

## Taller en Sala 4

### Backtracking para Grafos (Básico)



**Objetivos:** 1. Resolver problemas fundamentales de grafos, incluyendo búsqueda DFS y BFS. 2. Identificar un problema práctico para cada una de las estrategias de diseño de algoritmos. 3. Traducir problemas de la vida real a soluciones algorítmicas.



**Consideraciones:** Lean y verifiquen las consideraciones de entrega,



Trabajo en Parejas



Mañana, plazo de entrega



Docente entrega plantilla de código en GitHub



Sí .cpp, .py o .java



No .zip, .txt, html o .doc



Alumnos entregan código sin comprimir GitHub



En la carpeta Github del curso, hay **un código iniciado y un código de pruebas (tests)** que pueden explorar para solucionar los ejercicios



**Estructura del documento:** a) Datos de *vida real*, b) *Introducción* a un problema, c) Problema a resolver, d) Ayudas. Identifiquen esos elementos así:

a)



b)



c)



d)



**PhD. Mauricio Toro Bermúdez**

Docente | Escuela de Ingeniería | Informática y Sistemas  
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627  
Tel: (+57) (4) 261 95 00 Ext. 9473



**Teoría de grafos:** Los algoritmos que desarrollarán en este taller deben funcionar al menos para los siguientes tipos de grafo:

- Grafos con ciclos, también llamados grafos cíclicos
- Grafos con islas (donde dados dos vértices de este estos pueden no estar conectados de ninguna manera) también llamado grafos no conexos
- Grafos donde hay varios caminos entre un par de vértices



Los algoritmos para encontrar caminos en grafos se utilizan en aplicaciones de mapas. Vean <https://bit.ly/2wxPa9z>

## Ejercicios a resolver

- 1** En el videojuego *Age of Empires 1: Rise of Rome*, el objetivo es conquistar otras civilizaciones. Para lograrlo, es necesario enviar ejército entre diferentes puntos del mapa. El mapa se representa internamente como un grafo.

Un problema que existe es la existencia de islas porque esto significa que el grafo no es conexo y, por consiguiente, no se puede llegar desde cualquier punto del grafo a cualquier otro punto



- ▶ Usando búsqueda primero en profundidad (en Inglés, DFS), implementen un algoritmo que retorne si hay camino o no entre un nodo  $i$  y un nodo  $j$  en un grafo dirigido  $g$ .

**PhD. Mauricio Toro Bermúdez**

Docente | Escuela de Ingeniería | Informática y Sistemas  
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627  
Tel: (+57) (4) 261 95 00 Ext. 9473

## ESTRUCTURA DE DATOS 2

### Código ST0247

- 2** En el videojuego *Grand Theft Auto V*, cuando ordenamos a un personaje ir de un lugar a otro en el mapa, estamos interesados en que el personaje tome la ruta más corta.

Para calcular la ruta más corta, lo primero es modelar el mapa como un grafo dirigido debe ser dirigido porque no todas las calles son doble vía y no queremos una infracción de tránsito. El siguiente paso es diseñar el algoritmo



- ▶ Dado un grafo dirigido y el índice de dos de sus vértices *inicio* y *fin*, hallen el costo del camino de menor costo (valga la redundancia) que inicia en *inicio* y llega a *fin* utilizando *backtracking*.

- 3** [Ejercicio opcional] Una de las funciones sustanciales del grupo EMI es brindar el servicio médico en casa. Un problema que enfrenta actualmente EMI es encontrar el orden óptimo para visitar a sus clientes de tal forma que se minimice la distancia total del recorrido que tiene que hacer cada uno de sus médicos. Este problema se conoce como el problema del agente viajero.



- ▶ Dado un grafo dirigido, hallen el costo mínimo de un recorrido que pasa por todos los vértices --exactamente una vez-- y vuelve al nodo inicial, utilizando *backtracking*.

- 4** ▶ [Ejercicio opcional] Implementen un método que dado un grafo (*Digraph g*) y el identificador de un nodo en el grafo (*int start*), realice un recorrido en profundidad desde *start*, y retorne una lista en la que almacena los indicadores de los vértices a medida que los visita.

PhD. Mauricio Toro Bermúdez

Docente | Escuela de Ingeniería | Informática y Sistemas  
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627  
Tel: (+57) (4) 261 95 00 Ext. 9473

ESTRUCTURA DE DATOS 2  
Código ST0247

- 5** ▶ **[Ejercicio opcional]** Implementen un método que dado un grafo (*Digraph g*) y el identificador de un nodo en el grafo (*int start*), realice un recorrido en anchura desde *start*, y retorne una lista en la que almacena los indicadores de los vértices a medida que los visita.
- 6** ▶ **[Ejercicio opcional]** Usando BFS, escriban una implementación que retorne si hay camino o no entre un vértice *i* y un vértice *j* en un digrafo *g*.

