Laboratorio Nro. 1: Implementación de grafos

Alejandro Arroyave Bedoya
Universidad Eafit
Medellín, Colombia
aarroyaveb@eafit.edu.co

Luis Javier Palacio
Universidad Eafit
Medellín, Colombia
ljpalaciom@eafit.edu.co

Santiago Castrillon Galvis
Universidad Eafit
Medellín, Colombia
scastrillg@eafit.edu.co

3) Simulacro de preguntas de sustentación de Proyectos

1. El grafo de matrices de adyacencia se hizo con una matriz, pues era la mejor estructura para representar el grafo de esta manera. Para obtener los sucesores de un vertice utilizamos un ArrayList, pues era adecuado tener una estructura que se ampliara fácilmente. El grafo de lista de adyacencia se implementó con LinkedList de Pair dentro de otro LinkedList, pues necesitamos insertar cosas fácilmente en cualquier posición del arreglo, además de tener que almacenar el destino de una arista con su peso, y para obtener los sucesores de un vertice utilizamos un ArrayList por lo mencionado.
2. La implementación con matrices de adyacencia solo es conveniente usarla cuando el grafo es muy pequeño o cuando cada vértice está relacionado con todos o la mayoría de los demás vértices debido a que no se está perdiendo espacio en memoria y tenemos la ventaja de que la búsqueda de cada elemento se puede hacer en O(1). De lo contrario, si el grafo es muy grande y sus vértices se relacionan con otros pocos, entonces es mucho más conveniente usar listas de adyacencia pues con esto evitamos el gran desperdicio de memoria, haciendo que las relaciones entre vértices sean más puntuales y específicas.
3. Es mejor utilizar listas de adyacencia, porque al utilizar una matriz de adyacencia se tendría que hacer una matriz de igual ancho y largo, los cuales tendrían el tamaño de número de vertices que tenga el grafo, por lo que se gastaría demasiada memoria innecesariamente. En cambio, con listas de adyacencia, solo se se crea una lista la cual el tamaño es el número de vertices en el grafo, y en cada posición de la lista solo se pone con que vertice esta relacionado y el peso de la arista, ahorrando así memoria.
4. En la mayoría de los casos será mucho más conveniente usar listas de adyacencia debido a que no es común que los vértices de un grafo se relacionen con cada uno de los otros vértices, esto quiere decir que no encontraremos utilidad en representar un grafo en una matriz de adyacencia para guardar cada relación entre vértices incluso las inexistentes, tal evento no acontece en las listas de adyacencia puesto que estas solo guardan las relaciones existentes de cada uno de los vértices.
5. Para representar la tabla de enrutamiento, es mejor una matriz de adyacencia, pues para representar un camino de un nodo hacia otro es necesario el peso, por lo que en las listas tendrìan que almacenar una dupla de valores, en cambio en la matriz solo habrìa que cambiar el valor almacenado en la posiciòn.

6. Lo primero que debemos hacer es crear el grafo y añadir los arcos, luego inicializar con la cantidad de vértices un arreglo de enteros que nos representará si un vértice no ha sido pintado "0", o si está pintado del primer color 1, o del segundo color 2. Se toma el primer vértice y en su ubicación correspondiente en el arreglo inicializamos con un 1 significando que este se pinta del primer color. Luego recorreremos todos los sucesores del mismo pintándolos del color 2. Este procedimiento es aplicado con todos los vértices, es decir, se les pregunta su color y se pinta del contrario a todos sus sucesores y si en algún momento, un vértice había sido pintado de un color distinto, el grafo no será biyectivo y en estas circunstancias, el grafo no podía cumplir con lo que exigía el ejercicio y se debía imprimir not bicolorable. Sí al final no habían problemas, se imprimía bicolorable.

7. **2.1)**

```
public boolean Bicolorable() {
 for (int i = 0; i < list.size(); i++) {  //n
        ArrayList<Integer> vecinos = getSuccessors(i);  //C1
        color = arr[i] == 1 ? 2 : 1;  //C2
        for (Integer integer : vecinos) {  //n * n
            if (arr[integer] == arr[i] && arr[i] != 0) {  //C3n * n
                return false;  //C4n * n
            }
            arr[integer] = color;  //C5 + n * n
        }
    }
    return true;  //C6
}
```

T(n) = C + n * n
T (n) es O(C + n * n)
T (n) es O(n * n)
T (n) es O(n^2)

2.2) GroupSum6:

```
public boolean groupSum6(int start, int[] nums, int target) {
if (start == nums.length) { // C1
return target == 0;  // C2
}
return nums[start] == 6 ? groupSum6(start + 1, nums, target -6) //C3 + T(n-1)
: groupSum6(start + 1, nums, target -nums[start])
| | groupSum6(start + 1, nums, target); // C + T(n-1) + T(n-1)
}
```

T(n) = C1 + C2 start == nums.length
T(n) = C + T(n-1) + T(n-1) de lo contrario (peor caso)
T(n) = C*2^n + C'
T (n) es O(C*2^n + C')
T (n) es O(C*2^n)
T (n) es O(2^n)

