Problem 2728: Count Houses in a Circular Street

Problem Information

Difficulty: Easy
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

You are given an object

street

of class

Street

that represents a circular street and a positive integer

k

which represents a maximum bound for the number of houses in that street (in other words, the number of houses is less than or equal to

k

). Houses' doors could be open or closed initially.

Initially, you are standing in front of a door to a house on this street. Your task is to count the number of houses in the street.

The class

Street

contains the following functions which may help you:

void openDoor()

: Open the door of the house you are in front of.

void closeDoor()

: Close the door of the house you are in front of.

boolean isDoorOpen()

: Returns

true

if the door of the current house is open and

false

otherwise.

void moveRight()

: Move to the right house.

void moveLeft()

: Move to the left house.

Return

ans

which represents the number of houses on this street.

Example 1:

Input:

street = [0,0,0,0], k = 10

Output:

4

Explanation:

There are 4 houses, and all their doors are closed. The number of houses is less than k, which is 10.

Example 2:

Input:

street = [1,0,1,1,0], k = 5

Output:

5

Explanation:

There are 5 houses, and the doors of the 1st, 3rd, and 4th house (moving in the right direction) are open, and the rest are closed. The number of houses is equal to k, which is 5.

Constraints:

n == number of houses

1 <= n <= k <= 10

3

Code Snippets

C++:

```
/**
* Definition for a street.
* class Street {
* public:
* Street(vector<int> doors);
* void openDoor();
* void closeDoor();
* bool isDoorOpen();
* void moveRight();
* void moveLeft();
* };
*/
class Solution {
public:
int houseCount(Street* street, int k) {

}
};
```

Java:

```
/**
* Definition for a street.
* class Street {
* public Street(int[] doors);
* public void openDoor();
* public void closeDoor();
* public boolean isDoorOpen();
* public void moveRight();
* public void moveLeft();
* }
*/
class Solution {
public int houseCount(Street street, int k) {

}
}
```

Python3:

```
# Definition for a street.
# class Street:
# def openDoor(self):
# pass
# def closeDoor(self):
# pass
# def isDoorOpen(self):
# pass
# def moveRight(self):
# pass
# def moveLeft(self):
# pass
class Solution:
def houseCount(self, street: Optional['Street'], k: int) -> int:
```

Python:

```
# Definition for a street.
# class Street:
# def openDoor(self):
# pass
# def closeDoor(self):
# pass
# def isDoorOpen(self):
# pass
# def moveRight(self):
# pass
# def moveLeft(self):
# pass
class Solution(object):
def houseCount(self, street, k):
"""
:type street: Street
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* Definition for a street.
* class Street {
* @param {number[]} doors
```

```
* constructor(doors);
*
* @return {void}
* openDoor();
*
* @return {void}
* closeDoor();
*
* @return {boolean}
* isDoorOpen();
*
* @return {void}
* moveRight();
*
* @return {void}
* moveLeft();
* }
*/
/**
* @param {Street} street
* @param {number} k
* @return {number}
*/
var houseCount = function(street, k) {

};
```

TypeScript:

```
/**
* Definition for a street.
* class Street {
* constructor(doors: number[]);
* public openDoor(): void;
* public closeDoor(): void;
* public isDoorOpen(): boolean;
* public moveRight(): void;
* public moveLeft(): void;
* }
*/
function houseCount(street: Street | null, k: number): number {
```

```
};
```

C#:

```
/**
* Definition for a street.
* class Street {
* public Street(int[] doors);
* public void OpenDoor();
* public void CloseDoor();
* public bool IsDoorOpen();
* public void MoveRight();
* public void MoveLeft();
* }
*/
public class Solution {
public int HouseCount(Street street, int k) {

}
}
```

C:

```
/**
* Definition for a street.
*
* YOU HAVE TO PASS THE OBJECT ITSELF AS THE FIRST PARAMETER
*
* struct Street {
* void (*openDoor)(struct Street*);
* void (*closeDoor)(struct Street*);
* bool (*isDoorOpen)(struct Street*);
* void (*moveRight)(struct Street*);
* void (*moveLeft)(struct Street*);
* };
*/
int houseCount(struct Street* street, int k){

}
```

Go:

```
/**
* Definition for a street.
* type Street interface {
* OpenDoor()
* CloseDoor()
* IsDoorOpen() bool
* MoveRight()
* MoveLeft()
* }
*/
func houseCount(street Street, k int) int {

}
```

Kotlin:

```
/**
* Definition for a street.
* class Street(doors: IntArray) {
* fun openDoor()
* fun closeDoor()
* fun isDoorOpen(): Boolean
* fun moveRight()
* fun moveLeft()
* }
*/
class Solution {
fun houseCount(street: Street, k: Int): Int {

}
}
```

