2409. Count Days Spent Together

String Easy

## **Problem Description**

and departure dates if they overlap at all.

when they will leave. The objective is to calculate the total number of days both Alice and Bob will be in Rome simultaneously. We are provided with four dates: arriveAlice and leaveAlice for Alice's visit, and arriveBob and leaveBob for Bob's visit. Each date is given as a string in the "MM-DD" format, where "MM" represents the month and "DD" represents the day. The problem specifies that the time period we're considering occurs within the same year, and importantly, it is not a leap year,

Alice and Bob are both visiting Rome on separate business trips, and we're given specific dates for when they will arrive and

so February has 28 days. The number of days per month can be represented as [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31, 30, 31, 30, 31, 30, 31, 30, 31, 30, 31, 30, 31, 30, 31, 30, 31] corresponding to January through December, respectively.

We need to return the number of days that Alice and Bob will be in the city at the same time. This is inclusive of both their arrival

The intuition behind the provided solution is to find the common range of dates when both Alice and Bob are in Rome and then

Determine the latest arrival date between Alice and Bob. This is done by comparing arriveAlice and arriveBob and taking

## calculate the length of this overlap. The approach includes the following steps:

the maximum of the two. This marks the start of the period when both are in Rome. Determine the earliest leave date between Alice and Bob by comparing leaveAlice and leaveBob and taking the minimum of

- the two. This marks the end of the period when both are in Rome.
- With these boundaries (start and end), calculate the total number of days they overlap. This requires converting the dates into a cumulative day count from the start of the year and then finding the difference between the start and end dates.
- The day count for each date is found by summing the total number of days in the months leading up to the given month using the provided array for the days in each month, then adding the day of the month from the date. Once we have the day counts for both the start and end of the overlap, we calculate the difference. If the latest arrival is after
- overlap, we use the max function to return zero in such cases. If there is an overlap, we add 1 to the difference since both the start and end dates are inclusive.

the earliest departure, this would result in a negative number, indicating no overlap. Since we cannot have negative days of

By using these steps, we efficiently find the number of days two intervals overlap, which corresponds to the number of days Alice and Bob are in Rome simultaneously.

The implementation of the solution uses straightforward calculations and built-in Python functions to compute the number of

overlapping days. Below are the specific steps detailed with the algorithms and data structures used: **Determine Overlap Start and End:** 

• a and b are used to represent the start and end of the overlapping period.

which is appropriate for the constant values representing the days in each month.

Since the problem states the dates are inclusive, 1 is added to the difference.

• Alice's travel dates: arriveAlice = "02-08" and leaveAlice = "02-15"

• Bob's travel dates: arriveBob = "02-09" and leaveBob = "02-18"

Thus, the overlap starts on "02-09" and ends on "02-15".

**Prepare Data Structure for Days per Month:** 

Prepare Data Structure for Days per Month:

o The solution defines two variables, a and b, using the built-in max and min functions to calculate the latest arrival date and the earliest departure date, respectively.

**Convert Dates to Day Counts:** To convert the dates in "MM-DD" format to a cumulative count of days from the start of the year, the solution uses string slicing and the sum

function.

Solution Approach

∘ For example, sum(days[: int(a[:2]) - 1]) calculates the total days up to the start of the month represented by a. It slices the days tuple up to the month index (subtracting 1 since Python uses 0-indexing) and sums the values. Then, it adds the day of the month from a (or b for the end date): int(a[3:]).

A tuple named days is used to store the number of days in each month for a non-leap year. This is a fixed-size, immutable data structure,

∘ With the day counts for the start (x) and end (y) of the overlap determined, the difference y - x gives the number of days in between these dates, not including the start date.

**Example Walkthrough** 

Following the solution approach:

**Calculate Overlapping Days:** 

**Calculate Overlapping Days:** 

calculated overlap or  $\emptyset$  if the overlap calculation is negative:  $max(y - x + 1, \emptyset)$ . By working through these steps, the solution effectively transforms the date range comparison problem into one of simple

arithmetic and takes advantage of Python's powerful built-in functions and data structures to do so efficiently.

To ensure there is no negative count of overlap (which would represent no overlap at all), the max function is used to return either the

Let's walk through a small example to illustrate the solution approach: Assume we have the following travel dates for Alice and Bob:

**Determine Overlap Start and End:** Compare Alice's and Bob's arrival dates, find the latest arrival date: arriveBob is "02-09", which is later than arriveAlice.

• Compare Alice's and Bob's leave dates, find the earliest leave date: leaveAlice is "02-15", which is earlier than leaveBob.

## **Convert Dates to Day Counts:**

us 6 + 1 = 7.

Solution Implementation

46.

