Universidad Simón Bolívar
Departamento de Computación y
Tecnología de la Información
CI-2693: Laboratorio de Algoritmos III

Prof. Fernando Torre Mora

# Proyecto 1 Septiembre – Diciembre 2018

**Implementación del TAD Grafo genérico para multigrafos**

## 1 Introducción

El objetivo de este proyecto es la familiarización con las operaciones básicas de los Tipos Abstractos de Datos (TADs) Grafo, Grafo No Dirigido y Grafo Dirigido. Para ello se desea que implemente los siguientes TADs usando el lenguaje de programación JAVA y también que desarrolle una aplicación cliente que permita probar los TADs.

Para la implementación se creará una clase abstracta llamada GRAFO que contendrá las operaciones y estructura de datos asociados a un grafo, sea este dirigido o no. Los grafos podrán tener lados múltiples y bucles. La clase GRAFO tendrá dos clases concretas derivadas: Grafo no dirigido, y Grafo dirigido. Se desea que implemente el TAD GRAFO utilizando una Lista de Adyacencias.

Las clases de este proyecto deben ser *Generics*, lo que les permite actuar como contenedores del tipo especificado.

## 2 Requerimientos de la implementación

La estructura de datos para almacenar un grafo con lados múltiples sería como sigue: Como tipos base se asumen los tipos VERTICE y LADO.

### 2.1 El TAD Vértice<E>

El TAD Vértice tiene en su representación un identificador de tipo `String`, un dato almacenado del tipo especificado por el constructor genérico y un atributo de tipo `double` que es el peso asociado al vértice. **Este TAD debe ser implementado como una clase concreta**. Las operaciones mínimas que posee el TAD Vértice son las siguientes:

- Crear Vértice: (`String` id, E dato, `double` p) → Vértice
  Crea un nuevo vértice con un identificador *id*, conteniendo el *dato* y un peso *p*.
- getPeso: (Vértice v) → `double`
  Obtiene el peso del vértice *v*.
- getId: (Vértice v) → `String`
  Obtiene el identificador del vértice *v*.
- getDato: (Vértice v) → E
  Obtiene el dato contenido en el vértice *v*.
- toString: (Vértice v) → `String`
  Proporciona una representación del vértice v como una cadena de caracteres.

### 2.2 El TAD Lado<E>

El TAD Lado esta formado en su representación por un identificador de tipo `String`, un dato almacenado del tipo especificado por el constructor genérico y un peso de tipo `double`. **Este TAD debe ser implementado como una clase abstracta**. El TAD Lado tiene dos subtipos el Arco y la Arista. Las operaciones de este TAD son las siguientes:

- crearLado: (`String` id, E dato, `double` p) → Lado
  Crea un nuevo lado con un identificador *id*, conteniendo el *dato* y un peso *p*.
- getPeso: (Lado l) → `double`
  Obtiene el peso del lado *l*.
- getId: (Lado l) → `String`
  Obtiene el identificador del lado *l*.
- getDato: (Vértice) → E
  Obtiene el dato contenido en el vértice V
- toString: (Lado l) → `String`
  Método abstracto para la representación del lado *l* como una cadena de caracteres.

### 2.2.1 El TAD Arco<E>

Subtipo del TAD Lado<E> que representa a los lados que componen al TAD Grafo Dirigido. Es implementado como una clase concreta derivada de la clase abstracta Lado. Este TAD posee las siguientes operaciones:

- crearArco: (String id, E dato, `double` p, Vértice vi, Vértice vf) → Arco
  Crea un nuevo arco con un identificador *id*, conteniendo el *dato*, un peso *p*, un vértice en el extremo inicial *vi* y un vértice en el extremo final *vf*.
- getExtremoInicial: (Arco a) → Vértice
  Obtiene vértice que es el extremo inicial del arco *a*.
- getExtremoFinal: (Arco a) → Vértice
  Obtiene vértice que es el extremo final del arco *a*.
- toString: (Arco a) → `String`
  Retorna la representación en String del arco *a*.

