Introduction to matrices

- A matrix is a two dimensional rectangular grid of numbers:

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

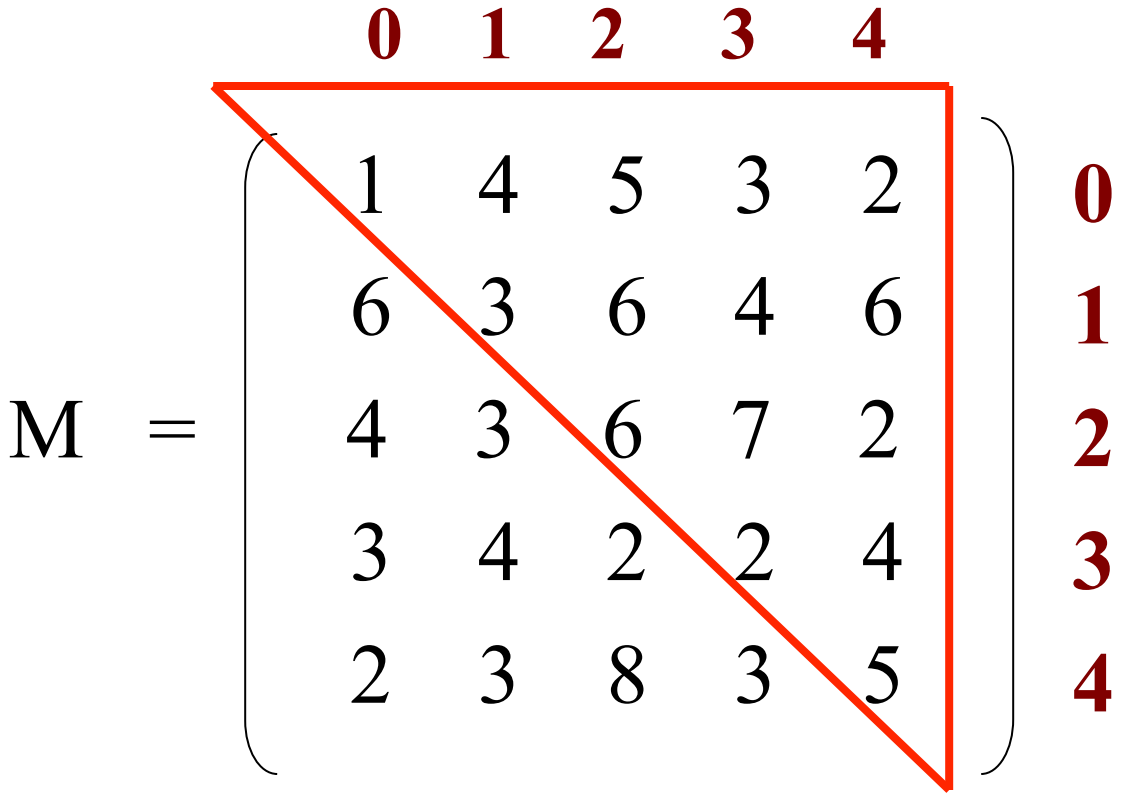
- The dimensions of the matrix are the numbers of rows and columns (in the above case: row dimension 3, column dimension 3).
- A value within a matrix is referred to by its row and column indices, in that order.
  - Math: number rows and columns from 1, from upper left corner
    - In math notation, $M_{1,2} = 2$
  - In Java, matrices are implemented via 2D Arrays and indices start from 0, as they do in (1D) arrays.
    - Thus, M[0][1] = 2

Matrix element processing

- To visit every element of an array, we had to use a loop.
- To visit every element of a matrix, we need to use a loop inside a loop:
  - Typically the outer loop goes through each row of a matrix
  - And the inner loop goes through each column within one row of a matrix.

Matrix example

Write a Java program that finds the sum of the upper triangle of a square matrix (i.e. the diagonal and up).



How do we know if an element of a square matrix is on or above the main diagonal?

row_index <= column_index

```java
import java.util.Scanner;

public class SumUpperTriangle{
   public static void main(String[] args) {
      Scanner input = new Scanner(System.in);

      System.out.print("Enter a 4 by 4 matrix row by row: ");
      double[][] m = new double[4][4];

      for (int i = 0; i < 4; i++){
         for (int j = 0; j < 4; j++){
            m[i][j] = input.nextDouble();
         }
      }

      System.out.print("The sum of the upper triangle of the given square matrix
   }

   public static double sumUppperTriangleMatrix(double[][] m) {
      double sum = 0;

      for (int row = 0; row < m.length; row++){
         for (int col = 0; col < m[0].length; col++){
            if(row<=col){
               sum += m[row][col];
            }
         }
      }
      return sum;
   }
}
```

Matrix element processing

Spend some time studying more examples of processing 2D-array from your textbook (included in this lab processing2DArrays.pdf

