1503. Last Moment Before All Ants Fall Out of a Plank

Medium Brainteaser Array Simulation

Leetcode Link

Problem Description

In this problem, we are given a wooden plank with a length of `n` units. There are ants on the plank that can move either to the left or the right at a pace of 1 unit per second. When two ants meet, they turn around without losing any time and continue walking in the opposite direction. Also, when an ant gets to one end of the plank, it falls off immediately.

We're tasked with determining at what time the last ant will fall off the plank. We are provided with two arrays `left` and `right`. The `left` array represents the positions of the ants that are moving to the left, and the `right` array represents the positions of the ants that are moving to the right. The goal is to return the moment when the last ant or ants fall out of the plank.

It's worth noting that the meeting of two ants doesn't affect the overall timing for when an ant will fall off the plank. This happens because two ants colliding and turning around is essentially the same as if they passed through each other. We can ignore the meetings between ants and just consider their initial positions and the distances they have to travel to fall off the end of the plank.

Intuition

The key insight for solving this problem is understanding that when two ants collide and turn around, it does not affect the time it takes for them to drop off the ends. It's counterintuitive because you might think collisions would change their paths, and this could complicate things. In reality, you can imagine that the ants "pass through" each other without altering their pace. This means the problem simplifies to finding the longest amount of time any single ant needs to reach an end of the plank.

For ants moving to the left, the time it takes for each of them to fall off is just their starting position value, as they are that many units away from the left end, and they move at one unit per second.

For ants moving to the right, the time it takes to fall off the plank is the distance between their starting position and the right end of the plank, which is `n - position`.

To find when the last ant falls off, we calculate these times for all ants, and the maximum time calculated will be the time when the last ant falls off. This approach leads us to the provided solution.

Solution Approach

The implementation of the solution directly follows the intuition we've outlined previously. The algorithm is quite simple and straightforward, employing neither complex data structures nor patterns. It uses a basic iteration over two arrays and finds the maximum time from both `left` and `right` ant arrays.

The mechanism goes as follows:

1. Initialize a variable `ans` to zero. This will hold the maximum time it takes for any ant to fall off the plank.
2. Iterate through each position in the `left` array. These ants are moving to the left, so the time it will take for each ant to fall off is simply the value of their starting position (`x`). Update `ans` to be the maximum of the current `ans` and `x`.
3. Repeat a similar process for the `right` array. These ants are moving to the right, so the time it takes them to drop off is the distance to the right end of the plank (`n - x`). Update `ans` to be the maximum of current `ans` and `n - x`.

This approach is based on the principle that each ant's fall-off time is individual and does not depend on interactions with other ants. Thus, we can consider each ant's time independently and select the maximum.

Finally, return `ans`, which is the last moment when any ant falls from the plank.

This method is efficient because it runs in linear time: `O(m + n)`, where `m` and `n` are the lengths of the `left` and `right` arrays, respectively. It's important to point out that no additional space is required, other than the input arrays and a few variables (`ans` and `x`), so the space complexity is `O(1)`.

That's all there is to the solution. It's a great example of how seemingly complex problems can sometimes have surprisingly simple solutions if analyzed correctly.

Example Walkthrough

Let's take a small example to illustrate the solution approach. Consider a wooden plank of length `n = 7` units. There are ants at positions `left = [2, 4]` moving to the left and ants at positions `right = [5, 6]` moving to the right. We want to find out when the last ant will fall off the plank.

According to our solution approach, we don't need to pay attention to the ants' collisions, only their starting positions and the direction they're moving. Here's how we will calculate the times:

1. Set `ans` to 0 to keep track of the maximum time required for an ant to fall off.
2. Iterate through the `left` array:
   - For the ant at position 2, it takes 2 seconds to fall off as it's 2 units away from the left end. Now `ans` is `max(0, 2)` which is 2.
   - For the ant at position 4, it takes 4 seconds to fall off. Now `ans` is `max(2, 4)` which is 4.
3. Iterate through the `right` array:
   - For the ant at position 5, it takes `7 - 5 = 2` seconds to fall off because it's 2 units away from the right end. `ans` remains 4 since `max(4, 2) = 4`.
   - For the ant at position 6, it takes `7 - 6 = 1` second to fall off. `ans` remains 4 as `max(4, 1) = 4`.

