Tema 2: Grafos y Árboles

Algoritmos y Estructuras de Datos 3

ÍNDICE

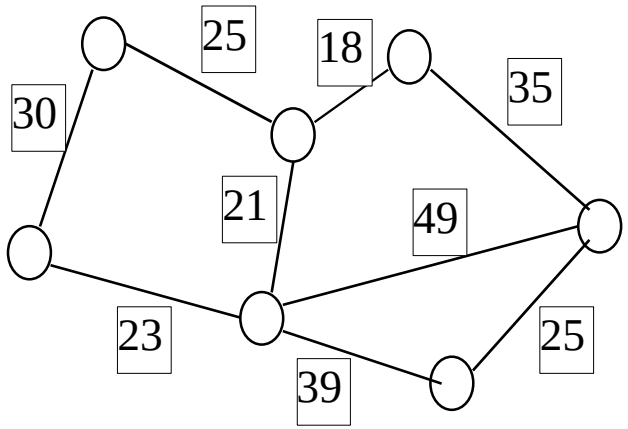
BIBLIOGRAFÍA

- T.H. Cormen, C.E. Leiserson, R.L. Rivest
  Introduction to Algorithms. MIT Press, 1990. Capítulos 5 y 23.
- Aho A.V., Hopcroft J.E., Ullman J.E.
  Estructuras de datos y Algoritmos.
  Addison-Wesley, 1988. Capítulos 6 y 7.

1. DEFINICIONES BÁSICAS

- Un grafo permite representar relaciones binarias entre los elementos de un conjunto.

  Ejemplo: mapa de carreteras.

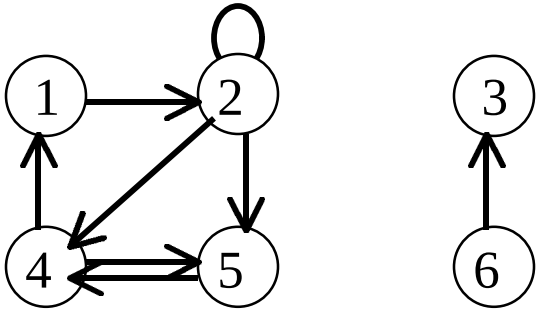


Definiciones básicas

- Grafos dirigidos y no dirigidos
- Relaciones de incidencia y adyacencia
- Caminos
- Subgrafos
- Conectividad
- Grafos etiquetados
- Árboles
- Relaciones entre nodos de un árbol

Grafos Dirigidos

- Un grafo dirigido (g.d.) es un par G=(V,E)
  - V es un conjunto finito de vértices
  - E es un conjunto de arcos (o aristas). Un arco es un par ordenado(u,v) con u,v∈ V

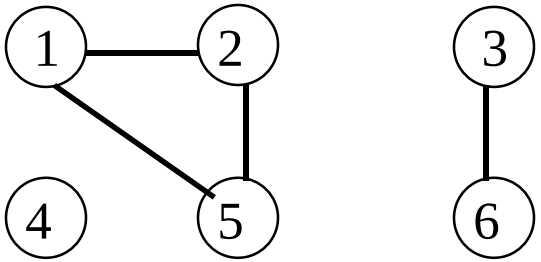


V={1,2,3,4,5,6}

E={(1,2),(2,2),(2,4),(2,5),(4,1),
(4,5),(5,4),(6,3)}

Grafos no dirigidos

- Un grafo no dirigido (g.n.d) es un par G=(V,E)
  - V es un conjunto finito de vértices
  - E es un conjunto de arcos. Un arco es un par NO ordenado (u,v) con u,v∈ V, u≠v



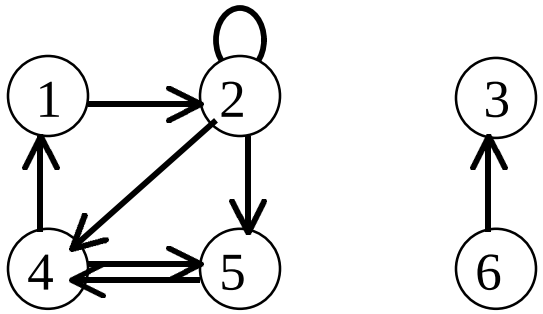
V={1,2,3,4,5,6}

E={(1,2),(1,5),(2,5),(3,6)}

Relaciones de Incidencia y Adyacencia

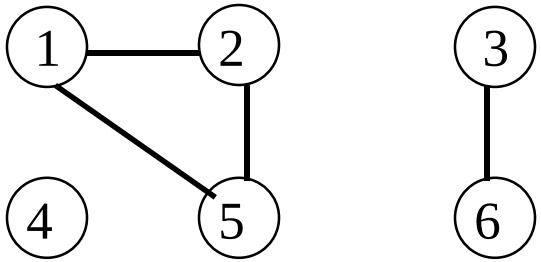
- Sea G=(V,E) un grafo dirigido. Si (u,v)∈ E, decimos que incide desde u (sale de..) e incide en v (llega a..).

  Ejemplo: vértice 2



Relaciones de Incidencia y Adyacencia (cont.)

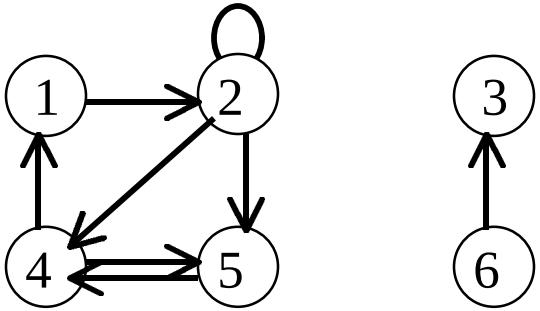
- Sea G=(V,E) un grafo no dirigido. Si (u,v)∈ E, decimos que incide sobre u y v.
- Ejemplo: vértice 2



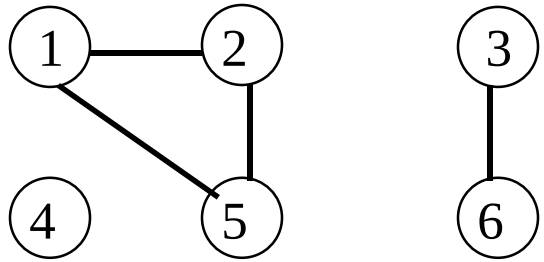
Relaciones de Incidencia y Adyacencia (cont.)

