Problem 120: Triangle

Problem Information

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

Problem Description

Given a

triangle

array, return

the minimum path sum from top to bottom

.

For each step, you may move to an adjacent number of the row below. More formally, if you are on index

i

on the current row, you may move to either index

i

or index

i + 1

on the next row.

Example 1:

Input:

triangle = [[2],[3,4],[6,5,7],[4,1,8,3]]

Output:

11

Explanation:

The triangle looks like:

2

3

4 6

5

7 4

1

8 3 The minimum path sum from top to bottom is 2 + 3 + 5 + 1 = 11 (underlined above).

Example 2:

Input:

triangle = [[-10]]

Output:

-10

Constraints:

1 <= triangle.length <= 200

triangle[0].length == 1

triangle[i].length == triangle[i - 1].length + 1

-10

4

<= triangle[i][j] <= 10

4

Follow up:

Could you do this using only

O(n)

extra space, where

n

is the total number of rows in the triangle?

Code Snippets

C++:

```
class Solution {
public:
int minimumTotal(vector<vector<int>>& triangle) {

}
};
```

Java:

```
class Solution {
public int minimumTotal(List<List<Integer>> triangle) {

}
}
```

Python3:

```
class Solution:
def minimumTotal(self, triangle: List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def minimumTotal(self, triangle):
"""
:type triangle: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} triangle
* @return {number}
*/
var minimumTotal = function(triangle) {

};
```

TypeScript:

```
function minimumTotal(triangle: number[][]): number {

};
```

C#:

```
public class Solution {
public int MinimumTotal(IList<IList<int>> triangle) {

}
}
```

C:

```
int minimumTotal(int** triangle, int triangleSize, int* triangleColSize) {

}
```

Go:

```
func minimumTotal(triangle [][]int) int {

}
```

Kotlin:

```
class Solution {
fun minimumTotal(triangle: List<List<Int>>): Int {

}
}
```

Swift:

```
class Solution {
func minimumTotal(_ triangle: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_total(triangle: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} triangle
# @return {Integer}
def minimum_total(triangle)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $triangle
* @return Integer
*/
function minimumTotal($triangle) {

}
}
```

Dart:

```
class Solution {
int minimumTotal(List<List<int>> triangle) {

}
}
```

Scala:

```
object Solution {
def minimumTotal(triangle: List[List[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_total(triangle :: [[integer]]) :: integer
def minimum_total(triangle) do

end
end
```

Erlang:

```
-spec minimum_total(Triangle :: [[integer()]]) -> integer().
minimum_total(Triangle) ->
.
```

Racket:

```
(define/contract (minimum-total triangle)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Triangle
* 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:
int minimumTotal(vector<vector<int>>& triangle) {

}
};
```

Java Solution:

```
/**
* Problem: Triangle
* 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 int minimumTotal(List<List<Integer>> triangle) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Triangle
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 minimumTotal(self, triangle: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minimumTotal(self, triangle):
"""
:type triangle: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Triangle
* 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[][]} triangle
* @return {number}
*/
var minimumTotal = function(triangle) {

};
```

TypeScript Solution:

```
/**
* Problem: Triangle
* 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 minimumTotal(triangle: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Triangle
* 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 int MinimumTotal(IList<IList<int>> triangle) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Triangle
* 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
*/

int minimumTotal(int** triangle, int triangleSize, int* triangleColSize) {

}
```

Go Solution:

```
// Problem: Triangle
// 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 minimumTotal(triangle [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumTotal(triangle: List<List<Int>>): Int {

}
}
```

Swift Solution:

```
class Solution {
func minimumTotal(_ triangle: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Triangle
// 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 minimum_total(triangle: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} triangle
# @return {Integer}
def minimum_total(triangle)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $triangle
* @return Integer
*/
function minimumTotal($triangle) {

}
}
```

Dart Solution:

```
class Solution {
int minimumTotal(List<List<int>> triangle) {

}
}
```

Scala Solution:

```
object Solution {
def minimumTotal(triangle: List[List[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_total(triangle :: [[integer]]) :: integer
def minimum_total(triangle) do

end
end
```

Erlang Solution:

```
-spec minimum_total(Triangle :: [[integer()]]) -> integer().
minimum_total(Triangle) ->
.
```

Racket Solution:

```
(define/contract (minimum-total triangle)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```