Day 44/180 2D arrays Interview Problems

1:Transpose of Matrix: (Use lower triangle for Transpose)
Solution:

```
void transpose(vector<vector<int> >& a, int n) {
    // Loop through the rows of the square matrix
    for (int row = 0; row < n; row++) {
        // Loop through the columns up to the current row
        for (int col = 0; col < row; col++) {
            // Swap elements at (row, col) and (col, row) to
transpose them
            swap(a[row][col], a[col][row]);
        }
    }
}
```

2: Given a Matrix of size n*m (n=rows and m= cols). Reverse each column of the matrix.

Example:

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

Solution

```cpp
#include <iostream>
using namespace std;

int main() {
    int n, m;

    // Get the dimensions of the matrix
    cout << "Enter the number of rows (n): ";
    cin >> n;
    cout << "Enter the number of columns (m): ";
    cin >> m;

    // Initialize the matrix
    int matrix[n][m];

    // Input the elements of the matrix
    cout << "Enter the elements of the matrix:" << endl;
    for (int row = 0; row < n; row++) {
        for (int col = 0; col < m; col++) {
            cin >> matrix[row][col];
        }
    }

    // Reverse each column of the matrix
    for (int col = 0; col < m; col++) {
        int top = 0;
        int bottom = n - 1;

        while (top < bottom) {
            // Swap elements from top and bottom
            int temp = matrix[top][col];
            matrix[top][col] = matrix[bottom][col];
            matrix[bottom][col] = temp;

            top++;
            bottom--;
        }
    }
```

```
    // Print the matrix with reversed columns
    cout << "Matrix with reversed columns:" << endl;
    for (int row = 0; row < n; row++) {
        for (int col = 0; col < m; col++) {
            cout << matrix[row][col] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

3: Spiral Matrix:

Solution:

```
vector<int> spiralOrder(vector<vector<int>>& matrix) {
    vector<int> result;  // The vector to store the elements in spiral order

    // Check if the matrix is empty or contains empty rows
    if (matrix.empty() || matrix[0].empty()) {
        return result;  // If empty, return an empty vector
    }

    int rows = matrix.size();    // Number of rows in the matrix
    int cols = matrix[0].size();  // Number of columns in the matrix

    int left = 0;    // Initialize the leftmost column
    int right = cols - 1;  // Initialize the rightmost column
    int top = 0;     // Initialize the topmost row
    int bottom = rows - 1;  // Initialize the bottom row
```

```
    // Traverse the matrix in a spiral order
    while (left <= right && top <= bottom) {
        // Traverse from left to right along the top row
        for (int i = left; i <= right; i++) {
            result.push_back(matrix[top][i]);
        }
        top++;

        // Traverse from top to bottom along the rightmost column
        for (int i = top; i <= bottom; i++) {
            result.push_back(matrix[i][right]);
        }
        right--;

        // Traverse from right to left along the bottom row, if it exists
        if (top <= bottom) {
            for (int i = right; i >= left; i--) {
                result.push_back(matrix[bottom][i]);
            }
            bottom--;
        }

        // Traverse from bottom to top along the leftmost column, if it exists
        if (left <= right) {
            for (int i = bottom; i >= top; i--) {
                result.push_back(matrix[i][left]);
            }
            left++;
        }
    }

    return result;  // Return the 1D vector containing elements in spiral order
}
```

4: [Spiral Matrix II:]

```
vector<vector<int>> generateMatrix(int n) {
    int rows = n;   // Number of rows in the matrix
    int cols = n;  // Number of columns in the matrix
```

```
    // Create a matrix of size n x n filled with zeros
    vector<vector<int>> result(n, vector<int>(n));

    int left = 0;   // Initialize the leftmost column
    int right = cols - 1;  // Initialize the rightmost column
    int top = 0;    // Initialize the topmost row
    int bottom = rows - 1;  // Initialize the bottom row
    int cur = 1;  // Initialize the current value to be filled in the matrix

