Laboratorio Nro. 3
Vuelta atrás (*Backtracking*)

Objetivos

1. Diseñar algoritmos usando la técnica de diseño de vuelta atrás
2. Resolver problemas fundamentales de grafos, incluyendo búsqueda DFS y BFS
3. Resolver problemas usando algoritmos de grafos, incluido el problema del camino más corto, y al menos, un algoritmo del árbol de expansión con costo mínimo

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/2g4TTKf, el cual será atendido en la menor brevedad posible.

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

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 3.7.***

GitHub

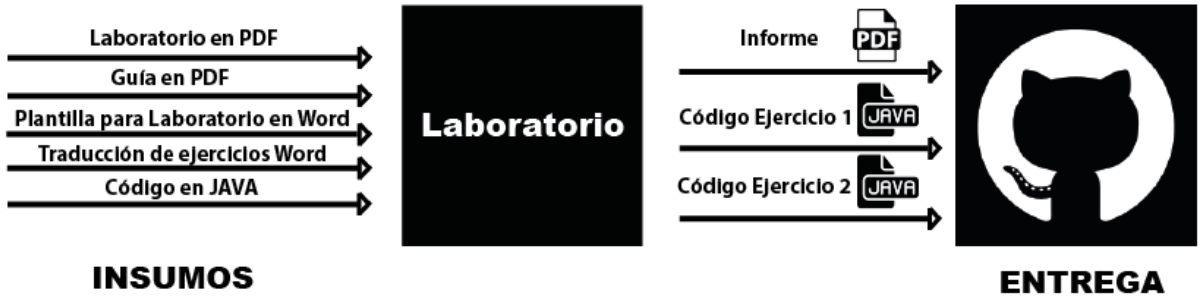
1. Crear un repositorio en su cuenta de GitHub con el nombre `st0247`. **2.** Crear una carpeta dentro de ese repositorio con el nombre `laboratorios`. **3.** Dentro de la carpeta laboratorio, crear una carpeta con nombre `lab03`. **4.** Dentro de la carpeta `lab03`, crear tres carpetas: `informe`, `codigo` 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
         ...
```

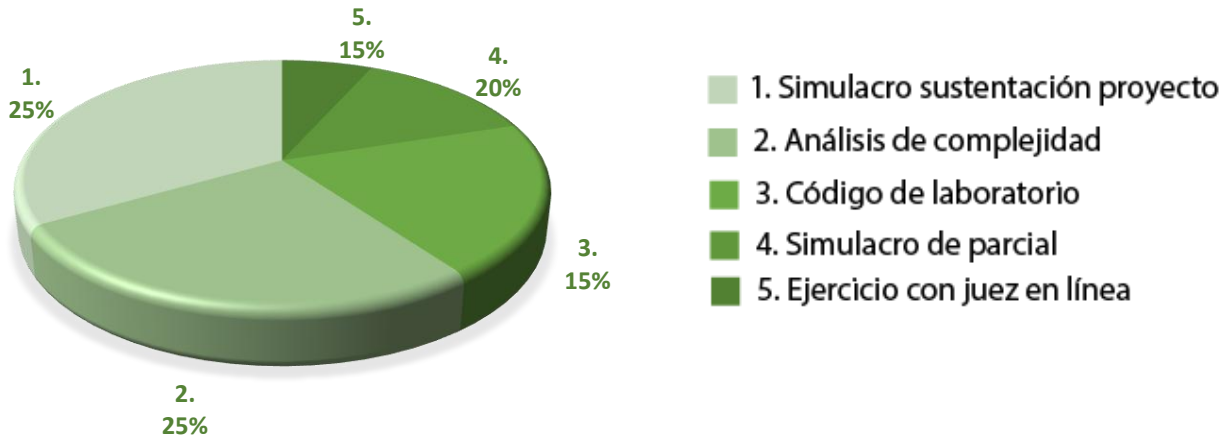
DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co

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



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

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**

Es opcional entregar documentación. Si lo hace, utilice **Javadoc** o equivalente. No suba el HTML a GitHub.

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

1.1 Implementen el algoritmo de *backtracking* para encontrar UNA solución de las N Reinas.

NOTA: Si el algoritmo entrega TODAS las soluciones, quedó malo

1.2 [Ejercicio opcional] Construyan ejemplos usando JUnit para probar su implementación de las N Reinas usando *backtracking*. Como muestra, usen los ejemplos que ya conocen para el problema de las 4 reinas

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

NOTA 1: Si utilizan Python o C++, utilicen una librería equivalente para pruebas unitarias en dichos lenguajes.

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.

1.3 [Ejercicio opcional] 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

1.4 [Ejercicio opcional] Teniendo en cuenta lo anterior, construyan ejemplos usando *JUnit* para probar su implementación de *BFS*

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

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

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

2) Ejercicios en línea sin documentación HTML en GitHub:

En la vida real, el camino más corto entre dos puntos en un grafo se aplica en sistemas de información de geográfica como *Google Maps* y en enrutadores de red como el *Cisco ISR* 4000

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

No entregar documentación **HTML**

*Entregar un archivo en **.JAVA***

No se reciben** archivos en **.PDF

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

En la vida real, los algoritmos para encontrar caminos se utilizan en los videojuegos como parte de la inteligencia artificial de la máquina en juegos como League of Legends. Más información en http://bit.ly/2D5Yo3z

2.1 **Resuelvan el siguiente problema usando *backtracking y SIN usar el algoritmo de Dijkstra ni otros algoritmos voraces***

NOTA: Esta técnica no es la más eficiente para resolver este problema, pero es la que usaremos en este ejercicio

A ustedes le entregan un grafo no dirigido con pesos. Los vértices están enumerados del 1 al *n*. Su tarea es encontrar la ruta más corta entre el vértice 1 y el vértice *n*.

Entrada

La primera línea contiene 2 enteros n y m ($2 \le n \le 105$, $0 \le m \le 105$), donde *n* es el número de vértices y m es el número de arcos. Después hay m líneas, donde cada uno contiene un arco de la forma *ai*, *bi* and *wi* ($1 \le ai, bi \le n$, $1 \le wi \le 106$), donde *ai, bi* son los vértices del arco y *wi* es el peso del arco.

Es posible que en el grafo haya ciclos y que haya varios vértices entre el mismo par de vértices.

Salida

Escriban -1 en caso de que no haya camino. Escriban el camino más corto de lo contrario. Si hay varias soluciones, impriman cualquiera de ellas.

Ejemplos

Entrada

