Problem 3560: Find Minimum Log Transportation Cost

Problem Information

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

Problem Description

You are given integers

n

,

m

, and

k

.

There are two logs of lengths

n

and

m

units, which need to be transported in three trucks where each truck can carry one log with length

at most

k

units.

You may cut the logs into smaller pieces, where the cost of cutting a log of length

x

into logs of length

len1

and

len2

is

cost = len1 * len2

such that

len1 + len2 = x

.

Return the

minimum total cost

to distribute the logs onto the trucks. If the logs don't need to be cut, the total cost is 0.

Example 1:

Input:

n = 6, m = 5, k = 5

Output:

5

Explanation:

Cut the log with length 6 into logs with length 1 and 5, at a cost equal to

1 * 5 == 5

. Now the three logs of length 1, 5, and 5 can fit in one truck each.

Example 2:

Input:

n = 4, m = 4, k = 6

Output:

0

Explanation:

The two logs can fit in the trucks already, hence we don't need to cut the logs.

Constraints:

2 <= k <= 10

5

1 <= n, m <= 2 * k

The input is generated such that it is always possible to transport the logs.

Code Snippets

C++:

```
class Solution {
public:
long long minCuttingCost(int n, int m, int k) {

}
};
```

Java:

```
class Solution {
public long minCuttingCost(int n, int m, int k) {

}
}
```

Python3:

```
class Solution:
def minCuttingCost(self, n: int, m: int, k: int) -> int:
```

Python:

```
class Solution(object):
def minCuttingCost(self, n, m, k):
"""
:type n: int
:type m: int
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number} m
* @param {number} k
* @return {number}
*/
var minCuttingCost = function(n, m, k) {
```

```
};
```

TypeScript:

```
function minCuttingCost(n: number, m: number, k: number): number {

};
```

C#:

```
public class Solution {
public long MinCuttingCost(int n, int m, int k) {

}
}
```

C:

```
long long minCuttingCost(int n, int m, int k) {

}
```

Go:

```
func minCuttingCost(n int, m int, k int) int64 {

}
```

Kotlin:

```
class Solution {
fun minCuttingCost(n: Int, m: Int, k: Int): Long {

}
}
```

Swift:

```
class Solution {
func minCuttingCost(_ n: Int, _ m: Int, _ k: Int) -> Int {

}
```

```
}
```

Rust:

```
impl Solution {
pub fn min_cutting_cost(n: i32, m: i32, k: i32) -> i64 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer} m
# @param {Integer} k
# @return {Integer}
def min_cutting_cost(n, m, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer $m
* @param Integer $k
* @return Integer
*/
function minCuttingCost($n, $m, $k) {

}
}
```

Dart:

```
class Solution {
int minCuttingCost(int n, int m, int k) {

}
}
```

Scala:

```
object Solution {
def minCuttingCost(n: Int, m: Int, k: Int): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_cutting_cost(n :: integer, m :: integer, k :: integer) :: integer
def min_cutting_cost(n, m, k) do

end
end
```

Erlang:

```
-spec min_cutting_cost(N :: integer(), M :: integer(), K :: integer()) ->
integer().
min_cutting_cost(N, M, K) ->
.
```

Racket:

```
(define/contract (min-cutting-cost n m k)
(-> exact-integer? exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find Minimum Log Transportation Cost
* Difficulty: Easy
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
```

```
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
long long minCuttingCost(int n, int m, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Find Minimum Log Transportation Cost
* Difficulty: Easy
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public long minCuttingCost(int n, int m, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Minimum Log Transportation Cost
Difficulty: Easy
Tags: math

Approach: Optimized algorithm based on problem constraints
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
```

```
def minCuttingCost(self, n: int, m: int, k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minCuttingCost(self, n, m, k):
"""
:type n: int
:type m: int
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Find Minimum Log Transportation Cost
* Difficulty: Easy
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} n
* @param {number} m
* @param {number} k
* @return {number}
*/
var minCuttingCost = function(n, m, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Find Minimum Log Transportation Cost
```

```
* Difficulty: Easy
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

function minCuttingCost(n: number, m: number, k: number): number {

};
```

C# Solution:

```
/*
* Problem: Find Minimum Log Transportation Cost
* Difficulty: Easy
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public long MinCuttingCost(int n, int m, int k) {

}
}
```

C Solution:

```
/*
* Problem: Find Minimum Log Transportation Cost
* Difficulty: Easy
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
long long minCuttingCost(int n, int m, int k) {

}
```

Go Solution:

```
// Problem: Find Minimum Log Transportation Cost
// Difficulty: Easy
// Tags: math
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

func minCuttingCost(n int, m int, k int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun minCuttingCost(n: Int, m: Int, k: Int): Long {

}
}
```

Swift Solution:

```
class Solution {
func minCuttingCost(_ n: Int, _ m: Int, _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Find Minimum Log Transportation Cost
// Difficulty: Easy
// Tags: math
//
```

```
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn min_cutting_cost(n: i32, m: i32, k: i32) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer} m
# @param {Integer} k
# @return {Integer}
def min_cutting_cost(n, m, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer $m
* @param Integer $k
* @return Integer
*/
function minCuttingCost($n, $m, $k) {

}
}
```

Dart Solution:

```
class Solution {
int minCuttingCost(int n, int m, int k) {

}
```

```
}
```

Scala Solution:

```
object Solution {
def minCuttingCost(n: Int, m: Int, k: Int): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_cutting_cost(n :: integer, m :: integer, k :: integer) :: integer
def min_cutting_cost(n, m, k) do

end
end
```

Erlang Solution:

```
-spec min_cutting_cost(N :: integer(), M :: integer(), K :: integer()) ->
integer().
min_cutting_cost(N, M, K) ->
.
```

Racket Solution:

```
(define/contract (min-cutting-cost n m k)
(-> exact-integer? exact-integer? exact-integer? exact-integer?)
)
```