Laboratory practice No. 5: Graphs

Omar Alexis Becerra Sierra
EAFIT University
Medellín, Colombia
oabecerras@eafit.edu.co

Juan Jose Tamayo
EAFIT University
Medellín, Colombia
jjtamayoa@eafit.edu.co

May 3, 2018

1) Mock project support questions

1.a. Include an image of the response of the tests of number 1.3

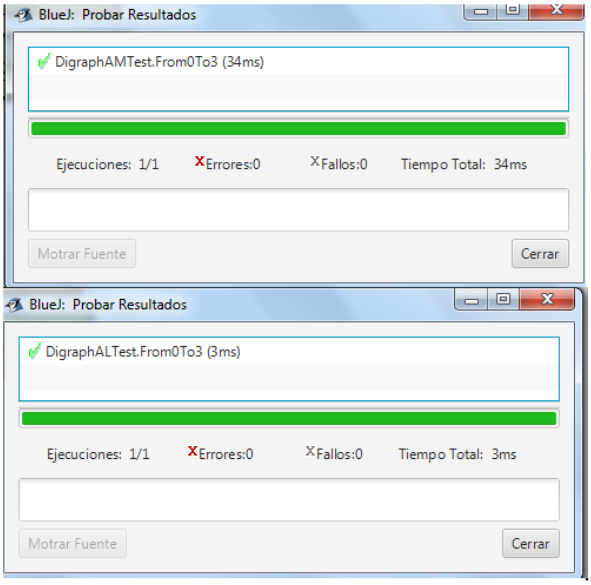

Figure 1: Tests done with Adjacency Lists and Adjacency Matrix

1.b. Write an explanation between 3 and 6 lines of text of the code of number 1.1. Say how it works, how the graph is implemented with matrices and with lists that you made, highlighting the data structures and algorithms used

The matrix graph was created with a matrix of nxn which was assigned by a size is taken each row and each column as a vertex, if the position n, m has a number then the vertex n is connected with the m and the number that is saved in that position is the weight, to obtain the successors of a vertex, we just stand in the row of that vertex and go through it, all the columns with a different number of 0 are successors of the graph.

The graph with adjacency lists was made with a list within another list b, in turn the list was saved objects of type "Pair" which keep 2 integers, to insert a bow you get from list b the vertex a and to that vertex is inserted an object of type Pair that has the vertex destiny and the weight, for get the weight of a bow you get the list that represents the vertex and you start to travel up to find the destination pair and from this you get the weight, to get the successors you get the list that it represents the vertice that is needed and it is adding to a list the vertices that it keeps.

1.c. In which graphs it is more convenient to use the matrix implementation of adjacency and in what cases in more convenient adjacency lists? Why?

Using adjacency matrix is more convenient when all nodes are related to all since it is easier to find which node is related to that other, and adjacent lists s are more effective when the nodes are not related much or have few connections between Yes, since it allows a greater efficiency in memory since it does not spend as much space as the matrix does but the search is slower.

1.d. To represent the map of the city of Medellín from the exercise of number 1.3, What is better to use, Adjacency Matrix or Adjacency Lists? Why?

It is better to use lists for the Medellin map, since this has many elements so the amount of memory used in a matrix would be ridiculous, in addition to the amount of vertices connected to each other is not very high, so it is to use a serious matrix plus a waste of memory what other thing.

1.e. Taking into account the above, answer: What is better to use, Matrix of Adjacency or Adjacency Lists? Why?

For a network like facebook, it would be better to use lists that are flattened since it allows us to have more people without wasting space while in a matrix we would spend more space and waste a bit of it since not all are related to each other.

1.f. Taking into account the above, to represent the routing table, answer: What is better to use, Adjacency Matrix or Adjacency Lists?

To make a routing table it is better to use an adjacency matrix since the port number will always be constant and we will remove any node.