```
5 6
1 2 2
2 5 5
2 3 4
1 4 1
4 3 3
```

3 5 1

Salida

1 4 3 5

Entrada

5 6
1 2 2
2 5 5
2 3 4
1 4 1
4 3 3
3 5 1

Salida

1 4 3 5

2.2 [Ejercicio Opcional]: Resuelvan el siguiente problema http://bit.ly/2k8CGSG

2.3[Ejercicio Opcional] Resuelvan el siguiente problema http://bit.ly/2qTLZ53

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

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

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

3. Simulacro de preguntas de sustentación de Proyectos

Véase Guía en ***Sección 3, Numeral 3.4***

Entregar informe de laboratorio en **PDF**

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

*Usen la **plantilla** para responder laboratorios*

***No** apliquen **Normas Icontec** para esto*

3.1 Para resolver el problema del camino más corto en un grafo, fuera de fuerza bruta y backtracking, ¿qué otras técnicas computacionales existen?

En la vida real, grandes compañías como Google, valoran más los conocimientos en complejidad computacional que un título de X o Y universidad

Tomado de **http://bit.ly/2hQAZHP**

3.2 [Ejercicio opcional] Teniendo en cuenta lo anterior, tomen los tiempos de ejecución del programa realizado en el numeral 1.1 y en el laboratorio anterior con la solución de fuerza bruta de las n reinas. Completen la siguiente tabla.

Si se demora más de 50 minutos, coloque *"se demora más de 50 minutos",* no sigan esperando, podría tomar siglos en dar la respuesta, literalmente.

Valor de N	Tiempo de ejecución
4	
5	
6	
7	
8	
9	
10	
11	
12	

13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	
28	
29	
30	
31	
32	
N	**O ( )**

3.3 Para recorrer grafos, ¿en qué problemas conviene usar *DFS*? ¿En qué problemas *BFS*?

3.4 ¿Qué otros algoritmos de búsqueda, fuera de DFS y BFS, existen para grafos? Basta con explicarlos, no hay que escribir los algoritmos ni programarlos.

3.5 Expliquen con sus propias palabras la estructura de datos que utiliza para resolver el problema del numeral 2.1 y 2.2 [Ejercicio Opcional] y digan cómo funciona el programa.

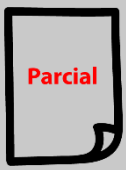
NOTA: Recuerden que deben explicar su implementación en el informe PDF

3.6 Calculen la complejidad de los ejercicios en línea del numeral 2.1 y 2.2 [Ejercicio Opcional] y agréguenla al informe PDF

3.7 Expliquen con sus palabras las variables (*qué es 'n', qué es 'm', etc.*) del cálculo de complejidad del numeral 3.6

3.8 Escriban una explicación entre 3 y 6 líneas de texto del código del ejercicio en línea del numeral 1.1. Digan cómo funciona, cómo está implementado y destaquen las estructuras de datos y algoritmos usados

4) Simulacro de Parcial en el informe PDF



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

Resuelva, como mínimo, los ejercicios marcados con color rojo.

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

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

1. Wilkenson y Sofronio están aquí de nuevo. En esta vez han traído un juego muy interesante, en el cual Sofronio, en primer lugar, escoge un numero n ($1 \leq n \leq 20$) y, en segundo lugar, escoge tres números a,b y c ($1 \leq a \leq 9, 1 \leq b \leq 9, 1 \leq c \leq 9$).

Después, Sofronio le entrega estos números a Wilkenson y Wilkenson le tiene que decir a Sofronio **la cantidad máxima de números, usando a, b y c (se puede tomar un número más de una vez), que al sumarlos den el valor n**.

Como un ejemplo, si Sofronio escoge n=14 y a=3, b=2, c=7. ¿Qué posibilidades hay de sumar 14 con a,b y c?

7+7=14	cantidad es 2
7+3+2+2=14	cantidad es 4
3+3+3+3+2=14	cantidad es 5
…	
2+2+2+2+2+2+2=14	cantidad es 7

La cantidad máxima de números es 7. Esta sería la respuesta que da Wilkenson a Sofronio.

Como Wilkenson es muy astuto, ha diseñado un algoritmo para determinar la cantidad máxima de números y quiere que le ayudes a terminar su código. Asuma que hay al menos una forma de sumar n usando los números a, b y c en diferentes cantidades, incluso si algunos de los números se suman 0 veces como sucede en el ejemplo anterior.