Therefore, we find that no ant will require more than 4 seconds to fall off the plank. Hence, the last ant falls off at the 4-second mark. The answer to this problem instance is `4`.

Python Solution

```python
class Solution:
    def getLastMoment(self, length: int, left_ants: List[int], right_ants: List[int]) -> int:
        # Initialize the variable to store the maximum amount of time
        max_time = 0

        # Iterate through the ants moving to the left
        # The time for each ant to fall off the plank is the ant's position (x)
        for ant_position in left_ants:
            max_time = max(max_time, ant_position)

        # Iterate through the ants moving to the right
        # The time for each ant to fall off the plank is the plank's length minus the ant's position (length - x)
        for ant_position in right_ants:
            max_time = max(max_time, length - ant_position)

        # Return the maximum time for all ants to fall off the plank
        return max_time
```

Java Solution

```java
class Solution {
    // Method to determine the last moment before all ants fall off a plank
    public int getLastMoment(int plankLength, int[] antsGoingLeft, int[] antsGoingRight) {
        int lastMoment = 0; // Initialize the last moment to be 0

        // Iterate over the positions of ants going left and find the maximum distance
        // they need to travel to fall off the plank
        for (int position : antsGoingLeft) {
            lastMoment = Math.max(lastMoment, position);
        }

        // Iterate over the positions of ants going right and calculate the distance
        // they need to travel to reach the end of the plank and fall off.
        for (int position : antsGoingRight) {
            lastMoment = Math.max(lastMoment, plankLength - position);
        }

        // Return the last moment, which is the maximum time taken for any ant to fall off
        return lastMoment;
    }
}
```

C++ Solution

```cpp
#include <vector>
#include <algorithm> // Include algorithm for max function

// Solution class encapsulates the method to determine the last moment.
class Solution {
public:
    // getLastMoment calculates the last moment before all ants fall off a plank.
    // Parameters:
    // - n: the length of the plank.
    // - left: a vector of integers representing positions of ants moving to the left.
    // - right: a vector of integers representing positions of ants moving to the right.
    // Returns the maximum number of seconds before the last ant falls off.
    int getLastMoment(int n, std::vector<int>& left, std::vector<int>& right) {
        int lastMoment = 0; // Initialize the last moment to zero.

        // Iterate over the positions of ants moving to the left.
        for (int position : left) {
            // Update last moment based on the ants moving to the left.
            lastMoment = std::max(lastMoment, position);
        }

        // Iterate over the positions of ants moving to the right.
        for (int position : right) {
            // Update last moment based on the distance of ants moving to the right from the end.
            lastMoment = std::max(lastMoment, n - position);
        }

        return lastMoment; // Return the calculated last moment.
    }
};
```

Typescript Solution

```typescript
/**
 * Finds the last moment when all the ants have fallen off the plank.
 *
 * @param {number} plankLength - The length of the plank.
 * @param {number[]} antsMovingLeft - The positions of ants moving to the left at the start.
 * @param {number[]} antsMovingRight - The positions of ants moving to the right at the start.
 * @return {number} The last moment when an ant falls off the plank.
 */
function getLastMoment(plankLength: number, antsMovingLeft: number[], antsMovingRight: number[]): number {
    let lastMoment = 0; // Initialize the last moment to 0

    // Iterate through ants moving to the left and find the maximum distance to the left end.
    for (const position of antsMovingLeft) {
        lastMoment = Math.max(lastMoment, position);
    }

    // Iterate through ants moving to the right and find the maximum distance to the right end.
    for (const position of antsMovingRight) {
        lastMoment = Math.max(lastMoment, plankLength - position);
    }

    // The last moment will be the maximum distance any of the ants is from its respective end.
    return lastMoment;
}
```

Time and Space Complexity

The time complexity of the provided code is `O(L + R)`, where `L` is the length of the `left` list and `R` is the length of the `right` list. This is because the code iterates once over each list to find the maximum time taken by an ant from the left and right.

The space complexity of the code is `O(1)`, as no additional space is required that is dependent on the input size. The variables used for calculating the answer do not scale with the number of elements in the `left` or `right` lists.