Problem 1687: Delivering Boxes from Storage to Ports

Problem Information

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

Problem Description

You have the task of delivering some boxes from storage to their ports using only one ship. However, this ship has a

limit

on the

number of boxes

and the

total weight

that it can carry.

You are given an array

boxes

, where

boxes[i] = [ports

i

, weight

i

]

, and three integers

portsCount

,

maxBoxes

, and

maxWeight

.

ports

i

is the port where you need to deliver the

i

th

box and

weights

i

is the weight of the

i

th

box.

portsCount

is the number of ports.

maxBoxes

and

maxWeight

are the respective box and weight limits of the ship.

The boxes need to be delivered

in the order they are given

. The ship will follow these steps:

The ship will take some number of boxes from the

boxes

queue, not violating the

maxBoxes

and

maxWeight

constraints.

For each loaded box

in order

, the ship will make a

trip

to the port the box needs to be delivered to and deliver it. If the ship is already at the correct port, no

trip

is needed, and the box can immediately be delivered.

The ship then makes a return

trip

to storage to take more boxes from the queue.

The ship must end at storage after all the boxes have been delivered.

Return

the

minimum

number of

trips

the ship needs to make to deliver all boxes to their respective ports.

Example 1:

Input:

boxes = [[1,1],[2,1],[1,1]], portsCount = 2, maxBoxes = 3, maxWeight = 3

Output:

4

Explanation:

The optimal strategy is as follows: - The ship takes all the boxes in the queue, goes to port 1, then port 2, then port 1 again, then returns to storage. 4 trips. So the total number of trips is 4. Note that the first and third boxes cannot be delivered together because the boxes need to be delivered in order (i.e. the second box needs to be delivered at port 2 before the third box).

Example 2:

Input:

boxes = [[1,2],[3,3],[3,1],[3,1],[2,4]], portsCount = 3, maxBoxes = 3, maxWeight = 6

Output:

6

Explanation:

The optimal strategy is as follows: - The ship takes the first box, goes to port 1, then returns to storage. 2 trips. - The ship takes the second, third and fourth boxes, goes to port 3, then returns to storage. 2 trips. - The ship takes the fifth box, goes to port 2, then returns to storage. 2 trips. So the total number of trips is 2 + 2 + 2 = 6.

Example 3:

Input:

boxes = [[1,4],[1,2],[2,1],[2,1],[3,2],[3,4]], portsCount = 3, maxBoxes = 6, maxWeight = 7

Output:

6

Explanation:

The optimal strategy is as follows: - The ship takes the first and second boxes, goes to port 1, then returns to storage. 2 trips. - The ship takes the third and fourth boxes, goes to port 2, then returns to storage. 2 trips. - The ship takes the fifth and sixth boxes, goes to port 3, then returns to storage. 2 trips. So the total number of trips is 2 + 2 + 2 = 6.

Constraints:

1 <= boxes.length <= 10

5

1 <= portsCount, maxBoxes, maxWeight <= 10

5

1 <= ports

i

<= portsCount

1 <= weights

i

<= maxWeight

Code Snippets

C++:

```
class Solution {
public:
int boxDelivering(vector<vector<int>>& boxes, int portsCount, int maxBoxes,
int maxWeight) {

}
};
```

Java:

```
class Solution {
public int boxDelivering(int[][] boxes, int portsCount, int maxBoxes, int
maxWeight) {

}
}
```

Python3:

```
class Solution:
def boxDelivering(self, boxes: List[List[int]], portsCount: int, maxBoxes:
int, maxWeight: int) -> int:
```

Python:

```
class Solution(object):
def boxDelivering(self, boxes, portsCount, maxBoxes, maxWeight):
"""
:type boxes: List[List[int]]
:type portsCount: int
:type maxBoxes: int
:type maxWeight: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} boxes
* @param {number} portsCount
* @param {number} maxBoxes
* @param {number} maxWeight
* @return {number}
*/
var boxDelivering = function(boxes, portsCount, maxBoxes, maxWeight) {

};
```

TypeScript:

```
function boxDelivering(boxes: number[][], portsCount: number, maxBoxes:
number, maxWeight: number): number {

};
```

C#:

```
public class Solution {
public int BoxDelivering(int[][] boxes, int portsCount, int maxBoxes, int
maxWeight) {

}
}
```

C:

```
int boxDelivering(int** boxes, int boxesSize, int* boxesColSize, int
portsCount, int maxBoxes, int maxWeight) {

}
```

Go:

```
func boxDelivering(boxes [][]int, portsCount int, maxBoxes int, maxWeight
int) int {

}
```

Kotlin:

```
class Solution {
fun boxDelivering(boxes: Array<IntArray>, portsCount: Int, maxBoxes: Int,
maxWeight: Int): Int {

}
}
```

Swift:

```
class Solution {
func boxDelivering(_ boxes: [[Int]], _ portsCount: Int, _ maxBoxes: Int, _
maxWeight: Int) -> Int {
}
}
```

Rust:

```
impl Solution {
pub fn box_delivering(boxes: Vec<Vec<i32>>, ports_count: i32, max_boxes: i32,
max_weight: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} boxes
# @param {Integer} ports_count
# @param {Integer} max_boxes
# @param {Integer} max_weight
# @return {Integer}
def box_delivering(boxes, ports_count, max_boxes, max_weight)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $boxes
* @param Integer $portsCount
* @param Integer $maxBoxes
* @param Integer $maxWeight
* @return Integer
*/
function boxDelivering($boxes, $portsCount, $maxBoxes, $maxWeight) {

}
}
```

