Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI2693 - Laboratorio de Algoritmos y Estructuras III
Trimestre Abril-Julio 2022

Proyecto 1

1. Introducción

El proyecto está compuesto por tres actividades. La primera actividad consiste en determinar *el grado de separación* de dos objetos en una red. La segunda actividad trata sobre encontrar la *menor distancia de edición* entre dos palabras, dado un diccionario. Para la tercera y última actividad debe realizar un *solucionador* para el *problema de 2-SAT*. Se quiere que para resolver los problemas, modele los mismos como grafos y haga uso de los algoritmos de *libGrafoKt*.

2. El problema del *grado de separación*

Se quiere resolver el **problema del grado de separación**, el cual consiste en encontrar **el menor número de conexiones** entre dos entes en una red. Este problema esta basado en el principio que se conoce como *seis grafos de separación* [1], que indica todas las personas están separadas por seis conexiones o menos. La idea es que la distancia entre dos entes, como personas, es logarítmica con respecto al tamaño de la población. Si no hay conexión entre dos entes, entonces el grado de separación es cero.

Para este proyecto la red estará representada por un grafo no dirigido y los entes serán vértices de un grafo. La Figura 1 muestra una red con algunos de los aeropuertos de Venezuela y parte de sus conexiones. En esta figura se tiene que entre el aeropuerto de Maiquetía (CCS) y el Porlamar (PMV) hay un grado de separación. Se tiene que el grado de separación entre el aeropuerto de Porlamar (PMV) y el de Valencia (VLN) es 2, y viene dado por el camino $PMV - CCS - VLN$. En este caso la red de estudio es un grafo no dirigido donde los vértices son todos del mismo tipo.

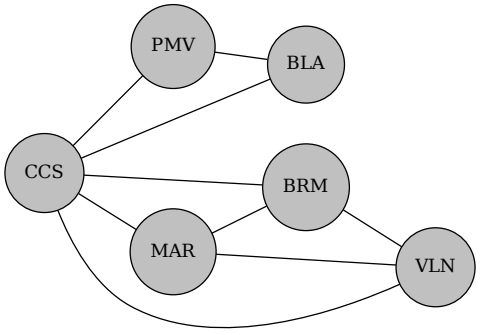

Figura 1: Grafo que representa a algunos de los aeropuertos de Venezuela y sus conexiones (ficticias).

Hay otro caso interesante de estudiar y es donde la red es un grafo bipartito. Es decir, hay dos tipos de entes y se quiere saber el grado de separación entre dos entes de un mismo tipo. Un ejemplo de este tipo de redes, es una red que conecta a actores y películas. En este caso, se quiere saber el grado de separación entre dos actores. La Figura 2 muestra una red, con un grafo bipartito, en donde hay dos tipos de entes, actores y películas. Cada película conecta con los actores que trabajan en esa película. Se quiere conocer el grado de separación entre dos actores. Un ejemplo de este tipo de problema, es el de los seis grados de separación de Kevin Bacon [2]. Siendo Kevin Bacon un actor que ha trabajo en un gran número de películas, es de interés saber el grado de separación entre Kevin Bacon y otros actores. En el caso de la Figura 2 se tiene que el grado de separación de Kevin Bacon y de Amber Heard es 2 y viene dado por el camino *Kevin Bacon - [Diner] - Mickey Rourke - [The Informers] - Amber Heard*. Asimismo, el grado de separación de Kevin Bacon y de Eliane Chipia es 0.

Por último, puede observar que el grado de separación de Kevin Bacon y de G. Marimuthu es 3, que es dado por el camino *G. Marimuthu - [Kootathil Oruthan] - Samuthirakani - [RRR] - Ray Stevenson - [Jayne Mansfield's Car] - Kevin Bacon*. Otro tipo de red, representada como grafo bipartito, es la de coautores y artículos. Este tipo de redes son las utilizadas para obtener el número de Erdös de un autor [3]. El número de Erdös de un autor consiste en el número de conexiones que separa al autor de un artículo científico con el prolífico y destacado matemático Paul Erdös.

