Lesson Plan

Questions on 2D Arrays

PW SKILLS

Pre-requisites:

- Basic knowledge of multidimensional array

Today's checklist

- Given two matrices, the task is to multiply them.
- Given a matrix mat[][], print it in Wave Form.
- Given a 2D array, print it in spiral form.

Q1. Given two matrices, the task is to multiply them.

Matrices can either be square or rectangular:

Examples:

(Square Matrix Multiplication)

```
Input: mat1[m][n] = { {1, 1}, {2, 2} }
mat2[n][p] = { {1, 1}, {2, 2} }
Output: result[m][p] = { {3, 3}, {6, 6} }
```

(Rectangular Matrix Multiplication)

```
Input: mat1[3][2] = { {1, 1}, {2, 2}, {3, 3} }
mat2[2][3] = { {1, 1, 1}, {2, 2, 2} }
Output: result[3][3] = { {3, 3, 3}, {6, 6, 6}, {9, 9, 9} }
```

Multiplication of two Square or Rectangular Matrices:

- The number of columns in Matrix-1 must be equal to the number of rows in Matrix-2.

- Output of multiplication of Matrix-1 and Matrix-2, results with equal to the number of rows of Matrix-1 and the number of columns of Matrix-2 i.e. rslt[R1][C2]

Below is the implementation of the multiplication of two matrices:

```
import java.io.*;
import java.util.*;

class Main{

    static int R1 = 2; // number of rows in Matrix-1
    static int C1 = 2; // number of columns in Matrix-1
    static int R2 = 2; // number of rows in Matrix-2
    static int C2 = 2; // number of columns in Matrix-2

    // This function multiplies mat1[][]
    // and mat2[][], and stores the result
    // in res[][]
    static void mulMat(int[][] mat1, int[][] mat2)
    {
        // To store result
        int[][] rslt = new int[R1][C2];
        System.out.println(
            "Multiplication of given two matrices is:");
        int i, j, k;
        for (i = 0; i < R1; i++) {
            for (j = 0; j < C2; j++) {
                rslt[i][j] = 0;
                for (k = 0; k < R2; k++)
                    rslt[i][j] += mat1[i][k] * mat2[k][j];
                System.out.print(rslt[i][j] + " ");
            }
            System.out.println("");
        }
    }
    public static void main(String[] args)
    {
        int[][] mat1 = { { 1, 1 },
                         { 2, 2 } };

        int[][] mat2 = { { 1, 1 },
                         { 2, 2 } };


        if (C1 != R2) {
            System.out.println(
                "The number of columns in Matrix-1  must be
equal to the number of rows in Matrix-2");
            System.out.println(
                "Please update the global variables
according to your array dimension");
```

```
        }
        else {

              // Function call
            mulMat(mat1, mat2);
        }
    }
}
```

Output:

Multiplication of given two matrices is:

3 3
6 6

Time complexity: O(R1 * C2 * R2) for given matrices mat1[R1][C1] and mat2[R2][C2]

Auxiliary space: O(R1 * C2)

Q2. Given a matrix mat[][], print it in Wave Form.

Input: mat[][] = {{ 1, 2, 3, 4}
{ 5, 6, 7, 8}
{ 9, 10, 11, 12}
{13, 14, 15, 16}
{17, 18, 19, 20}}

Output: 1 5 9 13 17 18 14 10 6 2 3 7 11 15 19 20 16 12 8 4

Explanation: Output is printed in wave form.

Input: mat[][] = {{1, 9, 4, 10}
{ 3, 6, 90, 11}
{ 2, 30, 85, 72}
{ 6, 31, 99, 15}}

Output: 1 3 2 6 31 30 6 9 4 90 85 99 15 72 11 10

Approach:
To get the desired waveform for a given matrix, first, print the elements of the first column of the matrix in the downward direction and then print the elements of the 2nd column in the upward direction, then print the elements in the third column in the downward direction and so on.

Below is the implementation of the above approach:

```
import java.io.*;
class Main{

  public static int R = 5;
  public static int C = 4;

  public static void WavePrint(int m, int n, int[][] arr)
  {

    // Loop to traverse matrix
    for (int j = 0; j < n; j++) {

      // If the current column
      // is even indexed, print
      // it in forward order
      if (j % 2 == 0) {
        for (int i = 0; i < m; i++) {
          System.out.print(arr[i][j] + " ");
        }
      }

      // If the current column
      // is odd indexed, print
      // it in reverse order
      else {
        for (int i = m - 1; i >= 0; i--) {
          System.out.print(arr[i][j] + " ");
        }
      }
    }
  }

  public static void main (String[] args)
  {
    int[][] arr = { { 1, 2, 3, 4 },
                    { 5, 6, 7, 8 },
                    { 9, 10, 11, 12 },
                    { 13, 14, 15, 16 },
                    { 17, 18, 19, 20 } };

    WavePrint(R, C, arr);
  }
}
```

Output:
1 5 9 13 17 18 14 10 6 2 3 7 11 15 19 20 16 12 8 4

Time Complexity: O(N^2)
Auxiliary Space: O(1)

Q3. Given a 2D array, print it in spiral form.

Examples:

Input: {{1, 2, 3, 4},
{5, 6, 7, 8},
{9, 10, 11, 12},
{13, 14, 15, 16 }}
Output: 1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10
Explanation: The output is matrix in spiral format.

Input: { {1, 2, 3, 4, 5, 6},
{7, 8, 9, 10, 11, 12},
{13, 14, 15, 16, 17, 18}}

Output: 1 2 3 4 5 6 12 18 17 16 15 14 13 7 8 9 10 11
Explanation: The output is matrix in spiral format.

Draw the path that the spiral makes. We know that the path should turn clockwise whenever it would go out of bounds or into a cell that was previously visited

Follow the given steps to solve the problem:

Let the array have R rows and C columns
seen[r] denotes that the cell on the r-th row and c-th column was previously visited. Our current position is (r, c), facing direction di, and we want to visit R x C total cells.

As we move through the matrix, our candidate's next position is (cr, cc).
If the candidate is in the bounds of the matrix and unseen, then it becomes our next position; otherwise, our next position is the one after performing a clockwise turn.

Below is the implementation of the above approach:

```
// Java program for the above approach
import java.util.*;
class Solution {
// Function to print in spiral order
public static List<Integer> spiralOrder(int[][] matrix)
{
List<Integer> ans = new ArrayList<Integer>();

if (matrix.length == 0)
return ans;
```

```
int m = matrix.length, n = matrix[0].length;
boolean[][] seen = new boolean[m][n];
int[] dr = { 0, 1, 0, -1 };
int[] dc = { 1, 0, -1, 0 };
int x = 0, y = 0, di = 0;
// Iterate from 0 to R * C - 1
for (int i = 0; i < m * n; i++) {
ans.add(matrix[x][y]);
seen[x][y] = true;
int cr = x + dr[di];
int cc = y + dc[di];
if (0 <= cr && cr < m && 0 <= cc && cc < n
&& !seen[cr][cc]) {
x = cr;
y = cc;
}
else {
di = (di + 1) % 4;
x += dr[di];
y += dc[di];
}
}
return ans;
}
// Driver Code
public static void main(String[] args)
{
int a[][] = { { 1, 2, 3, 4 },
{ 5, 6, 7, 8 },
{ 9, 10, 11, 12 },
{ 13, 14, 15, 16 } };
// Function call
System.out.println(spiralOrder(a));
}
}
```

Output: 1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10

Time Complexity: O(N), where N is the total number of elements in the input matrix. We add every element in the matrix to our final answer

Auxiliary Space: O(N), the information stored in seen and in ans.