Problem 265: Paint House II

Problem Information

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

Problem Description

There are a row of

n

houses, each house can be painted with one of the

k

colors. 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 k

cost matrix costs.

For example,

costs[0][0]

is the cost of painting house

0

with color

0

;

costs[1][2]

is the cost of painting house

1

with color

2

, and so on...

Return

the minimum cost to paint all houses

.

Example 1:

Input:

costs = [[1,5,3],[2,9,4]]

Output:

5

Explanation:

Paint house 0 into color 0, paint house 1 into color 2. Minimum cost: 1 + 4 = 5; Or paint house 0 into color 2, paint house 1 into color 0. Minimum cost: 3 + 2 = 5.

Example 2:

Input:

costs = [[1,3],[2,4]]

Output:

5

Constraints:

costs.length == n

costs[i].length == k

1 <= n <= 100

2 <= k <= 20

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

Follow up:

Could you solve it in

O(nk)

runtime?

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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


}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

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

}
};
```

Java Solution:

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

}
}
```

Python3 Solution:

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

Python Solution:

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

JavaScript Solution:

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

};
```

TypeScript Solution:

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

};
```

C# Solution:

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

}
}
```

C Solution:

```
/*
* Problem: Paint House II
* 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 minCostII(int** costs, int costsSize, int* costsColSize) {

}
```

Go Solution:

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

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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