Dart:

```
class Solution {
int boxDelivering(List<List<int>> boxes, int portsCount, int maxBoxes, int
maxWeight) {

}
}
```

Scala:

```
object Solution {
def boxDelivering(boxes: Array[Array[Int]], portsCount: Int, maxBoxes: Int,
maxWeight: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec box_delivering(boxes :: [[integer]], ports_count :: integer, max_boxes
:: integer, max_weight :: integer) :: integer
def box_delivering(boxes, ports_count, max_boxes, max_weight) do

end
end
```

Erlang:

```
-spec box_delivering(Boxes :: [[integer()]], PortsCount :: integer(),
MaxBoxes :: integer(), MaxWeight :: integer()) -> integer().
box_delivering(Boxes, PortsCount, MaxBoxes, MaxWeight) ->
.
```

Racket:

```
(define/contract (box-delivering boxes portsCount maxBoxes maxWeight)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?
exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Delivering Boxes from Storage to Ports
* Difficulty: Hard
* Tags: array, tree, dp, queue, heap
*
* 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 boxDelivering(vector<vector<int>>& boxes, int portsCount, int maxBoxes,
int maxWeight) {

}
};
```

Java Solution:

```
/**
* Problem: Delivering Boxes from Storage to Ports
* Difficulty: Hard
* Tags: array, tree, dp, queue, heap
*
* 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 boxDelivering(int[][] boxes, int portsCount, int maxBoxes, int
maxWeight) {

}
}
```

Python3 Solution:

```
"""
Problem: Delivering Boxes from Storage to Ports
```

```
Difficulty: Hard
Tags: array, tree, dp, queue, heap

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 boxDelivering(self, boxes: List[List[int]], portsCount: int, maxBoxes:
int, maxWeight: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def boxDelivering(self, boxes, portsCount, maxBoxes, maxWeight):
"""
:type boxes: List[List[int]]
:type portsCount: int
:type maxBoxes: int
:type maxWeight: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Delivering Boxes from Storage to Ports
* Difficulty: Hard
* Tags: array, tree, dp, queue, heap
*
* 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[][]} boxes
* @param {number} portsCount
* @param {number} maxBoxes
* @param {number} maxWeight
* @return {number}
*/
var boxDelivering = function(boxes, portsCount, maxBoxes, maxWeight) {

};
```

TypeScript Solution:

```
/**
* Problem: Delivering Boxes from Storage to Ports
* Difficulty: Hard
* Tags: array, tree, dp, queue, heap
*
* 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 boxDelivering(boxes: number[][], portsCount: number, maxBoxes:
number, maxWeight: number): number {

};
```

C# Solution:

```
/*
* Problem: Delivering Boxes from Storage to Ports
* Difficulty: Hard
* Tags: array, tree, dp, queue, heap
*
* 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 BoxDelivering(int[][] boxes, int portsCount, int maxBoxes, int
maxWeight) {
}
}
```

C Solution:

```c
/*
* Problem: Delivering Boxes from Storage to Ports
* Difficulty: Hard
* Tags: array, tree, dp, queue, heap
*
* 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 boxDelivering(int** boxes, int boxesSize, int* boxesColSize, int
portsCount, int maxBoxes, int maxWeight) {

}
```

Go Solution:

```go
// Problem: Delivering Boxes from Storage to Ports
// Difficulty: Hard
// Tags: array, tree, dp, queue, heap
//
// 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 boxDelivering(boxes [][]int, portsCount int, maxBoxes int, maxWeight
int) int {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun boxDelivering(boxes: Array<IntArray>, portsCount: Int, maxBoxes: Int,
maxWeight: Int): Int {
}
}
```

Swift Solution:

```
class Solution {
func boxDelivering(_ boxes: [[Int]], _ portsCount: Int, _ maxBoxes: Int, _
maxWeight: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Delivering Boxes from Storage to Ports
// Difficulty: Hard
// Tags: array, tree, dp, queue, heap
//
// 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 box_delivering(boxes: Vec<Vec<i32>>, ports_count: i32, max_boxes: i32,
max_weight: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} boxes
# @param {Integer} ports_count
# @param {Integer} max_boxes
# @param {Integer} max_weight
# @return {Integer}
def box_delivering(boxes, ports_count, max_boxes, max_weight)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $boxes
* @param Integer $portsCount
* @param Integer $maxBoxes
* @param Integer $maxWeight
* @return Integer
*/
function boxDelivering($boxes, $portsCount, $maxBoxes, $maxWeight) {

}
}
```

Dart Solution:

```
class Solution {
int boxDelivering(List<List<int>> boxes, int portsCount, int maxBoxes, int
maxWeight) {

}
}
```

Scala Solution:

```
object Solution {
def boxDelivering(boxes: Array[Array[Int]], portsCount: Int, maxBoxes: Int,
maxWeight: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec box_delivering(boxes :: [[integer]], ports_count :: integer, max_boxes
:: integer, max_weight :: integer) :: integer
def box_delivering(boxes, ports_count, max_boxes, max_weight) do

end
end
```

Erlang Solution:

```
-spec box_delivering(Boxes :: [[integer()]], PortsCount :: integer(),
MaxBoxes :: integer(), MaxWeight :: integer()) -> integer().
box_delivering(Boxes, PortsCount, MaxBoxes, MaxWeight) ->
.
```

Racket Solution:

```
(define/contract (box-delivering boxes portsCount maxBoxes maxWeight)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?
exact-integer? exact-integer?)
)
```