Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI-2693 - Laboratorio de Algoritmos y Estructuras III
Trimestre Septiembre-Diciembre 2021

Recorrido de grafos con DFS y BFS

1. Introducción

El principal objetivo de este laboratorio es la implementación de los algoritmos de recorrido de grafos, búsqueda en profundidad y búsqueda en amplitud. Como segundo objetivo es el de utilizar la implementación de estos algoritmos de recorrido de grafos, junto con la representación de grafos realizada anteriormente, para crear una librería de grafos. En las siguientes secciones se indican las actividades a realizar.

2. Actualización de la representación de grafos

Se quiere realizar una actualización de la representación de grafos para generalizarla y hacerla parte de una librería de grafos. En principio se crea una clase abstracta `Lado`, de la cual van a derivar las clases `Arista` y `Arco`. La Figura 1 muestra la jerarquía de clases. Observe que los métodos son ligeramente diferentes a los indicados en la primera versión de la representación de grafos.

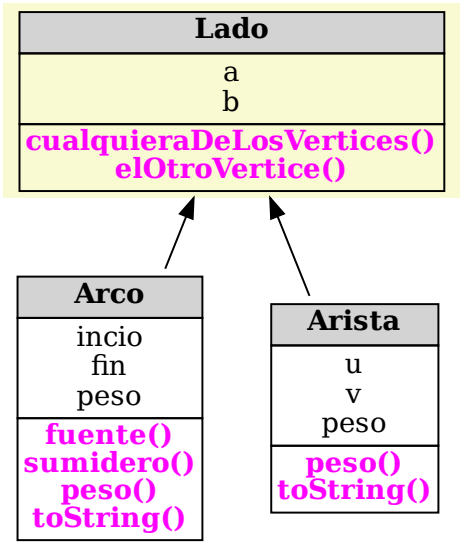

Figura 1: Jerarquía de las clases que representan a los lados en un Grafo. Para cada clase se indica los argumentos del constructor y sus métodos. El amarillo claro indica que la clase `Lado` es una clase abstracta.

Se agrega una interfaz `Grafo`, de la cual se derivan las clases `GrafoDirigido` y `GrafoNoDirigido`. La idea es que con una interfaz `Grafo`, podemos usar un objeto de tipo grafo en los algoritmos en los cuales el grafo de entrada puede un grafo no dirigido o digrafo. Ejemplo de este tipo de algoritmos son la búsqueda en profundidad (DFS) y la búsqueda en amplitud (BFS). La Figura 2 muestra la jerarquía de las clases que representan grafos.

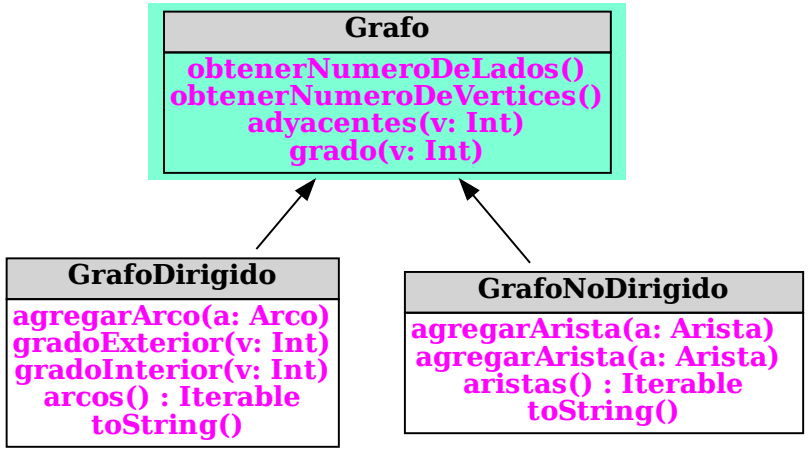

Figura 2: Jerarquía de las clases que representan a los Grafo. Para cada clase se indica sus métodos. La clase `Grafo` es una interfaz.

Debe realizar la modificación de la representación de grafos, para actualizarla a las nuevas clases incluidas.

