Laboratorio Nro. 1: Implementación y Recorrido De Grafos

Ricardo Rafael Azopardo Cadenas
Universidad Eafit
Medellín, Colombia
rrazopardc@eafit.edu.co

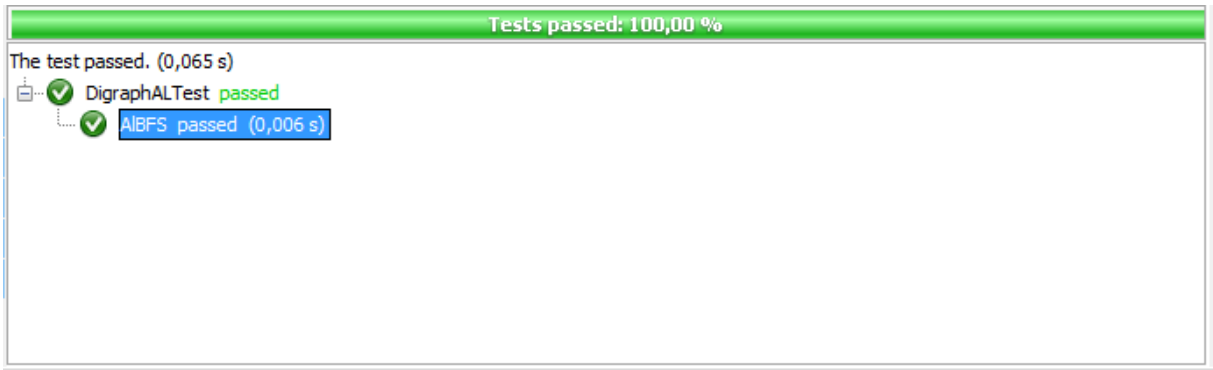
Jhon Jairo Chavarria Gaviria
Universidad Eafit
Medellín, Colombia
jjchavarrg@eafit.edu.co

3) Simulacro de preguntas de sustentación de Proyectos

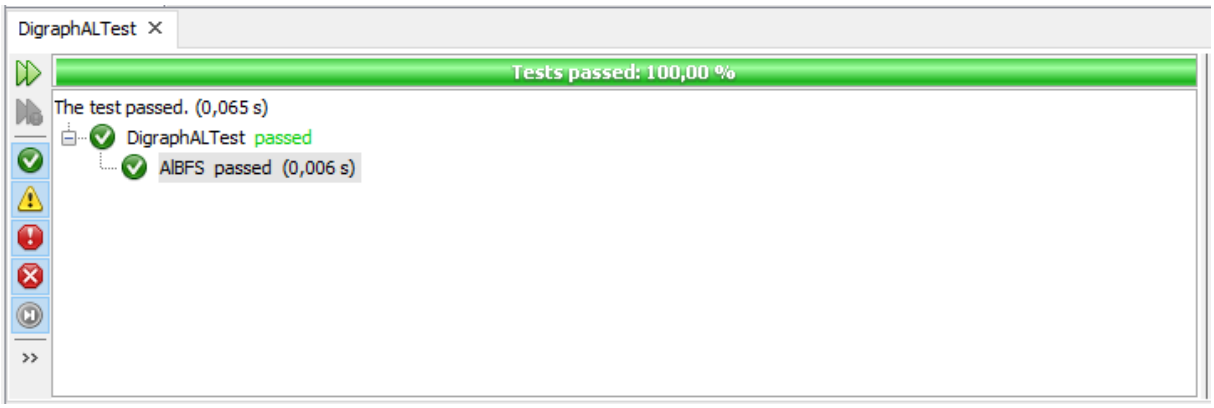
1.7 Explicación del punto 1.1:

El objetivo era implementar grafos dirigidos por listas de adyacencia y matrices de adyacencia, ambas clases heredan de la clase abstracta graph, la cual tiene un int size, representado el tamaño del grafo (su número de vértices), y tres métodos, addArc, para añadir una arista la cual enlaza un vértice (source) con otro (destination) y le asigna una etiqueta un peso (weight) a la arista, haciendo uso de la clase pareja. Un método getSuccesors, que retorna todos los sucesores (hijos) de un vértice determinado y un método getWeight que retorna el peso entre dos vértices.

