

Laboratorio Nro. 1

Implementación y Recorrido de Grafos

Objetivos

1. Entender la implementación de los grafos dirigidos
2. Entender el concepto de sucesor (*successor*) como vecino, es decir, un nodo que es vecino de otro, un nodo que está conectado a otro por un arco
Entender los algoritmos de Búsqueda de Profundidad (*Deep-First Search -DFS*) y Búsqueda en Amplitud (*Breath-First Search -BFS*)
3. Entender en el contexto de la vida real, cuándo usar, cómo y en qué casos *BFS* y *DFS*

Consideraciones iniciales

Leer la Guía



Antes de comenzar a resolver el presente laboratorio, leer la **“Guía Metodológica para la realización y entrega de laboratorios de Estructura de Datos y Algoritmos”** que les orientará sobre los requisitos de entrega para este y todos los laboratorios, las rúbricas de calificación, el desarrollo de procedimientos, entre otros aspectos importantes.

Registrar Reclamos



En caso de tener **algún comentario** sobre la nota recibida en este u otro laboratorio, pueden **enviarlo** a través de <http://bit.ly/2q4TTKf>, el cual será atendido en la menor brevedad posible.

Traducción de Ejercicios

En el GitHub del docente, encontrarán la traducción al español de los enunciados de los Ejercicios en Línea.

**Visualización de
Calificaciones**

A través de **Eafit Interactiva** encontrarán **un enlace** que les permitirá **ver un registro de las calificaciones** que **emite el docente** para cada taller de laboratorio y según las rubricas expuestas. **Véase sección 3, numeral 3.8.**

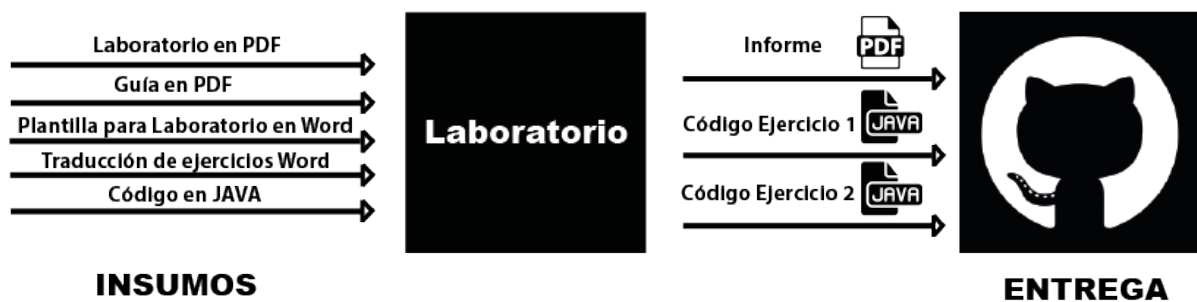
GitHub

1. Crear un repositorio en su cuenta de GitHub con el nombre `st0247-suCodigoAqui`. 2. Crear una carpeta dentro de ese repositorio con el nombre `laboratorios`. 3. Dentro de la carpeta `laboratorio`, crear una carpeta con nombre `lab01`. 4. Dentro de la carpeta `lab01`, crear tres carpetas: `informe`, `codigo` y `ejercicioEnLinea`. 5. Subir el informe pdf a la carpeta `infome`, el código del ejercicio 1 a la carpeta `codigo` y el código del ejercicio en línea a la carpeta `ejercicioEnLinea`. Así:

