1227. Airplane Seat Assignment Probability

Medium Brainteaser Math Dynamic Programming Probability and Statistics

Leetcode Link

Problem Description

A group of n passengers are boarding an airplane with exactly n seats, each with a seat assigned (numbered from 1 to n). However, the first passenger has lost their ticket, resulting in them choosing a seat at random. Subsequent passengers board the airplane in this order: if their assigned seat is available, they will sit in it; otherwise, if their seat is taken, they too will choose a seat at random from the remaining unoccupied seats. The question posed is to determine the probability that the last (the nth) passenger sits in their own assigned seat number n.

The challenge in solving this problem lies in abstracting from the potentially complex scenarios to a pattern or reasoning that simplifies the computation of the probability.

Intuition

Recursively or iteratively solving the problem for all passengers can become quite complex. However, there is an insight that simplifies this problem drastically.

We start by noticing that the situation of the last person being able to sit in their own seat is influenced solely by the actions of the first person. This is because if the first person picks their own seat (seat 1), then everyone subsequently will find their seat available and will sit in it. However, if the first person picks any seat other than their own or the last person's, there is a possibility that someone will be displaced from their seat. This ripple effect stops only if a displaced person randomly chooses to sit in the seat of the person who displaced them, effectively resolving the conflict that started with the first passenger.

Analyzing the situation simplistically:

- If n equals 1, then the first person is also the last person, and the probability of them sitting in their own seat is 100% (1 or 1.0 as a float).
- For n greater than 1, we rely on the principle that the problem scales down to the first and the last person's choice. In scenarios where n is 2 or more, two outcomes are possible for the last person:
  1. The first person picks their own seat, ensuring everyone else's seat is available to them.
  2. The first person picks the last person's seat, in which case the last person definitely won't sit in their seat.

All other cases in between are symmetrical, as whenever the first person picks a seat other than the last one, they effectively become the new "first" person for the subsequent passengers. Due to the symmetry and equal probability of choosing any seat except their own, the second case is equally as likely as the first one. Therefore, the probability for the last person getting to sit in their own seat when n is greater than 1 is 1/2 or 0.5.

The insightful solution is elegant and straightforward, avoiding the necessity of complex calculations or simulations.

Solution Approach

The implementation of the solution leverages a direct mathematical conclusion rather than building an elaborate algorithm or utilizing complex data structures. Since the probability boils down to a straightforward choice, the function nthPersonGetsNthSeat can be defined using just conditional logic.

Here's what happens in the code implementation:

- It checks if n is equal to 1. If yes, it returns 1, which is the probability of the last person getting their seat when there's only one person and one seat. This is a base case reflecting that when there's only one passenger, they are guaranteed to sit in their own seat.
- If n is greater than 1, it returns 0.5. From the intuition discussed earlier, we know that the problem for any n > 1 boils down to two equally likely events. Either the first person picks their seat by chance (which then causes everyone else to sit in their own seat) or they pick the last person's seat (which ensures that the last person won't get their seat). Since these two events are equally likely, the probability is 0.5.

The solution does not require loops or recursion, nor does it build upon any dynamic programming or greedy approaches. The key is the understanding of the problem's symmetry and the chain reaction that a single choice from the first person creates, which affects only the last person's outcome. It is insightful to note that here, complexity is avoided not through intricate code but through logical deduction.

Example Walkthrough

Let's use a small example to illustrate the solution approach. Consider an airplane with 3 seats and 3 passengers, where each seat is assigned to a specific passenger. For complexity's sake, let's denote the passengers as P1, P2, and P3, and their assigned seats as S1, S2, and S3 respectively. P1 has lost their ticket and so doesn't know they should sit in S1.

When P1 boards the plane, they can choose any of the 3 seats randomly. Let's examine the possible scenarios:

1. **P1 picks S1 (their own seat):**
   - P2 boards and finds S2 available and sits there.
   - P3 boards and finds S3 available and sits there. In this case, the last passenger (P3) sits in their own seat.
2. **P1 picks S2:**
   - P2 boards and finds S2 is taken, so they pick randomly from the remaining seats (S1 and S3).
     - If P2 picks S1, then:
       - P3 boards and finds S3 (their own seat) is available and sits there.
     - If P2 picks S3, then:
       - P3 boards and finds their seat (S3) taken and cannot sit in it. In these sub-scenarios, P3 has a 50% chance of sitting in their own seat (S3).
3. **P1 picks S3:**
   - P3 is the last to board and now has no choice but to sit in a seat that is not their own, as P1 has taken S3. So, P3 does not get to sit in their seat in this case.

