Grafos

Se presenta en este módulo, como lectura complementaria a los capítulos de Grafos del texto de clase: una lista de conceptos que deben ser definidos con precisión por los alumnos, los elementos necesarios para la definición de un TDA Grafo y los esquemas básicos para definir las diferentes estructuras de datos que se pueden usar para representar un grafo.

Conceptos básicos

Grafo dirigido	Subgrafo
Grafo no dirigido	Camino
Grafo etiquetado o rotulado	Longitud de camino
Multi-grafo	Camino simple
Vértice o Nodo	Ciclo
Arco o Arista	Ciclo simple
Arista paralela	Grafo conexo
	Componente conexa

Es importante destacar que al hablar de *grafo* nos referimos a una familia de modelos. Para definir uno en particular se debe especificar:

- si es dirigido o no dirigido,
- si puede tener aristas paralelas o es sin aristas paralelas,
- si es con vértices rotulados o con vértices no rotulados,
- si es con arcos rotulados o con arcos sin rótulos.

Por ejemplo, podemos definir, entre otros, los siguientes modelos:

- Grafo dirigido con aristas paralelas, con vértices rotulados y con arcos no rotulados,
- Grafo dirigido sin aristas paralelas, con vértices y arcos rotulados,
- Grafo no dirigido con aristas paralelas, con vértices no rotulados y con arcos con rótulos

Es decir, se pueden definir diferentes TDA: Grafo, según el modelo que se elija, y según el conjunto de operaciones que se seleccione. Se presenta a continuación un conjunto de operaciones que permiten completar la definición de un TDA Grafo: primero, sólo con alcance para hacer recorridos y recavar información del mismo y luego se amplía dicho conjunto de operaciones con otras que permiten crear y actualizar grafos.

TDA: Grafo

I- Definición del TDA: Grafo

1. **Modelo:** Se debe especificar el tipo de modelo que se elige**.**

Observación: de acá en más al decir grafo se hace referencia a uno que responde al modelo elegido en la definición.

2. Conjuntos involucrados (aquellos a los que pertenecen los argumentos de las operaciones) :

- Grafo = { g/g es un grafo }
- Vértice = { v/v es un vértice de un grafo } ∪ { Nodo Nulo }
- Rótulo_Vértice = { R/R es un rótulo de un vértice de un grafo }
- Rótulo_Arco = { R/R es un rótulo de un arco de un grafo }
- Índice = { i/i es la dirección que permite encontrar un vértice adyacente a un vértice dado } ∪ { Índice Nulo }

Para realizar un manejo más simple en la definición de las operaciones se consideran las constantes:

Nodo Nulo: es un valor constante compatible con los valores con los que se representan los vértices de un grafo. Se trata de un valor especial que nunca coincide con un vértice que existe en un grafo.

Índice Nulo: análogamente al caso anterior, se trata de es un valor constante compatible con los valores con los que se representan los índices de los vértices adyacentes a un nodo de un grafo. Se trata de un valor especial que nunca coincide con los de un índice en un grafo.

3. Operaciones y sus respectivas descripciones:

3.1. Con alcance para hacer recorridos y recavar información del grafo

- Primer Vértice: Grafo→Vértice
  Primer Vértice (G): Vértice

  Como el número de vértices de un grafo es finito, se puede hablar de la lista de vértices (siempre se los puede organizar como una lista, sin que esto implique que hay uno distinguido). Esta operación devuelve el primer vértice de la lista de vértices del grafo G. Si G no tiene nodos, devuelve Nodo Nulo.

- Siguiente Vértice: (Vértice × Grafo) → Vértice
  Siguiente Vértice (V,G): Vértice

  Esta operación devuelve el primer vértice que sigue a V en la lista de vértices del grafo G. Si G no tiene nodos o V=Nodo Nulo se plantea una situación de error. Cuando V es el último vértice de la lista de vértices de G la operación devuelve Nodo Nulo.

- Primer Adyacente: (Vértice × Grafo) → Índice
  Primer Adyacente (V,G): Índice

Como el número de vértices de un grafo es finito, obviamente el número de vértices adyacentes a uno dado, también lo es. Por lo tanto se puede hablar de la lista de vértices adyacentes (o lista de adyacentes) a un vértice V (siempre se los puede organizar como una lista, sin que esto implique que hay uno distinguido). Esta operación devuelve el índice del primer vértice de la lista de adyacentes del vértice V del grafo G. Si G no tiene nodos o V=Nodo Nulo se plantea una situación de error. Si V no tiene nodos adyacentes devuelve Índice Nulo.

