Problem 1861: Rotating the Box

Problem Information

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

Problem Description

You are given an

m x n

matrix of characters

boxGrid

representing a side-view of a box. Each cell of the box is one of the following:

A stone

'#'

A stationary obstacle

'*'

Empty

'.'

The box is rotated

90 degrees clockwise

, causing some of the stones to fall due to gravity. Each stone falls down until it lands on an obstacle, another stone, or the bottom of the box. Gravity

does not

affect the obstacles' positions, and the inertia from the box's rotation

does not

affect the stones' horizontal positions.

It is

guaranteed

that each stone in

boxGrid

rests on an obstacle, another stone, or the bottom of the box.

Return

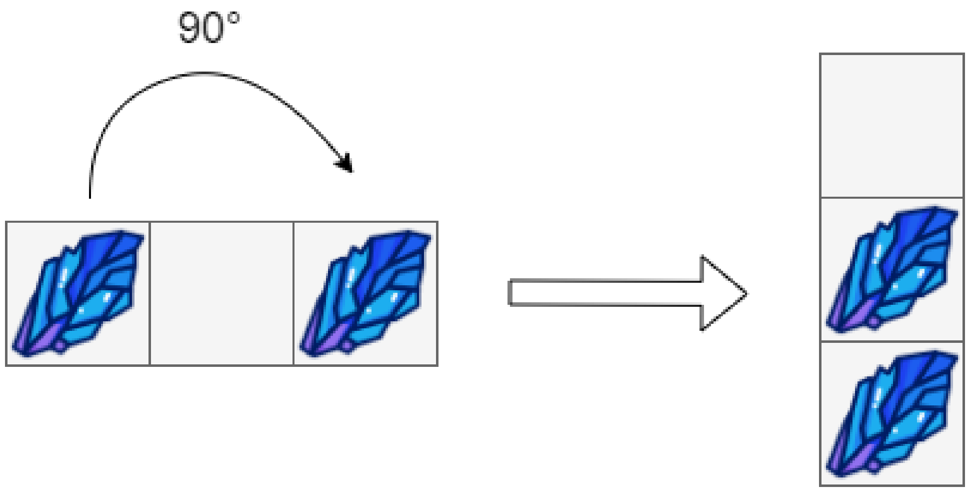
an

n x m

matrix representing the box after the rotation described above

.

Example 1:



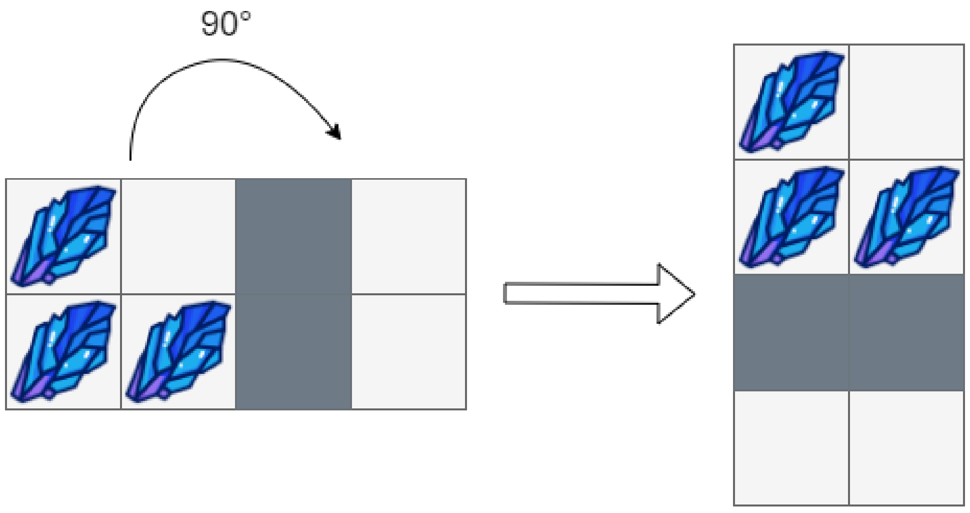
Input:

boxGrid = [["#",".","#"]]

Output:

[["."],   ["#"],   ["#"]]

Example 2:



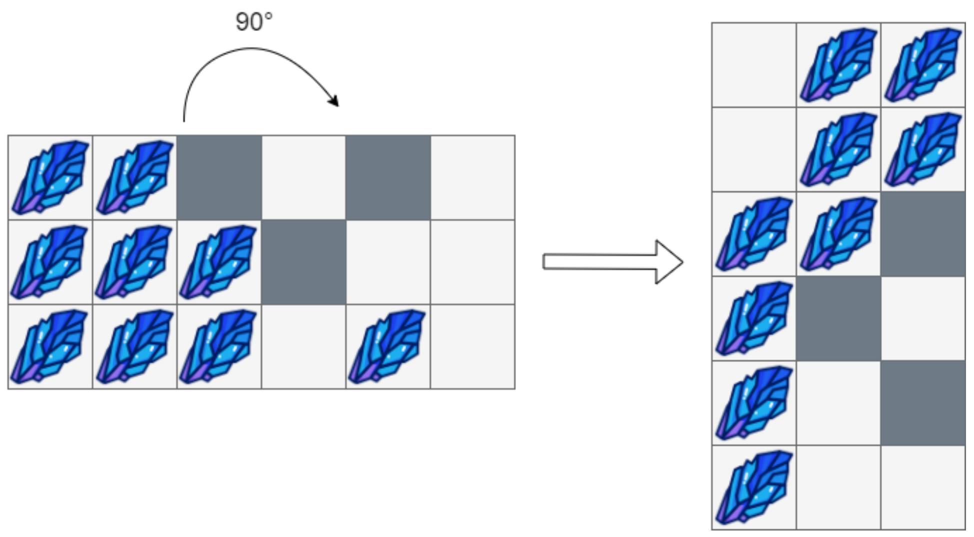
Input:

boxGrid = [["#",".","*","."],   ["#","#","*","."]]

Output:

[["#","."],   ["#","#"],   ["*","*"],   [".","."]]

Example 3:



Input:

boxGrid = [["#","#","*",".","*","."],   ["#","#","#","*",".","."],   ["#","#","#",".","#","."]]

Output:

[[".","#","#"],   [".","#","#"],   ["#","#","*"],   ["#","*","."],   ["#",".","*"],   ["#",".","."]]

Constraints:

m == boxGrid.length

n == boxGrid[i].length

1 <= m, n <= 500

boxGrid[i][j]

is either

'#'

,

'*'

, or

'.'

.

Code Snippets

C++:

```
class Solution {
public:
vector<vector<char>> rotateTheBox(vector<vector<char>>& boxGrid) {

}
};
```

Java:

```
class Solution {
public char[][] rotateTheBox(char[][] boxGrid) {

}
}
```

Python3:

```
class Solution:
def rotateTheBox(self, boxGrid: List[List[str]]) -> List[List[str]]:
```

Python:

```
class Solution(object):
def rotateTheBox(self, boxGrid):
"""
:type boxGrid: List[List[str]]
:rtype: List[List[str]]
"""
```

JavaScript:

```
/**
* @param {character[][]} boxGrid
* @return {character[][]}
*/
var rotateTheBox = function(boxGrid) {

};
```

TypeScript:

```
function rotateTheBox(boxGrid: string[][]): string[][] {

};
```

C#:

```
public class Solution {
public char[][] RotateTheBox(char[][] boxGrid) {

}
}
```

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().
*/
char** rotateTheBox(char** boxGrid, int boxGridSize, int* boxGridColSize,
int* returnSize, int** returnColumnSizes) {

}
```

Go:

```
func rotateTheBox(boxGrid [][]byte) [][]byte {

}
```

Kotlin:

```
class Solution {
fun rotateTheBox(boxGrid: Array<CharArray>): Array<CharArray> {

}
}
```

Swift:

```
class Solution {
func rotateTheBox(_ boxGrid: [[Character]]) -> [[Character]] {

}
}
```

Rust:

```
impl Solution {
pub fn rotate_the_box(box_grid: Vec<Vec<char>>) -> Vec<Vec<char>> {

