2728. Count Houses in a Circular Street

Easy Array Interactive

Problem Description

In this problem, you are given access to a `Street` object that represents a circular street lined with houses. Your goal is to count the total number of houses on this street. The street is circular, meaning that if you continue moving in one direction, you will eventually return to the starting point. A positive integer `k` is also given, which denotes the upper limit of the possible number of houses on the street. This means there are at most `k` houses. Each house has a door that can be either open or closed at the beginning.

To help you count the houses, you have a set of operations you can perform using the `Street` class:

- `openDoor()`: Opens the door of the house you are currently facing.
- `closeDoor()`: Closes the door of the house you are currently facing.
- `isDoorOpen()`: Returns a boolean indicating whether the door of the current house is open.
- `moveRight()`: Moves you to the house to the right of your current position.
- `moveLeft()`: Moves you to the house to the left of your current position.

You need to return an integer `ans` representing the total number of houses on the street.

Intuition

To solve this problem, we need to find a way to mark the houses we have visited to avoid re-counting. Since the street is circular, we need a strategy to identify when we have made a full loop. Here is the intuition for our approach:

1. We can use the doors as markers by opening them: By opening every door as we pass by, we can alter the state of the doors and use this change to identify when we've returned to the starting point.
2. Starting from the first house, we open and then move past `k` houses to the left: Since we know the number of houses is not more than `k`, if we move `k` houses to the left and the doors are still open, it means there are no more than `k` houses. This step guarantees that we are at least one house past our starting point.
3. We then begin counting houses while moving to the left: For each house we pass, we check if the door is open. If the door is open, we close it (to mark that we've visited this house) and increment our count.
4. We stop counting when we encounter a closed door: This indicates that we have circled back to the starting point and thus have counted all the houses on the street.

Solution Approach

The implementation of the solution involves using the `Street` class's methods to manipulate and check the state of the houses' doors as we count them. No additional data structures are used outside of the given `Street` class, and the main algorithm is a loop with conditionals. Here's a detailed walkthrough:

1. We start by making sure all doors on the street have been opened at least once, so we can use their state to check if we've circled back to the starting point. This is done by opening the doors as we move left `k` times, corresponding to the maximum possible number of houses. The code for this is:

```python
for _ in range(k):
    street.openDoor()
    street.moveLeft()
```

2. Now we need to start counting houses, but it's essential to ensure we only count each house exactly once. We start from the current position (which is guaranteed to be past our initial starting point because of step 1) and we initiate a counter `ans = 0`.
3. We then enter a loop where we will continue to move left and count the houses until we find a closed door, which will signal we've completed the loop and returned to the starting point. The loop and counting logic looks like this:

```python
ans = 0
while street.isDoorOpen():
    street.closeDoor()
    street.moveLeft()
    ans += 1
```

   - `while street.isDoorOpen()`: This condition ensures we are only counting houses that have their door opened by our previous step. Once we encounter a closed door, we can infer that we've completed a full circuit of the street.
   - `street.closeDoor()`: As we count each house by incrementing `ans`, we also close its door. This step is vital because it marks the house as counted and prevents recounting if the loop needs to continue.
   - `street.moveLeft()`: After each operation on a door, we move one house to the left.
   - `ans += 1`: We increment our house count each time we close an open door, which correlates to visiting a new house.
4. Finally, once we hit a closed door and exit the loop, we have the total count of houses in `ans`, which is then returned as the solution.

This algorithm assumes that the doors act as markers and that all the doors start in the same initial state (either all open or all closed). The approach would not work in a scenario where doors have a mix of initial states, as the markers would not be reliable. In our given problem, the initial state of the doors is not specified, but since we begin by opening all doors within the bounds of `k`, this creates a consistent initial state that allows the algorithm to function correctly.

Example Walkthrough

To illustrate the solution approach, let's consider a small example where the maximum possible number of houses `k` is 5. We don't know the exact number of houses at the beginning, but we know there can be at most 5 houses on this circular street.

