CSCI 204 – Introduction to Computer Science II

Lab 4: Matrix ADT and its Applications

1. Objectives

In this lab, you will learn the following:

- Matrix abstract data type (ADT) and its operations,
- Reviewing concepts of streams (files) and operator overloading,
- Reading files and storing the information into a piece of given data structure, and
- Some matrix applications in social networks.

2. Commercial Applications and the Matrix ADT

Websites like Pandora and Netflix use your *likes* and *dislikes* of music and movies and the *likes* and *dislikes* of other users to suggest further music and movies that you might like. A simple way to find people with similar taste to yours is to use matrices. You will be implementing a class for such a matrix today.

As you have seen in class, data types such as Python string and Python list may be built "on top" of other data types (like arrays). We've provided a class for 2-D Arrays which is built on top of the Python list. You will use this 2-D Array to build a Matrix class.

A matrix is a 2-dimensional array with a set of special operations defined for the 2-D array. Matrix retains the basic features of an array such that the elements in the matrix can be accessed by a pair of indices. For example, if *m* is a matrix of integers of the size 4 by 4, then we can do operations such as `m[2,3] = 10` and `print(m[0,1])`, which are very similar (or identical) to a 2-D array. However because matrix has many special properties, we should also be able to apply operations such as matrix addition, subtraction, multiplication, transposition, and possibly other operations. All these operations are beyond a traditional array.

For example, in the case of Netflix, we can represent the relationship between the top *n* movies and a group of *m* students who liked these movies as a matrix. Because of the space limitation on the paper, let's set values for *m* and *n* to be 4 and 3. (The number three and four are chosen arbitrarily, they can be any number.)

According to IMDb (2017, Internet Movie Database, http://www.imdb.com/), the top 3 movies of all times are

1. The Shawshank Redemption (1994)
2. The Godfather (1972)
3. The Godfather: Part II (1974)

We assume the four students are named, Alice, Bastian, Catlin, and Diego.

Here is a summary of the information where a 1 means the student likes the movie and a 0 means he/she doesn't.

	The Shawshank Redemption	The Godfather	The Godfather: Part II
Alice	1	1	0
Bastian	0	0	1
Catlin	0	0	1
Diego	1	1	1

We want to use a matrix to represent this information and use this matrix to compute the common movies the students have seen. Of course, for a small sample like this, no computation is really needed, one can find an answer by just manually examining the values. But imagine hundreds of movies and millions of viewers, some computation is then needed to come up anything meaningful. The matrix *M* for four students and three movies could look as follows. (The same matrix as above: columns are movies, rows are students.)

$$M = \begin{matrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{matrix}$$

By convention, column and row counting starts from the top left corner and are indexed by [row, column] so matrix[0,0] == 3 and matrix[1,2] == 5 in the matrix below.

3		
		5

In the example matrix M, each row represents the movie a student liked. In order to compute the common movies these students liked, we need to transpose this matrix and get M'. To do this, M'[row, col] = M[col, row] and the rows of M become columns of M' and vice versa.

M =
students (4 rows) by movies (3 cols)

1	1	0
0	0	1
0	0	1
1	1	1

Transposed as

M^t =
movies (3 rows) by students (4 cols)

1	**0**	0	1
1	0	0	1
0	1	1	1

For example, the top left index of M^t is row 0, col 0
$M^t[0,0] = M[0,0] = 1$ as seen in bold above
$M^t[0,1] = M[1,0] = 0$ as seen in bold above

If we multiply the two matrices *M* and M^t we get the common movies each pair of students liked.

In order to multiply matrices, $R = M * M^t$, the matrix dimensions effect the results and are used in the computation.

M	*	M^t	=	R
n rows by k cols		k rows by p cols		n rows by p cols

so the number of columns (k) of the 1st matrix must be the same as the number of rows (k) of the 2nd matrix. The common size, k, will be used in the multiplication. The resulting matrix will have the number of rows (n) of the 1st matrix and the number of columns (p) of the 2nd matrix.

As a result of this particular multiplication, we would have

M * M^t = R

M: students (4 rows) by movies (3 cols)

1	1	0
0	0	1
0	0	1
1	1	1

M^t: movies (3 rows) by students (4 cols)

1	0	0	1
1	0	0	1
0	1	1	1

R: students (4 rows) by students (4 cols)

2	0	0	2
0	1	1	1
0	1	1	1
2	1	1	**3**

where each element in *R* is computed as follows.