As you can see, when P1 picks their own seat (scenario 1), P3 will definitely sit in their seat. However, when P1 picks any seat other than S1, there's a 50% chance that P3 will sit in their seat (scenario 2). And there's also the possibility that P3 won't get to sit in their seat if P1 picks S3 (scenario 3).

The key insight is that regardless of the number of seats, the determinant factor boils down to the first person's choice. There are always two scenarios—either the first person picks their own seat, making everyone else's seat available, or they pick the last person's seat. These two possibilities imply that the final probability for P3, or any last passenger in general, to sit in their seat when there are more than two seats, will always be 0.5.

Hence, the conditional logic for this problem does not require iterating through all passengers or seats:

- If n is 1, return 1.00, as there's no one else to take P1's seat.
- If n is greater than 1, return 0.50, since the outcome for P3 (the last person) has an equal chance of occurring either way.

This example effectively demonstrates the reasoning behind the solution approach.

Python Solution

```python
class Solution:
    def nth_person_gets_nth_seat(self, n: int) -> float:
        # If there's only 1 person, they'll definitely get their own seat (the first seat)
        if n == 1:
            return 1.0
        # In all other cases, the probability that the nth person gets the nth seat is 0.5
        # This is due to the fact that after the first person picks randomly (as the first seat isn't always their own),
        # the problem can either resolve itself or reduce to a smaller version of the same problem.
        # In the end, it's always 0.5 for n > 1 due to symmetry.
        else:
            return 0.5

# Example usage:
# solution = Solution()
# probability = solution.nth_person_gets_nth_seat(100)
# print(probability)  # Output: 0.5

```

Java Solution

```java
class Solution {
    // This method calculates the probability of the nth person getting the nth seat.
    public double nthPersonGetsNthSeat(int n) {
        // If there is only one person, the probability of the first person
        // getting the first seat is 100%.
        if (n == 1) {
            return 1.0;
        } else {
            // For any number of people greater than 1, the probability of the
            // nth person getting the nth seat is always 50%.
            // This is a result of a known probability puzzle where the first person
            // takes a random seat, and each subsequent person takes their own seat if
            // available, or a random seat otherwise. Through detailed analysis, the
            // probability converges to 50% for the last person.
            return 0.5;
        }
    }
}

```

C++ Solution

```cpp
class Solution {
public:
    // The function calculates the probability that the nth person gets the nth seat.
    // According to the problem, if there is one seat, the probability is 100% (or 1).
    // For more than one seat, the probability is always 50% (or 0.5).
    double nthPersonGetsNthSeat(int n) {
        // Check if there's only one person.
        if (n == 1) {
            // If so, the probability that the first person gets the first seat is 100%.
            return 1.0;
        } else {
            // For any number of people greater than one, the probability
            // the nth person gets the nth seat is always 50%.
            return 0.5;
        }
    }
};

```

Typescript Solution

```typescript
// The function calculates the probability that the nth person gets the nth seat.
// According to the problem statement, functionality is as follows:
// If there is one seat, the probability is 100% (or 1).
// For more than one seat, the probability is always 50% (or 0.5).
function nthPersonGetsNthSeat(n: number): number {
    // Check if there's only one person.
    if (n === 1) {
        // If so, the probability that the first person gets the first seat is 100%.
        return 1.0;
    } else {
        // For any number of people greater than one, the probability
        // that the nth person gets the nth seat is always 50%.
        return 0.5;
    }
}

```

Time and Space Complexity

Time Complexity

The time complexity of the function is O(1) because it simply returns a constant value without any loops or recursive calls. It does not depend on the input size n, except for a single conditional check.

Space Complexity

The space complexity is also O(1) as the function uses a fixed small amount of space, only enough to store the return value, which does not change with the input size n.