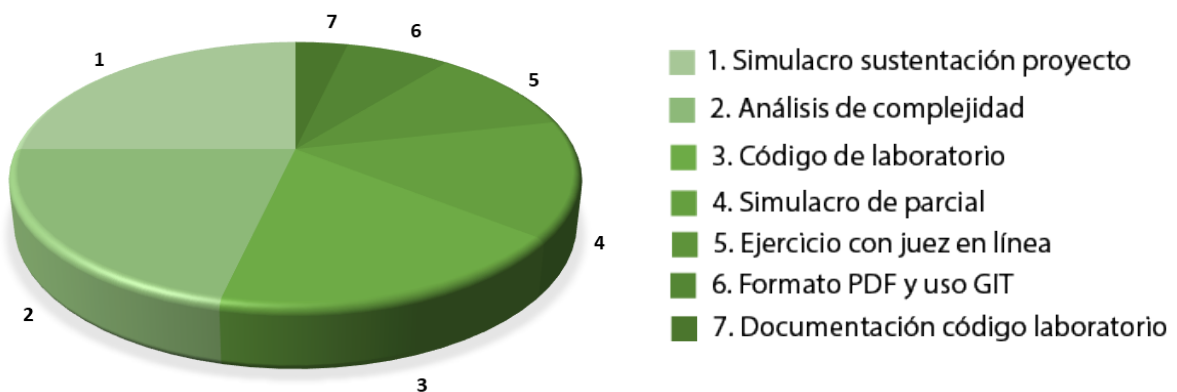
```
st0247-suCodigoAqui
  laboratorios
    lab01
      informe
      codigo
      ejercicioEnLinea
    lab02
    ...
```

Intercambio de archivos

Los archivos que **ustedes deben entregar** al docente son: **un archivo PDF** con el informe de laboratorio usando la plantilla definida, y **dos códigos**, uno con la solución al numeral 1 y otro al numeral 2 del presente. Todo lo anterior se entrega en **GitHub**.



Porcentajes y criterios de evaluación para el laboratorio



Resolver Ejercicios

1. Códigos para entregar en GitHub:



En la vida real, la documentación del software hace parte de muchos estándares de calidad como CMMI e ISO/IEC 9126



Véase Guía *en Sección 3, numeral 3.4*



Código de laboratorio *en GitHub*. Véase Guía en *Sección 4, numeral 4.24*



Documentación *en HTML*



No se reciben archivos *en .RAR ni en .ZIP*



En la vida real, los grafos se utilizan para representar redes sociales como *Facebook*, sistemas de información geográfica como *Google Earth* o enrutadores, como un enrutador ISR 4000 de Cisco

1.1 Teniendo en cuenta lo anterior:

- a) Realicen una implementación de la clase abstracta *Digraph*, llámela *DigraphAM* e implementen grafos con la estructura de datos Matrices de Adyacencia Etiquetadas



PISTA: Un error común es retornar el peso de los arcos en lugar de los identificadores de los vértices en el método *getSuccessors*

- b) Posteriormente, creen la clase *DigraphAL* e implementen grafos con la estructura de datos Listas de Adyacencia. Ambas clases heredan de la clase abstracta *Digraph* (digrafo o grafo dirigido)



PISTA: Un error común es retornar el peso de los arcos en lugar de los identificadores de los vértices en el método *getSuccessors*



PISTA 2: Véase *Guía en Sección 4, numeral 4.8 “Cómo definir una clase Pareja en Java”*

1.2 En la clase *GraphAlgorithms*, implementen un método que reciba como parámetro un grafo dirigido y que retorne cuál es el vértice que tiene más sucesores (vecinos). Debe funcionar para ambas implementaciones de grafo.

1.3 Prueben su código con los ejemplos contruidos en los numerales 1.1 y 1.2 para el *Algoritmo de Dijkstra*. Deben obtener la misma respuesta con ambas implementaciones.



PISTA: Véase *Guía en Sección 4, numeral 4.14 “Cómo hacer pruebas unitarias en BlueJ usando JUnit” y numeral 4.15 “Cómo compilar pruebas unitarias en Eclipse”*



NOTA: Todos los ejercicios del numeral 1 deben ser documentados en formato HTML. Véase *Guía en Sección 4, numeral 4.1 “Cómo escribir la documentación HTML de un código usando JavaDoc”*



En la vida real, una aplicación de los grafos es para describir mapas, como los usados por Google Maps. El archivo *medellin_colombia-grande.txt* que está en el ZIP que el docente les entregó, contiene un grafo que representa todas las calles de Medellín, es un grafo de aproximadamente 300.000 nodos

1.4 Teniendo en cuenta lo anterior, implementen un método que permita crear un grafo a partir de ese archivo del texto *medellin_colombia-grande.txt*



PISTA: Véase Guía en **Sección 4, numeral 4.13** “Cómo usar Scanner o *BufferedReader*”



PISTA 2: Hay información que sobra, por ejemplo, la latitud y la longitud de cada vértice y el nombre de cada arista.