R[row,col] = Sum for all values of k of M[row,k] * M^t[k,col])

For example, the bottom right index of R is row 3, column 3 (in bold above)
R[3,3] = Sum for k is 0..2 of M[3,k] * M^t[k,3])
= (M[3,0]*M^t[0,3]) + (M[3,1]*M^t[1,3]) + (M[3,2]*M^t[2,3])
= (1*1) + (1*1) + (1*1)
= 3

M * M^t = R

M

1	**1**	**1**

M^t

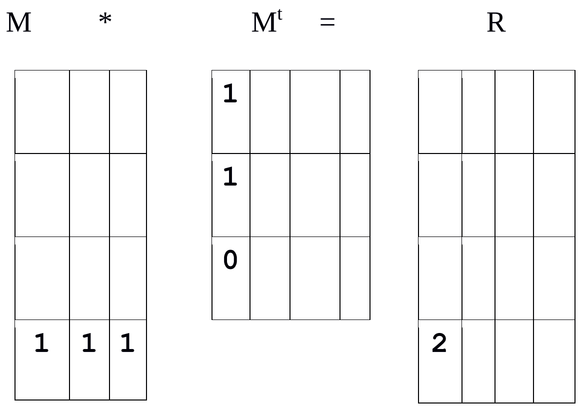
			1
			1
			1

R

			3

Another example, the bottom left index of R is row 3, column 0 (in bold above)

R[3,0] = Sum for k is 0..2 of M[3,k] * M^t[k,0])
= (M[3,0]*M'[0,0]) + (M[3,1]*M^t[1,0]) + (M[3,2]*M^t[2,0])
= (1*1) + (1*1) + (1*0)
= 2



The rows and columns of the students vs movies matrix after the multiplication (i.e., elements in *R*) now represent the common movies each pair of students liked.

	Alice	Bastian	Catlin	Diego
Alice	2	0	0	2
Bastian	0	1	1	1
Catlin	0	1	1	1
Diego	2	1	1	3

For example, the above matrix *R* represents the following facts: out of a total of three movies, Alice and Diego liked two common ones; Bastian and Catlin liked one common movie; and Catlin liked no common movies with Alice.

3. Getting started

Begin by copying lab files from the course Linux directory and examining them. When re-visiting this lab, you might also want to review the content about 2D array.

- `array204.py` contains completed classes `Array` and `Array2D`
- `matrix204.py` contains a partially implemented `Matrix` class
- `testarrays.py` contains codes that test the arrays and the matrix
- `moviegoers.py` contains a partially implemented class
- `input-data.txt` contains the information needed for testing the `MovieGoers` class
- `test.txt` contains the information needed for testing the `MovieGoers` class

Classes `Array` and `Array2D` are implemented in the file `array204.py`. The `array204.py` file also contains implementation of auxiliary `_ArrayIterator` class. The object of `_ArrayIterator` helps to move along the array (*iterate* over array). See how this class is implemented. In a future lecture session we'll have a more in-depth discussion of how iterators work.

4. Your work

In this lab, you are given a partially completed matrix class called `Matrix` in the file `matrix204.py`. This class contains defined data members, and some completed methods such as the constructor, access methods that return the number of rows and number of columns in the matrix, the method of `add()`, and overloaded operators which allow usage of [row, col] to access the matrix. Look at `testarray.py` to see how to create and access a Matrix object. Note that we've learned how operator overloading works in CSCI 203. This may be a good time to take a quick review.

You are to complete the following tasks.

a. Complete the method `__str__()` which generates and returns a string representation of the matrix object.

   The method *__str__ ()* generates and returns a string representation of the matrix object. If *a* is an matrix, *__str__()* makes a call to a print method such as `print(a)` prints the matrix in proper form;

b. Complete the method `transpose()` which returns a NEW transposed matrix. Do not modify the original!

   If *a* is a matrix, and *b* is the transposition of *a*, then for all elements in *b*, *b[i,j] = a[j,i]*.

c. Complete the `__mul__()` method that overloads the "*" operator. Again, this returns a new matrix.

   If *a*, *b*, and *r* are three matrices, *r = a * b* means that for all elements in *r*, *r[i, j] = sum over all k (a[i, k] * b[k, j])*;

d. Complete the `__sub__()` method that overloads the "-" operator. The method returns a new matrix.

   If *a*, *b*, and *r* are three matrices, *r = a – b* means that for all elements in *r*, *r[i, j] = a[i, j] – b[i, j]*, where *a[i, j]* and *b[i, j]* are elements in matrices *a* and *b*, respectively.

