Problem 1727: Largest Submatrix With Rearrangements

Problem Information

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

Problem Description

You are given a binary matrix

matrix

of size

m x n

, and you are allowed to rearrange the

columns

of the

matrix

in any order.

Return

the area of the largest submatrix within

matrix

where

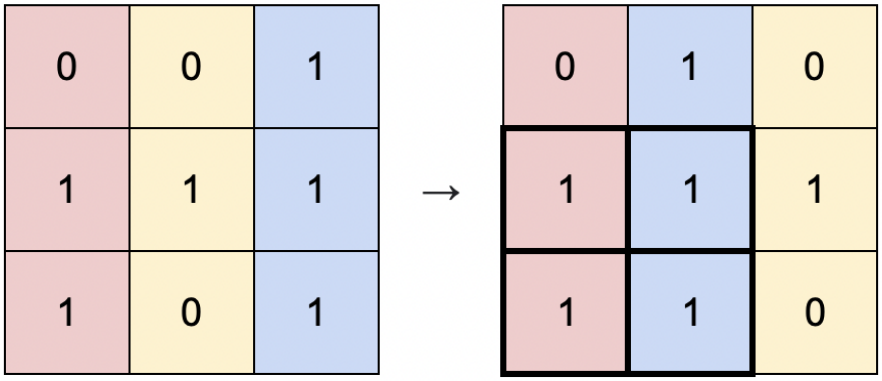
every

element of the submatrix is

1

after reordering the columns optimally.

Example 1:



Input:

matrix = [[0,0,1],[1,1,1],[1,0,1]]

Output:

4

Explanation:

You can rearrange the columns as shown above. The largest submatrix of 1s, in bold, has an area of 4.

Example 2:

Input:

matrix = [[1,0,1,0,1]]

Output:

3

Explanation:

You can rearrange the columns as shown above. The largest submatrix of 1s, in bold, has an area of 3.

Example 3:

Input:

matrix = [[1,1,0],[1,0,1]]

Output:

2

Explanation:

Notice that you must rearrange entire columns, and there is no way to make a submatrix of 1s larger than an area of 2.

Constraints:

m == matrix.length

n == matrix[i].length

1 <= m * n <= 10

5

matrix[i][j]

is either

0

or

1

.

Code Snippets

C++:

```
class Solution {
public:
int largestSubmatrix(vector<vector<int>>& matrix) {

}
};
```

Java:

```
class Solution {
public int largestSubmatrix(int[][] matrix) {

}
}
```

Python3:

```
class Solution:
def largestSubmatrix(self, matrix: List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def largestSubmatrix(self, matrix):
```

```
"""
:type matrix: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} matrix
* @return {number}
*/
var largestSubmatrix = function(matrix) {

};
```

TypeScript:

```
function largestSubmatrix(matrix: number[][]): number {

};
```

C#:

```
public class Solution {
public int LargestSubmatrix(int[][] matrix) {

}
}
```

C:

```
int largestSubmatrix(int** matrix, int matrixSize, int* matrixColSize) {

}
```

Go:

```
func largestSubmatrix(matrix [][]int) int {

}
```

Kotlin:

```
class Solution {
fun largestSubmatrix(matrix: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func largestSubmatrix(_ matrix: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn largest_submatrix(matrix: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} matrix
# @return {Integer}
def largest_submatrix(matrix)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $matrix
* @return Integer
*/
function largestSubmatrix($matrix) {

}
}
```

Dart:

```
class Solution {
int largestSubmatrix(List<List<int>> matrix) {

}
}
```

Scala:

```
object Solution {
def largestSubmatrix(matrix: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec largest_submatrix(matrix :: [[integer]]) :: integer
def largest_submatrix(matrix) do

end
end
```

Erlang:

```
-spec largest_submatrix(Matrix :: [[integer()]]) -> integer().
largest_submatrix(Matrix) ->
.
```

Racket:

```
(define/contract (largest-submatrix matrix)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Largest Submatrix With Rearrangements
* 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 largestSubmatrix(vector<vector<int>>& matrix) {

}
};
```

Java Solution:

```
/**
* Problem: Largest Submatrix With Rearrangements
* 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 largestSubmatrix(int[][] matrix) {

}
}
```

Python3 Solution:

```
"""
Problem: Largest Submatrix With Rearrangements
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 largestSubmatrix(self, matrix: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Largest Submatrix With Rearrangements
* 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[][]} matrix
* @return {number}
*/
var largestSubmatrix = function(matrix) {

};
```

TypeScript Solution:

```
/**
* Problem: Largest Submatrix With Rearrangements
* 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 largestSubmatrix(matrix: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Largest Submatrix With Rearrangements
* 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 LargestSubmatrix(int[][] matrix) {

}
}
```

C Solution:

```
/*
* Problem: Largest Submatrix With Rearrangements
* 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 largestSubmatrix(int** matrix, int matrixSize, int* matrixColSize) {

}
```

Go Solution:

```
// Problem: Largest Submatrix With Rearrangements
// 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 largestSubmatrix(matrix [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun largestSubmatrix(matrix: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func largestSubmatrix(_ matrix: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Largest Submatrix With Rearrangements
// 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 largest_submatrix(matrix: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} matrix
# @return {Integer}
def largest_submatrix(matrix)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $matrix
* @return Integer
*/
function largestSubmatrix($matrix) {

}
}
```

Dart Solution:

```
class Solution {
int largestSubmatrix(List<List<int>> matrix) {

}
}
```

Scala Solution:

```
object Solution {
def largestSubmatrix(matrix: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec largest_submatrix(matrix :: [[integer]]) :: integer
def largest_submatrix(matrix) do

end
end
```

Erlang Solution:

```
-spec largest_submatrix(Matrix :: [[integer()]]) -> integer().
largest_submatrix(Matrix) ->
.
```

Racket Solution:

```
(define/contract (largest-submatrix matrix)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```