|  |  |
| --- | --- |
| **Ex no: 1 B** | **PROGRAMS IN 2D ARRAY** |
| **Date:** |

# Aim

To execute programs in 2 dimensional array.

Program:1

Spiral matrix

Code:

class Solution(object):

    def spiralOrder(self, matrix):

        s=[]

        left,right=0,len(matrix[0])

        top,bottom=0,len(matrix)

        while left < right and top<bottom:

            for i in range(left,right):

                s.append(matrix[top][i])

            top+=1

            for i in range(top,bottom):

                s.append(matrix[i][right-1])

            right-=1

            if not(left<right and top<bottom):

                break

            for i in range(right-1,left-1,-1):

                s.append(matrix[bottom-1][i])

            bottom-=1

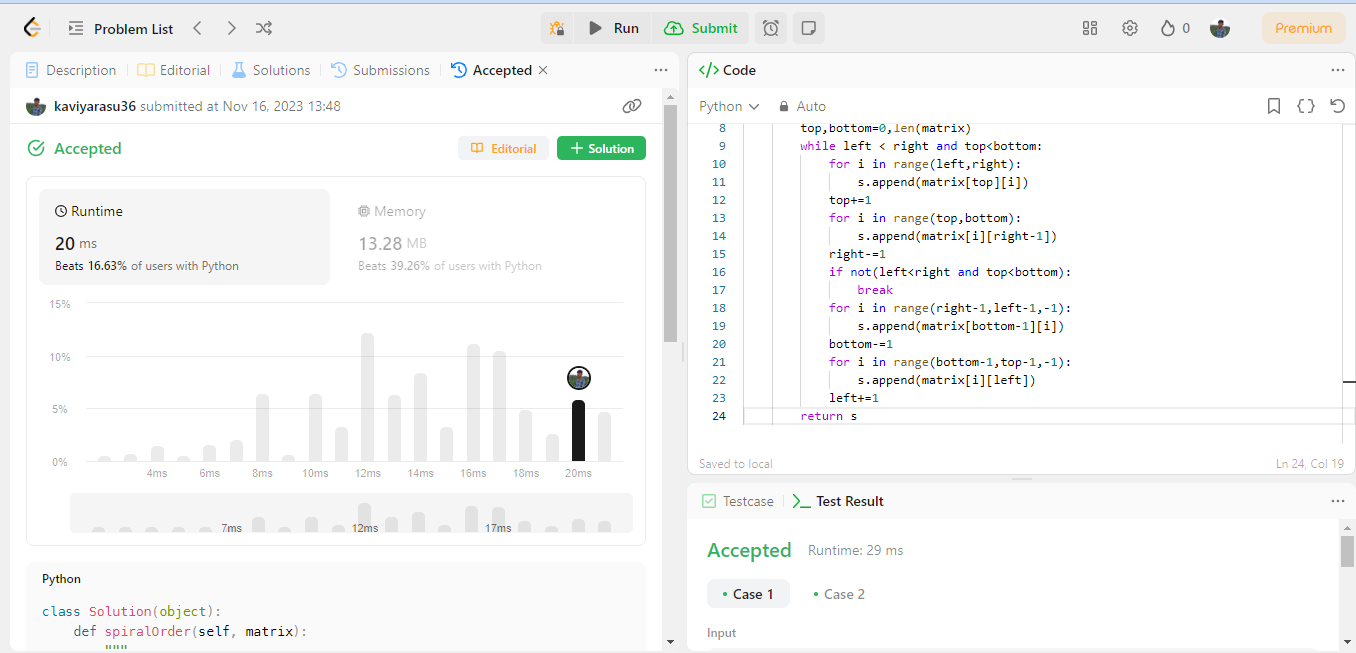
            for i in range(bottom-1,top-1,-1):

                s.append(matrix[i][left])

            left+=1

        return s

Output:



TIMECOMPLEXITY:O(n^2) SPACE COMPLEXITY: O(1)

717822i124-Kaviyarasu.c

# Program:2 Spiral matrix 2

# Code:

class Solution(object):

    def generateMatrix(self, n):

        """

