Rotate Image

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

Rotate the image by 90 degrees (clockwise).

Follow up:
Could you do this in-place?

Solution 1

here give a common method to solve the image rotation problems.

```
/*
 * clockwise rotate
 * first reverse up to down, then swap the symmetry
 * 1 2 3     7 8 9     7 4 1
 * 4 5 6  => 4 5 6  => 8 5 2
 * 7 8 9     1 2 3     9 6 3
*/
void rotate(vector<vector<int> > &matrix) {
    reverse(matrix.begin(), matrix.end());
    for (int i = 0; i < matrix.size(); ++i) {
        for (int j = i + 1; j < matrix[i].size(); ++j)
            swap(matrix[i][j], matrix[j][i]);
    }
}

/*
 * anticlockwise rotate
 * first reverse left to right, then swap the symmetry
 * 1 2 3     3 2 1     3 6 9
 * 4 5 6  => 6 5 4  => 2 5 8
 * 7 8 9     9 8 7     1 4 7
*/
void anti_rotate(vector<vector<int> > &matrix) {
    for (auto vi : matrix) reverse(vi.begin(), vi.end());
    for (int i = 0; i < matrix.size(); ++i) {
        for (int j = i + 1; j < matrix[i].size(); ++j)
            swap(matrix[i][j], matrix[j][i]);
    }
}
```

written by shichaotan original link here

Solution 2

While these solutions are Python, I think they're understandable/interesting for non-Python coders as well. But before I begin: No mathematician would call a matrix `matrix`, so I'll use the usual `A`. Also, btw, the 40 ms reached by two of the solutions is I think the fastest achieved by Python solutions so far.

Most Pythonic - `[::-1]` and `zip` - 44 ms

The most pythonic solution is a simple one-liner using `[::-1]` to flip the matrix upside down and then `zip` to transpose it. It assigns the result back into `A`, so it's "in-place" in a sense and the OJ accepts it as such, though some people might not.

```
class Solution:
    def rotate(self, A):
        A[:] = zip(*A[::-1])
```

Most Direct - 52 ms

A 100% in-place solution. It even reads and writes each matrix element only once and doesn't even use an extra temporary variable to hold them. It walks over the *"top-left quadrant"* of the matrix and directly rotates each element with the three corresponding elements in the other three quadrants. Note that I'm moving the four elements in parallel and that `[~i]` is way nicer than `[n-1-i]`.

```
class Solution:
    def rotate(self, A):
        n = len(A)
        for i in range(n/2):
            for j in range(n-n/2):
                A[i][j], A[~j][i], A[~i][~j], A[j][~i] = \
                         A[~j][i], A[~i][~j], A[j][~i], A[i][j]
```

Clean Most Pythonic - 56 ms

While the OJ accepts the above solution, the the result rows are actually tuples, not lists, so it's a bit dirty. To fix this, we can just apply `list` to every row:

```
class Solution:
    def rotate(self, A):
        A[:] = map(list, zip(*A[::-1]))
```

List Comprehension - 60 ms

If you don't like `zip`, you can use a nested list comprehension instead:

```
class Solution:
    def rotate(self, A):
        A[:] = [[row[i] for row in A[::-1]] for i in range(len(A))]
```

Almost as Direct - 40 ms

If you don't like the little repetitive code of the above "Most Direct" solution, we can instead do each four-cycle of elements by using three swaps of just two elements.

```
class Solution:
    def rotate(self, A):
        n = len(A)
        for i in range(n/2):
            for j in range(n-n/2):
                for _ in '123':
                    A[i][j], A[~j][i], i, j = A[~j][i], A[i][j], ~j, ~i
                i = ~j
```

Flip Flip - 40 ms

Basically the same as the first solution, but using `reverse` instead of `[::-1]` and transposing the matrix with loops instead of `zip`. It's 100% in-place, just instead of only moving elements around, it also moves the rows around.

```
class Solution:
    def rotate(self, A):
        A.reverse()
        for i in range(len(A)):
            for j in range(i):
                A[i][j], A[j][i] = A[j][i], A[i][j]
```

Flip Flip, all by myself - 48 ms

Similar again, but I first transpose and then flip left-right instead of upside-down, and do it all by myself in loops. This one is 100% in-place again in the sense of just moving the elements.

```
class Solution:
    def rotate(self, A):
        n = len(A)
        for i in range(n):
            for j in range(i):
                A[i][j], A[j][i] = A[j][i], A[i][j]
        for row in A:
            for j in range(n/2):
                row[j], row[~j] = row[~j], row[j]
```

written by StefanPochmann original link here

Solution 3

The idea was firstly transpose the matrix and then flip it symmetrically. For instance,

```
1  2  3
4  5  6
7  8  9
```

after transpose, it will be swap(matrix[i][j], matrix[j][i])

```
1  4  7
2  5  8
3  6  9
```

Then flip the matrix horizontally. (swap(matrix[i][j], matrix[i][matrix.length-1-j])

```
7  4  1
8  5  2
9  6  3
```

Hope this helps.

```
public class Solution {
    public void rotate(int[][] matrix) {
        for(int i = 0; i<matrix.length; i++){
            for(int j = i; j<matrix[0].length; j++){
                int temp = 0;
                temp = matrix[i][j];
                matrix[i][j] = matrix[j][i];
                matrix[j][i] = temp;
            }
        }
        for(int i =0 ; i<matrix.length; i++){
            for(int j = 0; j<matrix.length/2; j++){
                int temp = 0;
                temp = matrix[i][j];
                matrix[i][j] = matrix[i][matrix.length-1-j];
                matrix[i][matrix.length-1-j] = temp;
            }
        }
    }
}
```

written by LuckyIdiot original link here