Problem 2830: Maximize the Profit as the Salesman

Problem Information

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

Problem Description

You are given an integer

n

representing the number of houses on a number line, numbered from

0

to

n - 1

.

Additionally, you are given a 2D integer array

offers

where

offers[i] = [start

i

, end

i

, gold

i

]

, indicating that

i

th

buyer wants to buy all the houses from

start

i

to

end

i

for

gold

i

amount of gold.

As a salesman, your goal is to

maximize

your earnings by strategically selecting and selling houses to buyers.

Return

the maximum amount of gold you can earn

.

Note

that different buyers can't buy the same house, and some houses may remain unsold.

Example 1:

Input:

n = 5, offers = [[0,0,1],[0,2,2],[1,3,2]]

Output:

3

Explanation:

There are 5 houses numbered from 0 to 4 and there are 3 purchase offers. We sell houses in the range [0,0] to 1

st

buyer for 1 gold and houses in the range [1,3] to 3

rd

buyer for 2 golds. It can be proven that 3 is the maximum amount of gold we can achieve.

Example 2:

Input:

n = 5, offers = [[0,0,1],[0,2,10],[1,3,2]]

Output:

10

Explanation:

There are 5 houses numbered from 0 to 4 and there are 3 purchase offers. We sell houses in the range [0,2] to 2

nd

buyer for 10 golds. It can be proven that 10 is the maximum amount of gold we can achieve.

Constraints:

1 <= n <= 10

5

1 <= offers.length <= 10

5

offers[i].length == 3

0 <= start

i

<= end

i

<= n - 1

1 <= gold

i

<= 10

3

Code Snippets

C++:

```
class Solution {
public:
int maximizeTheProfit(int n, vector<vector<int>>& offers) {

}
};
```

Java:

```
class Solution {
public int maximizeTheProfit(int n, List<List<Integer>> offers) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maximizeTheProfit(n: number, offers: number[][]): number {

};
```

C#:

```
public class Solution {
public int MaximizeTheProfit(int n, IList<IList<int>> offers) {

}
}
```

C:

```
int maximizeTheProfit(int n, int** offers, int offersSize, int*
offersColSize) {

}
```

Go:

```
func maximizeTheProfit(n int, offers [][]int) int {

}
```

Kotlin:

```
class Solution {
fun maximizeTheProfit(n: Int, offers: List<List<Int>>): Int {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn maximize_the_profit(n: i32, offers: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} offers
# @return {Integer}
def maximize_the_profit(n, offers)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $offers
* @return Integer
*/
function maximizeTheProfit($n, $offers) {

}
}
```

Dart:

```
class Solution {
int maximizeTheProfit(int n, List<List<int>> offers) {
```

```
}
}
```

Scala:

```
object Solution {
def maximizeTheProfit(n: Int, offers: List[List[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec maximize_the_profit(n :: integer, offers :: [[integer]]) :: integer
def maximize_the_profit(n, offers) do

end
end
```

Erlang:

```
-spec maximize_the_profit(N :: integer(), Offers :: [[integer()]]) ->
integer().
maximize_the_profit(N, Offers) ->
.
```

Racket:

```
(define/contract (maximize-the-profit n offers)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximize the Profit as the Salesman
```

```
* Difficulty: Medium
* Tags: array, dp, hash, sort, search
*
* 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 maximizeTheProfit(int n, vector<vector<int>>& offers) {

}
};
```

Java Solution:

```
/**
* Problem: Maximize the Profit as the Salesman
* Difficulty: Medium
* Tags: array, dp, hash, sort, search
*
* 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 maximizeTheProfit(int n, List<List<Integer>> offers) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximize the Profit as the Salesman
Difficulty: Medium
Tags: array, dp, hash, sort, search

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 maximizeTheProfit(self, n: int, offers: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximize the Profit as the Salesman
* Difficulty: Medium
* Tags: array, dp, hash, sort, search
*
* 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[][]} offers
* @return {number}
*/
var maximizeTheProfit = function(n, offers) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximize the Profit as the Salesman
* Difficulty: Medium
* Tags: array, dp, hash, sort, search
*
* 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 maximizeTheProfit(n: number, offers: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Maximize the Profit as the Salesman
* Difficulty: Medium
* Tags: array, dp, hash, sort, search
*
* 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 MaximizeTheProfit(int n, IList<IList<int>> offers) {

}
}
```

C Solution:

```
/*
* Problem: Maximize the Profit as the Salesman
* Difficulty: Medium
* Tags: array, dp, hash, sort, search
*
* 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 maximizeTheProfit(int n, int** offers, int offersSize, int*
offersColSize) {

}
```

Go Solution:

```
// Problem: Maximize the Profit as the Salesman
// Difficulty: Medium
// Tags: array, dp, hash, sort, search
//
// 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 maximizeTheProfit(n int, offers [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maximizeTheProfit(n: Int, offers: List<List<Int>>): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximize the Profit as the Salesman
// Difficulty: Medium
```

```
// Tags: array, dp, hash, sort, search
//
// 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 maximize_the_profit(n: i32, offers: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} offers
# @return {Integer}
def maximize_the_profit(n, offers)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $offers
* @return Integer
*/
function maximizeTheProfit($n, $offers) {

}
}
```

Dart Solution:

```
class Solution {
int maximizeTheProfit(int n, List<List<int>> offers) {

}
```

```
}
```

Scala Solution:

```
object Solution {
def maximizeTheProfit(n: Int, offers: List[List[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximize_the_profit(n :: integer, offers :: [[integer]]) :: integer
def maximize_the_profit(n, offers) do

end
end
```

Erlang Solution:

```
-spec maximize_the_profit(N :: integer(), Offers :: [[integer()]]) ->
integer().
maximize_the_profit(N, Offers) ->
.
```

Racket Solution:

```
(define/contract (maximize-the-profit n offers)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```