        717822I124-kaviyarasu.C

        """

        matrix=[[0]\*n for i in range(n)]

        left,right=0,n-1

        top,bottom=0,n-1

        count=1

        while left<=right:

            for i in range(left,right+1):

                matrix[top][i]=count

                count+=1

            top+=1

            for i in range(top,bottom+1):

                matrix[i][right]=count

                count+=1

            right-=1

            for i in range(right,left-1,-1):

                matrix[bottom][i]=count

                count+=1

            bottom-=1

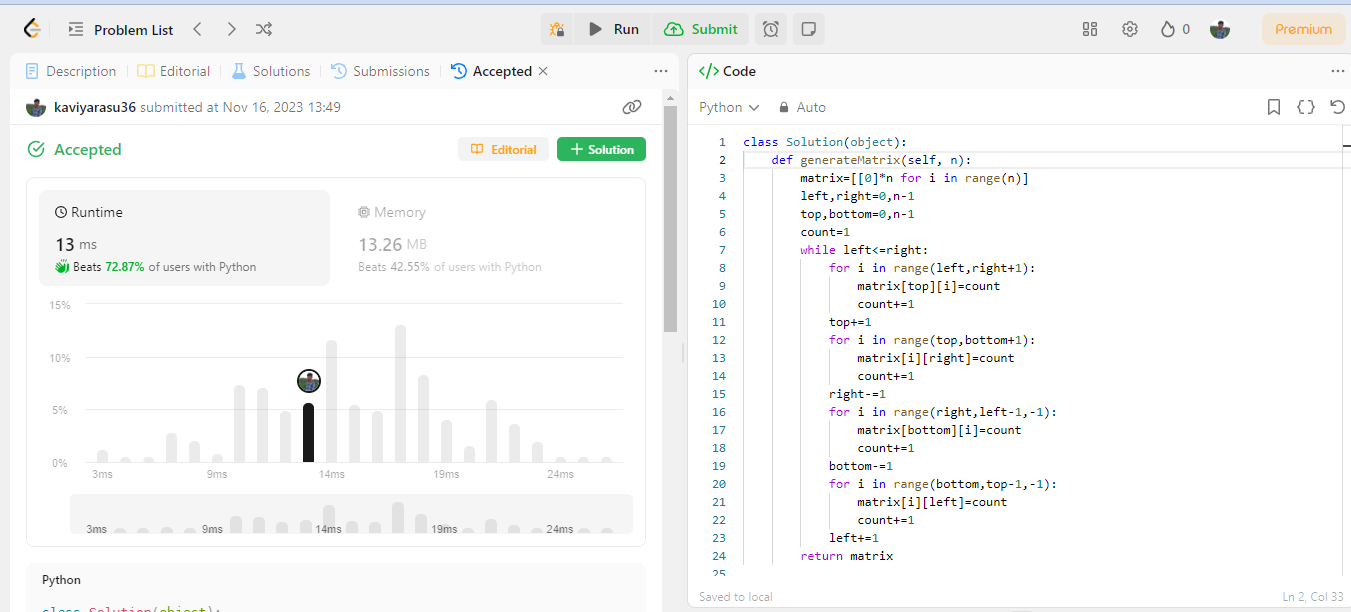
            for i in range(bottom,top-1,-1):

                matrix[i][left]=count

                count+=1

            left+=1

        return matrix



TIMECOMPLEXITY:O(n^2) SPACE COMPLEXITY: O(n)

717822i124-Kaviyarasu.c

# Program:3 Spiralmatrix 3

# Code:

class Solution(object):

    def spiralMatrixIII(self, rows, cols, rStart, cStart):

        ans = []

        left, right = cStart, cStart + 1

        top, bottom = rStart, rStart + 1

        current = 1

        move = 0

        while current <= rows \* cols:

            for i in range(left + move, right + 1):

                if self.inbound(top, i, rows, cols):

                    ans.append([top, i])

                    current += 1

            left -= 1

            for i in range(top + 1, bottom + 1):

                if self.inbound(i, right, rows, cols):

                    ans.append([i, right])

                    current += 1

            top -= 1

            for i in range(right - 1, left - 1, -1):

                if self.inbound(bottom, i, rows, cols):

                    ans.append([bottom, i])

                    current += 1

            right += 1

            for i in range(bottom - 1, top - 1, -1):

                if self.inbound(i, left, rows, cols):

                    ans.append([i, left])

