Problem 293: Flip Game

Problem Information

Difficulty: Easy
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 all possible states of the string

currentState

after

one valid move

. You may return the answer in

any order

. If there is no valid move, return an empty list

[]

.

Example 1:

Input:

currentState = "++++"

Output:

["--++","+--+","++--"]

Example 2:

Input:

currentState = "+"

Output:

[]

Constraints:

1 <= currentState.length <= 500

currentState[i]

is either

'+'

or

'-'

.

Code Snippets

C++:

```
class Solution {
public:
vector<string> generatePossibleNextMoves(string currentState) {

}
};
```

Java:

```
class Solution {
public List<String> generatePossibleNextMoves(String currentState) {

}
}
```

Python3:

```
class Solution:
def generatePossibleNextMoves(self, currentState: str) -> List[str]:
```

Python:

```
class Solution(object):
def generatePossibleNextMoves(self, currentState):
"""
:type currentState: str
:rtype: List[str]
"""
```

JavaScript:

```
/**
* @param {string} currentState
* @return {string[]}
*/
var generatePossibleNextMoves = function(currentState) {

};
```

TypeScript:

```
function generatePossibleNextMoves(currentState: string): string[] {

};
```

C#:

```
public class Solution {
public IList<string> GeneratePossibleNextMoves(string currentState) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** generatePossibleNextMoves(char* currentState, int* returnSize) {

}
```

Go:

```
func generatePossibleNextMoves(currentState string) []string {

}
```

Kotlin:

```
class Solution {
fun generatePossibleNextMoves(currentState: String): List<String> {

}
}
```

Swift:

```
class Solution {
func generatePossibleNextMoves(_ currentState: String) -> [String] {

}
}
```

Rust:

```
impl Solution {
pub fn generate_possible_next_moves(current_state: String) -> Vec<String> {

}
}
```

Ruby:

```
# @param {String} current_state
# @return {String[]}
def generate_possible_next_moves(current_state)

end
```

PHP:

```
class Solution {

/**
* @param String $currentState
* @return String[]
```

```
*/
function generatePossibleNextMoves($currentState) {

}
}
```

Dart:

```
class Solution {
List<String> generatePossibleNextMoves(String currentState) {

}
}
```

Scala:

```
object Solution {
def generatePossibleNextMoves(currentState: String): List[String] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec generate_possible_next_moves(current_state :: String.t) :: [String.t]
def generate_possible_next_moves(current_state) do

end
end
```

Erlang:

```
-spec generate_possible_next_moves(CurrentState :: unicode:unicode_binary())
-> [unicode:unicode_binary()].
generate_possible_next_moves(CurrentState) ->
.
```

Racket:

```
(define/contract (generate-possible-next-moves currentState)
(-> string? (listof string?))
```

```
)
```

Solutions

C++ Solution:

```
/*
* Problem: Flip Game
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
vector<string> generatePossibleNextMoves(string currentState) {

}
};
```

Java Solution:

```
/**
* Problem: Flip Game
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public List<String> generatePossibleNextMoves(String currentState) {

}
}
```

Python3 Solution:

```
"""
Problem: Flip Game
Difficulty: Easy
Tags: string

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def generatePossibleNextMoves(self, currentState: str) -> List[str]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def generatePossibleNextMoves(self, currentState):
"""
:type currentState: str
:rtype: List[str]
"""
```

JavaScript Solution:

```
/**
* Problem: Flip Game
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {string} currentState
* @return {string[]}
*/
```

```
var generatePossibleNextMoves = function(currentState) {

};
```

TypeScript Solution:

```
/**
* Problem: Flip Game
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function generatePossibleNextMoves(currentState: string): string[] {

};
```

C# Solution:

```
/*
* Problem: Flip Game
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public IList<string> GeneratePossibleNextMoves(string currentState) {

}
}
```

C Solution:

```
/*
* Problem: Flip Game
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** generatePossibleNextMoves(char* currentState, int* returnSize) {

}
```

Go Solution:

```
// Problem: Flip Game
// Difficulty: Easy
// Tags: string
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func generatePossibleNextMoves(currentState string) []string {

}
```

Kotlin Solution:

```
class Solution {
fun generatePossibleNextMoves(currentState: String): List<String> {

}
}
```

Swift Solution:

```
class Solution {
func generatePossibleNextMoves(_ currentState: String) -> [String] {

}
}
```

Rust Solution:

```
// Problem: Flip Game
// Difficulty: Easy
// Tags: string
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn generate_possible_next_moves(current_state: String) -> Vec<String> {

}
}
```

Ruby Solution:

```
# @param {String} current_state
# @return {String[]}
def generate_possible_next_moves(current_state)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $currentState
* @return String[]
*/
function generatePossibleNextMoves($currentState) {

}
}
```

Dart Solution:

```
class Solution {
List<String> generatePossibleNextMoves(String currentState) {

}
}
```

Scala Solution:

```
object Solution {
def generatePossibleNextMoves(currentState: String): List[String] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec generate_possible_next_moves(current_state :: String.t) :: [String.t]
def generate_possible_next_moves(current_state) do

end
end
```

Erlang Solution:

```
-spec generate_possible_next_moves(CurrentState :: unicode:unicode_binary())
-> [unicode:unicode_binary()].
generate_possible_next_moves(CurrentState) ->
.
```

Racket Solution:

```
(define/contract (generate-possible-next-moves currentState)
(-> string? (listof string?))
)
```