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

Proyecto 1: Extensiones para `grafoLib`

1. Introducción

El objetivo de este proyecto es el de agregar nuevos algoritmos sobre grafos a la librería `grafoLib`. Esta extensión abarca nuevos algoritmos sobre digrafos, métricas sobre grafos dirigidos, resolución de un problema de satisfacibilidad booleana y obtención de ancestros comunes.

2. Actividades a realizar

En esta sección se presentan las actividades a realizar. Cada una de las actividades debe ser implementada como una clase en `grafoLib`. Por ello, se le proporcionará el código asociado a cada actividad. Cada uno de los archivos Kotlin tendrá el código base con el cual usted va a trabajar. También tendrá una descripción detallada de cada una de las clases y métodos. Esa descripción se omite en este documento, en el cual se describe la actividad a realizar de forma general.

2.1. Ciclo euleriano de un digrafo

Un ciclo euleriano de un digrafo G que está fuertemente conectado, es un ciclo en que atraviesa cada uno de los lados de G una vez. En el ciclo está permitido que un vértice sea visitado más de una vez. El ejercicio 22-3 de [2] muestra la descripción y características del ciclo euleriano. Se quiere que implemente un algoritmo que encuentre un ciclo euleriano en un digrafo G que está fuertemente conectado en un $O(|E|)$. La clase `CicloEuleriano` es la encargada de determinar si un digrafo tiene o no ciclo euleriano.

2.2. Métricas de un grafo no dirigido

Las métricas sobre los grafos no dirigidos permiten mostrar algunas de sus características. Se quiere implementar una clase llamada `MetricasDeGrafo` que computa las métricas de un grafo no dirigido conexo, que se describen a continuación.

Excentricidad: La excentricidad de un vértice s consiste en la longitud del camino más corto desde s hasta un vértice t, tal que t es el vértice con el camino más corto de mayor longitud desde s.

Diámetro: El diámetro de un grafo consiste en el mayor valor de excentricidad que se puede obtener de los vértices de un grafo.

Radio: El radio de un grafo consiste en el menor valor de excentricidad que se puede obtener de los vértices de un grafo.

Centro: El centro de un grafo es el vértice v con el que se obtiene el valor del radio de un grafo.

Índice *Wiener*: El índice *Wiener* de un grafo es la suma de todos los caminos más cortos entre todos los pares distintos de vértices de un grafo.

La clase `MetricasDeGrafo` y los métodos que representan a las métricas, se deben implementar en el archivo `MetricasDeGrafo.kt`.

2.3. Detección de grafo bipartito

Un grafo bipartito es un grafo no dirigido $G = (V, E)$, tal que el conjunto de vértices V puede ser divido en dos conjuntos V_1 y V_2, tal que para cualquier lado $(u, v) \in E$, se cumple que $u \in V_1 \wedge v \in V_2$ o se cumple que $v \in V_1 \wedge u \in V_2$ [2]. Es decir, los lados tienen un extremo en V_1 y otro V_2. La Figura 1 muestra un ejemplo de un grafo bipartito.

El coloreo de los vértices de un grafo consiste en la asignación de colores a los vértices de un grafo, de tal forma que los vértices adyacentes de un vértice v tengan colores diferentes a v. Formalmente, se define el k-coloring de un grafo no dirigido $G = (V, E)$ como la función $c : V \to \{1, 2, \ldots, k\}$ tal que $c(u) \neq c(v)$ para cada lado $(u, v) \in E$ [2]. Esto es, a cada uno de los vértices se le asigna uno de los colores de $\{1, 2, \ldots, k\}$ y los vértices adyacentes deben tener colores diferentes. Un grafo es 2-*coloreable* si a los vértices del grafo se le puede asignar dos colores. Se tiene que es equivalente que un grafo sea 2-*coloreable* y que sea un grafo bipartito [2]. La Figura 1 muestra un grafo bipartito con dos colores.

Se quiere que implemente un algoritmo, basado en búsqueda en profundidad, que determine de manera eficiente si un grafo es bipartito o no. La idea es que el algoritmo haga uso de la propiedad que indica que los grafos bipartitos son 2-*coloreables*. El archivo con la clase que implementa este algoritmo se llama `DosColoreable.kt`.

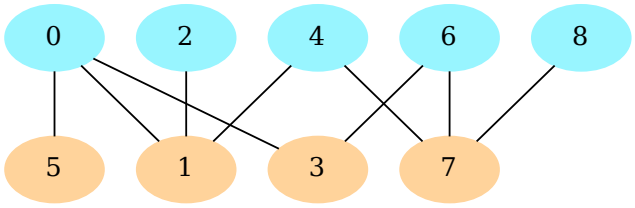

Figura 1: Ejemplo de un grafo bipartito coloreado con dos colores

2.4. Detección del ancestro común más bajo

El *ancestro común más bajo*, en inglés *lowest common ancestor* (LCA), de un par de vértices v y u, en un digrafo acíclico G (DAG), es un vértice s de G que es ancestro de u y v, tal que ninguno de sus descendientes es ancestro de u y v. En la Figura 2 se puede observar que de los vértices 0, 1, 2 y 5 son ancestros de 4 y 3, y de ellos el *ancestro común más bajo* es el vértice 5. Hay varias formas de obtener el LCA de un par de vértices. Aquí

sugerimos un algoritmo, usted es libre de aplicar otro, la única restricción es que el mismo debe ser correcto y eficiente. Se define la altura de un vértice v en un DAG, como el camino más largo desde alguno de los vértices fuentes s hasta el vértice v . Un vértice s es fuente, si su grado interior es cero. Dados dos vértices de un DAG, entonces el LCA de esos dos vértices, va a ser el vértice ancestro de ambos dos con mayor altura.

