Laboratorio Nro. 1: Implementacion de Grafos

Gonzalo Garcia
Universidad Eafit
Medellín, Colombia
ggarciah@eafit.edu.co

Eduard Damiam Londoño
Universidad Eafit
Medellín, Colombia
edlondonog@eafit.edu.co

3) Simulacro de preguntas de sustentación de Proyectos

1. El grafo con matrices se hizo haciendo una matriz de nxn (es es igual al atributo size), se toma cada fila y cada columna como un vértice, si la posición nm tiene un numero entonces el vértice n esta conectado con el m y el numero que se guarda en esa posición es el peso, para obtener los sucesores de un vértice, solo nos paramos en la fila de ese vértice y la recorremos, todas las columnas con un numero distinto de 0 son sucesores del grafo.

   El grafo con listas de adyacencia se hizo con una lista a detro de otra lista b, a su vez la lista a guardaba objetos de tipo “Pair” los cuales guardan 2 enteros, para insertar un arco se obtiene de la lista b el vértice a y a ese vértice se le inserta un objeto de tipo Pair que tiene el vértice destino y el peso, para obtener el peso de un arco se obtiene la lista que representa al vértice y se empieza a recorres hasta encontrar el Pair destino y de este se obtiene el peso, para obtener los sucesores obtiene la lista que representa el vertice que se necesita y va añadiendo a una lista los vertices que guarda.

2. Los grafos con matrices son mejores cuando se tienen pocos vertices ya que el problema de usar grafos con matrices es el espacio que gasta la matriz, por esta misma razón cuando ya se tiene un grafo muy extenso es mejor usar las listas enlazadas.

3. Es mejor usar listas para el mapa de medellin, ya que este tiene muchos elementos por lo que la cantidad de memoria usada en una matriz seria ridícula, además de que la cantidad de vertices conectados entre si no es muy alta, por lo que es usar una matriz seria mas un desperdicio de memoria que otra cosa

4. Depende del problema, hay problemas donde no se insertaran o eliminaran nodos y no serán muchos, en este caso es mejor la matriz porque el acceder en un matriz es O(1) lo que es muy bueno, pero en casos donde habran muchos nodos y se insertan o eliminan es mejor usar las listas de adyacencia ya que si bien son mas lentas en acceder, son mejores borrando y añadiendo, como también gastan menos memoria, otra cosa importante es cuantos nodos se conectan entre si ya que si no son muchos también es mejor usar una lista de adyacencia.

5. Para el enrutador seria mejor una matriz, la cantidad de nodos es pequeña y no se va a insertar ni remover ningún nodo, además de que al ser hardware no hay gasto de memoria realmente.

6. El algoritmo se apoya en tres metodos: crear(), es el método que se encarga de ir construyendo las aristas del grafo y este a su vez emplea el metodo de la clase DigraphAL llamado addArc el cual se encarga de agregar y unir nuevos vertices al grafo, pintar(), es el método que se encarga de asignarle a cada uno de los vértices un color ya sea color "1" o color "2" empleando un arreglo de enteros que representará el "color" que tiene cada uno de los vertices, el metodo getSuccessors() de la clase DigraphAL para recorrer los vertices vecinos, si el vertice es de color "1", todos sus vecinos los pintará de color "2" y viceversa (pinta solo los vecinos que aun no han sido pintados) y por último el método revisar(), el cual se encarga de verificar que cada vecino de cada uno de los vertices de nuestro grafo no cuente con el mismo color (utilizando dos ciclos anidados y el metodo getSuccessors) para así determinar si el grafo es Bicoloreable o no bicoloreable.

7.

```
        int v; //vertices //c1
        int a; //aristas //c2
        Scanner sc = new Scanner(System.in); // c3
        v = sc.nextInt(); //c4

        if(v != 0){ //c5
            a = sc.nextInt(); //c6
            DigraphAL grafo = new DigraphAL(v); //c7
            while( (v!=0) && (a>0) ){  //n
                int inicio = sc.nextInt(); // c8
                int fin = sc.nextInt(); //c9
                grafo = crear(grafo,inicio,fin); //c10
                a-    //c11
            }

            int[] pintados = pintar(grafo,v); // colores de los vertices. Color "1" y color "2" //m*j*a

            if(revisar(grafo,pintados)){ //l*j*a
                System.out.println("El grafo es BICOLOREABLE");
            }else{ // c12
                System.out.println("El grafo NO es BICOLOREABLE");
            }
        }
    }
```

T(n) = C+n+(m*j*a)+(l*j*a)
T(n) = n+(m*j*a)+(l*j*a) regla de la suma

Como no se si alguna de las variables es mayor a la otra la complejidad final es:
O(n+(m*j*a)+(l*j*a))

8. En el calculo anterior n es igual a la cantidad de arcos que inserte el usuario, ya que esto depende la duración del ciclo while, la variable m es igual a la cantidad de vertices ya que de esta depende la duración de un ciclo dentro de la función pintar, a es la complejidad de la función getsuccesors que se ejecuta tanto en pintar como en pintados, y j es la cantidad de vecinos que tiene cada vertice ya que de este depende un ciclo en ambas funciones.

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

4) Simulacro de Parcial

1.

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

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