PISTA 3: Es mejor usar *BufferedReader* porque es más rápido que *Scanner*. La idea es leer en una cadena de caracteres el contenido de cada línea y usando el método *split* de la clase *String* o usando *StringTokenizer*, dividir la cadena en partes cada que hay una coma (,).



PISTA 4: Como los códigos no son secuenciales, es decir, no empiezan en cero y tampoco están todos los números consecutivos, una forma de manejar los vértices es usar un mapa (en Java, *HashMap* o *TreeMap*).



PISTA 5: En las diapositivas de la clase “**Data Structures II: Graph Transversals**” encontrará los algoritmos y el recorrido *Deep-First Search*, y en *Eafit Interactiva*, las implementaciones de grafos no dirigidos usando matrices de adyacencia y listas de adyacencia

Error Común



En la vida real, la búsqueda en amplitud es usada para encontrar patrones en redes sociales, como por ejemplo Facebook, para recomendar nuevos amigos a sus usuarios.

1.5 Teniendo en cuenta lo anterior:

Implementen en un método el algoritmo de *BFS*, de tal forma de que funcione tanto para la implementación de grafos que utiliza matrices de adyacencia como para la implementación que usa listas de adyacencia, es decir, que reciba un objeto de la clase *Graph*, así como se hizo para *DFS*. El algoritmo debe retornar un *ArrayList* con los vértices en el orden en que los recorrió



En la vida real, el trabajo de *testing* es uno de los mejor remunerados y corresponde a un Ingeniero de Sistemas

1.6 Teniendo en cuenta lo anterior construyan ejemplos usando *JUnit* para probar su implementación de *BFS*

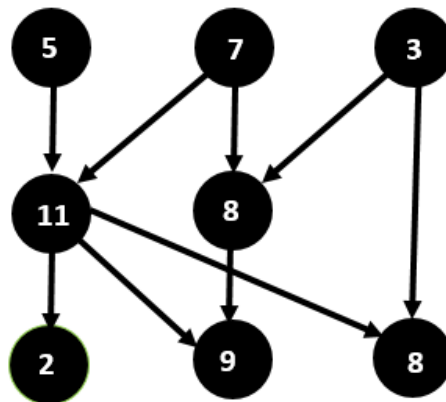
PISTA: Véase *Guía en Sección 4, numeral 4.14 “Cómo hacer pruebas unitarias en BlueJ usando JUnit” y numeral 4.15 “Cómo compilar pruebas unitarias en Eclipse*



PISTA 2: Como un ejemplo, construyan el grafo que hicimos en el taller en clase y comprueben que su algoritmo sí arroja la respuesta correcta. Si utilizan *Python* o *C++*, usen una librería para test unitarios disponible en esos lenguajes.



PISTA 3: Como un ejemplo, si se llama el método *BFS* con este grafo y con el vértice 7, el *ArrayList* que retorna debe ser así: [7, 8, 11, 2, 9, 10]



PISTA 4: Usen el método *AssertArrayEquals* de *JUnit* que encuentran en <http://junit.sourceforge.net/javadoc/org/junit/Assert.html>

1.7 Escriban una explicación entre 3 y 6 líneas de texto del código del numeral 1.1., es decir, digan cómo funciona y cómo está implementado



En la vida real, es importante saber si un grafo tiene o no ciclos porque muchos algoritmos sólo funcionan o sólo son eficientes cuando no hay ciclos

1.8 Teniendo en cuenta lo anterior implementen un método para un grafo que diga si un grafo tiene ciclos o no



NOTA: Todos los ejercicios del numeral 1 deben ser documentados en formato HTML. Véase *Guía en Sección 4, numeral 4.1 “Cómo escribir la documentación HTML de un código usando JavaDoc”*

2) Ejercicios en línea sin documentación HTML en GitHub



Véase Guía en **Sección 3, numeral 3.3**



No entregar
documentación **HTML**



Entregar un archivo
en **.JAVA**



No se reciben archivos
en **.PDF**



Resolver los problemas
de **CodingBat** usando
Recursión



Código del ejercicio en línea
en **GitHub**. Véase Guía en
Sección 4, numeral 4.24



NOTA: Recuerden que, si toman la respuesta de alguna fuente, deben referenciar según el tipo de cita correspondiente. Véase *Guía en Sección 4, numerales 4.16 y 4.17*

2.1 Resuelvan el siguiente ejercicio:

En 1976, el teorema de colorear un mapa con 4 colores fue probado con la ayuda de un computador. Este teorema muestra que un mapa puede ser coloreado solamente con 4 colores, de tal forma que no haya una región coloreada usando el mismo color que un vecino. Aquí hay un problema similar a ese problema, pero es mucho más simple.

