Problem 871: Minimum Number of Refueling Stops

Problem Information

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

Problem Description

A car travels from a starting position to a destination which is

target

miles east of the starting position.

There are gas stations along the way. The gas stations are represented as an array

stations

where

stations[i] = [position

i

, fuel

i

]

indicates that the

i

th

gas station is

position

i

miles east of the starting position and has

fuel

i

liters of gas.

The car starts with an infinite tank of gas, which initially has

startFuel

liters of fuel in it. It uses one liter of gas per one mile that it drives. When the car reaches a gas station, it may stop and refuel, transferring all the gas from the station into the car.

Return

the minimum number of refueling stops the car must make in order to reach its destination

. If it cannot reach the destination, return

-1

.

Note that if the car reaches a gas station with

0

fuel left, the car can still refuel there. If the car reaches the destination with

0

fuel left, it is still considered to have arrived.

Example 1:

Input:

target = 1, startFuel = 1, stations = []

Output:

0

Explanation:

We can reach the target without refueling.

Example 2:

Input:

target = 100, startFuel = 1, stations = [[10,100]]

Output:

-1

Explanation:

We can not reach the target (or even the first gas station).

Example 3:

Input:

target = 100, startFuel = 10, stations = [[10,60],[20,30],[30,30],[60,40]]

Output:

2

Explanation:

We start with 10 liters of fuel. We drive to position 10, expending 10 liters of fuel. We refuel from 0 liters to 60 liters of gas. Then, we drive from position 10 to position 60 (expending 50 liters of fuel), and refuel from 10 liters to 50 liters of gas. We then drive to and reach the target. We made 2 refueling stops along the way, so we return 2.

Constraints:

1 <= target, startFuel <= 10

9

0 <= stations.length <= 500

1 <= position

i

< position

i+1

< target

1 <= fuel

i

< 10

9

Code Snippets

C++:

```
class Solution {
public:
int minRefuelStops(int target, int startFuel, vector<vector<int>>& stations)
{

}
};
```

Java:

```
class Solution {
public int minRefuelStops(int target, int startFuel, int[][] stations) {

}
}
```

Python3:

```
class Solution:
def minRefuelStops(self, target: int, startFuel: int, stations:
List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def minRefuelStops(self, target, startFuel, stations):
"""
:type target: int
:type startFuel: int
:type stations: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} target
* @param {number} startFuel
* @param {number[][]} stations
```

```
* @return {number}
*/
var minRefuelStops = function(target, startFuel, stations) {

};
```

TypeScript:

```
function minRefuelStops(target: number, startFuel: number, stations:
number[][]): number {

};
```

C#:

```
public class Solution {
public int MinRefuelStops(int target, int startFuel, int[][] stations) {

}
}
```

C:

```
int minRefuelStops(int target, int startFuel, int** stations, int
stationsSize, int* stationsColSize) {

}
```

Go:

```
func minRefuelStops(target int, startFuel int, stations [][]int) int {

}
```

Kotlin:

```
class Solution {
fun minRefuelStops(target: Int, startFuel: Int, stations: Array<IntArray>):
Int {

}
}
```

Swift:

```
class Solution {
func minRefuelStops(_ target: Int, _ startFuel: Int, _ stations: [[Int]]) ->
Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_refuel_stops(target: i32, start_fuel: i32, stations:
Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} target
# @param {Integer} start_fuel
# @param {Integer[][]} stations
# @return {Integer}
def min_refuel_stops(target, start_fuel, stations)

end
```

PHP:

```
class Solution {

/**
* @param Integer $target
* @param Integer $startFuel
* @param Integer[][] $stations
* @return Integer
*/
function minRefuelStops($target, $startFuel, $stations) {

}
}
```

Dart:

```
class Solution {
int minRefuelStops(int target, int startFuel, List<List<int>> stations) {

}
}
```

Scala:

```
object Solution {
def minRefuelStops(target: Int, startFuel: Int, stations: Array[Array[Int]]):
Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_refuel_stops(target :: integer, start_fuel :: integer, stations ::
[[integer]]) :: integer
def min_refuel_stops(target, start_fuel, stations) do

end
end
```

