Articles → 566. Reshape the Matrix ▼

Copy

566. Reshape the Matrix **

April 29, 2017 | 32.9K views

*** Average Rating: 4.85 (27 votes)

In MATLAB, there is a very useful function called 'reshape', which can reshape a matrix into a new one with different size but keep its original data.

You're given a matrix represented by a two-dimensional array, and two **positive** integers **r** and **c** representing the row number and column number of the wanted reshaped matrix, respectively.

The reshaped matrix need to 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:
 nums =
 [[1,2],
  [3,4]]
 r = 1, c = 4
 Output:
 [[1,2,3,4]]
 Explanation:
 The row-traversing of nums is [1,2,3,4]. The new reshaped matrix is a 1 * 4 matrix, fi
Example 2:
```

Input:

```
nums =
[[1,2],
[3,4]]
r = 2, c = 4
Output:
[[1,2],
[3,4]]
Explanation:
There is no way to reshape a 2 * 2 matrix to a 2 * 4 matrix. So output the original matrix
```

1. The height and width of the given matrix is in range [1, 100].

Note:

2. The given r and c are all positive.

Solution

Algorithm

Approach 1: Using Queue

The simplest method is to extract all the elements of the given matrix by reading the elements in a row-wise

Java

fashion. In this implementation, we use a queue to put the extracted elements. Then, we can take out the elements of the queue formed in a serial order and arrange the elements in the resultant required matrix in a row-by-row order again. The formation of the resultant matrix won't be possible if the number of elements in the original matrix isn't equal to the number of elements in the resultant matrix.

1 class Solution { public int[][] matrixReshape(int[][] nums, int r, int c) {

```
int[][] res = new int[r][c];
            if (nums.length == 0 || r * c != nums.length * nums[0].length)
  5
                return nums;
             Queue<Integer> queue = new LinkedList();
             for (int i = 0; i < nums.length; i++) {
                for (int j = 0; j < nums[0].length; j++) {
                    queue.add(nums[i][j]);
  9
  10
 11
  12
             for (int i = 0; i < r; i++) {
  13
                for (int j = 0; j < c; j++) {
  14
                    res[i][j] = queue.remove();
 15
  16
            }
 17
             return res;
  18
 19 }
Complexity Analysis

    Time complexity: O(m · n). We traverse over m · n elements twice. Here, m and n refer to the
```

Space complexity: O(m · n). The queue formed will be of size m · n.

- Approach 2: Without Using Extra Space

number of rows and columns of the given matrix respectively.

Instead of unnecessarily using the queue as in the brute force approach, we can keep putting the numbers in the resultant matrix directly while iterating over the given matrix in a row-by-row order. While putting the

numbers in the resultant array, we fix a particular row and keep on incrementing the column numbers only till we reach the end of the required columns indicated by c. At this moment, we update the row index by

Algorithm

incrementing it and reset the column index to start from 0 again. Thus, we can save the space consumed by the queue for storing the data that just needs to be copied into a new array. Copy Java 1 public class Solution { public int[][] matrixReshape(int[][] nums, int r, int c) { int[][] res = new int[r][c]; if (nums.length == 0 || r * c != nums.length * nums[0].length)