Usted tiene que decidir si dado un grafo conexo arbitrario, ese grafo se puede colorear con 2 colores. Esto quiere decir, si uno puede asignar colores (de una paleta de 2 colores) a los nodos, de tal forma que no haya 2 nodos adyacentes del mismo color. Para simplificar el problema usted puede asumir que:

1. No hay un nodo que tenga un arco a sí mismo
2. El grafo es no dirigido, es decir que, si un nodo a está conectado a un nodo b , usted puede asumir que el nodo b también está conectado al nodo a .
3. El grafo será fuertemente conexo. Esto quiere decir, que hay al menos un camino de un nodo de grafo a cualquier otro nodo.

Entrada

La entrada consiste en varios casos de prueba. Cada caso de prueba comienza con una línea que tiene un número n ($1 < n < 200$) de nodos diferentes. La siguiente línea contiene el número de arcos.

Posteriormente, las siguientes líneas, cada una contiene 2 número que especifican que existe un arco entre dos nodos.

Un nodo en el grafo se representa con un número a ($0 < a < n$). Una entrada con $n = 0$ simboliza el fin de la entrada y no debe ser procesada.

Salida

Usted tiene que decidir si el grafo de entrada puede ser coloreado con dos colores o no, y debe imprimirlo como se muestra a continuación.

Entrada de los ejemplos

```
3
3
0 1
1 2
2 0
3
2
0 1
1 2
9
8
0 1
0 2
0 3
0 4
0 5
0 6
0 7
0 8
0
```

Salida de los ejemplos

```
NOT BICOLORABLE.
BICOLORABLE.
BICOLORABLE.
```



PISTA 1: Usen un algoritmo para corroborar si es un grafo bipartito. Léase qué es un grafo bipartito en <http://bit.ly/2hGwAo2>



PISTA 2: Si desean, pueden usar DFS o BFS para resolver este problema, pero existe otro tipo de algoritmos para resolverlo también.



PISTA 3: Spoiler Alert! En este sitio web explican un algoritmo para verificar si un grafo es bipartito <http://bit.ly/2IOsQFZ>



Entregar un archivo
en **.JAVA**



O entregar un archivo
en **.CPP**



O entregar un
archivo en **.PY**

2.2 [Ejercicio Opcional]: Resuelvan mínimo 5 ejercicios del nivel *Recursion 2* de la página *CodingBat*: <http://codingbat.com/java/Recursion-2>



No está permitido el ejercicio **GroupSum**. Pero a continuación encontrarán algunos errores comunes que pueden ser aplicados con los otros ejercicios



PISTA: El algoritmo *GroupSum* falla porque al llamarse recursivamente con el parámetro `start` se queda en una recursión infinita

```
public boolean groupSum(int start, int[] nums, int target) {  
    if (start >= nums.length) return target == 0;  
    return groupSum(start+1, nums, target - nums[start])  
        || groupSum(start, nums, target );  
}
```



PISTA 2: El algoritmo *GroupSum* falla porque al llamarse recursivamente con el parámetro `start-1` se sale del arreglo cuando `start = 0`.

```
public boolean groupSum(int start, int[] nums, int target) {  
    if (start >= nums.length) return target == 0;  
    return groupSum(start+1, nums, target - nums[start])  
        || groupSum(start-1, nums, target );  
}
```



PISTA 3: El algoritmo *GroupSum* falla porque al llamarse recursivamente con el parámetro `start` se sale del arreglo cuando `start = length-1`

```
public boolean groupSum(int start, int[] nums, int target) {  
    if (start >= nums.length) return target == 0;  
    return groupSum(start+1, nums, target - nums[start])  
        || groupSum(start+1, nums, target - nums[start - 1] );  
}
```

