Matrix (2D Array) Mastery Guide in C++ for DSA, CP & FAANG

📖Chapter 1: Introduction to Matrix

What is a Matrix?

A matrix is a 2D array (array of arrays) where elements are arranged in rows and columns. It is useful for grid-based problems like pathfinding, games, image processing, and DP.

Real-world Analogy

Think of a matrix like an Excel sheet or chessboard where cells store individual data points.

Declaration Syntax

```
int matrix[rows][cols]; // Static allocation
vector<vector<int>> matrix(rows, vector<int>(cols)); // Dynamic using STL
```

🛠️Chapter 2: CRUD Operations on Matrix

Create

```
int mat[3][3] = {{1,2,3}, {4,5,6}, {7,8,9}}; // Static
vector<vector<int>> mat(3, vector<int>(3, 0)); // Dynamic with 0s
```

Read / Access

```
cout << mat[i][j];
```

Update

```
mat[i][j] = newVal;
```

Delete

In static arrays, not possible. In dynamic:

```
mat.clear();
```

🧰Chapter 3: Important Built-in STL Functions

```
matrix.size();         // No. of rows
matrix[0].size();      // No. of columns
reverse(matrix[i].begin(), matrix[i].end()); // Reverse a row
sort(matrix.begin(), matrix.end()); // Sort rows
```

🔍Chapter 4: Common Patterns in Matrix

- **Row-wise traversal:**

```
for(int i=0; i<rows; i++)
  for(int j=0; j<cols; j++)
    cout << matrix[i][j];
```

- **Column-wise traversal**
- **Spiral traversal**
- **Diagonal traversal**
- **Transpose**
- **Rotate 90/180 degrees**
- **Prefix Sum Matrix**

💡 Chapter 5: Pro Tips & Strategies

Matrix Traversal Tips

- Always validate boundaries: `i >= 0 && i < rows && j >= 0 && j < cols`
- Use direction vectors for 4/8-direction movement

```
int dx[] = {0, 0, 1, -1};
int dy[] = {1, -1, 0, 0};
```

Optimization

- Use `vector<vector<int>>` for dynamic resizing
- Use prefix sums for submatrix queries
- Use BFS/DFS on grids for connected components

Edge Cases

- Empty matrix
- Uneven rows (for jagged matrix)
- Matrix with all same elements

Chapter 6: Basic to Advanced Problem List (No Solutions)

1. Print matrix in row-wise & column-wise
2. Transpose of matrix
3. Rotate matrix 90 degrees clockwise
4. Print matrix in spiral form
5. Search in a row-column sorted matrix
6. Snake pattern print
7. Diagonal traversal
8. Matrix multiplication
9. Matrix determinant
10. Set entire row/col to 0 if element is 0
11. Valid Sudoku board
12. Max submatrix sum (Kadane 2D)
13. Matrix chain multiplication
14. Unique paths in grid
15. Number of islands (BFS)
16. Word Search
17. Path sum in matrix (DP)
18. Shortest path in binary matrix
19. Flood fill algorithm
20. Boolean Matrix Problem
21. Search in matrix with binary search
22. Boundary traversal
23. Rotate outer ring of matrix
24. Minimum cost path in matrix
25. Find maximum size square sub-matrix with all 1s

🔗 Chapter 7: 10 Solved Problems (with Concepts)

1. **Print Diagonal Elements**\ Learn traversal with `i == j` or `i + j == n - 1`

2. **Spiral Print**\ 4 pointers: `top, bottom, left, right`
3. **Rotate 90 Clockwise**\ Transpose + reverse each row
4. **Search in Sorted Matrix**\ Start from top-right, move left/down
5. **Set Matrix Zero**\ Use 1st row/col as storage
6. **Flood Fill Algorithm**\ DFS/BFS
7. **Matrix Multiplication**\ Triple nested loops
8. **Prefix Sum Matrix**\ Use DP-style cumulative sum
9. **Max Area of Island**\ DFS on 2D grid
10. **Find Word in Matrix**\ Backtracking + DFS

Quick Notes

- Use `visited` matrix to avoid re-visiting
- BFS often faster than DFS in grid problems
- For shortest path: use BFS (for unweighted), Dijkstra (for weighted)
- Avoid recursion in deep grid problems (stack overflow)
- Memorize movement patterns (4-dir, 8-dir)
- Always check edge boundaries before accessing index

📘 LeetCode Practice Set

- Spiral Matrix
- Search a 2D Matrix
- Set Matrix Zeroes
- Word Search
- Max Area of Island
- Shortest Path in Binary Matrix
- Flood Fill
- Number of Islands
- Rotate Image
- Unique Paths

Want to move to **Searching & Sorting** next or go deep into advanced matrix techniques like binary lifting in grids, or matrix exponentiation?