                    current += 1

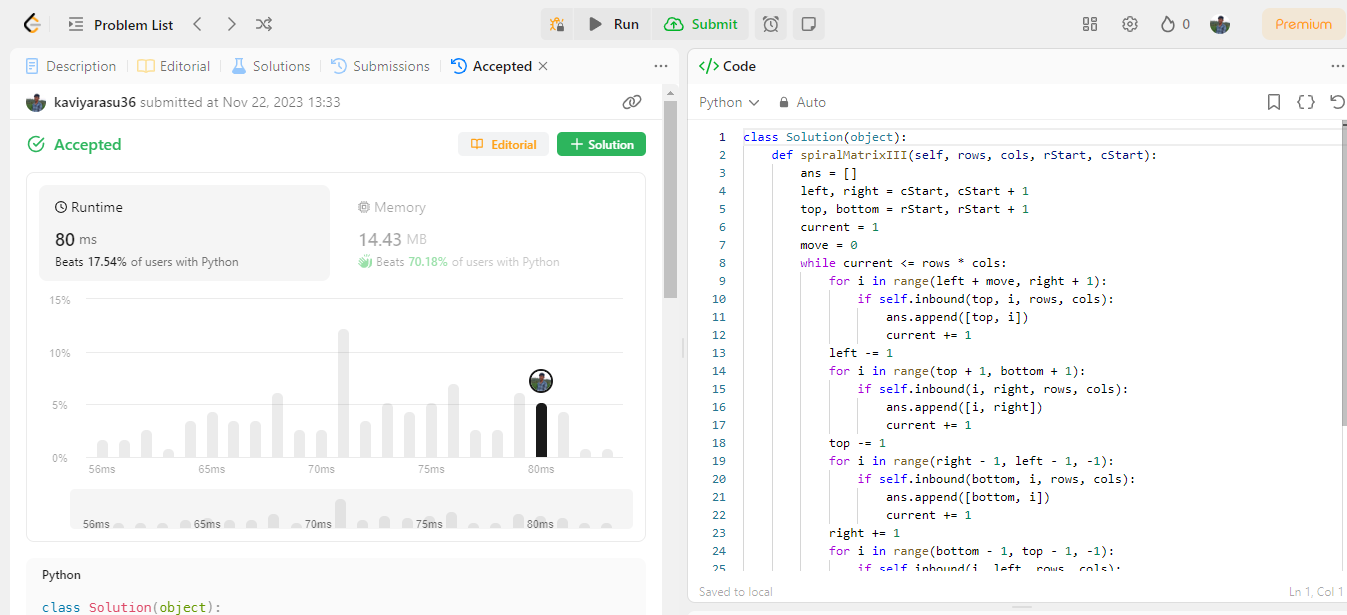
            bottom += 1

            move = 1

        return ans

    def inbound(self, r, c, rows, cols):

        return 0 <= r < rows and 0 <= c < cols



TIMECOMPLEXITY:O(n^2) SPACE COMPLEXITY: O(n)

717822i124-Kaviyarasu.c

# Program:4 Spiral matrix 4

# Code:

class Solution(object):

    def spiralMatrix(self, m, n, head):

        matrix = [[0 for i in range(n)] for j in range(m)]

        left, right = 0, n

        top, bottom = 0, m

        while left < right and top < bottom:

            for i in range(left, right):

                if not head:

                    matrix[top][i] = -1

                else:

                    matrix[top][i] = head.val

                    head = head.next

            top += 1

            for i in range(top, bottom):

                if not head:

                    matrix[i][right-1] = -1

                else:

                    matrix[i][right-1] = head.val

                    head = head.next

            right -= 1

            if not (left < right and top < bottom):

                break

            for i in range(right-1, left-1, -1):

                if not head:

                    matrix[bottom-1][i] = -1

                else:

                    matrix[bottom-1][i] = head.val

                    head = head.next

            bottom -= 1

            for i in range(bottom-1, top-1, -1):

                if not head:

                    matrix[i][left] = -1

                else:

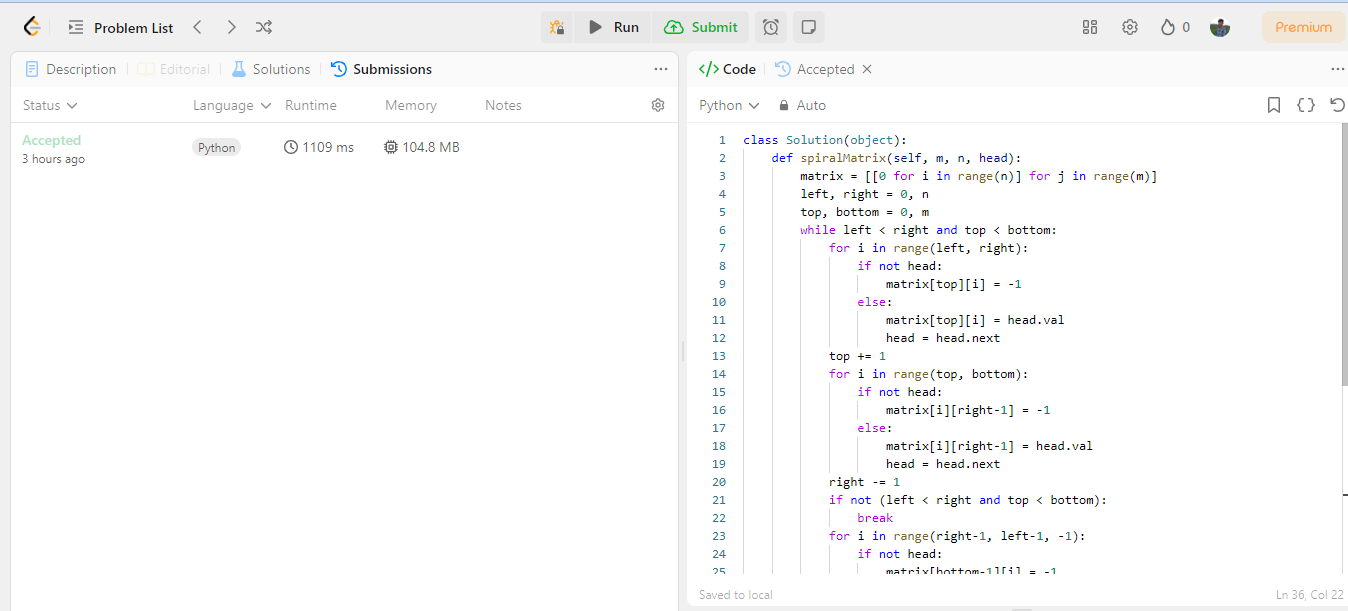
                    matrix[i][left] = head.val

                    head = head.next

            left += 1

        return matrix

717822i124-Kaviyarasu.c



TIMECOMPLEXITY:O(n^2) SPACE COMPLEXITY: O(1)

# Program:5 Set matrix zeroes Code:

class Solution(object):

    def setZeroes(self, matrix):

        row, col = set(), set()

        m = len(matrix)

        n = len(matrix[0])

        for i in range(m):

            for j in range(n):

                if matrix[i][j] == 0:

                    row.add(i)

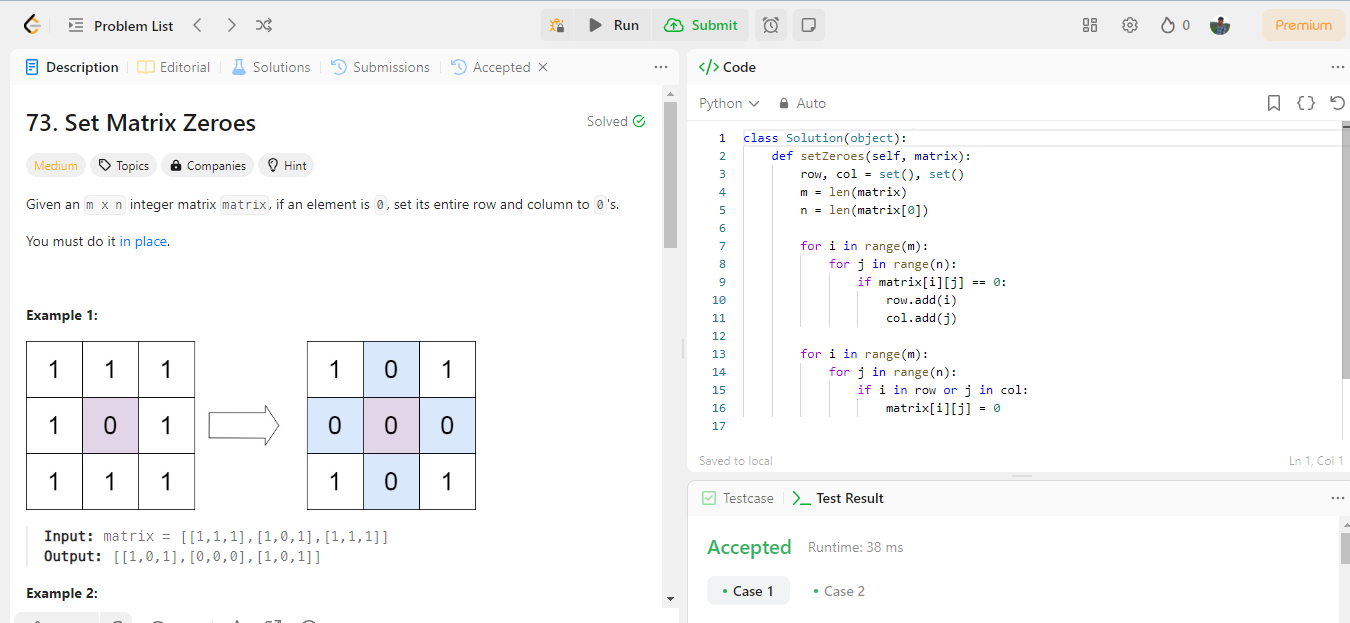
                    col.add(j)