- Siguiente Adyacente: (Vértice × Índice × Grafo) → Índice
  Siguiente Adyacente (V,I,G): Índice

  Esta operación devuelve el índice del siguiente vértice adyacente al de índice I de la lista de adyacentes del vértice V en el grafo G. Si V=Nodo Nulo o I=Índice Nulo o G no tiene nodos, se plantea una situación de error. Cuando I es el índice del último adyacente de la lista de adyacentes del vértice V, la operación devuelve Índice Nulo.

- Vértice: (Índice × Grafo) → Vértice
  Vértice (I,G): Vértice

  Esta operación devuelve el vértice que se encuentra en la dirección I correspondiente a la lista de adyacentes de V. Existirá error si I=Índice Nulo o si G no tiene nodos.

- Rótulo Vértice: (Vértice × Grafo) → Rótulo_Vértice
  Rótulo Vértice (V, G): Rótulo_Vértice

  Esta operación es sólo aplicable a grafos con vértices rotulados. Entrega la información asociada al vértice V del grafo G. Si G no tiene nodos o si V=Nodo Nulo se plantea una situación de error.

- Rótulo Arco: (Vértice × Vértice × Grafo) → Rótulo_Arco
  Rótulo Arco (V_1,V_2,G) → Rótulo_Arco

  Esta operación es sólo aplicable a grafos con arcos rotulados. Entrega la información asociada al arco (V_1,V_2) del grafo G. Si G no tiene nodos o si no existe el arco (V_1,V_2) se plantea una situación de error.

3.2. Para ampliar el alcance del TDA permitiendo la creación y/o modificación de un grafo, se incorporan las siguientes operaciones:

- Crear Grafo Nulo: Grafo → Grafo
  Crear Grafo Nulo (G)

  Al ejecutarse esta operación el grafo G queda sin nodos y en condiciones de recibirlos. Es aplicable a cualquier grafo.

- Insertar Vértice: (Rótulo_Vértice × Grafo) → Grafo × Vértice

Insertar Vértice (R, G): Vértice

Esta operación incorpora un nuevo vértice V en el grafo G. Si se trata de un grafo con vértices rotulados, le asocia a V el rótulo R (siempre que se trate de un grafo con vértices rotulados, en caso contrario no debería figurar el argumento R). Devuelve el valor V.

- Insertar Arco: (Rótulo_Arco × Vértice × Vértice × Grafo) → Grafo × Índice
  Insertar Arco (R,V_1,V_2,G): Índice

  Esta operación incorpora un nuevo arco (V_1,V_2) en el grafo G. Si se trata de un grafo con arcos rotulados le asocia a dicho arco el rótulo R (siempre que se trate de un grafo con arcos rotulados, en caso contrario no debería figurar el argumento R). Devuelve el valor I que es la dirección del vértice V_2, en la lista de adyacentes a V_1 en el grafo G.

- Eliminar Vértice: (Vértice × Grafo) → Grafo
  Eliminar Vértice (V, G)

  Esta operación elimina el vértice V del grafo G. Elimina también todos los arcos en los que V está involucrado. Si V no es un vértice de G, G permanece sin modificaciones.

- Eliminar Arco: (Vértice × Vértice × Grafo) → Grafo
  Eliminar Arco (V_1,V_2,G)

  Si (V_1,V_2) es un arco del grafo G entonces esta operación lo elimina, si no el grafo G permanece sin modificaciones. Si se trata de un multi-grafo, se deberá incorporar un parámetro más que permita identificar el arco a eliminar.

II- Implementación del TDA: Grafo

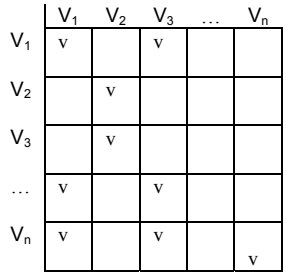
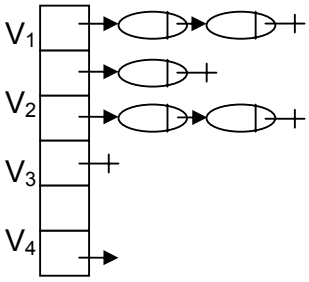
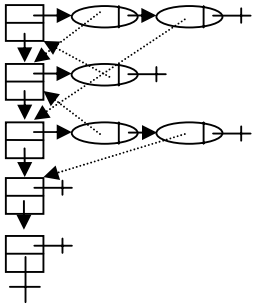
Para implementar un TDA se debe:

1. elegir una estructura de datos adecuada para representar el modelo especificado en la definición
2. hacer los procedimientos (o funciones) correspondientes a las operaciones especificadas en la definición

Se presentan a continuación (en forma esquemática) diferentes propuestas para la elección de la estructura de datos. En base a ellas, los alumnos deben hacer las correspondientes definiciones. También deben implementar las operaciones del TDA, para ello, obviamente, es indispensable seguir fielmente la descripción de las mismas.

Estructuras de datos para representar un grafo

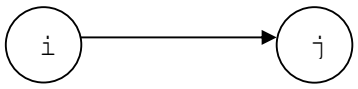
Para representar grafos existen tres esquemas básicos; según el modelo del grafo, dichos esquemas se deben completar adecuadamente.
Los esquemas están pensados para grafos dirigidos. Para el caso de grafos no dirigidos, se describe al final de este apartado. Las alternativas que se proponen son:

Matriz de Adyacencias	**Arreglo de listas Adyacencias**	**Lista de Listas de Adyacencias**
		

1- Representación mediante una matriz de adyacencias

Se trata de una representación mediante un arreglo de dos dimensiones (matriz). Los índices de la matriz representan los vértices del grafo y cada una de las componentes [i,j] toma un valor lógico según el arco (i,j) pertenezca o no al grafo que representa.
En este caso, los conjuntos Vértice e Índice coinciden.

Para el siguiente ejemplo



El esquema es:

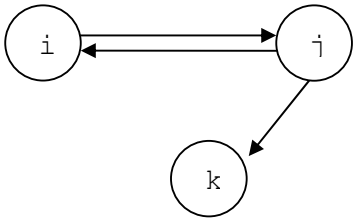
	1	2	i	…	j	…	LM
2							
…							
i					v		
…							
j			f				
…							
LM							

– Si el grafo tiene los vértices rotulados, la estructura se debe completar con un mapeo Vértice→Rótulo_Vértice.
– Si el grafo tiene los arcos rotulados, las componentes de la matriz además deberán alojar esa información.
– Si se trata de un multi-grafo, cada una de las componentes [i,j] de la matriz alojará una lista con los rótulos de los arcos (i,j). Si (i,j) no es un arco del grafo, la componente [i,j] tendrá una lista vacía.
– Si se quiere dar cierta flexibilidad a la representación en relación con el número de vértices, puede declararse una matriz más grande, siempre guardar el grafo desde la componente [1,1] y completar la estructura con un cursor al último vértice. De esta forma se puede trabajar con operaciones de actualización, siempre con la limitación que implica un arreglo como estructura subyacente.

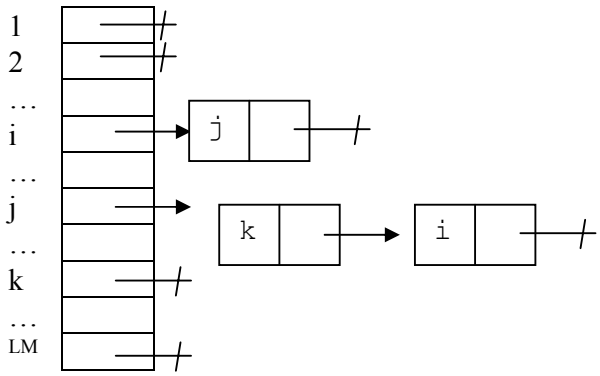
2- Representación mediante una arreglo de listas de adyacencias

Los índices del arreglo representan los vértices del grafo y cada una de las componentes [i] tiene un enlace a la lista de adyacentes al vértice i.
En este caso el tipo Índice está compuesto por las direcciones de las celdas de las listas de adyacentes y el enlace Nulo.

Es decir, al siguiente caso:



Le corresponde:



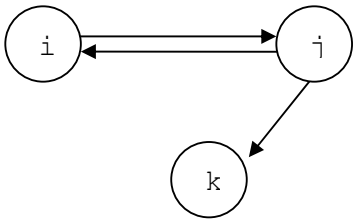
– Si el grafo tiene los vértices rotulados, la estructura se debe completar con un mapeo Vértice→Rótulo_Vértice.