```
1 int solucionar (int n, int a, int b, int c)
2   if (n == 0 )
3      return 0;
4   int res = solucionar(_____________) + 1;
5   res = Math.max(_______,____________);
6   res = Math.max(_______,____________);
7   return res;
```

a) Complete el espacio de la línea 04 (10 %)
_____

b) Complete los espacios de la línea 05 (10 %)
_____, _____

c) Complete los espacios de la línea 06 (10 %)

_____, _____

2. ***Un camino hamiltoniano*** en un grafo dirigido es un camino que visita cada vértice exactamente una vez. Un ***ciclo hamiltoniano*** es un camino hamiltoniano para el cual existe un arco (en el grafo) que conecta el último vértice del camino hamiltoniano con el primer vértice del camino hamiltoniano.

Su tarea ***es determinar si dado un grafo, este grafo contiene un ciclo hamiltoniano o no. Si lo contiene, retorne verdadero; de lo contrario, retorne falso.***

Parte de su tarea ya está hecha. La función `sePuede` verifica si un vértice `v` se puede agregar al ciclo hamiltoniano que está almacenado en el arreglo `path` en la posición `pos`, dado un grafo representado con matrices de adyancencia `graph`.

Por simplicidad, sólo se busca si existe un camino que empieza y termina en el primer vértice (es decir, en el vértice 0).

Por esta razón, en el arreglo `path` se entrega con todas sus posiciones en −1, excepto la posición 0, como se muestra en la función `cicloHamil`. También, por esta razón, en el ciclo de la línea 08, `v` inicia se con 1.