        for i in range(m):

            for j in range(n):

                if i in row or j in col:

                    matrix[i][j] = 0



TIMECOMPLEXITY:O(n^2) SPACE COMPLEXITY: O(n)

717822i124-Kaviyarasu.c

# Program:6

**Search a 2d matrix 2**

**Code:**

class Solution(object):

    def searchMatrix(self, matrix, target):

        rows = len(matrix)

        cols = len(matrix[0])

        i = 0

        j = cols - 1

        while i < rows and j >= 0:

            if matrix[i][j] == target:

                return True

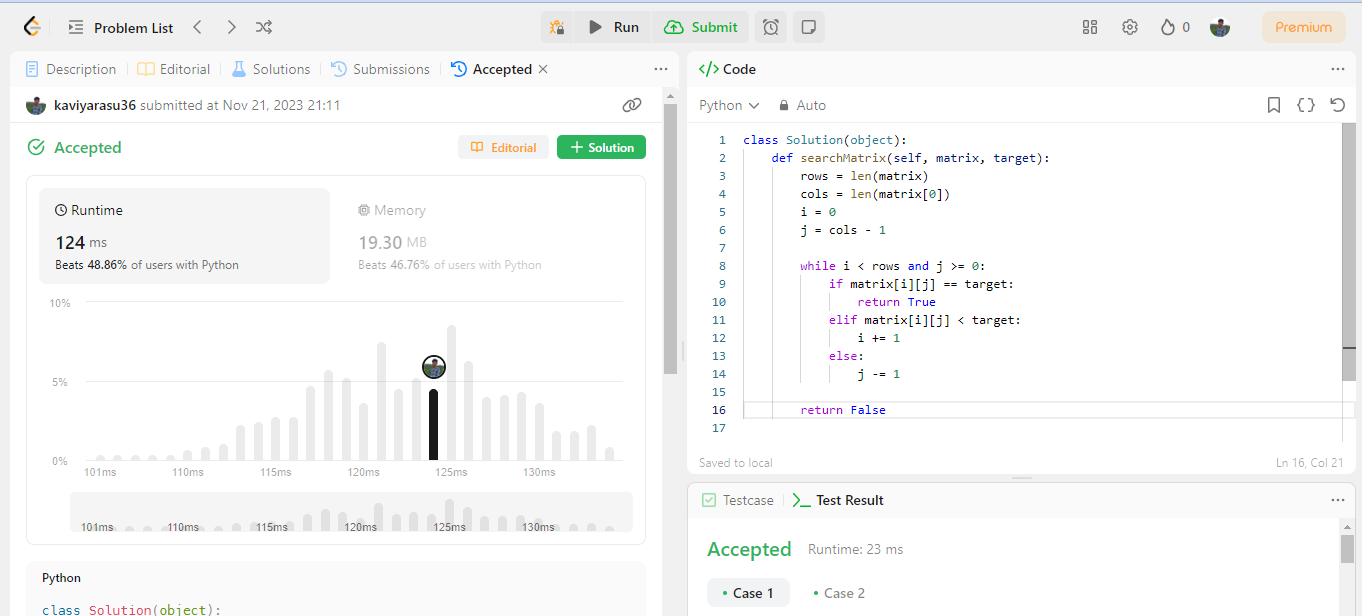
            elif matrix[i][j] < target:

                i += 1

            else:

                j -= 1

        return False



TIME COMPLEXITY: O(n) SPACE COMPLEXITY: O(1)

