2437. Number of Valid Clock Times

String Enumeration Easy

# **Problem Description**

include the character ?, which stands for an unknown digit that you'll need to replace with a number from 0 to 9. Your task is to determine how many unique valid times can be generated by replacing every ? in the time string. A valid time is one that is between "00:00" and "23:59" inclusive. The problem is essentially asking to calculate the count of all possible valid times that can be created from the given string by

You are given a string time representing the current time on a digital clock in the 24-hour format "hh:mm". The string time can

substituting the ? with appropriate digits, while also respecting the restrictions of time format, where the hours range from 00 to 23 and the minutes range from 00 to 59. For example, if the input time is "0?:4?", the first ? can be replaced by any digit from 0 to 9 to still make a valid hour (since 00

to 09 are all valid hours), and the second? can be replaced by any digit from 0 to 9 to make valid minutes (since 40 to 49 are all valid minutes). You must count all such valid combinations. ntuition

The key intuition behind the solution is to handle the hour and minute parts of the time independently since they have different

#### valid ranges. The first two characters ("hh") of time can range from 00 to 23, while the last two characters ("mm") can range from 00 to 59.

The strategy is to count the number of possibilities for the hour and minute parts separately and then multiply those two counts to get the total number of valid times. For each part, we need to consider two scenarios:

1. The character is a ?, which means it can take any value within the allowed range. 2. The character is a digit, which imposes a constraint on the possible values.

This is achieved by a function f that takes a substring (either the hour or the minute) and the maximum value it can take (24 for

hours, 60 for minutes). It then iterates over all possible values within the range, checks if they are compatible with the given

Here's the breakdown of the implementation:

substring, and counts the number of valid possibilities.

countTime to count the possibilities for each part of the time string (hour and minute).

Variable Assignments: Within the loop, two boolean variables a and b are assigned.

**Solution Approach** The solution approach leverages a simple, yet effective methodology for deciphering the number of valid times that can be created from a string with unknowns represented by ?. The approach uses a nested helper function f inside the main function

#### The helper function f(s: str, m: int) -> int is designed to work with a substring s of the time and a maximum limit m (24 for hours, 60 for minutes), and returns the count of possible numbers that fit within the constraints, matching the given pattern s.

(time[3:]) part of the time and an integer m that is the maximum value (24 for hours, 60 for minutes). The return value is the count of valid numbers for this part of the time. Loop Over Range: A loop runs from 0 to m (exclusive), checking each potential valid integer value within the constraint range. The value i represents the current number being checked for validity against the substring pattern s.

**Nested Function Definition:** The function f takes a substring s representing either the hour (time[:2]) or the minute

- Counting Possibilities: The count cnt increments only if both a and b are True, meaning the number i is valid according to the substring pattern s.
- **Return Total Counts:** For the final result, f is applied to the hours and minutes separately and the respective counts are multiplied. This gives the total number of valid times, as the possibilities for hours and minutes are independent of each other.
  - avoids unnecessary computation by ignoring invalid potential values.

because any such number paired with 1 at the start will form a valid hour (from 10 to 19).

Therefore, there are 10 possible hour combinations when the hour part of time is 1?.

As a result, there are 10 possible minute combinations when the minute part of time is 2?.

generated by replacing each ? with a number from 0 to 9.

every time the value i satisfies the conditions:

• a will be True since the first digit of s is 1, not ?.

a checks if the first character of s is a ? or if it matches the tenth digit of i.

b checks if the second character of s is a ? or if it matches the unit digit of i.

Data structures used are minimal in this approach, with a primary focus on the algorithm and logical checks. There aren't any complex patterns - just straightforward comparison logic and a loop to tally up the valid combinations.