```
boolean cicloHamil(int graph[][]) {
  path = new int[g.length];
  for (int i = 0; i < g.length; i++)
      path[i] = -1;
  path[0] = 0;
  return cicloHamilAux(graph, path, 1);
}


boolean sePuede(int v, int graph[][],
                int path[], int pos) {
 if (graph[path[pos - 1]][v] == 0)
    return false;
 for (int i = 0; i < pos; i++)
    if (path[i] == v)
      return false;
 return true;
}
```

```
01 boolean cicloHamilAux(int graph[][],
                  int path[], int pos) {
02   if (pos == ____________){
03    if (graph[path[pos-1]][path[0]] == 1)
04           return true;
05    else
06           return false;
07   }
08   for (int v = 1; v < graph.length; v++) {
09       if (sePuede(__,__,__,__)) {
10           path[pos] = v;
11           if (cicloHamilAux(__,__,__))
12               return true;
13           path[pos] = -1;
14       }
15   }
16   return false;
17 }
```

a) Completen el espacio en línea 02 que corresponde a la condición de parada (10%)

_____

b) Completen los espacios en línea 09 que corresponden al llamado de la función *sePuede* (10%)

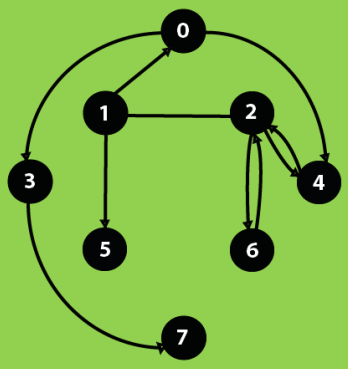
_____, _____, _____, _____

c) Completen los espacios en la línea 11 que corresponden al llamado recursivo de la función *cicloHamil* (10%)

_____, _____, _____

3. Para el grafo siguiente, completen la salida que darían los siguientes algoritmos:

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



a) Completen 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, elijan siempre el vértice más pequeño.

b) Completen 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, elijan siempre el vértice más pequeño.

4. 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) {
       ...
    }
```

```
}
```

Tengan 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. El problema de la **subsecuencia común más larga** es el siguiente. Dadas dos secuencias, encontrar la longitud de la secuencia más larga presente en ambas. Una subsecuencia es una secuencia que aparece en el mismo orden relativo, pero no necesariamente de forma contigua.

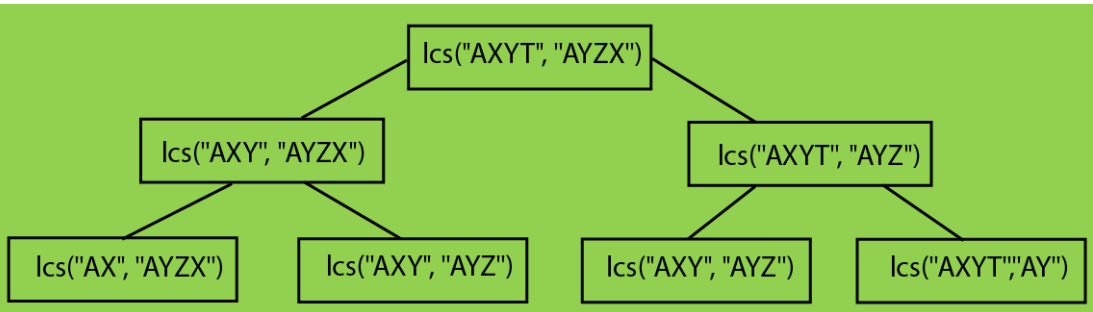
Como un ejemplo, “abc”, “abg”, “bdf”, “aeg” y “acefg” son subsecuencias de “abcdefg”. Entonces, para una cadena de longitud *n* existen posibles subsecuencias.