3.1 Incluyan una imagen de la respuesta de las pruebas del numeral 1.6



Pruebas de grafos de listas de adyacencia en recorridos BFS.



Pruebas de Grafos de Matrices de Adyacencia en recorridos BFS.

3.2 Escriban una explicación entre 3 y 6 líneas de texto del código del numeral 1.4. Digan cómo funciona, cómo está implementado el grafo con matrices y con listas que hizo, destacando las estructuras de datos y algoritmos usados.

3.3 ¿En qué grafos es más conveniente utilizar la implementación con matrices de Adyacencia y en qué casos en más convenientes listas de adyacencia? ¿Por qué?

La elección entre listas o matriz está relacionada a que tantas relaciones (aristas) existen en el grafo. El tamaño de un grafo representado por listas de adyacencia solo es el número de aristas, sin embargo para buscar una arista en específico es necesario buscar en toda la lista de adyacencia. Por otro lado en una matriz, para encontrar una arista, seria en tiempo constante, ya que solo sería necesario indicar sus índices i, j, sin embargo, incluso si un grafo es disperso en el cual los vértices no tienen conexiones con casi todos los vértices, la matriz seguiría teniendo un tamaño del número de vértices al cuadrado. El criterio de selección es simple, si el grafo en el grafo casi todo está conectado, la matriz es muy conveniente ya que se ocuparía casi o igual de espacio en una lista y se realizarían búsquedas muy rápidas, pero si es disperso una representación de listas de adyacencia ocuparía solo el espacio adecuado.

3.4 Para representar el mapa de la ciudad de Medellín del ejercicio del numeral 1.4, ¿qué es mejor usar, Matrices de Adyacencia o Listas de Adyacencia? ¿Por qué?*

En la practica utilizamos hashmap y arraylist para guardas los vértices y los arcos como mapas debido a que era más rápido y fácil. Entre matrices o listas de adyacencia es mejor utilizar las lista de adyacencia ya que en una matriz quedarían muchos espacio vacios.

3.5 Respondan: ¿Que es mejor usar, Matrices de Adyacencia o Listas de Adyacencia? ¿Por qué?

No existe respuesta única, como ya fue establecido en el punto 3.3, es un criterio de selección relacionado a la dispersión del grafo, si son pocas conexiones entre todos los vértices es más conveniente usar listas ya que se ocuparía poco espacio, si en el grafo casi todo está conectado es más conveniente usar matrices, ya que se ocuparía casi todo el espacio y se realizarían búsquedas más rápidas.

3.6 Para representar la tabla de enrutamiento, respondan: ¿Qué es mejor usar, Matrices de Adyacencia o Listas de Adyacencia?*

Lista de adyacencia debido al espacio, porque en la matriz pueden reservar mucho espacio de memoria que no seba a utilizar.

3.7 Para recorrer grafos, ¿en qué casos conviene usar *DFS*? ¿En qué casos *BFS*?

La elección depende de la situación y el objetivo del grafo. Si es más conveniente revisar a profundidad cada uno de los hijos de cada vértice DFS es una mejor opción, por ejemplo si se requiere encontrar la salida de un laberinto, se prueban todos los caminos a profundidad y se hace backtracking hasta el punto de inicio, hasta que se encuentre el camino correcto. Por otro si es más conveniente revisar por amplitud todos los hijos de un vertice, es más conveniente BFS, por ejemplo, si es necesario encontrar todos los amigos de mis amigos en una red social como Facebook, BFS sería una opción muy óptima.

3.8 ¿Qué otros algoritmos de búsqueda existen para grafos? Basta con explicarlos, no hay que escribir los algoritmos ni programarlos.

