Problem 1105: Filling Bookcase Shelves

Problem Information

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

Problem Description

You are given an array

books

where

books[i] = [thickness

i

, height

i

]

indicates the thickness and height of the

i

th

book. You are also given an integer

shelfWidth

.

We want to place these books in order onto bookcase shelves that have a total width

shelfWidth

.

We choose some of the books to place on this shelf such that the sum of their thickness is less than or equal to

shelfWidth

, then build another level of the shelf of the bookcase so that the total height of the bookcase has increased by the maximum height of the books we just put down. We repeat this process until there are no more books to place.

Note that at each step of the above process, the order of the books we place is the same order as the given sequence of books.

For example, if we have an ordered list of

5

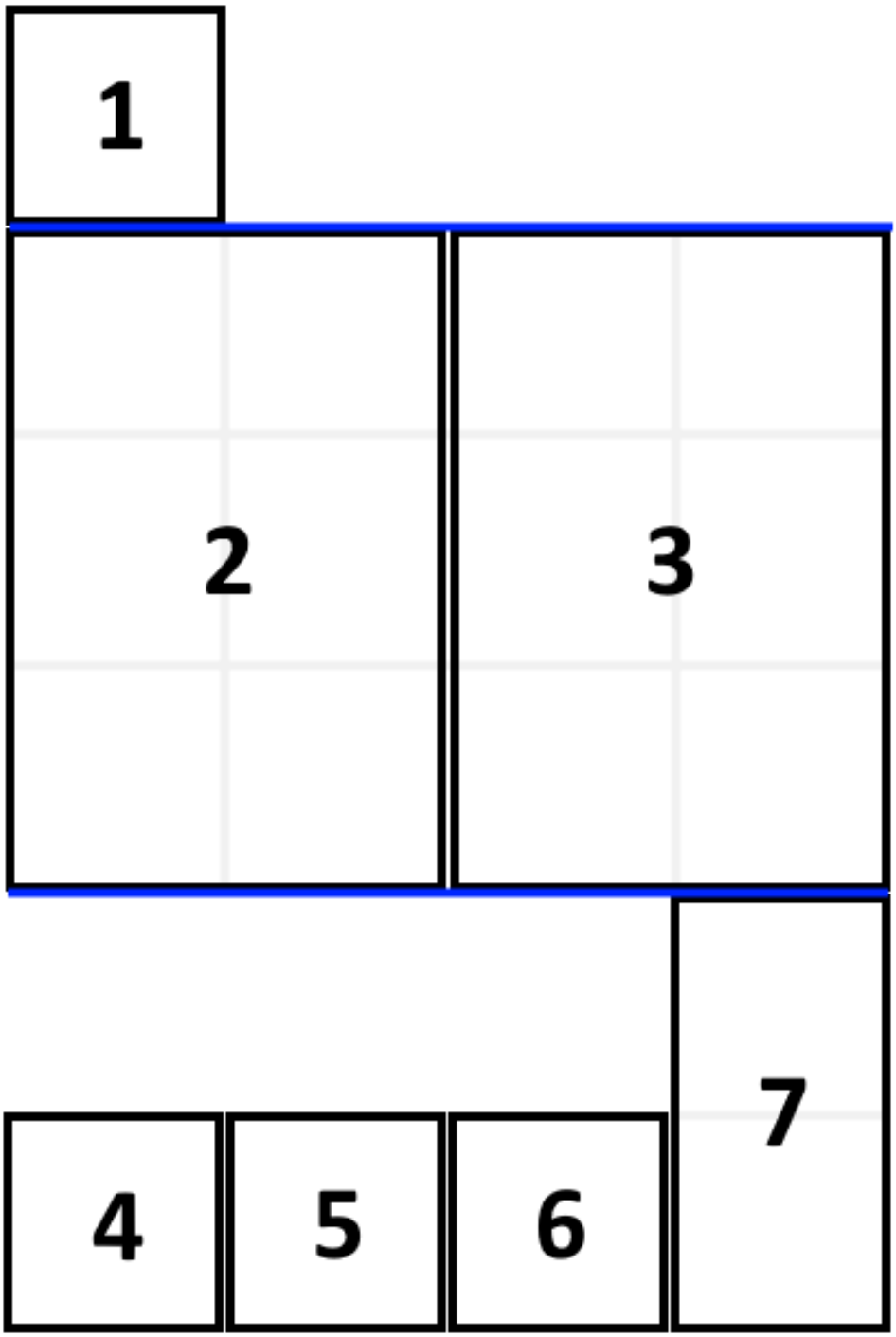
books, we might place the first and second book onto the first shelf, the third book on the second shelf, and the fourth and fifth book on the last shelf.

Return

the minimum possible height that the total bookshelf can be after placing shelves in this manner

.

Example 1:



Input:

books = [[1,1],[2,3],[2,3],[1,1],[1,1],[1,1],[1,2]], shelfWidth = 4

Output:

6

Explanation:

The sum of the heights of the 3 shelves is 1 + 3 + 2 = 6. Notice that book number 2 does not have to be on the first shelf.

