Problem 1654: Minimum Jumps to Reach Home

Problem Information

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

Problem Description

A certain bug's home is on the x-axis at position

x

. Help them get there from position

0

.

The bug jumps according to the following rules:

It can jump exactly

a

positions

forward

(to the right).

It can jump exactly

b

positions

backward

(to the left).

It cannot jump backward twice in a row.

It cannot jump to any

forbidden

positions.

The bug may jump forward

beyond

its home, but it

cannot jump

to positions numbered with

negative

integers.

Given an array of integers

forbidden

, where

forbidden[i]

means that the bug cannot jump to the position

forbidden[i]

, and integers

a

,

b

, and

x

, return

the minimum number of jumps needed for the bug to reach its home

. If there is no possible sequence of jumps that lands the bug on position

x

, return

-1.

Example 1:

Input:

forbidden = [14,4,18,1,15], a = 3, b = 15, x = 9

Output:

3

Explanation:

3 jumps forward (0 -> 3 -> 6 -> 9) will get the bug home.

Example 2:

Input:

forbidden = [8,3,16,6,12,20], a = 15, b = 13, x = 11

Output:

-1

Example 3:

Input:

forbidden = [1,6,2,14,5,17,4], a = 16, b = 9, x = 7

Output:

2

Explanation:

One jump forward (0 -> 16) then one jump backward (16 -> 7) will get the bug home.

Constraints:

1 <= forbidden.length <= 1000

1 <= a, b, forbidden[i] <= 2000

0 <= x <= 2000

All the elements in

forbidden

are distinct.

Position

x

is not forbidden.

Code Snippets

C++:

```
class Solution {
public:
int minimumJumps(vector<int>& forbidden, int a, int b, int x) {

}
};
```

Java:

```
class Solution {
public int minimumJumps(int[] forbidden, int a, int b, int x) {

}
}
```

Python3:

```
class Solution:
def minimumJumps(self, forbidden: List[int], a: int, b: int, x: int) -> int:
```

Python:

```
class Solution(object):
def minimumJumps(self, forbidden, a, b, x):
"""
:type forbidden: List[int]
:type a: int
:type b: int
:type x: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} forbidden
* @param {number} a
* @param {number} b
* @param {number} x
* @return {number}
*/
var minimumJumps = function(forbidden, a, b, x) {

};
```

TypeScript:

```
function minimumJumps(forbidden: number[], a: number, b: number, x: number):
number {

};
```

C#:

```
public class Solution {
public int MinimumJumps(int[] forbidden, int a, int b, int x) {

}
}
```

C:

```
int minimumJumps(int* forbidden, int forbiddenSize, int a, int b, int x) {

}
```

Go:

```
func minimumJumps(forbidden []int, a int, b int, x int) int {

}
```

Kotlin:

```
class Solution {
fun minimumJumps(forbidden: IntArray, a: Int, b: Int, x: Int): Int {

}
}
```

Swift:

```
class Solution {
func minimumJumps(_ forbidden: [Int], _ a: Int, _ b: Int, _ x: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_jumps(forbidden: Vec<i32>, a: i32, b: i32, x: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} forbidden
# @param {Integer} a
# @param {Integer} b
# @param {Integer} x
# @return {Integer}
def minimum_jumps(forbidden, a, b, x)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $forbidden
* @param Integer $a
* @param Integer $b
* @param Integer $x
* @return Integer
```

```
*/
function minimumJumps($forbidden, $a, $b, $x) {

}
}
```

Dart:

```
class Solution {
int minimumJumps(List<int> forbidden, int a, int b, int x) {

}
}
```

Scala:

```
object Solution {
def minimumJumps(forbidden: Array[Int], a: Int, b: Int, x: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_jumps(forbidden :: [integer], a :: integer, b :: integer, x ::
integer) :: integer
def minimum_jumps(forbidden, a, b, x) do

end
end
```

Erlang:

```
-spec minimum_jumps(Forbidden :: [integer()], A :: integer(), B :: integer(),
X :: integer()) -> integer().
minimum_jumps(Forbidden, A, B, X) ->
.
```

Racket:

```
(define/contract (minimum-jumps forbidden a b x)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Jumps to Reach Home
* Difficulty: Medium
* Tags: array, dp, search
*
* 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 minimumJumps(vector<int>& forbidden, int a, int b, int x) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Jumps to Reach Home
* Difficulty: Medium
* Tags: array, dp, search
*
* 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 minimumJumps(int[] forbidden, int a, int b, int x) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Minimum Jumps to Reach Home
Difficulty: Medium
Tags: array, dp, search

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 minimumJumps(self, forbidden: List[int], a: int, b: int, x: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minimumJumps(self, forbidden, a, b, x):
"""
:type forbidden: List[int]
:type a: int
:type b: int
:type x: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Jumps to Reach Home
* Difficulty: Medium
* Tags: array, dp, search
*
* 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[]} forbidden
* @param {number} a
* @param {number} b
* @param {number} x
* @return {number}
*/
var minimumJumps = function(forbidden, a, b, x) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Jumps to Reach Home
* Difficulty: Medium
* Tags: array, dp, search
*
* 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 minimumJumps(forbidden: number[], a: number, b: number, x: number):
number {

};
```

C# Solution:

```
/*
* Problem: Minimum Jumps to Reach Home
* Difficulty: Medium
* Tags: array, dp, search
*
* 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 MinimumJumps(int[] forbidden, int a, int b, int x) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Jumps to Reach Home
* Difficulty: Medium
* Tags: array, dp, search
*
* 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
*/

int minimumJumps(int* forbidden, int forbiddenSize, int a, int b, int x) {

}
```

Go Solution:

```
// Problem: Minimum Jumps to Reach Home
// Difficulty: Medium
// Tags: array, dp, search
//
// 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 minimumJumps(forbidden []int, a int, b int, x int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumJumps(forbidden: IntArray, a: Int, b: Int, x: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minimumJumps(_ forbidden: [Int], _ a: Int, _ b: Int, _ x: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Jumps to Reach Home
// Difficulty: Medium
// Tags: array, dp, search
//
// 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 minimum_jumps(forbidden: Vec<i32>, a: i32, b: i32, x: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} forbidden
# @param {Integer} a
# @param {Integer} b
# @param {Integer} x
# @return {Integer}
def minimum_jumps(forbidden, a, b, x)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $forbidden
* @param Integer $a
* @param Integer $b
* @param Integer $x
* @return Integer
*/
function minimumJumps($forbidden, $a, $b, $x) {

}
}
```

Dart Solution:

```
class Solution {
int minimumJumps(List<int> forbidden, int a, int b, int x) {

}
}
```

Scala Solution:

```
object Solution {
def minimumJumps(forbidden: Array[Int], a: Int, b: Int, x: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_jumps(forbidden :: [integer], a :: integer, b :: integer, x ::
integer) :: integer
def minimum_jumps(forbidden, a, b, x) do

end
end
```

Erlang Solution:

```
-spec minimum_jumps(Forbidden :: [integer()], A :: integer(), B :: integer(),
X :: integer()) -> integer().
minimum_jumps(Forbidden, A, B, X) ->
.
```

Racket Solution:

```
(define/contract (minimum-jumps forbidden a b x)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?
exact-integer?)
)
```