2582. Pass the Pillow

Easy Math Simulation

Problem Description

In this problem, we have `n` people standing in a line numbered from `1` to `n`. The unique feature here is an ongoing game where a pillow is being passed among them. The game starts with the first person and the pillow is passed one person at a time every second in a linear fashion. When the pillow reaches the last person, it changes direction and starts to go back towards the first person. This back and forth continues perpetually with no end. The challenge is to calculate the position of the person holding the pillow after a given number of seconds, referred to as `time`.

It's important to understand the flow of the game here: It is akin to a ping-pong ball going back and forth on a linear table. Each traverse from the start to the end and back to the start constitutes what can be thought of as a "round" in the game.

The task is to determine who exactly is holding the pillow after a specified `time` has passed, with just two elements in hand - the number of people `n` and the passage of `time`.

Intuition

To tackle this problem, we immediately recognize that the core of it is repetitive patterns over time. With every `n - 1` seconds, the pillow returns to the start or the last person, depending on the direction it's currently moving in. This realization helps us understand that time can be broken down into chunks of `n - 1` seconds, after which the system essentially resets.

Understanding this cyclical nature, we can calculate the number of complete cycles (`k`) and the remainder, which is the position of the extra passes (`mod`). This is done through simple integer division and modulo operations. From there, the direction of pillow travel after complete cycles is key to finding the final position. If `k` is even, the pillow is moving towards the end, and if `k` is odd, it's moving towards the start.

Now we look at the remainder `mod`. If the pillow is moving towards the end (`k` is even), the person holding the pillow will be `mod + 1` (because we index from 1, not 0). If it's moving towards the start (`k` is odd), the person holding the pillow will be `n - mod`. The solution hinges on these insights, and the code is simply an implementation of this logic.

By encapsulating the process in a mathematical formula, we achieve a solution that operates in constant time O(1), which is tremendously more efficient than simulating the entire process in O(time) complexity.

Solution Approach

The implementation of the solution follows a mathematical approach rather than a simulation. This choice is rooted in a desire for efficiency as simulating the pillow passing process could take linear time, which is not optimal when `time` is a large number.

Algorithm

The algorithm used is quite straightforward once the pattern in the pillow passing game is recognized:

1. Calculate the number of complete passes (each pass spanning `n - 1` seconds) and the remaining seconds after these complete passes using integer division and modulo operations. This is done through the `divmod` function in Python which returns a tuple containing the quotient and the remainder.
2. Determine the direction of the pillow passing for the current pass (whether it's going towards the end or towards the start of the line). This is derived from whether the number of complete rounds `k` is even or odd.
3. Based on the direction and the remainder, calculate the position of the person who will be holding the pillow after the specified time.

Data Structures

In this solution, there are no special data structures used. The algorithm operates with simple integers and therefore requires no more than a few variables to execute.

Patterns

The following pattern is used:

- **Modulo arithmetic:** Helps cycle through the people in a linear fashion and to represent the cyclical nature of the pillow passing.
- **Even-odd parity:** Determines the direction of the pillow passing, using the parity of the number of complete cycles.

Code Explanation

The solution code is a direct application of the identified pattern:

```
class Solution:
    def passThePillow(self, n: int, time: int) -> int:
        # Calculate the number of complete rounds 'k' and the remainder 'mod'
        k, mod = divmod(time, n - 1)

        # If 'k' is even, the direction is towards the end, and the position is 'mod + 1'
        # If 'k' is odd, the direction is towards the start, and the position is 'n - mod'
        return n - mod if k & 1 else mod + 1
```

In the code, `k & 1` is used as a quick check for odd parity, as checking the least significant bit is a common way to check if an integer is odd (`k & 1` would be `1` for odd numbers).

In conclusion, the solution leverages mathematical properties of cycles to deduce the position of the person with the pillow at any given `time`, making it an elegant and efficient approach to the problem.

Example Walkthrough

Let's consider `n = 5` people in the line and `time = 13` seconds to illustrate the solution approach. Here's how we apply the algorithm step by step:

1. Calculate the complete rounds 'k' and the remainder 'mod':
   - Since there are 5 people, each full round takes `5 - 1 = 4` seconds.
   - After 13 seconds, `k = 13 // 4 = 3` complete rounds have occurred.
   - The remainder `mod = 13 % 4 = 1` second is the extra time after the last complete round.
2. Determine the direction of the pillow passing:
   - The number of complete rounds `k = 3` is odd, indicating that after 3 complete rounds, the pillow is moving towards the start.
3. Based on the direction and the remainder, calculate the position:
   - We determined that the pillow is moving towards the start (`k` is odd).
   - With a remainder of `mod = 1` and `n = 5`, the final position is `n - mod (5 - 1) = 4`.

Using this approach, we can conclude that after 13 seconds, the 4th person in the line will be holding the pillow.

