2-D MATRICES

Matrix Diagonal Sum: https://leetcode.com/problems/matrix-diagonal-sum/

```
class Solution {
public int diagonalSum(int[][] mat) {
int verticalSum=0;
for(int i=0; i<mat.length; i++){
verticalSum+=mat[i][i];
}
int count=0;
int horizontalSum=0;
for(int j=mat[0].length-1; j>=0; j--){
if(mat[0].length/2==count && mat[0].length%2!=0){
count++;
continue;
}
horizontalSum+=mat[count][j];
count++;
}
return (verticalSum+horizontalSum);
}
}
```

Rotate Image :https://leetcode.com/problems/rotate-image/

```
class Solution {
public void rotate(int[][] matrix) {
int n = matrix.length;
//Column Reverse
for(int i = 0; i < n; i++){
int a = 0;
int b = n-1;
while(a <= b){
int temp = matrix[a][i];
matrix[a][i] = matrix[b][i];
matrix[b][i] = temp;
a++;
b--;
}
}
//Transpose
```

```
for(int i = 0; i < n; i++){
for(int j =i+1; j< n; j++){
int temp = matrix[i][j];
matrix[i][j] = matrix[j][i];;
matrix[j][i] = temp;
}
}
}
}
```

59. Spiral Matrix II : https://leetcode.com/problems/spiral-matrix-ii/

```
class Solution {

    public int[][] generateMatrix(int n) {

        int[][] ans = new int[n][n];

        //right -> bottom -> left -> top

        int top = 0, left = 0;

        int bottom = n-1, right = n-1;

        int k = 1;

        while(top<=bottom && left<=right){

            //right

            for(int i=left;i<=right;i++){

                ans[top][i] = k++ ;

            }

            top++;
```

```
//bottom
for(int i=top;i<=bottom;i++){
    ans[i][right] = k++;
}
right--;
if(top<=bottom){
    //left
    for(int i=right;i>=left;i--){
        ans[bottom][i] = k++;
     }
    bottom--;
}
if(left<=right){
    //top
    for(int i=bottom;i>=top;i--){
        ans[i][left] = k++;
    }
    left++;
```

```
        }


    }


    return ans;


  }

}
```

867. Transpose Matrix : https://leetcode.com/problems/transpose-matrix/

```
class Solution {
public int[][] transpose(int[][] matrix) {
int[][] answer = new int[matrix[0].length][matrix.length];
for(int i=0; i < matrix.length; i++){
for (int j = 0; j < matrix[0].length; j++){
answer[j][i] = matrix[i][j];
}
}
return answer;
}
}
```

73. Set Matrix Zeroes : https://leetcode.com/problems/set-matrix-zeroes/

```
public class Solution {

public void setZeroes(int[][] matrix) {

    boolean fr = false,fc = false;

    for(int i = 0; i < matrix.length; i++) {

        for(int j = 0; j < matrix[0].length; j++) {
```

```
        if(matrix[i][j] == 0) {
            if(i == 0) fr = true;
            if(j == 0) fc = true;
            matrix[0][j] = 0;
            matrix[i][0] = 0;
        }
    }
}
for(int i = 1; i < matrix.length; i++) {
    for(int j = 1; j < matrix[0].length; j++) {
        if(matrix[i][0] == 0 || matrix[0][j] == 0) {
            matrix[i][j] = 0;
    }}
}
if(fr) {
    for(int j = 0; j < matrix[0].length; j++) {
        matrix[0][j] = 0;
    }
```

```
    }
    if(fc) {
        for(int i = 0; i < matrix.length; i++) {
            matrix[i][0] = 0;
        }
    }
}}
```

1424. Diagonal Traverse II : https://leetcode.com/problems/diagonal-traverse-ii/

```
class Solution {
    public int[] findDiagonalOrder(List<List<Integer>> nums) {
        int m = nums.size(), maxSum = 0, size = 0, index = 0;
        List<Integer>[] map = new ArrayList[100001];
        for (int i = 0; i < m; i++) {
            size += nums.get(i).size();
            for (int j = 0; j < nums.get(i).size(); j++) {
                int sum = i + j;
                if (map[sum] == null) map[sum] = new ArrayList<>();
                map[sum].add(nums.get(i).get(j));
```

```
                maxSum = Math.max(maxSum, sum);
            }
        }
        int[] res = new int[size];
        for (int i = 0; i <= maxSum; i++) {
            List<Integer> cur = map[i];
            for (int j = cur.size() - 1; j >= 0; j--) {
                res[index++] = cur.get(j);
            }
        }
        return res;
    }
}
```