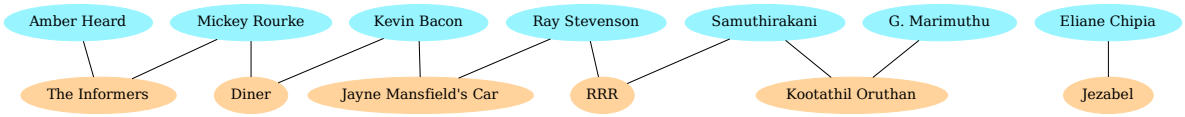

Figura 2: Grafo que representa una red de actores y las películas en que han trabajado. Los actores son los vértices en color azul y las películas son los de color crema. Fuente: [4] y [5].

La solución del problema debe hacerse en un programa cliente llamada `GradosDeSeparacion.kt`. Este programa debe ejecutarse por medio de un *shell script*, llamado `runGradosDeSeparacion.sh`. El comando de ejecución es el siguiente:

```
>./runGradosDeSeparacion.sh [-b] <archivoGrafo> <ente1> <ente2>
```

Donde:

-b : indica que el grafo es bipartito. Este indicador es opcional, si no se usa se asume que el grafo tiene cualquier topología.

archivoGrafo : Archivo que contiene los datos de un grafo con nombres en los vértices. El formato del archivo es el indicado en el laboratorio de la semana 5.

ente1 : Consiste en el primer vértice a determinar el grado de separación.

ente2 : Consiste en el segundo vértice a determinar el grado de separación.

Si el indicador `-b` es usado, entonces se deben verificar dos condiciones de la entrada. Lo primero es verificar si el grafo dado en `archivoGrafo` es bipartito. Si el grafo no es bipartito se debe indicar esto con un mensaje de error al usuario y se debe terminar la ejecución del

programa. La segunda condición es que los vértices indicados en `ente1` y `ente2` sean del mismo tipo, es decir, que pertenezcan al mismo grupo de la partición de los vértices de grafo bipartito. En caso de no ser del mismo tipo, entonces se indica el error al usuario y se termina la ejecución del programa. Para cualquier tipo de grafo, si algunos de los vértices indicados en la entrada no pertenece al grafo, entonces se debe dar un mensaje de error al usuario y terminar la ejecución del programa.

La salida del programa consiste en dos líneas. La primea línea muestra la conexión más corta entre los dos entes, separadas por un espacio en blanco, un guión y un espacio en blanco. Luego en la segunda línea se indica el grado de separación entre los dos entes. Si alguno de los dos entes de la entrada, no corresponde a ningún vértice del grafo, entonces se debe indicar este error al usuario y terminar la ejecución del programa.

2.1. Ejemplos de entrada y salida de la actividad 1

El primer ejemplo consiste en tener como entrada un grafo como el indicado en la Figura 1, en un archivo llamado `aeropuertos.txt`. Luego queremos saber el grado de separación entre el aeropuerto de Porlamar y el de Valencia. La línea de comando a ejecutar es:

```
>./runGradosDeSeparacion.sh aeropuertos.txt PMV VLN
```

Se debe mostrar por la salida estándar el siguiente resultado:

```
PMV - CCS - VLN
2
```

Como segundo ejemplo, dado el grafo de la Figura 2, contenido en el archivo `peliculas.txt`, se tiene que una llamada válida es la siguiente:

```
>./runGradosDeSeparacion.sh -b peliculas.txt "Kevin Bacon" "Amber Heard"
```

Se debe obtener la siguiente salida:

```
Kevin Bacon - Diner - Mickey Rourke - The Informers - Amber Heard
2
```

3. El problema de la *escalera de edición*

Este problema *está basado en un problema de la ICPC* [6]. Se define **un paso de edición**, como la transformación de una palabra w_1 en una palabra w_2, por cambiar, eliminar o agregar una letra a w_1 para convertirla en w_2. Por ejemplo, la palabra *carro* se puede transforma en *caro* por la eliminación de la letra r, por lo tanto entre las dos palabras hay un paso de edición. En otro ejemplo, entre la palabra *paro* y *paso* hay un paso de edición.