- Sea G=(V,E) un grafo. Si (u,v)∈ E, decimos que el vértice v es adyacente al u.
  - La relación es simétrica en grafos no dirigidos

2 es adyacente a 1
1 NO es adyacente a 2

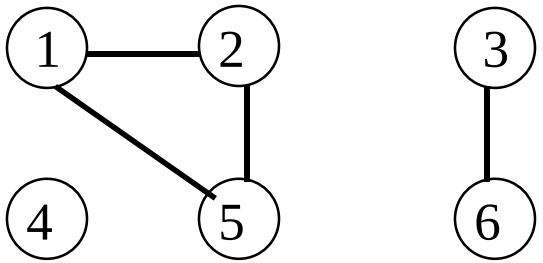


2 es adyacente a 1
1 es adyacente a 2



Relaciones de Incidencia y Adyacencia (cont.)

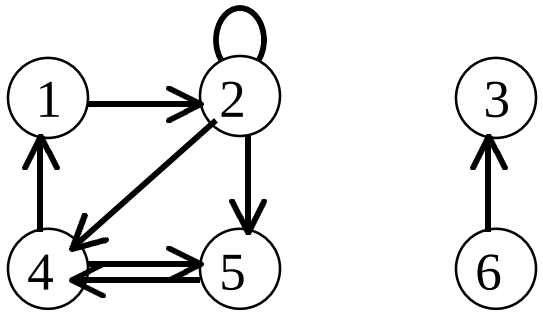
- Llamaremos grado de un vértice en un g.n.d. al nº de arcos que inciden sobre él.



El grado de 2 es 2

Relaciones de Incidencia y Adyacencia (cont.)

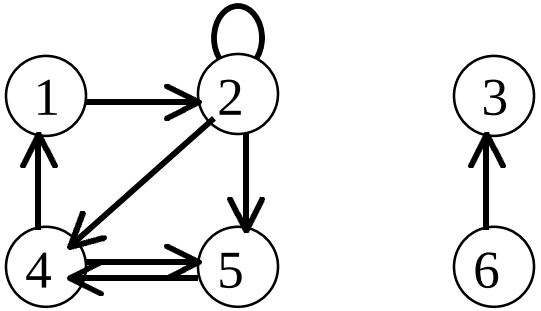
- El grado de un vértice en un g.d. es el nº de arcos que salen de él (grado de salida) más el nº de arcos que entran (grado de entrada).



Grado de entrada del 2=2, Grado de salida del 2 =3
Grado de 2=5

Relaciones de Incidencia y Adyacencia (cont.)

- El grado de un grafo es el del vértice de máximo grado.



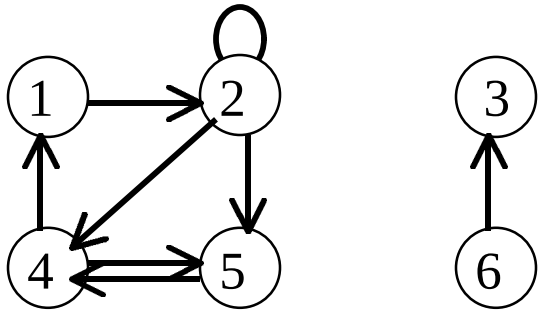
El grado de este grafo es 5

Caminos

- Un camino de longitud k desde u a u’ en un grafo G=(V,E) es una secuencia de vértices $\langle v_0,v_1,..,v_k\rangle$ tal que v_o=u y v_k=u’ y $\forall$i:1..k:$(v_{i-1},v_i)\in$ E. La longitud del camino es el número de arcos.
- Si hay un camino P desde u hasta u’, decimos que u’ es alcanzable desde u via P.

Caminos (cont.)

- Un camino es simple si todos sus vértices son distintos.

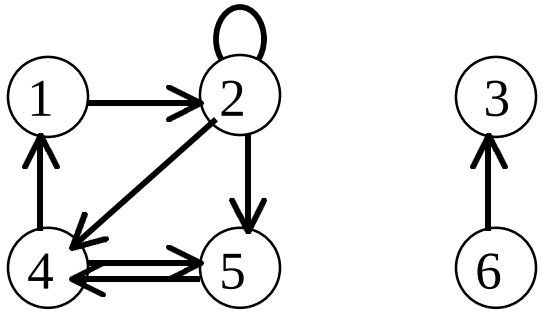


<1,2,5,4> es un camino simple de longitud 3

<2,5,4,5> no es un camino simple

Caminos (cont.)

- En un g.d. un camino $<v_0,v_1,..,v_k>$ forma un ciclo si $v_0=v_k$ y el camino contiene al menos un arco.
  - El ciclo es simple si los vértices son distintos
  - Un bucle es un ciclo de longitud 1



<1,2,5,4> es un ciclo

Caminos (cont.)

- En un g.n.d. un camino $<v_0,v_1,..,v_k>$ forma un ciclo si $v_0=v_k$ y los v_i son distintos.
- Un grafo sin ciclos diremos que es acíclico

Subgrafos

- Un grafo G’=(V’,E’) es un subgrafo de G=(V,E) si V’$\subseteq$V y E’$\subseteq$E.
- Dado un conjunto V’$\subseteq$V, el subgrafo de G inducido por V’ es G’=(V’,E’):E’={(u,v)$\in$ E:u,v$\in$ V’}

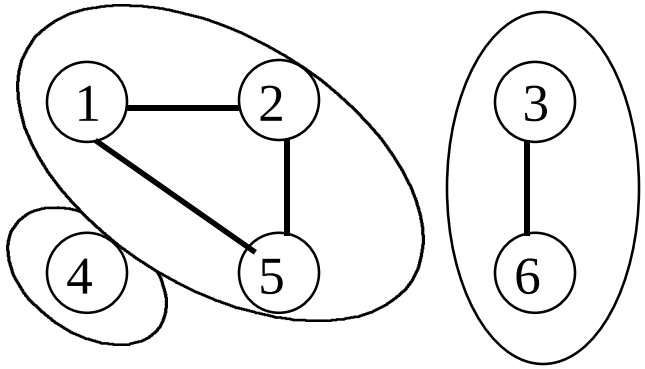
Ejemplo: subgrafo inducido por {1,2,3,6}

Grafos etiquetados

- Un grafo etiquetado es un grafo G=(V,E) sobre el que se define una función f:E–>A, donde A es un conjunto cuyas componentes se llaman etiquetas.
- Un grafo ponderado es un grafo etiquetado con números reales (A≡$\Re$)