2.3 [Ejercicio Opcional] Resuelvan el siguiente problema <http://bit.ly/2gTLZ53>



Pueden **entregar** un
archivo en **JAVA**



O **entregar** un archivo
en **.CPP**




O **entregar** un
archivo en **.PY**



PISTA 1: Utilicen **Búsqueda en Profundidad** (Siglas en inglés *DFS*)



PISTA 2: Véase Guía en **Sección 4, numeral 4.13** “Cómo usar Scanner o *BufferedReader*”

	UNIVERSIDAD EAFIT ESCUELA DE INGENIERÍA DEPARTAMENTO DE INFORMÁTICA Y SISTEMAS	Cód. ST0247
		Estructuras de Datos 2

2.4 [Ejercicio opcional] Resuelvan el siguiente ejercicio <http://bit.ly/2hGqJPB>



PISTA 1: Usen un algoritmo para corroborar si es un grafo bipartito. Léase qué es bipartito en <http://bit.ly/2hGwAo2>





2.5 [Ejercicio Opcional]: Resuelvan el siguiente ejercicio <http://bit.ly/2hrrCfS>



PISTA 2: Algoritmos para hallar componentes fuertemente conexos. Ordenamiento topológico. DFS. Léase en <http://bit.ly/2gTeJKh>

2.6[Ejercicio Opcional]: Resuelvan el siguiente ejercicio <http://bit.ly/2k8CGSG>

3) Simulacro de preguntas de sustentación de Proyectos

	Véase Guía en Sección 3, Numeral 3.5		Entregar informe de laboratorio en PDF
	Usen la plantilla para responder laboratorios		No apliquen Normas Icontec para esto

3.1 Incluyan una imagen de la respuesta de las pruebas del numeral 1.6

3.2 Escriban una explicación entre 3 y 6 líneas de texto del código del numeral 1.1. Digan cómo funciona, cómo está implementado el grafo con matrices y con listas que hizo, destacando las estructuras de datos y algoritmos usados

DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co

3.3 ¿En qué grafos es más conveniente utilizar la implementación con matrices de adyacencia y en qué casos en más convenientes listas de adyacencia? ¿Por qué?



PISTA: <http://bit.ly/2gzZPLD>



PISTA 2: <http://bit.ly/2gSMq1Z>

3.4 Para representar el mapa de la ciudad de Medellín del ejercicio del numeral 1.4, ¿qué es mejor usar, Matrices de Adyacencia o Listas de Adyacencia? ¿Por qué?



En la vida real para una red social como *Facebook*, donde hay al menos 100 millones de usuarios, pero cada usuario tiene en promedio 200 amigos,

3.5 Teniendo en cuenta lo anterior, respondan: ¿Qué es mejor usar, Matrices de Adyacencia o Listas de Adyacencia? ¿Por qué?



En la vida real, los enrutadores tienen una tabla de enrutamiento. Una tabla de enrutamiento guarda la distancia más corta para ir de un dispositivo a otro en la red. Un ejemplo de un enrutador es el ISR 4000 de Cisco. Otro ejemplo, es el que tiene en su casa para el Wifi

3.6 Teniendo en cuenta lo anterior, para representar la tabla de enrutamiento, respondan: ¿Qué es mejor usar, Matrices de Adyacencia o Listas de Adyacencia?



Versiones mejoradas de algoritmos como BFS y DFS son usando para calcular las rutas óptimas que toman los personajes en videojuegos o vehículos autónomos en las carreteras

3.7 Teniendo en cuenta lo anterior, para recorrer grafos, ¿en qué casos conviene usar *DFS*? ¿En qué casos *BFS*?

Error Común



3.8 ¿Qué otros algoritmos de búsqueda existen para grafos? Basta con explicarlos, no hay que escribir los algoritmos ni programarlos.



PISTA: Véase http://kevanahlquist.com/osm_pathfinding/



NOTA: Recuerden que debe explicar su implementación en el informe PDF

3.9 Expliquen con sus propias palabras la estructura de datos que utilizan para resolver los problemas, y cómo funcionan los algoritmos realizados en el numeral 2.1 y los ejercicios opcionales que hayan hecho del punto 2.