# Ayudas para resolver los Ejercicios

Ejercicio 1 .....	<u>Pág. 6</u>
Ejercicio 2 .....	<u>Pág. 6</u>
Ejercicio 3 .....	<u>Pág. 8</u>
Ejercicio 4 .....	<u>Pág. 9</u>
Ejercicio 5 .....	<u>Pág. 10</u>
Ejercicio 6 .....	<u>Pág. 11</u>



## Ejercicio 1



**Pista 1:** Consideren el siguiente código:

```
public static boolean hayCaminoDFS(Digraph g, int i, int j)
{
}
```



**Error Común 1:** No hacer uso de lo que retorna el llamado recursivo

```
public static boolean hayCamino1(Graph g, int a, int b) {
    boolean visitados[] = new boolean[g.size()];
    return hayCamino1Aux(g,a,b,visitados);
}

public boolean hayCamino1Aux(Graph g, int v, int w, boolean[] visitados) {
    visitados[v] = true;
    ArrayList<Integer> sucesores = g.getSuccessors(v);
    for (int i = 0; i < sucesores.size(); i++) {
        if (!visitados[sucesores.get(i)] )
            boolean b =
                hayCamino1Aux(g, sucesores.get(i), w, visitados) );
    }
}
```

El problema es que se están haciendo  $n$  llamados recursivos, uno por cada sucesor del vértice, pero no se hace nada con el retorno del llamado recursivo. Como un ejemplo, el primer llamado recursivo retorna verdadero y el segundo falso. ¿Con cuál respuesta se quedan? ¿Con el primero? ¿Con el último? ¿Con una operación lógica entre todos?



## Ejercicio 2



**Pista 1:** Asuman que los vértices dados son válidos y que para ir del vértice  $k$  a sí mismo el costo es 0. Si no hay camino retorne -1.

**PhD. Mauricio Toro Bermúdez**

Docente | Escuela de Ingeniería | Informática y Sistemas  
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627  
Tel: (+57) (4) 261 95 00 Ext. 9473

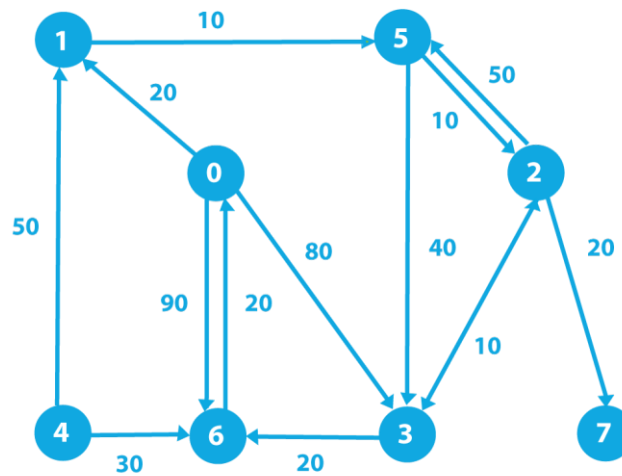
## ESTRUCTURA DE DATOS 2

### Código ST0247

```
public static int costoMinimo(Digraph g, int inicio, int
fin) {
// complete...
}
```



**Ejemplo 1:** En el siguiente grafo el camino más corto para ir de 0 a 6 tiene costo total 70: 0, 1, 5, 2, 3, 6.



Nótese que hay otros caminos como 0, 6 y 0, 3, 6; pero tienen un costo mayor (90 y 100 respectivamente).



**Pista 2:** Utilicen DFS y en vez de un arreglo de visitados usen uno de distancias, y vayan mejorando las distancias paso a paso.

```
private static void dfs(Digraph g, int v, int[] costo) {
// complete...
}
```



**Recomendación:** Una forma menos eficiente de encontrar el camino más corto entre dos vértices en un grafo, lo pueden ver en el siguiente problema: <https://bit.ly/2tEEeYa>

**PhD. Mauricio Toro Bermúdez**

Docente | Escuela de Ingeniería | Informática y Sistemas  
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627  
Tel: (+57) (4) 261 95 00 Ext. 9473

Se recomienda que solucionen este problema ya que les dará una mejor comprensión del algoritmo y probará con mayor rigurosidad la exactitud de su solución.