Se define una **escalera de edición** como una secuencia de palabras $w_1, w_2, \ldots, w_{n-1}, w_n$ ordenadas lexicográficamente, tal que entre cada palabra w_i y w_{i+1}, hay un paso de edición, para todo i, desde 1 hasta $n-1$.

El problema consiste en que dado un diccionario, se quiere encontrar la más larga escalera de edición posible. Para la solución del problema debe implementar un programa cliente llamada `EscaleraEdicion.kt`. Este programa debe ejecutarse con un *shell script*, llamado `runEscaleraEdicion.sh`. La línea de comando que la ejecuta es la siguiente:

```
>./runEscaleraEdicion.sh <archivoEntrada>
```

Donde `archivoEntrada` es el nombre del archivo con los datos de entrada. La entrada consiste en un diccionario de palabras, las cuales están ordenadas lexicográfimanete, y en donde hay una palabra por línea.

La salida consiste dos líneas. La primera línea muestra la **escalera de edición** más larga que se puede construir con las palabras del diccionario, contenido del archivo de entrada. Cada palabra debe estar separada por un espacio en blanco. La segunda línea indica el número de palabras de la **escalera de edición**.

3.1. Ejemplo de entrada y salida de la actividad 2

Se tiene como entrada un archivo llamado `miDiccionario.txt`, cuyo contenido se muestra a continuación:

```
ama
amas
amo
amor
ano
aro
caro
carro
cerro
mama
pare
paro
pero
perro
raro
```

Se tiene que se debe ejecutar el comando:

```
>./runEscaleraEdicion.sh miDiccionario.txt
```

La salida correcta es la siguiente:

```
ama amo ano aro paro raro
6
```

4. 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 [7], 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.

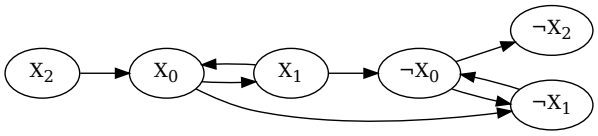

Figura 3: *Digrafo de implicación* que representa la fórmula booleana 1

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.

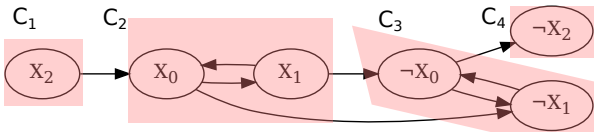

Figura 4: *Digrafo de implicación* con sus componentes fuertemente conexas

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 $x_2 = false$

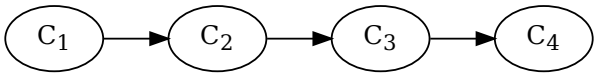

Figura 5: Grafo componente asociado a las componentes fuertemente conexas de la Figura 4

En [8] en la sección *Strongly connected components*, se presenta el algoritmo aquí explicado para resolver 2-SAT.

El solucionador de 2-SAT debe implementarse en programa cliente llamado `TwoSATSolver.kt`. Este programa debe ejecutarse por medio de un *shell script*, llamado `runTwoSATSolver.sh`. La línea de comando de ejecución es la siguiente:

```
>./runTwoSATSolver.sh <archivoEntrada>
```

Donde `archivoEntrada` es el nombre del archivo con los datos de entrada. 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.

La salida es como sigue. Si no existe una asignación que haga cierta la fórmula, entonces se retorna la frase *"No hay asignación válida"*. En caso contrario, se retorna la asignación de los literales $x_0, x_1, \ldots, x_{n-1}$, cada una separada por un espacio en blanco, usando las palabras $TRUE$ o $FALSE$, para indicar el valor dado al literal.

