Problem 3429: Paint House IV

Problem Information

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

Problem Description

You are given an

even

integer

n

representing the number of houses arranged in a straight line, and a 2D array

cost

of size

n x 3

, where

cost[i][j]

represents the cost of painting house

i

with color

j + 1

.

The houses will look

beautiful

if they satisfy the following conditions:

No

two

adjacent houses are painted the same color.

Houses

equidistant

from the ends of the row are

not

painted the same color. For example, if

n = 6

, houses at positions

(0, 5)

,

(1, 4)

, and

(2, 3)

are considered equidistant.

Return the

minimum

cost to paint the houses such that they look

beautiful

.

Example 1:

Input:

n = 4, cost = [[3,5,7],[6,2,9],[4,8,1],[7,3,5]]

Output:

9

Explanation:

The optimal painting sequence is

[1, 2, 3, 2]

with corresponding costs

[3, 2, 1, 3]

. This satisfies the following conditions:

No adjacent houses have the same color.

Houses at positions 0 and 3 (equidistant from the ends) are not painted the same color

(1 != 2)

.

Houses at positions 1 and 2 (equidistant from the ends) are not painted the same color

(2 != 3)

.

The minimum cost to paint the houses so that they look beautiful is

3 + 2 + 1 + 3 = 9

.

Example 2:

Input:

n = 6, cost = [[2,4,6],[5,3,8],[7,1,9],[4,6,2],[3,5,7],[8,2,4]]

Output:

18

Explanation:

The optimal painting sequence is

[1, 3, 2, 3, 1, 2]

with corresponding costs

[2, 8, 1, 2, 3, 2]

. This satisfies the following conditions:

No adjacent houses have the same color.

Houses at positions 0 and 5 (equidistant from the ends) are not painted the same color

(1 != 2)

.

Houses at positions 1 and 4 (equidistant from the ends) are not painted the same color

(3 != 1)

.

Houses at positions 2 and 3 (equidistant from the ends) are not painted the same color

(2 != 3)

.

The minimum cost to paint the houses so that they look beautiful is

2 + 8 + 1 + 2 + 3 + 2 = 18

.

Constraints:

2 <= n <= 10

5

n

is even.

cost.length == n

cost[i].length == 3

0 <= cost[i][j] <= 10

5

Code Snippets

C++:

```
class Solution {
public:
long long minCost(int n, vector<vector<int>>& cost) {

}
};
```

Java:

```
class Solution {
public long minCost(int n, int[][] cost) {

}
}
```

Python3:

```
class Solution:
def minCost(self, n: int, cost: List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def minCost(self, n, cost):
"""
:type n: int
:type cost: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} cost
* @return {number}
*/
var minCost = function(n, cost) {

};
```

TypeScript:

```
function minCost(n: number, cost: number[][]): number {

};
```

C#:

```
public class Solution {
public long MinCost(int n, int[][] cost) {

}
}
```

C:

```
long long minCost(int n, int** cost, int costSize, int* costColSize) {

}
```

Go:

```
func minCost(n int, cost [][]int) int64 {

}
```

Kotlin:

```
class Solution {
fun minCost(n: Int, cost: Array<IntArray>): Long {

}
}
```

Swift:

```
class Solution {
func minCost(_ n: Int, _ cost: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_cost(n: i32, cost: Vec<Vec<i32>>) -> i64 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} cost
# @return {Integer}
def min_cost(n, cost)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $cost
* @return Integer
*/
function minCost($n, $cost) {

}
}
```

Dart:

```
class Solution {
int minCost(int n, List<List<int>> cost) {
```

```
}
}
```

Scala:

```
object Solution {
def minCost(n: Int, cost: Array[Array[Int]]): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_cost(n :: integer, cost :: [[integer]]) :: integer
def min_cost(n, cost) do

end
end
```

Erlang:

```
-spec min_cost(N :: integer(), Cost :: [[integer()]]) -> integer().
min_cost(N, Cost) ->
.
```

Racket:

```
(define/contract (min-cost n cost)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Paint House IV
* Difficulty: Medium
```

```
* 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:
long long minCost(int n, vector<vector<int>>& cost) {

}
};
```

Java Solution:

```
/**
* Problem: Paint House IV
* Difficulty: Medium
* 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 long minCost(int n, int[][] cost) {

}
}
```

Python3 Solution:

```
"""
Problem: Paint House IV
Difficulty: Medium
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 minCost(self, n: int, cost: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minCost(self, n, cost):
"""
:type n: int
:type cost: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Paint House IV
* Difficulty: Medium
* 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} n
* @param {number[][]} cost
* @return {number}
*/
var minCost = function(n, cost) {

};
```

TypeScript Solution:

```
/**
* Problem: Paint House IV
* Difficulty: Medium
* 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 minCost(n: number, cost: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Paint House IV
* Difficulty: Medium
* 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 long MinCost(int n, int[][] cost) {

}
}
```

C Solution:

```
/*
* Problem: Paint House IV
* Difficulty: Medium
* 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
```

```
*/

long long minCost(int n, int** cost, int costSize, int* costColSize) {

}
```

Go Solution:

```
// Problem: Paint House IV
// Difficulty: Medium
// 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 minCost(n int, cost [][]int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun minCost(n: Int, cost: Array<IntArray>): Long {

}
}
```

Swift Solution:

```
class Solution {
func minCost(_ n: Int, _ cost: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Paint House IV
// Difficulty: Medium
// 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 min_cost(n: i32, cost: Vec<Vec<i32>>) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} cost
# @return {Integer}
def min_cost(n, cost)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $cost
* @return Integer
*/
function minCost($n, $cost) {

}
}
```

Dart Solution:

```
class Solution {
int minCost(int n, List<List<int>> cost) {

}
}
```

Scala Solution:

```
object Solution {
def minCost(n: Int, cost: Array[Array[Int]]): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_cost(n :: integer, cost :: [[integer]]) :: integer
def min_cost(n, cost) do

end
end
```

Erlang Solution:

```
-spec min_cost(N :: integer(), Cost :: [[integer()]]) -> integer().
min_cost(N, Cost) ->
.
```

Racket Solution:

```
(define/contract (min-cost n cost)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```