Example 2:

Input:

books = [[1,3],[2,4],[3,2]], shelfWidth = 6

Output:

4

Constraints:

1 <= books.length <= 1000

1 <= thickness

i

<= shelfWidth <= 1000

1 <= height

i

<= 1000

Code Snippets

C++:

```
class Solution {
public:
int minHeightShelves(vector<vector<int>>& books, int shelfWidth) {
```

```
}
};
```

Java:

```
class Solution {
public int minHeightShelves(int[][] books, int shelfWidth) {

}
}
```

Python3:

```
class Solution:
def minHeightShelves(self, books: List[List[int]], shelfWidth: int) -> int:
```

Python:

```
class Solution(object):
def minHeightShelves(self, books, shelfWidth):
"""
:type books: List[List[int]]
:type shelfWidth: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} books
* @param {number} shelfWidth
* @return {number}
*/
var minHeightShelves = function(books, shelfWidth) {

};
```

TypeScript:

```
function minHeightShelves(books: number[][], shelfWidth: number): number {
```

```
};
```

C#:

```
public class Solution {
public int MinHeightShelves(int[][] books, int shelfWidth) {

}
}
```

C:

```
int minHeightShelves(int** books, int booksSize, int* booksColSize, int
shelfWidth) {

}
```

Go:

```
func minHeightShelves(books [][]int, shelfWidth int) int {

}
```

Kotlin:

```
class Solution {
fun minHeightShelves(books: Array<IntArray>, shelfWidth: Int): Int {

}
}
```

Swift:

```
class Solution {
func minHeightShelves(_ books: [[Int]], _ shelfWidth: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_height_shelves(books: Vec<Vec<i32>>, shelf_width: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} books
# @param {Integer} shelf_width
# @return {Integer}
def min_height_shelves(books, shelf_width)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $books
* @param Integer $shelfWidth
* @return Integer
*/
function minHeightShelves($books, $shelfWidth) {

}
}
```

Dart:

```
class Solution {
int minHeightShelves(List<List<int>> books, int shelfWidth) {

}
}
```

Scala:

```
object Solution {
def minHeightShelves(books: Array[Array[Int]], shelfWidth: Int): Int = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec min_height_shelves(books :: [[integer]], shelf_width :: integer) ::
integer
def min_height_shelves(books, shelf_width) do

end
end
```

Erlang:

```
-spec min_height_shelves(Books :: [[integer()]], ShelfWidth :: integer()) ->
integer().
min_height_shelves(Books, ShelfWidth) ->
.
```

Racket:

```
(define/contract (min-height-shelves books shelfWidth)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Filling Bookcase Shelves
* 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 minHeightShelves(vector<vector<int>>& books, int shelfWidth) {

}
};
```

Java Solution:

```
/**
* Problem: Filling Bookcase Shelves
* 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 minHeightShelves(int[][] books, int shelfWidth) {

}
}
```

Python3 Solution:

```
"""
Problem: Filling Bookcase Shelves
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 minHeightShelves(self, books: List[List[int]], shelfWidth: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minHeightShelves(self, books, shelfWidth):
"""
:type books: List[List[int]]
:type shelfWidth: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Filling Bookcase Shelves
* 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[][]} books
* @param {number} shelfWidth
* @return {number}
*/
var minHeightShelves = function(books, shelfWidth) {

};
```

TypeScript Solution:

```
/**
* Problem: Filling Bookcase Shelves
* 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 minHeightShelves(books: number[][], shelfWidth: number): number {

};
```

C# Solution:

```
/*
* Problem: Filling Bookcase Shelves
* 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 MinHeightShelves(int[][] books, int shelfWidth) {

}
}
```

C Solution:

```
/*
* Problem: Filling Bookcase Shelves
* 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 minHeightShelves(int** books, int booksSize, int* booksColSize, int
shelfWidth) {

}
```

Go Solution:

```
// Problem: Filling Bookcase Shelves
// 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 minHeightShelves(books [][]int, shelfWidth int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minHeightShelves(books: Array<IntArray>, shelfWidth: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minHeightShelves(_ books: [[Int]], _ shelfWidth: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Filling Bookcase Shelves
// 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 min_height_shelves(books: Vec<Vec<i32>>, shelf_width: i32) -> i32 {

}
```

```
}
```

Ruby Solution:

```
# @param {Integer[][]} books
# @param {Integer} shelf_width
# @return {Integer}
def min_height_shelves(books, shelf_width)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $books
* @param Integer $shelfWidth
* @return Integer
*/
function minHeightShelves($books, $shelfWidth) {

}
}
```

Dart Solution:

```
class Solution {
int minHeightShelves(List<List<int>> books, int shelfWidth) {

}
}
```

Scala Solution:

```
object Solution {
def minHeightShelves(books: Array[Array[Int]], shelfWidth: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_height_shelves(books :: [[integer]], shelf_width :: integer) ::
integer
def min_height_shelves(books, shelf_width) do

end
end
```

Erlang Solution:

```
-spec min_height_shelves(Books :: [[integer()]], ShelfWidth :: integer()) ->
integer().
min_height_shelves(Books, ShelfWidth) ->
.
```

Racket Solution:

```
(define/contract (min-height-shelves books shelfWidth)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```