

Problem 3466: Maximum Coin Collection

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Mario drives on a two-lane freeway with coins every mile. You are given two integer arrays,

lane1

and

lane2

, where the value at the

i

th

index represents the number of coins he

gains or loses

in the

i

th

mile in that lane.

If Mario is in lane 1 at mile

i

and

$\text{lane1}[i] > 0$

, Mario gains

$\text{lane1}[i]$

coins.

If Mario is in lane 1 at mile

i

and

$\text{lane1}[i] < 0$

, Mario pays a toll and loses

$\text{abs}(\text{lane1}[i])$

coins.

The same rules apply for

lane2

.

Mario can enter the freeway anywhere and exit anytime after traveling

at least

one mile. Mario always enters the freeway on lane 1 but can switch lanes

at most

2 times.

A

lane switch

is when Mario goes from lane 1 to lane 2 or vice versa.

Return the

maximum

number of coins Mario can earn after performing

at most 2 lane switches

.

Note:

Mario can switch lanes immediately upon entering or just before exiting the freeway.

Example 1:

Input:

lane1 = [1,-2,-10,3], lane2 = [-5,10,0,1]

Output:

14

Explanation:

Mario drives the first mile on lane 1.

He then changes to lane 2 and drives for two miles.

He changes back to lane 1 for the last mile.

Mario collects

$$1 + 10 + 0 + 3 = 14$$

coins.

Example 2:

Input:

lane1 = [1,-1,-1,-1], lane2 = [0,3,4,-5]

Output:

8

Explanation:

Mario starts at mile 0 in lane 1 and drives one mile.

He then changes to lane 2 and drives for two more miles. He exits the freeway before mile 3.

He collects

$$1 + 3 + 4 = 8$$

coins.

Example 3:

Input:

lane1 = [-5,-4,-3], lane2 = [-1,2,3]

Output:

5

Explanation:

Mario enters at mile 1 and immediately switches to lane 2. He stays here the entire way.

He collects a total of

$$2 + 3 = 5$$

coins.

Example 4:

Input:

lane1 = [-3,-3,-3], lane2 = [9,-2,4]

Output:

11

Explanation:

Mario starts at the beginning of the freeway and immediately switches to lane 2. He stays here the whole way.

He collects a total of

$$9 + (-2) + 4 = 11$$

coins.

Example 5:

Input:

lane1 = [-10], lane2 = [-2]

Output:

-2

Explanation:

Since Mario must ride on the freeway for at least one mile, he rides just one mile in lane 2.

He collects a total of -2 coins.

Constraints:

$1 \leq \text{lane1.length} == \text{lane2.length} \leq 10$

5

-10

9

$\text{lane1}[i], \text{lane2}[i] \leq 10$

9

Code Snippets

C++:

```
class Solution {
public:
    long long maxCoins(vector<int>& lane1, vector<int>& lane2) {

    }
};
```

Java:

```
class Solution {
    public long maxCoins(int[] lane1, int[] lane2) {
```

```
}  
}
```

Python3:

```
class Solution:  
    def maxCoins(self, lane1: List[int], lane2: List[int]) -> int:
```

Python:

```
class Solution(object):  
    def maxCoins(self, lane1, lane2):  
        """  
        :type lane1: List[int]  
        :type lane2: List[int]  
        :rtype: int  
        """
```

JavaScript:

```
/**  
 * @param {number[]} lane1  
 * @param {number[]} lane2  
 * @return {number}  
 */  
var maxCoins = function(lane1, lane2) {  
  
};
```

TypeScript:

```
function maxCoins(lane1: number[], lane2: number[]): number {  
  
};
```

C#:

```
public class Solution {  
    public long MaxCoins(int[] lane1, int[] lane2) {  
  
    }  
}
```

```
}
```

C:

```
long long maxCoins(int* lane1, int lane1Size, int* lane2, int lane2Size) {  
  
}
```

Go:

```
func maxCoins(lane1 []int, lane2 []int) int64 {  
  
}
```

Kotlin:

```
class Solution {  
    fun maxCoins(lane1: IntArray, lane2: IntArray): Long {  
  
    }  
}
```

Swift:

```
class Solution {  
    func maxCoins(_ lane1: [Int], _ lane2: [Int]) -> Int {  
  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn max_coins(lane1: Vec<i32>, lane2: Vec<i32>) -> i64 {  
  
    }  
}
```

Ruby:

```
# @param {Integer[]} lane1  
# @param {Integer[]} lane2
```

```
# @return {Integer}
def max_coins(lane1, lane2)

end
```