3. Sobre DFS y BFS

Se quiere que realice una implementación de los algoritmos BFS y DFS, basada en el pseudo código visto clase de [1]. Ambos algoritmos se implementaran como clases. La idea que el algoritmo de recorrido, DFS o BFS, se ejecute en el instante en que se esté creando un nuevo objeto que sea una instancia de las clases grafos.

4. Librería de grafos

Con la representación de grafos y los algoritmos trabajados en esta clase, se quiere realizar una librería de grafos, que llamaremos *grafoLib*. La librería debe ser implementada como un paquete de Kotlin. La idea es que a medida que avance el curso se agreguen más algoritmos sobre grafos. El paquete se llama `ve.usb.grafoLib` por lo que todo el código de la librería debe estar contenido en el directorio *ve/usb/grafoLib*.

5. Aplicación de los algoritmos de recorridos de grafos

Se debe implementar un programa cliente que haga uso de los algoritmos DFS y BFS para resolver dos problemas. El cliente se llama `pruebaRecorridoGrafo` y se ejecuta mediante la siguiente línea de comandos: la siguiente línea de comando:

```
>./pruebaRecorridoGrafo [n|d] archivo_grafo
```

Donde:

`n:` indica que el archivo de entrada contiene un grafo no dirigido.

`d:` indica que el archivo de entrada contiene un grafo dirigido.

`archivo_grafo:` archivo con los datos de un grafo en el formato indicado en el laboratorio anterior.

Una llamada de línea de comando del programa que es válida, sería:

```
>./pruebaRecorridoGrafo d miDigrafo.txt
```

Los dos argumentos son obligatorios. La entrada por la línea de comando debe ser verificada y en caso de un error, se debe indicar apropiadamente al usuario. Al ejecutarse el programa `pruebaRecorridoGrafo` se deben obtener los siguientes dos resultados por la salida estándar:

1. Para cada par de vértices distintos del grafo, debe mostrar el camino de menos lados entre ambos vértices y se debe indicar la longitud de ese camino. El camino debe estar representado como una secuencia de vértices. Si el grafo es no dirigido, solo debe mostrar la mitad de los caminos posibles, debido a la simetría de los mismos.

2. Para cada uno de los lados de del grafo, debe indicar si son *Tree edges*, *Back edges*, *Forward edge* y *Cross edges*. Puede hacer uso del corolario 22.8 y el ejercicio 22.3-5 de [1].

6. Sobre la implementación

Se le proporcionará de un código base, contenido en el archivo `codigoBaselabSem3.tar.xz`. Este código usted lo debe completar y documentar. Puede hacer uso de las clases de la librería de Kotlin para su implementación. Cada una de las operaciones en las clases tiene una breve descripción la misma. Esa descripción debe borrada de su código de entrega. En su lugar deben colocar para cada una documentación de las operaciones en las que se indique *descripción*, *precondiciones*, *postcondiciones* y *tiempo de la operación*. El tiempo de las operaciones debe ser dado en número de lados y/o número de vértices, cuando eso sea posible. Los cambios hechos en la representación del grafo son menores, por lo que prácticamente la totalidad del código utilizado en el laboratorio anterior, puede ser utilizado en la librería de grafos. Sus implementaciones deber ser razonablemente eficientes, capaces de manejar grafos de millones de vértices y lados sin problemas.

7. Condiciones de entrega

Los códigos del laboratorio y la declaración de autenticidad debidamente firmada, deben estar contenidos en un archivo comprimido, con formato *tar.xz*, llamado *LabSem3_X.tar.xz*, donde *X* es el número de carné del estudiante. La entrega del archivo *LabSem3_X.tar.xz*, debe hacerse por medio de la plataforma *Classroom* antes de las 12:00 pm del día lunes 1 de noviembre de 2021.

Referencias

[1] Cormen, T., Leirserson, C., Rivest, R., and Stein, C. *Introduction to Algorithms*, 3ra ed. McGraw Hill, 2009.