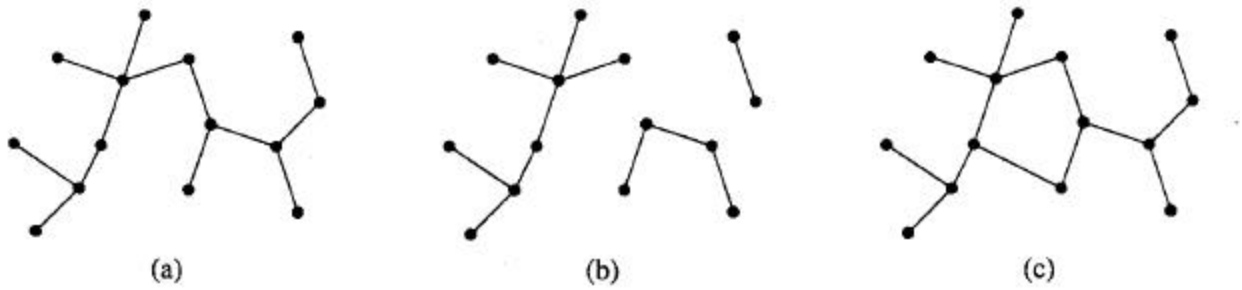
Conectividad de un grafo

- Un g.n.d. es conexo si cualquier par de vértices están conectados por un camino. Las componentes conexas de un grafo son las clases de equivalencia en V definidas por la relación "*es alcanzable desde..*"



Árboles

- Un bosque es un grafo acíclico no dirigido.
- Un grafo acíclico, no dirigido y conexo es un árbol (libre).



(a) Árbol (b) Bosque (c) Grafo

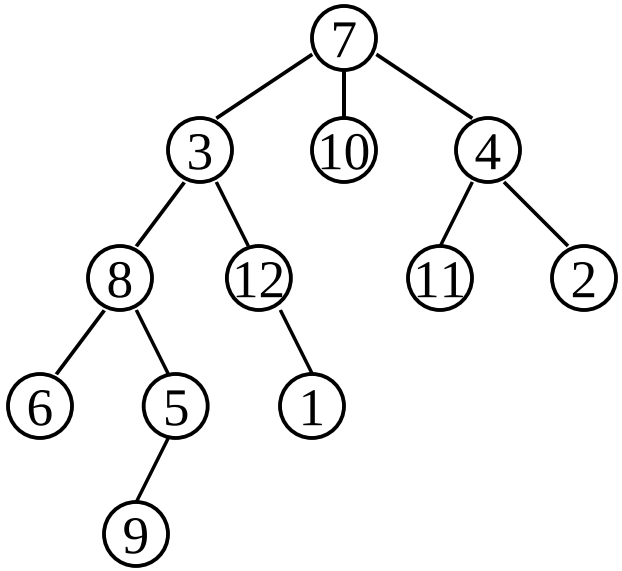
Árboles

- *Teorema*: Sea G=(V,E) un g.n.d. Las siguientes afirmaciones son equivalentes:
  - G es un árbol (libre)
  - Cualquier par de vértices en G están conectados por un único camino simple
  - G es conexo y |E|=|V|-1
  - G es acíclico pero si añadimos un arco a E, el grafo resultante contiene un ciclo

Dem. Pág. 91-93 [Cormen,90]

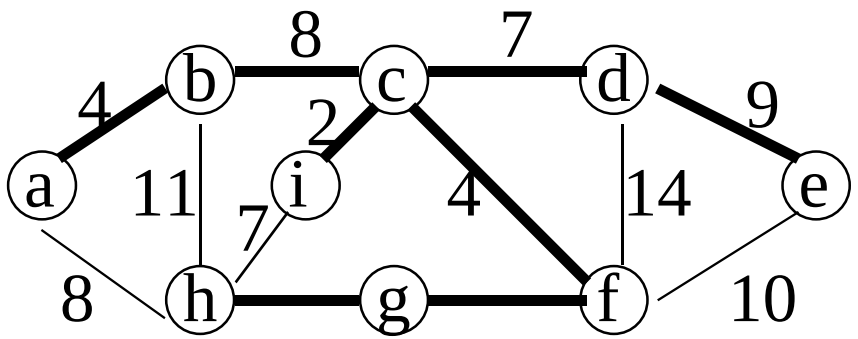
Árboles con raíz

- Un Árbol con raíz es un árbol con un vértice distinguido denominado raíz.



Árbol de recubrimiento de un gnd

- Un árbol de recubrimiento del grafo G=(V,E) es un árbol libre T=(V',E') tal que V'=V y E' $\subseteq$E
- Ejemplo:



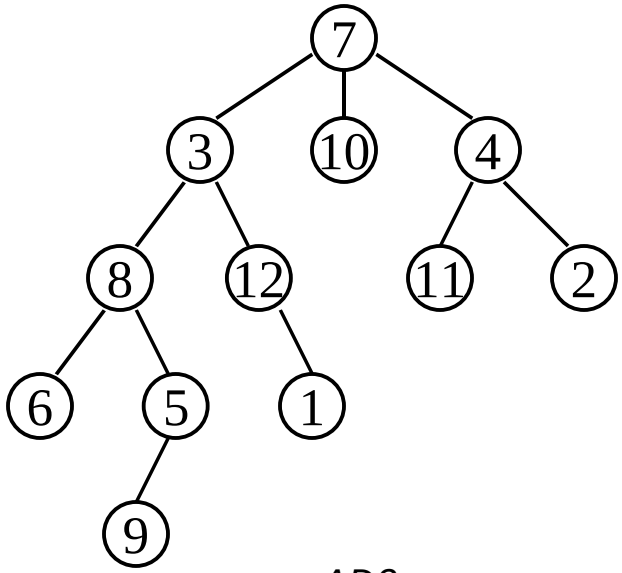
Peso total=37

Relaciones entre nodos

- Si (y,x) es el último arco en el camino desde la raíz r del árbol T hasta el nodo x,
  - y es el padre de x y
  - x es hijo de y.
- La raíz es el único nodo en T que no tiene padre.
- Si dos nodos tienen el mismo padre son hermanos

