1360. Number of Days Between Two Dates

Easy Math String Leetcode Link

Problem Description

The task is to write a program that calculates the number of days between two given dates. The dates are provided in the string format 'YYYY-MM-DD', indicating the year, month, and day. The main challenge is to correctly handle the variations in the number of days in each month, leap years, and the possibility that the dates span multiple years. The solution requires attention to the rules that govern the Gregorian calendar, which include correct number of days in each month, leap year calculations, and the accumulation of days across multiple months and years.

Intuition

The intuition behind the provided solution lies in breaking down the task into smaller parts that are easier to manage:

1. **Understand leap years:** Leap years need special attention because they have an extra day in February (29 days instead of 28). The solution includes a function `isLeapYear` to check if a given year is a leap year. The rule for determining a leap year is that it should be divisible by 4 but not by 100 unless it is also divisible by 400.
2. **Count days per month:** The solution defines `daysInMonth` to handle the variation in the number of days per month, adjusting for leap years. An array holds the number of days in each month, with February being conditionally set to 28 or 29 days based on whether the year is a leap year.
3. **Calculate days since a reference date:** To calculate the total number of days of a date, `calcDays` processes each date independently, starting from a reference year (1971 in this implementation). It accumulates total days year by year and month by month up to the given date. Days in full years are counted by adding 365 for each year plus an additional day for each leap year. Days in months are counted by accumulating the days for each month leading up to the given month in the specified year.
4. **Compute the absolute difference:** Finally, the difference in days between the two dates is found by calculating the total days for each date since the reference date and subtracting the smaller one from the larger one to ensure a positive result. The `abs` function is used to return the absolute value of this difference.

This approach systematically adds up all the days from the reference date to each of the specified dates, and the difference between these sums represents the number of days between the dates.

Solution Approach

The implementation follows a straightforward algorithmic approach without using complex data structures or patterns. It focuses on accurately counting the days by considering each component of the date (year, month, day) one by one.

Here's a step-by-step breakdown of the algorithm:

1. **Leap Year Check:** The function `isLeapYear(year: int) -> bool` determines whether a given year is a leap year. This is used to add an extra day for leap years when counting days.

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

2. **Days in Month Calculation:** `daysInMonth(year: int, month: int) -> int` calculates the number of days in a given month for a specific year. It uses an array where each index corresponds to a month, with February's days conditionally set to 29 if it is a leap year:

   ```
   days = [31, 28 + int(isLeapYear(year)), 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
   return days[month - 1]
   ```

3. **Total Days Calculator:** `calcDays(date: str) -> int` converts a `YYYY-MM-DD` format date into the total number of days elapsed since a reference date, which in this code is January 1, 1971. It does this by first summing up all the days for the full years that have passed and then adding the days for the full months that have passed in the current year:

   ```
   year, month, day = map(int, date.split("-"))
   days = 0
   for y in range(1971, year):
       days += 365 + int(isLeapYear(y))
   for m in range(1, month):
       days += daysInMonth(year, m)
   days += day
   return days
   ```

4. **Calculation of the Difference Between Dates:** The days between the two dates are calculated by finding the absolute difference between the total days from the reference date to each input date:

   ```
   return abs(calcDays(date1) - calcDays(date2))
   ```

As you can see, the algorithm uses a combination of iteration and basic arithmetic operations to accurately calculate the total number of days between two dates. The problem can also be solved using Python's built-in `datetime` module, but this approach shows how it can be done manually, which could be beneficial in understanding how date calculations work or in environments where built-in date operations are not available.

Example Walkthrough

Let's consider two dates for this example: `2023-01-12` and `2023-03-04`. Our task is to find the number of days between these dates using the steps outlined in the solution approach.

1. **Leap Year Check:** We first check if the years for both dates are leap years. Since we're only considering dates within 2023 for this example, this step is simple:

   ```
   isLeapYear(2023) # returns False, because 2023 is not divisible by 4
   ```

   Since 2023 is not a leap year, February will have 28 days.

2. **Days in Month Calculation:** We then calculate the number of days in each month up to the month before the given date for the year 2023:

   ```
   daysInMonth(2023, 1) # returns 31 days for January
   daysInMonth(2023, 2) # returns 28 days for February, as it's not a leap year
   ```

   January has 31 days, and February, being in a non-leap year, has 28 days.