### Ejercicio 3

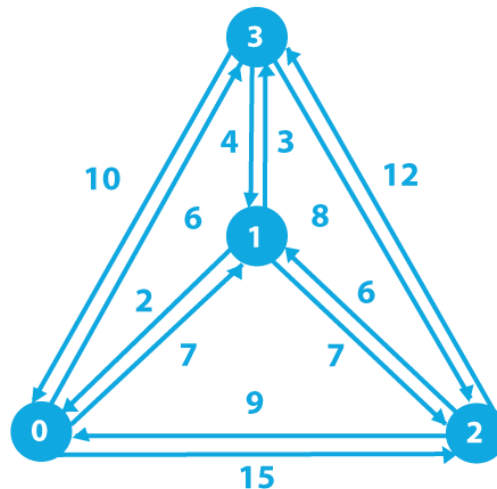


**Pista 1:** Consideren el siguiente código:

```
public static int recorrido(Digraph g) {
    // complete...
}
```



**Ejemplo 1:** En el siguiente grafo el costo total del recorrido de costo mínimo que pasa por los cuatro vértices (iniciando y volviendo a 0) de 22:



**PhD. Mauricio Toro Bermúdez**

Docente | Escuela de Ingeniería | Informática y Sistemas  
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627  
Tel: (+57) (4) 261 95 00 Ext. 9473



## ESTRUCTURA DE DATOS 2

### Código ST0247



**Pista 2:** Nótese que puede asumir que inicia en cualquier vértice, como por ejemplo 0.



**Pista 3:** Calculen todas las posibles permutaciones (*sin repetición*) y retornen la que tenga menor costo total. Para esto es posible que quieran crear un método que dado un arreglo de enteros *a* elimine el elemento en la posición *k*, al igual que una función auxiliar que solo tenga en cuenta los vértices sin visitar.

```
private static int[] removeAt(int k, int a[]) {
    // complete...
}

private static int recorrido(Digraph g, int v, int[]
unvisited) {
    // complete...
}
```



**Pista 4:** Recuerden tener en cuenta el costo para volver al vértice inicial



## Ejercicio 4



**Pista 1:** En caso de poder visitar más de un hijo en el recorrido, visítelos en base a su identificador de menor a mayor.



**Ejemplo 1:** Así es un recorrido en profundidad desde 7 en el grafo de la imagen:

$7 \rightarrow 7, 8, 9, 11, 2, 10$

**PhD. Mauricio Toro Bermúdez**

Docente | Escuela de Ingeniería | Informática y Sistemas  
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627  
Tel: (+57) (4) 261 95 00 Ext. 9473

**ESTRUCTURA DE DATOS 2**  
**Código ST0247**

```
public static ArrayList<Integer> dfs(Digraph g, int start) {
}
```



**Pista 2:** Implementen el método de forma recursiva y creen una función auxiliar similar a esta:

```
private static void dfs(Digraph g, int nodo, boolean[]
visitados, ArrayList<Integer> list) {
}
```



**Error Común 1:** Olvidar hacer el arreglo de visitados

```
public void dfsMalo(Graph g, int v) {
dfsAux(Graph g, int v, visitados);
}
```



**Error Común 2:** No utilizar el arreglo de visitados:

```
public void dfsAuxMalo(Graph g, int v, boolean[] vi){
    vi[v] = true;
    System.out.println(v);
    ArrayList<Integer> vecinos = g.successors(v);
    for (Integer vecino: vecinos) {
        dfsAux(g, vecino, vi);
    }
}
```

**PhD. Mauricio Toro Bermúdez**

Docente | Escuela de Ingeniería | Informática y Sistemas  
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627  
Tel: (+57) (4) 261 95 00 Ext. 9473



## Ejercicio 5



**Pista 1:** En caso de poder visitar más de un hijo en el recorrido, visítenlos en base a su identificador de menor a mayor.

En caso de que el nodo start sea un nodo aislado (es decir, no se conecta a ningún otro nodo) retorne null. Por ejemplo, en el grafo de la imagen los nodos 0, 1, 2, 4, 6, 9 y 10 son nodos aislados.



**Ejemplo 1**, así es un recorrido en anchura desde 7 en el grafo de la imagen:

7 → 7, 8, 11, 9, 2, 10

```
public static ArrayList<Integer> bfs(Digraph g, int start) {  
  
}
```



## Ejercicio 6



**Pista:** Consideren el siguiente código:

```
public static boolean hayCaminoBFS(Digraph g, int i, int j) {  
}
```

**PhD. Mauricio Toro Bermúdez**

Docente | Escuela de Ingeniería | Informática y Sistemas  
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627  
Tel: (+57) (4) 261 95 00 Ext. 9473

# ¿Alguna inquietud?

## CONTACTO

Docente Mauricio Toro Bermúdez

Teléfono: (+57) (4) 261 95 00 Ext. 9473

Correo: mtorobe@eafit.edu.co

Oficina: 19- 627

Agenden una cita dando clic en la pestaña

-*Semana*- de <http://bit.ly/2gzVg10>