Problem 1246: Palindrome Removal

Problem Information

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

Problem Description

You are given an integer array

arr

.

In one move, you can select a

palindromic

subarray

arr[i], arr[i + 1], ..., arr[j]

where

i <= j

, and remove that subarray from the given array. Note that after removing a subarray, the elements on the left and on the right of that subarray move to fill the gap left by the removal.

Return

the minimum number of moves needed to remove all numbers from the array

.

Example 1:

Input:

arr = [1,2]

Output:

2

Example 2:

Input:

arr = [1,3,4,1,5]

Output:

3

Explanation:

Remove [4] then remove [1,3,1] then remove [5].

Constraints:

1 <= arr.length <= 100

1 <= arr[i] <= 20

Code Snippets

C++:

```
class Solution {
public:
int minimumMoves(vector<int>& arr) {
}
};
```

Java:

```
class Solution {
public int minimumMoves(int[] arr) {

}
}
```

Python3:

```
class Solution:
def minimumMoves(self, arr: List[int]) -> int:
```

Python:

```
class Solution(object):
def minimumMoves(self, arr):
"""
:type arr: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} arr
* @return {number}
*/
var minimumMoves = function(arr) {

};
```

TypeScript:

```
function minimumMoves(arr: number[]): number {

};
```

C#:

```
}
```

```
public class Solution {
public int MinimumMoves(int[] arr) {

}
}
```

C:

```
int minimumMoves(int* arr, int arrSize) {

}
```

Go:

```
func minimumMoves(arr []int) int {

}
```

Kotlin:

```
class Solution {
fun minimumMoves(arr: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func minimumMoves(_ arr: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_moves(arr: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} arr
# @return {Integer}
def minimum_moves(arr)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $arr
* @return Integer
*/
function minimumMoves($arr) {

}
}
```

Dart:

```
class Solution {
int minimumMoves(List<int> arr) {

}
}
```

Scala:

```
object Solution {
def minimumMoves(arr: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_moves(arr :: [integer]) :: integer
def minimum_moves(arr) do

end
end
```

Erlang:

```
-spec minimum_moves(Arr :: [integer()]) -> integer().
minimum_moves(Arr) ->
.
```

Racket:

```
(define/contract (minimum-moves arr)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Palindrome Removal
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int minimumMoves(vector<int>& arr) {

}
};
```

Java Solution:

```
/**
* Problem: Palindrome Removal
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int minimumMoves(int[] arr) {

}
}
```

Python3 Solution:

```
"""
Problem: Palindrome Removal
Difficulty: Hard
Tags: array, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def minimumMoves(self, arr: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minimumMoves(self, arr):
"""
:type arr: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Palindrome Removal
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[]} arr
* @return {number}
*/
var minimumMoves = function(arr) {

};
```

TypeScript Solution:

```
/**
* Problem: Palindrome Removal
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function minimumMoves(arr: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Palindrome Removal
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

public class Solution {
public int MinimumMoves(int[] arr) {

}
}
```

C Solution:

```
/*
* Problem: Palindrome Removal
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int minimumMoves(int* arr, int arrSize) {

}
```

Go Solution:

```
// Problem: Palindrome Removal
// Difficulty: Hard
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func minimumMoves(arr []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumMoves(arr: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func minimumMoves(_ arr: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Palindrome Removal
// Difficulty: Hard
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn minimum_moves(arr: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} arr
# @return {Integer}
def minimum_moves(arr)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer[] $arr
* @return Integer
*/
function minimumMoves($arr) {

}
}
```

Dart Solution:

```
class Solution {
int minimumMoves(List<int> arr) {

}
}
```

Scala Solution:

```
object Solution {
def minimumMoves(arr: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_moves(arr :: [integer]) :: integer
def minimum_moves(arr) do

end
end
```

Erlang Solution:

```
-spec minimum_moves(Arr :: [integer()]) -> integer().
minimum_moves(Arr) ->
.
```

Racket Solution:

```
(define/contract (minimum-moves arr)
(-> (listof exact-integer?) exact-integer?)
)
```