}
}
```

Ruby:

```
# @param {Character[][]} box_grid
# @return {Character[][]}
def rotate_the_box(box_grid)

end
```

PHP:

```
class Solution {

/**
* @param String[][] $boxGrid
* @return String[][]
*/
function rotateTheBox($boxGrid) {

}
}
```

Dart:

```
class Solution {
List<List<String>> rotateTheBox(List<List<String>> boxGrid) {

}
}
```

Scala:

```
object Solution {
def rotateTheBox(boxGrid: Array[Array[Char]]): Array[Array[Char]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec rotate_the_box(box_grid :: [[char]]) :: [[char]]
def rotate_the_box(box_grid) do

end
end
```

Erlang:

```
-spec rotate_the_box(BoxGrid :: [[char()]]) -> [[char()]].
rotate_the_box(BoxGrid) ->
.
```

Racket:

```
(define/contract (rotate-the-box boxGrid)
(-> (listof (listof char?)) (listof (listof char?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Rotating the Box
* Difficulty: Medium
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
vector<vector<char>> rotateTheBox(vector<vector<char>>& boxGrid) {

}
};
```

Java Solution:

```
/**
* Problem: Rotating the Box
* Difficulty: Medium
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public char[][] rotateTheBox(char[][] boxGrid) {

}
}
```

Python3 Solution:

```
"""
Problem: Rotating the Box
Difficulty: Medium
Tags: array, tree
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def rotateTheBox(self, boxGrid: List[List[str]]) -> List[List[str]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def rotateTheBox(self, boxGrid):
"""
:type boxGrid: List[List[str]]
:rtype: List[List[str]]
"""
```

JavaScript Solution:

```
/**
* Problem: Rotating the Box
* Difficulty: Medium
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {character[][]} boxGrid
* @return {character[][]}
*/
var rotateTheBox = function(boxGrid) {

};
```

TypeScript Solution:

```
/**
* Problem: Rotating the Box
* Difficulty: Medium
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function rotateTheBox(boxGrid: string[][]): string[][] {

};
```

C# Solution:

```
/*
* Problem: Rotating the Box
* Difficulty: Medium
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public char[][] RotateTheBox(char[][] boxGrid) {

}
}
```

C Solution:

```
/*
* Problem: Rotating the Box
* Difficulty: Medium
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
```

```
*/

/**
* 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().
*/
char** rotateTheBox(char** boxGrid, int boxGridSize, int* boxGridColSize,
int* returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Rotating the Box
// Difficulty: Medium
// Tags: array, tree
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func rotateTheBox(boxGrid [][]byte) [][]byte {

}
```

Kotlin Solution:

```
class Solution {
fun rotateTheBox(boxGrid: Array<CharArray>): Array<CharArray> {

}
}
```

Swift Solution:

```
class Solution {
func rotateTheBox(_ boxGrid: [[Character]]) -> [[Character]] {

}
}
```

Rust Solution:

```
// Problem: Rotating the Box
// Difficulty: Medium
// Tags: array, tree
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn rotate_the_box(box_grid: Vec<Vec<char>>) -> Vec<Vec<char>> {

}
}
```

Ruby Solution:

```
# @param {Character[][]} box_grid
# @return {Character[][]}
def rotate_the_box(box_grid)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[][] $boxGrid
* @return String[][]
*/
function rotateTheBox($boxGrid) {

}
}
```

Dart Solution:

```
class Solution {
List<List<String>> rotateTheBox(List<List<String>> boxGrid) {

}
}
```

Scala Solution:

```
object Solution {
def rotateTheBox(boxGrid: Array[Array[Char]]): Array[Array[Char]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec rotate_the_box(box_grid :: [[char]]) :: [[char]]
def rotate_the_box(box_grid) do

end
end
```

Erlang Solution:

```
-spec rotate_the_box(BoxGrid :: [[char()]]) -> [[char()]].
rotate_the_box(BoxGrid) ->
.
```

Racket Solution:

```
(define/contract (rotate-the-box boxGrid)
(-> (listof (listof char?)) (listof (listof char?)))
)
```