# Problem 354: Russian Doll Envelopes

## Problem Information

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

## Problem Description

You are given a 2D array of integers

envelopes

where

envelopes[i] = [w

i

, h

i

]

represents the width and the height of an envelope.

One envelope can fit into another if and only if both the width and height of one envelope are greater than the other envelope's width and height.

Return

the maximum number of envelopes you can Russian doll (i.e., put one inside the other)

.

Note:

You cannot rotate an envelope.

Example 1:

Input:

envelopes = [[5,4],[6,4],[6,7],[2,3]]

Output:

3

Explanation:

The maximum number of envelopes you can Russian doll is

3

([2,3] => [5,4] => [6,7]).

Example 2:

Input:

envelopes = [[1,1],[1,1],[1,1]]

Output:

1

Constraints:

1 <= envelopes.length <= 10

5

envelopes[i].length == 2

1 <= w

i

, h

i

<= 10

5

## Code Snippets

**C++:**

```
class Solution {
public:
int maxEnvelopes(vector<vector<int>>& envelopes) {

}
};
```

**Java:**

```
class Solution {
public int maxEnvelopes(int[][] envelopes) {

}
}
```

**Python3:**

```
class Solution:
def maxEnvelopes(self, envelopes: List[List[int]]) -> int:
```

**Python:**

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

**JavaScript:**

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

};
```

**TypeScript:**

```
function maxEnvelopes(envelopes: number[][]): number {

};
```

**C#:**

```
public class Solution {
public int MaxEnvelopes(int[][] envelopes) {

}
}
```

**C:**

```
int maxEnvelopes(int** envelopes, int envelopesSize, int* envelopesColSize) {

}
```

**Go:**

```
func maxEnvelopes(envelopes [][]int) int {

}
```

**Kotlin:**

```
class Solution {
fun maxEnvelopes(envelopes: Array<IntArray>): Int {

}
}
```

**Swift:**

```
class Solution {
func maxEnvelopes(_ envelopes: [[Int]]) -> Int {

}
}
```

**Rust:**

```
impl Solution {
pub fn max_envelopes(envelopes: Vec<Vec<i32>>) -> i32 {

}
}
```

**Ruby:**

```
# @param {Integer[][]} envelopes
# @return {Integer}
def max_envelopes(envelopes)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer[][] $envelopes
* @return Integer
*/
function maxEnvelopes($envelopes) {

}
}
```

**Dart:**

```
class Solution {
int maxEnvelopes(List<List<int>> envelopes) {

}
}
```

**Scala:**

```
object Solution {
def maxEnvelopes(envelopes: Array[Array[Int]]): Int = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec max_envelopes(envelopes :: [[integer]]) :: integer
def max_envelopes(envelopes) do

end
end
```

**Erlang:**

```
-spec max_envelopes(Envelopes :: [[integer()]]) -> integer().
max_envelopes(Envelopes) ->
.
```

**Racket:**

```
(define/contract (max-envelopes envelopes)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

## Solutions

**C++ Solution:**

```
/*
* Problem: Russian Doll Envelopes
* Difficulty: Hard
* Tags: array, dp, 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 maxEnvelopes(vector<vector<int>>& envelopes) {

}
};
```

**Java Solution:**

```
/**
* Problem: Russian Doll Envelopes
* Difficulty: Hard
* Tags: array, dp, 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 maxEnvelopes(int[][] envelopes) {

}
}
```

**Python3 Solution:**

```
"""
Problem: Russian Doll Envelopes
Difficulty: Hard
Tags: array, dp, 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 maxEnvelopes(self, envelopes: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

### Python Solution:

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

### JavaScript Solution:

```
/**
* Problem: Russian Doll Envelopes
* Difficulty: Hard
* Tags: array, dp, 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[][]} envelopes
* @return {number}
*/
var maxEnvelopes = function(envelopes) {

};
```

### TypeScript Solution:

```
/**
* Problem: Russian Doll Envelopes
* Difficulty: Hard
* Tags: array, dp, 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 maxEnvelopes(envelopes: number[][]): number {

};
```

### C# Solution:

```
/*
* Problem: Russian Doll Envelopes
* Difficulty: Hard
* Tags: array, dp, 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 MaxEnvelopes(int[][] envelopes) {

}
}
```

### C Solution:

```
/*
* Problem: Russian Doll Envelopes
* Difficulty: Hard
* Tags: array, dp, 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 maxEnvelopes(int** envelopes, int envelopesSize, int* envelopesColSize) {

}
```

**Go Solution:**

```
// Problem: Russian Doll Envelopes
// Difficulty: Hard
// Tags: array, dp, 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 maxEnvelopes(envelopes [][]int) int {

}
```

**Kotlin Solution:**

```
class Solution {
fun maxEnvelopes(envelopes: Array<IntArray>): Int {

}
}
```

**Swift Solution:**

```
class Solution {
func maxEnvelopes(_ envelopes: [[Int]]) -> Int {

}
}
```

**Rust Solution:**

```
// Problem: Russian Doll Envelopes
// Difficulty: Hard
// Tags: array, dp, 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_envelopes(envelopes: Vec<Vec<i32>>) -> i32 {

}
}
```

### Ruby Solution:

```
# @param {Integer[][]} envelopes
# @return {Integer}
def max_envelopes(envelopes)

end
```

### PHP Solution:

```
class Solution {

/**
* @param Integer[][] $envelopes
* @return Integer
*/
function maxEnvelopes($envelopes) {

}
}
```

### Dart Solution:

```
class Solution {
int maxEnvelopes(List<List<int>> envelopes) {

}
}
```

### Scala Solution:

```
object Solution {
def maxEnvelopes(envelopes: Array[Array[Int]]): Int = {

}
}
```

### Elixir Solution:

```
defmodule Solution do
@spec max_envelopes(envelopes :: [[integer]]) :: integer
def max_envelopes(envelopes) do

end
end
```

### Erlang Solution:

```
-spec max_envelopes(Envelopes :: [[integer()]]) -> integer().
max_envelopes(Envelopes) ->
.
```

### Racket Solution:

```
(define/contract (max-envelopes envelopes)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```