4.1. Ejemplo de entrada y salida de la actividad 3

Sea un archivo llamado `formula.txt` que contendría la Fórmula 1, el mismo tiene el siguiente contenido:

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

Si se ejecuta:

```
>./runTwoSATSolver.sh formula.txt
```

La salida esperada es la siguiente:

```
FALSE FALSE FALSE
```

5. Detalles de la implementación

La idea es que para implementar la solución de las actividades, debe modelar el problema como un grafo y utilizar los algoritmos sobre grafo vistos en clase, para su solución. Los clientes de las actividades deben hacer uso de la librería *libGrafoKt*. La librería puede modificarse y/o ampliarse para poder resolver los problemas planteados. Debe crear una carpeta llamada *proyecto1_X_Y*, donde donde X y Y son los carné son los integrantes del equipo. Esta carpeta debe contener tres directorios llamadas *actividad1*, *actividad2* y *actividad3*. Dentro de cada una de estos directorios debe estar el código de la solución de cada una de las actividades. Como ya se indicó, para cada una de las actividades hay que realizar un programa cliente. En un directorio de una actividad, debe estar el programa cliente, el archivo `Makefile`, el *shell script* ejecutable, la librería *libGrafoKt* y otros archivos de soporte que usted haya implementado.

Puede hacer uso de las clases de la librería de Kotlin para su implementación. Todo el código debe estar debidamente documentado. Al comienzo de los archivos `GradosDeSeparacion.kt`, `EscaleraEdicion.kt` y `TwoSATSolver.kt`, debe presentar una explicación detallada de como la actividad planteada fue resuelta. Debe indicar las estructuras de datos y los algoritmos usados para su solución. Sin esta explicación el código de la actividad puede ser invalidado. La documentación de las operaciones implementadas debe incluir *descripción*, *parámetros* 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.

Los tres programas de las actividades, reciben como una de sus entradas un archivo de datos. Debe tener en cuenta que lo que se quiere es que su programa pueda recibir sin problema el camino, en el sistema de archivos, hasta el archivo con los datos.

Si una actividad no puede compilarse o ejecutarse tiene cero en la nota de la actividad. La plataforma en la que debe ejecutarse el proyecto es GNU/Linux.

6. 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 *proyecto1_X_Y.tar.xz*, donde X y Y son los número de carné de los estudiantes. La entrega del archivo *proyecto1_X_Y.tar.xz*, debe hacerse por medio de la plataforma *Classroom* antes de las 11:59 P.M. del día domingo 10 de julio de 2022.

Referencias

[1] Wikipedia contributors, "Six degrees of separation — Wikipedia, the free encyclopedia," 2022, [Online; accessed 24-Junio-2021]. [Online]. Available: https://en.wikipedia.org/wiki/Six_degrees_of_separation

[2] ——, "Six degrees of kevin bacon — Wikipedia, the free encyclopedia," 2022, [Online; accessed 24-Junio-2021]. [Online]. Available: https://en.wikipedia.org/wiki/Six_Degrees_of_Kevin_Bacon

[3] ——, "Erdos number — Wikipedia, the free encyclopedia," 2022, [Online; accessed 24-Junio-2021]. [Online]. Available: https://en.wikipedia.org/wiki/Erd%C5%91s_number

[4] The oracle of Bacon contributors, "The oracle of bacon," 2022, [Online; accessed 24-Junio-2021]. [Online]. Available: http://oracleofbacon.org/movielinks.php

[5] IMDB, "Jezabel," 2022, [Online; accessed 24-Junio-2021]. [Online]. Available: https://www.imdb.com/title/tt9339850/

[6] International Collegiate Programming Contest, "Past problems," 2022, [Online; accessed 24-Junio-2021]. [Online]. Available: https://icpc.global/worldfinals/problems

[7] B. Aspvall, M. F. Plass, and R. E. Tarjan, "A linear-time algorithm for testing the truth of certain quantified boolean formulas," *Information processing letters*, vol. 8, no. 3, pp. 121–123, 1979.

[8] Wikipedia contributors, "2-satisfiability — Wikipedia, the free encyclopedia," 2021, [Online; accessed 18-November-2021]. [Online]. Available: https://en.wikipedia.org/wiki/2-satisfiability