

Problem 2350: Shortest Impossible Sequence of Rolls

Problem Information

Difficulty: **Hard**

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given an integer array

rolls

of length

n

and an integer

k

. You roll a

k

sided dice numbered from

1

to

k

,

n

times, where the result of the

i

th

roll is

rolls[i]

.

Return

the length of the

shortest

sequence of rolls so that there's no such

subsequence

in

rolls

.

A

sequence of rolls

of length

len

is the result of rolling a

k

sided dice

len

times.

Example 1:

Input:

rolls = [4,2,1,2,3,3,2,4,1], k = 4

Output:

3

Explanation:

Every sequence of rolls of length 1, [1], [2], [3], [4], can be taken from rolls. Every sequence of rolls of length 2, [1, 1], [1, 2], ..., [4, 4], can be taken from rolls. The sequence [1, 4, 2] cannot be taken from rolls, so we return 3. Note that there are other sequences that cannot be taken from rolls.

Example 2:

Input:

rolls = [1,1,2,2], k = 2

Output:

2

Explanation:

Every sequence of rolls of length 1, [1], [2], can be taken from rolls. The sequence [2, 1] cannot be taken from rolls, so we return 2. Note that there are other sequences that cannot be taken from rolls but [2, 1] is the shortest.

Example 3:

Input:

rolls = [1,1,3,2,2,2,3,3], k = 4

Output:

1

Explanation:

The sequence [4] cannot be taken from rolls, so we return 1. Note that there are other sequences that cannot be taken from rolls but [4] is the shortest.

Constraints:

$n == \text{rolls.length}$

$1 \leq n \leq 10$

5

$1 \leq \text{rolls}[i] \leq k \leq 10$

5

Code Snippets

C++:

```
class Solution {
public:
    int shortestSequence(vector<int>& rolls, int k) {
```

```
}  
};
```

Java:

```
class Solution {  
    public int shortestSequence(int[] rolls, int k) {  
  
    }  
}
```

Python3:

```
class Solution:  
    def shortestSequence(self, rolls: List[int], k: int) -> int:
```

Python:

```
class Solution(object):  
    def shortestSequence(self, rolls, k):  
        """  
        :type rolls: List[int]  
        :type k: int  
        :rtype: int  
        """
```

JavaScript:

```
/**  
 * @param {number[]} rolls  
 * @param {number} k  
 * @return {number}  
 */  
var shortestSequence = function(rolls, k) {  
  
};
```

TypeScript:

```
function shortestSequence(rolls: number[], k: number): number {  
  
};
```

C#:

```
public class Solution {  
    public int ShortestSequence(int[] rolls, int k) {  
  
    }  
}
```

C:

```
int shortestSequence(int* rolls, int rollsSize, int k) {  
  
}
```

Go:

```
func shortestSequence(rolls []int, k int) int {  
  
}
```

Kotlin:

```
class Solution {  
    fun shortestSequence(rolls: IntArray, k: Int): Int {  
  
    }  
}
```

Swift:

```
class Solution {  
    func shortestSequence(_ rolls: [Int], _ k: Int) -> Int {  
  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn shortest_sequence(rolls: Vec<i32>, k: i32) -> i32 {  
  
    }  
}
```

Ruby:

```
# @param {Integer[]} rolls
# @param {Integer} k
# @return {Integer}
def shortest_sequence(rolls, k)

end
```

PHP:

```
class Solution {

    /**
     * @param Integer[] $rolls
     * @param Integer $k
     * @return Integer
     */
    function shortestSequence($rolls, $k) {

    }

}
```

Dart:

```
class Solution {
  int shortestSequence(List<int> rolls, int k) {

  }
}
```

Scala:

```
object Solution {
  def shortestSequence(rolls: Array[Int], k: Int): Int = {

  }
}
```

Elixir:

```
defmodule Solution do
  @spec shortest_sequence(rolls :: [integer], k :: integer) :: integer
```

```
def shortest_sequence(rolls, k) do

end

end
```

Erlang:

```
-spec shortest_sequence(Rolls :: [integer()], K :: integer()) -> integer().
shortest_sequence(Rolls, K) ->
.
```

