1904. The Number of Full Rounds You Have Played

Medium Math String

Problem Description

The aim of the given problem is to compute the number of complete chess rounds played in an online tournament based on the login and logout times provided. Chess rounds commence every 15 minutes, starting from 00:00 and then at 00:15, 00:30, 00:45, and so on until 23:45. When a user logs in and logs out, we want to determine how many of these 15-minute rounds have fully passed. A loginTime is given which represents the time the user logs in to the game.

- A logoutTime is also provided, which represents the time the user logs out of the game.
- If the logoutTime is earlier than loginTime, this suggests the user played from loginTime through midnight to the next day, until logoutTime.
- A single chess round is counted only if the user is logged in for the entire duration of the 15-minute round. The task is to calculate the number of such complete rounds played.

For example, if the loginTime is 00:00 and logoutTime is 00:30, even though the user has been online for 30 minutes, they have played only 2 full rounds (00:00 and 00:15), because they logout before the round starting at 00:30 could complete.

Intuition

Convert both loginTime and logoutTime to minutes. This simplification is done because working with minutes is easier than

working with hours and minutes separately. It also helps to compare and perform arithmetic operations on the times effectively.

The solution approach involves a series of steps to compute the total number of complete rounds:

- Account for the scenario when logoutTime is earlier than loginTime. This indicates that the user played through midnight. To manage this, 24 hours (1440 minutes) are added to logoutTime to properly calculate the number of rounds.
- Rounds are started every 15 minutes, but we are only interested in complete rounds. Therefore, for the loginTime that is inbetween rounds, we need to round it up to the next round start time. On the other side, for the logoutTime, we need to round it down to the closest round start time that has been completed.
- Finally, the total number of complete rounds is the difference between the rounded times, divided by 15 (since each round is 15 minutes). We need to make sure not to count negative values in case logoutTime is right after loginTime, hence, we use
- we're only counting the complete chess rounds. The code provided is a direct translation of this intuition. It uses simple arithmetic operations and logical reasoning to find the solution without additional data structures or complicated algorithms, ensuring an efficient and clear solution to the problem.

By rounding the loginTime and logoutTime to the closest round boundaries and then calculating the difference, we ensure that

Solution Approach

The solution's implementation follows a direct and methodical approach, utilizing the built-in functions of the coding language

the total number of minutes. It does so by multiplying the first two characters (hours) by 60 and adding to the last two

A helper function named get is defined, which takes a string s representing time in the format "HH:MM" and converts this to

characters (minutes), both of which are converted to integers.

max(0, finish - start).

and basic arithmetic. Here's how the algorithm unfolds:

arithmetic operations to calculate the desired result.

We then check if the startTime is greater than the finishTime. If it is, it means the player has played overnight, and therefore we add 24 hours (1440 minutes) to the finishTime.

Next, we handle the round start and end times. For the startTime (now in minutes), we round up to the nearest round start

round (e.g., 00:07 or 00:14), they miss that round and can only start playing in the next one. The division and multiplication

time by adding 14 minutes before dividing by 15 and multiplying by 15. This is because if the player logs in at any time within a

The difference between the finish and start values gives us the total number of rounds. But, we must ensure that this number

The startTime and finishTime are then converted to minutes using the helper function get.

account for this, ensuring that the minimum number of rounds returned is zero.

- discard the remainder if any, effectively rounding up to the next multiple of 15. For the finishTime, we round down to the last complete round by simply dividing by 15. This is because if the player logs out anytime during a round, that round does not count as it is not completed.
- The combination of helper function, careful addition for overnight sessions, and rounding the times to the nearest round boundaries are crucial in this implementation. There's no need for complex data structures; the entire solution leverages

is not negative, which is possible if the finishTime immediately follows the startTime. We use max(0, finish - start) to

Example Walkthrough Let's go through a small example to illustrate the solution approach. Assume the following login and logout times:

Following the steps of the solution: Convert loginTime and logoutTime to minutes: • get("22:47") would result in 22 * 60 + 47 = 1367 minutes.

