Problem 1465: Maximum Area of a Piece of Cake After Horizontal and Vertical Cuts

Problem Information

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

Problem Description

You are given a rectangular cake of size

h x w

and two arrays of integers

horizontalCuts

and

verticalCuts

where:

horizontalCuts[i]

is the distance from the top of the rectangular cake to the

i

th

horizontal cut and similarly, and

verticalCuts[j]

is the distance from the left of the rectangular cake to the

j

th

vertical cut.

Return

the maximum area of a piece of cake after you cut at each horizontal and vertical position provided in the arrays

horizontalCuts

and

verticalCuts

. Since the answer can be a large number, return this

modulo

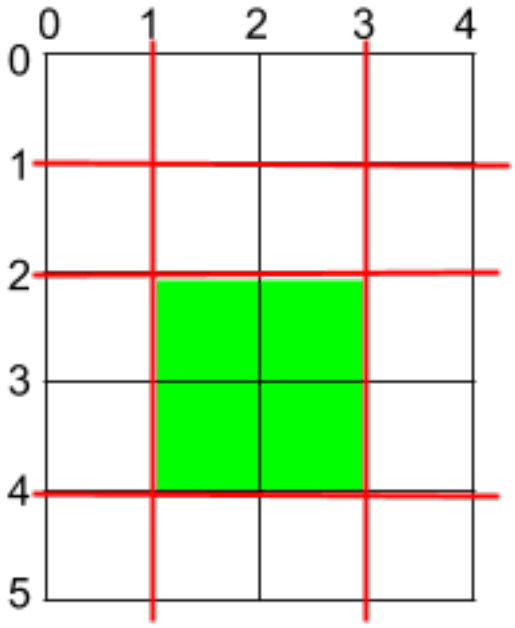
10

9

+ 7

.

Example 1:



Input:

h = 5, w = 4, horizontalCuts = [1,2,4], verticalCuts = [1,3]

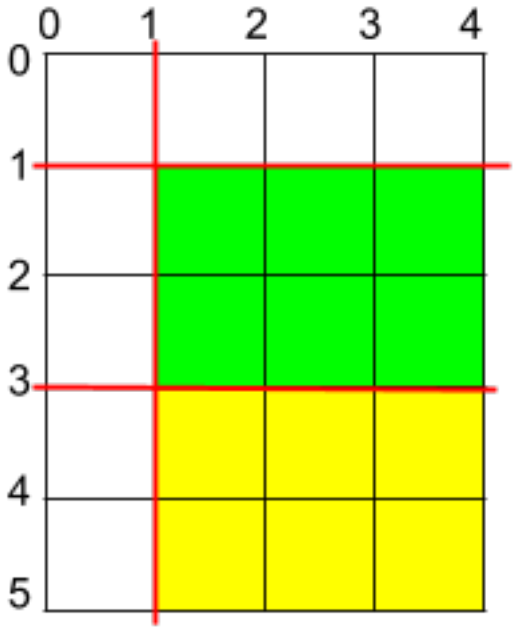
Output:

4

Explanation:

The figure above represents the given rectangular cake. Red lines are the horizontal and vertical cuts. After you cut the cake, the green piece of cake has the maximum area.

Example 2:



Input:

h = 5, w = 4, horizontalCuts = [3,1], verticalCuts = [1]

Output:

6

Explanation:

The figure above represents the given rectangular cake. Red lines are the horizontal and vertical cuts. After you cut the cake, the green and yellow pieces of cake have the maximum area.

Example 3:

Input:

h = 5, w = 4, horizontalCuts = [3], verticalCuts = [3]

Output:

9

Constraints:

2 <= h, w <= 10

9

1 <= horizontalCuts.length <= min(h - 1, 10

5

)

1 <= verticalCuts.length <= min(w - 1, 10

5

)

1 <= horizontalCuts[i] < h

1 <= verticalCuts[i] < w

All the elements in

horizontalCuts

are distinct.

All the elements in

verticalCuts

are distinct.

Code Snippets

C++:

```
class Solution {
public:
```

```
int maxArea(int h, int w, vector<int>& horizontalCuts, vector<int>&
verticalCuts) {

}
};
```

Java:

```
class Solution {
public int maxArea(int h, int w, int[] horizontalCuts, int[] verticalCuts) {

}
}
```

Python3:

```
class Solution:
def maxArea(self, h: int, w: int, horizontalCuts: List[int], verticalCuts:
List[int]) -> int:
```

Python:

```
class Solution(object):
def maxArea(self, h, w, horizontalCuts, verticalCuts):
"""
:type h: int
:type w: int
:type horizontalCuts: List[int]
:type verticalCuts: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} h
* @param {number} w
* @param {number[]} horizontalCuts
* @param {number[]} verticalCuts
* @return {number}
*/
var maxArea = function(h, w, horizontalCuts, verticalCuts) {
```

```
};
```

TypeScript:

```
function maxArea(h: number, w: number, horizontalCuts: number[],
verticalCuts: number[]): number {

};
```

C#:

```
public class Solution {
public int MaxArea(int h, int w, int[] horizontalCuts, int[] verticalCuts) {

}
}
```

C:

```
int maxArea(int h, int w, int* horizontalCuts, int horizontalCutsSize, int*
verticalCuts, int verticalCutsSize) {

}
```

Go:

```
func maxArea(h int, w int, horizontalCuts []int, verticalCuts []int) int {

}
```

Kotlin:

```
class Solution {
fun maxArea(h: Int, w: Int, horizontalCuts: IntArray, verticalCuts:
IntArray): Int {

}
}
```

