Problem 2008: Maximum Earnings From Taxi

Problem Information

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

Problem Description

There are

n

points on a road you are driving your taxi on. The

n

points on the road are labeled from

1

to

n

in the direction you are going, and you want to drive from point

1

to point

n

to make money by picking up passengers. You cannot change the direction of the taxi.

The passengers are represented by a

0-indexed

2D integer array

rides

, where

rides[i] = [start

i

, end

i

, tip

i

]

denotes the

i

th

passenger requesting a ride from point

start

i

to point

end

i

who is willing to give a

tip

i

dollar tip.

For

each

passenger

i

you pick up, you

earn

end

i

- start

i

+ tip

i

dollars. You may only drive

at most one

passenger at a time.

Given

n

and

rides

, return

the

maximum

number of dollars you can earn by picking up the passengers optimally.

Note:

You may drop off a passenger and pick up a different passenger at the same point.

Example 1:

Input:

n = 5, rides = [

[2,5,4]

,[1,5,1]]

Output:

7

Explanation:

We can pick up passenger 0 to earn 5 - 2 + 4 = 7 dollars.

Example 2:

Input:

n = 20, rides = [[1,6,1],

[3,10,2]

,

[10,12,3]

,[11,12,2],[12,15,2],

[13,18,1]

]

Output:

20

Explanation:

We will pick up the following passengers: - Drive passenger 1 from point 3 to point 10 for a profit of 10 - 3 + 2 = 9 dollars. - Drive passenger 2 from point 10 to point 12 for a profit of 12 - 10 + 3 = 5 dollars. - Drive passenger 5 from point 13 to point 18 for a profit of 18 - 13 + 1 = 6 dollars. We earn 9 + 5 + 6 = 20 dollars in total.

Constraints:

1 <= n <= 10

5

1 <= rides.length <= 3 * 10

4

rides[i].length == 3

1 <= start

i

< end

i

<= n

1 <= tip

i

<= 10

5

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public long maxTaxiEarnings(int n, int[][] rides) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maxTaxiEarnings(n: number, rides: number[][]): number {

};
```

C#:

```
public class Solution {
public long MaxTaxiEarnings(int n, int[][] rides) {

}
}
```

C:

```
long long maxTaxiEarnings(int n, int** rides, int ridesSize, int*
ridesColSize) {

}
```

Go:

```
func maxTaxiEarnings(n int, rides [][]int) int64 {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} rides
# @return {Integer}
def max_taxi_earnings(n, rides)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $rides
* @return Integer
*/
function maxTaxiEarnings($n, $rides) {

}
}
```

Dart:

```
class Solution {
int maxTaxiEarnings(int n, List<List<int>> rides) {

}
}
```

Scala:

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

}
}
```

Elixir:

```
defmodule Solution do
@spec max_taxi_earnings(n :: integer, rides :: [[integer]]) :: integer
def max_taxi_earnings(n, rides) do

end
end
```

Erlang:

```
-spec max_taxi_earnings(N :: integer(), Rides :: [[integer()]]) -> integer().
max_taxi_earnings(N, Rides) ->
```

```
.
```

Racket:

```
(define/contract (max-taxi-earnings n rides)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Earnings From Taxi
* 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:
long long maxTaxiEarnings(int n, vector<vector<int>>& rides) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Earnings From Taxi
* 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 long maxTaxiEarnings(int n, int[][] rides) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Earnings From Taxi
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 maxTaxiEarnings(self, n: int, rides: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Earnings From Taxi
* 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[][]} rides
* @return {number}
*/
var maxTaxiEarnings = function(n, rides) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Earnings From Taxi
* 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 maxTaxiEarnings(n: number, rides: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Earnings From Taxi
* 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 long MaxTaxiEarnings(int n, int[][] rides) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Earnings From Taxi
* 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
*/

long long maxTaxiEarnings(int n, int** rides, int ridesSize, int*
ridesColSize) {

}
```

Go Solution:

```
// Problem: Maximum Earnings From Taxi
// 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 maxTaxiEarnings(n int, rides [][]int) int64 {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Earnings From Taxi
// 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 max_taxi_earnings(n: i32, rides: Vec<Vec<i32>>) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} rides
# @return {Integer}
def max_taxi_earnings(n, rides)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $rides
* @return Integer
*/
function maxTaxiEarnings($n, $rides) {

}
}
```

Dart Solution:

```
class Solution {
int maxTaxiEarnings(int n, List<List<int>> rides) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_taxi_earnings(n :: integer, rides :: [[integer]]) :: integer
def max_taxi_earnings(n, rides) do

end
end
```

Erlang Solution:

```
-spec max_taxi_earnings(N :: integer(), Rides :: [[integer()]]) -> integer().
max_taxi_earnings(N, Rides) ->
.
```

Racket Solution:

```
(define/contract (max-taxi-earnings n rides)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```