Racket:

```
(define/contract (shortest-sequence rolls k)
  (-> (listof exact-integer?) exact-integer? exact-integer?)
  )
```

Solutions

C++ Solution:

```
/*
 * Problem: Shortest Impossible Sequence of Rolls
 * Difficulty: Hard
 * Tags: array, greedy, hash
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map
 */

class Solution {
public:
    int shortestSequence(vector<int>& rolls, int k) {

    }
};
```

Java Solution:

```

/**
 * Problem: Shortest Impossible Sequence of Rolls
 * Difficulty: Hard
 * Tags: array, greedy, hash
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map
 */

class Solution {
public int shortestSequence(int[] rolls, int k) {

}

}

```

Python3 Solution:

```

"""
Problem: Shortest Impossible Sequence of Rolls
Difficulty: Hard
Tags: array, greedy, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def shortestSequence(self, rolls: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass

```

Python Solution:

```

class Solution(object):
def shortestSequence(self, rolls, k):
"""
:type rolls: List[int]
:type k: int
:rtype: int
"""

```

JavaScript Solution:

```
/**
 * Problem: Shortest Impossible Sequence of Rolls
 * Difficulty: Hard
 * Tags: array, greedy, hash
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map
 */

/**
 * @param {number[]} rolls
 * @param {number} k
 * @return {number}
 */
var shortestSequence = function(rolls, k) {

};
```

TypeScript Solution:

```
/**
 * Problem: Shortest Impossible Sequence of Rolls
 * Difficulty: Hard
 * Tags: array, greedy, hash
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map
 */

function shortestSequence(rolls: number[], k: number): number {

};
```

C# Solution:

```
/*
 * Problem: Shortest Impossible Sequence of Rolls
 * Difficulty: Hard
```

```

* Tags: array, greedy, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int ShortestSequence(int[] rolls, int k) {

}
}

```

C Solution:

```

/*
* Problem: Shortest Impossible Sequence of Rolls
* Difficulty: Hard
* Tags: array, greedy, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int shortestSequence(int* rolls, int rollsSize, int k) {

}

```

Go Solution:

```

// Problem: Shortest Impossible Sequence of Rolls
// Difficulty: Hard
// Tags: array, greedy, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func shortestSequence(rolls []int, k int) int {

```

```
}
```

Kotlin Solution:

```
class Solution {  
    fun shortestSequence(rolls: IntArray, k: Int): Int {  
  
    }  
}
```

Swift Solution:

```
class Solution {  
    func shortestSequence(_ rolls: [Int], _ k: Int) -> Int {  
  
    }  
}
```

Rust Solution:

```
// Problem: Shortest Impossible Sequence of Rolls  
// Difficulty: Hard  
// Tags: array, greedy, hash  
//  
// Approach: Use two pointers or sliding window technique  
// Time Complexity: O(n) or O(n log n)  
// Space Complexity: O(n) for hash map  
  
impl Solution {  
    pub fn shortest_sequence(rolls: Vec<i32>, k: i32) -> i32 {  
  
    }  
}
```

Ruby Solution:

```
# @param {Integer[]} rolls  
# @param {Integer} k  
# @return {Integer}  
def shortest_sequence(rolls, k)
```

```
end
```

PHP Solution:

```
class Solution {  
  
    /**  
     * @param Integer[] $rolls  
     * @param Integer $k  
     * @return Integer  
     */  
    function shortestSequence($rolls, $k) {  
  
    }  
}
```

Dart Solution:

```
class Solution {  
    int shortestSequence(List<int> rolls, int k) {  
  
    }  
}
```

Scala Solution:

```
object Solution {  
    def shortestSequence(rolls: Array[Int], k: Int): Int = {  
  
    }  
}
```

Elixir Solution:

```
defmodule Solution do  
    @spec shortest_sequence(rolls :: [integer], k :: integer) :: integer  
    def shortest_sequence(rolls, k) do  
  
    end  
end
```

Erlang Solution:

```
-spec shortest_sequence(Rolls :: [integer()], K :: integer()) -> integer().  
shortest_sequence(Rolls, K) ->  
.
```

Racket Solution:

```
(define/contract (shortest-sequence rolls k)  
  (-> (listof exact-integer?) exact-integer? exact-integer?)  
  )
```