Swift:

```
class Solution {
func maxArea(_ h: Int, _ w: Int, _ horizontalCuts: [Int], _ verticalCuts:
[Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_area(h: i32, w: i32, horizontal_cuts: Vec<i32>, vertical_cuts:
Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} h
# @param {Integer} w
# @param {Integer[]} horizontal_cuts
# @param {Integer[]} vertical_cuts
# @return {Integer}
def max_area(h, w, horizontal_cuts, vertical_cuts)

end
```

PHP:

```
class Solution {

/**
* @param Integer $h
* @param Integer $w
* @param Integer[] $horizontalCuts
* @param Integer[] $verticalCuts
* @return Integer
*/
function maxArea($h, $w, $horizontalCuts, $verticalCuts) {

}
}
```

Dart:

```
class Solution {
int maxArea(int h, int w, List<int> horizontalCuts, List<int> verticalCuts) {

}
}
```

Scala:

```
object Solution {
def maxArea(h: Int, w: Int, horizontalCuts: Array[Int], verticalCuts:
Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_area(h :: integer, w :: integer, horizontal_cuts :: [integer],
vertical_cuts :: [integer]) :: integer
def max_area(h, w, horizontal_cuts, vertical_cuts) do

end
end
```

Erlang:

```
-spec max_area(H :: integer(), W :: integer(), HorizontalCuts :: [integer()],
VerticalCuts :: [integer()]) -> integer().
max_area(H, W, HorizontalCuts, VerticalCuts) ->
.
```

Racket:

```
(define/contract (max-area h w horizontalCuts verticalCuts)
(-> exact-integer? exact-integer? (listof exact-integer?) (listof
exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Area of a Piece of Cake After Horizontal and Vertical Cuts
* Difficulty: Medium
* 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 maxArea(int h, int w, vector<int>& horizontalCuts, vector<int>&
verticalCuts) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Area of a Piece of Cake After Horizontal and Vertical Cuts
* Difficulty: Medium
* 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 maxArea(int h, int w, int[] horizontalCuts, int[] verticalCuts) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Area of a Piece of Cake After Horizontal and Vertical Cuts
Difficulty: Medium
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 maxArea(self, h: int, w: int, horizontalCuts: List[int], verticalCuts:
List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxArea(self, h, w, horizontalCuts, verticalCuts):
"""
:type h: int
:type w: int
:type horizontalCuts: List[int]
:type verticalCuts: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Area of a Piece of Cake After Horizontal and Vertical Cuts
* Difficulty: Medium
* 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} h
```

```
* @param {number} w
* @param {number[]} horizontalCuts
* @param {number[]} verticalCuts
* @return {number}
*/
var maxArea = function(h, w, horizontalCuts, verticalCuts) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Area of a Piece of Cake After Horizontal and Vertical Cuts
* Difficulty: Medium
* 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 maxArea(h: number, w: number, horizontalCuts: number[],
verticalCuts: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Area of a Piece of Cake After Horizontal and Vertical Cuts
* Difficulty: Medium
* 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 MaxArea(int h, int w, int[] horizontalCuts, int[] verticalCuts) {
```

```
}
}
```

C Solution:

```
/*
* Problem: Maximum Area of a Piece of Cake After Horizontal and Vertical Cuts
* Difficulty: Medium
* 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 maxArea(int h, int w, int* horizontalCuts, int horizontalCutsSize, int*
verticalCuts, int verticalCutsSize) {

}
```

Go Solution:

```
// Problem: Maximum Area of a Piece of Cake After Horizontal and Vertical
Cuts
// Difficulty: Medium
// 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 maxArea(h int, w int, horizontalCuts []int, verticalCuts []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxArea(h: Int, w: Int, horizontalCuts: IntArray, verticalCuts:
IntArray): Int {
```

```
}
}
```

Swift Solution:

```
class Solution {
func maxArea(_ h: Int, _ w: Int, _ horizontalCuts: [Int], _ verticalCuts:
[Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Area of a Piece of Cake After Horizontal and Vertical
Cuts
// Difficulty: Medium
// 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 max_area(h: i32, w: i32, horizontal_cuts: Vec<i32>, vertical_cuts:
Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} h
# @param {Integer} w
# @param {Integer[]} horizontal_cuts
# @param {Integer[]} vertical_cuts
# @return {Integer}
def max_area(h, w, horizontal_cuts, vertical_cuts)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $h
* @param Integer $w
* @param Integer[] $horizontalCuts
* @param Integer[] $verticalCuts
* @return Integer
*/
function maxArea($h, $w, $horizontalCuts, $verticalCuts) {

}
}
```

Dart Solution:

```
class Solution {
int maxArea(int h, int w, List<int> horizontalCuts, List<int> verticalCuts) {

}
}
```

Scala Solution:

```
object Solution {
def maxArea(h: Int, w: Int, horizontalCuts: Array[Int], verticalCuts:
Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_area(h :: integer, w :: integer, horizontal_cuts :: [integer],
vertical_cuts :: [integer]) :: integer
def max_area(h, w, horizontal_cuts, vertical_cuts) do

end
end
```

Erlang Solution:

```
-spec max_area(H :: integer(), W :: integer(), HorizontalCuts :: [integer()],
VerticalCuts :: [integer()]) -> integer().
max_area(H, W, HorizontalCuts, VerticalCuts) ->
.
```

Racket Solution:

```
(define/contract (max-area h w horizontalCuts verticalCuts)
(-> exact-integer? exact-integer? (listof exact-integer?) (listof
exact-integer?) exact-integer?)
)
```