717822i124-Kaviyarasu.c

# Program:7 Matrix diagonal sum

# Code:

class Solution(object):

    def diagonalSum(self, mat):

        n = len(mat)

        total\_sum = 0

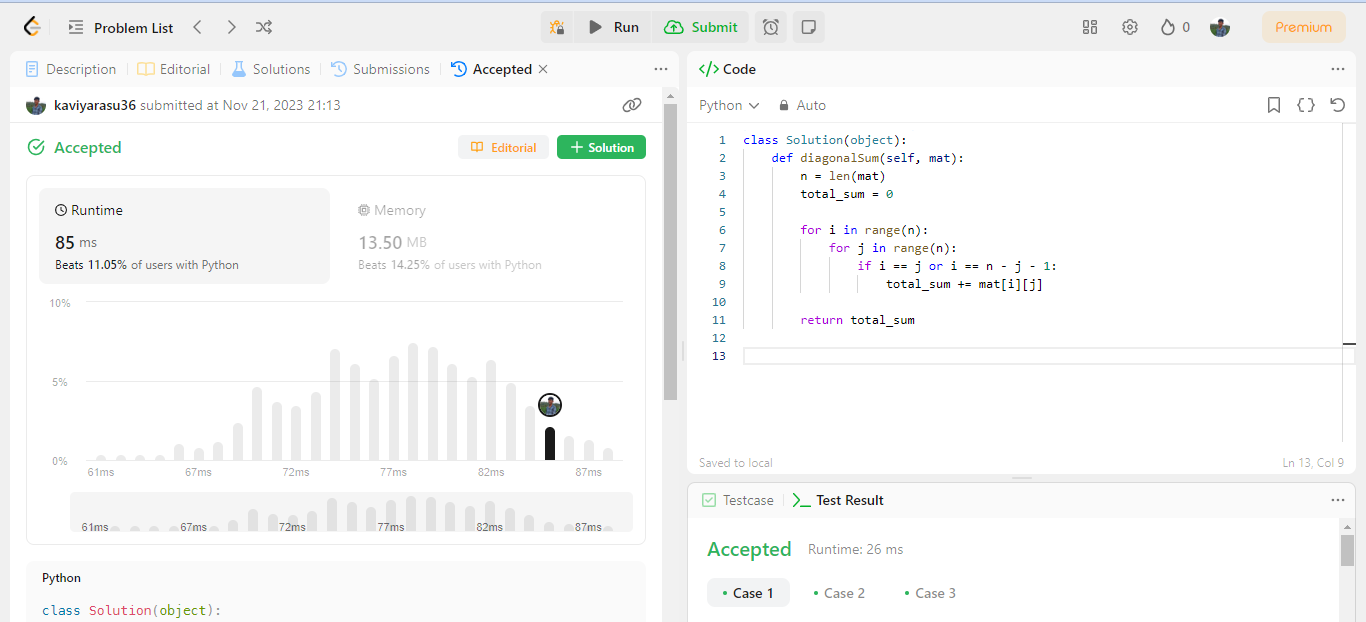
        for i in range(n):

            for j in range(n):

                if i == j or i == n - j - 1:

                    total\_sum += mat[i][j]

        return total\_sum



TIMECOMPLEXITY:O(n^2) SPACE COMPLEXITY: O(1)

# Program:8 Rotate image Code:

class Solution(object):

    def rotate(self, matrix):

        n = len(matrix)

        for i in range(n):

            for j in range(i + 1, n):

                matrix[i][j], matrix[j][i] = matrix[j][i], matrix[i][j]

        for i in range(n):

