Spiral Matrix II

Given an integer n, generate a square matrix filled with elements from 1 to n^2 in spiral order.

For example,
Given $n =$ `3`,

You should return the following matrix:

```
[
 [ 1, 2, 3 ],
 [ 8, 9, 4 ],
 [ 7, 6, 5 ]
]
```

Solution 1

Solution 1: *Build it inside-out* - 44 ms, 5 lines

Start with the empty matrix, add the numbers in reverse order until we added the number 1. Always rotate the matrix clockwise and add a top row:

```
    ||  =>  |9|  =>  |8|      |6 7|      |4 5|      |1 2 3|
                    |9|  =>  |9 8|  =>  |9 6|  =>  |8 9 4|
                                        |8 7|      |7 6 5|
```

The code:

```
def generateMatrix(self, n):
    A, lo = [], n*n+1
    while lo > 1:
        lo, hi = lo - len(A), lo
        A = [range(lo, hi)] + zip(*A[::-1])
    return A
```

While this isn't O(n^2), it's actually quite fast, presumably due to me not doing much in Python but relying on `zip` and `range` and `+` being fast. I got it accepted in 44 ms, matching the fastest time for recent Python submissions (according to the submission detail page).

Solution 2: *Ugly inside-out* - 48 ms, 4 lines

Same as solution 1, but without helper variables. Saves a line, but makes it ugly. Also, because I access A[0][0], I had to handle the n=0 case differently.

```
def generateMatrix(self, n):
    A = [[n*n]]
    while A[0][0] > 1:
        A = [range(A[0][0] - len(A), A[0][0])] + zip(*A[::-1])
    return A * (n>0)
```

Solution 3: *Walk the spiral* - 52 ms, 9 lines

Initialize the matrix with zeros, then walk the spiral path and write the numbers 1 to n*n. Make a right turn when the cell ahead is already non-zero.

```
def generateMatrix(self, n):
    A = [[0] * n for _ in range(n)]
    i, j, di, dj = 0, 0, 0, 1
    for k in xrange(n*n):
        A[i][j] = k + 1
        if A[(i+di)%n][(j+dj)%n]:
            di, dj = dj, -di
        i += di
        j += dj
    return A
```

written by StefanPochmann original link here

Solution 2

```
class Solution {
    public:
        vector<vector<int> > generateMatrix(int n) {
            vector<vector<int> > ret( n, vector<int>(n) );
            int k = 1, i = 0;
            while( k <= n * n )
            {
                int j = i;
                    // four steps
                while( j < n - i )              // 1. horizonal, left to right
                    ret[i][j++] = k++;
                j = i + 1;
                while( j < n - i )              // 2. vertical, top to bottom
                    ret[j++][n-i-1] = k++;
                j = n - i - 2;
                while( j > i )                   // 3. horizonal, right to left
                    ret[n-i-1][j--] = k++;
                j = n - i - 1;
                while( j > i )                   // 4. vertical, bottom to  top
                    ret[j--][i] = k++;
                i++;      // next loop
            }
            return ret;
        }
    };
```

written by AllenYick original link here

Solution 3

This is my solution for Spiral Matrix I, https://oj.leetcode.com/discuss/12228/super-simple-and-easy-to-understand-solution. If you can understand that, this one is a no brainer :)

Guess what? I just made several lines of change (with comment "//change") from that and I have the following AC code:

```
public class Solution {
    public int[][] generateMatrix(int n) {
        // Declaration
        int[][] matrix = new int[n][n];

        // Edge Case
        if (n == 0) {
            return matrix;
        }

        // Normal Case
        int rowStart = 0;
        int rowEnd = n-1;
        int colStart = 0;
        int colEnd = n-1;
        int num = 1; //change

        while (rowStart <= rowEnd && colStart <= colEnd) {
            for (int i = colStart; i <= colEnd; i ++) {
                matrix[rowStart][i] = num ++; //change
            }
            rowStart ++;

            for (int i = rowStart; i <= rowEnd; i ++) {
                matrix[i][colEnd] = num ++; //change
            }
            colEnd --;

            for (int i = colEnd; i >= colStart; i --) {
                if (rowStart <= rowEnd)
                    matrix[rowEnd][i] = num ++; //change
            }
            rowEnd --;

            for (int i = rowEnd; i >= rowStart; i --) {
                if (colStart <= colEnd)
                    matrix[i][colStart] = num ++; //change
            }
            colStart ++;
        }

        return matrix;
    }
}
```

Obviously, you could merge colStart and colEnd into rowStart and rowEnd because

it is a square matrix. But this is easily extensible to matrices that are m*n.

Hope this helps :)

written by qwl5004 original link here