GroupNoAdj

```
public boolean groupNoAdj(int start, int[] nums, int target) {
        if (start >= nums.length) { //C1
            return target == 0;  //C2
```

```
            } else {
                return groupNoAdj(start + 2, nums, target - nums[start])
                || groupNoAdj(start + 1, nums, target);  // C + T(n-2) + T(n-1)
            }
        }
```

T(n) = C1 + C2 start >= nums.length
T(n) = C + T(n-2) + T(n-1) de lo contrario (peor de los casos)
T(n) = C*2^n + C'
T (n) es O(C*2^n + C')
T (n) es O(C*2^n)
T (n) es O(2^n)

GroupSum5

```
public boolean groupSum5(int start, int[] nums, int target) {
        if (start >= nums.length) {  //C1
            return target == 0;  //C2
    } else {
        if (nums[start] % 5 == 0) { //C3
        return groupSum5(start + 1, nums, target - nums[start]);  //T(n-1)
    }
            if (start != 0 && nums[start] == 1 && nums[start - 1] % 5 == 0) {  //C4
                return groupSum5(start + 1, nums, target);  /T(n-1)
            }
        }
        return groupSum5(start + 1, nums, target - nums[start])
            || groupSum5(start + 1, nums, target);  //T(n-1) + T(n-1)
    }
```

T(n) = C1 + C2 start >= nums.length ó
T(n) = C + T(n-1)
T(n) = C + T(n-1) + T(n-1) de lo contrario (peor caso)
T(n) = C*2^n + C'
T (n) es O(C*2^n + C')
T (n) es O(C*2^n)
T (n) es O(2^n)

GroupSumClump

```
public boolean groupSumClump(int start, int[] nums, int target) {
    if (start >= nums.length) {  //C1
        return target == 0;  //C2
    } else {
        int n = 1;  //C3
        while (start <= nums.length - 2) {  //n
        if (nums[start] != nums[start + 1]) {  //C4
            break;  /C5
        }
        start++;  //C6
        n++;  //C7
        }
        n = n * nums[start];  //C8
        return groupSumClump(start + 1, nums, target - n)
```

```
                || groupSumClump(start + 1, nums, target);  //C + T(n-1) + T(n-1)
        }
}
```

T(n) = C1 + C2 start >= nums.length
T(n) = C + T(n-1) + T(n-1) de lo contrario (peor caso)
T(n) = C*2^n + C'
T (n) es O(C*2^n + C')
T (n) es O(C*2^n)
T (n) es O(2^n)

SplitArray

```
public boolean splitArray(int[] nums) {
return splitArrayAux(nums, 0, 0, 0);  //C + T(n-1) + T(n-1)
 }

public boolean splitArrayAux(int[] nums, int start, int g1, int g2) {
if (start >= nums.length) {  //C1
return g1 == g2;  //C2
        } else {
            return splitArrayAux(nums, start + 1, g1 + nums[start], g2)
                    || splitArrayAux(nums, start + 1, g1, g2 + nums[start]);  // C + T(n-1) + T(n-1)
        }
}
```

T(n) = C1 + C2 start >= nums.length
T(n) = C + T(n-1) + T(n-1) de lo contrario (peor caso)
T(n) = C*2^n + C'
T (n) es O(C*2^n + C')
T (n) es O(C*2^n)
T (n) es O(2^n)

8. **2.1)** La variable n significa el número de vertices del grafo.
2.2) La variable n en todos los ejercicios de este punto significa el índice que se está tomando del arreglo.

4) Simulacro de Parcial

1.

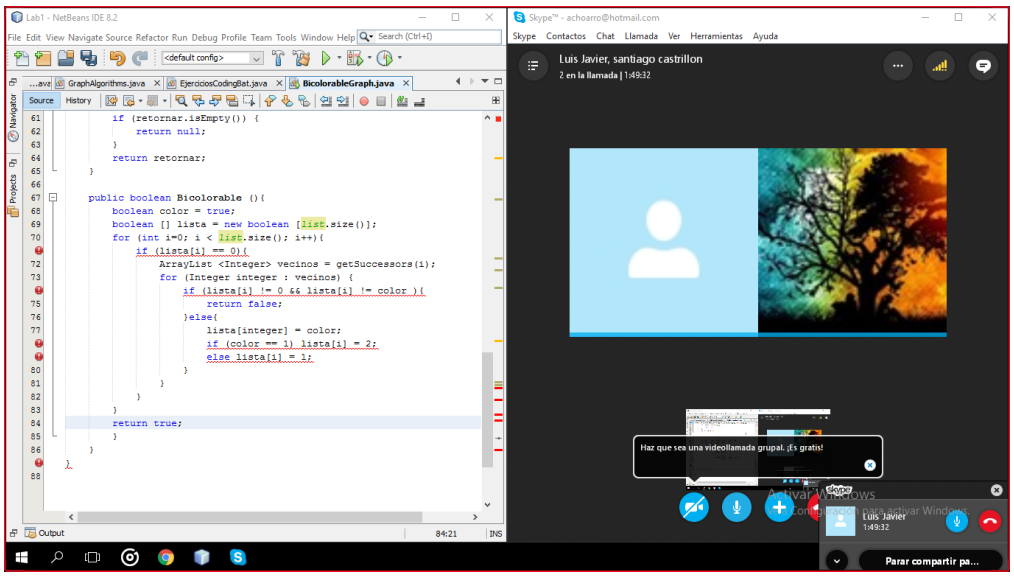
	0	1	2	3	4	5	6	7
0				1	1			
1	1		1			1		
2		1			1		1	
3								1
4			1					
5								
6			1					
7								

DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co

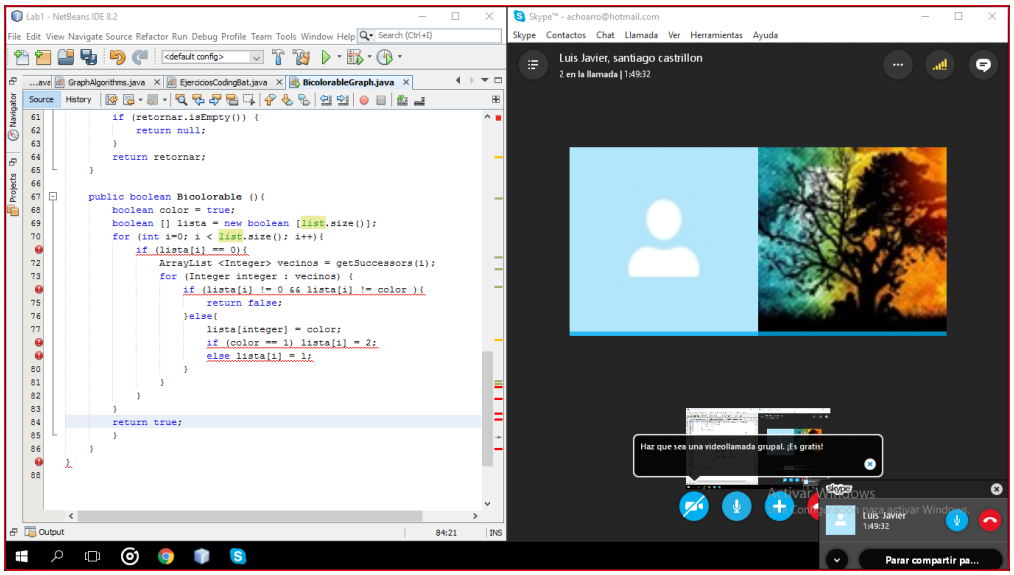
2. 0 -> [3,4]
   1 -> [0,2,5]
   2-> [1,4,6]
   3 -> [7]
   4 -> [2]
   5 -> []
   6 -> [2]
   7 -> []
3. *B) $O(n^2)$*

6) Trabajo en Equipo y Progreso Gradual (Opcional)

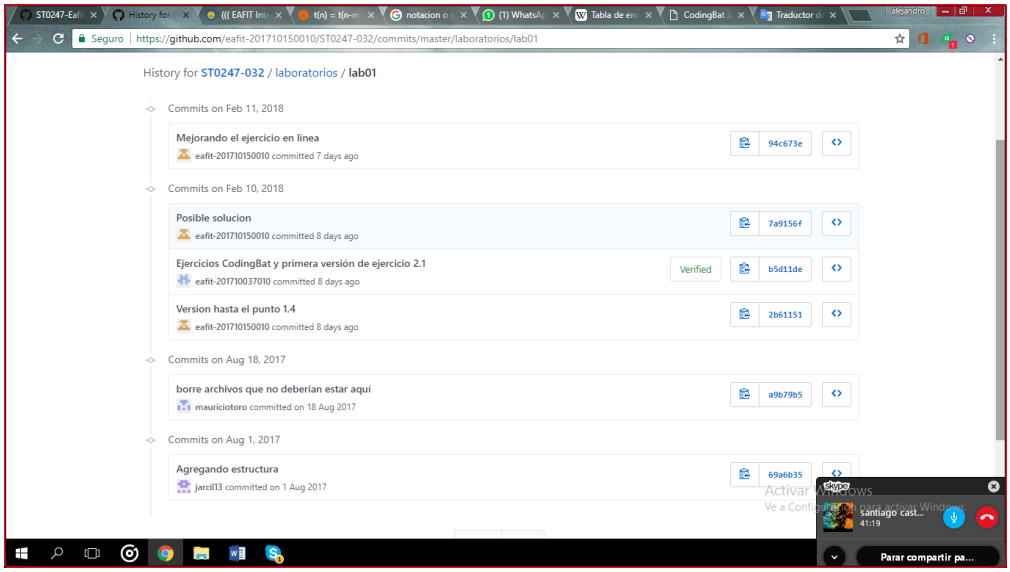
a)



DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co



b)



DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co