	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II</b>	Código:	MADO-20
		Versión:	01
		Página	80/180
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

# Guía Práctica de Estudio 7


## Algoritmos de Grafos parte 2

Elaborado por:

M.I. Elba Karen Sáenz García

Revisión:

Ing. Laura Sandoval Montaña

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II</b>	Código:	MADO-20
		Versión:	01
		Página	81/180
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Guía Práctica 7

### Estructura de datos y Algoritmos II

### Algoritmos de Grafos. Parte 2.

**Objetivo:** El estudiante conocerá e identificará las características necesarias para entender el algoritmo de búsqueda por profundidad en un grafo.

#### Actividades

Implementar la búsqueda por profundidad en un grafo representado por una lista de adyacencia en algún lenguaje de programación.

#### Antecedentes


- Análisis previo del concepto de grafo, su representación y algoritmo visto en clase teórica.
- Desarrollo de la Guía práctica de estudio 6.
- Manejo de listas, diccionarios, estructuras de control, funciones y clases en Python 3.
- Conocimientos básicos de la programación orientada a objetos.

#### Introducción

Recorrer un grafo consiste en “visitar” cada uno de los vértices a través de las aristas del mismo.

La búsqueda en profundidad (del inglés Depth-First Search DFS) es un algoritmo para recorrer un grafo  $G = (V, A)$  visitando primero los vértices más profundos en  $G$  siempre que sea posible.

La estrategia de recorrido en profundidad (DFS), explora sistemáticamente las aristas de  $G$ , de manera que primero se visitan los vértices adyacentes a los visitados más recientemente. Se parte de un vértice  $v$  y el recorrido comienza por alguno de sus vértices adyacentes, luego un adyacente de este, y así sucesivamente hasta llegar a un vértice que ya no tiene vértices adyacentes que visitar. Luego la búsqueda vuelve atrás (*backtrack*) para seguir otro camino por algún vértice adyacente del último vértice visitado que aún tiene vértices adyacentes sin visitar. Este proceso continúa hasta que todos los vértices alcanzables desde el vértice  $v$  de la llamada original han sido descubiertos.

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II</b>	Código:	MADO-20
		Versión:	01
		Página	82/180
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

A igual que el algoritmo BFS visto en la guía 6, en el recorrido, cada vértice se colorea de blanco, gris o negro. Primero todos se inicializan en blanco, cuando el vértice se descubre por primera vez se colorea en gris y cuando toda su lista de adyacentes se ha examinado por completo se colorea en negro.

Durante el progreso del algoritmo también sucede lo siguiente:

- Cuando se descubre un vértice  $u$  durante la lectura de la lista de adyacencia de un vértice  $v$  ya descubierto, el algoritmo registra este evento asignando  $v$  al antecesor de  $u$  ( $v.pred = u$ ).
- También se registran dos tiempos, el primero  $u.d$  que es el instante o momento en que el vértice  $u$  se descubre (y se colorea de gris), y el tiempo en  $u.f$ , el instante de tiempo en el que se termina de examinar la lista de adyacentes a  $u$  (y se colorea de negro). Estos dos tiempos proveen información importante acerca de la estructura del grafo y ayudan a la inferencia del desarrollo del mismo.

A continuación, se presenta un algoritmo para DFS en pseudocódigo [1], donde los tiempos mencionados son enteros entre 1 y  $2|V|$  y para cada vértice  $u$ ,  $u.d < u.f$ .

DFS ( $G = (V, E)$ )

Inicio

Para cada vértice  $u \in V$

Inicio

$u.color = \text{blanco}$

$u.pred = \text{ninguno}$

Fin Para

Para cada vértice  $u \in V$

Inicio


Si  $u.color == \text{blanco}$

DFS-VISITAR( $G, u$ )

Fin Si

Fin Para

Fin

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II</b>	Código:	MADO-20
		Versión:	01
		Página	83/180
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

DFS-VISITAR( $G, u$ )

Inicio

tiempo=tiempo +1

$u.d$ =tiempo

$u.color$ =gris

Para cada vértice  $v$  que pertenece a la lista de adyacencia de  $u$

Inicio

Si  $v.color == \text{blanco}$

$v.pred = u$

DFS-VISITAR( $G, v$ )

Fin Si

Fin Para

$u.color$ =negro

tiempo=tiempo+1

$u.f$ =tiempo

Fin


Analizando la función DFS (), primero se utiliza una estructura de repetición para colorear todos los vértices del grafo en blanco e inicializar su atributo del predecesor en “ninguno”. Una vez terminado se inicializa el tiempo en cero y con otra estructura de repetición se revisa cada vértice  $u \in V$  en turno; cuando está en blanco realiza una visita en profundidad utilizando la función DFS-VISITAR () y al término de esta, al vértice  $u$  se le habrá asignado un tiempo de descubrimiento y el tiempo de término de examinar su lista de adyacencia.

