Problem 2005: Subtree Removal Game with Fibonacci Tree

Problem Information

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

Problem Description

A

Fibonacci

tree is a binary tree created using the order function

order(n)

:

order(0)

is the empty tree.

order(1)

is a binary tree with only

one node

.

order(n)

is a binary tree that consists of a root node with the left subtree as

order(n - 2)

and the right subtree as

order(n - 1)

.

Alice and Bob are playing a game with a

Fibonacci

tree with Alice staring first. On each turn, a player selects a node and removes that node

and

its subtree. The player that is forced to delete

root

loses.

Given the integer

n

, return

true

if Alice wins the game or

false

if Bob wins, assuming both players play optimally.

A subtree of a binary tree

tree

is a tree that consists of a node in

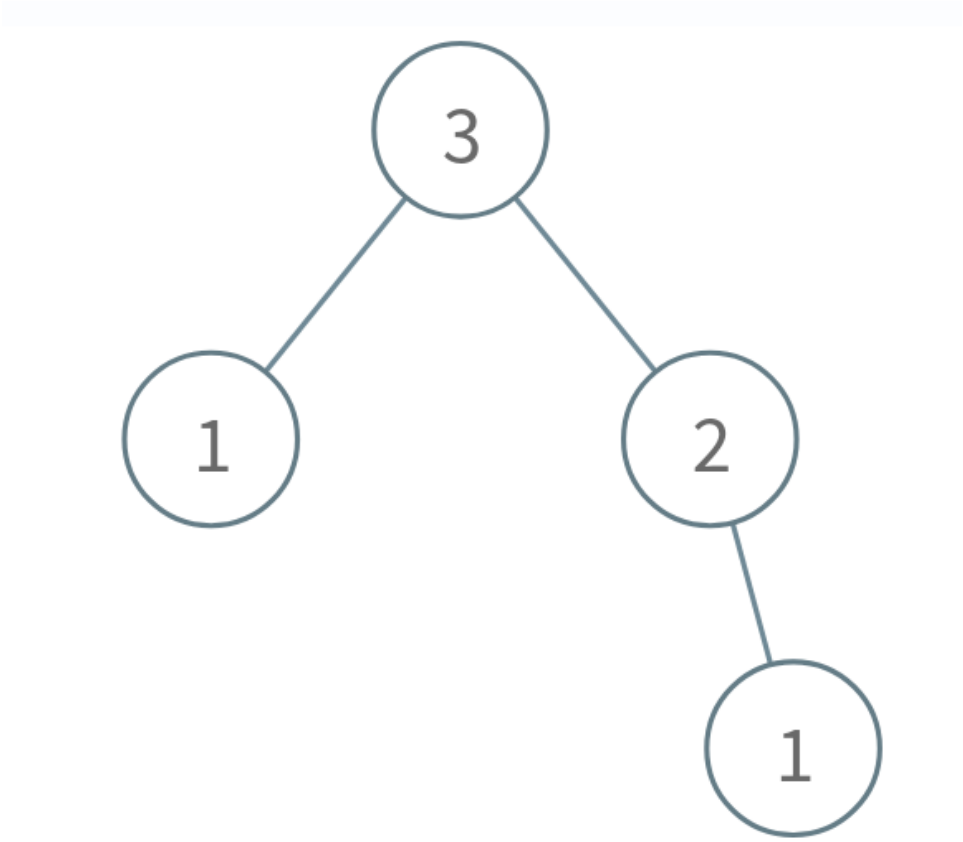
tree

and all of this node's descendants. The tree

tree

could also be considered as a subtree of itself.

Example 1:



Input:

n = 3

Output:

true

Explanation:

Alice takes the node 1 in the right subtree. Bob takes either the 1 in the left subtree or the 2 in the right subtree. Alice takes whichever node Bob doesn't take. Bob is forced to take the root node 3, so Bob will lose. Return true because Alice wins.

Example 2:

Input:

n = 1

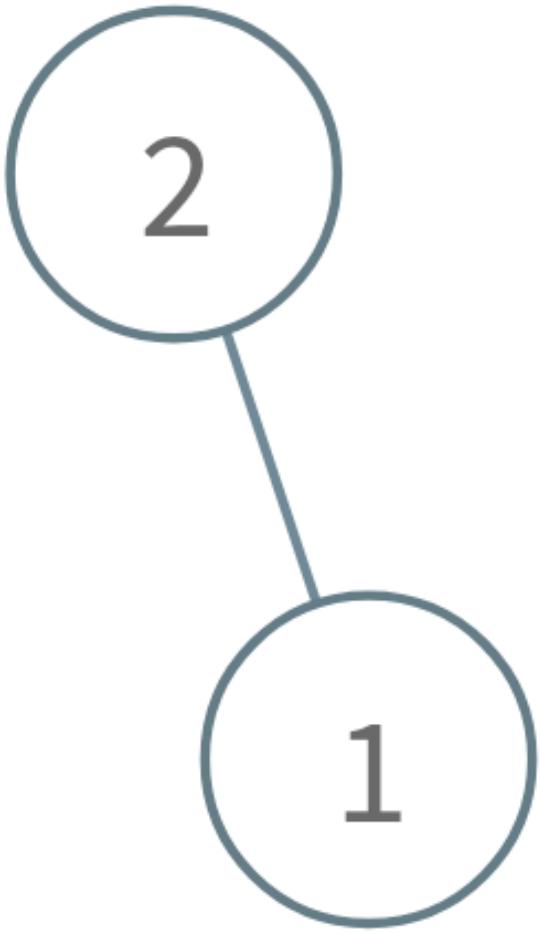
Output:

false

Explanation:

Alice is forced to take the root node 1, so Alice will lose. Return false because Alice loses.