```
return nums;
            int rows = 0, cols = 0;
            for (int i = 0; i < nums.length; i++) {
                for (int j = 0; j < nums[0].length; j++) {
                    res[rows][cols] = nums[i][j];
                   if (cols == c) {
  11
  12
                        rows++;
 13
                        cols = 0;
  15
                }
  16
 17
             return res;
Complexity Analysis
  • Time complexity : O(m \cdot n). We traverse the entire matrix of size m \cdot n once only. Here, m and n
     refers to the number of rows and columns in the given matrix.
```

• Space complexity : $O(m \cdot n)$. The resultant matrix of size $m \cdot n$ is used.

maths to help ease the situation.

Approach 3: Using division and modulus

checking the current indices every time. Instead of doing these limit checks at every step, we can make use of

memory(which is 1-D in nature)? It is internally represented as a 1-D array only. The element nums[i][j] of

The idea behind this approach is as follows. Do you know how a 2-D array is stored in the main

nums array is represented in the form of a one dimensional array by using the index in the form: nums[n*i+j], where m is the number of columns in the given matrix. Looking at the same in the reverse order, while putting the elements in the elements in the resultant matrix, we can make use of a count variable which gets incremented for every element traversed as if we are putting the elements in a 1-

Algorithm In the last approach, we needed to keep a track of when we reached the end of columns for the resultant matrix and needed to update the current row and column number for putting the extracted elements by

for (int j = 0; j < nums[0].length; j++) {

D resultant array. But, to convert the count back into 2-D matrix indices with a column count of c, we can obtain the indices as res[count/c][count%c] where count/c is the row number and count%c is the coloumn number. Thus, we can save the extra checking required at each step. Copy Java 1 public class Solution { public int[][] matrixReshape(int[][] nums, int r, int c) { int[][] res = new int[r][c]; if (nums.length == 0 || r * c != nums.length * nums[0].length) return nums; int count = 0; for (int i = 0; i < nums.length; i++) {

res[count / c][count % c] = nums[i][j]; 10 count++; 11 12 13 return res; 14 15 } **Complexity Analysis** • Time complexity : $O(m \cdot n)$. We traverse the entire matrix of size $m \cdot n$ once only. Here, m and nrefers to the number of rows and columns in the given matrix. • Space complexity : $O(m \cdot n)$. The resultant matrix of size $m \cdot n$ is used. Rate this article: * * * * *

O Previous

8

Comments: 24 Sort By ▼

Next

Post

Type comment here... (Markdown is supported) Preview

What's the purpose of int count = 0; in the first approach?

a-b-c ★ 692 ② April 1, 2018 4:13 PM

17 A V E Share A Reply

It never been used.

Python Onliner

class Solution:

def matrixReshape(self, nums, r, c):

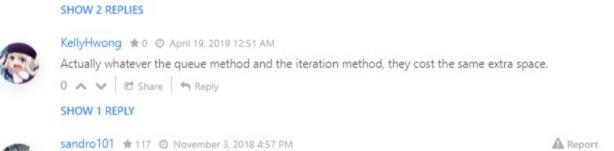
```
SHOW 1 REPLY
A Report
Python Code:
A = [x for row in nums for x in row ]
       if r * c == len(A):
          return [A[i*c : (i + 1)*c] for i in range(r) ]
11 ∧ ∨ Et Share ► Reply
sjw214 * 183 O November 29, 2018 7:46 PM
JavaScript Solution:
var matrixReshape = function(nums, r, c) {
  const currentRows = nums.length;
   const currentCols = nums[0].length
                                   Read More
1 A V & Share A Reply
inctrl # 77 @ June 5, 2018 1:44 PM
```



return f f numsf(v*c+x)//len(numsf01)1f(v*c+x)%len(numsf01)1 for x in ran

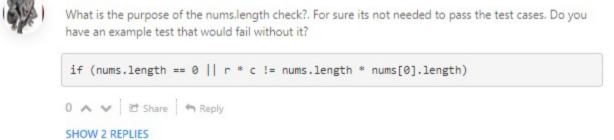
Read More

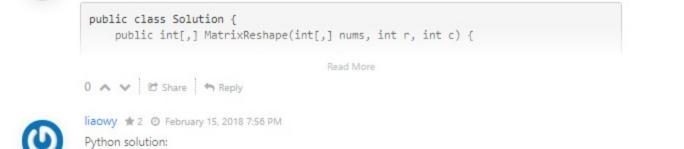




return f f numsf(v*c+x)//len(numsf01)1f(v*c+x)%len(numsf01)1 for x in ran

Read More





0 ∧ ∨ Ø Share ♠ Reply

def matrixReshape(self, nums, r, c):

class Solution:

(123)

ericksoa # 41 @ October 7, 2018 5:36 AM Hacky C# Approach using an enumerator