Previous Next

48. Rotate Image

Dec. 9, 2018 | 103.2K views

Average Rating: 3.09 (78 votes)

You are given an *n* x *n* 2D matrix representing an image.

Rotate the image by 90 degrees (clockwise).

Note:

You have to rotate the image **in-place**, which means you have to modify the input 2D matrix directly. **DO NOT** allocate another 2D matrix and do the rotation.

Example 1:

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

rotate the input matrix in-place such that it becomes:
[
  [7,4,1],
  [8,5,2],
  [9,6,3]
]
```

Example 2:

```
Given input matrix =
[
  [ 5, 1, 9,11],
  [ 2, 4, 8,10],
  [13, 3, 6, 7],
  [15,14,12,16]
],

rotate the input matrix in-place such that it becomes:
[
  [15,13, 2, 5],
  [14, 3, 4, 1],
  [12, 6, 8, 9],
  [16, 7,10,11]
]
```

Solution

Approach 1 : Transpose and then reverse

The obvious idea would be to transpose the matrix first and then reverse each row. This simple approach already demonstrates the best possible time complexity $\mathcal{O}(N^2)$.

Java Python Copy

```python
class Solution:
    def rotate(self, matrix):
        """
        :type matrix: List[List[int]]
        :rtype: void Do not return anything, modify matrix in-place instead.
        """
        n = len(matrix[0])
        # transpose matrix
        for i in range(n):
            for j in range(i, n):
                matrix[j][i], matrix[i][j] = matrix[i][j], matrix[j][i]

