Problem 1033: Moving Stones Until Consecutive

Problem Information

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

Problem Description

There are three stones in different positions on the X-axis. You are given three integers

a

,

b

, and

c

, the positions of the stones.

In one move, you pick up a stone at an endpoint (i.e., either the lowest or highest position stone), and move it to an unoccupied position between those endpoints. Formally, let's say the stones are currently at positions

x

,

y

, and

z

with

x < y < z

. You pick up the stone at either position

x

or position

z

, and move that stone to an integer position

k

, with

x < k < z

and

k != y

.

The game ends when you cannot make any more moves (i.e., the stones are in three consecutive positions).

Return

an integer array

answer

of length

2

where

:

answer[0]

is the minimum number of moves you can play, and

answer[1]

is the maximum number of moves you can play

.

Example 1:

Input:

a = 1, b = 2, c = 5

Output:

[1,2]

Explanation:

Move the stone from 5 to 3, or move the stone from 5 to 4 to 3.

Example 2:

Input:

a = 4, b = 3, c = 2

Output:

[0,0]

Explanation:

We cannot make any moves.

Example 3:

Input:

a = 3, b = 5, c = 1

Output:

[1,2]

Explanation:

Move the stone from 1 to 4; or move the stone from 1 to 2 to 4.

Constraints:

1 <= a, b, c <= 100

a

,

b

, and

c

have different values.

Code Snippets

C++:

```
class Solution {
public:
vector<int> numMovesStones(int a, int b, int c) {

}
};
```

Java:

```
class Solution {
public int[] numMovesStones(int a, int b, int c) {

}
}
```

Python3:

```
class Solution:
def numMovesStones(self, a: int, b: int, c: int) -> List[int]:
```

Python:

```
class Solution(object):
def numMovesStones(self, a, b, c):
"""
:type a: int
:type b: int
:type c: int
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number} a
* @param {number} b
* @param {number} c
* @return {number[]}
*/
var numMovesStones = function(a, b, c) {
};
```

TypeScript:

```
function numMovesStones(a: number, b: number, c: number): number[] {

};
```

C#:

```
public class Solution {
public int[] NumMovesStones(int a, int b, int c) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* numMovesStones(int a, int b, int c, int* returnSize) {

}
```

Go:

```
func numMovesStones(a int, b int, c int) []int {

}
```

Kotlin:

```
class Solution {
fun numMovesStones(a: Int, b: Int, c: Int): IntArray {

}
}
```

Swift:

```
class Solution {
func numMovesStones(_ a: Int, _ b: Int, _ c: Int) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn num_moves_stones(a: i32, b: i32, c: i32) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer} a
# @param {Integer} b
# @param {Integer} c
# @return {Integer[]}
def num_moves_stones(a, b, c)

end
```

PHP:

```
class Solution {

/**
* @param Integer $a
* @param Integer $b
* @param Integer $c
* @return Integer[]
*/
function numMovesStones($a, $b, $c) {

}
}
```

Dart:

```
class Solution {
List<int> numMovesStones(int a, int b, int c) {
}
}
```

Scala:

```
object Solution {
def numMovesStones(a: Int, b: Int, c: Int): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec num_moves_stones(a :: integer, b :: integer, c :: integer) :: [integer]
def num_moves_stones(a, b, c) do

end
end
```

Erlang:

```
-spec num_moves_stones(A :: integer(), B :: integer(), C :: integer()) ->
[integer()].
num_moves_stones(A, B, C) ->
.
```

Racket:

```
(define/contract (num-moves-stones a b c)
(-> exact-integer? exact-integer? exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Moving Stones Until Consecutive
* Difficulty: Medium
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
vector<int> numMovesStones(int a, int b, int c) {

}
};
```

Java Solution:

```
/**
* Problem: Moving Stones Until Consecutive
* Difficulty: Medium
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int[] numMovesStones(int a, int b, int c) {

}
}
```

Python3 Solution:

```
"""
Problem: Moving Stones Until Consecutive
Difficulty: Medium
Tags: array, dp, math

Approach: Use two pointers or sliding window technique
```

```
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def numMovesStones(self, a: int, b: int, c: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def numMovesStones(self, a, b, c):
"""
:type a: int
:type b: int
:type c: int
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Moving Stones Until Consecutive
* Difficulty: Medium
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number} a
* @param {number} b
* @param {number} c
* @return {number[]}
*/
var numMovesStones = function(a, b, c) {

};
```

TypeScript Solution:

```
/**
* Problem: Moving Stones Until Consecutive
* Difficulty: Medium
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function numMovesStones(a: number, b: number, c: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Moving Stones Until Consecutive
* Difficulty: Medium
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int[] NumMovesStones(int a, int b, int c) {

}
}
```

C Solution:

```
/*
* Problem: Moving Stones Until Consecutive
* Difficulty: Medium
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* numMovesStones(int a, int b, int c, int* returnSize) {

}
```

Go Solution:

```
// Problem: Moving Stones Until Consecutive
// Difficulty: Medium
// Tags: array, dp, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func numMovesStones(a int, b int, c int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun numMovesStones(a: Int, b: Int, c: Int): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func numMovesStones(_ a: Int, _ b: Int, _ c: Int) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Moving Stones Until Consecutive
// Difficulty: Medium
// Tags: array, dp, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn num_moves_stones(a: i32, b: i32, c: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer} a
# @param {Integer} b
# @param {Integer} c
# @return {Integer[]}
def num_moves_stones(a, b, c)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $a
* @param Integer $b
* @param Integer $c
* @return Integer[]
*/
function numMovesStones($a, $b, $c) {

}
}
```

Dart Solution:

```
class Solution {
List<int> numMovesStones(int a, int b, int c) {

}
}
```

Scala Solution:

```
object Solution {
def numMovesStones(a: Int, b: Int, c: Int): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec num_moves_stones(a :: integer, b :: integer, c :: integer) :: [integer]
def num_moves_stones(a, b, c) do

end
end
```

Erlang Solution:

```
-spec num_moves_stones(A :: integer(), B :: integer(), C :: integer()) ->
[integer()].
num_moves_stones(A, B, C) ->
.
```

Racket Solution:

```
(define/contract (num-moves-stones a b c)
(-> exact-integer? exact-integer? exact-integer? (listof exact-integer?))
)
```