Example 3:



Input:

n = 2

Output:

true

Explanation:

Alice takes the node 1. Bob is forced to take the root node 2, so Bob will lose. Return true because Alice wins.

Constraints:

1 <= n <= 100

Code Snippets

C++:

```
class Solution {
public:
bool findGameWinner(int n) {

}
};
```

Java:

```
class Solution {
public boolean findGameWinner(int n) {

}
}
```

Python3:

```
class Solution:
def findGameWinner(self, n: int) -> bool:
```

Python:

```
class Solution(object):
def findGameWinner(self, n):
"""
:type n: int
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {number} n
* @return {boolean}
*/
var findGameWinner = function(n) {
```

```
};
```

TypeScript:

```
function findGameWinner(n: number): boolean {

};
```

C#:

```
public class Solution {
public bool FindGameWinner(int n) {

}
}
```

C:

```
bool findGameWinner(int n) {

}
```

Go:

```
func findGameWinner(n int) bool {

}
```

Kotlin:

```
class Solution {
fun findGameWinner(n: Int): Boolean {

}
}
```

Swift:

```
class Solution {
func findGameWinner(_ n: Int) -> Bool {

}
```

```
}
```

Rust:

```
impl Solution {
pub fn find_game_winner(n: i32) -> bool {

}
}
```

Ruby:

```
# @param {Integer} n
# @return {Boolean}
def find_game_winner(n)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @return Boolean
*/
function findGameWinner($n) {

}
}
```

Dart:

```
class Solution {
bool findGameWinner(int n) {

}
}
```

Scala:

```
object Solution {
def findGameWinner(n: Int): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_game_winner(n :: integer) :: boolean
def find_game_winner(n) do

end
end
```

Erlang:

```
-spec find_game_winner(N :: integer()) -> boolean().
find_game_winner(N) ->
.
```

Racket:

```
(define/contract (find-game-winner n)
(-> exact-integer? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Subtree Removal Game with Fibonacci Tree
* Difficulty: Hard
* Tags: tree, dp, math
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(n) or O(n * m) for DP table
*/
```

```
class Solution {
public:
bool findGameWinner(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Subtree Removal Game with Fibonacci Tree
* Difficulty: Hard
* Tags: tree, dp, math
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public boolean findGameWinner(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Subtree Removal Game with Fibonacci Tree
Difficulty: Hard
Tags: tree, dp, math

Approach: DFS or BFS traversal
Time Complexity: O(n) where n is number of nodes
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

```
class Solution(object):
def findGameWinner(self, n):
"""
:type n: int
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Subtree Removal Game with Fibonacci Tree
* Difficulty: Hard
* Tags: tree, dp, math
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number} n
* @return {boolean}
*/
var findGameWinner = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Subtree Removal Game with Fibonacci Tree
* Difficulty: Hard
* Tags: tree, dp, math
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(n) or O(n * m) for DP table
*/

function findGameWinner(n: number): boolean {
```

```
};
```

C# Solution:

```
/*
* Problem: Subtree Removal Game with Fibonacci Tree
* Difficulty: Hard
* Tags: tree, dp, math
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public bool FindGameWinner(int n) {

}
}
```

C Solution:

```
/*
* Problem: Subtree Removal Game with Fibonacci Tree
* Difficulty: Hard
* Tags: tree, dp, math
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(n) or O(n * m) for DP table
*/

bool findGameWinner(int n) {

}
```

Go Solution:

```
// Problem: Subtree Removal Game with Fibonacci Tree
// Difficulty: Hard
```

```
// Tags: tree, dp, math
//
// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(n) or O(n * m) for DP table

func findGameWinner(n int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun findGameWinner(n: Int): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func findGameWinner(_ n: Int) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Subtree Removal Game with Fibonacci Tree
// Difficulty: Hard
// Tags: tree, dp, math
//
// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn find_game_winner(n: i32) -> bool {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @return {Boolean}
def find_game_winner(n)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @return Boolean
*/
function findGameWinner($n) {

}
}
```

Dart Solution:

```
class Solution {
bool findGameWinner(int n) {

}
}
```

Scala Solution:

```
object Solution {
def findGameWinner(n: Int): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_game_winner(n :: integer) :: boolean
def find_game_winner(n) do
```

```
end
end
```

Erlang Solution:

```
-spec find_game_winner(N :: integer()) -> boolean().
find_game_winner(N) ->
.
```

Racket Solution:

```
(define/contract (find-game-winner n)
(-> exact-integer? boolean?)
)
```