Problem 1473: Paint House III

Problem Information

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

Problem Description

There is a row of

m

houses in a small city, each house must be painted with one of the

n

colors (labeled from

1

to

n

), some houses that have been painted last summer should not be painted again.

A neighborhood is a maximal group of continuous houses that are painted with the same color.

For example:

houses = [1,2,2,3,3,2,1,1]

contains

5

neighborhoods

[{1}, {2,2}, {3,3}, {2}, {1,1}]

.

Given an array

houses

, an

m x n

matrix

cost

and an integer

target

where:

houses[i]

: is the color of the house

i

, and

0

if the house is not painted yet.

cost[i][j]

: is the cost of paint the house

i

with the color

j + 1

.

Return

the minimum cost of painting all the remaining houses in such a way that there are exactly

target

neighborhoods

. If it is not possible, return

-1

.

Example 1:

Input:

houses = [0,0,0,0,0], cost = [[1,10],[10,1],[10,1],[1,10],[5,1]], m = 5, n = 2, target = 3

Output:

9

Explanation:

Paint houses of this way [1,2,2,1,1] This array contains target = 3 neighborhoods, [{1}, {2,2}, {1,1}]. Cost of paint all houses (1 + 1 + 1 + 1 + 5) = 9.

Example 2:

Input:

houses = [0,2,1,2,0], cost = [[1,10],[10,1],[10,1],[1,10],[5,1]], m = 5, n = 2, target = 3

Output:

11

Explanation:

Some houses are already painted, Paint the houses of this way [2,2,1,2,2] This array contains target = 3 neighborhoods, [{2,2}, {1}, {2,2}]. Cost of paint the first and last house (10 + 1) = 11.

Example 3:

Input:

houses = [3,1,2,3], cost = [[1,1,1],[1,1,1],[1,1,1],[1,1,1]], m = 4, n = 3, target = 3

Output:

-1

Explanation:

Houses are already painted with a total of 4 neighborhoods [{3},{1},{2},{3}] different of target = 3.

Constraints:

m == houses.length == cost.length

n == cost[i].length

1 <= m <= 100

1 <= n <= 20

1 <= target <= m

0 <= houses[i] <= n

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

4

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int minCost(int[] houses, int[][] cost, int m, int n, int target) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public int MinCost(int[] houses, int[][] cost, int m, int n, int target) {

}
}
```

C:

```
int minCost(int* houses, int housesSize, int** cost, int costSize, int*
costColSize, int m, int n, int target) {

}
```

Go:

```
func minCost(houses []int, cost [][]int, m int, n int, target int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Paint House III
* 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 minCost(vector<int>& houses, vector<vector<int>>& cost, int m, int n, int
target) {
}
};
```

Java Solution:

```
/**
* Problem: Paint House III
* 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 minCost(int[] houses, int[][] cost, int m, int n, int target) {

}
}
```

Python3 Solution:

```
"""
Problem: Paint House III
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 minCost(self, houses: List[int], cost: List[List[int]], m: int, n: int,
target: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Paint House III
* 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[]} houses
* @param {number[][]} cost
* @param {number} m
* @param {number} n
* @param {number} target
* @return {number}
*/
var minCost = function(houses, cost, m, n, target) {

};
```

TypeScript Solution:

```
/**
* Problem: Paint House III
* 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 minCost(houses: number[], cost: number[][], m: number, n: number,
target: number): number {

};
```

C# Solution:

```
/*
* Problem: Paint House III
* 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 MinCost(int[] houses, int[][] cost, int m, int n, int target) {

}
}
```

C Solution:

```
/*
* Problem: Paint House III
* 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 minCost(int* houses, int housesSize, int** cost, int costSize, int*
costColSize, int m, int n, int target) {

}
```

Go Solution:

```
// Problem: Paint House III
// 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 minCost(houses []int, cost [][]int, m int, n int, target int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Paint House III
// 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(houses: Vec<i32>, cost: Vec<Vec<i32>>, m: i32, n: i32,
target: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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