Problem 256: Paint House

Problem Information

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

Problem Description

There is a row of

n

houses, where each house can be painted one of three colors: red, blue, or green. The cost of painting each house with a certain color is different. You have to paint all the houses such that no two adjacent houses have the same color.

The cost of painting each house with a certain color is represented by an

n x 3

cost matrix

costs

.

For example,

costs[0][0]

is the cost of painting house

0

with the color red;

costs[1][2]

is the cost of painting house 1 with color green, and so on...

Return

the minimum cost to paint all houses

.

Example 1:

Input:

costs = [[17,2,17],[16,16,5],[14,3,19]]

Output:

10

Explanation:

Paint house 0 into blue, paint house 1 into green, paint house 2 into blue. Minimum cost: 2 + 5 + 3 = 10.

Example 2:

Input:

costs = [[7,6,2]]

Output:

2

Constraints:

costs.length == n

costs[i].length == 3

1 <= n <= 100

1 <= costs[i][j] <= 20

Code Snippets

C++:

```
class Solution {
public:
int minCost(vector<vector<int>>& costs) {

}
};
```

Java:

```
class Solution {
public int minCost(int[][] costs) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minCost(costs: number[][]): number {

};
```

C#:

```
public class Solution {
public int MinCost(int[][] costs) {

}
}
```

C:

```
int minCost(int** costs, int costsSize, int* costsColSize) {

}
```

Go:

```
func minCost(costs [][]int) int {

}
```

Kotlin:

```
class Solution {
fun minCost(costs: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func minCost(_ costs: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_cost(costs: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} costs
# @return {Integer}
def min_cost(costs)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $costs
* @return Integer
*/
function minCost($costs) {

}
}
```

Dart:

```
class Solution {
int minCost(List<List<int>> costs) {

}

}
```

Scala:

```
object Solution {
def minCost(costs: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_cost(costs :: [[integer]]) :: integer
def min_cost(costs) do

end
end
```

Erlang:

```
-spec min_cost(Costs :: [[integer()]]) -> integer().
min_cost(Costs) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Paint House
* 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:
int minCost(vector<vector<int>>& costs) {

}
};
```

Java Solution:

```
/**
* Problem: Paint House
* 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 int minCost(int[][] costs) {

}
}
```

Python3 Solution:

```
"""
Problem: Paint House
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, costs: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Paint House
* 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[][]} costs
* @return {number}
*/
var minCost = function(costs) {

};
```

TypeScript Solution:

```
/**
* Problem: Paint House
* 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(costs: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Paint House
* 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 int MinCost(int[][] costs) {

}
}
```

C Solution:

```
/*
* Problem: Paint House
* 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
*/

int minCost(int** costs, int costsSize, int* costsColSize) {


}
```

Go Solution:

```
// Problem: Paint House
// 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(costs [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minCost(costs: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func minCost(_ costs: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Paint House
// 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(costs: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} costs
# @return {Integer}
def min_cost(costs)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $costs
* @return Integer
*/
function minCost($costs) {

}
}
```

Dart Solution:

```
class Solution {
int minCost(List<List<int>> costs) {

}
}
```

Scala Solution:

```
object Solution {
def minCost(costs: Array[Array[Int]]): Int = {


}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_cost(costs :: [[integer]]) :: integer
def min_cost(costs) do

end
end
```

Erlang Solution:

```
-spec min_cost(Costs :: [[integer()]]) -> integer().
min_cost(Costs) ->
.
```

Racket Solution:

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