La detección del *ancestro común más bajo* de un par de vértices de un DAG se debe realizar en un archivo llamado `LCA.kt`.

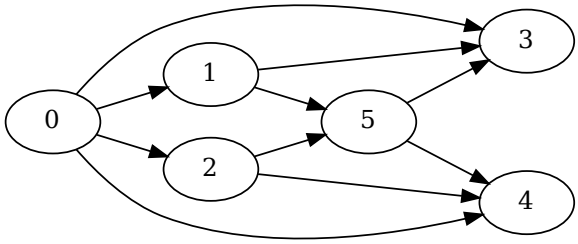

Figura 2: Digrafo acíclico en donde el ancestro común más bajo de los vértices 4 y 3 es el vértice 5.

2.5. Solucionador de 2-SAT

Se quiere que implemente un solucionador del problema 2-SAT. Primero definimos el problema de 2-SAT y luego se explica cual es el algoritmo que se debe aplicar para solucionar el problema en tiempo lineal.

Se define a un **literal** en una fórmula booleana, como la ocurrencia de una variable o su negación. Se dice que una fórmula booleana está en **forma normal conjuntiva**, en inglés *conjunctive normal form* (CNF), si está es expresada como una conjunción ($\wedge$) de cláusulas, en donde cada una de cláusulas es una disyunción ($\vee$) de uno o más literales. Una fórmula booleana está en **2-CNF**, si cada una de las cláusulas que la componen tiene exactamente dos literales diferentes. Por ejemplo, la siguiente fórmula booleana se encuentra en 2-CNF:

$$(x_0 \vee \neg x_1) \wedge (\neg x_0 \vee x_1) \wedge (\neg x_0 \vee \neg x_1) \wedge (x_0 \vee \neg x_2) \tag{1}$$

La primera de las cuatro cláusulas es $(x_0 \vee \neg x_1)$, la cual contiene dos literales: x_0 y $\neg x_1$.

El problema de **2-SAT** consiste en que dada una fórmula booleana en 2-CNF se quiere determinar si hay alguna asignación de valores para sus variables que hace que la expresión sea verdadera. La Fórmula 1 es verdadera si se hace la siguiente asignación a las variables: $x_0 = false$, $x_1 = false$ y $x_2 = false$.

Se tiene que el problema de **2-SAT** se puede resolver de forma eficiente. Para resolver el problema de 2-SAT, se va a utilizar el algoritmo presentado en [1], el cual vamos a describir. En principio se crea un *digrafo de implicación* $G = (V, E)$. Se tiene que el conjunto de vértices va a contener a todos los literales $V = \{x_0, \neg x_0, x_1, \neg x_1, \ldots, x_{n-1}, \neg x_{n-1}\}$. Entonces, $|V| = 2n$. Por cada cláusula $l_i \vee l_j$, donde l es una variable o su negación, se van agregar dos lados a G, estos son $\neg l_i \to l_j$ y $\neg l_j \to l_i$. Por ejemplo, para construir el digrafo de implicación de la Fórmula 1, se tiene agregar los siguientes lados por cada cláusula:

- Por $(x_0 \vee \neg x_1)$ se agregan los lados $\neg x_0 \to \neg x_1$ y $x_1 \to x_0$
- Por $(\neg x_0 \vee x_1)$ se agregan los lados $x_0 \to x_1$ y $\neg x_1 \to \neg x_0$
- Por $(\neg x_0 \vee \neg x_1)$ se agregan los lados $x_0 \to \neg x_1$ y $x_1 \to \neg x_0$
- Por $(x_0 \vee \neg x_2)$ se agregan los lados $\neg x_0 \to \neg x_2$ y $x_2 \to x_0$

La Figura 3 muestra el digrafo de implicación de la Fórmula 1. Se puede interpretar cada lado $l_i \to l_j$ como si se tiene que $l_i = true$, entonces debemos tener que $l_j = true$ en cualquier asignación que haga verdadera la fórmula.

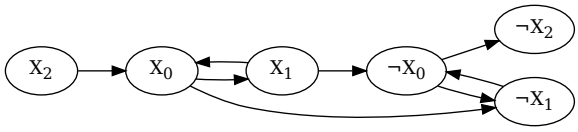

Figura 3: *Digrafo de implicación* que representa la fórmula booleana 1

Una vez construido el grafo de implicación, el próximo paso del algoritmo es el obtener las componentes fuertemente conexas del grafo. La Figura 4 muestra las componentes fuertemente conexas del digrafo de implicación de la Figura 3.