Anti Diagonals:

https://www.interviewbit.com/problems/anti-diagonals/

```
int ** diagonal(int** a, int row, int col, int *length) {
    *length = 2*row -1 ;
    int ** res = malloc(sizeof(int *)*(*length));
    int i,j,k,x,len ,count = 0 ;
    for(len=1;len<=row;len++)
```

```
    {
        i=0;
        j=len-1 ;
        res[count] = malloc(sizeof(int)*(len+1));
        res[count][0] = len ;
        while(j>=0)
        {
            res[count][i+1] = a[i][j] ;
            i++ ;
            j--;
        }
        count++ ;
    }
    for(len-=2,k=1;len>0;len--,k++)
    {
        j= row-1 ;
        i= k ;
        res[count] = malloc(sizeof(int)*(len+1));
        res[count][0] = len ;
        x=1 ;
        while(i<row)
        {
            res[count][x++]=a[i][j] ;
            i++;
            j--;
        }
        count++ ;
    }

    return res;
}
```

You are given two matrices A & B of same size, you have to return another matrix which is the sum of A and B.**Note:** Matrices are of same size means the number of **rows** and number of **columns** of both matrices are equal.

```
// Java implementation to concatenate
// two matrices of size N x M
```

```
class  con{

static final int M = 2;
static final int N = 2;

// Function to concatenate two
// matrices A[][] and B[][]
static void mergeMatrix(int A[][],
int B[][])
{

// Matrix to store
// the result
int [][]C = new int[M][2 * N];

// Merge the two matrices
for(int i = 0; i < M; i++)
{
for(int j = 0; j < N; j++)
{
// To store elements
// of matrix A
C[i][j] = A[i][j];
// To store elements
// of matrix B
C[i][j + N] = B[i][j];
}
}

// Print the result
for(int i = 0; i < M; i++)
{
for(int j = 0; j < 2 * N; j++)
System.out.print(C[i][j] + " ");

System.out.println();
```

```
}
}

// Driven Code
public static void main(String[] args)
{
int A[][] = { { 1, 2 },
{ 3, 4 } };

int B[][] = { { 5, 6 },
{ 7, 8 } };

// Find the merge of
// the 2 matrices
mergeMatrix(A, B);
}
}
```

You are given two integer matrices **A** and **B** having same size(Both having same number of rows (**N**) and columns (**M**). You have to subtract matrix **B** from **A** and return the resultant matrix. (i.e. return the matrix **A - B**).
If **A** and **B** are two matrices of the same order (same dimensions). Then **A - B** is a matrix of the same order as **A** and **B** and its elements are obtained by doing an element wise subtraction of **A** from **B**.

```
// Java implementation to concatenate
// two matrices of size N x M
class con{

static final int M = 2;
static final int N = 2;
```

```
// Function to concatenate two
// matrices A[][] and B[][]
static void mergeMatrix(int A[][],
int B[][])
{

// Matrix to store
// the result
int [][]C = new int[M][2 * N];

// Merge the two matrices
for(int i = 0; i < M; i++)
{
for(int j = 0; j < N; j++)
{
// To store elements
// of matrix A
C[i][j] = A[i][j];
// To store elements
// of matrix B
C[i][j + N] = B[i][j];
}
}

// Print the result
for(int i = 0; i < M; i++)
{
for(int j = 0; j < 2 * N; j++)
System.out.print(C[i][j] + " ");

System.out.println();
}
}

// Driven Code
public static void main(String[] args)
```

```
{
int A[][] = { { 1, 2 },
{ 3, 4 } };

int B[][] = { { 5, 6 },
{ 7, 8 } };

// Find the merge of
// the 2 matrices
mergeMatrix(A, B);
}
}
```