PHP:

```
class Solution {

    /**
     * @param Integer[] $lane1
     * @param Integer[] $lane2
     * @return Integer
     */
    function maxCoins($lane1, $lane2) {

    }

}
```

Dart:

```
class Solution {
  int maxCoins(List<int> lane1, List<int> lane2) {

  }

}
```

Scala:

```
object Solution {
  def maxCoins(lane1: Array[Int], lane2: Array[Int]): Long = {

  }

}
```

Elixir:

```
defmodule Solution do
  @spec max_coins(lane1 :: [integer], lane2 :: [integer]) :: integer
  def max_coins(lane1, lane2) do

  end
end
```

```
end
```

Erlang:

```
-spec max_coins(Lane1 :: [integer()], Lane2 :: [integer()]) -> integer().  
max_coins(Lane1, Lane2) ->  
.
```

Racket:

```
(define/contract (max-coins lane1 lane2)  
  (-> (listof exact-integer?) (listof exact-integer?) exact-integer?)  
  )
```

Solutions

C++ Solution:

```
/*  
 * Problem: Maximum Coin Collection  
 * Difficulty: Medium  
 * 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:  
    long long maxCoins(vector<int>& lane1, vector<int>& lane2) {  
  
    }  
};
```

Java Solution:

```
/**  
 * Problem: Maximum Coin Collection  
 * Difficulty: Medium
```

```

* 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 long maxCoins(int[] lane1, int[] lane2) {

}
}

```

Python3 Solution:

```

"""
Problem: Maximum Coin Collection
Difficulty: Medium
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 maxCoins(self, lane1: List[int], lane2: List[int]) -> int:
# TODO: Implement optimized solution
pass

```

Python Solution:

```

class Solution(object):
def maxCoins(self, lane1, lane2):
"""
:type lane1: List[int]
:type lane2: List[int]
:rtype: int
"""

```

JavaScript Solution:

```

/**
 * Problem: Maximum Coin Collection
 * Difficulty: Medium
 * 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[]} lane1
 * @param {number[]} lane2
 * @return {number}
 */
var maxCoins = function(lane1, lane2) {

};

```

TypeScript Solution:

```

/**
 * Problem: Maximum Coin Collection
 * Difficulty: Medium
 * 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 maxCoins(lane1: number[], lane2: number[]): number {

};

```

C# Solution:

```

/*
 * Problem: Maximum Coin Collection
 * Difficulty: Medium
 * 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 long MaxCoins(int[] lane1, int[] lane2) {

}

}

```

C Solution:

```

/*
* Problem: Maximum Coin Collection
* Difficulty: Medium
* 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
*/

long long maxCoins(int* lane1, int lane1Size, int* lane2, int lane2Size) {

}

```

Go Solution:

```

// Problem: Maximum Coin Collection
// Difficulty: Medium
// 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 maxCoins(lane1 []int, lane2 []int) int64 {

}

```

Kotlin Solution:

```
class Solution {  
    fun maxCoins(lane1: IntArray, lane2: IntArray): Long {  
  
    }  
}
```

Swift Solution:

```
class Solution {  
    func maxCoins(_ lane1: [Int], _ lane2: [Int]) -> Int {  
  
    }  
}
```

Rust Solution:

```
// Problem: Maximum Coin Collection  
// Difficulty: Medium  
// 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 max_coins(lane1: Vec<i32>, lane2: Vec<i32>) -> i64 {  
  
    }  
}
```

Ruby Solution:

```
# @param {Integer[]} lane1  
# @param {Integer[]} lane2  
# @return {Integer}  
def max_coins(lane1, lane2)  
  
end
```

PHP Solution:

```

class Solution {

    /**
     * @param Integer[] $lane1
     * @param Integer[] $lane2
     * @return Integer
     */
    function maxCoins($lane1, $lane2) {

    }

}

```

Dart Solution:

```

class Solution {
  int maxCoins(List<int> lane1, List<int> lane2) {

  }

}

```

Scala Solution:

```

object Solution {
  def maxCoins(lane1: Array[Int], lane2: Array[Int]): Long = {

  }

}

```

Elixir Solution:

```

defmodule Solution do
  @spec max_coins(lane1 :: [integer], lane2 :: [integer]) :: integer
  def max_coins(lane1, lane2) do

  end

end

```

Erlang Solution:

```

-spec max_coins(Lane1 :: [integer()], Lane2 :: [integer()]) -> integer().
max_coins(Lane1, Lane2) ->

.

```

Racket Solution:

```
(define/contract (max-coins lane1 lane2)
  (-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
  )
```