CSE 220

Data Structures

Lecture 06:
2D Arrays

Anwarul Bashir Shuaib [AWBS]
Lecturer
Department of Computer Science and Engineering
BRAC University

1D Arrays

```
int[] x = new int[5];
```

```
{0, 0, 0, 0, 0}
```

```
int[] x = {1, 2, 3, 4, 5};
```

1D Arrays

```
int[] x = new int[5];
```

Creating an array

```
{0, 0, 0, 0, 0}
```

```
int[] x = {1, 2, 3, 4, 5};
```

1D Arrays

whether each element is an integer

```
int[] x = new int[5];
```

Creating an array

```
{0, 0, 0, 0, 0}
```

```
int[] x = {1, 2, 3, 4, 5};
```

1D Arrays

```
x = np.zeros(5, dtype=int)      {0, 0, 0, 0, 0}
```

```
x = np.array([1, 2, 3, 4, 5], dtype=np.int32)
```

1D Arrays

- Elements are stored consecutively in memory

Memory Address	Elements
0x100	x[0]
0x104	x[1]
0x108	x[2]
0x10C	x[3]
…	…

2D Arrays

```java
int[][] x = new int[3][3];
```

```python
x = np.zeros((3,3), dtype=np.int32)
```

	Column 0	Column 1	Column 2
Row 0	x[0][0]	x[0][1]	x[0][2]
Row 1	x[1][0]	x[1][1]	x[1][2]
Row 2	x[2][0]	x[2][1]	x[2][2]

2D Arrays

```
int[][] x = new int[3][3];
```

Creating an array

```
x = np.zeros((3,3), dtype=np.int32)
```

	Column 0	Column 1	Column 2
Row 0	x[0][0]	x[0][1]	x[0][2]
Row 1	x[1][0]	x[1][1]	x[1][2]
Row 2	x[2][0]	x[2][1]	x[2][2]

2D Arrays

Where each element is an integer array

```
int[][] x = new int[3][3];
```

Creating an array

```
x = np.zeros((3,3), dtype=np.int32)
```

	Column 0	Column 1	Column 2
Row 0	x[0][0]	x[0][1]	x[0][2]
Row 1	x[1][0]	x[1][1]	x[1][2]
Row 2	x[2][0]	x[2][1]	x[2][2]

2D Arrays

```
int[][] x = {
   {1, 2, 3},
   {4, 5, 6},
   {7, 8, 9}
}

x = np.array(
     [
        [1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]
     ], dtype=np.int32)
```

1	2	3
4	5	6
7	8	9

2D Arrays

x.length = # rows in the array

```
int[][] x = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
}
```

1	2	3
4	5	6
7	8	9

```
x = np.array(
    [
        [1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]
    ], dtype=np.int32)
```

2D Arrays

x[0].length = # columns in the array

```
int[][] x = {
   {1, 2, 3},
   {4, 5, 6},
   {7, 8, 9}
}

x = np.array(
     [
        [1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]
     ], dtype=np.int32)
```

1	2	3
4	5	6
7	8	9

2D Arrays

- Elements are still stored consecutively.
- Consider: `int[][] arr = new int[2][3];`

Memory Address	Element
0	arr[0][0]
1	arr[0][1]
2	arr[0][2]
3	arr[1][0]
4	arr[1][1]
5	arr[1][2]

arr[0][0]	arr[0][1]	arr[0][2]
arr[1][0]	arr[1][1]	arr[1][2]

2D Arrays

- Elements are still stored consecutively.
- Consider: `int[][] arr = new int[2][3];`

Memory Address	Element
0	`arr[0][0]`
1	`arr[0][1]`
2	`arr[0][2]`
3	`arr[1][0]`
4	`arr[1][1]`
5	`arr[1][2]`

`row = 2, col = 3`

`arr[0][0]`	`arr[0][1]`	`arr[0][2]`
`arr[1][0]`	`arr[1][1]`	`arr[1][2]`

2D Arrays

- Elements are still stored consecutively.
- Consider: `int[][] arr = new int[2][3];`

Memory Address	Element
0	arr[0][0]
1	arr[0][1]
2	arr[0][2]
3	arr[1][0]
4	arr[1][1]
5	arr[1][2]

row = 2, col = 3

arr[0][0]	arr[0][1]	arr[0][2]
arr[1][0]	arr[1][1]	arr[1][2]

arr[0][2] ⇒ 0 * col + 2 = $\boxed{2}$

2D Arrays

- What about finding row, column from linear index?

Memory Address	Element
0	arr[0][0]
1	arr[0][1]
2	arr[0][2]
3	arr[1][0]
4	arr[1][1]
5	arr[1][2]

Given linear index 5, find rowIndex and columnIndex.

2D Arrays

- What about finding row, column from linear index?

Memory Address	Element
0	`arr[0][0]`
1	`arr[0][1]`
2	`arr[0][2]`
3	`arr[1][0]`
4	`arr[1][1]`
5	`arr[1][2]`

Given linear index 5, find row and col.