Swift:

```
/**
* Definition for a street.
* class Street {
* init(doors: [Int]) {}
* func openDoor() {}
* func closeDoor() {}
* func isDoorOpen() -> Bool {}
* func moveRight() {}
* func moveLeft() {}
```

```
* }
*/
class Solution {
func houseCount(_ street: Street, _ k: Int) -> Int {

}
}
```

Rust:

```
/**
* Definition for a street.
* impl Street {
* pub fn new(doors: Vec<i32>) -> Self {}
* pub fn open_door(&mut self) {}
* pub fn close_door(&mut self) {}
* pub fn is_door_open(&self) -> bool {}
* pub fn move_right(&mut self) {}
* pub fn move_left(&mut self) {}
* }
*/
impl Solution {
pub fn house_count(street: Street, k: i32) -> i32 {

}
}
```

Ruby:

```
# Definition for a street.
# class Street
# def initialize(doors)
# end
# def open_door
# end
# def close_door
# end
# def is_door_open
# end
# def move_right
# end
# def move_left
```

```
# end
# end
# @param {Street} street
# @param {Integer} k
# @return {Integer}
def house_count(street, k)

end
```

PHP:

```
/**
* Definition for a street.
* class Street {
* function __construct($doors);
* function openDoor();
* function closeDoor();
* * @return Boolean
* function isDoorOpen();
* function moveRight();
* function moveLeft();
* }
*/
class Solution {

/**
* @param Street $street
* @param Integer $k
* @return Integer
*/
function houseCount($street, $k) {

}
}
```

Dart:

```
/**
* Definition for a street.
* class Street {
* Street(List<int> doors);
* void openDoor();
```

```
* void closeDoor();
* bool isDoorOpen();
* void moveRight();
* void moveLeft();
* }
*/
class Solution {
int houseCount(Street street, int k) {

}
}
```

Scala:

```
/**
* Definition for a street.
* class Street(doors: Array[Int]) {
* def openDoor(): Unit
* def closeDoor(): Unit
* def isDoorOpen(): Boolean
* def moveRight(): Unit
* def moveLeft(): Unit
* }
*/

object Solution {
def houseCount(street: Street, k: Int): Int = {

}
}
```

Racket:

```
; Definition for a street:
#|

(define street%
(class object%
(super-new)
(init (doors '()))
(define/public (open-door) (-> void?))
(define/public (close-door) (-> void?))
```

```
(define/public (is-door-open) (-> boolean?))
(define/public (move-right) (-> void?))
(define/public (move-left) (-> void?))
))

|#

(define (house-count street k)
;; (-> street? exact-integer? exact-integer?)

)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Houses in a Circular Street
* Difficulty: Easy
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* Definition for a street.
* class Street {
* public:
* Street(vector<int> doors);
* void openDoor();
* void closeDoor();
* bool isDoorOpen();
* void moveRight();
* void moveLeft();
* };
*/
class Solution {
public:
```

```
int houseCount(Street* street, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Count Houses in a Circular Street
* Difficulty: Easy
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* Definition for a street.
* class Street {
* public Street(int[] doors);
* public void openDoor();
* public void closeDoor();
* public boolean isDoorOpen();
* public void moveRight();
* public void moveLeft();
* }
*/
class Solution {
public int houseCount(Street street, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Houses in a Circular Street
Difficulty: Easy
Tags: array, tree
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

# Definition for a street.
# class Street:
# def openDoor(self):
# pass
# def closeDoor(self):
# pass
# def isDoorOpen(self):
# pass
# def moveRight(self):
# pass
# def moveLeft(self):
# pass
class Solution:
def houseCount(self, street: Optional['Street'], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
# Definition for a street.
# class Street:
# def openDoor(self):
# pass
# def closeDoor(self):
# pass
# def isDoorOpen(self):
# pass
# def moveRight(self):
# pass
# def moveLeft(self):
# pass
class Solution(object):
def houseCount(self, street, k):
"""
:type street: Street
:type k: int
```

```
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Count Houses in a Circular Street
* Difficulty: Easy
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* Definition for a street.
* class Street {
* @param {number[]} doors
* constructor(doors);
*
* @return {void}
* openDoor();
*
* @return {void}
* closeDoor();
*
* @return {boolean}
* isDoorOpen();
*
* @return {void}
* moveRight();
*
* @return {void}
* moveLeft();
* }
*/
/**
* @param {Street} street
* @param {number} k
* @return {number}
```

```
*/
var houseCount = function(street, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Houses in a Circular Street
* Difficulty: Easy
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* Definition for a street.
* class Street {
* constructor(doors: number[]);
* public openDoor(): void;
* public closeDoor(): void;
* public isDoorOpen(): boolean;
* public moveRight(): void;
* public moveLeft(): void;
* }
*/
function houseCount(street: Street | null, k: number): number {

};
```

