# Laboratory practice No. 1: Graph Implementation

**Sarah Henao Gallego**
Universidad EAFIT
Medellín, Colombia
shenaog@eafit.edu.co

October 25, 2018

## *1) Code uploaded to GitHub*

### *1.a. Implementation of DigraphAM using the data structure Adjacency Matrices.*

– The code is uploaded in GitHub along with a Main to test the functions. The file uploaded to GitHub is called **DigraphAM.java** which extends the abstract class Digraph which can be found in the **Digraph.java** file.

### *1.b. Implementation of DigraphAL using the data structure Adjacency Lists.*

– The code is uploaded in GitHub along with a Main to test the functions. The file uploaded to GitHub is called **DigraphAL.java** which extends the abstract class Digraph which can be found in the **Digraph.java** file.

### *1.c. Implementation of a method that returns the vertex with the most neighbors from a given graph.*

– The code is uploaded in GitHub along with a Main to test the functions. The file uploaded to GitHub is called **DigraphAlgorithms.java** which inlcudes a method called **mostSuccessors** which finds the vertex with the most successors in a directed graph.

### *1.d. Implementation of a method that creates a graph from a given text file that represents a map of Medellin.*

—

## *2) Online Exercise*

—

## *3) Project-type Questions*

### *3.a. Explain the algorithm in problem 1.1.*

– In this problem we created a class **DigraphAM** that represents a graph with an adjacency matrix. This class inherit from an abstrac class called **Digraph**. This class contains the attribute *size* which represents the number of vertices of that graph. The class consists of multiple methods:

i. addArc() : Fills up the Adjacency Matrix in order to represent the specific directed graph

ii. showMatrix() : Prints out the Adjacency Matrix

iii. getSuccessors() Finds the connected neighbors of a specific vertex

iv. getWeight() : Finds the weight of the given arc

### *3.b. When is it more convenient to use adjacency matrices versus adjacency lists and vice versa?*

– It is more convenient to use adjacency matrices when the graph is small, in other words, has a small amount of vertices. In this case, the complexity is constant O(1). However, if the problem is large it is better to use adjacency lists because they occupy less space than the adjacency matrices do.

### *3.c. To represent the Medellin map, which data structure is more convenient?*

– Given that Medellin is large and has many cities, it would be more convenient to use adjacency lists. These reasons were explained in the previous question.

### *3.d. Which data structure is better overall?*

– The data structure that is better overall would be adjacency lists. The adjacency matrices are useful when the graphs are connected, in other words, each vertex is related to all the other vertices. Usually this is not the case, and therefore, using adjacency matrices would be a waste of memory space for the computer and hence, adjacency lists are better suited.

### *3.e. Which is better to represent the routing table?*

– The routing table is easier to represent with adjacency matrices given that it is easier to locate the positions of the verices here than in the adjacency lists.

### *3.f. Explain the algorithm in problem 2.*

—

### *3.g. Calculate the complexity of the algorithm in problem 1.2.*

– $T(n) = 2 * (n * m) + C \longrightarrow O(n) = n * m$ By applying the rule of sum and product we eliminate the constants and the scaling factors.

### *3.h. What do the variables 'n' and 'm' mean?*

– Let **n** denote the number of vertices in a given graph and **m** be the number of edges.

## *4) Practice Test Problems*

### *4.a. GRAPHS*

#### 4.1.1 Adjacency Matrix Representation

| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| **1** | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| **2** | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| **3** | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| **4** | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| **5** | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| **6** | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| **7** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

Table 1: Adjacency Matrix Representation for a Graph

#### 4.1.2 Adjacency Lists Representation

```
0 -> [3,4]
1 -> [0, 2, 5]
2 -> [1, 4, 6]
3 -> 7
4 -> 2
5 -> $\oldemptyset$
6 -> 2
7 -> $\oldemptyset$
```

#### 4.1.3 Memory occupation using Adjacency Lists – Worse case scenario

```
O(n^2)
```

Profoessor Mauricio Toro Bermudez
Phone: (+57)(4)2619500 Ext. 9473. Office: 19 – 627
E-mail: mtorobe@eafit.edu.co