Una característica importante que cabe señalar es que cada vez que se llama a la función DFS-VISITAR( $G, u$ ) el vértice  $u$  se convierte en raíz de un nuevo árbol de búsqueda en profundidad y el conjunto de árboles forma un bosque de árboles de búsqueda o subgrafo de predecesores  $G_{pred}$ , definido de la siguiente manera:

$G_{pred} = (V, A_{pred})$  donde  $A_{pred} = \{(v.pred, v) : v \in V \text{ y } v.pred \neq \text{ninguno}\}$

En cada llamada a la función DFS-VISITAR( $G, u$ ), para el vértice  $u$  visitado, se modifica el tiempo incrementándolo en uno, se registra en  $u.d$  como tiempo de descubrimiento y se colorea  $u$  de gris. Después se examina con una estructura de repetición cada vértice  $v$  adyacente a  $u$  y si este está en blanco se le realiza una visita de forma recursiva, aquí se dice que cada arista  $(u, v)$  se exploró por búsqueda primero en profundidad.

Una vez terminada esta exploración se colorea  $u$  de negro se incrementa el tiempo y se registra en  $u.f$  como de término de la exploración de  $u$  por DFS.

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II</b>	Código:	MADO-20
		Versión:	01
		Página	84/180
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Una propiedad de este algoritmo es que se puede usar para la clasificación de aristas del grafo de entrada, información que es útil para implementar otros algoritmos como por ejemplo para verificar si un grafo es acíclico.

Se pueden definir cuatro tipos de aristas: **Aristas de árbol:** Son las aristas de un árbol de búsqueda en profundidad y por tanto del bosque *Gpred*. La arista  $(u, v)$  es una arista de árbol si  $v$  se alcanzó por primera vez al recorrer dicha arista [2].

**Aristas de retorno:** Son las aristas  $(u, v)$  que se conectan al vértice  $u$  con un antecesor  $v$  de un árbol de búsqueda en profundidad. Las aristas cíclicas se consideran aristas de retorno [2][1].

**Aristas de avance:** Son las aristas  $(u, v)$  que no pertenecen al árbol de búsqueda. Pero conectan un vértice  $u$  con un descendiente  $v$  que pertenece al árbol de búsqueda en profundidad [2].

**Aristas de cruce:** Son todas las otras aristas, que pueden conectar vértices en el mismo árbol de búsqueda en profundidad o en dos árboles de búsqueda en profundidad diferentes [2][1].

A continuación, se muestra la evolución de la búsqueda por profundidad para un grafo dirigido, figura 7.1-figura 7.8, considerar que dentro del vértice se coloca el tiempo de *descubrimiento/tiempo de finalización*. Las aristas sombreadas indican que forman parte de un árbol mientras si no lo son, se representan como líneas discontinuas y son etiquetadas con *B*, *C* o *F* para indicar si son aristas de retorno, de cruce o de avance respectivamente.

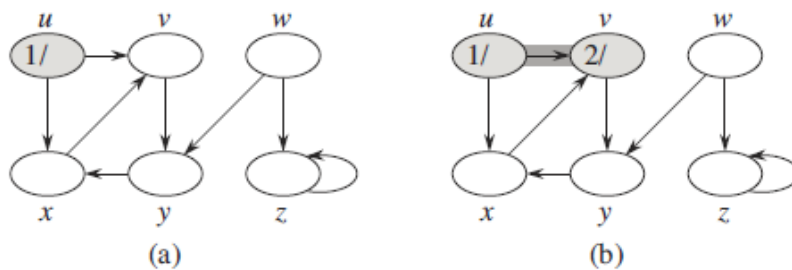


Figura 7.1 [1]

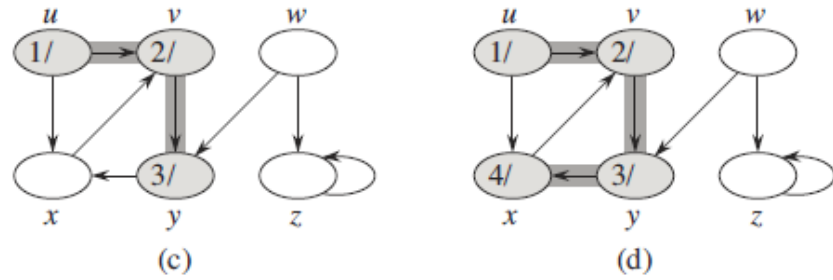


Figura 7.2 [1]