Initially, we don't know the state of the doors, so we will first open all doors as we move to ensure that we can tell when we've completed a full loop:

1. For a `k` of 5, we perform the following operations:

```python
for _ in range(5):
    street.openDoor()
    street.moveLeft()
```

   After running this segment of code, we will have moved to the left 5 times and opened 5 doors. If there are 5 or fewer houses, we've ensured that all doors on the street are open.

2. We now start counting houses, moving left and expecting all doors to be open:

```python
ans = 0
while street.isDoorOpen():
    street.closeDoor()
    street.moveLeft()
    ans += 1
```

   Let's walk through this loop using an example street layout where there are, in fact, just 3 houses. Here's what would happen:

   - We're at house 3 (0-based indexing), the door is open. Close it and move left to house 2. Increment `ans` to 1.
   - At house 2, the door is open. Close it and move left to house 1. Increment `ans` to 2.
   - At house 1, the door is open. Close it and move left to house 0. Increment `ans` to 3.
   - At house 0, the door is open (because we opened it as part of the preparation phase). Close it and if the street had more than 3 houses, we would move left to the next house. However, since there are only 3 houses, when we move left, we reach house 2 again but now the door is closed. This is where the loop ends.

   At this point, since the door is closed, we exit the while loop, and `ans`, which is 3, correctly represents the total number of houses on the street.

The counting stops when we encounter the first closed door, signaling we have circled back to the point we started closing doors, which was the house after our original starting point, completing the count of all unique houses.

Solution Implementation

Python

```python
# Assuming the Street class has been defined with the methods openDoor(), closeDoor(),
# isDoorOpen(), moveRight(), and moveLeft() as specified in the original problem statement.

class Solution:
    def house_count(self, street: Optional[Street], k: int) -> int:
        # Loop k times to open the door and move to the left each time,
        # effectively moving k positions to the left from the starting point.
        for _ in range(k):
            street.openDoor()
            street.moveLeft()

        # Initialize a counter for the number of houses.
        ans = 0

        # Loop until a closed door is found.
        # The loop invariant is that the door where the agent is currently positioned is open.
        while street.isDoorOpen():
            # Close the current door.
            street.closeDoor()
            # Move one position to the left.
            street.moveLeft()
            # Increment the counter for each house visited.
            ans += 1

        # Return the total number of houses visited.
        return ans
```

Java

```java
// Class to solve the problem of counting houses on a Street
class Solution {

    /**
     * Counts the number of consecutive open doors to the left of the starting position
     * after performing a given number of operations on the doors.
     *
     * @param street the Street object representing the street with doors.
     * @param k the number of times to perform the openDoor and moveLeft operations.
     * @return the number of consecutive open doors to the left of the starting position.
     */
    public int houseCount(Street street, int k) {
        // Open doors k times and move left after each opening
        while (k-- > 0) {
            street.openDoor(); // open the door at the current position
            street.moveLeft(); // move to the left (to the previous door)
        }

        int count = 0; // Initialize the counter for the number of open doors
        // Count the number of open doors until a closed door is encountered
        while (street.isDoorOpen()) { // Check if the current door is open
            count++; // Increment the counter for each open door
            street.closeDoor(); // Close the currently open door
            street.moveLeft(); // Move to the left (to the previous door)
        }

        // Return the total count of open doors to the left of the starting position
        return count;
    }
}
```

