Problem 566: Reshape the Matrix

Problem Information

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

Problem Description

In MATLAB, there is a handy function called

reshape

which can reshape an

m x n

matrix into a new one with a different size

r x c

keeping its original data.

You are given an

m x n

matrix

mat

and two integers

r

and

c

representing the number of rows and the number of columns of the wanted reshaped matrix.

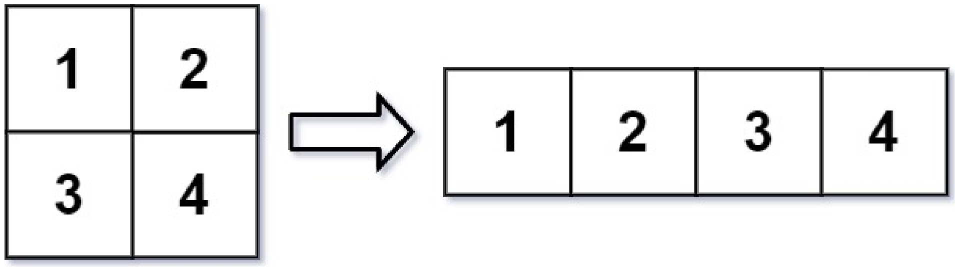
The reshaped matrix should be filled with all the elements of the original matrix in the same row-traversing order as they were.

If the

reshape

operation with given parameters is possible and legal, output the new reshaped matrix; Otherwise, output the original matrix.

Example 1:



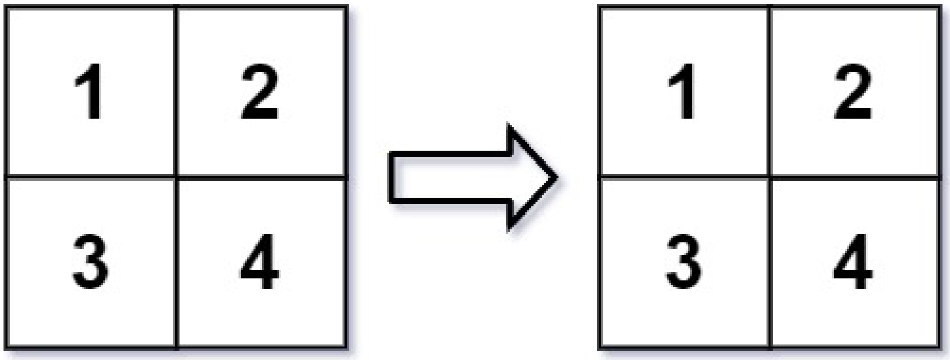
Input:

mat = [[1,2],[3,4]], r = 1, c = 4

Output:

[[1,2,3,4]]

Example 2:



Input:

mat = [[1,2],[3,4]], r = 2, c = 4

Output:

[[1,2],[3,4]]

Constraints:

m == mat.length

n == mat[i].length

1 <= m, n <= 100

-1000 <= mat[i][j] <= 1000

1 <= r, c <= 300

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> matrixReshape(vector<vector<int>>& mat, int r, int c) {

}
};
```

Java:

```
class Solution {
public int[][] matrixReshape(int[][] mat, int r, int c) {

}
}
```

Python3:

```
class Solution:
def matrixReshape(self, mat: List[List[int]], r: int, c: int) ->
List[List[int]]:
```

Python:

```
class Solution(object):
def matrixReshape(self, mat, r, c):
"""
:type mat: List[List[int]]
:type r: int
:type c: int
:rtype: List[List[int]]
"""
```

JavaScript:

```
/**
* @param {number[][]} mat
* @param {number} r
* @param {number} c
* @return {number[][]}
*/
var matrixReshape = function(mat, r, c) {

};
```

TypeScript:

```
function matrixReshape(mat: number[][], r: number, c: number): number[][] {

};
```

C#:

```
public class Solution {
public int[][] MatrixReshape(int[][] mat, int r, int c) {

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** matrixReshape(int** mat, int matSize, int* matColSize, int r, int c,
int* returnSize, int** returnColumnSizes) {

}
```

Go:

```
func matrixReshape(mat [][]int, r int, c int) [][]int {

}
```

Kotlin:

```
class Solution {
fun matrixReshape(mat: Array<IntArray>, r: Int, c: Int): Array<IntArray> {

}
}
```

Swift:

```
class Solution {
func matrixReshape(_ mat: [[Int]], _ r: Int, _ c: Int) -> [[Int]] {

}
}
```

Rust:

```
impl Solution {
pub fn matrix_reshape(mat: Vec<Vec<i32>>, r: i32, c: i32) -> Vec<Vec<i32>> {