### 2.2.2 El TAD Arista<E>

Subtipo del TAD Lado que representa a los lados que componen al TAD Grafo No Dirigido. Es implementado como una clase concreta derivada de la clase abstracta Lado. Las operaciones que corresponden al TAD Arista son las siguientes:

- crearArista: (`String` id, E dato, `double` p, Vértice u, Vértice v) → Arista
  Crea una nueva arista con un identificador *id*, conteniendo el *dato*, un peso *p*, un vértice en el extremo inicial *u* y un vértice en el extremo final *v*.
- getExtremo1: (Arista a) → Vértice
  Obtiene vértice que es el primer extremo de la arista *a*.
- getExtremo2: (Arista a) → Vértice
  Obtiene vértice que es el segundo extremo de la arista *a*.
- toString: (Arista a) → `String`
  Retorna la representación de la arista *a* como un `String`.

## 2.3 El TAD Grafo<V,L>

Este TAD contendrá las operaciones asociados a un grafo, sea dirigido o no dirigido. Los grafos podrán tener lados múltiples y bucles. El TAD Grafo debe ser implementado como una interfaz de JAVA llamada Grafo. Todos los identificadores de los vértices que componen a un grafo deben ser únicos. De la misma manera no deben haber identificadores repetidos de los lados componen a un grafo. En las clases del Generic, *V* especifica la clase de los datos que contendrán los vértices y L especifica la clase de los datos que contendrán los lados Para garantizar el dar cumplimiento de esto, se observa que las `Lists` son colecciones iterables. Se presentan las operaciones del TAD Grafo que deben ser implementadas:

- cargarGrafo: (Grafo<V,L> g, `String` archivo) → `boolean`
  Carga en un grafo la información almacenada en el archivo de texto cuya dirección, incluyendo el nombre del archivo, viene dada por archivo. El archivo dado tiene un formato determinado que se indicará más abajo. Se retorna `true` si los datos del archivo son cargados satisfactoriamente en el grafo, y `false` en caso contrario. Este método debe manejar los casos en los que haya problemas al abrir un archivo y el caso en el que el formato del archivo sea incorrecto.
- numeroDeVertices: (Grafo<V,L> g) → `entero`
  Indica el número de vértices que posee el grafo.
- numeroDeLados: (Grafo<V,L> g) → `entero`
  Indica el número de Lados que posee el grafo.
- agregarVertice: (Grafo<V,L> g, Vértice<V> v) → `boolean`
  Agrega el vértice *v* previamente creado al grafo *g* previamente creado. Si en el grafo no hay vértice con el mismo identificador que el vértice *v*, entonces lo agrega al grafo y retorna `true`, de lo contrario retorna `false`.
- agregarVertice: (Grafo<V,L> g, `String` id, V dato, `double` p) → `boolean`
  crea un vértice con las características dadas y las agrega al grafo *g* previamente creado. Si en el grafo no hay vértice con el identificador *id*, entonces se crea un nuevo vértice y se agrega al grafo y se retorna `true`, de lo contrario retorna `false`.
- obtenerVertice : (Grafo<V,L> g, `String` id) → Vértice<V>
  Retorna el vértice contenido en el grafo que posee el identificador *id*. En caso que en el grafo no contenga ningún vértice con el identificador *id*, se lanza la excepción `NoSuchElementException`.
- estaVertice : (Grafo<V,L> g, `String` id) → `boolean`
  Se indica si un vértice con el identificador *id*, se encuentra o no en el grafo. Retorna `true` en caso de que el vértice pertenezca al grafo, `false` en caso contrario.
- estaLado : (Grafo<V,L> g, `String` u, `String` v) → `boolean`
  Determina si un lado pertenece a un grafo. La entrada son los identificadores de los vértices que son los extremos del lado. En caso de ser aplicada esta función con un grafo dirigido, se tiene que *u* corresponde al extermo inicial y *v* al extermo final.
- eliminarVertice: (Grafo<V,L> g, `String` id) → `Boolean`
  Elimina el vértice del grafo *g*. Si existe un vértice identificado con *id* y éste es eliminado exitosamente del grafo se retorna `true`, en caso contrario `false`.
- vertices: (Grafo<V,L> g) → Lista de Vertices<V>
  Retorna una lista con los vértices del grafo *g*.
- lados: (Grafo<V,L> g) → Lista de Lados<L>
  Retorna una lista con los lados del grafo *g*.
- grado: (Grafo<V,L> g, `String` id) → `entero`
  Calcula el grado del vértice identificado por *id* en el grafo *g*. En caso que en el grafo no contenga ningún vértice con el identificador *id*, se lanza la excepción `NoSuchElementException`.
- adyacentes: (Grafo<V,L> g, `String` id) → Lista de Vertices
  Obtiene los vértices adyacentes al vértice identicado por id en el grafo *g* y los retorna en una lista. En caso que en el grafo no contenga ningún vértice con el identificador *id*, se lanza la excepción `NoSuchElementException`.
- incidentes: (Grafo<V,L> g, `String` *id*) → Lista de Lados<L>
  Obtiene los lados incidentes al vértice identificado por *id* en el grafo *g* y los retorna en una lista. En