3. **Total Days Calculator:** Now, let's convert each date to the total number of days elapsed since the reference date of January 1, 1971:

   ```
   calcDays("2023-01-12") # we sum the days from 1971 to 2022, plus days in Jan 2023, and then add 12
   calcDays("2023-03-04") # we sum the days from 1971 to 2022, plus days in Jan and Feb 2023, and then add 4
   ```

   Without going through all the calculations step by step, let's assume the function `calcDays` returns the total days correctly according to the algorithm.

4. **Calculation of the Difference Between Dates:** Finally, to find the difference in days between the two dates, we subtract the total days for `2023-01-12` from the total days for `2023-03-04` and take the absolute value:

   ```
   abs(calcDays("2023-03-04") - calcDays("2023-01-12"))
   ```

   Suppose `calcDays("2023-01-12")` returns `19044` and `calcDays("2023-03-04")` returns `19095`, then our calculation will be:

   ```
   abs(19095 - 19044) # which gives us 51 days
   ```

   Based on our steps, there are 51 days between `2023-01-12` and `2023-03-04`.

Through this small example, we illustrated how each step of the solution contributes to the final count of the days between two dates without relying on any external date-related libraries.

Python Solution

```python
class Solution:
    def days_between_dates(self, date1: str, date2: str) -> int:
        """Calculate the number of days between two dates."""

        def is_leap_year(year: int) -> bool:
            """Check if a year is a leap year."""
            return year % 4 == 0 and (year % 100 != 0 or year % 400 == 0)

        def days_in_month(year: int, month: int) -> int:
            """Calculate the number of days in a month for a given year."""
            # Days in each month. For February in a leap year, add an extra day.
            days_per_month = [
                31,
                28 + int(is_leap_year(year)),
                31,
                30,
                31,
                30,
                31,
                31,
                30,
                31,
                30,
                31,
            ]
            # Return the number of days in the specified month.
            return days_per_month[month - 1]

        def calculate_days(date: str) -> int:
            """Calculate the total number of days from a fixed date to the given date."""
            # Extract year, month, and day as integers from the date string.
            year, month, day = map(int, date.split("-"))

            # Initialize the days counter.
            days = 0

            # Count days for the full years up to the given year.
            for y in range(1971, year):
                days += 365 + int(is_leap_year(y))

            # Count days for the full months in the given year.
            for m in range(1, month):
                days += days_in_month(year, m)

            # Add the days in the given month.
            days += day
            return days

        # Calculate the total days for each date and return their absolute difference.
        return abs(calculate_days(date1) - calculate_days(date2))

```

Java Solution

```java
class Solution {
    // Computes the number of days between two given dates
    public int daysBetweenDates(String date1, String date2) {
        // The absolute difference between the day counts gives the number of days between the two dates
        return Math.abs(calculateDays(date1) - calculateDays(date2));
    }

    // Determines if a given year is a leap year
    private boolean isLeapYear(int year) {
        // A year is a leap year if it's divisible by 4 but not by 100, or it's divisible by 400
        return (year % 4 == 0) && ((year % 100 != 0) || (year % 400 == 0));
    }

    // Returns the number of days in a given month for a given year
    private int daysInMonth(int year, int month) {
        // An array storing the number of days in each month
        int[] daysPerMonth = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
        // Adjust February for leap years
        daysPerMonth[1] += isLeapYear(year) ? 1 : 0;
        // Return the number of days in the specified month
        return daysPerMonth[month - 1];
    }

    // Calculates the total number of days from a fixed date (1971-01-01) to the given date
    private int calculateDays(String date) {
        // Parse the input date string to extract year, month, and day
        int year = Integer.parseInt(date.substring(0, 4));
        int month = Integer.parseInt(date.substring(5, 7));
        int day = Integer.parseInt(date.substring(8));
        int totalDays = 0;
        // Add days for each year from the base year (1971) to the given year
        for (int y = 1971; y < year; ++y) {
            totalDays += isLeapYear(y) ? 366 : 365;
        }
        // Add days for each month from January to the month before the given month in the given year
        for (int m = 1; m < month; ++m) {
            totalDays += daysInMonth(year, m);
        }
        // Add days in the given month
        totalDays += day;
        // Return the total day count
        return totalDays;
    }
}

