Problem 464: Can I Win

Problem Information

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

Problem Description

In the "100 game" two players take turns adding, to a running total, any integer from

1

to

10

. The player who first causes the running total to

reach or exceed

100 wins.

What if we change the game so that players

cannot

re-use integers?

For example, two players might take turns drawing from a common pool of numbers from 1 to 15 without replacement until they reach a total >= 100.

Given two integers

maxChoosableInteger

and

desiredTotal

, return

true

if the first player to move can force a win, otherwise, return

false

. Assume both players play

optimally

.

Example 1:

Input:

maxChoosableInteger = 10, desiredTotal = 11

Output:

false

Explanation:

No matter which integer the first player choose, the first player will lose. The first player can choose an integer from 1 up to 10. If the first player choose 1, the second player can only choose integers from 2 up to 10. The second player will win by choosing 10 and get a total = 11, which is >= desiredTotal. Same with other integers chosen by the first player, the second player will always win.

Example 2:

Input:

maxChoosableInteger = 10, desiredTotal = 0

Output:

true

Example 3:

Input:

maxChoosableInteger = 10, desiredTotal = 1

Output:

true

Constraints:

1 <= maxChoosableInteger <= 20

0 <= desiredTotal <= 300

Code Snippets

C++:

```
class Solution {
public:
bool canIWin(int maxChoosableInteger, int desiredTotal) {

}
};
```

Java:

```
class Solution {
public boolean canIWin(int maxChoosableInteger, int desiredTotal) {

}
```

```
}
```

Python3:

```
class Solution:
def canIWin(self, maxChoosableInteger: int, desiredTotal: int) -> bool:
```

Python:

```
class Solution(object):
def canIWin(self, maxChoosableInteger, desiredTotal):
"""
:type maxChoosableInteger: int
:type desiredTotal: int
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {number} maxChoosableInteger
* @param {number} desiredTotal
* @return {boolean}
*/
var canIWin = function(maxChoosableInteger, desiredTotal) {

};
```

TypeScript:

```
function canIWin(maxChoosableInteger: number, desiredTotal: number): boolean
{

};
```

C#:

```
public class Solution {
public bool CanIWin(int maxChoosableInteger, int desiredTotal) {

}
}
```

C:

```
bool canIWin(int maxChoosableInteger, int desiredTotal) {

}
```

Go:

```
func canIWin(maxChoosableInteger int, desiredTotal int) bool {

}
```

Kotlin:

```
class Solution {
fun canIWin(maxChoosableInteger: Int, desiredTotal: Int): Boolean {

}
}
```

Swift:

```
class Solution {
func canIWin(_ maxChoosableInteger: Int, _ desiredTotal: Int) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn can_i_win(max_choosable_integer: i32, desired_total: i32) -> bool {

}
}
```

Ruby:

```
# @param {Integer} max_choosable_integer
# @param {Integer} desired_total
# @return {Boolean}
def can_i_win(max_choosable_integer, desired_total)
```

```
end
```

PHP:

```
class Solution {

/**
* @param Integer $maxChoosableInteger
* @param Integer $desiredTotal
* @return Boolean
*/
function canIWin($maxChoosableInteger, $desiredTotal) {

}
}
```

Dart:

```
class Solution {
bool canIWin(int maxChoosableInteger, int desiredTotal) {

}
}
```

Scala:

```
object Solution {
def canIWin(maxChoosableInteger: Int, desiredTotal: Int): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec can_i_win(max_choosable_integer :: integer, desired_total :: integer)
:: boolean
def can_i_win(max_choosable_integer, desired_total) do

end
end
```

Erlang:

```
-spec can_i_win(MaxChoosableInteger :: integer(), DesiredTotal :: integer())
-> boolean().
can_i_win(MaxChoosableInteger, DesiredTotal) ->
.
```

Racket:

```
(define/contract (can-i-win maxChoosableInteger desiredTotal)
(-> exact-integer? exact-integer? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Can I Win
* Difficulty: Medium
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
bool canIWin(int maxChoosableInteger, int desiredTotal) {

}
};
```

Java Solution:

```
/**
* Problem: Can I Win
* Difficulty: Medium
* Tags: dp, math
*
```

```
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public boolean canIWin(int maxChoosableInteger, int desiredTotal) {

}
}
```

Python3 Solution:

```
"""
Problem: Can I Win
Difficulty: Medium
Tags: dp, math

Approach: Dynamic programming with memoization or tabulation
Time Complexity: O(n * m) where n and m are problem dimensions
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def canIWin(self, maxChoosableInteger: int, desiredTotal: int) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def canIWin(self, maxChoosableInteger, desiredTotal):
"""
:type maxChoosableInteger: int
:type desiredTotal: int
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Can I Win
* Difficulty: Medium
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number} maxChoosableInteger
* @param {number} desiredTotal
* @return {boolean}
*/
var canIWin = function(maxChoosableInteger, desiredTotal) {

};
```

TypeScript Solution:

```
/**
* Problem: Can I Win
* Difficulty: Medium
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

function canIWin(maxChoosableInteger: number, desiredTotal: number): boolean
{

};
```

C# Solution:

```
/*
* Problem: Can I Win
* Difficulty: Medium
* Tags: dp, math
```

```
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public bool CanIWin(int maxChoosableInteger, int desiredTotal) {

}
}
```

C Solution:

```
/*
* Problem: Can I Win
* Difficulty: Medium
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

bool canIWin(int maxChoosableInteger, int desiredTotal) {

}
```

Go Solution:

```
// Problem: Can I Win
// Difficulty: Medium
// Tags: dp, math
//
// Approach: Dynamic programming with memoization or tabulation
// Time Complexity: O(n * m) where n and m are problem dimensions
// Space Complexity: O(n) or O(n * m) for DP table

func canIWin(maxChoosableInteger int, desiredTotal int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun canIWin(maxChoosableInteger: Int, desiredTotal: Int): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func canIWin(_ maxChoosableInteger: Int, _ desiredTotal: Int) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Can I Win
// Difficulty: Medium
// Tags: dp, math
//
// Approach: Dynamic programming with memoization or tabulation
// Time Complexity: O(n * m) where n and m are problem dimensions
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn can_i_win(max_choosable_integer: i32, desired_total: i32) -> bool {

}
}
```

Ruby Solution:

```
# @param {Integer} max_choosable_integer
# @param {Integer} desired_total
# @return {Boolean}
def can_i_win(max_choosable_integer, desired_total)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $maxChoosableInteger
* @param Integer $desiredTotal
* @return Boolean
*/
function canIWin($maxChoosableInteger, $desiredTotal) {

}
}
```

Dart Solution:

```
class Solution {
bool canIWin(int maxChoosableInteger, int desiredTotal) {

}
}
```

Scala Solution:

```
object Solution {
def canIWin(maxChoosableInteger: Int, desiredTotal: Int): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec can_i_win(max_choosable_integer :: integer, desired_total :: integer)
:: boolean
def can_i_win(max_choosable_integer, desired_total) do

end
end
```

Erlang Solution:

```
-spec can_i_win(MaxChoosableInteger :: integer(), DesiredTotal :: integer())
-> boolean().
can_i_win(MaxChoosableInteger, DesiredTotal) ->
.
```

Racket Solution:

```
(define/contract (can-i-win maxChoosableInteger desiredTotal)
(-> exact-integer? exact-integer? boolean?)
)
```