**Python** 

class Solution:

○ To calculate the day count for "02-09" (arriveBob), we sum the days for January and add the days in February up to the 9th: 31 + 9 = 40. ○ To calculate the day count for "02-15" (leaveAlice), we sum the days for January and add the days in February up to the 15th: 31 + 15 =

# Retrieve the earlier departure date between Alice and Bob

# Calculate the day of the year for the later arrival date

# Calculate the day of the year for the earlier departure date

# Calculate and return the number of shared days between Alice and Bob

shared\_days = max(departure\_day\_of\_year - arrival\_day\_of\_year + 1, 0)

// Array to store the number of days in each month considering a non-leap year.

\* Calculates the number of days Alice and Bob spend together.

\* @param arriveAlice Arrival date of Alice in "MM-DD" format.

\* @param leaveBob Departure date of Bob in "MM-DD" format.

\* @param arriveBob Arrival date of Bob in "MM-DD" format.

\* @return The number of days they spend together.

// Determine the later arrival date of the two.

// Determine the earlier leave date of the two.

int startDay = convertToDateInYear(laterArrival);

int endDay = convertToDateInYear(earlierLeave);

// Convert the dates to the day of the year.

public int countDaysTogether(

\* @param leaveAlice Departure date of Alice in "MM-DD" format.

private int[] daysInMonth = new int[] {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

String arriveAlice, String leaveAlice, String arriveBob, String leaveBob) {

// An array storing the number of days in each month, assuming it is not a leap year

const latestArrival = arriveAlice < arriveBob ? arriveBob : arriveAlice;</pre>

const earliestDeparture = leaveAlice < leaveBob ? leaveAlice : leaveBob;</pre>

const dayOfYearEarliestDeparture = convertDateToDayOfYear(earliestDeparture);

return Math.max(0, dayOfYearEarliestDeparture - dayOfYearLatestArrival + 1);

\* Converts a date in MM-DD format to a number representing the day of the year

console.log(countDaysTogether('08-15', '08-18', '08-16', '08-20'));

# Retrieve the later arrival date between Alice and Bob

# Retrieve the earlier departure date between Alice and Bob

# Calculate the day of the year for the later arrival date

# Calculate the day of the year for the earlier departure date

# Calculate and return the number of shared days between Alice and Bob

shared\_days = max(departure\_day\_of\_year - arrival\_day\_of\_year + 1, 0)

days\_in\_months = (31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31)

later\_arrival = max(arrive\_alice, arrive\_bob)

earlier\_departure = min(leave\_alice, leave\_bob)

# Tuple containing the number of days in each month

// Calculate the number of days they spend together and ensure it's not negative

const dayOfYearLatestArrival = convertDateToDayOfYear(latestArrival);

\* Calculates the number of days Alice and Bob spend together based on their arrival and departure dates

function countDaysTogether(arriveAlice: string, leaveAlice: string, arriveBob: string, leaveBob: string): number {

// Accumulate the total number of days from the beginning of the year to the end of the previous month

def countDaysTogether(self, arrive\_alice: str, leave\_alice: str, arrive\_bob: str, leave\_bob: str) -> int:

arrival\_day\_of\_year = sum(days\_in\_months[:int(later\_arrival[:2]) - 1]) + int(later\_arrival[3:])

departure day of year = sum(days in months[:int(earlier departure[:2]) - 1]) + int(earlier departure[3:])

const daysInMonth = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31];

\* @param arriveAlice - The arrival date for Alice in MM-DD format

\* @param arriveBob — The arrival date for Bob in MM—DD format

\* @returns The number of days Alice and Bob spend together

\* @returns The day of the year based on the date provided

function convertDateToDayOfYear(date: string): number {

// Parse the month and day from the date string

const [month, day] = date.split('-').map(Number);

// Find the later of the two arrival dates

// Convert dates to day of the year

\* @param date - The date in MM-DD format

for (let i = 0; i < month - 1; ++i) {

// Find the earlier of the two departure dates

\* @param leaveBob - The departure date for Bob in MM-DD format

\* @param leaveAlice - The departure date for Alice in MM-DD format

String laterArrival = arriveAlice.compareTo(arriveBob) < 0 ? arriveBob : arriveAlice;

String earlierLeave = leaveAlice.compareTo(leaveBob) < 0 ? leaveAlice : leaveBob;

days\_in\_months = (31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31)

earlier\_departure = min(leave\_alice, leave\_bob)

# Tuple containing the number of days in each month

o The overlap is 7 days, meaning Alice and Bob will be in Rome simultaneously for a period of 7 days.

By executing these steps with the given dates and applying the logic and calculations described in the solution approach, we find that Alice and Bob will be in Rome at the same time for a total of 7 days.

