Problem 649: Dota2 Senate

Problem Information

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

Problem Description

In the world of Dota2, there are two parties: the Radiant and the Dire.

The Dota2 senate consists of senators coming from two parties. Now the Senate wants to decide on a change in the Dota2 game. The voting for this change is a round-based procedure. In each round, each senator can exercise

one

of the two rights:

Ban one senator's right:

A senator can make another senator lose all his rights in this and all the following rounds.

Announce the victory:

If this senator found the senators who still have rights to vote are all from the same party, he can announce the victory and decide on the change in the game.

Given a string

senate

representing each senator's party belonging. The character

'R'

and

'D'

represent the Radiant party and the Dire party. Then if there are

n

senators, the size of the given string will be

n

.

The round-based procedure starts from the first senator to the last senator in the given order. This procedure will last until the end of voting. All the senators who have lost their rights will be skipped during the procedure.

Suppose every senator is smart enough and will play the best strategy for his own party. Predict which party will finally announce the victory and change the Dota2 game. The output should be

"Radiant"

or

"Dire"

.

Example 1:

Input:

senate = "RD"

Output:

"Radiant"

Explanation:

The first senator comes from Radiant and he can just ban the next senator's right in round 1. And the second senator can't exercise any rights anymore since his right has been banned. And in round 2, the first senator can just announce the victory since he is the only guy in the senate who can vote.

Example 2:

Input:

senate = "RDD"

Output:

"Dire"

Explanation:

The first senator comes from Radiant and he can just ban the next senator's right in round 1. And the second senator can't exercise any rights anymore since his right has been banned. And the third senator comes from Dire and he can ban the first senator's right in round 1. And in round 2, the third senator can just announce the victory since he is the only guy in the senate who can vote.

Constraints:

n == senate.length

1 <= n <= 10

4

senate[i]

is either

'R'

or

'D'

.

Code Snippets

C++:

```
class Solution {
public:
string predictPartyVictory(string senate) {

}
};
```

Java:

```
class Solution {
public String predictPartyVictory(String senate) {

}
}
```

Python3:

```
class Solution:
def predictPartyVictory(self, senate: str) -> str:
```

Python:

```
class Solution(object):
def predictPartyVictory(self, senate):
"""
:type senate: str
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string} senate
* @return {string}
*/
var predictPartyVictory = function(senate) {

};
```

TypeScript:

```
function predictPartyVictory(senate: string): string {

};
```

C#:

```
public class Solution {
public string PredictPartyVictory(string senate) {

}
}
```

C:

```
char* predictPartyVictory(char* senate) {

}
```

Go:

```
func predictPartyVictory(senate string) string {

}
```

Kotlin:

```
class Solution {
fun predictPartyVictory(senate: String): String {

}
}
```

Swift:

```
class Solution {
func predictPartyVictory(_ senate: String) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn predict_party_victory(senate: String) -> String {

}
}
```

Ruby:

```
# @param {String} senate
# @return {String}
def predict_party_victory(senate)

end
```

PHP:

```
class Solution {

/**
* @param String $senate
* @return String
*/
function predictPartyVictory($senate) {

}
}
```

Dart:

```
class Solution {
String predictPartyVictory(String senate) {

}
}
```

Scala:

```
object Solution {
def predictPartyVictory(senate: String): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec predict_party_victory(senate :: String.t) :: String.t
def predict_party_victory(senate) do

end
end
```

Erlang:

```
-spec predict_party_victory(Senate :: unicode:unicode_binary()) ->
unicode:unicode_binary().
predict_party_victory(Senate) ->
.
```

Racket:

```
(define/contract (predict-party-victory senate)
(-> string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Dota2 Senate
* Difficulty: Medium
* Tags: string, greedy, queue
*
* 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:
string predictPartyVictory(string senate) {

}
};
```

Java Solution:

```
/**
* Problem: Dota2 Senate
* Difficulty: Medium
* Tags: string, greedy, queue
*
* 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 String predictPartyVictory(String senate) {

}
}
```

Python3 Solution:

```
"""
Problem: Dota2 Senate
Difficulty: Medium
Tags: string, greedy, queue

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 predictPartyVictory(self, senate: str) -> str:
```

```
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def predictPartyVictory(self, senate):
"""
:type senate: str
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Dota2 Senate
* Difficulty: Medium
* Tags: string, greedy, queue
*
* 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} senate
* @return {string}
*/
var predictPartyVictory = function(senate) {

};
```

TypeScript Solution:

```
/**
* Problem: Dota2 Senate
* Difficulty: Medium
* Tags: string, greedy, queue
*
* 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 predictPartyVictory(senate: string): string {

};
```

C# Solution:

```
/*
* Problem: Dota2 Senate
* Difficulty: Medium
* Tags: string, greedy, queue
*
* 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 string PredictPartyVictory(string senate) {

}
}
```

C Solution:

```
/*
* Problem: Dota2 Senate
* Difficulty: Medium
* Tags: string, greedy, queue
*
* 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
*/

char* predictPartyVictory(char* senate) {

}
```

Go Solution:

```
// Problem: Dota2 Senate
// Difficulty: Medium
// Tags: string, greedy, queue
//
// 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 predictPartyVictory(senate string) string {

}
```

Kotlin Solution:

```
class Solution {
fun predictPartyVictory(senate: String): String {

}
}
```

Swift Solution:

```
class Solution {
func predictPartyVictory(_ senate: String) -> String {

}
}
```

Rust Solution:

```
// Problem: Dota2 Senate
// Difficulty: Medium
// Tags: string, greedy, queue
//
// 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 predict_party_victory(senate: String) -> String {
}
}
```

Ruby Solution:

```
# @param {String} senate
# @return {String}
def predict_party_victory(senate)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $senate
* @return String
*/
function predictPartyVictory($senate) {

}
}
```

Dart Solution:

```
class Solution {
String predictPartyVictory(String senate) {

}
}
```

Scala Solution:

```
object Solution {
def predictPartyVictory(senate: String): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec predict_party_victory(senate :: String.t) :: String.t
def predict_party_victory(senate) do

end
end
```

Erlang Solution:

```
-spec predict_party_victory(Senate :: unicode:unicode_binary()) ->
unicode:unicode_binary().
predict_party_victory(Senate) ->
.
```

Racket Solution:

```
(define/contract (predict-party-victory senate)
(-> string? string?)
)
```