}
}
```

Ruby:

```
# @param {Integer[][]} mat
# @param {Integer} r
# @param {Integer} c
# @return {Integer[][]}
def matrix_reshape(mat, r, c)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $mat
* @param Integer $r
* @param Integer $c
* @return Integer[][]
*/
function matrixReshape($mat, $r, $c) {

}
}
```

Dart:

```
class Solution {
List<List<int>> matrixReshape(List<List<int>> mat, int r, int c) {

}
}
```

Scala:

```
object Solution {
def matrixReshape(mat: Array[Array[Int]], r: Int, c: Int): Array[Array[Int]]
= {

}
}
```

Elixir:

```
defmodule Solution do
@spec matrix_reshape(mat :: [[integer]], r :: integer, c :: integer) ::
[[integer]]
def matrix_reshape(mat, r, c) do

end
end
```

Erlang:

```
-spec matrix_reshape(Mat :: [[integer()]], R :: integer(), C :: integer()) ->
[[integer()]].
matrix_reshape(Mat, R, C) ->
.
```

Racket:

```
(define/contract (matrix-reshape mat r c)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer? (listof
(listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Reshape the Matrix
* Difficulty: Easy
* Tags: array
*
* 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:
vector<vector<int>> matrixReshape(vector<vector<int>>& mat, int r, int c) {

}
};
```

Java Solution:

```
/**
* Problem: Reshape the Matrix
* Difficulty: Easy
* Tags: array
*
* 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[][] matrixReshape(int[][] mat, int r, int c) {

}
}
```

Python3 Solution:

```
"""
Problem: Reshape the Matrix
Difficulty: Easy
Tags: array

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 matrixReshape(self, mat: List[List[int]], r: int, c: int) ->
List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def matrixReshape(self, mat, r, c):
"""
:type mat: List[List[int]]
:type r: int
:type c: int
:rtype: List[List[int]]
"""
```

JavaScript Solution:

```
/**
* Problem: Reshape the Matrix
* Difficulty: Easy
* Tags: array
*
* 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[][]} mat
* @param {number} r
* @param {number} c
* @return {number[][]}
*/
var matrixReshape = function(mat, r, c) {

};
```

TypeScript Solution:

```
/**
* Problem: Reshape the Matrix
* Difficulty: Easy
* Tags: array
*
* 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 matrixReshape(mat: number[][], r: number, c: number): number[][] {

};
```

C# Solution:

```
/*
* Problem: Reshape the Matrix
* Difficulty: Easy
* Tags: array
*
* 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[][] MatrixReshape(int[][] mat, int r, int c) {

}
}
```

C Solution:

```
/*
* Problem: Reshape the Matrix
* Difficulty: Easy
* Tags: array
*
* 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
```

```
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** matrixReshape(int** mat, int matSize, int* matColSize, int r, int c,
int* returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Reshape the Matrix
// Difficulty: Easy
// Tags: array
//
// 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 matrixReshape(mat [][]int, r int, c int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun matrixReshape(mat: Array<IntArray>, r: Int, c: Int): Array<IntArray> {

}
}
```

Swift Solution:

```
class Solution {
func matrixReshape(_ mat: [[Int]], _ r: Int, _ c: Int) -> [[Int]] {

}

}
```

Rust Solution:

```
// Problem: Reshape the Matrix
// Difficulty: Easy
// Tags: array
//
// 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 matrix_reshape(mat: Vec<Vec<i32>>, r: i32, c: i32) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} mat
# @param {Integer} r
# @param {Integer} c
# @return {Integer[][]}
def matrix_reshape(mat, r, c)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $mat
* @param Integer $r
* @param Integer $c
* @return Integer[][]
*/
function matrixReshape($mat, $r, $c) {

}


}
```

Dart Solution:

```
class Solution {
List<List<int>> matrixReshape(List<List<int>> mat, int r, int c) {

}
}
```

Scala Solution:

```
object Solution {
def matrixReshape(mat: Array[Array[Int]], r: Int, c: Int): Array[Array[Int]]
= {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec matrix_reshape(mat :: [[integer]], r :: integer, c :: integer) ::
[[integer]]
def matrix_reshape(mat, r, c) do

end
end
```

Erlang Solution:

```
-spec matrix_reshape(Mat :: [[integer()]], R :: integer(), C :: integer()) ->
[[integer()]].
matrix_reshape(Mat, R, C) ->
.
```

Racket Solution:

```
(define/contract (matrix-reshape mat r c)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer? (listof
(listof exact-integer?)))
)
```