2753. Count Houses in a Circular Street II

Hard

Leetcode Link

Problem Description

In this problem, we are given a circular street with an upper bound `n` on the number of houses present on that street. Each house has a door that could be either open or closed initially, and it is guaranteed that at least one door is open at the beginning. Our position is initially in front of one of these doors, but we don't know which one it is. We cannot count the houses directly, but we can interact with them through the methods of a `Street` class. These methods allow us to close the current door we're in front of (`closeDoor()`), check if the current door is open (`isDoorOpen()`), and move to the house on the right (`moveRight()`).

Since the street is circular, moving right from the last house brings us back to the first house. The goal is to count the total number of houses on the street and return this count as `ans`.

Intuition

To solve this issue, we need a way to track a full rotation on the circular street without skipping any houses. We can do this by utilizing the given methods of the `Street` class.

Here's the reasoning behind the solution:

1. We begin by moving to the right until we find an open door. This ensures that we are starting our count from a known state, as we are guaranteed that at least one door is initially open.
2. Once we find an open door, we employ a trick: as we move around the circular street, we start closing every open door we encounter. This way, we can use the closed doors as markers.
3. We move from one house to the next (to the right) and keep track of the number of houses we pass. For each house, we check if its door is open. The very first time we encounter a closed door, we know we have completed a full rotation since it's a door we closed ourselves during the current traversal.
4. We can deduce the number of houses based on the number of moves it took until we reached the first closed door on our second encounter.
5. We continuously close doors on our first pass and stop when we find the door we closed in step 2. The reason this works is because the street is circular. When we find the door we closed, it means we have visited all the houses on the street exactly once.
6. To ensure that we do not close all doors and lose our marker, we only close one door (the first open door we find). After that, we just count the houses as we move to the right.
7. The count we get just before finding the closed door we marked is the total number of houses on the street.

By using the above approach, we can count the number of houses on the street without prior knowledge of our starting position.

Solution Approach

The solution approach is based on the idea that we can mark our traversal by closing a door and checking for its status during subsequent moves. The solution consists of the following steps:

1. First, we need to make sure we start counting from an open door. To do this, we move to the right using the `moveRight()` method until we find an open door with the `isDoorOpen()` method.

   ```
   while not street.isDoorOpen():
       street.moveRight()
   ```

   This ensures that we start in a consistent state.

2. Once we find an open door, we close it using the `closeDoor()` method to mark the beginning of our count.
3. Now we establish a loop that will allow us to count each house on the street. We loop for `k` times, where `k` is the maximum bound for the number of houses (the actual number of houses could be lesser).

   ```
   for i in range(1, k + 1):
       street.moveRight()
   ```

   Inside this loop, we move to the next house using `moveRight()`. After moving, we check if the door of the current house is open using `isDoorOpen()`.

4. If we find an open door, it means we have not yet completed a full circle, so we increment our count and close the door. By closing doors as we encounter them, we ensure we'll be able to detect when we've completed one full loop around the street.

   ```
   if street.isDoorOpen():
       ans = i
       street.closeDoor()
   ```

5. When we find a door that is closed (which will only happen after we complete one full circle), we have our `ans`, which is indicative of the number of houses we've passed, including the starting house.
6. At the end of the loop, we return the `ans` variable, which now contains the number of houses on the street.

The only data structure that is implicitly used in this case is the `Street` object provided to us that contains the methods we need to interact with the houses. The algorithm does not use any additional data structures or complex patterns; it relies purely on the ability to modify the state of the street by closing doors and the assumption that you can only complete a full circular loop if you've moved past all houses.

This approach is particularly elegant and efficient because it requires no extra space and runs in O(n) time where n is the actual number of houses—not the upper bound, `k`.

Example Walkthrough

Let's say we have a circular street with a maximum of `k = 5` houses. We do not know our starting position or the exact number of houses, but we do know at least one door is open initially.

Here's a step-by-step walk through an example scenario:

1. We start at an unknown position. We call `moveRight()` and `isDoorOpen()` in a loop to find the first open door.

   Suppose the third door is open (we don't know this; we are just finding the first open door).

   ```
   Starting status: Unknown
   ```

2. We close this door using `closeDoor()`. This marks our starting point.

   ```
   Status after marking: [?, ?, Closed, ?, ?]
   ```

3. We now loop until we've done at most `k` moves. Each move is one step to the right.

   ```
   for i in range(1, 5):  # Since k is 5
       street.moveRight()
   ```

4. Inside this loop, we use `isDoorOpen()` to check the door's status.
5. We find the first house after the closed door with an open door. Let's say the fifth house had an open door.

   ```
   Status while moving: [?, ?, Closed, ?, Open]
   ```

6. We close the door after counting another house.

   ```
   Second closure: [?, ?, Closed, ?, Closed]
   ```

7. We continue moving around the circle and closing open doors. If we encounter a closed door, it indicates we've made a full circle.

   Assume that there were 3 open doors initially and we closed 2 in total. As we loop, we increment our count and close the open doors until we find our marked door.

8. After closing the second door, when we move right, we would find the third door which we have closed in step 2. This tells us that we've made a full circle.

   ```
   Final status: [?, ?, Closed, ?, Closed]
   Count: 3 (2 open doors found and closed, plus the first closed door considered once)
   ```

9. At this point, we stop counting since we've completed a full rotation of the street. Our count is 3, which means there are 3 houses on the street.