NOTA: Recuerden que debe explicar su implementación en el informe PDF

3.10 Calculen la complejidad del ejercicio 2.1 y, voluntariamente, los Ejercicios Opcionales.



PISTA: Véase *Guía en Sección 4, numeral 4.11 “Cómo escribir la complejidad de un ejercicio en línea”*

3.11 Expliquen con sus palabras las variables (*qué es ‘n’, qué es ‘m’, etc.*) del cálculo de complejidad del numeral 3.10

Errores Comunes



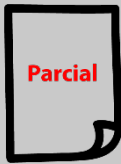
4) Simulacro de Parcial en el informe PDF



PISTA: Véase *Guía en Sección 4, Numeral 4.18* “Respuestas del Quiz”



PISTA 2: Lean las diapositivas tituladas “*Data Structures II: Graph Implementation*”, encontrarán la mayoría de las respuestas

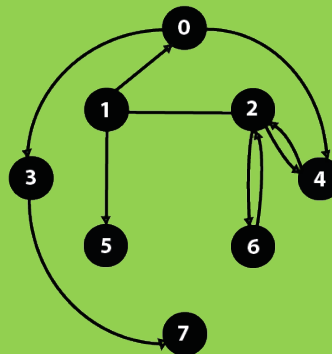


Para este simulacro, agreguen ***sus respuestas*** en el informe PDF.



El día del Parcial no tendrán computador, JAVA o acceso a internet.

1. Considere el siguiente grafo y complete la representación de **matrices de adyacencia**. Si no hay arco, por simplicidad, deje el espacio en blanco.



2. Para el mismo grafo, complete la representación de **listas de adyacencia**. Como el grafo no tiene pesos, sólo se colocan los sucesores en la lista de adyacencia.

	0	1	2	3	4	5	6	7
0				1	1			
1								
2								
3								
4								
5								
6								
7								

3. Para el grafo anterior, complete la salida que darían los siguientes algoritmos:

- a) Complete el orden en que se recorren los nodos usando **búsqueda en profundidad** (en Inglés DFS) a partir de cada nodo. Si hay varias opciones de recorrer el grafo con DFS, elija siempre el vértice más pequeño.
- b) Complete el orden en que se recorren los nodos usando **búsqueda en amplitud** (en Inglés BFS) a partir de cada nodo. Si hay varias opciones de recorrer el grafo con BFS, elija siempre el vértice más pequeño.

4. ¿Cuánta memoria (ojo, no tiempo sino memoria) ocupa una representación usando listas de adyacencia para el peor grafo dirigido con n vértices?

- a) $O(n)$
- b) $O(n^2)$
- c) $O(1)$
- d) $O(\log n)$
- e) $O(n \log n)$

5. La empresa *Gugol Mas* creó un nuevo sistema de mapas georeferenciados. Sus tecnólogos implementaron fácilmente las funcionalidades de GPS, y la interfaz gráfica web y móvil para el sistema.

Desafortunadamente, Gugol Mas no tiene ingenieros en su nómina y nadie ha podido escribir un método que calcule un camino entre 2 vértices en un grafo dirigido.

Su misión es escribir un método que reciba un digrafo y el identificador de dos vértices, y que retorne un camino entre los dos vértices representado como una lista de enteros. Si no hay camino, retorne una lista vacía.

PISTA: En un mapa, las intersecciones se representan como vértices y las vías como arcos.

```
public class EjemplosGrafo {  
    public static LinkedList<Integer>  
        unCamino(Graph g, int p, int q) {  
        ...  
    }  
}
```

Tenga en cuenta que la clase Graph está definida de la siguiente forma:

```
public class Graph { // Todos los mtodos  
    Graph(int vertices); // son públicos  
    ArrayList<Integer> getSuccessors(int vertice);  
    int size(); // Nmero de vrtices  
    int getWeight(int p, int q); // peso del arco  
}
```

5. [Ejercicio Opcional] Lecturas recomendadas



"Quienes se preparan para el ejercicio de una profesión requieren la adquisición de competencias que necesariamente se sustentan en procesos comunicativos. Así cuando se entrevista a un ingeniero recién egresado para un empleo, una buena parte de sus posibilidades radica en su capacidad de comunicación; pero se ha observado que esta es una de sus principales debilidades..."

