Laboratorio Nro. 1: Implementación De Grafos

Isabela Muriel Roldán
Universidad Eafit
Medellín, Colombia
imurielr@eafit.edu.co

Mateo Florez Restrepo
Universidad Eafit
Medellín, Colombia
mflorezr@eafit.edu.co

3) Simulacro de preguntas de sustentación de Proyectos

1. **La clase DigraphAM se encarga de crear un grafo a partir de una matriz, para esto implementa el método addArc, en el que se le asigna un valor a cada arco entre los vértices dados. También se tiene el método getSuccessors que se encarga de devolver una lista con los vértices sucesores al vértice dado, esto lo hace recorriendo la matriz y verificando si los vértices cercanos al vértice original existen o no.**
   Al igual que la clase anterior, DigraphAL utiliza los mismos métodos para crear un grafo a partir de una lista de adyacencia, por lo que en addArc se utiliza una lista y una lista enlazada con el fin de asignar un valor al arco entre dos vértices. El método getSuccesors recorre la lista y obtiene los vértices vecinos, agregándolos a una lista para al final devolverla.
2. **Para utilizar un grafo con la implementación con matrices, es recomendado que todas las posiciones de la matriz estén ocupadas, ya que si, por ejemplo, solo se ocupa una posición por cada columna sería un gasto de memoria innecesario que se podría evitar utilizando la implementación con listas de adyacencia.**
3. **Para presentar el mapa de Medellín en un grafo es preferible utilizar la implementación con listas de adyacencia ya que si se utiliza la implementación con matrices se gasta mucha memoria, y como hay vértices que no están conectados con ningún otro vértice es posible que la memoria se llene y no logre generar la matriz que se requiere.**
4. **Debido a que las matrices ocupan tanto espacio en memoria, y en la mayoría de los casos no todas las posiciones de la matriz necesitan ser utilizadas, es mejor utilizar grafos con la implementación de listas de adyacencia pues estas, a diferencia de la matriz, no ocupan n*n posiciones de memoria.**
5. **En este caso podrían usarse matrices siempre y cuando el grafo sea denso y haya gran cantidad de conexiones y vértices entre los nodos, aunque por lo ya dicho anteriormente podrían gastar mucha memoria en caso de que no exista tantas conexiones entre dispositivos, pues, espacios de la matriz quedaran sin uso, entonces recurriríamos a usar listas para mejorar la eficiencia de la estructura en cuanto a la memoria y el tiempo.**
6. **Para resolver los problemas de este laboratorio hemos usado tanto listas de adyacencia como matrices de adyacencia dependiendo de cada implementación de los grafos que nos piden.**
   Para el ejercicio 2.1 nos pedían que, dados los nodos, arcos y las conexiones entre nodos el programa decidiera si se podía pintar tal grafo con dos colores distintos y arrojara si era BICOLORABLE o NO BICOLORABLE. Para resolver el problema lo que hicimos fue lo siguiente:
   - **Para poder saber si el grafo es bicolorable debíamos verificar si el grafo era bipartito ("Un grafo que se puede particionar en dos subconjuntos V1 y V2, de manera que los nodos de cada subconjunto se puedan relacionar con los del otro subconjunto, pero nunca con los**

del subconjunto al que pertenecen."). Entonces creamos un método booleano llamado "esbipartito" que recibe una matriz y un índice, lo que hace el método es crear un arreglo de colores y una lista en la que se añadirán los nodos, y luego recorrerá la matriz verificando que no haya arcos de un nodo a si mismo, y que en el arreglo de colores haya un color diferente en cada posición con respecto a la anterior (dos colores). Básicamente parte los nodos en dos subconjuntos verificando que cada conjunto pueda tener un color diferente y si se cumple devuelve verdadero, y si no falso.

- **El segundo método llamado "esBicolorable") lo hicimos por método entrada por teclado "Scanner" este recibe los nodos, los arcos y las conexiones, utiliza el valor de los arcos para pedir el nodo inicial y el destino el numero n de veces que corresponde al arco y crea una matriz a partir del grafo dado, luego llama al método "esBipartito" y le pasa la matriz creada para que evalué si se puede colorear de dos colores distintos, en caso de que si se pueda devuelve Bicolorable y en caso contrario No Colorable. El método seguirá pidiendo datos hasta que el valor del nodo sea 0, en cuyo caso se acabará el programa.**

7. Complejidad:

```
boolean esBipartito(int G[][],int src){
    int colorArr[] = new int[G.length];                    //C1
    for (int i=0; i<G.length; ++i)                         //C2*n
       colorArr[i] = -1;                                   //C3*n

    colorArr[src] = 1;                                     //C4
    LinkedList<Integer>q = new LinkedList<Integer>(); //C5
    q.add(src);                                            //C6

    while (q.size() != 0){                                 //C6*n
       int u = q.poll();                                   //C7*n
       if (G[u][u] == 1)                                   //C8*n
          return false;                                    //C9
       for (int v=0; v<G.length; ++v){                     //C10*n^2
          if (G[u][v]==1 && colorArr[v]==-1){              //C11*n^2
             colorArr[v] = 1-colorArr[u];                  //C12*n^2
             q.add(v);                                     //C13*n^2
          }
          else if (G[u][v]==1 && colorArr[v]==colorArr[u])      //C14
             return false;                                      //C15
       }
    }
    return true;                                           //C16
 }
```

T(n) = C1+C4+C5+C6+C9+C13+C14+C15+C16+n*(C2+C3+C6+C7+C8)+n^2*(C10+C11+C12+C13)
T(n) = O(C+C*n+C*n^2)
T(n) = O(C*n^2)
T(n) = O (n^2)

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

```
private static void esBicolorable (){
      Scanner sc= new Scanner(System.in);          //C1
      System.out.println(" ");                     //C2
      int nodos= sc.nextInt();                     //C3
        if(nodos!=0){                              //C4
        int arcos= sc.nextInt();                   //C5
        int G[][] = new int[nodos][nodos];         //C6
        for(int x =0; x<arcos; ++x){               //C7*n
           int ini= sc.nextInt();                  //C8*n
           int fin= sc.nextInt();                  //C9*n
           G[ini][fin]=1;                          //C10*n
           G[fin][ini]=1;                          //C11*n

        }

        BiColorable b = new BiColorable();         //C12
        if (b.esBipartito(G, 0))                   //T(n^2)
          System.out.println("BICOLORABLE");       //C13
        else
          System.out.println("NO BICOLORABLE"); //C14

        esBicolorable();                           //T(n)
      }

   }
```

T(n) = C1+C2+C3+C4+C5+C6+C12+C13+C14+n*(C7+C8+C9+C10+C11)+T(n^2)+T(n)

T(n) = O(C+C*n)
T(n) = O(n) → Creando la matriz a partir del grafo

T(n)= T(n^2)
T(n) = O(n^2) → Evaluando el grafo en "esBipartito"

8. **n y m en la complejidad: En los cálculos de la complejidad anterior, estas variables se refieren a la cantidad de veces que el algoritmo tiene que ejecutar algún paso o acción al momento de compilarse para llegar al resultado esperado. En los ejercicios anteriores usamos en la gran mayoría solo la variable n ya que en los códigos solo se trabajaba con un solo tipo de entrada igual para cada caso. La variable m la usaríamos si hubiera más de una entrada para poder diferenciar el alcance de la complejidad de los códigos.**

4) Simulacro de Parcial

1.

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

2. *0 -> [3,4]*
1 -> [2,5]
2 -> [1,4,6]
3 -> [7]
4 -> [2]
5 -> []
6 -> [2]
7 -> []

3. *b*

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