Algoritmos voraces: también llamados algoritmos greedy, son algoritmos de búsqueda que en cada vértice eligen el camino, que basado en un criterio de selección, o una heurística, eligen el camino más óptimo para llegar a la solución.

A*: es una combinación entre algoritmos voraces y algoritmos greedy, son algoritmos que en cada paso basado en un criterio de selección, revisa los hijos más cercanos a la meta y busca el camino más óptimo a la solución.

3.9 Expliquen con sus propias palabras la estructura de datos que utilizan para resolver los problemas y cómo funcionan los algoritmos realizados en el numeral 2.1 y, voluntariamente, todos los ejercicios opcionales del punto 2.

En el ejercicio en línea sobre determinar si un grafo es bicolorable o no, el algoritmo usado fue el siguiente: Se crea un grafo basado en las primeras dos entradas, la primera el número de vértices y la segunda el número de aristas, como condición del ejercicio se asume que no hay vértices que apunten así mismo, el grafo es no dirigido y el grafo es fuertemente conexo. Luego se crea un arreglo de colores por cada grafo el cual sus posiciones varían entre 3 números:

- 0 para simbolizar sin color.
- 1 para simbolizar un color.
- 2 para simbolizar otro color.

El primer vértice leído se añade a una cola y se pinta de color 1 y se empieza a recorrer el grafo en BFS, en cada vértice que se recorra se hace una pregunta, si el vértice que apunta es del mismo color del primer vértice, de ser este el caso, simboliza que existen dos colores adyacentes y el grafo no es bicolorable (NOTBICOLORABLE), de no ser el caso simplemente se le asigna un color diferente al vértice inicial (1 o 2) y se sigue recorriendo el grafo hasta que todos los vértices estén pintados, si se logra completar el recorrido el grafo es bicolorable (BICOLORABLE).

3.10 Calculen la complejidad del ejercicio 2.1 y, voluntariamente, todos los Ejercicios Opcionales del punto 2.

Por cada grafo que se compruebe, al ser un recorrido en BFS, comprobar si es bicolorable o no, tendrá una complejidad de O(|V| + |E|).

3.11 Expliquen con sus palabras las variables (*qué es 'n', qué es 'm', etc.*) del cálculo de complejidad del numeral 3.10.

En la notación de grafos el cálculo se hace en términos de vértices y aristas.

V simboliza el número de vértices.

E simboliza el número de aristas.

4) Simulacro de Parcial

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

4 -> [2]
5 -> [ ]
6 -> [2]
7 -> [ ]

2.

	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								

3. 0->3->7->4->2->1->5->6
1->0->3->7->4->2->6->5
2->1->0->3->7->4->5->6
3->7
4->2->1->0->3->7->5->6
5->
6->2->1->0->3->7->4->5
7->

4. 0->3->4->2->7->1->6->5
1->0->2->5->3->4->6->7
2->1->4->6->0->5->3->7
3->7
4->2->1->6->0->5->3->7
5->
6->2->1->4->0->5->3->7
7->

5. a) O(n)

6.

```
public static ArrayList hayCamino(Graph gd, int ini, int fin) {
    ArrayList<Integer> visitados = new ArrayList<>(gd.size());
```

```
        boolean[] flags = new boolean[gd.size()];
        if (hayCamino(gd, ini, fin, visitados, flags)) {
            return visitados;
        } else {
            System.out.println("No se encontro el vertice.");
            visitados = new ArrayList<>(gd.size());
            return visitados;
        }
    }
 public static boolean hayCamino(Graph gd, int ini, int fin, ArrayList<Integer>
 visitados, boolean[] flags) {
        visitados.add(ini);
        flags[ini] = true;
        if (ini == fin) {
            return true;
        }
        ArrayList<Integer> hijos = gd.getSuccessors(ini);
        if (hijos == null) {
            return false;
        }
        for (int i = 0; i < hijos.size(); i++) {
            if (!flags[hijos.get(i)] && hayCamino(gd, hijos.get(i), fin, visitados,
 flags)) {
                return true;
            }
        }
        return false;
}
```

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