

Problem 2355: Maximum Number of Books You Can Take

Problem Information

Difficulty: Hard

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given a

0-indexed

integer array

books

of length

n

where

$\text{books}[i]$

denotes the number of books on the

i

th

shelf of a bookshelf.

You are going to take books from a

contiguous

section of the bookshelf spanning from

l

to

r

where

$0 \leq l \leq r < n$

. For each index

i

in the range

$l \leq i < r$

, you must take

strictly fewer

books from shelf

i

than shelf

$i + 1$

.

Return

the

maximum

number of books you can take from the bookshelf.

Example 1:

Input:

books = [8,5,2,7,9]

Output:

19

Explanation:

- Take 1 book from shelf 1. - Take 2 books from shelf 2. - Take 7 books from shelf 3. - Take 9 books from shelf 4. You have taken 19 books, so return 19. It can be proven that 19 is the maximum number of books you can take.

Example 2:

Input:

books = [7,0,3,4,5]

Output:

12

Explanation:

- Take 3 books from shelf 2. - Take 4 books from shelf 3. - Take 5 books from shelf 4. You have taken 12 books so return 12. It can be proven that 12 is the maximum number of books you can take.

Example 3:

Input:

```
books = [8,2,3,7,3,4,0,1,4,3]
```

Output:

```
13
```

Explanation:

- Take 1 book from shelf 0. - Take 2 books from shelf 1. - Take 3 books from shelf 2. - Take 7 books from shelf 3. You have taken 13 books so return 13. It can be proven that 13 is the maximum number of books you can take.

Constraints:

```
1 <= books.length <= 10
```

```
5
```

```
0 <= books[i] <= 10
```

```
5
```

Code Snippets

C++:

```
class Solution {
public:
    long long maximumBooks(vector<int>& books) {
        }
};
```

Java:

```
class Solution {
public long maximumBooks(int[] books) {
```

```
}
```

```
}
```

Python3:

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

Python:

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

JavaScript:

```
/**  
 * @param {number[]} books  
 * @return {number}  
 */  
var maximumBooks = function(books) {  
  
};
```

TypeScript:

```
function maximumBooks(books: number[]): number {  
  
};
```

C#:

```
public class Solution {  
    public long MaximumBooks(int[] books) {  
  
    }  
}
```

C:

```
long long maximumBooks(int* books, int booksSize) {  
  
}
```

Go:

```
func maximumBooks(books []int) int64 {  
  
}
```

Kotlin:

```
class Solution {  
    fun maximumBooks(books: IntArray): Long {  
  
    }  
}
```

Swift:

```
class Solution {  
    func maximumBooks(_ books: [Int]) -> Int {  
  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn maximum_books(books: Vec<i32>) -> i64 {  
  
    }  
}
```

Ruby:

```
# @param {Integer[]} books  
# @return {Integer}  
def maximum_books(books)  
  
end
```

PHP:

```
class Solution {  
  
    /**  
     * @param Integer[] $books  
     * @return Integer  
     */  
    function maximumBooks($books) {  
  
    }  
}
```

Dart:

```
class Solution {  
int maximumBooks(List<int> books) {  
  
}  
}
```

Scala:

```
object Solution {  
def maximumBooks(books: Array[Int]): Long = {  
  
}  
}
```

Elixir:

```
defmodule Solution do  
@spec maximum_books(list(integer)) :: integer  
def maximum_books(books) do  
  
end  
end
```

Erlang:

```
-spec maximum_books(list(integer())) -> integer().  
maximum_books(Books) ->  
.
```

Racket:

```
(define/contract (maximum-books books)
  (-> (listof exact-integer?) exact-integer?))
```

Solutions

C++ Solution:

```
/*
 * Problem: Maximum Number of Books You Can Take
 * Difficulty: Hard
 * Tags: array, dp, stack
 *
 * 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 maximumBooks(vector<int>& books) {

    }
};
```

Java Solution:

```
/**
 * Problem: Maximum Number of Books You Can Take
 * Difficulty: Hard
 * Tags: array, dp, stack
 *
 * 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 maximumBooks(int[] books) {
```

```
}
```

```
}
```

Python3 Solution:

```
"""
Problem: Maximum Number of Books You Can Take
Difficulty: Hard
Tags: array, dp, stack

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 maximumBooks(self, books: List[int]) -> int:
        # TODO: Implement optimized solution
        pass
```

Python Solution:

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


```

JavaScript Solution:

```
/**
 * Problem: Maximum Number of Books You Can Take
 * Difficulty: Hard
 * Tags: array, dp, stack
 *
 * 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
 * @return {number}
 */
var maximumBooks = function(books) {

};

```

TypeScript Solution:

```

/**
 * Problem: Maximum Number of Books You Can Take
 * Difficulty: Hard
 * Tags: array, dp, stack
 *
 * 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 maximumBooks(books: number[]): number {

};

```

C# Solution:

```

/*
 * Problem: Maximum Number of Books You Can Take
 * Difficulty: Hard
 * Tags: array, dp, stack
 *
 * 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 MaximumBooks(int[] books) {

    }
}
```

```
}
```

C Solution:

```
/*
 * Problem: Maximum Number of Books You Can Take
 * Difficulty: Hard
 * Tags: array, dp, stack
 *
 * 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 maximumBooks(int* books, int booksSize) {

}
```

Go Solution:

```
// Problem: Maximum Number of Books You Can Take
// Difficulty: Hard
// Tags: array, dp, stack
//
// 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 maximumBooks(books []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
    fun maximumBooks(books: IntArray): Long {
        return 0
    }
}
```

Swift Solution:

```
class Solution {  
func maximumBooks(_ books: [Int]) -> Int {  
  
}  
}  
}
```

Rust Solution:

```
// Problem: Maximum Number of Books You Can Take  
// Difficulty: Hard  
// Tags: array, dp, stack  
//  
// 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 maximum_books(books: Vec<i32>) -> i64 {  
  
}  
}
```

Ruby Solution:

```
# @param {Integer[]} books  
# @return {Integer}  
def maximum_books(books)  
  
end
```

PHP Solution:

```
class Solution {  
  
/**  
 * @param Integer[] $books  
 * @return Integer  
 */  
function maximumBooks($books) {  
  
}  
}
```

Dart Solution:

```
class Solution {  
    int maximumBooks(List<int> books) {  
  
    }  
}
```

Scala Solution:

```
object Solution {  
    def maximumBooks(books: Array[Int]): Long = {  
  
    }  
}
```

Elixir Solution:

```
defmodule Solution do  
  @spec maximum_books(list(integer())) :: integer()  
  def maximum_books(books) do  
  
  end  
end
```

Erlang Solution:

```
-spec maximum_books(list(integer())) -> integer().  
maximum_books(Books) ->  
.
```

Racket Solution:

```
(define/contract (maximum-books books)  
  (-> (listof exact-integer?) exact-integer?)  
)
```