Since logoutTime is on the next day (01:12 is after midnight), we add 1440 minutes (a full day) to logoutTime to get the

correct duration:

• loginTime: 22:47

• logoutTime: 01:12

 Updated logoutTime in minutes: 72 + 1440 = 1512 minutes. Round the times to the nearest chess round start times:

Calculate the difference in rounded times and convert to the number of complete rounds:

- ∘ For loginTime, rounded up to the next round: (1367 + 14) / 15 * 15 = 1380 minutes. (This corresponds to the round that starts at 23:00.) ∘ For logoutTime, rounded down to the last complete round: 1512 / 15 * 15 = 1510 minutes. (This corresponds to the round that starts at

01:00.)

complete rounds.

Python

class Solution:

converting the given times into minutes, adjusting for overnight play, rounding the times to the nearest round boundaries, and computing the difference. This example cleanly demonstrates the efficiency of the solution approach in evaluating the number of

def convert_to_minutes(time_str: str) -> int:

Convert start and finish times to minutes

if start_minutes > finish_minutes:

start_minutes = convert_to_minutes(start_time)

finish_minutes = convert_to_minutes(finish_time)

start_minutes_rounded = (start_minutes + 14) // 15

Calculate the number of complete 15-minute rounds

Ensure that it's not negative by using max(0, ...)

finish_minutes_rounded = finish_minutes // 15

def number_of_rounds(self, start_time: str, finish_time: str) -> int:

return hours * 60 + minutes # Return total minutes

Define a function to convert time from "HH:MM" format to minutes

hours = int(time_str[:2]) # Extract hours and convert to integer

If start time is after finish time, assume finish time is the next day

The 14 is added before division to ensure proper rounding up for start time

// Calculates the number of full rounds that can be played between the start and finish times

finishMinutes += 24 * 60; // Add 24 hours in minutes to the finish time

// Rounds start time to the next quarter hour if not already rounded

// Calculate the difference in the number of quarter hours played

// If the finish time is less than the start time, it means the finish time is on the next day

int numberOfRounds(string startTime, string finishTime) {

int finishMinutes = convertToMinutes(finishTime);

// Convert start and finish times to minutes

startMinutes = (startMinutes + 14) / 15 * 15;

finishMinutes = finishMinutes / 15 * 15;

// and ensure the result is not negative

// Calculate the number of full 15-minute rounds

def convert_to_minutes(time_str: str) -> int:

Convert start and finish times to minutes

if start_minutes > finish_minutes:

start_minutes = convert_to_minutes(start_time)

finish_minutes = convert_to_minutes(finish_time)

Round start time up to the next 15-minute mark

start minutes rounded = (start minutes + 14) // 15

return rounds > 0 ? rounds : 0;

class Solution:

const rounds = Math.floor(finishMinutes / 15) - Math.ceil(startMinutes / 15);

hours = int(time_str[:2]) # Extract hours and convert to integer

If start time is after finish time, assume finish time is the next day

The 14 is added before division to ensure proper rounding up for start time

minutes = int(time_str[3:]) # Extract minutes and convert to integer

// Return positive number of full rounds or 0 if the result is negative

def number_of_rounds(self, start_time: str, finish_time: str) -> int:

return hours * 60 + minutes # Return total minutes

finish_minutes += 24 * 60 # Add 24 hours in minutes

and round finish time down to the previous 15-minute mark

Define a function to convert time from "HH:MM" format to minutes

// Rounds finish time down to the previous quarter hour

return max(0, (finishMinutes - startMinutes) / 15);

if (startMinutes > finishMinutes) {

int startMinutes = convertToMinutes(startTime);

minutes = int(time_str[3:]) # Extract minutes and convert to integer

Number of complete rounds: 130 / 15 = 8 full rounds.

○ Difference in minutes: 1510 - 1380 = 130 minutes.