            left, right = 0, n - 1

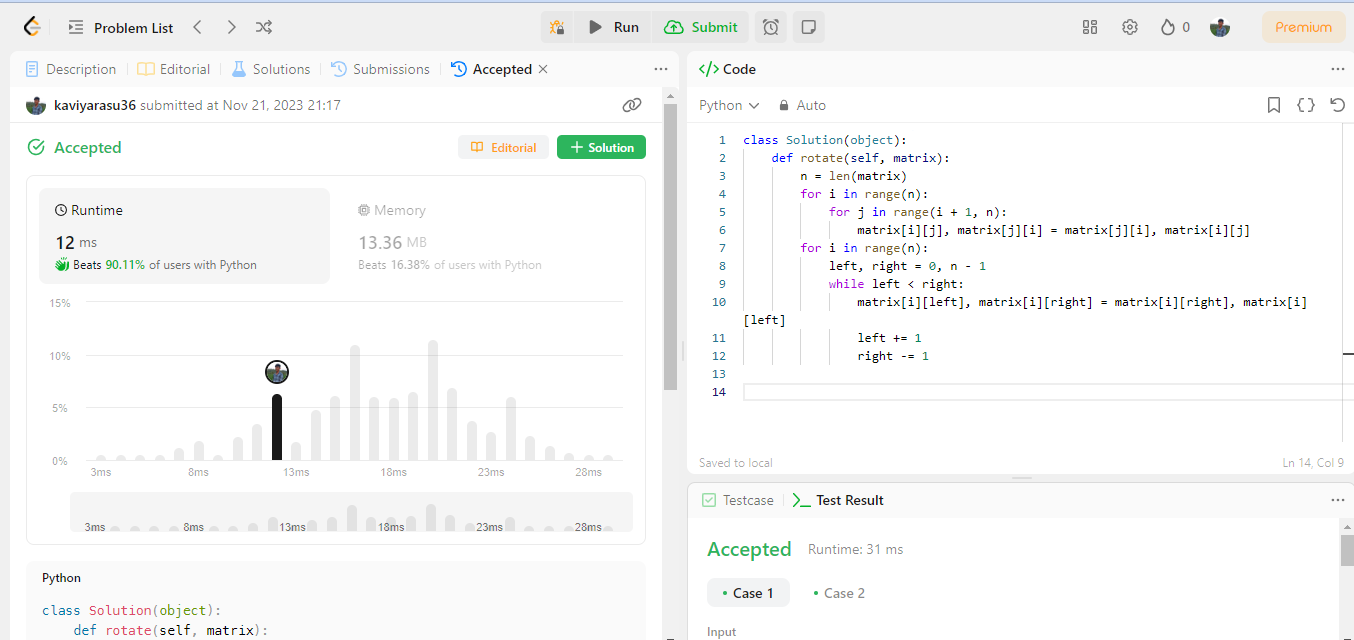
            while left < right:

                matrix[i][left], matrix[i][right] = matrix[i][right], matrix[i]  [left]

                left += 1

                right -= 1

717822i124-Kaviyarasu.c



TIMECOMPLEXITY:O(n^2) SPACE COMPLEXITY: O(1)

# Program:9 Prime in diagonal

# Code:

import math

class Solution(object):

    def isprime(self, n):

        if n <= 1:

            return False

        for i in range(2, int(math.sqrt(n)) + 1):

            if n % i == 0:

                return False

        return True

    def diagonalPrime(self, nums):

        l, ll = [], []

        n = len(nums)

        for i in range(n):

            for j in range(n):

                if i == j or i == n - j - 1:

                    l.append(nums[i][j])

        for i in range(len(l)):

            if self.isprime(l[i]):