– Si el grafo tiene los arcos rotulados, las celdas de las listas de adyacentes deberán alojar dicha información.
– Para dar cierta flexibilidad a la representación respecto del número de vértices, puede declararse un arreglo con más componentes, siempre guardar el grafo desde la componente [1] en lugares consecutivos y completar la estructura con un cursor al último vértice del arreglo. De esta forma se puede trabajar con operaciones de actualización, siempre con la limitación que implica un arreglo como estructura subyacente.

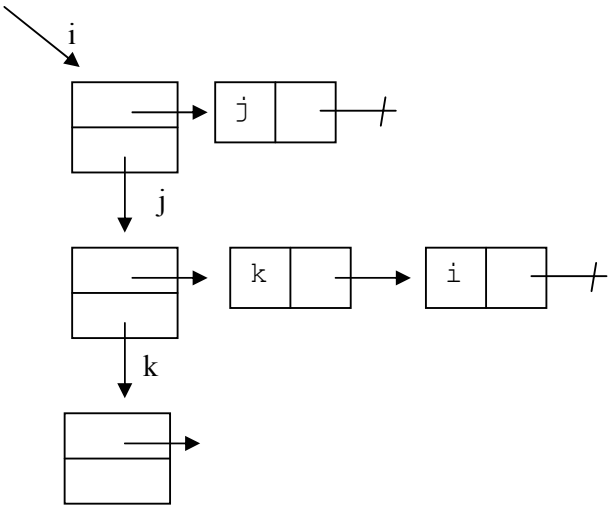
3- Representación mediante una lista enlazada de listas de adyacencia

Con el propósito de flexibilizar más la propuesta anterior, en este caso se reemplaza el arreglo de vértices por una lista enlazada de vértices. Cada celda tendrá un apuntador a la lista de adyacentes correspondiente.
En esta opción cada vértice es la dirección de la celda correspondiente en la lista enlazada de vértices y los índices son las direcciones de las celdas de las listas de adyacentes.

Entonces, para el caso:



El esquema de representación es:



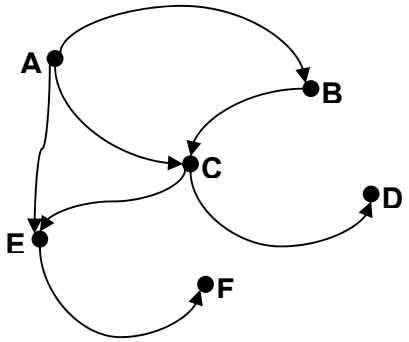
– Si el grafo tiene los vértices rotulados, la estructura se debe completar incorporando la información en las celdas de la lista de vértices.
– Si el grafo tiene los arcos rotulados, las celdas de las listas de adyacentes deberán alojar dicha información.

Para el caso de un grafo no dirigido *G*, se los representa primero como un grafo dirigido equivalente *G´* y luego se elige alguna de las estructuras de datos presentadas anteriormente. *G´* se obtiene a partir de G de la siguiente manera:

Para cada vértice v de *G , v* es vértice de *G´*
Para cada arco (v_1,v_2) de *G* se cumple que (v_1,v_2) y (v_1,v_2) deben pertenecer a *G´*

Recorridos de Grafos

El recorrido de un grafo, o su navegación, se realiza siguiendo las relaciones de adyacencia, es decir a través de las conexiones entre los nodos. En otras palabras desde un vértice se puede ir solamente a uno de sus adyacentes. Recorrerlo significa pasar una y sólo una vez por cada nodo, cumpliendo en él con la tarea que se deba hacer según el caso; dicha tarea se llamará genéricamente *Visitar*.



Vértices	Adyacentes
A	B, C, E
B	C
C	D, E
D	
E	F
F	

Según la manera en que se vayan recorriendo los adyacentes de cada vértice del grafo podemos distinguir:

1. Recorrido en Profundidad
2. Recorrido en Anchura

1. Recorrido en Profundidad

La estrategia en la que se basa este recorrido es la misma que aplicamos en el recorrido en *preorden* de los árboles.
Dado un *digrafo* como el de la figura, la idea es que partiendo de uno de los vértices avanzar lo más lejos que se pueda, a través de uno de los adyacentes (por ejemplo el primero de ellos). Una vez que no se puede avanzar más, se continúa a partir del próximo adyacente del último visitado.
Para el ejemplo mostrado si comenzamos el recorrido a partir del vértice rotulado con la *A*, este recorrido nos permitiría visitar los restantes vértices en el siguiente orden:
A, B, C, D, E, F.
Analicemos este recorrido. Comenzamos con el vértice rotulado con A y tomando como convención que de los adyacentes a *A* elegimos el primero, luego visitamos el vértice rotulado con *B*. Una vez alcanzado el vértice con rótulo *B*, pasamos a su único

adyacente el rotulado con *C*. De los dos adyacentes de éste, los vértices rotulados con *D* y *E* respectivamente, optamos por el primero visitando de esta manera al que tiene rótulo *D*. En este punto no podemos avanzar más pues *D* no tiene adyacentes, razón por la cual retomamos los adyacentes del vértice visitado inmediatamente antes. Como ese vértice fue el rotulado con *C*, continuamos el recorrido con el próximo de sus adyacentes el rotulado con *E*. Una vez visitado el vértice con rótulo *E*, continuamos el recorrido por su primer adyacente, el vértice con rótulo *F*. Visitamos *F* y nuevamente como éste no tiene adyacentes debemos retornar y continuar con los adyacentes a *C*. Dado que para nuestro ejemplo el vértice rotulado con *E* era el último de los adyacentes a *C*, debemos continuar explorando los adyacentes que dejamos pendientes al pasar a *C*. En otras palabras debemos tomar el próximo adyacente al vértice con rótulo *B*. En el digrafo ejemplo, *B* tenía un único adyacente, por lo cual esa instancia ya está completa. El recorrido por consiguiente debería retomar los adyacentes a *A* a partir del último explorado. Siendo que de estos vértices el único explorado fue el rotulado con *B*, el recorrido debería considerar a *C* que es el próximo de sus adyacentes.
En este punto debemos detenernos en el hecho de que ya visitamos dicho vértice. Si lo que nosotros estamos planteando es recorrer todos los vértices del digrafo, obviamente pasando por cada vértice una sola vez, no debemos visitarlo nuevamente. Además como ya lo visitamos, también hicimos lo propio con todos los demás vértices que se pueden alcanzar a partir del mismo.
Situación similar a la anterior es la que enfrentamos al intentar el recorrido con el próximo de los adyacentes al vértice con rótulo *A*, el vértice con rótulo *E*. Este vértice ya fue visitado por lo cual deberíamos descartarlo y continuar con el siguiente. Como este vértice es el último de los adyacentes al rotulado con *A*, finalizamos nuestro recorrido.
La situación anterior nos lleva a plantearnos la necesidad de llevar un control o un registro de los vértices que se han ido visitando previamente para no volver a visitarlos.
Contar con un registro tal, nos permite detectar la presencia de *ciclos* en el grafo que estamos recorriendo. Si en el digrafo del ejemplo hubiera una arista con origen en el vértice rotulado con *C* y destino en el de rótulo *A*, al recorrerlo quedaríamos atrapados en el ciclo *ABC*. Esto lo evitamos al saber que el vértice con rótulo *A* ya había sido visitado anteriormente.

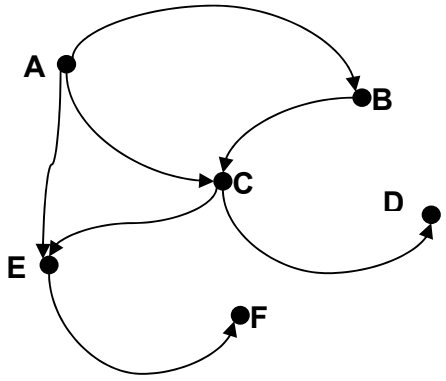
A continuación planteamos un algoritmo en término de las operaciones del TDA Grafo que recorre un grafo G en profundidad a partir de un vértice V. El registro de los vértices visitados lo hacemos utilizando un *conjunto de vértices* para tal fin, al que llamamos *Visitados*.

```
Profundidad ( G: Grafo, V: Vértice, VAR Visitados: Conjunto); (* V es vértice
no nulo y en la primera invocación del procedimiento Visitados={V}  *)

Var
        Ady:  Índice;
        Vert_Ady: Vértice;


Begin
```

```
        Ady := Primer Adyacente( V, G);
        Mientras Ady ≠ Índice_Nulo hacer
                Vert_Ady := Vértice ( V, Ady, G);
                Si Vert_Ady ∉ Visitados
                        Entonces
                                Insertar ( Vert_Ady, Visitados);
                                Profundidad ( Vert_Ady, G, Visitados)
                Ady := Siguiente Adyacente ( V, Ady, G)
End;
```

