Solution: Flipping an Image

Let's solve the Flipping an Image problem using the Bitwise Manipulation pattern.

We'll cover the following

Statement

Given that an image is represented by an $(n \times n)$ matrix containing 0s and 1s, flip and invert the image, and return the resultant image.

Horizontally flipping an image means that the mirror image of the matrix should be returned. Flipping $[1, 0, 0]$ horizontally results in $[0, 0, 1]$.

Inverting an image means that every 0 is replaced by 1, and every 1 is replaced by 0. Inverting $[0, 1, 1]$ results in $[1, 0, 0]$.

Constraints:

- Image should be a square matrix.
- $1 \leq n \leq 20$
- `images[i][j]` is either 0 or 1.

Solution

So far, you've probably brainstormed some approaches and have an idea of how to solve this problem. Let's explore some of these approaches and figure out which one to follow based on considerations such as time complexity and any implementation constraints.

Naive approach

The naive approach of solving this problem is to create separate functions for flipping the matrix horizontally and then inverting each individual element of the matrix. We'll use nested loops to flip the rows of the matrix by swapping the elements from both ends until we reach the middle of the row. Once the matrix has been flipped, we'll traverse it again and replace 1s with 0s and 0s with 1s to invert it. The naive approach uses two iterations of the whole matrix to flip and invert the matrix.

The time complexity of flipping the matrix is $O(n^2)$. The time complexity for inverting each element is also $O(n^2)$.

Optimized approach using bitwise manipulation

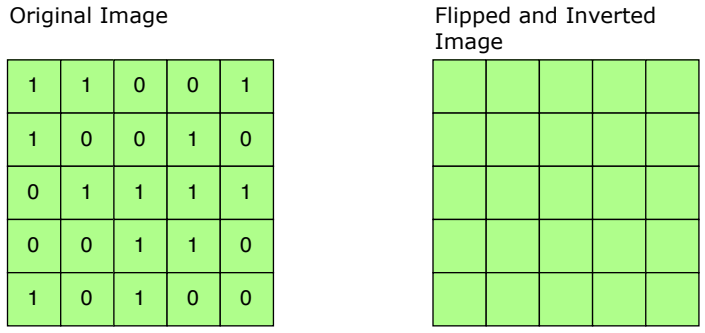
The problem statement asks us to flip and invert the binary 1s and 0s. This can be done by performing bitwise XOR on each row of the matrix. Problems such as this one are good examples to solve using the bitwise manipulation pattern. We will flip and invert the matrix using one iteration of the matrix.

While the overall time complexity of this approach is the same as the naive approach discussed above, bitwise operations have hardware support and, hence, are relatively faster compared to conditional statements. Instead of adding `if` statements to invert the elements, we can simply compute their XOR with `1`. When a `0` is encountered, it gets replaced with `1` as `0 XOR 1 = 1` and when a `1` is encountered, it's replaced with `0` as `1 XOR 1 = 0`.

Here is how we implement this algorithm:

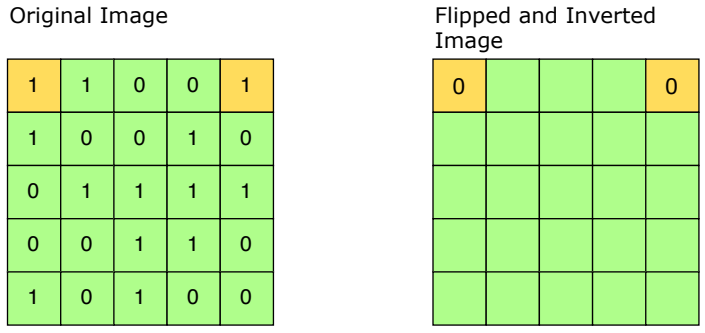
1. Compute the index of the middle element in each row.
2. For each element in the first half of each row, we will perform the following two operations:
    - We will compute the bitwise XOR of the current element with 1, which will invert the element's value. If the value is 0, it will be replaced with 1, and if the value is 1, it will be replaced with 0.
    - We will then swap the current element with the corresponding last element in the second half of the row, i.e., the element at the same distance from the end of the row as the current element is from the beginning of the row, after performing the same XOR operation on the second element as well.
3. Return the modified array after iterating through all the rows and flipping/inverting each element.

Let's look at the following illustration to get a better understanding of the solution:



Given the above image, flip and invert it.

1 of 17



Take bitwise XOR of these two elements with 1 and swap them.

2 of 17

Original Image

1	1	0	0	1
1	0	0	1	0
0	1	1	1	1
0	0	1	1	0
1	0	1	0	0

Flipped and Inverted Image

0	1		0	0

Take bitwise XOR of these two elements with 1 and swap them.