    // Traverse the matrix in a spiral order
    while (left <= right && top <= bottom) {
        // Traverse from left to right along the top row
        for (int i = left; i <= right; i++) {
            result[top][i] = cur++;  // Fill the current value and increment
        }
        top++;
        // Traverse from top to bottom along the rightmost column
        for (int i = top; i <= bottom; i++) {
            result[i][right] = cur++;  // Fill the current value and increment
        }
        right--;
        // Traverse from right to left along the bottom row, if it exists
        if (top <= bottom) {
            for (int i = right; i >= left; i--) {
                result[bottom][i] = cur++;  // Fill the current value and increment
            }
            bottom--;
        }
        // Traverse from bottom to top along the leftmost column, if it exists
        if (left <= right) {
            for (int i = bottom; i >= top; i--) {
                result[i][left] = cur++;  // Fill the current value and increment
            }
            left++;
        }
    }
    return result;  // Return the filled matrix in a spiral order
}
```

5: Print Diagonally

```
vector<int> downwardDiagonal(int N, vector<vector<int>> A) {
    vector<int> res;  // Initialize a vector to store the elements of the downward diagonal

    // Loop to traverse the diagonals starting from the first row (top to bottom)
    for (int row = 0; row < N; row++) {
        int l = 0;  // Initialize the leftmost column
        int r = row;  // Initialize the row
        while (l < N && r >= 0) {
            res.push_back(A[l][r]);  // Add the element to the result vector
            l++;  // Move to the next column (right)
            r--;  // Move to the previous row (down)
        }
    }

    // Loop to traverse the diagonals starting from the second column (left to right)
    for (int col = 1; col < N; col++) {
        int l = col;  // Initialize the column
        int r = N - 1;  // Initialize the bottom row
        while (l < N && r >= 0) {
            res.push_back(A[l][r]);  // Add the element to the result vector
            l++;  // Move to the next column (right)
            r--;  // Move to the previous row (down)
        }
    }

    return res;  // Return the result vector containing the elements of the downward diagonal
}
```

6: Print matrix in diagonal pattern (Hard level)

```cpp
vector<int> matrixDiagonally(vector<vector<int>>& mat) {
    int row = 0;  // Initialize the starting row
    int col = 0;  // Initialize the starting column
    int n = mat.size();  // Get the size of the square matrix

    bool isUp = true;  // Initialize the direction flag (upwards or downwards)
    vector<int> ans;  // Initialize a vector to store the diagonal elements

    // Traverse the matrix till all elements get traversed
    for (int k = 0; k < n * n;) {
        // If isUp = true, then traverse from downward to upward
        if (isUp) {
            for (; row >= 0 && col < n; col++, row--) {
                ans.push_back(mat[row][col]);  // Add the element to the result vector
                k++;
            }
            // Set row and col according to the direction
            if (row < 0 && col <= n - 1)
                row = 0;
            if (col == n)
                row = row + 2, col--;
        } else {
            // If isUp = 0, then traverse up to down
            for (; col >= 0 && row < n; row++, col--) {
                ans.push_back(mat[row][col]);  // Add the element to the result vector
                k++;
            }
            // Set row and col according to the direction
            if (col < 0 && row <= n - 1)
                col = 0;
            if (row == n)
                col = col + 2, row--;
        }
        // Revert the isUp flag to change the direction
        isUp = !isUp;
    }

    return ans;  // Return the result vector containing the diagonal elements
}
```

7: [Print Matrix in snake Patter]

```
vector<int> spiralOrder(vector<vector<int>>& matrix) {
    vector<int> result;  // Initialize a vector to store the elements in spiral
order

    for (int row = 0; row < matrix.size(); row++) {
        // For even rows, traverse from left to right (0 to n-1)
        if (row % 2 == 0) {
            for (int col = 0; col < matrix.size(); col++) {
                result.push_back(matrix[row][col]);  // Add the element to the
result vector
            }
        } else {
            // For odd rows, traverse from right to left (n-1 to 0)
            for (int col = matrix.size() - 1; col >= 0; col--) {
                result.push_back(matrix[row][col]);  // Add the element to the
result vector
            }
        }
    }

    return result;  // Return the result vector containing elements in a zigzag
pattern
}
```