2. Recorrido en Anchura

La idea de este recorrido es la misma que inspira el recorrido por niveles visto para árboles. Partiendo de un vértice, se lo visita y a continuación se hace lo propio con todos sus adyacentes. Una vez visitados todos, se procede a visitar los adyacentes de cada de uno de ellos, para luego continuar el recorrido a partir de cada uno de ellos y así sucesivamente.
Para el ejemplo planteado:

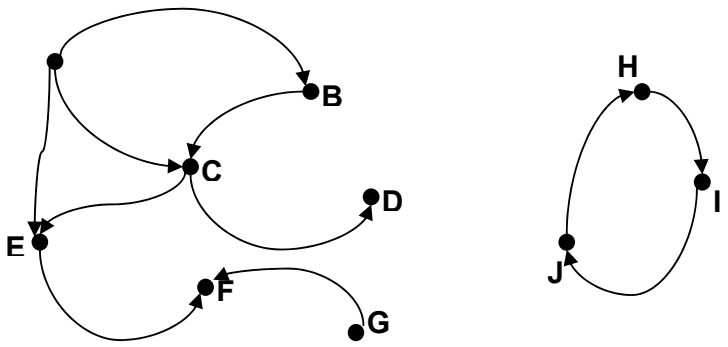


Si se comienza a recorrer en anchura a partir del vértice rotulado con *A*, los vértices se irán visitando en el siguiente orden: *A, B, C, E, D, F*.
Lo planteado en el recorrido anterior sobre los controles necesarios para evitar visitar vértices más de una vez y sobre la detección de ciclos es válido también en este caso. También lo es la solución presentada de llevar un registro de los vértices visitados.

A continuación presentamos un algoritmo para recorrer en *Anchura* a un grafo G a partir de un vértice V. Como en el caso del recorrido por niveles para árboles usaremos una Cola de Vértices auxiliar para ir manteniendo los vértices, a medida que se los visita, para poder obtener después sus adyacentes. El control de los vértices visitados lo haremos utilizando como en el recorrido en *Profundidad* un conjunto de vértices al que llamamos nuevamente *Visitados*.

```
Anchura ( G: Grafo, V: Vértice, VAR Visitados: Conjunto);

Var
        Ady:  Índice;
        Vert_Ady: Vértice;
        C: Cola (* cola de vértices *)

Begin
        Crear Cola Vacía (C);
        Poner En Cola (V, C);

        Mientras No (ColaVacía C))
                Vert := Frente (C);
                Retirar De Cola (C);
                Ady := Primer Adyacente( V, G);
                Mientras Ady ≠ Índice_Nulo hacer
                        Vert_Ady := Vértice ( V, Ady, G);
                        Si Vert_Ady ∉ Visitados
                                Entonces
                                        Insertar ( Vert_Ady, Visitados);
                                        Poner En Cola ( Vert_Ady, C)
                        Ady := Siguiente Adyacente ( V, Ady, G)
End;
```

Los recorridos planteados pueden llegar a no ser suficientes. Analicemos su comportamiento para el siguiente grafo no conexo:



Uno de los problemas que nos enfrentamos al momento de recorrer un grafo, es que el mismo puede tener más de una componente conexa. Este hecho nos evidencia la necesidad de contar con un algoritmo que plantee un recorrido a partir de cada vértice del grafo, asegurando que no visitemos más de una vez a cada uno de ellos. Para realizar este control utilizaremos el conjunto de vértices *Visitados* que presentamos en los recorridos anteriores.
A continuación se muestra un algoritmo que realiza lo propuesto:

```
Recorrido Abarcador (G: Grafo);

VAR
        Visitados: Conjunto de Vértices;
        V: Vértice;

Begin
        Crear Conjunto Vacío (Visitados);
        V:= Primer Vértice (G);
        Mientras V ≠ Vértice Nulo
                Si V ∉ Visitados
                        Entonces
                                Insertar (V, Visitados);
                                Profundidad (V, G, Visitados);
                                (* En este punto se podría invocar a Anchura*)
                V:= Siguiente Vértice(V, G)
End;
```