3 of 17

Original Image

1	1	0	0	1
1	0	0	1	0
0	1	1	1	1
0	0	1	1	0
1	0	1	0	0

Flipped and Inverted Image

0	1	1	0	0

Take bitwise XOR of the middle element with 1.

4 of 17

Original Image

1	1	0	0	1
1	0	0	1	0
0	1	1	1	1
0	0	1	1	0
1	0	1	0	0

Flipped and Inverted Image

0	1	1	0	0
1				0

Take bitwise XOR of these two elements with 1 and swap them.

5 of 17

Original Image

1	1	0	0	1
1	0	0	1	0
0	1	1	1	1
0	0	1	1	0
1	0	1	0	0

Flipped and Inverted Image

0	1	1	0	0
1	0		1	0

Take bitwise XOR of these two elements with 1 and swap them.

6 of 17

Original Image

1	1	0	0	1
1	0	0	1	0
0	1	1	1	1
0	0	1	1	0
1	0	1	0	0

Flipped and Inverted Image

0	1	1	0	0
1	0	1	1	0

Take bitwise XOR of the middle element with 1.

7 of 17

Original Image

1	1	0	0	1
1	0	0	1	0
0	1	1	1	1
0	0	1	1	0
1	0	1	0	0

Flipped and Inverted Image

0	1	1	0	0
1	0	1	1	0
0				1

Take bitwise XOR of these two elements with 1 and swap them.

8 of 17

Original Image

1	1	0	0	1
1	0	0	1	0
0	1	1	1	1
0	0	1	1	0
1	0	1	0	0

Flipped and Inverted Image

0	1	1	0	0
1	0	1	1	0
0	0		0	1

Take bitwise XOR of these two elements with 1 and swap them.

9 of 17

Original Image

1	1	0	0	1
1	0	0	1	0
0	1	1	1	1
0	0	1	1	0
1	0	1	0	0

Flipped and Inverted Image

0	1	1	0	0
1	0	1	1	0
0	0	0	0	1

Take bitwise XOR of the middle element with 1.

10 of 17

Original Image

1	1	0	0	1
1	0	0	1	0
0	1	1	1	1
0	0	1	1	0
1	0	1	0	0

Flipped and Inverted Image

0	1	1	0	0
1	0	1	1	0
0	0	0	0	1
1				1

Take bitwise XOR of these two elements with 1 and swap them.

11 of 17

Original Image

1	1	0	0	1
1	0	0	1	0
0	1	1	1	1
0	0	1	1	0
1	0	1	0	0

Flipped and Inverted Image

0	1	1	0	0
1	0	1	1	0
0	0	0	0	1
1	0		1	1

Take bitwise XOR of these two elements with 1 and swap them.

Original Image

1	1	0	0	1
1	0	0	1	0
0	1	1	1	1
0	0	1	1	0
1	0	1	0	0

Flipped and Inverted Image

0	1	1	0	0
1	0	1	1	0
0	0	0	0	1
1	0	0	1	1

Take bitwise XOR of the middle element with 1.

Original Image

1	1	0	0	1
1	0	0	1	0
0	1	1	1	1
0	0	1	1	0
1	0	1	0	0

Flipped and Inverted Image

0	1	1	0	0
1	0	1	1	0
0	0	0	0	1
1	0	0	1	1
1				0

Take bitwise XOR of these two elements with 1 and swap them.

Original Image

1	1	0	0	1
1	0	0	1	0
0	1	1	1	1
0	0	1	1	0
1	0	1	0	0

Flipped and Inverted Image

0	1	1	0	0
1	0	1	1	0
0	0	0	0	1
1	0	0	1	1
1	1		1	0

Take bitwise XOR of these two elements with 1 and swap them.

15 of 17

Original Image

1	1	0	0	1
1	0	0	1	0
0	1	1	1	1
0	0	1	1	0
1	0	1	0	0

Flipped and Inverted Image

0	1	1	0	0
1	0	1	1	0
0	0	0	0	1
1	0	0	1	1
1	1	0	1	0

Take bitwise XOR of the middle element with 1.

Original Image

1	1	0	0	1
1	0	0	1	0
0	1	1	1	1
0	0	1	1	0
1	0	1	0	0

Flipped and Inverted Image

0	1	1	0	0
1	0	1	1	0
0	0	0	0	1
1	0	0	1	1
1	1	0	1	0

Above is the flipped and inverted image.

17 of 17

Let’s look at the code for this solution below:

Java

```
import java.util.*;

class FlipImage {

    public static int[][] flipAndInvertImage(int[][] image) {
    // Get the number of rows in the image
    int rowCount = image.length;
```

```
 7      int rowCount = image.length;
 8      // Calculate the middle index of the rows
 9      int mid = (rowCount + 1) / 2;
10
11      // Iterate over each row of the image
12      for (int[] row : image) {
13          // Iterate over the first half of each row
```