Solution Implementation

Python

```
class Solution:
    def passThePillow(self, num_people: int, time: int) -> int:
        # Find how many full cycles are there and what's the remainder time after the cycles
        full_cycles, remainder_time = divmod(time, num_people - 1)

        # If the number of full cycles is odd, the game is moving in reverse
        if full_cycles % 2 == 1:
            # Compute the current holder's position during reverse passing
            # If remainder_time is 0, it means the pillow is at the starting position
            # So the last person will have the pillow, otherwise calculate from the end
            final_position = num_people - remainder_time
        else:
            # If the number of full cycles is even, the game is moving normally
            # The current holder's position will be remainder_time + 1,
            # since positions are 1-indexed
            final_position = remainder_time + 1

        return final_position

# Example:
# num_people = 4, time = 7
# full_cycles = 1 (one full reverse cycle), remainder_time = 3
# final_position = 4 - 3 = 1 (1-indexed)
```

Java

```
class Solution {

    /**
     * Determines the final position of the pillow after a specified time.
     *
     * @param totalParticipants the number of participants in the game.
     * @param totalTime the total time for which the pillow is passed.
     * @return the final position of the pillow after the specified time.
     */
    public int passThePillow(int totalParticipants, int totalTime) {
        // Compute the number of complete rounds that have been made
        int completeRounds = totalTime / (totalParticipants - 1);

        // Compute the remaining time after the complete rounds
        int remainingTime = totalTime % (totalParticipants - 1);

        // If the number of complete rounds is odd, the direction of passing is reversed
        // Since the passing starts with participant 1, the final position will be counted backwards
        // from the total number of participants
        if ((completeRounds & 1) == 1) {
            return totalParticipants - remainingTime;
        } else {
            // If the number of complete rounds is even, the final position will be counted forward
            // The '+1' is required to adjust the position to a 1-indexed based
            return remainingTime + 1;
        }
    }
}
```

C++

```
class Solution {
public:
    // Function to find the final position of the pillow after 'time' passes
    int passThePillow(int numPeople, int time) {
        // Determine how many complete rounds the pillow makes
        int completeRounds = time / (numPeople - 1);

        // Calculate the remaining time after the complete rounds
        int remainingTime = time % (numPeople - 1);

        // If the number of complete rounds is odd, the direction of passing is reversed
        if (completeRounds % 2 == 1) {
            // The pillow is moving in the reverse direction with (n - remainingTime) people to pass
            // If remainingTime is 0, pillow stays at the starting person, and position is numPeople
            return remainingTime == 0 ? numPeople : numPeople - remainingTime;
        } else {
            // The pillow is moving in the forward direction with remainingTime people to pass
            // If remainingTime is 0, pillow stays at the starting person, and position is 1
            return remainingTime == 0 ? 1 : remainingTime + 1;
        }
    }
};
```

TypeScript

```
function passThePillow(numberOfPlayers: number, musicTime: number): number {
    // Determine the number of full cycles by dividing the music time by the number of turns
    const fullCycles = Math.floor(musicTime / (numberOfPlayers - 1));

    // Calculate the remaining time after the full cycles
    const remainingTime = musicTime % (numberOfPlayers - 1);

    // If the number of full cycles is odd, the pillow moves in reverse order
    // Calculate the player position based on the remaining time
    if (fullCycles % 2 === 1) {
        return numberOfPlayers - remainingTime;
    } else {
        // If the number of full cycles is even, the pillow moves in forward order
        // Calculate the player position based on the remaining time
        return remainingTime + 1;
    }
}
```

```
class Solution:
    def passThePillow(self, num_people: int, time: int) -> int:
        # Find how many full cycles are there and what's the remainder time after the cycles
        full_cycles, remainder_time = divmod(time, num_people - 1)

        # If the number of full cycles is odd, the game is moving in reverse
        if full_cycles % 2 == 1:
            # Compute the current holder's position during reverse passing
            # If remainder_time is 0, it means the pillow is at the starting position
            # So the last person will have the pillow, otherwise calculate from the end
            final_position = num_people - remainder_time
        else:
            # If the number of full cycles is even, the game is moving normally
            # The current holder's position will be remainder_time + 1,
            # since positions are 1-indexed
            final_position = remainder_time + 1

        return final_position

# Example:
# num_people = 4, time = 7
# full_cycles = 1 (one full reverse cycle), remainder_time = 3
# final_position = 4 - 3 = 1 (1-indexed)
```

Time and Space Complexity

The given code calculates which person will have the pillow after a certain amount of time in a game where the pillow is passed around in a circle. The function `passThePillow` computes the position using basic arithmetic operations.

- Time Complexity: The `divmod` function that is used to calculate both the quotient `k` and remainder `mod` of `time` divided by `n - 1` is an inbuilt Python function operating in constant time. After `divmod`, there are only a handful of basic operations (addition, subtraction, and bitwise operation). These operations also take constant time. Therefore, the overall time complexity of the function is `O(1)`.
- Space Complexity: The function only uses a fixed amount of extra space to store the variables `k` and `mod`, and no additional space that scales with the input size is used. Therefore, the space complexity of the function is `O(1)`.