Erlang:

```
-spec min_refuel_stops(Target :: integer(), StartFuel :: integer(), Stations
:: [[integer()]]) -> integer().
min_refuel_stops(Target, StartFuel, Stations) ->
.
```

Racket:

```
(define/contract (min-refuel-stops target startFuel stations)
(-> exact-integer? exact-integer? (listof (listof exact-integer?))
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Number of Refueling Stops
* Difficulty: Hard
* Tags: array, dp, greedy, queue, heap
*
* 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 minRefuelStops(int target, int startFuel, vector<vector<int>>& stations)
{

}
};
```

Java Solution:

```
/**
* Problem: Minimum Number of Refueling Stops
* Difficulty: Hard
* Tags: array, dp, greedy, queue, heap
*
* 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 minRefuelStops(int target, int startFuel, int[][] stations) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Number of Refueling Stops
Difficulty: Hard
Tags: array, dp, greedy, queue, heap

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 minRefuelStops(self, target: int, startFuel: int, stations:
List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minRefuelStops(self, target, startFuel, stations):
"""
:type target: int
:type startFuel: int
:type stations: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Number of Refueling Stops
* Difficulty: Hard
* Tags: array, dp, greedy, queue, heap
*
* 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} target
* @param {number} startFuel
```

```
* @param {number[][]} stations
* @return {number}
*/
var minRefuelStops = function(target, startFuel, stations) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Number of Refueling Stops
* Difficulty: Hard
* Tags: array, dp, greedy, queue, heap
*
* 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 minRefuelStops(target: number, startFuel: number, stations:
number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Number of Refueling Stops
* Difficulty: Hard
* Tags: array, dp, greedy, queue, heap
*
* 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 MinRefuelStops(int target, int startFuel, int[][] stations) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Number of Refueling Stops
* Difficulty: Hard
* Tags: array, dp, greedy, queue, heap
*
* 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 minRefuelStops(int target, int startFuel, int** stations, int
stationsSize, int* stationsColSize) {

}
```

Go Solution:

```
// Problem: Minimum Number of Refueling Stops
// Difficulty: Hard
// Tags: array, dp, greedy, queue, heap
//
// 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 minRefuelStops(target int, startFuel int, stations [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minRefuelStops(target: Int, startFuel: Int, stations: Array<IntArray>):
Int {

}
}
```

Swift Solution:

```
class Solution {
func minRefuelStops(_ target: Int, _ startFuel: Int, _ stations: [[Int]]) ->
Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Number of Refueling Stops
// Difficulty: Hard
// Tags: array, dp, greedy, queue, heap
//
// 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_refuel_stops(target: i32, start_fuel: i32, stations:
Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} target
# @param {Integer} start_fuel
# @param {Integer[][]} stations
# @return {Integer}
def min_refuel_stops(target, start_fuel, stations)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $target
* @param Integer $startFuel
* @param Integer[][] $stations
```

```
* @return Integer
*/
function minRefuelStops($target, $startFuel, $stations) {

}
}
```

Dart Solution:

```
class Solution {
int minRefuelStops(int target, int startFuel, List<List<int>> stations) {

}
}
```

Scala Solution:

```
object Solution {
def minRefuelStops(target: Int, startFuel: Int, stations: Array[Array[Int]]):
Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_refuel_stops(target :: integer, start_fuel :: integer, stations ::
[[integer]]) :: integer
def min_refuel_stops(target, start_fuel, stations) do

end
end
```

Erlang Solution:

```
-spec min_refuel_stops(Target :: integer(), StartFuel :: integer(), Stations
:: [[integer()]]) -> integer().
min_refuel_stops(Target, StartFuel, Stations) ->
.
```

Racket Solution:

```
(define/contract (min-refuel-stops target startFuel stations)
(-> exact-integer? exact-integer? (listof (listof exact-integer?))
exact-integer?)
)
```