caso que en el grafo no contenga ningún vértice con el identificador *id*, se lanza la excepción `NoSuchElementException`.

- clone: (Grafo<V,L> g) → Grafo
  Retorna un nuevo grafo con la misma composición que el grafo de entrada.
- toString: (Grafo<V,L> g) → `String`
  Devuelve una representación del contenido del grafo como una cadena de caracteres.

## 2.4 El TAD Grafo No Dirigido<V,L>

Este TAD es una subtipo del TAD Grafo. Debe ser implementado como una clase concreta que implementa los métodos de la interfaz Grafo. El tipo de lado que con el que está constituido esta representación del TAD Grafo, es la Arista<L>. Adicionalmente posee las siguientes operaciones:

- crearGrafoNoDirigido<V,L>: () → GrafoNoDirigido<V,L>
  Crea un nuevo GrafoNoDirigido
- agregarArista : (Grafo<V,L> g, Arista<L> a) → `boolean`
  Agrega una nueva arista al grafo si el identificador de la arista no lo posee ninguna arista en el grafo. Retorna `true` en caso en que la inserción se lleve a cabo, `false` en contrario.
- agregarArista : (Grafo<V,L> g, `String` id, L dato, `double` p, `String` u, `String` v) → `boolean`
  Si el identificador id no lo posee ninguna arista en el grafo, crea una nueva arista y la agrega en el grafo. Retorna `true` en caso en que la inserción se lleve a cabo, `false` en contrario.
- eliminarArista : (Grafo<V,L> g, `String` id) → `boolean`
  Elimina la arista en el grafo que esté identificada con id. Se retorna `true` en caso que se haya eliminado la arista del grafo y `false` en caso de que no exista una arista con ese identificador en el grafo.
- obtenerArista : (Grafo<V,L> g, `String` id) → Arista<L>
  Devuelve la arista que tiene como identificador id. En caso de que no exista ninguna arista con ese identificador, se lanza la excepción NoSuchElementException.

## 2.5 El TAD Grafo Dirigido<V,L>

Este TAD es una subtipo del TAD Grafo. Debe ser implementado como una clase concreta que implementa los métodos de la interfaz Grafo. El tipo de lado que con el que está constituido el Digrafo, es el Arco. Adicionalmente posee las siguientes operaciones:

- crearGrafoDirigido<V,L>: () → GrafoDirigido<V,L>
  Crea un nuevo GrafoDirigido
- agregarArco : (Grafo<V,L> g, Arco<L> a) → `boolean`
  Agrega un nuevo arco al grafo si el identificador del arco no lo posee ningún arco en el grafo. Retorna `true` en caso en que la inserción se lleva a cabo, `false` en caso contrario.
- agregarArco : (Grafo<V,L> g, `String` id, L dato, `double` p, `String` vInicial, `String` vFinal) → `boolean`
  Si el identificador *id* no lo posee ningún arco en el grafo, crea un nuevo arco y lo agrega en el grafo. Retorna `true` en caso en que la inserción se lleva a cabo, `false` en contrario .
- eliminarArco : (Grafo<V,L> g, `String` id) → `boolean`
  Elimina el arco en el grafo que esté identificado con id. Se retorna true en caso que se haya eliminado el arco del grafo y false en caso que no exista un arco con ese identificador en el grafo.