C++

```cpp
#include <vector> // Include necessary header

/**
 * Definition for a street.
 * Assume the class implementation and methods details are defined somewhere else.
 */
class Street {
public:
    // Constructor that takes a vector of integers representing doors
    Street(std::vector<int> doors);

    // Method to open a door
    void openDoor();

    // Method to close a door
    void closeDoor();

    // Method to check if a door is open
    bool isDoorOpen();

    // Method to move to the door on the right
    void moveRight();

    // Method to move to the door on the left
    void moveLeft();
};

class Solution {
public:
    /**
     * Count the number of houses on a street starting from the end and moving left.
     * It assumes every door corresponds to a house and opening the door signifies visiting the house.
     *
     * @param street Pointer to the Street object we want to operate on.
     * @param k Number of doors to initially open by moving left from the current position.
     * @return Number of houses counted (number of opened doors).
     */
    int countHouses(Street* street, int k) {
        // Open k doors by moving to the left starting from the initial position
        while (k--) {
            street->openDoor(); // Open the door
            street->moveLeft(); // Move to the next door on the left
        }

        // Initialize the count of houses
        int count = 0;

        // Count the doors that are open by moving to the left until a closed door is found
        while (street->isDoorOpen()) { // While the current door is open
            count++; // Increment the house count
            street->closeDoor(); // Close the door
            street->moveLeft(); // Move to the next door to the left
        }

        // Return total number of houses counted
        return count;
    }
};
```

TypeScript

```typescript
// Function to count the number of houses (doors) on a street
// on the left side up to a specified number (k) of moves.
// Assumes the doors array is positioned from right to left.
//
// @param street The Street object representing the street with doors.
// @param k The number of moves to the left before counting begins.
// @return The number of doors that are open.
function countHouses(street: Street | null, steps: number): number {
    // Move 'steps' times to the left, opening doors as we go.
    while (steps-- > 0) {
        street.openDoor();
        street.moveLeft();
    }
    // Counter for open doors.
    let openDoorCount = 0;
    // Count all consecutive open doors starting from the
    // current position moving further left until reaching
    // a closed door.
    while (street.isDoorOpen()) {
        openDoorCount++; // Increment count for each open door.
        street.closeDoor(); // Close the current open door.
        street.moveLeft(); // Move left to the next door.
    }
    // Return the final count of open doors.
    return openDoorCount;
}
```

```python
# Assuming the Street class has been defined with the methods openDoor(), closeDoor(),
# isDoorOpen(), moveRight(), and moveLeft() as specified in the original problem statement.

class Solution:
    def house_count(self, street: Optional[Street], k: int) -> int:
        # Loop k times to open the door and move to the left each time,
        # effectively moving k positions to the left from the starting point.
        for _ in range(k):
            street.openDoor()
            street.moveLeft()

        # Initialize a counter for the number of houses.
        ans = 0

        # Loop until a closed door is found.
        # The loop invariant is that the door where the agent is currently positioned is open.
        while street.isDoorOpen():
            # Close the current door.
            street.closeDoor()
            # Move one position to the left.
            street.moveLeft()
            # Increment the counter for each house visited.
            ans += 1

        # Return the total number of houses visited.
        return ans
```

Time and Space Complexity

The given code represents a solution that effectively quantifies the number of times the `moveLeft` operation can be performed on a `street` object until an open door is no longer detected. This is accomplished by first moving left `k` times and opening doors, then continuing to move left and closing doors until a closed door is encountered, which concludes the counting sequence.

Time Complexity

The time complexity of the code depends on two distinct phases:

1. **The first for-loop**: This loop runs exactly `k` times, each iteration involving two constant-time operations: `openDoor()` and `moveLeft()`. This phase contributes a time complexity of `O(k)`.
2. **The while-loop**: This loop continues until `isDoorOpen()` returns `false`. In the worst-case scenario, this could involve traversing the entire length of the street that was previously traversed during the for-loop. If we assume that `n` is the total distance from the starting point to the point where `isDoorOpen()` returns `false`, then the while-loop contributes a time complexity of `O(n)`.

Combining these two phases, the total worst-case time complexity of the code is `O(k + n)`, where `k` is the number of initial left-move operations and `n` is the number of closed-door checks before reaching a closed door.

Space Complexity

The space complexity of the code is the amount of additional memory used by the algorithm as a function of the input size. Since there are no data structures like arrays, lists, or maps being used to store data and the use of variables is constant regardless of input size:

- The space complexity of the code is `O(1)` since the memory used does not scale with input size but remains constant due to the fixed number of variables and object method calls.