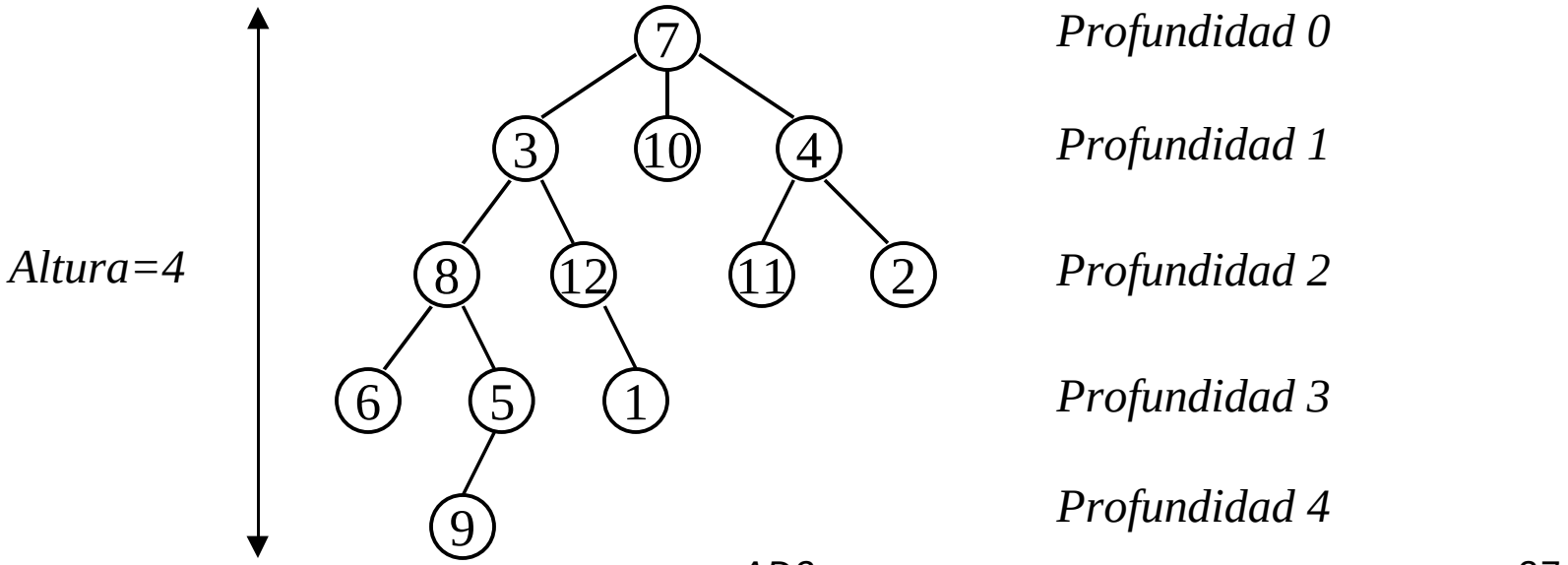
Relaciones entre nodos (cont.)

- Un nodo sin hijos lo denominaremos hoja. El resto son nodos internos.
- El grado de un nodo es el número de hijos que tiene
  (el grado de entrada de cualquier nodo del árbol es 1)



Altura de un árbol

- Se llama profundidad de un nodo a la longitud del camino desde la raíz a ese nodo.
- La altura de un árbol es la profundidad del nodo más profundo.



Árboles de Posición

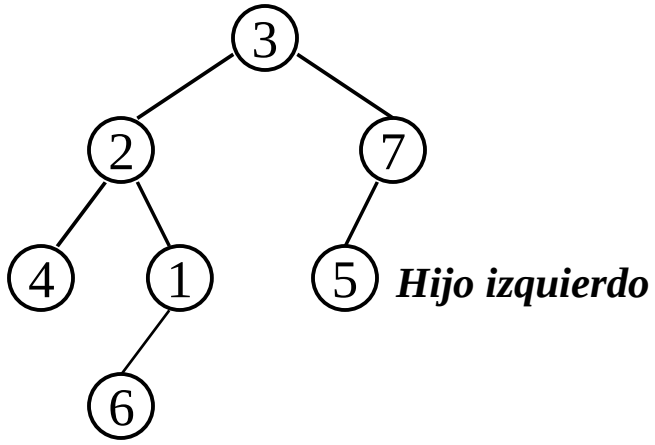
- En un árbol de posición los hijos de cada nodo están etiquetados con un entero positivo. El hijo í-ésimo de un nodo no existe si no hay ningún hijo etiquetado con i.

- Un árbol k-ario es un árbol de posición en el que todos los nodos tienen hijos etiquetados con valores menores o iguales a k.

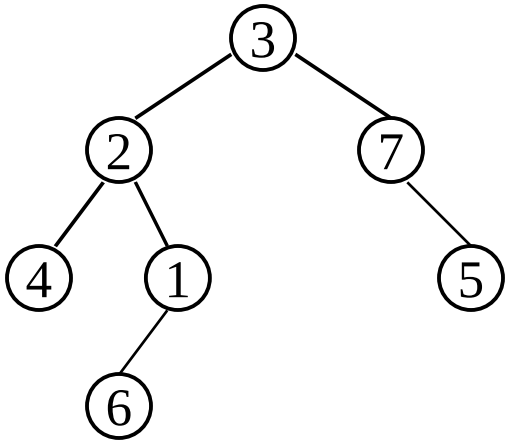
Árboles binarios

- Un árbol binario es un árbol k-ario con k=2.

- Ejemplos:



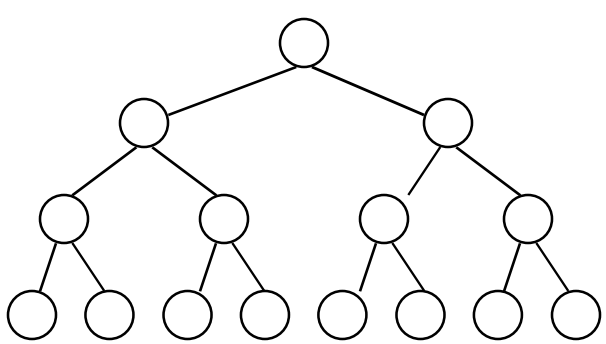
Hijo izquierdo



Hijo derecho

Árboles completos

- Un árbol k-ario se dice que es completo si todas las hojas tienen la misma profundidad y todos los nodos internos tienen grado k.



