int> daysInMonth = {0, 31, isLeapYear ? 29 : 28, 31, 30, 31, 31, 30, 31, 30, 31, 30, 31};

        // Return the number of days in the specified month
        return daysInMonth[month];
    }
};
```

Typescript Solution

```typescript
// This function calculates the number of days in a given month for a specified year.
// It accounts for leap years when determining the number of days in February.
function numberOfDays(year: number, month: number): number {
    // Check if the year is a leap year. A year is a leap year if it is divisible by 4
    // but not by 100, or if it is divisible by 400.
    const isLeapYear: boolean = (year % 4 === 0 && year % 100 !== 0) || year % 400 === 0;

    // Create an array representing the number of days in each month.
    // For February (index 2), use 29 days if it's a leap year, otherwise use 28.
    const daysPerMonth: number[] = [0, 31, isLeapYear ? 29 : 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31];

    // Return the number of days of the specified month.
    return daysPerMonth[month];
}
```

Time and Space Complexity

Time Complexity

The time complexity of the given code is `O(1)` because it performs a constant number of operations no matter the value of the input `year` and `month`. Checking if a year is a leap year and accessing an element from a pre-defined list both take constant time.

Space Complexity

The space complexity of the code is also `O(1)` as it uses a fixed amount of additional memory. The list `days` is of a constant size (13 elements), and the space required does not grow with the size of the input `year` or `month`.