Este problema es utilizado en la implementación del comando `diff`, para comparación de archivos, disponible en sistemas Unix. También tiene muchas aplicaciones en bioinformática.

Consideren los siguientes ejemplos para el problema:

- Para “ABCDGH” y “AEDFHR” es “ADH” y su longitud es 3.
- Para “AGGTAB” y “GXTXAYB” es “GTAB” y su longitud es 4.

Una forma de resolver este problema es usando *backtracking*, como un ejemplo, para las cadenas “AXYT” y “AYZX”, dada una función recursiva `lcs` que resuelve el problema, se obtendría el siguiente árbol (parcial) de recursión:



Al siguiente código le faltan algunas líneas, complétenlas por favor:

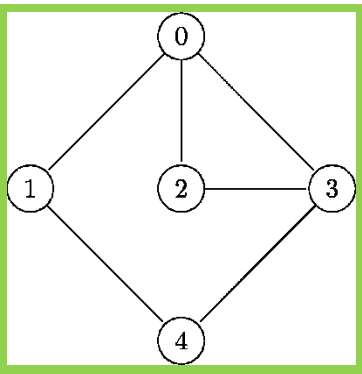
```
01 private int lcs(int i, int j, String s1, String s2){
02     if(i == 0 || j == 0){
03         return 0;
04     }
05     boolean prev = i < s1.length() && j < s2.length();
06     if(prev && s1.charAt(i) == s2.charAt(j)){
07         return ______+ lcs(i - 1, j - 1, s1, s2);
08     }
09     int ni = lcs(i - 1, j, s1, s2);
10     int nj = lcs(i, j - 1, s1, s2);
11     return Math.max(_____, _____);
12 }
13 public int lcs(String s1, String s2){
14     return lcs(s1.length(), s2.length(), s1, s2);
15 }
```

5.1 Línea 7 _____

5.2 Línea 11 _____, _____
Y completen la complejidad por favor

5.3 Supongan que *n* es la suma de la longitud de las dos cadenas. El algoritmo `lcs` ejecuta, en el peor de los casos, $T(n)$ = _____ instrucciones.

6. Los ejercicios de esta sección se deberán resolver de acuerdo al siguiente grafo.



6.1 DFS. Un posible recorrido ***DFS*** del grafo anterior, al ejecutarlo desde el vértice 0, es:

(a)0,4,1,2,3
(b)0,2,4,3,1
(c)0,1,4,3,2
(d)0,4,2,3,1

6.2 BFS. Un posible recorrido ***BFS*** del grafo anterior, al ejecutarlo desde el vértice 0, es:

(a)0,1,2,3,4
(b)0,1,4,2,3
(c)0,4,3,2,1
(d)0,4,2,1,3

5. [Ejercicio Opcional] Lectura recomendada

"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 3.5 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 ***"R.C.T Lee et al., Introducción al análisis y diseño de Algoritmos. Capítulo 5. Páginas 157 – 181.",*** 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

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

NOTA 1: Si desean una lectura adicional en español, consideren la siguiente ***"John Hopcroft et al., Estructuras de Datos y Algoritmos, Sección 10.4. 1983",*** que encuentran en biblioteca

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

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 3.6** 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

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

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

NOTA 1: Estas respuestas también deben incluirlas en el informe PDF

7. [Ejercicio Opcional] Laboratorio en inglés

El inglés es un idioma muy importante en la Ingeniería de Sistemas porque la mayoría de los avances en tecnología se publican en este idioma y la traducción, usualmente se demora un tiempo y es sólo un resumen de la información original.