K^h hojas

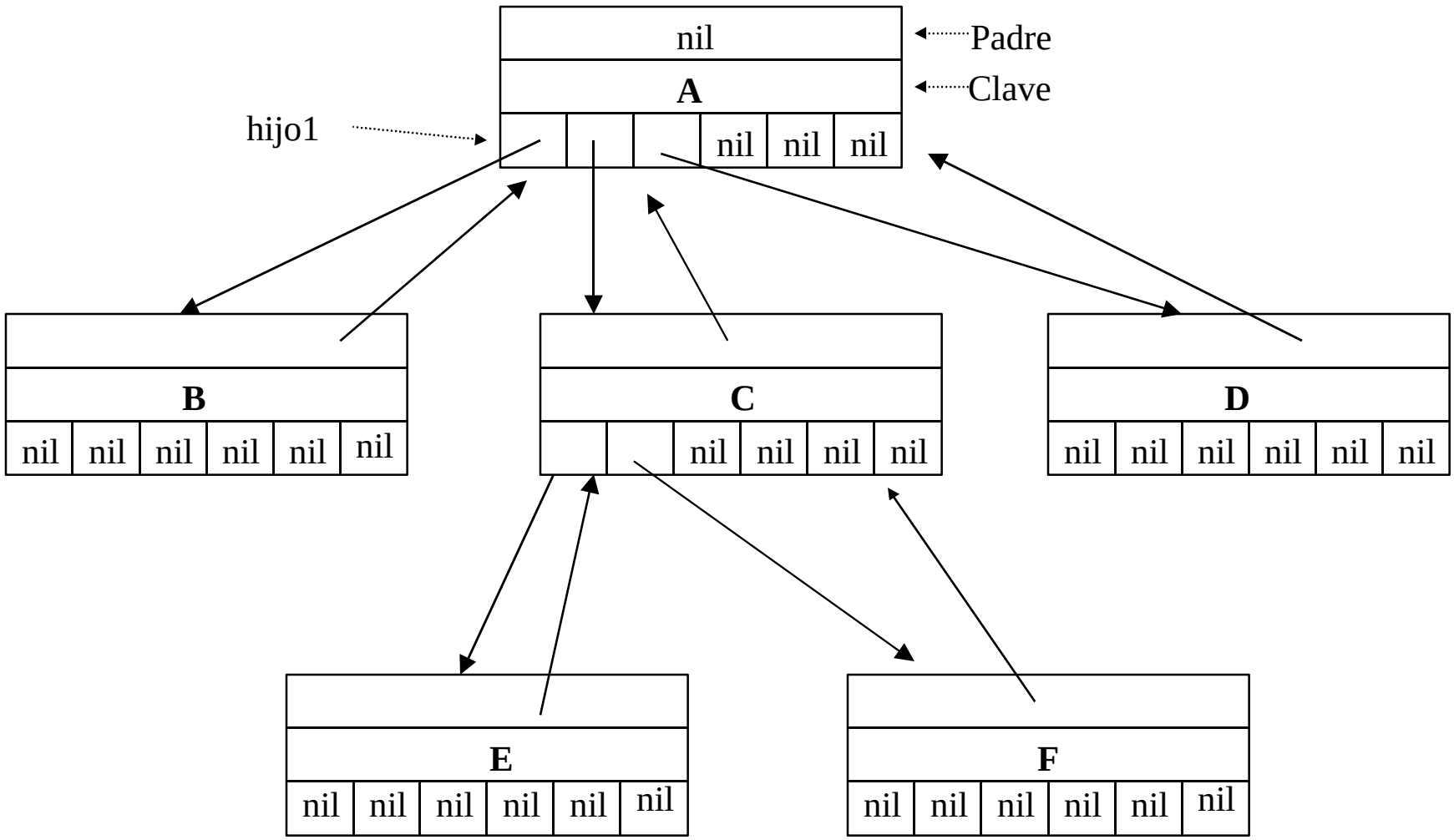
$(k^h-1)/(k-1)$ internos

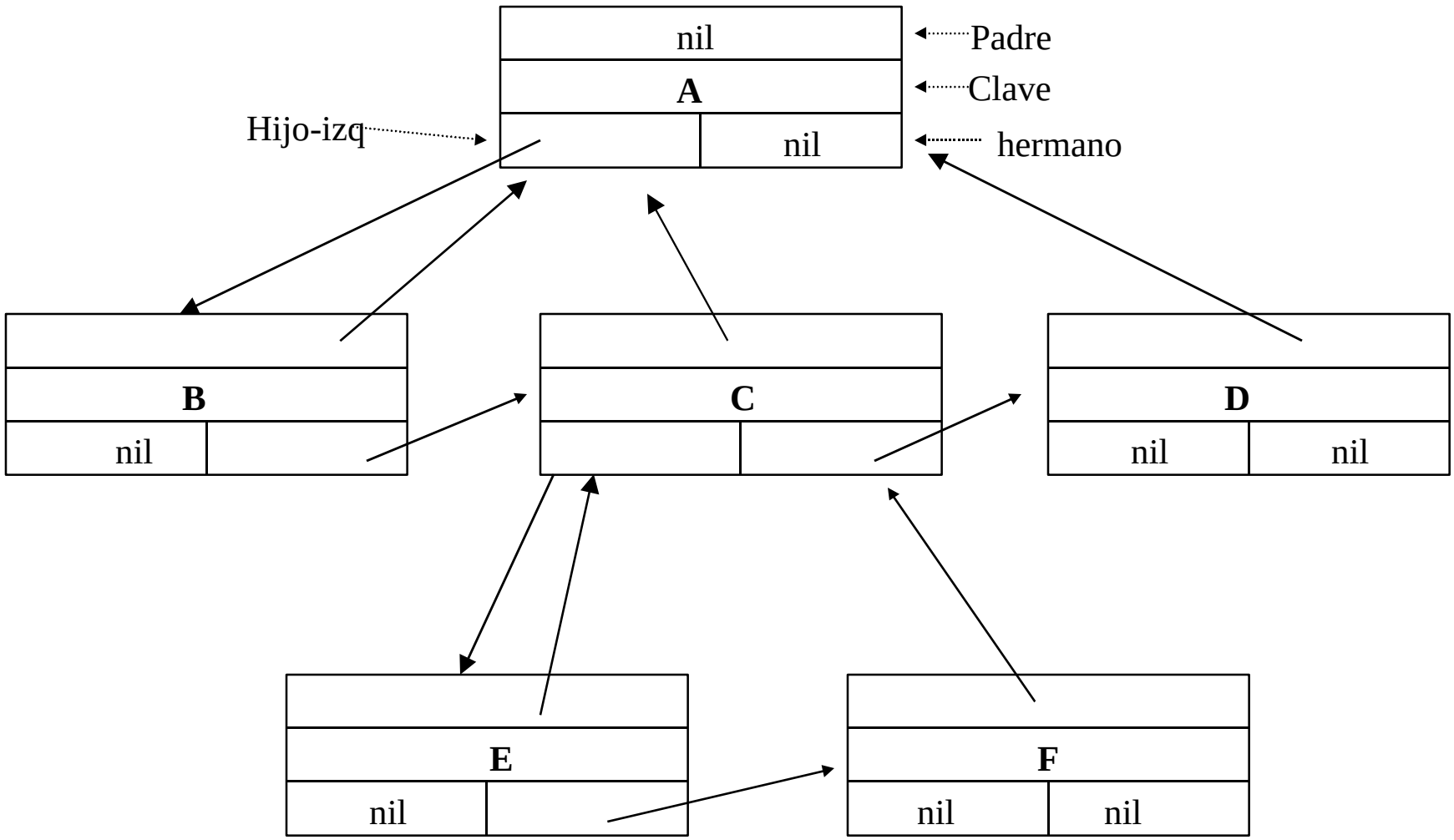
¿Cuántas hojas tiene un árbol k-ario completo de altura h?