        # reverse each row
        for i in range(n):
            matrix[i].reverse()
```

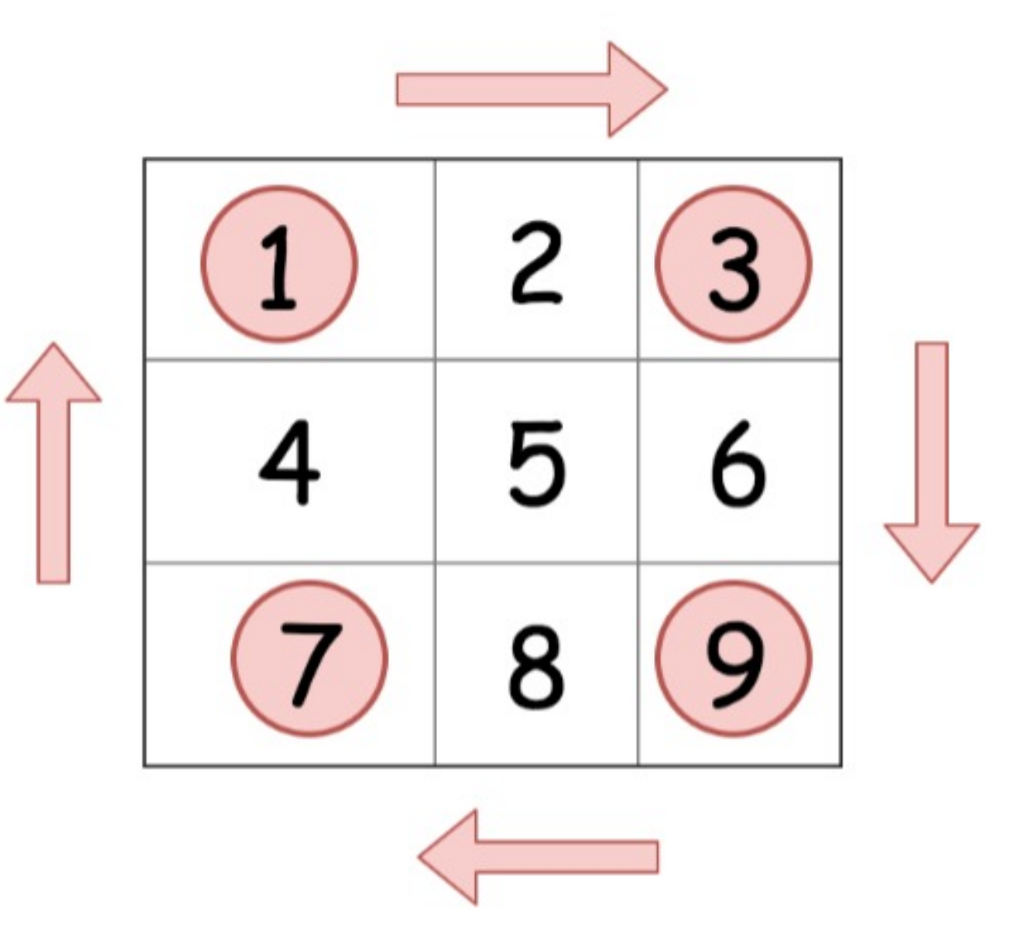
- Time complexity : $\mathcal{O}(N^2)$.
- Space complexity : $\mathcal{O}(1)$ since we do a rotation *in place*.

Approach 2 : Rotate four rectangles

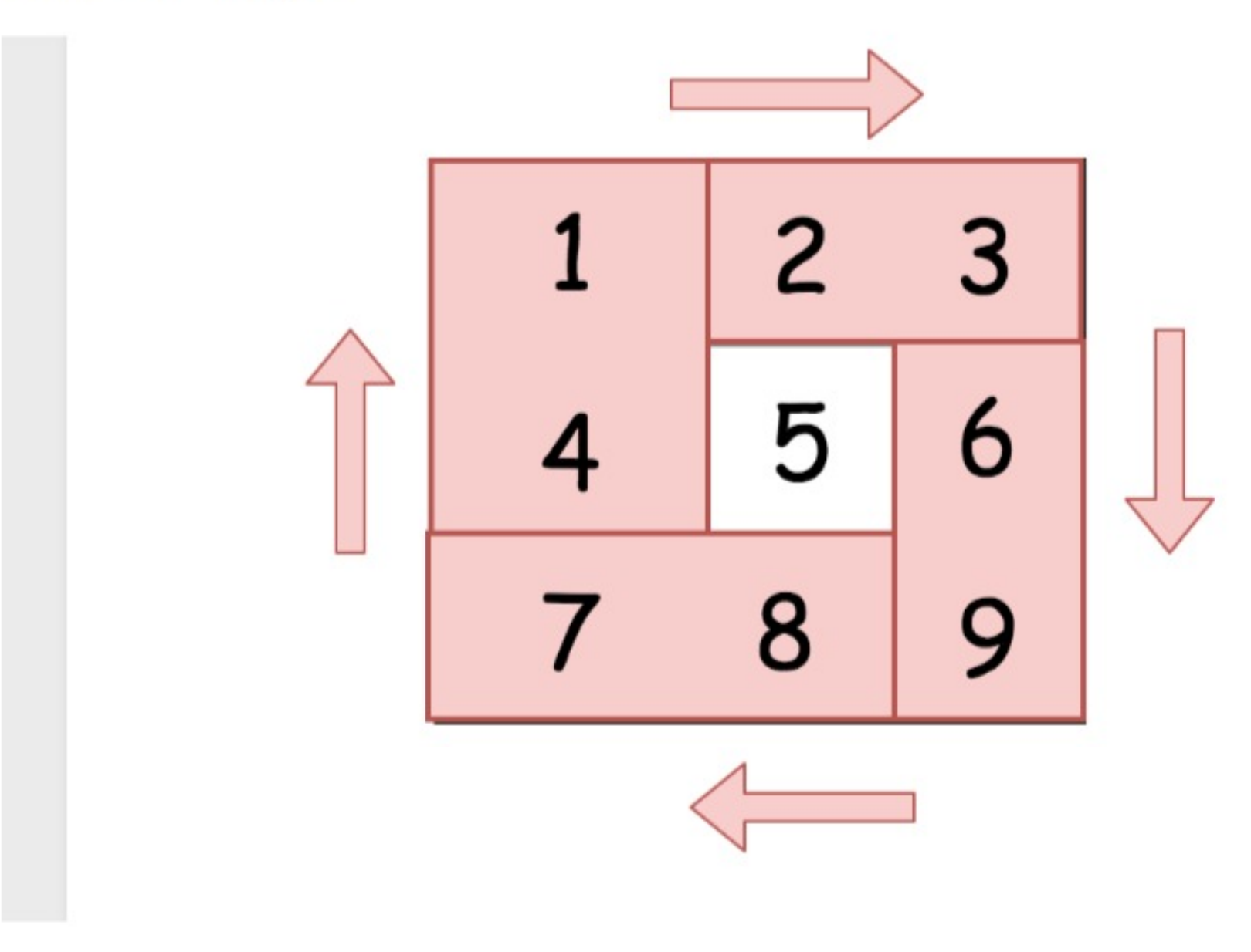
Intuition

Approach 1 makes two passes through the matrix, though it's possible to make a rotation in one pass.

