Problem 1823: Find the Winner of the Circular Game

Problem Information

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

Problem Description

There are

n

friends that are playing a game. The friends are sitting in a circle and are numbered from

1

to

n

in

clockwise order

. More formally, moving clockwise from the

i

th

friend brings you to the

(i+1)

th

friend for

1 <= i < n

, and moving clockwise from the

n

th

friend brings you to the

1

st

friend.

The rules of the game are as follows:

Start

at the

1

st

friend.

Count the next

k

friends in the clockwise direction

including

the friend you started at. The counting wraps around the circle and may count some friends more than once.

The last friend you counted leaves the circle and loses the game.

If there is still more than one friend in the circle, go back to step

2

starting

from the friend

immediately clockwise

of the friend who just lost and repeat.

Else, the last friend in the circle wins the game.

Given the number of friends,

n

, and an integer

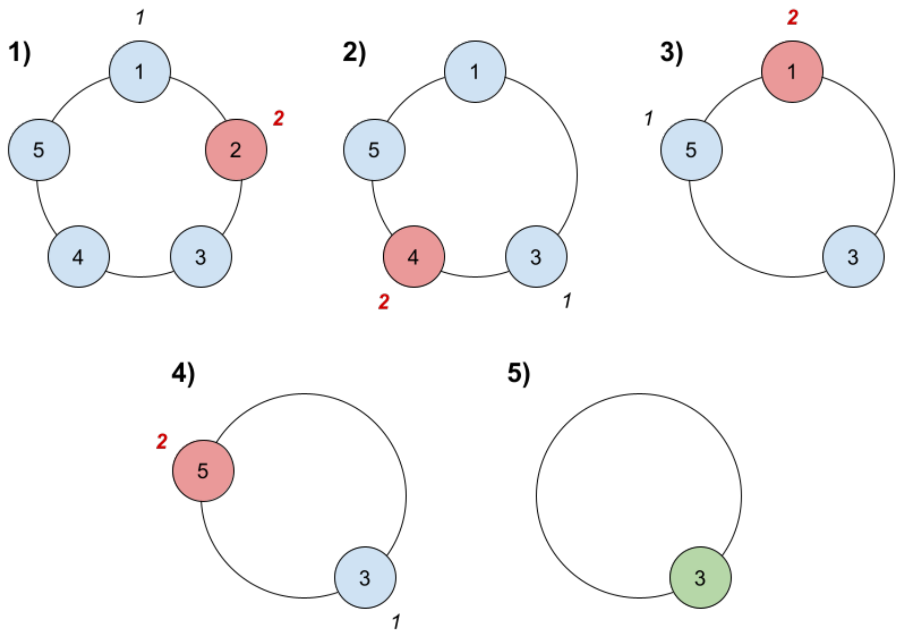
k

, return

the winner of the game

.

Example 1:



Input:

n = 5, k = 2

Output:

3

Explanation:

Here are the steps of the game: 1) Start at friend 1. 2) Count 2 friends clockwise, which are friends 1 and 2. 3) Friend 2 leaves the circle. Next start is friend 3. 4) Count 2 friends clockwise, which are friends 3 and 4. 5) Friend 4 leaves the circle. Next start is friend 5. 6) Count 2 friends clockwise, which are friends 5 and 1. 7) Friend 1 leaves the circle. Next start is friend 3. 8) Count 2 friends clockwise, which are friends 3 and 5. 9) Friend 5 leaves the circle. Only friend 3 is left, so they are the winner.

Example 2:

Input:

n = 6, k = 5

Output:

1

Explanation:

The friends leave in this order: 5, 4, 6, 2, 3. The winner is friend 1.

Constraints:

1 <= k <= n <= 500

Follow up:

Could you solve this problem in linear time with constant space?

Code Snippets

C++:

```
class Solution {
public:
int findTheWinner(int n, int k) {

}
};
```

Java:

```
class Solution {
public int findTheWinner(int n, int k) {

}
}
```

Python3:

```
class Solution:
def findTheWinner(self, n: int, k: int) -> int:
```

Python:

```
class Solution(object):
def findTheWinner(self, n, k):
"""
:type n: int
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number} k
* @return {number}
*/
var findTheWinner = function(n, k) {

};
```

TypeScript:

```
function findTheWinner(n: number, k: number): number {

};
```

C#:

```
public class Solution {
public int FindTheWinner(int n, int k) {

}
}
```

C:

```
int findTheWinner(int n, int k) {

}
```

Go:

```
func findTheWinner(n int, k int) int {
}
```

Kotlin:

```
class Solution {
fun findTheWinner(n: Int, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func findTheWinner(_ n: Int, _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn find_the_winner(n: i32, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer} k
# @return {Integer}
def find_the_winner(n, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer $k
* @return Integer
*/
function findTheWinner($n, $k) {

}
}
```

Dart:

```
class Solution {
int findTheWinner(int n, int k) {

}
}
```

Scala:

```
object Solution {
def findTheWinner(n: Int, k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_the_winner(n :: integer, k :: integer) :: integer
def find_the_winner(n, k) do

end
end
```

Erlang:

```
-spec find_the_winner(N :: integer(), K :: integer()) -> integer().
find_the_winner(N, K) ->
.
```

Racket:

```
(define/contract (find-the-winner n k)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the Winner of the Circular Game
* Difficulty: Medium
* Tags: array, math, queue
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int findTheWinner(int n, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Winner of the Circular Game
* Difficulty: Medium
* Tags: array, math, queue
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int findTheWinner(int n, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the Winner of the Circular Game
Difficulty: Medium
Tags: array, math, queue

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def findTheWinner(self, n: int, k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findTheWinner(self, n, k):
"""
:type n: int
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Find the Winner of the Circular Game
* Difficulty: Medium
* Tags: array, math, queue
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} n
* @param {number} k
* @return {number}
*/
var findTheWinner = function(n, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the Winner of the Circular Game
* Difficulty: Medium
* Tags: array, math, queue
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function findTheWinner(n: number, k: number): number {

};
```

C# Solution:

```
/*
* Problem: Find the Winner of the Circular Game
* Difficulty: Medium
* Tags: array, math, queue
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int FindTheWinner(int n, int k) {

}
}
```

C Solution:

```
/*
* Problem: Find the Winner of the Circular Game
* Difficulty: Medium
* Tags: array, math, queue
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int findTheWinner(int n, int k) {

}
```

Go Solution:

```
// Problem: Find the Winner of the Circular Game
// Difficulty: Medium
// Tags: array, math, queue
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func findTheWinner(n int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun findTheWinner(n: Int, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func findTheWinner(_ n: Int, _ k: Int) -> Int {

}

}
```

Rust Solution:

```
// Problem: Find the Winner of the Circular Game
// Difficulty: Medium
// Tags: array, math, queue
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn find_the_winner(n: i32, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer} k
# @return {Integer}
def find_the_winner(n, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer $k
* @return Integer
*/
function findTheWinner($n, $k) {

}
}
```

Dart Solution:

```
class Solution {
int findTheWinner(int n, int k) {

}
}
```

Scala Solution:

```
object Solution {
def findTheWinner(n: Int, k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_the_winner(n :: integer, k :: integer) :: integer
def find_the_winner(n, k) do

end
end
```

Erlang Solution:

```
-spec find_the_winner(N :: integer(), K :: integer()) -> integer().
find_the_winner(N, K) ->
.
```

Racket Solution:

```
(define/contract (find-the-winner n k)
(-> exact-integer? exact-integer? exact-integer?)
)
```