- obtenerArco : (Grafo<V,L> g, `String` id) → Arco
  Devuelve el arco que tiene como identificador id. En caso de que no exista ningún arco con ese identificador, se lanza la excepción `NoSuchElementException`.
- gradoInterior : (Grafo<V,L> g, `String` id) → `entero`
  Calcula el grado interior del vértice identificado por id en el grafo. En caso de que no exista ningún vértice con ese identificador, se lanza la excepción `NoSuchElementException`.
- gradoExterior : (Grafo<V,L> g, `String` id) → `entero`
  Calcula el grado exterior del vértice identificado por id en el grafo. En caso de que no exista ningún vértice con ese identificador, se lanza la excepción `NoSuchElementException`.
- sucesores: (Grafo<V,L> g, `String` id) → Lista de Vértices<V>
  Devuelve una lista con los vértices que sucesores del vértice con identificador id. En caso de que no exista ningún vértice con ese identificador, se lanza la excepción `NoSuchElementException`.
- predecesores: (Grafo<V,L> g, `String` id) → Lista de Vértices<V>
  Devuelve una lista con los vértices predecesores del vértice con identificador id. En caso de que no exista ningún vértice con ese identificador, se lanza la excepción `NoSuchElementException`.

### 2.6 Programa Cliente

Para verificar el funcionamiento de la implementación del TAD GRAFO, debe desarrollar una pequeña aplicación que permita, por medio de un menú, tener acceso a todas las funciones del TAD. El archivo ClienteGrafo.java tiene como objetivo servir como un cliente en el cual se puedan probar las funcionalidades de los TADs.

## 3 Formato de Archivo

El formato del archivo que contiene los datos de un grafo es el siguiente:

```
V
L
O
n
m
v1 dv1 pv1
v2 dv2 pv2
⋮
vn dvn pvn
l1 dl1 pl1 vi1 vf1
l2 dl2 pl2 vi2 vf2
⋮
lm dlm plm vim vfm
```

donde

- V es el tipo de los datos almacenados en los vértices (B para Boolean, D para Double, S para String)
- L es el tipo de los datos almacenados en los lados (B para Boolean, D para Double, S para String)
- O es la orientación del grafo (D para dirigido, N para no dirigido)
- n es el número de vértices
- m es el número de lados
- $v_i$ es el identificador del vértice al que le corresponden los datos de esa línea

- $dv_i$ es el dato contenido en el vértice $v_i$
- $pv_i$ es el peso del vértice $v_i$
- $l_i$ es el identificador del lado al que le corresponden los datos de esa línea
- $dl_i$ es el dato del lado $l_i$
- $pl_i$ es el peso del lado $l_i$
- $vi_i$ es el vértice inicial del lado $l_i$
- $vf_i$ es el vértice final del lado $l_i$

# 4 Informe

Debe incluir un informe de 3 páginas en donde se expliquen sus decisiones de diseño y se indiquen los detalles más relevantes de la implementación realizada. Debe explicar cómo ejecutar su programa cliente y los casos de prueba probados.

# 5 Entrega

Sus implementaciones de los operadores deben ser razonablemente eficientes. Todo el código debe estar debidamente documentado. Se deben indicar una descripción del método, la descripción de los parámetros de entrada y salida, aplicando el estándar para la documentación de código en JAVA. Su implementación debe incluir manejo de excepciones. Puede usar las librerías de JAVA que considere útiles.

Debe entregar su código en un archivo comprimido (.zip, .tgz, etc.) libre de archivos intermedios o ejecutables. Deberá subirlo al Moodle de la materia en la sección marcada como "  Proyecto 1" hasta el jueves, 25 de octubre. Sólo deberá efectuar una entrega por grupo.

El día de la entrega del proyecto deberán entregar una "Declaración de autenticidad de la entrega" firmada por los autores del proyecto. El no cumplimiento de estos requisitos resultará en que su trabajo no sea evaluado.

# 6 Evaluación

En la evaluación del proyecto se tomará en cuenta aspectos como la documentación, el estilo de programación, la modularidad y mantenibilidad del código, la eficiencia en tiempo de ejecución y memoria, el uso de herencia, el manejo de excepciones, el buen uso de las librerías y la robustez. Usted debe realizar los casos de pruebas que muestren el correcto funcionamiento de las funciones implementadas.

Se asignarán

- 6 puntos por código (1 punto por cada clase)
- 9 puntos por ejecución (1,5 puntos por cada clase)
- 3 puntos por documentación (0,5 puntos por cada clase)
- 2 puntos por su informe