2534. Time Taken to Cross the Door

Leetcode Link

Problem Explanation

In this problem, there are `n` persons who want to enter or exit through a door. Each person can enter or exit through the door once, taking one second. We are given a non-decreasing integer array `arrival` of size `n`, where `arrival[i]` is the arrival time of the `i`th person at the door. Also, there is an array `state` of size `n`, where `state[i]` is 0 if person `i` wants to enter through the door or 1 if they want to exit through the door.

The main goal is to find an array `answer` of size `n` where `answer[i]` is the second at which the `i`th person crosses the door, considering various rules regarding entering and exiting the door.

Example Walkthrough

Consider the example: arrival = `[0, 1, 1, 2, 4]` state = `[0, 1, 0, 0, 1]`

The expected output is `[0, 3, 1, 2, 4]`.

Here's how we can process each second:

- At t = 0: Person 0 is the only one who wants to enter, so they just enter through the door.
- At t = 1: Person 1 wants to exit, and person 2 wants to enter. Since the door was used the previous second for entering, person 2 enters.
- At t = 2: Person 1 still wants to exit, and person 3 wants to enter. Since the door was used the previous second for entering, person 3 enters.
- At t = 3: Person 1 is the only one who wants to exit, so they just exit through the door.
- At t = 4: Person 4 is the only one who wants to exit, so they just exit through the door.

Therefore, the answer is `[0, 3, 1, 2, 4]`.

Solution Approach

We can use two pointers approach for solving this problem. First, we can create two lists, one for the entering persons and one for the exiting persons, and initialize pointers at the beginning of those lists. Next, we can iterate until both pointers reach the end of their respective lists. Finally, we can follow the rules stated in the problem description for entering and exiting persons and update the answer array accordingly.

Python Solution

```python
class Solution:
    def timeTaken(self, arrival: List[int], state: List[int]) -> List[int]:
        n = len(arrival)
        answer = [0] * n
        enter = []
        exit = []

        # Separate entering and exiting persons
        for i in range(n):
            if state[i] == 0:
                enter.append(i)
            else:
                exit.append(i)

        enter_ptr = 0
        exit_ptr = 0
        prev_action = -1
        time = 0

        # Iterate until both pointers reach the end
        while enter_ptr < len(enter) or exit_ptr < len(exit):
            # Check rules for entering and exiting persons
            if (enter_ptr < len(enter) and exit_ptr < len(exit) and
                arrival[enter[enter_ptr]] <= time and arrival[exit[exit_ptr]] <= time):
                if prev_action == 0:
                    answer[enter[enter_ptr]] = time
                    enter_ptr += 1
                    prev_action = 0
                else:
                    answer[exit[exit_ptr]] = time
                    exit_ptr += 1
                    prev_action = 1
            elif enter_ptr < len(enter) and arrival[enter[enter_ptr]] <= time:
                answer[enter[enter_ptr]] = time
                enter_ptr += 1
                prev_action = 0
            elif exit_ptr < len(exit) and arrival[exit[exit_ptr]] <= time:
                answer[exit[exit_ptr]] = time
                exit_ptr += 1
                prev_action = 1
            else:
                prev_action = -1

            time += 1

        return answer
```

*Note: Solutions for Java, JavaScript, C++, and C# are beyond the scope of this prompt, but following the same approach mentioned above, we can create solutions for those languages as well.***JavaScript Solution**

```javascript
class Solution {
    timeTaken(arrival, state) {
        const n = arrival.length;
        const answer = Array(n).fill(0);
        const enter = [];
        const exit = [];

        // Separate entering and exiting persons
        for (let i = 0; i < n; i++) {
            if (state[i] === 0) {
                enter.push(i);
            } else {
                exit.push(i);
            }
        }

        let enter_ptr = 0;
        let exit_ptr = 0;
        let prev_action = -1;
        let time = 0;

        // Iterate until both pointers reach the end
        while (enter_ptr < enter.length || exit_ptr < exit.length) {
            // Check rules for entering and exiting persons
            if (enter_ptr < enter.length && exit_ptr < exit.length &&
                arrival[enter[enter_ptr]] <= time && arrival[exit[exit_ptr]] <= time) {
                if (prev_action === 0) {
                    answer[enter[enter_ptr]] = time;
                    enter_ptr++;
                    prev_action = 0;
                } else {
                    answer[exit[exit_ptr]] = time;
                    exit_ptr++;
                    prev_action = 1;
                }
            } else if (enter_ptr < enter.length && arrival[enter[enter_ptr]] <= time) {
                answer[enter[enter_ptr]] = time;
                enter_ptr++;
                prev_action = 0;
            } else if (exit_ptr < exit.length && arrival[exit[exit_ptr]] <= time) {
                answer[exit[exit_ptr]] = time;
                exit_ptr++;
                prev_action = 1;
            } else {
                prev_action = -1;
            }

            time++;
        }

        return answer;
    }
}
```

Java Solution

```java
import java.util.*;

class Solution {
    public int[] timeTaken(int[] arrival, int[] state) {
        int n = arrival.length;
        int[] answer = new int[n];
        List<Integer> enter = new ArrayList<>();
        List<Integer> exit = new ArrayList<>();

        // Separate entering and exiting persons
        for (int i = 0; i < n; i++) {
            if (state[i] == 0) {
                enter.add(i);
            } else {
                exit.add(i);
            }
        }

        int enter_ptr = 0;
        int exit_ptr = 0;
        int prev_action = -1;
        int time = 0;

        // Iterate until both pointers reach the end
        while (enter_ptr < enter.size() || exit_ptr < exit.size()) {
            // Check rules for entering and exiting persons
            if (enter_ptr < enter.size() && exit_ptr < exit.size() &&
                arrival[enter.get(enter_ptr)] <= time && arrival[exit.get(exit_ptr)] <= time) {
                if (prev_action == 0) {
                    answer[enter.get(enter_ptr)] = time;
                    enter_ptr++;
                    prev_action = 0;
                } else {
                    answer[exit.get(exit_ptr)] = time;
                    exit_ptr++;
                    prev_action = 1;
                }
            } else if (enter_ptr < enter.size() && arrival[enter.get(enter_ptr)] <= time) {
                answer[enter.get(enter_ptr)] = time;
                enter_ptr++;
                prev_action = 0;
            } else if (exit_ptr < exit.size() && arrival[exit.get(exit_ptr)] <= time) {
                answer[exit.get(exit_ptr)] = time;
                exit_ptr++;
                prev_action = 1;
            } else {
                prev_action = -1;
            }

            time++;
        }

        return answer;
    }
}
```

These solutions follow the same approach mentioned above and can be used for the respective programming languages.