Adicionalmente, dominar el inglés permite conseguir trabajos en el exterior que son muy bien remunerados

Tomado de goo.gl/4s3LmZ

Entreguen el código y el informe traducido al inglés. Utilicen la plantilla dispuesta en este idioma para el laboratorio

Resumen de ejercicios a resolver

1.1 Implementen el algoritmo de *backtracking* para encontrar UNA solución de las N Reinas.

1.2 [Ejercicio opcional] Construyan ejemplos usando JUnit para probar su implementación de las N Reinas usando *backtracking*. Como muestra, usen los ejemplos que ya conoce para el problema de las 4 reinas

1.3 [Ejercicio opcional] 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

1.4 [Ejercicio opcional] Construyan ejemplos usando *JUnit* para probar su implementación de *BFS*

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

2.1 **Resuelvan el siguiente problema usando *backtracking y SIN usar el algoritmo de Dijkstra ni otros algoritmos voraces***

2.2 [Ejercicio Opcional] Resuelvan el siguiente problema http://bit.ly/2k8CGSG

2.5 [Ejercicio Opcional] Resuelvan el siguiente problema http://bit.ly/2qTLZ53

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

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

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

3.1 Para **resolver el problema del camino más corto en un grafo**, fuera de fuerza bruta y backtracking, **¿qué otras técnicas computacionales existen?**

3.2 [Ejercicio opcional] Tomen los tiempos de ejecución del programa realizado en el numeral 1.1 y en el laboratorio anterior con la solución de fuerza bruta de las n reinas.

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

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

3.5 Expliquen con sus propias palabras la estructura de datos que utiliza para resolver el problema del numeral 2.1 y 2.2 [Ejercicio Opcional] y digan cómo funciona el programa.

3.6 Calculen la complejidad de los ejercicios en línea del numeral 2.1 y 2.2 [Ejercicio Opcional] y agréguenla al informe PDF

3.7 Expliquen con sus palabras las variables (*qué es 'n', qué es 'm', etc.*) del cálculo de complejidad del numeral 3.6

3.8 Escriban una explicación entre 3 y 6 líneas de texto del código del ejercicio en línea del numeral 1.1

4. **Simulacro de Parcial**

5. **Lectura recomendada [Ejercicio Opcional]**

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

7. **Entreguen el código y el informe traducido al inglés. [Ejercicio Opcional]**

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

Ayudas para resolver los ejercicios

Ayudas para el Ejercicio 1.2……. Pág. 25

Ayudas para el Ejercicio 1.4……. Pág. 25

Ayudas para el Ejercicio 1.5……. Pág. 26

Ayudas para el Ejercicio 1.6……. Pág. 26

Ayudas para el Ejercicio 1.8……. Pág. 27

Ayudas para el Ejercicio 2.1……. Pág. 27

Ayudas para el Ejercicio 2.4……. Pág. 27

Ayudas para el Ejercicio 2.5……. Pág. 27

Ayudas para el Ejercicio 3..1……. Pág. 28

Ayudas para el Ejercicio 3.2…….. Pág. 28

Ayudas para el Ejercicio 3.3…….. Pág. 28

Ayudas para el Ejercicio 3.4…….. Pág. 28

Ayudas para el Ejercicio 3.6…….. Pág. 29

Ayudas para el Ejercicio 4.0…….. Pág. 29

Ayudas para el Ejercicio 5A……. Pág. 29

Ayudas para el Ejercicio 5B……. Pág. 29

Ayudas para el Ejercicio 6A……. Pág. 30

Ayudas para el Ejercicio 6B……. Pág. 30

Ayudas para el Ejercicio 6C……. Pág. 30

Ayudas para el Ejercicio 1.2

PISTA 1: *Véase Guía,* ***Sección 4, numeral 4.14*** *"Cómo hacer pruebas unitarias en BlueJ usando JUnit"*