You are given a 2D integer matrix A, return a 1D integer array containing column-wise sums of original matrix.

```
// Java program to find the sum
// of each row and column of a matrix

import java.io.*;

class sum{

// Get the size m and n
static int m = 4;
static int n = 4;

// Function to calculate sum of each row
static void row_sum(int arr[][])
{

int i, j, sum = 0;

System.out.print("\nFinding Sum of each row:\n\n");

// finding the row sum
for (i = 0; i < m; ++i) {
```

```
for (j = 0; j < n; ++j) {

// Add the element
sum = sum + arr[i][j];
}

// Print the row sum
System.out.println("Sum of the row " + i + " = "
+ sum);

// Reset the sum
sum = 0;
}
}

// Function to calculate sum of each column
static void column_sum(int arr[][])
{

int i, j, sum = 0;

System.out.print(
"\nFinding Sum of each column:\n\n");

// finding the column sum
for (i = 0; i < m; ++i) {
for (j = 0; j < n; ++j) {

// Add the element
sum = sum + arr[j][i];
}

// Print the column sum
System.out.println("Sum of the column " + i
+ " = " + sum);
```

```
// Reset the sum
sum = 0;
}
}

// Driver code
public static void main(String[] args)
{
int i, j;
int[][] arr = new int[m][n];

// Get the matrix elements
int x = 1;
for (i = 0; i < m; i++)
for (j = 0; j < n; j++)
arr[i][j] = x++;

// Get each row sum
row_sum(arr);

// Get each column sum
column_sum(arr);
}
}
```

Problem Description You are given a 2D integer matrix A, return a 1D integer array containing row-wise sums of original matrix.

```
class Solution {
public int[][] transpose(int[][] matrix) {
int[][] answer = new int[matrix[0].length][matrix.length];
for(int i=0; i < matrix.length; i++){
for (int j = 0; j < matrix[0].length; j++){
answer[j][i] = matrix[i][j];
}
}
return answer;
}
}
```

You are given a **N X N**integer matrix. You have to find the sum of all the main diagonal elements of **A**.
Main diagonal of a matrix **A**is a collection of elements **A[i, j]**such that **i = j**.

```
// Java program to find the longest path in a matrix
// with given constraints

class long{
public static int n = 3;

// Function that returns length of the longest path
// beginning with mat[i][j]
// This function mainly uses lookup table dp[n][n]
static int findLongestFromACell(int i, int j,
int mat[][], int dp[][])
{
// Base case
if (i < 0 || i >= n || j < 0 || j >= n)
return 0;

// If this subproblem is already solved
if (dp[i][j] != -1)
return dp[i][j];

// To store the path lengths in all the four
// directions
int x = Integer.MIN_VALUE, y = Integer.MIN_VALUE,
z = Integer.MIN_VALUE, w = Integer.MIN_VALUE;
// Since all numbers are unique and in range from 1
// to n*n, there is atmost one possible direction
// from any cell
if (j < n - 1 && ((mat[i][j] + 1) == mat[i][j + 1]))
x = dp[i][j]
= 1
+ findLongestFromACell(i, j + 1, mat, dp);

if (j > 0 && (mat[i][j] + 1 == mat[i][j - 1]))
```

```
y = dp[i][j]
= 1
+ findLongestFromACell(i, j - 1, mat, dp);

if (i > 0 && (mat[i][j] + 1 == mat[i - 1][j]))
z = dp[i][j]
= 1
+ findLongestFromACell(i - 1, j, mat, dp);

if (i < n - 1 && (mat[i][j] + 1 == mat[i + 1][j]))
w = dp[i][j]
= 1
+ findLongestFromACell(i + 1, j, mat, dp);

// If none of the adjacent fours is one greater we
// will take 1 otherwise we will pick maximum from
// all the four directions
return dp[i][j]
= Math.max(
x,
Math.max(y, Math.max(z, Math.max(w, 1))));
}

// Function that returns length of the longest path
// beginning with any cell
static int finLongestOverAll(int mat[][])
{
// Initialize result
int result = 1;

// Create a lookup table and fill all entries in it
// as -1
int[][] dp = new int[n][n];
for (int i = 0; i < n; i++)
for (int j = 0; j < n; j++)
dp[i][j] = -1;
```

```
// Compute longest path beginning from all cells
for (int i = 0; i < n; i++) {
for (int j = 0; j < n; j++) {
if (dp[i][j] == -1)
findLongestFromACell(i, j, mat, dp);

// Update result if needed
result = Math.max(result, dp[i][j]);
}
}

return result;
}

// driver program
public static void main(String[] args)
{
int mat[][]
= { { 1, 2, 9 }, { 5, 3, 8 }, { 4, 6, 7 } };
System.out.println("Length of the longest path is "
+ finLongestOverAll(mat));
}
}
```