Problem 1710: Maximum Units on a Truck

Problem Information

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

Problem Description

You are assigned to put some amount of boxes onto

one truck

. You are given a 2D array

boxTypes

, where

boxTypes[i] = [numberOfBoxes

i

, numberOfUnitsPerBox

i

]

:

numberOfBoxes

i

is the number of boxes of type

i

.

numberOfUnitsPerBox

i

is the number of units in each box of the type

i

.

You are also given an integer

truckSize

, which is the

maximum

number of

boxes

that can be put on the truck. You can choose any boxes to put on the truck as long as the number of boxes does not exceed

truckSize

.

Return

the

maximum

total number of

units

that can be put on the truck.

Example 1:

Input:

boxTypes = [[1,3],[2,2],[3,1]], truckSize = 4

Output:

8

Explanation:

There are: - 1 box of the first type that contains 3 units. - 2 boxes of the second type that contain 2 units each. - 3 boxes of the third type that contain 1 unit each. You can take all the boxes of the first and second types, and one box of the third type. The total number of units will be = (1 * 3) + (2 * 2) + (1 * 1) = 8.

Example 2:

Input:

boxTypes = [[5,10],[2,5],[4,7],[3,9]], truckSize = 10

Output:

91

Constraints:

1 <= boxTypes.length <= 1000

1 <= numberOfBoxes

i

, numberOfUnitsPerBox

i

<= 1000

1 <= truckSize <= 10

6

Code Snippets

C++:

```
class Solution {
public:
int maximumUnits(vector<vector<int>>& boxTypes, int truckSize) {

}
};
```

Java:

```
class Solution {
public int maximumUnits(int[][] boxTypes, int truckSize) {

}
}
```

Python3:

```
class Solution:
def maximumUnits(self, boxTypes: List[List[int]], truckSize: int) -> int:
```

Python:

```
class Solution(object):
def maximumUnits(self, boxTypes, truckSize):
"""
:type boxTypes: List[List[int]]
:type truckSize: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} boxTypes
* @param {number} truckSize
* @return {number}
*/
var maximumUnits = function(boxTypes, truckSize) {

};
```

TypeScript:

```
function maximumUnits(boxTypes: number[][], truckSize: number): number {

};
```

C#:

```
public class Solution {
public int MaximumUnits(int[][] boxTypes, int truckSize) {

}
}
```

C:

```
int maximumUnits(int** boxTypes, int boxTypesSize, int* boxTypesColSize, int
truckSize) {

}
```

Go:

```
func maximumUnits(boxTypes [][]int, truckSize int) int {

}
```

Kotlin:

```
class Solution {
fun maximumUnits(boxTypes: Array<IntArray>, truckSize: Int): Int {

}
}
```

Swift:

```
class Solution {
func maximumUnits(_ boxTypes: [[Int]], _ truckSize: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn maximum_units(box_types: Vec<Vec<i32>>, truck_size: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} box_types
# @param {Integer} truck_size
# @return {Integer}
def maximum_units(box_types, truck_size)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $boxTypes
```

```
* @param Integer $truckSize
* @return Integer
*/
function maximumUnits($boxTypes, $truckSize) {

}
}
```

Dart:

```
class Solution {
int maximumUnits(List<List<int>> boxTypes, int truckSize) {

}
}
```

Scala:

```
object Solution {
def maximumUnits(boxTypes: Array[Array[Int]], truckSize: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec maximum_units(box_types :: [[integer]], truck_size :: integer) ::
integer
def maximum_units(box_types, truck_size) do

end
end
```

Erlang:

```
-spec maximum_units(BoxTypes :: [[integer()]], TruckSize :: integer()) ->
integer().
maximum_units(BoxTypes, TruckSize) ->
.
```

Racket:

```
(define/contract (maximum-units boxTypes truckSize)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Units on a Truck
* Difficulty: Easy
* Tags: array, greedy, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int maximumUnits(vector<vector<int>>& boxTypes, int truckSize) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Units on a Truck
* Difficulty: Easy
* Tags: array, greedy, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int maximumUnits(int[][] boxTypes, int truckSize) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Maximum Units on a Truck
Difficulty: Easy
Tags: array, greedy, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def maximumUnits(self, boxTypes: List[List[int]], truckSize: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maximumUnits(self, boxTypes, truckSize):
"""
:type boxTypes: List[List[int]]
:type truckSize: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Units on a Truck
* Difficulty: Easy
* Tags: array, greedy, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
/**
* @param {number[][]} boxTypes
* @param {number} truckSize
* @return {number}
*/
var maximumUnits = function(boxTypes, truckSize) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Units on a Truck
* Difficulty: Easy
* Tags: array, greedy, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function maximumUnits(boxTypes: number[][], truckSize: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Units on a Truck
* Difficulty: Easy
* Tags: array, greedy, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int MaximumUnits(int[][] boxTypes, int truckSize) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Maximum Units on a Truck
* Difficulty: Easy
* Tags: array, greedy, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int maximumUnits(int** boxTypes, int boxTypesSize, int* boxTypesColSize, int
truckSize) {

}
```

Go Solution:

```
// Problem: Maximum Units on a Truck
// Difficulty: Easy
// Tags: array, greedy, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func maximumUnits(boxTypes [][]int, truckSize int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maximumUnits(boxTypes: Array<IntArray>, truckSize: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func maximumUnits(_ boxTypes: [[Int]], _ truckSize: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Units on a Truck
// Difficulty: Easy
// Tags: array, greedy, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn maximum_units(box_types: Vec<Vec<i32>>, truck_size: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} box_types
# @param {Integer} truck_size
# @return {Integer}
def maximum_units(box_types, truck_size)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $boxTypes
* @param Integer $truckSize
* @return Integer
*/
```

```
function maximumUnits($boxTypes, $truckSize) {

}
}
```

Dart Solution:

```
class Solution {
int maximumUnits(List<List<int>> boxTypes, int truckSize) {

}
}
```

Scala Solution:

```
object Solution {
def maximumUnits(boxTypes: Array[Array[Int]], truckSize: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximum_units(box_types :: [[integer]], truck_size :: integer) ::
integer
def maximum_units(box_types, truck_size) do

end
end
```

Erlang Solution:

```
-spec maximum_units(BoxTypes :: [[integer()]], TruckSize :: integer()) ->
integer().
maximum_units(BoxTypes, TruckSize) ->
.
```

Racket Solution:

```
(define/contract (maximum-units boxTypes truckSize)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```