Programación 3

Trabajo Práctico Especial

Cursada 2023

Primera parte

El objetivo de esta primera parte del trabajo práctico especial es implementar la clase Grafo, siguiendo la interfaz propuesta, para luego resolver un par de algoritmos simples sobre el mencionado contenedor.

Implementación

A partir de la definición de una interfaz para la clase **Grafo** (Grafo.java), la declaración de la clase **GrafoDirigido** y la implementación de la clase **Arco**, que se muestran resumidamente a continuación:

```
public interface Grafo<T> {

	// Agrega un vertice
	public void agregarVertice(int verticeId);

	// Borra un vertice
	public void borrarVertice(int verticeId);

	…
}

public class GrafoDirigido<T> implements Grafo<T> {

	@Override
	public void agregarVertice(int verticeId) {
		// TODO Auto-generated method stub

	}

	@Override
	public void borrarVertice(int verticeId) {
		// TODO Auto-generated method stub

	}
	…
}

public class Arco<T> {

	private int verticeOrigen;
	private int verticeDestino;
	private T etiqueta;

	…
}
```

Será necesario:

- Elegir una estructura de implementación capaz de almacenar el conjunto de vértices (valores enteros) y el conjunto de arcos (tipo de datos parametrizado T)
- Implementar cada uno de los métodos públicos propuestos por la interfaz.

- Detallar la complejidad de cada uno de los métodos implementados de acuerdo a la estructura elegida.

Se propone el siguiente formato para explicitar la complejidad de cada método:

```
/**
 * Complejidad: O(X) donde X es ... debido a que debe
 * "realizar lo siguiente" para verificar si existe un arco.
 */
@Override
public boolean existeArco(int verticeId1, int verticeId2) {
      // TODO Auto-generated method stub
      return false;
}
```

Una vez implementado el contenedor Grafo se deberán resolver las tres funciones siguientes, también respetando la interfaz propuesta en las clases ServicioBFS, ServicioDFS y ServicioCaminos.

```
// Servicio bfs
public ServicioBFS(Grafo<?> grafo) {...}

public List<Integer> bfsForest(){...}


// Servicio dfs
public ServicioDFS(Grafo<?> grafo) {...}

public List<Integer> dfsForest() {...}


// Servicio caminos
public ServicioCaminos(Grafo<?> grafo, int origen, int destino, int
lim) {...}

public List<List<Integer>> caminos(){...}
```

Descripción de los servicios:
BFS Forest: dado un grafo, realiza un recorrido en anchura y retorna un orden posible de descubrimiento para los vértices durante ese recorrido.
DFS Forest: dado un grafo, realiza un recorrido en profundidad y retorna un orden posible de descubrimiento para los vértices durante ese recorrido.
Caminos: dado un origen, un destino y un límite "lim" retorna todos los caminos que, partiendo del vértice origen, llega al vértice de destino sin pasar por más de "lim" arcos. **Aclaración importante:** en un camino no se puede pasar 2 veces por el mismo arco.

Nota: Si bien se podrán agregar los métodos públicos y privados que se consideren necesarios, **NO** se podrá modificar la interfaz propuesta de las clases ni de los servicios. Esto es muy importante ya que **no se corregirá** la implementación de los métodos o servicios cuya interfaz no respete lo propuesto.
Para implementar los servicios es recomendable crear métodos privados auxiliares para agregar los parámetros que sean necesarios en la ejecución del servicio.
Los esqueletos de las clases pueden encontrarse en el GitHub de la materia.

Requisitos de la entrega

Se deberá entregar un proyecto (junto al código fuente) que compile correctamente el código de la resolución solicitada.

Fecha de entrega y modalidad

La primera entrega se deberá presentar el **sábado 27 de mayo** mediante un envío por email a la cuenta de la materia.