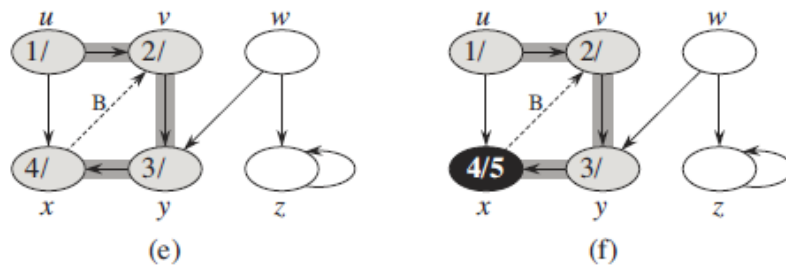


Figura 7.3 [1]

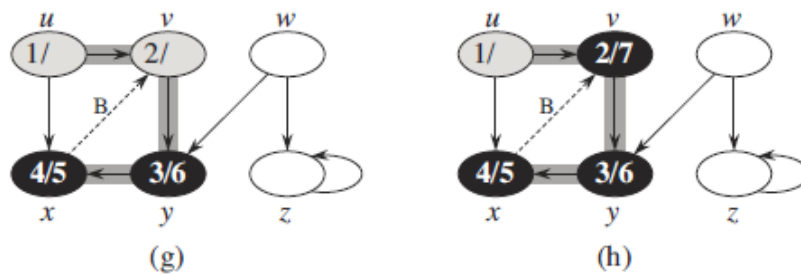



Figura 7.4 [1]

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II</b>	Código:	MADO-20
		Versión:	01
		Página	86/180
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

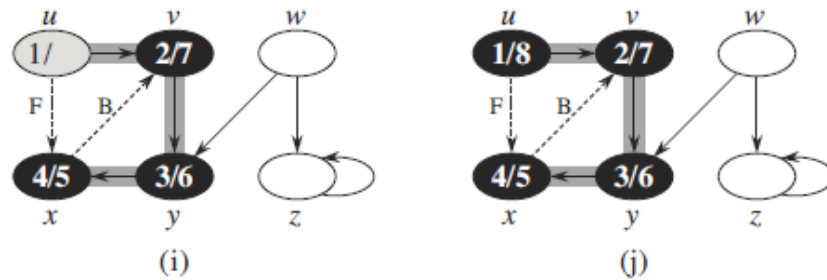


Figura 7.5 [1]

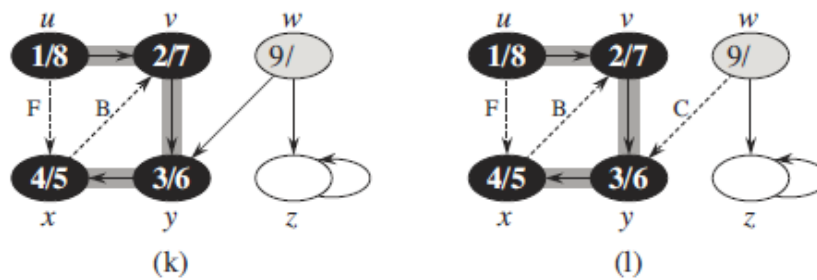


Figura 7.6 [1]

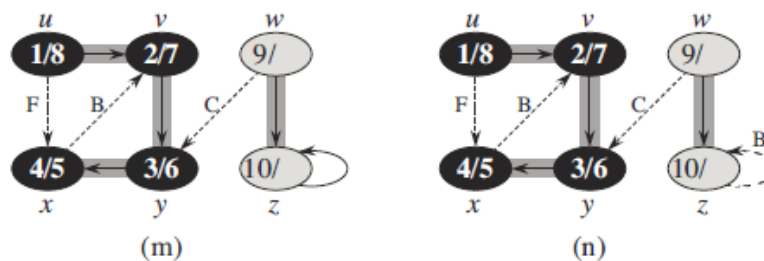



Figura 7.7 [1]

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II</b>	Código:	MADO-20
		Versión:	01
		Página	87/180
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