To figure out how let's check how each element in the angle moves during the rotation.

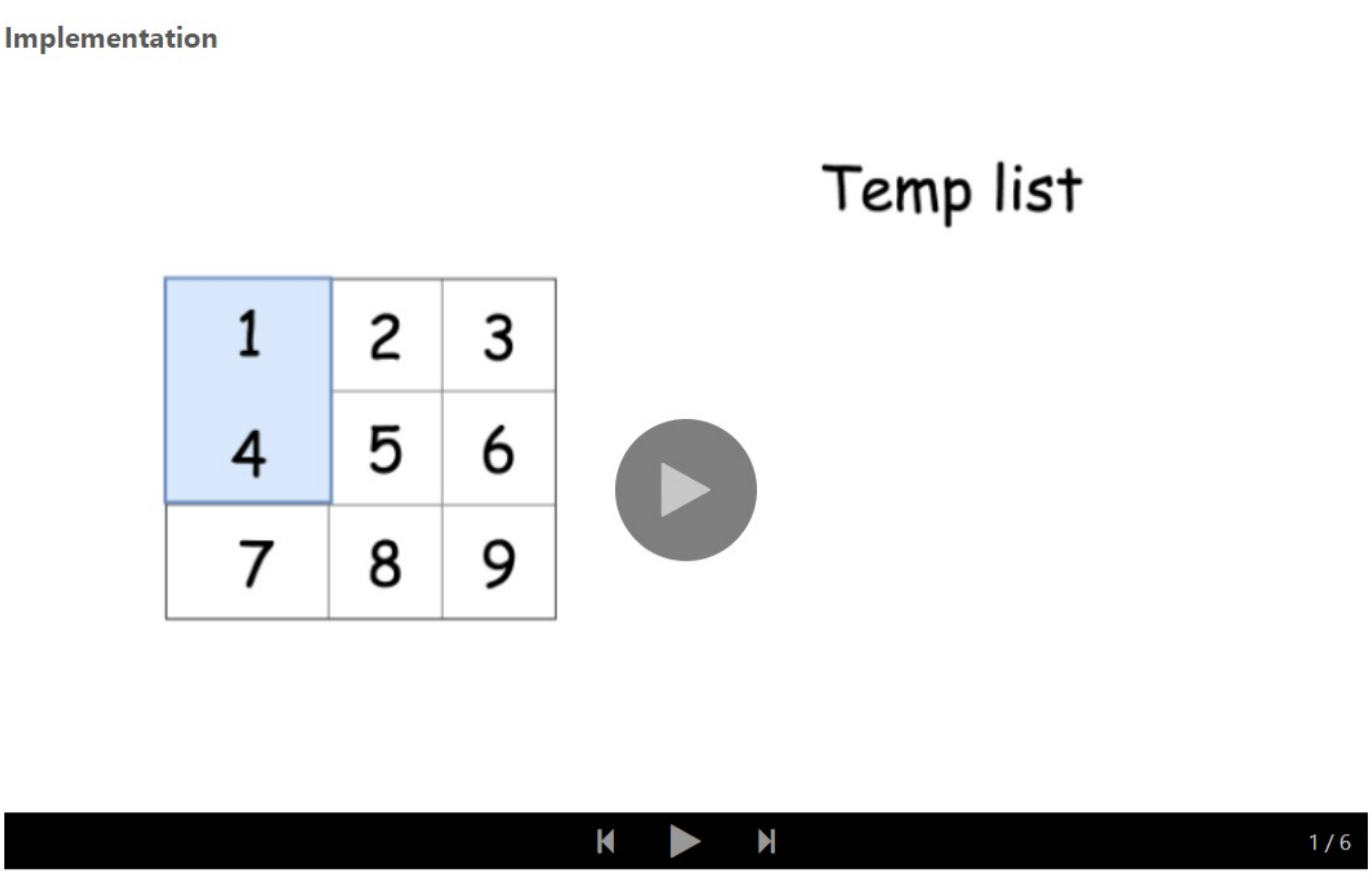


That gives us an idea to split a given matrix in four rectangles and reduce the initial problem to the rotation of these rectangles.



Now the solution is quite straightforward - one could move across the elements in the first rectangle and rotate them using a temp list of `4` elements.

Implementation



Java Python Copy

