Problem 294: Flip Game II

Problem Information

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

Problem Description

You are playing a Flip Game with your friend.

You are given a string

currentState

that contains only

'+'

and

'-'

. You and your friend take turns to flip

two consecutive

"++"

into

"--"

. The game ends when a person can no longer make a move, and therefore the other person will be the winner.

Return

true

if the starting player can

guarantee a win

, and

false

otherwise.

Example 1:

Input:

currentState = "++++"

Output:

true

Explanation:

The starting player can guarantee a win by flipping the middle "++" to become "+--+".

Example 2:

Input:

currentState = "+"

Output:

false

Constraints:

1 <= currentState.length <= 60

currentState[i]

is either

'+'

or

'-'

.

There cannot be more than 20 consecutive

'+'

.

Follow up:

Derive your algorithm's runtime complexity.

Code Snippets

C++:

```
class Solution {
public:
bool canWin(string currentState) {

}
};
```

Java:

```
class Solution {
public boolean canWin(String currentState) {

}
}
```

Python3:

```
class Solution:
def canWin(self, currentState: str) -> bool:
```

Python:

```
class Solution(object):
def canWin(self, currentState):
"""
:type currentState: str
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string} currentState
* @return {boolean}
*/
var canWin = function(currentState) {

};
```

TypeScript:

```
function canWin(currentState: string): boolean {

};
```

C#:

```
public class Solution {
public bool CanWin(string currentState) {

}
}
```

C:

```
bool canWin(char* currentState) {

}
```

Go:

```
func canWin(currentState string) bool {

}
```

Kotlin:

```
class Solution {
fun canWin(currentState: String): Boolean {

}
}
```

Swift:

```
class Solution {
func canWin(_ currentState: String) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn can_win(current_state: String) -> bool {

}
}
```

Ruby:

```
# @param {String} current_state
# @return {Boolean}
def can_win(current_state)

end
```

PHP:

```
class Solution {

/**
* @param String $currentState
* @return Boolean
*/
function canWin($currentState) {

}
}
```

Dart:

```
class Solution {
bool canWin(String currentState) {

}
}
```

Scala:

```
object Solution {
def canWin(currentState: String): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec can_win(current_state :: String.t) :: boolean
def can_win(current_state) do

end
end
```

Erlang:

```
-spec can_win(CurrentState :: unicode:unicode_binary()) -> boolean().
can_win(CurrentState) ->
.
```

Racket:

```
(define/contract (can-win currentState)
(-> string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Flip Game II
* Difficulty: Medium
* Tags: string, dp, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
bool canWin(string currentState) {

}
};
```

Java Solution:

```
/**
* Problem: Flip Game II
* Difficulty: Medium
* Tags: string, dp, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public boolean canWin(String currentState) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Flip Game II
Difficulty: Medium
Tags: string, dp, math

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def canWin(self, currentState: str) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def canWin(self, currentState):
"""
:type currentState: str
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Flip Game II
* Difficulty: Medium
* Tags: string, dp, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/
```

```
/**
* @param {string} currentState
* @return {boolean}
*/
var canWin = function(currentState) {

};
```

TypeScript Solution:

```
/**
* Problem: Flip Game II
* Difficulty: Medium
* Tags: string, dp, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function canWin(currentState: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Flip Game II
* Difficulty: Medium
* Tags: string, dp, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public bool CanWin(string currentState) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Flip Game II
* Difficulty: Medium
* Tags: string, dp, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

bool canWin(char* currentState) {

}
```

Go Solution:

```
// Problem: Flip Game II
// Difficulty: Medium
// Tags: string, dp, math
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func canWin(currentState string) bool {

}
```

Kotlin Solution:

```
class Solution {
fun canWin(currentState: String): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func canWin(_ currentState: String) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Flip Game II
// Difficulty: Medium
// Tags: string, dp, math
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn can_win(current_state: String) -> bool {

}
}
```

Ruby Solution:

```
# @param {String} current_state
# @return {Boolean}
def can_win(current_state)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $currentState
* @return Boolean
*/
function canWin($currentState) {

}
}
```

Dart Solution:

```
class Solution {
bool canWin(String currentState) {

}
}
```

Scala Solution:

```
object Solution {
def canWin(currentState: String): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec can_win(current_state :: String.t) :: boolean
def can_win(current_state) do

end
end
```

Erlang Solution:

```
-spec can_win(CurrentState :: unicode:unicode_binary()) -> boolean().
can_win(CurrentState) ->
.
```

Racket Solution:

```
(define/contract (can-win currentState)
(-> string? boolean?)
)
```