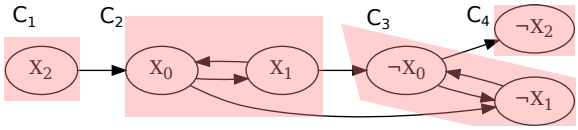

Figura 4: *Digrafo de implicación* con sus componentes fuertemente conexas

Sea C una función que asigna a un literal de la fórmula booleana, su componente fuertemente conexa. Si ocurre que si para una variable x_i, se tiene que los literales x_i y $\neg x_i$ están en la misma componente fuertemente conexa, es decir $C(x_i) = C(\neg x_i)$, entonces **no hay ninguna asignación que haga verdadera la fórmula**. En caso contrario entonces buscamos una **asignación que haga verdadera la fórmula**. Para ello construimos el grafo componente asociado a las componentes fuertemente conexas. El grafo componente es un DAG. La Figura 5 muestra el grafo componente resultante de la Figura 4. Luego se aplica un ordenamiento topológico al grafo componente, usando la relación de orden parcial $<$. Si aplicamos el ordenamiento topológico al grafo componente de la Figura 5, tenemos el orden $C_1 < C_2 < C_3 < C_4$. Si hay un camino desde x_i hasta x_j, entonces $C(x_i) \leq C(x_j)$. Entonces en caso de haber una **asignación que haga verdadera la fórmula**, se cumple para todas las variables x_i de la fórmula que $C(x_i) \neq C(\neg x_i)$. La asignación de las variables se realiza de la siguiente manera:

- si $C(\neg x_i) < C(x_i)$, entonces $x_i = true$
- si $C(x_i) < C(\neg x_i)$, entonces $x_i = false$

En el ejemplo, tenemos que buscamos las asignaciones de las variables:

- se tiene que $C(x_0) < C(\neg x_0)$, entonces $x_0 = false$
- se tiene que $C(x_1) < C(\neg x_1)$, entonces $x_1 = false$
- se tiene que $C(x_2) < C(\neg x_2)$, entonces $x_2 = false$

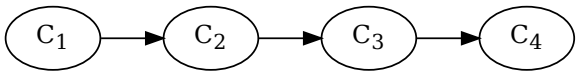

Figura 5: Grafo componente asociado a las componentes fuertemente conexas de la Figura 4

En [3] en la sección *Strongly connected components*, se presenta el algoritmo aquí explicado para resolver 2-SAT.

El programa solucionador de 2-SAT debe implementarse en una clase llamada `Sol2SAT.kt`. La clase `Sol2SAT` recibe en constructor un archivo con la fórmula 2-SAT. El formato del archivo de entrada es como sigue. Cada línea corresponde a una cláusula. Los literales x_i estarán identificados con el número i correspondiente a la variable. Si el literal está negado entonces el número será negativo. Los literales estarán separados por un espacio en blanco. Por ejemplo, el archivo que contendría la Fórmula de 1 es como sigue:

```
0 -1
-0 1
-0 -1
0 -2
```

2.6. Detalles de la implementación

Se le proporcionará de un código base, contenido en el archivo `codigoBaseProy.tar.xz`. Este código contiene los archivos adicionales que usted debe agregar a la librería *grafoLib*. Se debe completar y documentar el código de las actividades a realizar. 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 ser 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. Sus implementaciones deber ser razonablemente eficientes. Debe entregar la librería *grafoLib* completa, con todos los códigos de las implementaciones de este proyecto y de las semanas anteriores.

Debe realizar un programa cliente llamado `Main.kt` que muestre el correcto funcionamiento de las clases requeridas en el proyecto. También de realizar un archivo makefile, llamado `Makefile`, que compila la librería completa y al programa cliente. Debe crear un *script*, llamado `pruebaProyecto1.sh` para ejecutar el programa cliente sin recibir argumentos de entrada.

Si un proyecto que *no pueda compilarse o ejecutarse tiene cero de nota*. La plataforma en la que debe ejecutarse el proyecto es Linux.

3. Condiciones de entrega

Los códigos del proyecto y la declaración de autenticidad debidamente firmada, deben estar contenidos en un archivo comprimido, con formato *tar.xz*, llamado *Proy1_X_Y.tar.xz*, donde X y Y son los número de carné de los estudiantes. La entrega del archivo *Proy1_X_Y.tar.xz*, debe hacerse por medio de la plataforma *Classroom* antes de las 12:00 pm del día lunes 29 de noviembre de 2021.

Referencias

[1] Aspvall, B., Plass, M. F., and Tarjan, R. E. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information processing letters 8*, 3 (1979), 121–123.

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

[3] Wikipedia contributors. 2-satisfiability — Wikipedia, the free encyclopedia, 2021. [Online; accessed 18-November-2021].

Guillermo Palma / gvpalma@usb.ve / 19 de noviembre de 2021