This methodology is both efficient and sufficient, as it directly counts the valid possibilities without having to materialize them. It

that can be made by replacing ? with digits from 0 to 9. **Example Walkthrough** 

Let's use the time "17:27" to illustrate the solution approach. We need to determine how many unique valid times can be

By applying this solution approach to the countTime function, the code effectively calculates the number of distinct valid times

**Analyzing the Hours** First, we look at the hour part 1?. Since the first digit is 1, the second digit (which is ?) can be any number from 0 to 9,

Using the helper function f for hours (s = "1?" and m = 24), the function will loop from 0 to 23. It will increase the count cnt

## • b will be True for any value of i from 10 to 19 because ? can be any digit.

**Analyzing the Minutes** 

class Solution:

Java

class Solution {

Now we look at the minute part 2?. Since the first digit is 2, the second digit (which is ?) can also be any number from 0 to 9, forming a valid minute (from 20 to 29).

every time the value i satisfies the conditions: • a will be True because the first minute digit is 2, not ?. • b will be True for any value of i from 20 to 29.

We multiply the possibilities for the hours (10) by the possibilities for the minutes (10), which gives us a total of  $10 \times 10 = 100$ 

In this walkthrough, we saw how to use the solution approach to find the total possible valid times. We separately calculated the

possibilities for each segment of the time pattern and then combined them to find the overall number of valid combinations.

Using the helper function f for minutes (s = "2?" and m = 60), the function will loop from 0 to 59. It will increase the count cnt

Solution Implementation

def count time(self, time: str) -> int:

for i in range(max value):

unique valid times that can be generated from the time 17:27.

**Combining Hour and Minute Possibilities** 

**Python** 

def count valid\_combinations(part: str, max\_value: int) -> int:

# Increment count if both characters match

# Separating the time string into hours and minutes parts

count += matches\_first\_char and matches\_second\_char

# Helper function to count the valid numbers for a part of the time (hour/minute)

# Check if the first character is '?' or matches the tens place of 'i'

# Check if the second character is '?' or matches the ones place of 'i'

// This method calculates the number of valid times that can be represented by the given string

// Split the time string into hours and minutes and pass to helper method

int possibleHours = calculatePossibilities(time.substring(0, 2), 24);

int possibleMinutes = calculatePossibilities(time.substring(3), 60);

auto countMatches = [ ](string pattern, int maxValue) {

// Apply the lambda to the hour portion of the time

int hourMatches = countMatches(time.substr(0, 2), 24);

// Apply the lambda to the minute portion of the time

int minuteMatches = countMatches(time.substr(3, 2), 60);

// Helper function to count the valid numbers that can replace the '?'

// Check if the first character matches or is a '?'

// Check if the second character matches or is a '?'

// `timeSegment` is the part of the time string ('HH' or 'MM')

// Check if the first character matches or is a wildcard

// Check if the second character matches or is a wildcard

count += isMatchFirstChar && isMatchSecondChar;

// Increment count if both characters match or are wildcards

// Return the product of the matches for hour and minute as the total count

const countValidCombinations = (timeSegment: string, maxValue: number): number => {

matches first char = part[0] == "?" or (int(part[0]) == i // 10)

matches second char = part[1] == "?" or (int(part[1]) == i % 10)

# Return the product of the count of valid combinations for hours and minutes

# Increment count if both characters match

# Separating the time string into hours and minutes parts

count += matches\_first\_char and matches\_second\_char

# Check if the second character is '?' or matches the ones place of 'i'

return count\_valid\_combinations(hour\_part, 24) \* count\_valid\_combinations(minute\_part, 60)

let validCount = 0; // to keep track of the count of valid combinations

bool isMatchFirstChar = pattern[0] == '?' || pattern[0] - '0' == i / 10;

bool isMatchSecondChar = pattern[1] == '?' || pattern[1] - '0' == i % 10;

// `maxValue` is the maximum value that the time segment can have (24 for hours, 60 for minutes)

for (int i = 0; i < maxValue; ++i) {</pre>

int count = 0;

return count;

**}**;

**TypeScript** 

return hourMatches \* minuteMatches;

for (let i = 0; i < maxValue; ++i) {</pre>

function countTime(time: string): number {

# Iterate through all possible values based on maximum for hour/minute

matches first char = part[0] == "?" or (int(part[0]) == i // 10)

matches second char = part[1] == "?" or (int(part[1]) == i % 10)

# Return the product of the count of valid combinations for hours and minutes return count\_valid\_combinations(hour\_part, 24) \* count\_valid\_combinations(minute\_part, 60)