Exercise 1: Algebra - Matrix Transpose

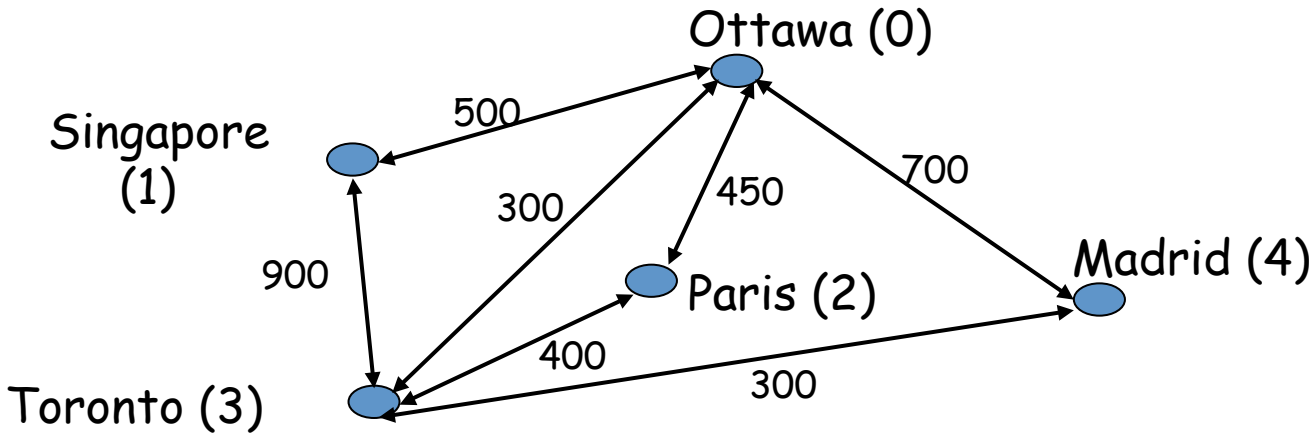
- Write an Java method, called transpose, that takes a matrix of integers A as input and transposes the matrix to produce a new matrix A^T. Write your solution inside clearly indicated space in TransposeStudents.java. The transpose of the matrix is such that element a_{rc} in the original matrix will be element a^T_{cr} in the transposed matrix. The number of rows in A becomes the number of columns in A^T, and the number of columns in A becomes the number of rows in A^T.

- For example:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \qquad A^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

Adjacency Matrix

- Escape Airlines has flights between certain cities. The flights and their costs can be represented as a graph in which an edge between city x and city y with a weight (label) of w means Escape Airlines has a flight between x and y costing w dollars.



Matrix Representation

- This graph can be represented with an adjacency matrix. There is a row and a column for each city, and cost[x][y] is the cost of a flight from x to y if one exists and is infinity (∞) if there is no such flight.

$$\text{cost} = \begin{bmatrix} 0 & 500 & 450 & 300 & 700 \\ 500 & 0 & \infty & 900 & \infty \\ 450 & \infty & 0 & 400 & \infty \\ 300 & 900 & 400 & 0 & 300 \\ 700 & \infty & \infty & 300 & 0 \end{bmatrix}$$

- Here, "infinity" is actually a very large number, greater than any number.
  - In Java: a predefined constant is available for the largest possible integer: **`Integer.MAX_VALUE`**

Exercise 2: Finding the Affordable Direct Flight

- Suppose you live in one of Escape's cities and have $d to spend. Write a Java method, called cheapDirectFlights that returns an array of the cities you can afford to fly to **directly**.
- What do you know (Givens: input your method)
  1. The city where you live.
  2. The cost of flight between two cities.
  3. The amount you can spend.
- What you want (Result: output of your method)
  - An array of cities that can be visited.
- Idea:
  - First, find the cities that can be visited.
  - Then, create an array of the right size.
  - Finally, place cities that can be visited in the array. Your method should return that array, unless no city can be visited in which case null should be returned

Use the provided Java program

- To solve the exercise, you are provided with a file called AdjacencyMatrixEscapeAirStudents.java. You should open this file work with it i.e. all your code for this exercise should go inside this file in the clearly indicated spaces.

Programs with more than one Class

- A program may have more than one class. If you save all classes in a program in one directory/folder, any class may call a **`public`** method in any other class in the same directory.

- When a (**`static`**) method is called from another class, use the name of the class with the dot operator.
  - For example, if we include class **`Library`** that has a method called **`aMethod`**, then another class that is in the same directory, for example one a class that contains your assignment, may call that method with call **`Library.aMethod( );`**

Library Classes

- Instead of putting all our methods in the same class as **`main`** (the class that contains our program) it is often better to separate them into coherent groups and put each group in a class of its own.

- These classes will not be programs - they have no **`main`** method. Each will be a small library of methods that can be used by other methods.

- Such classes can be compiled on their own but cannot be run as standalone programs.

Exercise 3: Using your own libraries

1. Open and study to two provided libraries: myPrintLibrary.java and mySearchSortLibrary

2. Write a short Java program that creates and array {5.0, 4.4, 1.9, 2.9, 3.4, 3.5} and then uses the methods in the two provided libraries to

- print the array,
- sort the array,
- print it again
- and finally look for an element in that array both with linear search and binary search.