Problem 832: Flipping an Image

Problem Information

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

Problem Description

Given an

n x n

binary matrix

image

, flip the image

horizontally

, then invert it, and return

the resulting image

.

To flip an image horizontally means that each row of the image is reversed.

For example, flipping

[1,1,0]

horizontally results in

[0,1,1]

.

To invert an image means that each

0

is replaced by

1

, and each

1

is replaced by

0

.

For example, inverting

[0,1,1]

results in

[1,0,0]

.

Example 1:

Input:

image = [[1,1,0],[1,0,1],[0,0,0]]

Output:

[[1,0,0],[0,1,0],[1,1,1]]

Explanation:

First reverse each row: [[0,1,1],[1,0,1],[0,0,0]]. Then, invert the image: [[1,0,0],[0,1,0],[1,1,1]]

Example 2:

Input:

image = [[1,1,0,0],[1,0,0,1],[0,1,1,1],[1,0,1,0]]

Output:

[[1,1,0,0],[0,1,1,0],[0,0,0,1],[1,0,1,0]]

Explanation:

First reverse each row: [[0,0,1,1],[1,0,0,1],[1,1,1,0],[0,1,0,1]]. Then invert the image: [[1,1,0,0],[0,1,1,0],[0,0,0,1],[1,0,1,0]]

Constraints:

n == image.length

n == image[i].length

1 <= n <= 20

images[i][j]

is either

0

or

1

.

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> flipAndInvertImage(vector<vector<int>>& image) {

}
};
```

Java:

```
class Solution {
public int[][] flipAndInvertImage(int[][] image) {

}
}
```

Python3:

```
class Solution:
def flipAndInvertImage(self, image: List[List[int]]) -> List[List[int]]:
```

Python:

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

JavaScript:

```
/**
* @param {number[][]} image
* @return {number[][]}
*/
```

```
var flipAndInvertImage = function(image) {

};
```

TypeScript:

```
function flipAndInvertImage(image: number[][]): number[][] {

};
```

C#:

```
public class Solution {
public int[][] FlipAndInvertImage(int[][] image) {

}
}
```

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** flipAndInvertImage(int** image, int imageSize, int* imageColSize, int*
returnSize, int** returnColumnSizes) {

}
```

Go:

```
func flipAndInvertImage(image [][]int) [][]int {

}
```

Kotlin:

```
class Solution {
fun flipAndInvertImage(image: Array<IntArray>): Array<IntArray> {
```

```
}
}
```

Swift:

```
class Solution {
func flipAndInvertImage(_ image: [[Int]]) -> [[Int]] {

}
}
```

Rust:

```
impl Solution {
pub fn flip_and_invert_image(image: Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
}
```

Ruby:

```
# @param {Integer[][]} image
# @return {Integer[][]}
def flip_and_invert_image(image)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $image
* @return Integer[][]
*/
function flipAndInvertImage($image) {

}
}
```

Dart:

```
class Solution {
List<List<int>> flipAndInvertImage(List<List<int>> image) {

}
}
```

Scala:

```
object Solution {
def flipAndInvertImage(image: Array[Array[Int]]): Array[Array[Int]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec flip_and_invert_image(image :: [[integer]]) :: [[integer]]
def flip_and_invert_image(image) do

end
end
```

Erlang:

```
-spec flip_and_invert_image(Image :: [[integer()]]) -> [[integer()]].
flip_and_invert_image(Image) ->
.
```

Racket:

```
(define/contract (flip-and-invert-image image)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Flipping an Image
```

```
* 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>> flipAndInvertImage(vector<vector<int>>& image) {

}
};
```

Java Solution:

```
/**
* Problem: Flipping an Image
* 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[][] flipAndInvertImage(int[][] image) {

}
}
```

Python3 Solution:

```
"""
Problem: Flipping an Image
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 flipAndInvertImage(self, image: List[List[int]]) -> List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Flipping an Image
* 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[][]} image
* @return {number[][]}
*/
var flipAndInvertImage = function(image) {

};
```

TypeScript Solution:

```
/**
* Problem: Flipping an Image
* 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 flipAndInvertImage(image: number[][]): number[][] {

};
```

C# Solution:

```
/*
* Problem: Flipping an Image
* 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[][] FlipAndInvertImage(int[][] image) {

}
}
```

C Solution:

```
/*
* Problem: Flipping an Image
* 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** flipAndInvertImage(int** image, int imageSize, int* imageColSize, int*
returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Flipping an Image
// 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 flipAndInvertImage(image [][]int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun flipAndInvertImage(image: Array<IntArray>): Array<IntArray> {

}
}
```

Swift Solution:

```
class Solution {
func flipAndInvertImage(_ image: [[Int]]) -> [[Int]] {

}
```

```
}
```

Rust Solution:

```
// Problem: Flipping an Image
// 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 flip_and_invert_image(image: Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} image
# @return {Integer[][]}
def flip_and_invert_image(image)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $image
* @return Integer[][]
*/
function flipAndInvertImage($image) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> flipAndInvertImage(List<List<int>> image) {

}
}
```

Scala Solution:

```
object Solution {
def flipAndInvertImage(image: Array[Array[Int]]): Array[Array[Int]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec flip_and_invert_image(image :: [[integer]]) :: [[integer]]
def flip_and_invert_image(image) do

end
end
```

Erlang Solution:

```
-spec flip_and_invert_image(Image :: [[integer()]]) -> [[integer()]].
flip_and_invert_image(Image) ->
.
```

Racket Solution:

```
(define/contract (flip-and-invert-image image)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```