## hour part = time[:2] minute\_part = time[3:]

public int countTime(String time) {

return count

count = 0

```
// Total combinations are the product of possibilities for hours and minutes
        return possibleHours * possibleMinutes;
    // Helper method to calculate the number of valid possibilities for a given time component
    private int calculatePossibilities(String timeComponent, int maxValue) {
        int count = 0;
        // Loop through all possible values for the given time component
        for (int i = 0; i < maxValue; ++i) {</pre>
            // Check if the first character matches or is a wildcard '?'
            boolean firstCharMatches = timeComponent.charAt(0) == '?' || timeComponent.charAt(0) - '0' == i / 10;
            // Check if the second character matches or is a wildcard '?'
            boolean secondCharMatches = timeComponent.charAt(1) == '?' || timeComponent.charAt(1) - '0' == i % 10;
            // Increment the count if both characters match the current possibility
            count += (firstCharMatches && secondCharMatches) ? 1 : 0;
        return count; // Return the total count of valid possibilities
C++
class Solution {
public:
    /**
     * Count the number of valid times that can be represented by the given time string.
     * @param time A time string that includes wildcards '?'.
     * @return The number of valid times that the input can represent.
    int countTime(string time) {
        // Define a lambda function that counts the possible valid numbers for the given pattern
```

```
const matchesSecondChar = timeSegment[1] === '?' || timeSegment[1] === (i % 10).toString();
           // Increment the count if both characters match or are '?'
           if (matchesFirstChar && matchesSecondChar) {
               ++validCount;
       return validCount; // return the final count of valid combinations
   };
   // Extract the hour and minute segments from the input time string
   const hoursSegment = time.slice(0, 2); // 'HH'
   const minutesSegment = time.slice(3); // 'MM'
   // Count valid hour and minute combinations and multiply them to get total valid time combinations
   const hourCombinations = countValidCombinations(hoursSegment, 24);
   const minuteCombinations = countValidCombinations(minutesSegment, 60);
   // The total number of possible valid times is the product of the number of
   // valid hour combinations and the number of valid minute combinations.
   return hourCombinations * minuteCombinations;
class Solution:
   def count time(self, time: str) -> int:
       # Helper function to count the valid numbers for a part of the time (hour/minute)
       def count valid_combinations(part: str, max_value: int) -> int:
           count = 0
           # Iterate through all possible values based on maximum for hour/minute
            for i in range(max value):
               # Check if the first character is '?' or matches the tens place of 'i'
```

const matchesFirstChar = timeSegment[0] === '?' || timeSegment[0] === Math.floor(i / 10).toString();

## 1. To calculate the number of valid hours when the input is the first two characters of the string corresponding to hours. 2. To calculate the number of valid minutes when the input is the last two characters of the string corresponding to minutes.

For minutes: range(60)

The function f is called twice:

**Time Complexity** 

return count

hour part = time[:2]

minute\_part = time[3:]

Time and Space Complexity

In both cases, f iterates over a fixed range: • For hours: range(24)

The first call to f considers all possible hour values, which in a 24-hour format, are 00 to 23, totaling 24 possibilities. The second

The given Python function countTime primarily consists of two nested calls to a helper function f. The helper function is

responsible for counting the number of valid permutations for given position constraints in a digital clock format.

Within the function f, there are constant-time operations being performed, such as comparison and arithmetic operations. These constant-time operations do not depend on the size of the input and thus have a time complexity 0(1).

call to f considers all possible minute values, which are from 00 to 59, totaling 60 possibilities.

Since we call f twice, once for hours and once for minutes, and each call has a 0(1) time complexity, our total time complexity for function countTime is also 0(1).

Therefore, the time complexity of f is 0(24) for hours and 0(60) for minutes, which are both considered constant, and we can

# The space complexity of the function countTime is determined by the space used by variables inside the function that are

**Space Complexity** 

simplify this to O(1) complexity for each call of f.

required to perform the computation. The helper function f uses a few local variables (like cnt, a, and b) to store temporary calculation results. These variables

require a constant amount of space that does not depend on the size of the input; hence, they have a space complexity of 0(1).

The outer function countTime similarly uses only a constant amount of space, calling f twice and storing the results in a return statement.

All variables are of primitive data types, and there are no data structures used that would grow with the size of the input. Therefore, the overall space complexity of the countTime function is 0(1) as well.