5. Test your work

Once you complete the four Matrix methods, you should be able to run the program `testarrays.py` which should generate the following result. (Only the matrix relevant parts are shown in the output file below.)

```
matrix element [1,2]:  0  (should be 0)
scaled matrix element [1,2]:  8  (should be 8)

original matrix:
0 1 2
1 2 11
```

```
transposed matrix:
0 1
1 2
2 11

matrix e = c X d :
5 24
24 126

matrix f = e + e :
10 48
48 252

matrix g = f - e :
5 24
24 126
```

If your program generates the above result, your implementation of the missing matrix methods are likely correct.

6. Completing the `MovieGoers` class in `moviegoers.py`

When Matrix class is completed, you then need to work on another partially completed class called *MovieGoers* in the file *moviegoers.py* to compute the common movies that a group of students have liked. The given *MovieGoers* class already contains all necessary data members (thus a part of the constructor). Some of the data members are arrays and others are matrices. Students need to complete the constructor, and three additional methods, two of which are used in the constructor and one is used to find the common movies among students.

Examine the file `moviegoers.py`, you will find an incomplete class `MovieGoers` and a `main()` method that test the `MovieGoers` class within the implementation file `moviegoers.py`. In the implementation file, you will find that the data members of the class `MovieGoers` have all been defined for you. You need to add a few lines of code in the constructor to read the information from a text file, interpret these lines of text properly, and store the information into the matrix. You are asked to complete the two methods that will be called from the constructor to build the matrix. Here are a few tasks you need to complete.

1. Open a text file by a given name and read all the lines in the file into the data member *self.data*

2. Parse the lines in the first part of the input data, which consists of an integer *n* and the names of a total of *n* top movies, followed by an integer *m* and a list of *m* student names. The `split()` method may be helpful here. The values of *n* and *m* can be any positive integers. You should look at the input file *input-data.txt* at this point to understand the structure of the first part of the input file. You need to implement the method `self._read_data()` which reads the list of movies into the 1-D array *self.movies* and the list of names into the 1-D array *self.people*. Note that the two integers *n* and *m* give you the size needed for creating the two arrays.

3. Create the relation matrix *self.matrix* which stores the relation between the students and the movies they have watched. The dimension of the matrix is determined by the size of the people array (row) and the size of the movie array (column).

4. Read the rest of the data that correspond to the matrix into the matrix just created using the method `self._read_matrix()`.

When you finish these four tasks properly, you should be able to run the program *moviegoers.py* without errors, except that no results will be displayed since you haven't completed the method `self.find_common()`, which is our last task of the lab.

The method `find_common()` will compute the common movies seen by the group of students and display the result of calculation. The computation in this method is straightforward now that your program has read the information properly. You need first to create a matrix that is the transposition of the original `self.matrix` using the matrix `transpose()` method. You then multiply the two matrices together to get the result. Use a print method to print the resultant matrix. When you run the program using the following command,

```
python moviegoers.py input-data.txt
```

You should see the result similar to the following.

```
matrix read as :
0 1 0 0 1 1 0 0 0 1
1 0 1 0 1 0 0 0 0 0
1 1 0 0 1 1 0 0 0 1
0 0 0 0 0 1 0 0 0 1
1 0 1 0 0 1 0 0 0 0
1 0 0 0 1 1 0 0 1 1

here is the interest measuring matrix
4 1 4 2 1 3
1 3 2 0 2 2
4 2 5 2 2 4
2 0 2 2 1 2
1 2 2 1 3 2
3 2 4 2 2 5
```

The first part of the output shows the movie watching matrix (given in the input file). Each line in this matrix shows the movies liked by a student. For example, Alice (first row) liked the 2nd, 5th, 6th, and 10th movies while Fred (sixth row) liked the 1st, 5th, 6th, 9th, and 10th movies. The second part of the output shows the interest measure matrix. Recall that this is the result of matrix multiplication. So the interpretation of the matrix should be something like, Bastian (2nd row) liked two of the same movies as Catlin (3rd column), Eva (5th column), and Fred (6th column), while he doesn't have any movies in common with Diego (4th column).

If your program produces the above result (the exact text could be different), your program is likely working correctly. Copy and paste the result into a text file called *outputa.txt*. Now please try your program with a different input data

```
python moviegoers.py test.txt
```

Copy and paste the result of this execution into a file called *outputb.txt* and make this a part of the submission file.

7. Submitting your work

Congratulations! You've completed this lab. Please make sure that your files are well-commented. Submit your program (all four Python files, including the ones given) and the two output data files to Moodle.