Total columns in the original array

```
row = 5 / 3 = 1
col = 5 % 3 = 2
```

Summary

- `int[][] arr = new int[row][col];`
  - `arr[i][j]` located at address `i*col + j`
  - Given index N: `i = N / col, j = N % col`
- `int[][][] arr = new int[P][Q][R];`
  - `arr[i][j][k]` located at address `i*(Q*R) + j*R + k`
  - Also possible to calculate `i,j,k` from given index N (next slide)
- Linear indices can always map N-dimensional arrays to 1D arrays.

Summary

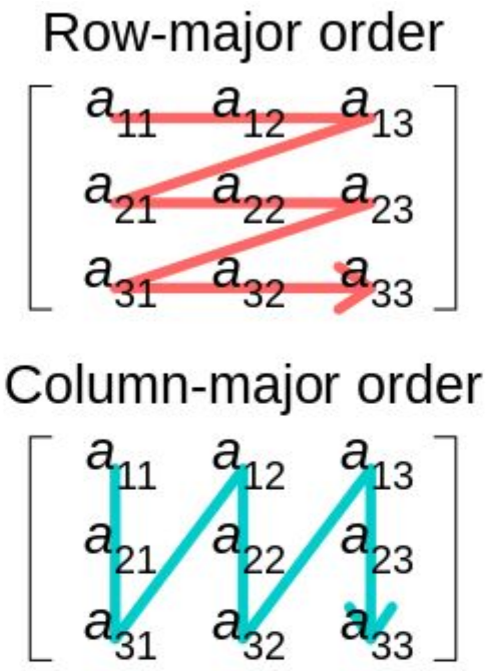
- $N = i*(Q*R) + \underbrace{\boxed{j*R + k}}_{}$

$$N = i*(Q*R) + \overbrace{\boxed{j*R + k}}^{N\%(Q*R)}$$

$$i = N/(Q*R)$$
$$j = (N\%(Q*R))/R$$
$$k = (N\%(Q*R))\%R$$

No need to memorize. Try to understand how the calculations work here.

Ordering



For example, the array

$$A = a_{y,x} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}$$

could be stored in two possible ways:

Address	Row-major order	Column-major order
0	a_{11}	a_{11}
1	a_{12}	a_{21}
2	a_{13}	a_{12}
3	a_{21}	a_{22}
4	a_{22}	a_{13}
5	a_{23}	a_{23}

Column-major order is rarely (almost never) used nowadays.

Exercises

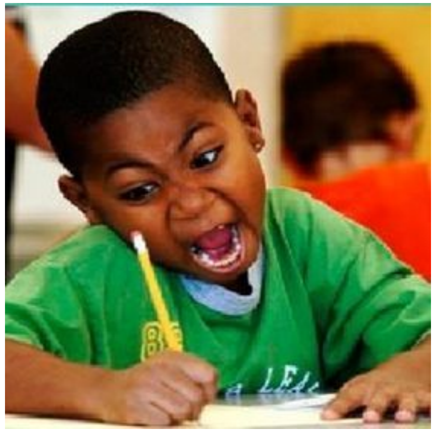
1. Create and Print a 2D Array
   a. Declare an `m x n` integer array.
   b. Fill it with values and print it row by row.
2. Find Maximum and Minimum Elements
3. Compute the sum of all elements in a 2D array.
4. Row-wise and Column-wise Sum
5. Transpose of a Matrix
   a. Swap `arr[i][j]` with `arr[j][i]` for a square matrix.

More Exercises

1. Check if a given number exists in the 2D array and return its position (both linear index and row, col)
2. Diagonal Sum (For Square Matrices)
   a. Compute the sum of the main diagonal `(arr[i][i])` and secondary diagonal `(arr[i][n-i-1])`.
3. Rotate 90 Degrees (Clockwise/Counterclockwise)
   a. Clockwise ⇒ transpose, then reverse rows.
4. Check Symmetry (Palindrome Matrix)
   a. Verify if a matrix is symmetric `(arr[i][j] == arr[j][i])`.

Even More Exercises!

1. Matrix multiplication
2. Determinant of an square matrix (try if you have time)
   a. 2x2, 3x3 is easy. For nxn, you will need recursion



Complete the function **compress_matrix** that takes a 2D array as a parameter and return a new compressed 2D array. In the given array the number of row and column will always be even. **Compressing a matrix means grouping elements in 2x2 blocks and sums the elements within each block. Check the sample input output for further clarification.**

Hint: Generally the block consists of the (i,j), (i+1,j), (i,j+1) and (i+1, j+1) elements for 2x2 blocks.

You cannot use any built-in function except len() and range(). You can use the np variable to create an array.

Python Notation	**Java Notation**

import numpy as np
def compress_matrix (mat):
        # To Do
	public int[ ][ ] compress_matrix (int[] [] mat) {
	        // To Do
	}

Sample Input array	**All Box (No need to create these arrays)**	**Returned Array**	**Explanation**
[[1, 2, 3, 4], [5, 6, 7, 8], [1, 3, 5, 2], [-2, 0, 6,-3]]	[[1, 2], [[3, 4], [5, 6]] [[7, 8]] [[1, 3], [[5, 2], [-2, 0]] [[6, -3]]	[[14, 22], [2, 10]]	[[1+2+5+6, 3+4+7+8], [1+3+-2+0, 5+2+6+-3]]