1.g. Calculate the complexity of the online exercises, numerals 2.1 and [optionally] 2.2, and add it to the PDF report

```
import java.util.*;
/**
 * Write a description of class punto2 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class punto2
{
    public  static DigraphAL crear(DigraphAL grafo, int inicio, int fin){
        grafo.addArc(inicio,fin,0);
        grafo.addArc(fin,inicio,0);
        return grafo;
    }

    public static int[] pintar(DigraphAL grafo, int vertices){ //O(m*j*a)
        int[] pintados = new int[vertices]; //c1
        ArrayList<Integer> vecinos; //Aqui se almacenarán todos los vertices que se rela

        for(int i = 0; i < pintados.length; i++){ //m
            if(pintados[i] == 0){ //el grafo no está pintado? //c3
                pintados[i] = 1; //pintar grafo //c4
            }
            vecinos = grafo.getSuccessors(i); // a

            if(pintados[i] == 1){ //el grafo es de color 1? //c6
                for(int j = 0; j < vecinos.size(); j++){ //recorrer vecinos //j
                    int vec = vecinos.get(j); //vecino visitado //c7
                    if(pintados[vec] == 0){ // el vecino está pintado? //c8
                        pintados[vec] = 2; //pintar vecino //c9
                    }
                }
            }else{
                for(int j = 0; j < vecinos.size(); j++){ // j
                    int vec = vecinos.get(j); //c10
```

```
                if(pintados[vec] == 0){ //c11
                    pintados[vec] = 1; //c12
                }
            }
        }
    }

    return pintados; //c13
}

public static boolean revisar(DigraphAL grafo, int[] pintados){ //O(l*j*a)
    ArrayList<Integer> vecinos; //c1
    boolean resp = true; //c2
    for(int i = 0; i < pintados.length; i++){ //l
        vecinos = grafo.getSuccessors(i); // a

        for(int j = 0; j < vecinos.size(); j++){ // j
            int vec = vecinos.get(j);//vecino visitado (numero del vertice vecino) /
            if(pintados[vec] == pintados[i]){ //c5
                resp = false; //c6
            }
        }
    }
    return resp; //c7
}

public static void main(String[] ags){
    int v; //vertivces
    int a; //aristas
    Scanner sc = new Scanner(System.in);
    v = sc.nextInt();

    if(v != 0){
        a = sc.nextInt();
        DigraphAL grafo = new DigraphAL(v);
        while( (v!=0) && (a>0) ){
            int inicio = sc.nextInt();
            int fin = sc.nextInt();
            grafo = crear(grafo,inicio,fin);
            a--;
        }

        int[] pintados = pintar(grafo,v); // colores de los vertices. Color "1" y co
```

```
            if(revisar(grafo,pintados)){
                System.out.println("El grafo es BICOLOREABLE");
            }else{
                System.out.println("El grafo NO es BICOLOREABLE");
            }
        }
    }
}
```

1.h. Explain with your words the variables (what is 'n', what is 'm', etc.) of the complexity calculation of number 3.7

In the previous calculation, n is equal to the number of arcs that the user inserts, since this depends on the duration of the while cycle, the variable m is equal to the number of vertices since this depends on the duration of a cycle within the paint function, a is the complexity of the function getsuccesors that is it executes as much in painting as in painted, and j is the amount of neighbors that has each vertex since of This depends on a cycle in both functions.

1.i. Mock Exam

1.9.1 Complete the graph according to the graph

	0	1	2	3	4	5	6	7
0				1	1			
1	1		1			1		
2					1		1	
3								1
4			1				1	
5								
6			1					

Figure 2: graph

1.9.2 Representation in the form of a list

$0 \Longrightarrow [3, 4]$
$1 \Longrightarrow [0, 2, 5]$
$2 \Longrightarrow [4, 6]$

$3 \Longrightarrow [7]$
$4 \Longrightarrow [2, 6]$
$5 \Longrightarrow []$
$6 \Longrightarrow [1]$

1.9.3 How much memory (care, not time but memory) occupies a representation using adjacency lists for the worst graph directed with n vertices?

$O(n^2)$

1.j. Recommended reading

Graphs

Graphs are data structures rather like trees. The graph doesn't attempt to reflect the geographical positions shown on the map; it shows only the relationships of the vertices and the edges.
Adjacency: Two vertices are said to be adjacent to one another if they are connected by a single edge. The vertices adjacent to a given vertex are sometimes said to be its neighbors.
Paths: A path is a sequence of edges.
Connected Graphs: A graph is said to be connected if there is at least one path from every vertex to every other vertex. However, if you can't go to one place to another, the graph is not connected. A non-connected graph consists of several connected components. *Directed Graphs:* A directed graph is a type of graph in which the edges have a definite sense, unlike the non-directed graph, in which the edges are symmetric relationships and do not point in any direction.
Weighted Graphs: A graphic is weighted when its edges contain data (labels). A label can be a name, a cost or a value of any type of data. This network is also called the activity network, and the number associated with the arc is called the weighting factor.

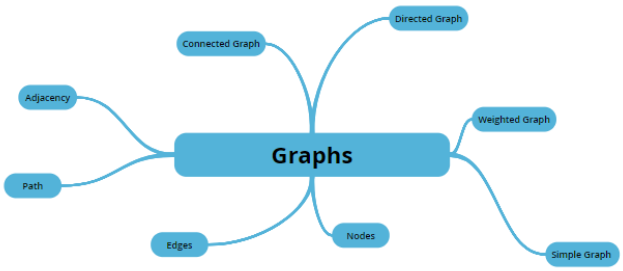

Figure 3: Mental map of the recommended reading