¿Cuántos nodos internos?

2. Representación de árboles y grafos

- Árboles
  - Hijo más a la izquierda-hermano derecha
  - vector de k hijos para cada nodo
  - hijo izqdo e hijo dcho para cada nodo si k=2 (árbol binario)


nil
Padre
A
Clave
hijo1
nil nil nil
B
nil nil nil nil nil nil
C
nil nil nil nil
D
nil nil nil nil nil nil
E
nil nil nil nil nil nil
F
nil nil nil nil nil nil


nil
Padre
A
Clave
Hijo-izq
nil
hermano
B
nil
C
D
nil
nil
E
nil
F
nil
nil

REPRESENTACIÓN DE GRAFOS

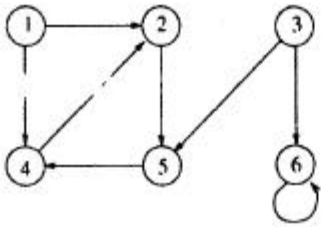
- Listas de Adyacencia: Un grafo G=(V,E) se representa como un vector de listas de vértices indexado por vértices (G: vector[V] de V). G[v] es una lista de los vértices emergentes y/o incidentes de/a v∈ V.
  - Memoria: O(|V|+|E|)
  - Tiempo de acceso: O(Grado(G))

REPRESENTACIÓN DE GRAFOS (cont.)

- Matriz de Adyacencias: Un grafo G=(V,E) se representa como una matriz G: matriz[V,V] de booleanos. La componente G[u,v] es 1 si $(u,v) \in E$, sino G[u,v]=0.
  - Memoria: $O(|V|^2)$
  - Tiempo de acceso: O(1)

REPRESENTACIÓN DE GRAFOS (cont.)

Grafo Dirigido



Lista de Ayacencia

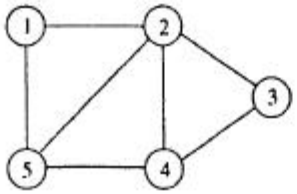
G

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

Matriz de Adyacencias

G	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

Grafo No Dirigido



Lista de Ayacencia

G

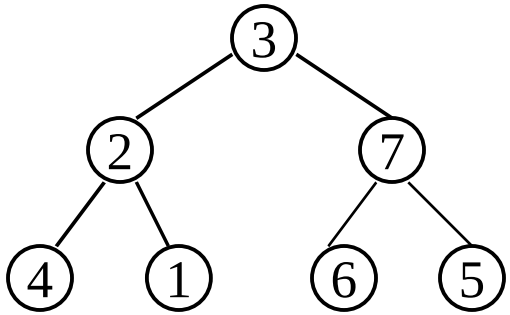
1	2	5 NIL
2	1	5 4 3 NIL
3	4	2 NIL
4	5	3 2 NIL
5	4	1 2 NIL

Matriz de Adyacencias

G	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Representación de ARBOL BINARIO