Tomado de <http://bit.ly/2gJKzJD>



Véase Guía en **Sección 3, numeral 3.6 y 4.20** de la Guía Metodológica, "Lectura recomendada" y "Ejemplo para realización de actividades de las Lecturas Recomendadas", respectivamente

Posterior a la lectura del texto "**Robert Lafore, Data Structures and Algorithms in Java (2nd edition), Chapter 13: Graphs. 2002**" realicen las siguientes actividades que les permitirán sumar puntos adicionales:

- a) Escriban un resumen de la lectura que tenga una longitud de 100 a 150 palabras



PISTA: En el siguiente enlace, unos consejos de cómo hacer un buen resumen <http://bit.ly/2knU3Pv>



PISTA 2: Aquí le explican cómo contar el número de palabras en Microsoft Word

- b) Hagan un mapa conceptual que destaque los principales elementos teóricos.



PISTA: Para que hagan el mapa conceptual se recomiendan herramientas como las que encuentran en <https://cacoo.com/> o <https://www.mindmup.com/#m:new-a-1437527273469>



NOTA 1: Si desean otra lectura, consideren la siguiente: “*John Hopcroft et al., Estructuras de Datos y Algoritmos, Capítulo 6: Grafos dirigidos. Páginas 267 – 276. 1983*” que pueden encontrarla en biblioteca



NOTA 2: Estas respuestas también deben incluirlas en el informe PDF

Otras sugerencias de lectura

Si desean otras lecturas, consideren las siguientes:

- ☒ Thomas Cormen, *Introduction to Algorithms (3th edition), Sections 23.2, 23.3 y 23.5. 2009*, que pueden encontrar en biblioteca
- ☒ “*John Hopcroft et al., Estructuras de Datos y Algoritmos, Capítulo 7: Grafos no dirigidos. 1983*” que pueden encontrar en biblioteca.

6. [Ejercicio Opcional] Trabajo en Equipo y Progreso Gradual



El trabajo en equipo es una exigencia actual del mercado. “Mientras algunos medios retratan la programación como un trabajo solitario, la realidad es que requiere de mucha comunicación y trabajo con otros. Si trabajas para una compañía, serás parte de un equipo de desarrollo y esperarán que te comuniques y trabajes bien con otras personas”

Tomado de <http://bit.ly/1B6hUDp>



Véase Guía en **Sección 3, numeral 3.7** y **Sección 4, numerales 4.21, 4.22 y 4.23** de la Guía Metodológica

- a) Entreguen copia de todas las actas de reunión usando el tablero Kanban, con fecha, hora e integrantes que participaron



PISTA: Véase **Guía en Sección 4, Numeral 4.21** “Ejemplo de cómo hacer actas de trabajo en equipo usando Tablero Kanban”

- b) Entreguen el reporte de *git*, *svn* o *mercurial* con los cambios en el código y quién hizo cada cambio, con fecha, hora e integrantes que participaron



PISTA: Véase Guía en Sección 4, Numeral 4.23 “Cómo generar el historial de cambios en el código de un repositorio que está en svn”

- c) Entreguen el reporte de cambios del informe de laboratorio que se genera *Google docs* o herramientas similares



PISTA: Véase Guía en Sección 4, Numeral 4.22 “Cómo ver el historial de revisión de un archivo en Google Docs”




NOTA: Estas respuestas también deben incluirlas en el informe PDF

Resumen de ejercicios a resolver

1.1.a. Realicen una implementación de la clase abstracta *Digraph*, llámela *DigraphAM* e implementen grafos con la estructura de datos Matrices de Adyacencia Etiquetadas

1.1.b. Posteriormente, creen la clase *DigraphAL* e implementen grafos con la estructura de datos Listas de Adyacencia. Ambas clases heredan de la clase abstracta *Digraph* (digrafo o grafo dirigido)

	UNIVERSIDAD EAFIT ESCUELA DE INGENIERÍA DEPARTAMENTO DE INFORMÁTICA Y SISTEMAS	Cód. ST0247
		Estructuras de Datos 2

1.2 En la clase *GraphAlgorithms*, implementen un método que reciba como parámetro un grafo dirigido y que retorne cuál es el vértice que tiene más sucesores (vecinos). Debe funcionar para ambas implementaciones de grafo.