• get("01:12") would result in 1 * 60 + 12 = 72 minutes.

Solution Implementation

So, between 22:47 and 01:12 the next day, the user has played 8 complete chess rounds. We arrived at this conclusion by

finish_minutes += 24 * 60 # Add 24 hours in minutes # Round start time up to the next 15-minute mark # and round finish time down to the previous 15-minute mark

```
complete_rounds = max(0, finish_minutes_rounded - start_minutes_rounded)
       return complete_rounds
Java
class Solution {
    // Method to calculate the number of rounds played, given a start and finish time.
   public int numberOfRounds(String startTime, String finishTime) {
       // Convert start and finish times to minutes
       int startInMinutes = convertToMinutes(startTime);
       int finishInMinutes = convertToMinutes(finishTime);
       // If the finish time is less than the start time, it indicates the game went past midnight.
       if (startInMinutes > finishInMinutes) {
            finishInMinutes += 24 * 60; // Add 24 hours (in minutes) to the finish time to handle overnight duration
       // The start time is rounded up to the next 15-minute mark, and the finish time is rounded down.
       startInMinutes = (startInMinutes + 14) / 15 * 15;
        finishInMinutes = (finishInMinutes / 15) * 15;
       // Calculate the difference in 15-minute rounds and return the maximum of 0 or the computed rounds.
       return Math.max(0, finishInMinutes / 15 - startInMinutes / 15);
   // Helper method to convert a time String "HH:MM" to minutes.
   private int convertToMinutes(String time) {
       // Parse the hours and minutes from the time String and convert to minutes
       int hours = Integer.parseInt(time.substring(0, 2));
        int minutes = Integer.parseInt(time.substring(3));
       return hours * 60 + minutes;
```

C++

public:

class Solution {

```
private:
   // Helper function to convert time in "HH:MM" format to total number of minutes
   int convertToMinutes(const string& time) {
       int hours, minutes;
       sscanf(time.c_str(), "%d:%d", &hours, &minutes); // Parse the string for hours and minutes
       return hours * 60 + minutes; // Convert time to minutes
};
TypeScript
/**
* Helper function to convert a time string to its equivalent total minutes.
* @param {string} time - Time in the format of "HH:MM".
* @returns {number} Numeric representation of time in minutes.
function toMinutes(time: string): number {
   // Split the time string into hours and minutes and convert them into numbers
   const [hours, minutes] = time.split(':').map(Number);
   // Calculate the total minutes
   return hours * 60 + minutes;
/**
* Calculates the number of full 15-minute rounds that can be played
* between a start time and a finish time.
* @param {string} startTime - Start time in the format of "HH:MM".
* @param {string} finishTime - Finish time in the format of "HH:MM".
* @returns {number} The number of full 15-minute rounds that can be played.
function numberOfRounds(startTime: string, finishTime: string): number {
   // Convert start and finish times to minutes
    let startMinutes = toMinutes(startTime),
        finishMinutes = toMinutes(finishTime);
   // If the start time is after finish time, add 24 hours to finish time
   if (startMinutes > finishMinutes) {
       finishMinutes += 24 * 60; // Add one full day in minutes
```

finish_minutes_rounded = finish_minutes // 15 # Calculate the number of complete 15-minute rounds # Ensure that it's not negative by using max(0, ...) complete_rounds = max(0, finish_minutes_rounded - start_minutes_rounded)

return complete_rounds

Time and Space Complexity

Time Complexity

The time complexity of the number of Rounds function is o(1). This is because the function consists of a few arithmetic operations and conditional checks, which are all constant time operations. The function calculates the minutes for the start and finish times, then computes the rounds by dividing by 15. The constants do not scale with the size of the input, as the input is always a time in HH:MM format.

Space Complexity

The space complexity of the number of Rounds function is also o(1). This function uses only a fixed number of integer variables to store the start and finish times, and for the intermediate calculations. It does not allocate any additional space that grows with the input size, resulting in constant space complexity.