```python
class Solution:
    def rotate(self, matrix):
        """
        :type matrix: List[List[int]]
        :rtype: void Do not return anything, modify matrix in-place instead.
        """
        n = len(matrix[0])
        for i in range(n // 2 + n % 2):
            for j in range(n // 2):
                tmp = [0] * 4
                row, col = i, j
                # store 4 elements in tmp
                for k in range(4):
                    tmp[k] = matrix[row][col]
                    row, col = col, n - 1 - row
                # rotate 4 elements
                for k in range(4):
                    matrix[row][col] = tmp[(k - 1) % 4]
                    row, col = col, n - 1 - row
```

Complexity Analysis

- Time complexity : $\mathcal{O}(N^2)$ is a complexity given by two inserted loops.
- Space complexity : $\mathcal{O}(1)$ since we do a rotation *in place* and allocate only the list of `4` elements as a temporary helper.

Approach 3 : Rotate four rectangles in one single loop

The idea is the same as in the approach 2, but everything is done in one single loop and hence it's a way more elegant (kudos go to @gxldragon).

Java Python Copy

```python
class Solution:
    def rotate(self, matrix):
        """
        :type matrix: List[List[int]]
        :rtype: void Do not return anything, modify matrix in-place instead.
        """
        n = len(matrix[0])
        for i in range(n // 2 + n % 2):
            for j in range(n // 2):
                tmp = matrix[n - 1 - j][i]
                matrix[n - 1 - j][i] = matrix[n - 1 - i][n - j - 1]
                matrix[n - 1 - i][n - j - 1] = matrix[j][n - 1 -i]
                matrix[j][n - 1 - i] = matrix[i][j]
                matrix[i][j] = tmp
```

- Time complexity : $\mathcal{O}(N^2)$ is a complexity given by two inserted loops.
- Space complexity : $\mathcal{O}(1)$ since we do a rotation *in place*.

Rate this article: ★★★★★

Previous Next

Comments: 37 Sort By

Type comment here... (Markdown is supported)

Preview Post

tamilarasan88 ★21 December 11, 2018 1:52 AM

Step1 : doTranspose(matrix); //do the inplace transporse of matrix. consider lower diagonal of matrix and interchange matrix[row][col] to matrix[col][row]
Step2: reverseRow(matrix[i]); // for each row of matrix 'i', do reverse of the matrix row by parsing from start till mid index.

21 Share Reply

SHOW 1 REPLY

hydezhao ★21 January 6, 2019 3:36 PM Report

To rotate, we can transpose the matrix firstly, then reverse columns:
for a matrix:
[1,2,3],
[4,5,6],
[7,8,9]

Read More

21 Share Reply

SHOW 2 REPLIES

kkzeng ★222 June 8, 2019 11:30 AM

Essentially, the approach in soln 1 vs. soln 2 and 3 are the same. If you look at the formula in soln 2 and 3 of entry (i, j) -> (j, n-1-i). This is the same as transposing the the element and then reversing the columns.
We first transpose (i, j) -> (j, i) and then flip the columns (j, i) -> (j, n-1-i).

Read More

13 Share Reply

SHOW 2 REPLIES

sammyboy ★14 March 7, 2020 10:03 PM Report

This is such an irritating problem. It's not "hard" but under the pressure of an interview it's prime choking material just because there is so much mental juggling. I feel like beyond the low bar of understanding matrix rotation, this proves nothing. If you tested someone by asking them to do this without worrying about allocating a second matrix you'd probably learn all you needed about their actual competence and then followup with a question about "how would you do this in place" and let

Read More

9 Share Reply

SHOW 1 REPLY

Dr_Sean ★535 January 4, 2019 8:00 AM Report

Easy 4-line Python code:

```
class Solution:
    def rotate(self, matrix):
        matrix.reverse()
```

Read More

11 Share Reply

SHOW 3 REPLIES

gxldragon ★9 January 11, 2019 10:54 AM

Same solution with less code.

```
    public void rotate(int[][] matrix) {
        int n = matrix.length;
        for (int i = 0; i < (n + 1) / 2; i ++){
```

Read More

9 Share Reply

SHOW 1 REPLY

Nevsanev ★1112 March 2, 2019 8:34 PM Report

For approach 3, if you are confused with the range of the index `i` and `j`, you can check my post:

https://leetcode.com/problems/rotate-image/discuss/247174/Easy-Java-solution-with-explanation-processing-the-matrix-from-outer-to-inner

Read More

5 Share Reply

SHOW 1 REPLY

pinkfloyda ★740 March 12, 2019 11:08 AM

The transpose and reverse way is elegant, but could someone help explain why it works? I cannot see the intuition here. Seems the same trick can be used for anti-clock wise rotation, clock wise rotation 1 time, 2 times...... etc.

6 Share Reply

aconfee ★1 April 19, 2020 9:43 PM Report

When doing the final (most optimal) solution, I like to think of the outer loop as 'layer' and the inner loop as 'offset'. You can think of the layer as moving diagonally inwards on either end of the matrix. Offset is the next increment (in any direction). So this way you iterate your outer loop N / 2 times (until the layer is 1 or 0 elements in the middle) and increment the offset starting at layer, and ending at N - layer - 1; basically within the bounds of the layer, minus one since that last element was the starting

Read More

1 Share Reply

hanjuTsai ★0 January 24, 2019 8:24 PM Report

I think that my solution is much more easier

```
class Solution:
    def rotate(self, matrix):
        """
```

Read More

1 Share Reply

‹ 1 2 3 4 ›