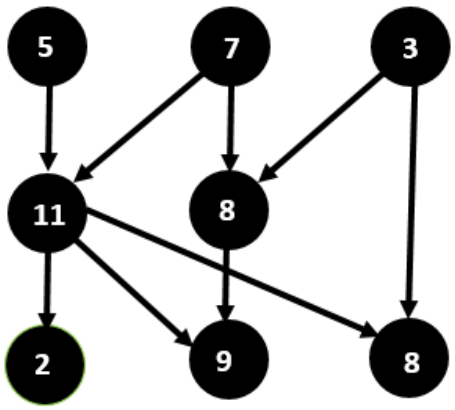
PISTA 2: Usen el método *AssertArrayEquals* de *JUnit* que encuentra en **http://junit.sourceforge.net/javadoc/org/junit/Assert.html**

Ayudas para el Ejercicio 1.4

PISTA 1: 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**

Ayudas para el Ejercicio 1.5

PISTA 1: Si deciden hacer la documentación, consulten la *Guía en* ***Sección 4, numeral 4.1*** *"Cómo escribir la documentación HTML de un código usando JavaDoc"*

Ayudas para el Ejercicio 1.6

PISTA 1: Solución en pseudocódigo

```
llenar el arreglo de distancias con infinito
marcar la distancia al nodo inicial como 0
llamar dfs con el nodo raiz
dfs {
  Para (cada hijo) {
     si (puedo mejorar distancias hasta este) {
 marque nueva distancia (mejorada)
 llamese recursivamente para el hijo
    }
  }
}
```

PISTA 1: Vean **Problema** y **Solución**

Ayudas para el Ejercicio 1.8

PISTA 1: Si deciden hacer la documentación, consulten la *Guía en* ***Sección 4, numeral 4.1*** *"Cómo escribir la documentación HTML de un código usando JavaDoc"*

Ayudas para el Ejercicio 2.1

PISTA 1: Construyan un grafo. Utilicen el recorrido DFS.

PISTA 2: Retornen una pareja que contiene el camino y el peso total

Ayudas para el Ejercicio 2.4

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

Ayudas para el Ejercicio 2.5

PISTA 1: Algoritmos para hallar componentes fuertemente conexos. Ordenamiento topológico. DFS. Léase en **http://bit.ly/2gTeJKh**

Ayudas para el Ejercicio 3.1

PISTA 1: *Lean* **http://bit.ly/2hPomyn**

Ayudas para el Ejercicio 3.2

PISTA 1: *Véase* ***Guía en Sección 4, numeral 4.6*** *"Cómo usar la escala logarítmica en Microsoft Excel 2013"*

Ayudas para el Ejercicio 3.3

Error Común

Ayudas para el Ejercicio 3.4

PISTA 1: Véase **http://kevanahlquist.com/osm_pathfinding/**

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

Ayudas para el Ejercicio 3.6

PISTA 1: Véase ***Guía en Sección 4, numeral 4.11*** *"Cómo escribir la complejidad de un ejercicio en línea"*

Ayudas para el Ejercicio 4.0

PISTA 1: *Véase* ***Guía en Sección 4, Numeral 4.18*** "Respuestas del Quiz"

PISTA 2: Lean las diapositivas tituladas "***Data Structures II: Backtracking"***, encontrarán la mayoría de las respuestas

Ayudas para el Ejercicio 5a

PISTA 1: En el siguiente enlace, unos consejos de cómo hacer un buen resumen **http://bit.ly/2knU3Pv**

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

Ayudas para el Ejercicio 5b

PISTA 1: 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**

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

Ayudas para el Ejercicio 6a

PISTA 1: *Véase* ***Guía en Sección 4, Numeral 4.21*** "*Ejemplo de cómo hacer actas de trabajo en equipo usando Tablero Kanban"*

Ayudas para el Ejercicio 6b

PISTA 1: ***Véase*** *Guía en Sección 4, Numeral 4.23* "**Cómo generar el historial de cambios en el código de un repositorio que está en svn"**

Ayudas para el Ejercicio 6c

PISTA 1: ***Véase*** *Guía en Sección 4, Numeral 4.22* "***Cómo ver el historial de revisión de un archivo en Google Docs*"**

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