

Problem 867: Transpose Matrix

Problem Information

Difficulty: Easy

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Given a 2D integer array

matrix

, return

the

transpose

of

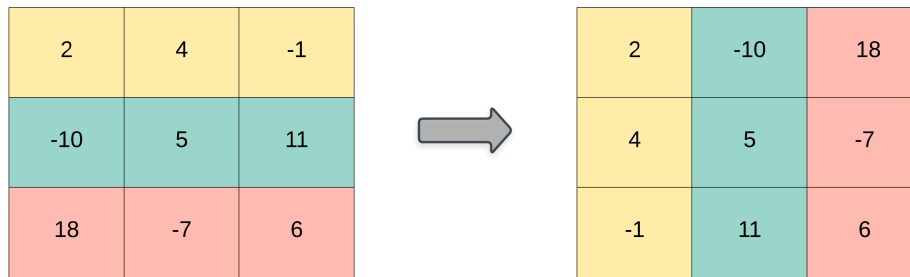
matrix

.

The

transpose

of a matrix is the matrix flipped over its main diagonal, switching the matrix's row and column indices.



Example 1:

Input:

```
matrix = [[1,2,3],[4,5,6],[7,8,9]]
```

Output:

```
[[1,4,7],[2,5,8],[3,6,9]]
```

Example 2:

Input:

```
matrix = [[1,2,3],[4,5,6]]
```

Output:

```
[[1,4],[2,5],[3,6]]
```

Constraints:

```
m == matrix.length
```

```
n == matrix[i].length
```

```
1 <= m, n <= 1000
```

```
1 <= m * n <= 10
```

-10

9

`<= matrix[i][j] <= 10`

9

Code Snippets

C++:

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

    }
};
```

Java:

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

    }
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

    }
}
```

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** transpose(int** matrix, int matrixSize, int* matrixColSize, int*
returnSize, int** returnColumnSizes) {

}
```

Go:

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

}
```

Kotlin:

```
class Solution {  
    fun transpose(matrix: Array<IntArray>): Array<IntArray> {  
  
    }  
}
```

Swift:

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

Rust:

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

Ruby:

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

PHP:

```
class Solution {  
  
    /**  
     * @param Integer[][] $matrix  
     * @return Integer[][]  
     */  
    function transpose($matrix) {  
  
    }  
}
```

```
}
```

Dart:

```
class Solution {  
  List<List<int>> transpose(List<List<int>> matrix) {  
  
  }  
}
```

Scala:

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

Elixir:

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

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
 * Problem: Transpose 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>> transpose(vector<vector<int>>& matrix) {

    }
};
```

Java Solution:

```
/**
 * Problem: Transpose 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[][] transpose(int[][] matrix) {

    }
}
```

Python3 Solution:

```
"""
Problem: Transpose 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 transpose(self, matrix: List[List[int]]) -> List[List[int]]:
        # TODO: Implement optimized solution
        pass

```

Python Solution:

```

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

```

JavaScript Solution:

```

/**
 * Problem: Transpose 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[][]} matrix
 * @return {number[][]}
 */
var transpose = function(matrix) {

```

TypeScript Solution:

```

/**
 * Problem: Transpose 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 transpose(matrix: number[][]): number[][] {

};

```

C# Solution:

```

/*
 * Problem: Transpose 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[][] Transpose(int[][] matrix) {

    }
}

```

C Solution:

```

/*
 * Problem: Transpose 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** transpose(int** matrix, int matrixSize, int* matrixColSize, int*
returnSize, int** returnColumnSizes) {

}

```

Go Solution:

```

// Problem: Transpose 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 transpose(matrix [][]int) [][]int {

}

```

Kotlin Solution:

```

class Solution {
    fun transpose(matrix: Array<IntArray>): Array<IntArray> {

    }
}

```

Swift Solution:

```

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

    }
}

```

```
}
```

Rust Solution:

```
// Problem: Transpose 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 transpose(matrix: Vec<Vec<i32>>) -> Vec<Vec<i32>> {

    }
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

    }

}
```

Dart Solution:

```

class Solution {
  List<List<int>> transpose(List<List<int>> matrix) {

  }
}

```

Scala Solution:

```

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

  }
}

```

Elixir Solution:

```

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

  end
end

```

Erlang Solution:

```

-spec transpose(Matrix :: [[integer()]]) -> [[integer()]].
transpose(Matrix) ->
.

```

Racket Solution:

```

(define/contract (transpose matrix)
  (-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)

```