Laboratorio Nro. 4
Implementación de Grafos

Susana Álvarez
Universidad Eafit
Medellín, Colombia
salvarezz1@eafit.edu.co

Mariajose Franco Orozco
Universidad Eafit
Medellín, Colombia
mfrancoo@eafit.edu.co

3) Simulacro de preguntas de sustentación de Proyectos

3.1) En el punto 1, creamos una estructura de datos que contiene un arraylist para representar el mapa de la ciudad. El primer arrayList es un arraylist de vértices y este arraylist contiene toda la información de cada vértice que es: el id, coordenada en x, coordenada en y el nombre. El segundo arrayList es el que contiene los arcos. Cada arco tiene 2 ids, cada uno correspondiente a un vértice, la distancia entre los vértices, y un nombre que representa el nombre de la calle. En la clase estructura de datos están varios métodos tales como getSuccesors, que retorna con que vértices se relaciona un vértice dado y getWeight que retorna la distancia que hay entre 2 vértices.

3.2) Como hay alrededor de 300,000 vértices entonces la matriz es de 300,000*300,000. Así, se estarían guardando 90,000,000,000 datos que están representados por doubles por que la distancia no siempre esta dada por números enteros. Cada double ocupa un espacio de 64 bits, o de 8 bytes. Así la matriz entera ocupa 90,000,000,000*8=720,000,000,000 bytes que equivale a 720 gigabytes.

3.3) Como los identificadores de los puntos del mapa no empiezan en cero tuvimos mucho problema al principio porque no sabíamos de que tamaño crear el Digraph. Luego de intentar mucho hacer la estructura de datos con la clase DiagraphAl, nos dimos cuenta de que era imposible. Al crear un DiagrahpAL, cada vértice representa una posición del arreglo que esta determinada por su identificador. Pero en este caso los identificadores al ser tan grandes no son int sino long. Así estábamos asignando posiciones en un arreglo no con int si no con long lo cual es imposible porque las posiciones en un arreglo siempre son representadas con int. Por esto tuvimos que reestructurar nuestra estructura de datos y decidimos utilizar 2 arraylists: uno que guardara vértices y otro que guardara los arcos. De esta forma las posiciones del arraylist de vértices no representan el id de cada vértice. Acá, cada vértice toma cualquier posición en el arraylist y para ver su id u otra información, se accede a los atributos del vértice. En el arraylist de arcos, la posición en que esta un determinado arco no dice nada de este. Para ver la información del arco se debe acceder a los atributos de este.

3.4) El algoritmo 2.1, consiste en una clase llamada grafo, esta tiene como atributos un entero numero de vértices del grafo y la lista de adyacencia, es decir, la que nos muestra que vértices tienen aristas entre sí. El primer método es el constructor, este tiene como parámetro un entero v, que va a ser el numero de vértices que el usuario le ingrese al grafo; de cada vértice va a salir una lista, con el objetivo de mostrar ese vértice a que otro vértice esta apuntando,

es decir, las aristas entre ellos. Para determinar las aristas entre los vértices hay un método llamado agregarArista, el cual tiene dos enteros a y b como parámetros, este se encarga de agregar las aristas entre los vértices a y b, es decir, el entero a ingresado estará apuntando a el entero b ingresado y viceversa. Luego, hay un método llamado color, en este método, lo primero que se realiza es crear un arreglo de enteros llamado respuesta, la longitud de este arreglo va a depender del numero de vértices del grafo. Luego, se inicializa este arreglo llenándolo de -1, lo cual hará referencia a que no se le ha asignado ningún color a ningún vértice todavía. Luego, al primer vértice del arreglo respuesta se le asigna el 0, que va a hacer referencia al primer color. Luego se crea un arreglo de booleanos llamado temp, este va a guardar todos los colores permitidos, es decir, cuando el vértice al que se le está apuntando no tiene el mismo color del vértice anterior; la longitud de este también va a depender del numero de vértices, y se va a inicializar llenándolo de valores booleanos true, ya que inicialmente todos los colores van a estar permitidos. Luego, se realiza un ciclo en el cual se convierte la lista de adyacencia en la posición i en una iteración, se realiza otro ciclo en el que se revisa la posición siguiente de la iteración, si el siguiente ya está ocupado, pone el arreglo temp en falso para que no pueda tomar ese valor. Luego se halla el color de cada vértice en el arreglo temp y se le asigna ese color encontrado a el arreglo respuesta. Como solo queremos determinar si se puede colorear de dos colores, estos colores serán referenciados como 0 y 1, entonces cuando el color es mayor o igual a 2, el grafo no va a ser bicoloreable, de lo contrario si va a ser coloreable.

3.5) La complejidad del algoritmo 2.1 es $O(n^2)$

3.6) La variable n representa el número de vértices.

4) Simulacro de Parcial

4.1)

	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								

4.2)

0 → [3,4]
1 → [0,2,5]
2 → [1,4,6]
3 → [7]
4 → [2]
5 →
6 → [2]
7 →

4.3) B