Representación contigua: Vectores

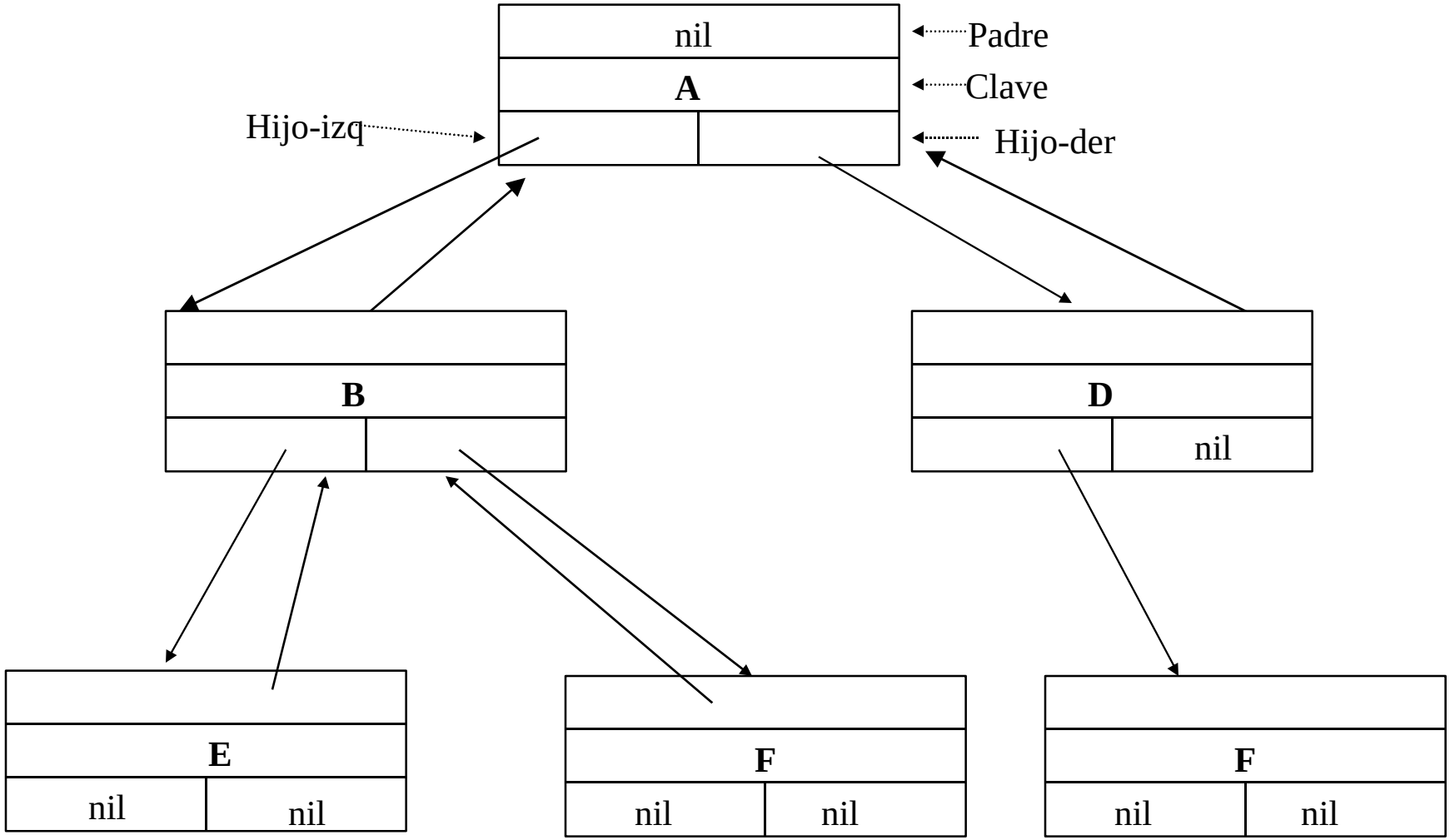


3	2	7	4	1	6	5

Hijo-izq(i)=2*i

Hijo-der(i)=2*i+1

Representación de ARBOL BINARIO



REPRESENTACIÓN ENLAZADA

```
tipo
        elemento= ...;
        nodoptr= ↑nodotipo;
        nodotipo= tupla
                                info:elemento;
                                izq,der:nodoptr;
                        ftupla
        arbol-binario=nodoptr
ftipo
```

Es, normalmente, la representación escogida (sobre todo para árboles no llenos)

```
    procedimiento creaAB (e/s a:arbol-binario)
        a:=nil
    fprocedimiento

funcion esvacioAB (a:arbol-binario) devuelve logico
        devuelve (a=nil)
ffuncion

funcion raizAB (a:arbol-binario) devuelve elemento
        devuelve a↑.info
ffuncion
funcion izqAB (a:arbol-binario) devuelve nodoptr
        devuelve a↑.izq
ffuncion


funcion derAB (a:arbol-binario) devuelve nodoptr
        devuelve a↑.der
ffuncion
```

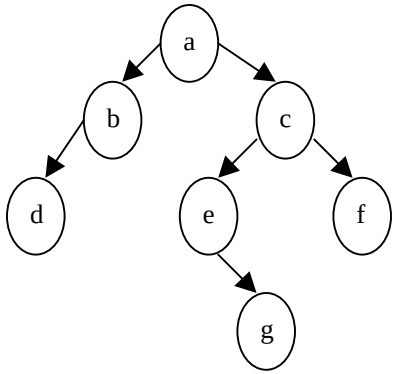
3. ALGORITMOS DE RECORRIDO DE ARBOLES BINARIOS

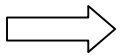
Recorrido de un árbol: visita de cada elemento del árbol 1 sóla vez.

Recorrido en amplitud (por niveles)
Recorrido en profundidad:

- Inorden
- Preorden
- Postorden

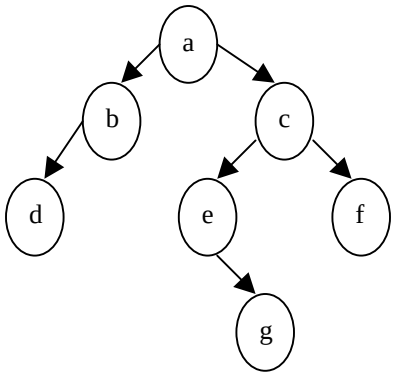
RECORRIDO EN AMPLITUD: recorrer nivel a nivel, de izquierda a derecha





a b c d e f g

RECORRIDO EN PROFUNDIDAD: recorrer “ahondando” en el árbol.



IRD= d b a e g c f

DIR= d b g e f c a

RID= a b d c e g f

PREORDEN = RID

- Visitar la raíz
- Recorrer en preorden el subárbol izquierdo
- Recorrer en preorden el subárbol derecho

```
procedimiento preorden (a:arbol-binario)
Si no esvacioAB(a) entonces
    proceso(raizAB(a));
    preorden(izqAB(a));
    preorden(derAB(a))
fsi
fprocedimiento
```

INORDEN = IRD

- Recorrer en inorden el subárbol izquierdo
- Visitar la raíz
- Recorrer en inorden el subárbol derecho

```
procedimiento inorden (a:arbol-binario)
Si no esvacioAB(a) entonces
   inorden(izqAB(a));
   proceso(raizAB(a));
   inorden(derAB(a))
fsi
fprocedimiento
```

POSTORDEN = IDR

- Recorrer en postorden el subárbol izquierdo
- Recorrer en postorden el subárbol derecho
- Visitar la raíz

```
procedimiento postorden (a:arbol-binario)
Si no esvacioAB(a) entonces
   postorden(izqAB(a));
   postorden(derAB(a));
   proceso(raizAB(a));
fsi
fprocedimiento
```

```
procedimiento postorden_it(a:arbol-binario);
var p:pila_arboles;
      psal:pila_elemento;
      aux:arbol-binario;
crear_pila_arb(p); crear_pila_ele(psal);
apilar_arb(p,a);
mientras not vacia_arb(p) hacer
    aux:=tope_arb(p); desapilar_arb(p);
    si not esvacioAB(aux) entonces
                apilar_arb(p,izqAB(aux)); apilar_arb(derAB(aux));
                apilar_ele(psal,raizAB(aux));
    fsi;
fmientras
mientras not vacia_ele(psal) hacer
        procesar(tope_ele(psal);
        desapilar_ele(psal);
fmientras;
```

```
procedimiento preorden_it(a:arbol-binario);
var p:pila_arboles;
      aux:arbol-binario;
crear_pila_arb(p); apilar_arb(p,a);
mientras not vacia_arb(p) hacer
    aux:=tope_arb(p); desapilar_arb(p);
    si not esvacioAB(aux) entonces
                apilar_arb(p,derAB(aux)); apilar_arb(izqAB(aux));
                procesar(raizAB(aux));
    fsi;
fmientras
```