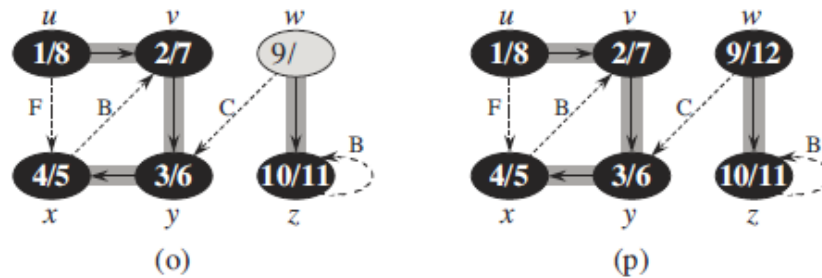


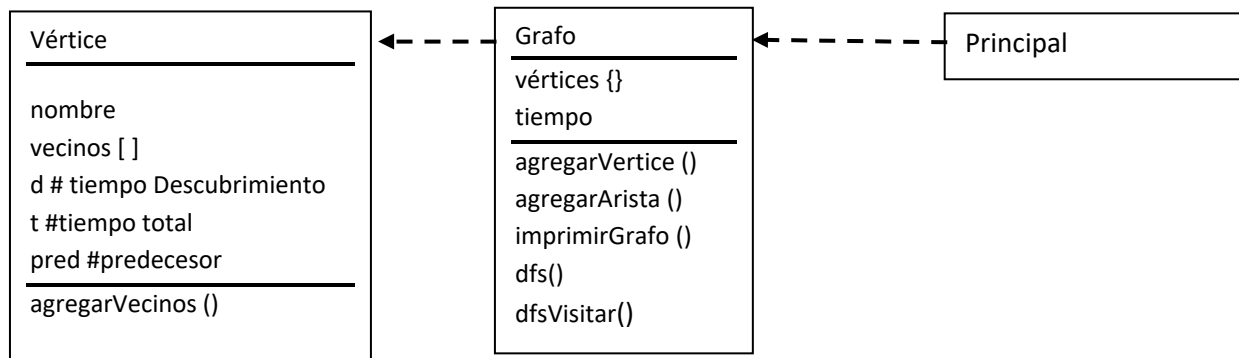
Figura 7.8 [1]

## Desarrollo

### Actividad 1


Realizar un programa donde se implemente el algoritmo DFS que se analizó en esta guía utilizando el pseudocódigo explicado del mismo, además probar que la salida coincide con los resultados del ejemplo mostrado en las figuras 7.1 - 7.8 .

Para ello se plantea el algoritmo en el siguiente diagrama de clases:



A continuación, se dan las implementaciones de las clases *Grafo* y *Vertice* en Python, y como parte de la actividad se debe realizar la controladora que contenga un método que dé la secuencia de solución para la búsqueda por profundidad del grafo dirigido de la figura 7.1.



	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II</b>	Código:	MADO-20
		Versión:	01
		Página	88/180
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

La clase *Vertice* es:

```
class Vertice:
    def __init__(self, n):
        self.nombre = n
        self.vecinos = list()

        self.d = 0 # tiempo de descubrimiento
        self.f=0 # tiempo de término
        self.color = 'white'
        self.pred = -1


    def agregarVecino(self, v):
        if v not in self.vecinos:
            self.vecinos.append(v)
            self.vecinos.sort()
```

Para la clase *Grafo* primero se muestran los atributos y después por separado cada uno de los métodos:

```
class Grafo:
    vertices = {}
    tiempo = 0
```

```
def agregarVertice(self, vertice):
    if isinstance(vertice, Vertice) and vertice.nombre not in self.vertices:
        self.vertices[vertice.nombre] = vertice
        return True
    else:
        return False
```

```
def agregarArista(self, u, v):
    if u in self.vertices and v in self.vertices:
        for key, value in self.vertices.items():
            if key == u:
                value.agregarVecino(v)
                #if key == v: #Se comenta porque es grafo dirigido
                #    value.agregarVecino(u)
        return True
    else:
        return False
```

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II</b>	Código:	MADO-20
		Versión:	01
		Página	89/180
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

```
def imprimeGrafo(self):
    for key in sorted(list(self.vertices.keys())):
        print("Vertice: "+key)
        print("Descubierto/Termino:"+str(self.vertices[key].d)+"/"+str(self.vertices[key].f))
```

```
def dfs(self,vert):
    global tiempo
    tiempo=0
    for u in sorted(list(self.vertices.keys())):
        if self.vertices[u].color=='white':
            self.dfsVisitar(self.vertices[u])
```

```
def dfsVisitar(self,vert):
    global tiempo
    tiempo = tiempo + 1
    vert.d=tiempo
    vert.color='gris'

    for v in vert.vecinos:
        if self.vertices[v].color == 'white':
            self.vertices[v].pred=vert
            self.dfsVisitar(self.vertices[v])
    vert.color="black"
    tiempo=tiempo+1
    vert.f=tiempo
```

## Actividad 2

Ejercicios sugeridos por el profesor

## Referencias

[1] CORMEN, Thomas, LEISERSON, Charles, et al.

Introduction to Algorithms

3rd edition

MA, USA

The MIT Press, 2009

[2] Ziviani, Nivio

Diseño de algoritmos con implementaciones en Pascal y C

Ediciones paraninfo /Thomson Learning

2007