We did not have to know where we started or the exact number of houses initially. We were able to deduce it using the given methods and following the approach described. The answer, in this case, would be that there are 3 houses on the street.

Python Solution

```python
# Assuming 'Optional' and 'Street' are already defined elsewhere in the code.
from typing import Optional

class Solution:
    def house_count(self, street: Optional['Street'], k: int) -> int:
        # First, move right until a door is open.
        while not street.isDoorOpen():
            street.moveRight()

        # Initialized answer variable
        answer = 0

        # Then, move right 'k' times to reach the target house.
        for i in range(1, k + 1):
            street.moveRight()
            # If the target house's door is open, set the answer and close the door.
            if street.isDoorOpen():
                answer = i
                street.closeDoor()

        # Return the index of the target house.
        return answer
```

Java Solution

```java
/**
 * Class representing the solution to count houses based on doors that are open.
 */
class Solution {

    /**
     * Counts the number of open doors on a street within a given range.
     *
     * @param street A Street object, representing the street to be checked.
     * @param k The range within which we need to count the open doors.
     * @return The number of open doors within the range or the furthest open door encountered.
     */
    public int countOpenDoors(Street street, int k) {
        // Move to the first open door if the current door is closed
        while (!street.isDoorOpen()) {
            street.moveRight();
        }

        // Initialize the count of open doors
        int openDoorCount = 0;

        // Loop through 'k' doors starting from the current position
        for (int i = 1; i <= k; ++i) {
            street.moveRight(); // Move to the next door
            // Check if the current door is open
            if (street.isDoorOpen()) {
                openDoorCount++; // Update the count with the latest door number
                street.closeDoor(); // Close the current door
            }
        }

        // Return the total count of open doors encountered
        return openDoorCount;
    }
}
```

C++ Solution

```cpp
/**
 * Definition for a street.
 * This class represents a street with multiple doors.
 */
class Street {
public:
    // Constructor that initializes streets with doors.
    Street(vector<int> doors);

    // Closes the current door at the street's position.
    void closeDoor();

    // Checks if the current door at the street's position is open.
    bool isDoorOpen();

    // Moves the street's position one door to the right.
    void moveRight();
};

/**
 * Solution class that works with streets and doors.
 */
class Solution {
public:
    /**
     * Counts the houses on the street until either a door is found open
     * after moving `k` steps to the right or until `k` steps are completed.
     *
     * @param street Pointer to the Street object.
     * @param k The number of doors to check.
     * @return The 1-based index of the first open door, or 0 if no open door is found within `k` steps.
     */
    int houseCount(Street* street, int k) {
        // Find the first open door by moving right until an open door is found.
        while (!street->isDoorOpen()) {
            street->moveRight();
        }

        int answer = 0; // This variable will hold the 1-based index of the open door if found within k moves.

        // Loop from 1 through k to find an open door.
        for (int i = 1; i <= k; ++i) {
            street->moveRight(); // Move to the next door.

            // Check if the current door is open.
            if (street->isDoorOpen()) {
                answer = i; // Upon, store the 1-based index of the current door.
                street->closeDoor(); // Close the door as per the problem's requirement.
                break; // Door found and closed, no need to check further.
            }
        }

        // Return the 1-based index of the open door or 0 if no open door found within k moves.
        return answer;
    }
};
```

Typescript Solution

```typescript
// Assuming that the methods `closeDoor`, `isDoorOpen`, and `moveRight` are globally
// available with the behavior as they would have within the original `Street` class.

// Declare the types for the globally defined methods as if they were part of the class.
declare function closeDoor(): void;
declare function isDoorOpen(): boolean;
declare function moveRight(): void;

/**
 * Function to count the number of houses with open doors up to a distance `k` from the starting position.
 * Assumes that all doors start closed, and that a door will be closed again once counted.
 * @param {Street|null} street - The Street object.
 * @param {number} k - The distance (in terms of doors) to move and check.
 * @returns {number} - The number of open doors encountered within distance `k`.
 */
function houseCount(street: Street | null, k: number): number {
    // If street is null, we wait early as there are no doors to count.
    if (street) return 0;

    // Loop until we find an open door. This assumes that we start at a 'door' and move right to find an open one.
    while (!isDoorOpen()) {
        moveRight();
    }

    // Once an open door is found, start the count.
    let count = 0;
    // Iterate for k doors starting after the open door found above.
    for (let i = 1; i <= k; ++i) {
        moveRight(); // Move to the next door.
        if (isDoorOpen()) {
            // If this door is open, increment the count of open doors found.
            count++;
            // Assume we close the door after counting it.
            closeDoor();
        }
    }
    // Return the total number of open doors found within the specified distance k.
    return count;
}
```

Time and Space Complexity

Time Complexity

The time complexity of the given code depends on two factors: the number of door movements till the first open door is found, and then moving `k` more steps to the right.

First, we have a while loop that keeps moving to the right until it finds the first open door. In the worst-case scenario, if the first open door is `n` steps away, the loop will execute `n` times.

After the first open door is found, the code goes into a for loop which runs exactly `k` times, each time moving one step to the right and checking if the door is open.

Therefore, in the worst-case scenario, the time complexity is `O(n + k)`, where `n` is the number of steps to the first open door and `k` is the steps we move after finding the first open door.

Space Complexity

The space complexity of the code is `O(1)`. It only uses a fixed amount of additional space: the variable `ans` is all that is stored in terms of extra memory usage, apart from the given input and the function calls themselves. The space used by `ans` does not scale with the input size.