○ The difference between the day counts is 46 - 40 = 6. Since both the start and end dates are inclusive, we add 1 to this difference, giving

def countDaysTogether(self, arrive\_alice: str, leave\_alice: str, arrive\_bob: str, leave\_bob: str) -> int: # Retrieve the later arrival date between Alice and Bob later\_arrival = max(arrive\_alice, arrive\_bob)

departure\_day\_of\_year = sum(days\_in\_months[:int(earlier\_departure[:2]) - 1]) + int(earlier\_departure[3:])

arrival\_day\_of\_year = sum(days\_in\_months[:int(later\_arrival[:2]) - 1]) + int(later\_arrival[3:])

# Add 1 because if they arrive and leave on the same day, they have spent 1 day together

## return shared\_days Java

class Solution {

/\*\*

\*/

```
// Calculate the total number of days they spend together.
       // Ensure it does not result in a negative number if there is no overlap.
       return Math.max(endDay - startDay + 1, 0);
    /**
    * Converts a date string "MM-DD" to the day of the year.
     * @param date String formatted as "MM-DD".
     * @return The day of the year.
    private int convertToDateInYear(String date) {
       // Parse the month from the date string and adjust for 0-based index.
       int monthIndex = Integer.parseInt(date.substring(0, 2)) - 1;
       int dayOfYear = 0;
       // Add the number of days in the months preceding the given month.
        for (int i = 0; i < monthIndex; ++i) {
            dayOfYear += daysInMonth[i];
       // Add the day of the month to the total.
       dayOfYear += Integer.parseInt(date.substring(3));
       return dayOfYear;
C++
#include <vector>
#include <string>
class Solution {
public:
    std::vector<int> daysInMonth = {31, 28, 31, 30, 31, 30, 31, 30, 31, 30, 31};
    // Calculates the number of days Alice and Bob spend together based on their arrival and departure dates
   int countDaysTogether(std::string arriveAlice, std::string leaveAlice, std::string arriveBob, std::string leaveBob) {
       // Find the later of the two arrival dates
        std::string latestArrival = arriveAlice < arriveBob ? arriveBob : arriveAlice;</pre>
       // Find the earlier of the two departure dates
       std::string earliestDeparture = leaveAlice < leaveBob ? leaveAlice : leaveBob;</pre>
       // Convert dates to day of the year
       int dayOfYearLatestArrival = convertDateToDayOfYear(latestArrival);
        int dayOfYearEarliestDeparture = convertDateToDayOfYear(earliestDeparture);
       // Calculate the number of days they spend together and ensure it's not negative
       return std::max(0, dayOfYearEarliestDeparture - dayOfYearLatestArrival + 1);
    // Converts a date in MM-DD format to a number representing the day of the year
    private:
    int convertDateToDayOfYear(std::string date) {
        int month, day;
       // Parse the month and day from the date string
        sscanf(date.c_str(), "%d-%d", &month, &day);
        int dayOfYear = 0;
       // Accumulate the total number of days from the beginning of the year to the end of the previous month
        for (int i = 0; i < month - 1; ++i) {
            dayOfYear += daysInMonth[i];
       // Add the days in the current month
       dayOfYear += day;
        return dayOfYear;
};
```

```
dayOfYear += daysInMonth[i];
// Add the days in the current month
dayOfYear += day;
return dayOfYear;
```

class Solution:

// Example usage:

let dayOfYear = 0;

return shared\_days

departure dates.

**Time Complexity** 

**TypeScript** 

/\*\*

/\*\*

Time and Space Complexity The given Python code calculates the number of overlapping days that Alice and Bob spend together based on their arrival and

# Add 1 because if they arrive and leave on the same day, they have spent 1 day together

formats). • Slicing and accessing elements of the date strings: These operations are also constant in time since dates are fixed-length strings. • Two sum operations: The days tuple is a fixed size (12 elements representing the days in each month), so summing over a slice of this tuple is

The time complexity of this code can be considered to be 0(1). Here is the breakdown of operations:

- Since all the operations above are done in constant time and do not depend on the size of the input, the overall time complexity is 0(1).
- **Space Complexity** The space complexity of the function can also be considered 0(1) because:

The tuple days has a fixed size of 12.

0(1).

also a constant-time operation.

• There are only a fixed number of integer and string variables, a, b, x, and y.

• Integer conversion and arithmetic operations are again done in constant time.

 No additional data structures that grow with the input size are being used. Therefore, the amount of memory used by the program is constant, irrespective of the input, leading to a space complexity of

• Two max and min operations on dates: These operations take constant time since the operations are performed on fixed-length strings (date