1.3 Prueben su código con los ejemplos construidos en los numerales 1.1 y 1.2 para el *Algoritmo de Dijkstra*. Deben obtener la misma respuesta con ambas implementaciones.

1.4 Implementen un método que permita crear un grafo a partir de ese archivo del texto *medellin_colombia-grande.txt*

1.5 Implementen en un método el algoritmo de *BFS*, de tal forma de que funcione tanto para la implementación de grafos que utiliza matrices de adyacencia como para la implementación que usa listas de adyacencia, es decir, que reciba un objeto de la clase *Graph*, así como se hizo para *DFS*. El algoritmo debe retornar un *ArrayList* con los vértices en el orden en que los recorrió

1.6 Construyan ejemplos usando *JUnit* para probar su implementación de *BFS*

1.7 Escriban una explicación entre 3 y 6 líneas de texto del código del numeral 1.1., es decir, digan cómo funciona y cómo está implementado

1.8 Implementen un método para un grafo que diga si un grafo tiene ciclos o no

2.1 Resuelvan el siguiente ejercicio: <http://bit.ly/2kxi2LZ>

2.2 [Ejercicio Opcional] Resuelvan mínimo 5 ejercicios del nivel *Recursion 2* de la página *CodingBat*: <http://codingbat.com/java/Recursion-2>


2.3 [Ejercicio Opcional] Resuelvan el siguiente problema <http://bit.ly/2gTLZ53>

2.4 [Ejercicio opcional] Resuelvan el siguiente ejercicio <http://bit.ly/2hGqJPB>

2.5 [Ejercicio Opcional]: Resuelvan el siguiente ejercicio <http://bit.ly/2hrrCfS>

2.7 [Ejercicio Opcional]: Resuelvan el siguiente ejercicio <http://bit.ly/2k8CGSG>

3.1 Incluyan una imagen de la respuesta de las pruebas del numeral 1.6

	UNIVERSIDAD EAFIT ESCUELA DE INGENIERÍA DEPARTAMENTO DE INFORMÁTICA Y SISTEMAS	Cód. ST0247
		Estructuras de Datos 2

3.2 Escriban una explicación entre 3 y 6 líneas de texto del código del numeral 1.4. Digan cómo funciona, cómo está implementado el grafo con matrices y con listas que hizo, destacando las estructuras de datos y algoritmos usados

3.3 ¿En qué grafos es más conveniente utilizar la implementación con matrices de adyacencia y en qué casos en más convenientes listas de adyacencia? ¿Por qué?

3.4 Para representar el mapa de la ciudad de Medellín del ejercicio del numeral 1.4, ¿qué es mejor usar, Matrices de Adyacencia o Listas de Adyacencia? ¿Por qué?

3.5 Respondan: ¿Qué es mejor usar, Matrices de Adyacencia o Listas de Adyacencia? ¿Por qué?

3.6 Para representar la tabla de enrutamiento, respondan: ¿Qué es mejor usar, Matrices de Adyacencia o Listas de Adyacencia?

3.7 Para recorrer grafos, ¿en qué casos conviene usar *DFS*? ¿En qué casos *BFS*?

3.8 ¿Qué otros algoritmos de búsqueda existen para grafos? Basta con explicarlos, no hay que escribir los algoritmos ni programarlos.

3.9 Expliquen con sus propias palabras la estructura de datos que utilizan para resolver los problemas y cómo funcionan los algoritmos realizados en el numeral 2.1 y, voluntariamente, todos los ejercicios opcionales del punto 2.

3.10 Calculen la complejidad del ejercicio 2.1 y, voluntariamente, todos los Ejercicios Opcionales del punto 2.

3.11 Expliquen con sus palabras las variables (*qué es 'n', qué es 'm', etc.*) del cálculo de complejidad del numeral 3.10

4. Simulacro de Parcial en el informe PDF

5. [Ejercicio Opcional] Lecturas recomendadas

6. [Ejercicio Opcional] Trabajo en Equipo y Progreso Gradual