C# Solution:

```
/*
* Problem: Count Houses in a Circular Street
* Difficulty: Easy
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* Definition for a street.
* class Street {
* public Street(int[] doors);
* public void OpenDoor();
* public void CloseDoor();
* public bool IsDoorOpen();
* public void MoveRight();
* public void MoveLeft();
* }
*/
public class Solution {
public int HouseCount(Street street, int k) {

}
}
```

C Solution:

```
/*
* Problem: Count Houses in a Circular Street
* Difficulty: Easy
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* Definition for a street.
*
* YOU HAVE TO PASS THE OBJECT ITSELF AS THE FIRST PARAMETER
*
* struct Street {
* void (*openDoor)(struct Street*);
* void (*closeDoor)(struct Street*);
* bool (*isDoorOpen)(struct Street*);
```

```
* void (*moveRight)(struct Street*);
* void (*moveLeft)(struct Street*);
* };
*/
int houseCount(struct Street* street, int k){

}
```

Go Solution:

```
// Problem: Count Houses in a Circular Street
// Difficulty: Easy
// Tags: array, tree
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

/**
* Definition for a street.
* type Street interface {
* OpenDoor()
* CloseDoor()
* IsDoorOpen() bool
* MoveRight()
* MoveLeft()
* }
*/
func houseCount(street Street, k int) int {

}
```

Kotlin Solution:

```
/**
* Definition for a street.
* class Street(doors: IntArray) {
* fun openDoor()
* fun closeDoor()
* fun isDoorOpen(): Boolean
* fun moveRight()
```

```
* fun moveLeft()
* }
*/
class Solution {
fun houseCount(street: Street, k: Int): Int {

}
}
```

Swift Solution:

```
/**
* Definition for a street.
* class Street {
* init(doors: [Int]) {}
* func openDoor() {}
* func closeDoor() {}
* func isDoorOpen() -> Bool {}
* func moveRight() {}
* func moveLeft() {}
* }
*/
class Solution {
func houseCount(_ street: Street, _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count Houses in a Circular Street
// Difficulty: Easy
// Tags: array, tree
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

/**
* Definition for a street.
* impl Street {
```

```
* pub fn new(doors: Vec<i32>) -> Self {}
* pub fn open_door(&mut self) {}
* pub fn close_door(&mut self) {}
* pub fn is_door_open(&self) -> bool {}
* pub fn move_right(&mut self) {}
* pub fn move_left(&mut self) {}
* }
*/
impl Solution {
pub fn house_count(street: Street, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# Definition for a street.
# class Street
# def initialize(doors)
# end
# def open_door
# end
# def close_door
# end
# def is_door_open
# end
# def move_right
# end
# def move_left
# end
# end
# @param {Street} street
# @param {Integer} k
# @return {Integer}
def house_count(street, k)

end
```

PHP Solution:

```
/**
* Definition for a street.
* class Street {
* function __construct($doors);
* function openDoor();
* function closeDoor();
* * @return Boolean
* function isDoorOpen();
* function moveRight();
* function moveLeft();
* }
*/
class Solution {

/**
* @param Street $street
* @param Integer $k
* @return Integer
*/
function houseCount($street, $k) {

}
}
```

Dart Solution:

```
/**
* Definition for a street.
* class Street {
* Street(List<int> doors);
* void openDoor();
* void closeDoor();
* bool isDoorOpen();
* void moveRight();
* void moveLeft();
* }
*/
class Solution {
int houseCount(Street street, int k) {

}
}
```

Scala Solution:

```
/**
* Definition for a street.
* class Street(doors: Array[Int]) {
* def openDoor(): Unit
* def closeDoor(): Unit
* def isDoorOpen(): Boolean
* def moveRight(): Unit
* def moveLeft(): Unit
* }
*/

object Solution {
def houseCount(street: Street, k: Int): Int = {

}
}
```

Racket Solution:

```
; Definition for a street:
#|

(define street%
(class object%
(super-new)
(init (doors '()))
(define/public (open-door) (-> void?))
(define/public (close-door) (-> void?))
(define/public (is-door-open) (-> boolean?))
(define/public (move-right) (-> void?))
(define/public (move-left) (-> void?))
))

|#

(define (house-count street k)
;; (-> street? exact-integer? exact-integer?)

)
```