```

C++ Solution

```cpp
class Solution {
public:
    // Calculate the number of days between two dates
    int daysBetweenDates(string date1, string date2) {
        // Calculate the absolute difference between the days counts of both dates
        return abs(calculateDays(date1) - calculateDays(date2));
    }

private:
    // Helper function to determine if a given year is a leap year
    bool isLeapYear(int year) {
        // A leap year is divisible by 4, but not by 100 unless it is also divisible by 400
        return (year % 4 == 0) && ((year % 100 != 0) || (year % 400 == 0));
    }

    // Helper function to get the number of days in a given month of a particular year
    int daysInMonth(int year, int month) {
        // Days in each month for a non-leap year
        int days[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
        // Adjust for leap year by adding one day to February
        days[1] += isLeapYear(year);
        return days[month - 1];
    }

    // Helper function to calculate the total number of days from a fixed date to the given date
    int calculateDays(string date) {
        // Parse the year, month, and day from the input string
        int year = stoi(date.substr(0, 4));
        int month = stoi(date.substr(5, 2));
        int day = stoi(date.substr(8, 2));

        // Count the days from the base date (1971-01-01) to the given date
        int totalDays = 0;
        for (int currentYear = 1971; currentYear < year; ++currentYear) {
            // Add 365 days for each year plus one more if it's a leap year
            totalDays += 365 + isLeapYear(currentYear);
        }
        for (int currentMonth = 1; currentMonth < month; ++currentMonth) {
            // Add the days of each month in the given year
            totalDays += daysInMonth(year, currentMonth);
        }
        // Add the days of the given month
        totalDays += day;

        return totalDays;
    }
};

```

Typescript Solution

```typescript
// Determines the number of days between two dates
function daysBetweenDates(date1: string, date2: string): number {
    // Calculate the absolute difference in days using the helper function 'calcDays'
    return Math.abs(calcDays(date1) - calcDays(date2));
}

// Checks if a given year is a leap year
function isLeapYear(year: number): boolean {
    // Leap year if divisible by 4 and not by 100 unless also divisible by 400
    return year % 4 === 0 && (year % 100 !== 0 || year % 400 === 0);
}

// Returns the number of days in a given month of a given year
function daysOfMonth(year: number, month: number): number {
    // An array representing the number of days in each month
    const daysPerMonth = [
        31,                             // January
        isLeapYear(year) ? 29 : 28,     // February - conditional on leap year
        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 - 1];
}

// Calculates the total number of days from a fixed date (1971-01-01) to the given 'date'
function calcDays(date: string): number {
    let totalDays = 0;
    // Parse the date string into year, month, and day components
    const [year, month, day] = date.split('-').map(Number);

    // Sum up all the days for the full years since the fixed date
    for (let currentYear = 1971; currentYear < year; ++currentYear) {
        totalDays += isLeapYear(currentYear) ? 366 : 365;
    }
    // Add the days for the months passed in the final year
    for (let currentMonth = 1; currentMonth < month; ++currentMonth) {
        totalDays += daysOfMonth(year, currentMonth);
    }
    // Finally, add the days passed in the final month, subtracting 1 to account for the start day
    totalDays += day - 1;

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

```

Time and Space Complexity

The given Python code calculates the number of days between two dates. We'll analyze the time complexity and space complexity separately.

Time Complexity:

1. The `isLeapYear` function is O(1), as it consists of only arithmetic operations.
2. The `daysInMonth` function is O(1) since it simply returns a value from a predefined list after calculating the leap year condition.
3. The `calcDays` function is the main function that determines time complexity:
   - It involves two for-loops:
     - The first loop runs from 1971 to the given year (excluding). If the given year `Y`, this first loop runs in O(Y−1971).
     - The second loop runs from 1 to the given month (excluding). Since the number of months is constant, this second loop runs in O(1).
   - The rest of the operations in `calcDays` are constant time calculations.
   - Assuming `Y1` and `Y2` are the respective years in `date1` and `date2`, the total time complexity for invoking `calcDays` for both dates is `O(Y1-1971 + Y2-1971)`.
4. The `daysBetweenDates` function calls `calcDays` twice and calculates the absolute difference between the returned values, which is an O(1) operation.
5. Combining all the above, the overall time complexity is `O(Y1-1971 + Y2-1971)`.

Space Complexity:

1. The space used by the algorithm is constant since only a fixed amount of extra space is needed for the variables and arrays regardless of the input:
   - The `days` array in `daysInMonth`, which has constant size of 12.
   - The integer variables used throughout the functions for calculations.
2. There is no variable space allocation that depends on the size of the input, so the space complexity is O(1).