Problem 1306: Jump Game III

Problem Information

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

Problem Description

Given an array of non-negative integers

arr

, you are initially positioned at

start

index of the array. When you are at index

i

, you can jump to

i + arr[i]

or

i - arr[i]

, check if you can reach

any

index with value 0.

Notice that you can not jump outside of the array at any time.

Example 1:

Input:

arr = [4,2,3,0,3,1,2], start = 5

Output:

true

Explanation:

All possible ways to reach at index 3 with value 0 are: index 5 -> index 4 -> index 1 -> index 3
index 5 -> index 6 -> index 4 -> index 1 -> index 3

Example 2:

Input:

arr = [4,2,3,0,3,1,2], start = 0

Output:

true

Explanation:

One possible way to reach at index 3 with value 0 is: index 0 -> index 4 -> index 1 -> index 3

Example 3:

Input:

arr = [3,0,2,1,2], start = 2

Output:

false

Explanation:

There is no way to reach at index 1 with value 0.

Constraints:

1 <= arr.length <= 5 * 10

4

0 <= arr[i] < arr.length

0 <= start < arr.length

Code Snippets

C++:

```
class Solution {
public:
bool canReach(vector<int>& arr, int start) {

}
};
```

Java:

```
class Solution {
public boolean canReach(int[] arr, int start) {

}
}
```

Python3:

```
class Solution:
def canReach(self, arr: List[int], start: int) -> bool:
```

Python:

```
class Solution(object):
def canReach(self, arr, start):
"""
:type arr: List[int]
:type start: int
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {number[]} arr
* @param {number} start
* @return {boolean}
*/
var canReach = function(arr, start) {

};
```

TypeScript:

```
function canReach(arr: number[], start: number): boolean {

};
```

C#:

```
public class Solution {
public bool CanReach(int[] arr, int start) {

}
}
```

C:

```
bool canReach(int* arr, int arrSize, int start) {

}
```

Go:

```
func canReach(arr []int, start int) bool {

}
```

Kotlin:

```
class Solution {
fun canReach(arr: IntArray, start: Int): Boolean {

}
}
```

Swift:

```
class Solution {
func canReach(_ arr: [Int], _ start: Int) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn can_reach(arr: Vec<i32>, start: i32) -> bool {

}
}
```

Ruby:

```
# @param {Integer[]} arr
# @param {Integer} start
# @return {Boolean}
def can_reach(arr, start)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $arr
```

```
* @param Integer $start
* @return Boolean
*/
function canReach($arr, $start) {

}
}
```

Dart:

```
class Solution {
bool canReach(List<int> arr, int start) {

}
}
```

Scala:

```
object Solution {
def canReach(arr: Array[Int], start: Int): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec can_reach(arr :: [integer], start :: integer) :: boolean
def can_reach(arr, start) do

end
end
```

Erlang:

```
-spec can_reach(Arr :: [integer()], Start :: integer()) -> boolean().
can_reach(Arr, Start) ->
.
```

Racket:

```
(define/contract (can-reach arr start)
(-> (listof exact-integer?) exact-integer? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Jump Game III
* Difficulty: Medium
* Tags: array, search
*
* 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:
bool canReach(vector<int>& arr, int start) {

}
};
```

Java Solution:

```
/**
* Problem: Jump Game III
* Difficulty: Medium
* Tags: array, search
*
* 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 boolean canReach(int[] arr, int start) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Jump Game III
Difficulty: Medium
Tags: array, search

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 canReach(self, arr: List[int], start: int) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def canReach(self, arr, start):
"""
:type arr: List[int]
:type start: int
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Jump Game III
* Difficulty: Medium
* Tags: array, search
*
* 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[]} arr
* @param {number} start
* @return {boolean}
*/
var canReach = function(arr, start) {

};
```

TypeScript Solution:

```
/**
* Problem: Jump Game III
* Difficulty: Medium
* Tags: array, search
*
* 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 canReach(arr: number[], start: number): boolean {

};
```

C# Solution:

```
/*
* Problem: Jump Game III
* Difficulty: Medium
* Tags: array, search
*
* 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 bool CanReach(int[] arr, int start) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Jump Game III
* Difficulty: Medium
* Tags: array, search
*
* 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
*/

bool canReach(int* arr, int arrSize, int start) {

}
```

Go Solution:

```
// Problem: Jump Game III
// Difficulty: Medium
// Tags: array, search
//
// 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 canReach(arr []int, start int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun canReach(arr: IntArray, start: Int): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func canReach(_ arr: [Int], _ start: Int) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Jump Game III
// Difficulty: Medium
// Tags: array, search
//
// 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 can_reach(arr: Vec<i32>, start: i32) -> bool {

}
}
```

Ruby Solution:

```
# @param {Integer[]} arr
# @param {Integer} start
# @return {Boolean}
def can_reach(arr, start)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $arr
* @param Integer $start
* @return Boolean
*/
function canReach($arr, $start) {
```

```
}
}
```

Dart Solution:

```
class Solution {
bool canReach(List<int> arr, int start) {

}
}
```

Scala Solution:

```
object Solution {
def canReach(arr: Array[Int], start: Int): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec can_reach(arr :: [integer], start :: integer) :: boolean
def can_reach(arr, start) do

end
end
```

Erlang Solution:

```
-spec can_reach(Arr :: [integer()], Start :: integer()) -> boolean().
can_reach(Arr, Start) ->
.
```

Racket Solution:

```
(define/contract (can-reach arr start)
(-> (listof exact-integer?) exact-integer? boolean?)
)
```