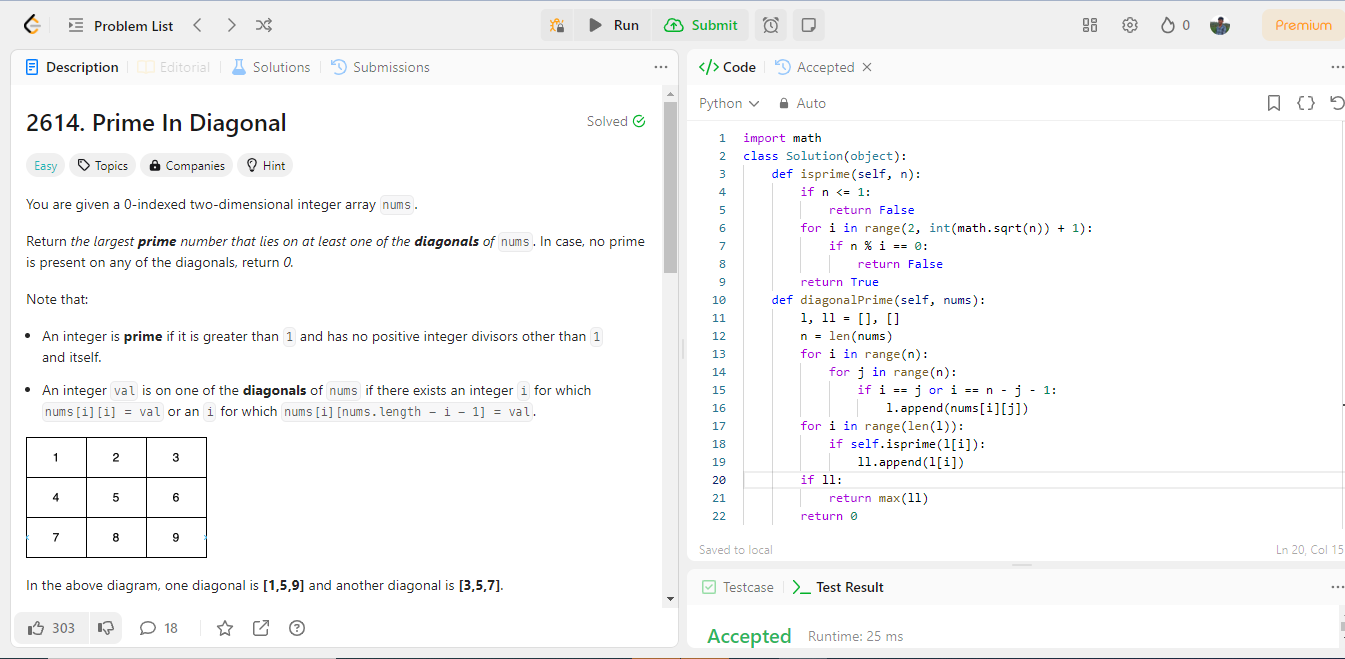
                ll.append(l[i])

        if ll:

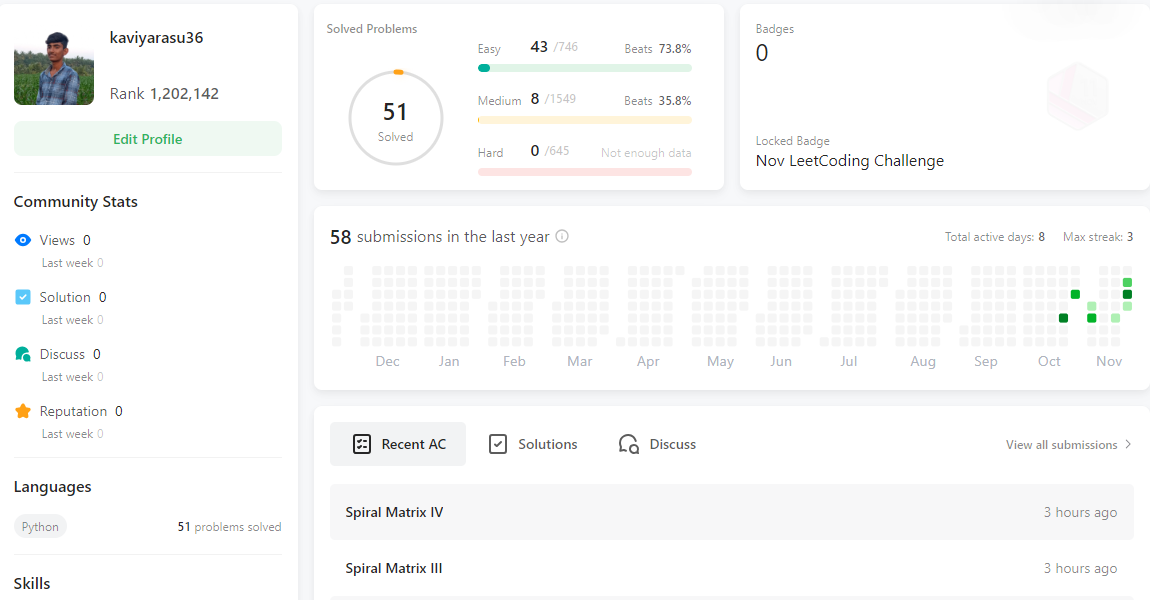
            return max(ll)

        return 0

717822i124-Kaviyarasu.c



TIMECOMPLEXITY:O(n^2) SPACE COMPLEXITY: O(n)



717822i124-Kaviyarasu.c

|  |  |  |
| --- | --- | --- |
| Preparation | 30 |  |
| Lab Performance | 30 |  |
| Report | 40 |  |
| Total | 100 |  |
| Initial of Faculty | |  |

# Result :

Thus the program on Two-Dimensional Array is executed successfully by implementing appropriate logics.

717822i124-Kaviyarasu.c