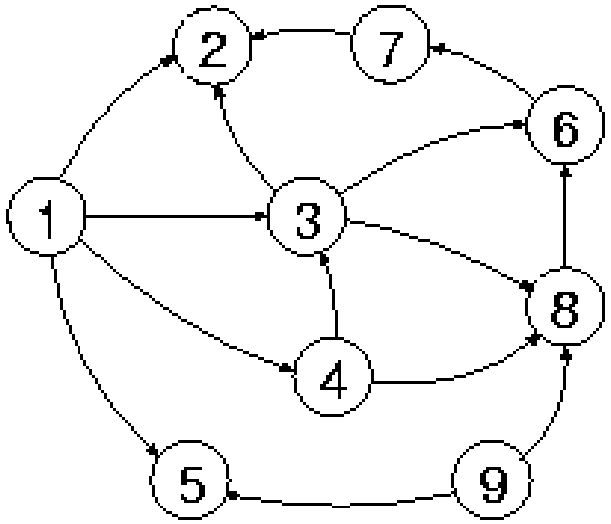
```
funcion altura(a:arbol-binario) devuelve entero;
si vacioAB(a) entonces devuelve 0
        sino si altura(izqAB(a))>altura(derAB(a) entonces devuelve altura(izqAB(a))+1
                                                        sino devuleve altura(derAB(a))+1
                fsi
fsi;
faltura

funcion altura(a:arbol-binario) devuelve entero;
var h1,h2:entero
si vacioAB(a) entonces devuelve 0
        sino h1:= altura(izqAB(a))
                h2:= altura(derAB(a))
                si h1>h2 entonces devuelve h1+1 sino devuleve h2+1   fsi
fsi;
faltura
```

```
/* los vertices están numerados  1..n */
tipo grafo_mat= vector[1..n,1..n] de {1,0}
       list=lista de enteros
       grafo_list_ady= vector[1..n] de lista;

var gmat:grafo_mat; glist:grafo_list_ady
para v:=1 hasta n hacer glist[v]:=listavacia;  fpara
para v:=1 hasta n hacer
    para w:=1 hasta n hacer
        si gmat[v,w]=1 entonces añadir(glist[v],w);
fpara
fpara
```

```
tipo vector_sal= vector [1..n] de enteros
var global R:vector_sal;
                    n: entero;
función Rec_profundidad (G:grafo) devuelve vector_sal
n:=0;
para v:=1 hasta nvertices hacer R[v]:=0  fpara; /*∀v∈ V R[v]:=0 */
para v:=1 hasta nvertices hacer
          si  R[v]:=0 entonces DFS(v) fsi
fpara
devuelve R;
fRec_profundidad

algoritmo DFS(v:entero);  /*Depth first search */
n:=n+1; R[v]:=n;
∀w∈ adyacentes(v) hacer
          si R[w]=0 entonces DFS(w) fsi
f∀
fDFS
```

Recorrido en profundidad

Grafo



Recorrido en Profundidad

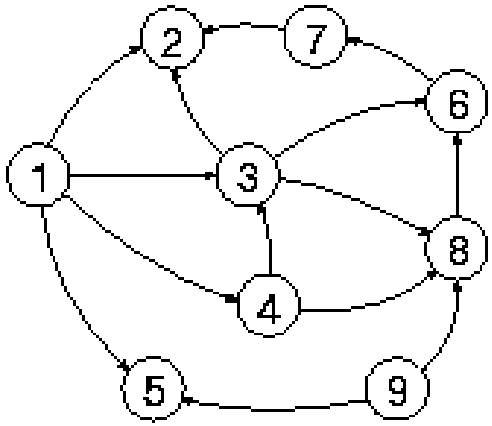
nodos	H								
v/w	1	2	3	4	5	6	7	8	9
	0	0	0	0	0	0	0	0	0
1/2,3,4,5	1	-	-	-	-	-	-	-	-
2	-	2	-	-	-	-	-	-	-
3/2,6,8	-	-	3	-	-	-	-	-	-
6/7	-	-	-	-	-	4	-	-	-
7/2	-	-	-	-	-	-	5	-	-
8/6	-	-	-	-	-	-	-	6	-
4/3,8	-	-	-	7	-	-	-	-	-
5/-	-	-	-	-	8	-	-	-	-
9/5,8	-	-	-	-	-	-	-	-	9
H	1	2	3	7	8	4	5	6	9

Orden de Visita de Nodos: 1, 2, 3, 6, 7, 8, 4, 5, 9

```
tipo vector_sal= vector [1..n] de enteros
var global R:vector_sal;
                    n: entero;
                    Q: cola de enteros
función Rec_anchura (G:grafo) devuelve vector_sal
n:=0; creaq(Q);
para v:=1 hasta nvertices hacer R[v]:=0  fpara; /*∀v∈ V R[v]:=0 */
para v:=1 hasta nvertices hacer
          si  R[v]:=0 entonces BFS(v) fsi
fpara
devuelveR;
fRec_profundidad


algoritmo BFS(v:entero);  /*Breadth first search */
n:=n+1; R[v]:=n; encolar(Q,v);
mientras not vacia(Q) hacer
u:=cabeza(Q); descabezar(Q);
∀w∈ adyacentes(u) hacer
          si R[w]=0 entonces n:=n+1; R[u]:=n; encolar(Q,w) fsi
f∀
fmientras
fDFS
```

Recorrido en anchura

Recorrido en Anchura: Ejemplo

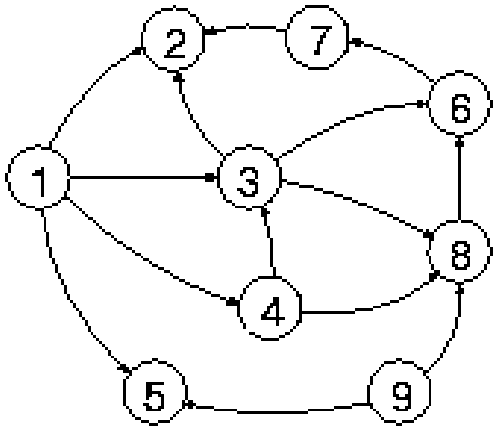


v	u	w	1	2	3	4	5	6	7	8	9	
1			1	0	0	0	0	0	0	0	0	< 1 >
	1		-	-	-	-	-	-	-	-	-	<>
		2	-	2	-	-	-	-	-	-	-	< 2 >
		3	-	-	3	-	-	-	-	-	-	< 2,3 >
		4	-	-	-	4	-	-	-	-	-	< 2,3,4 >
		5	-	-	-	-	5	-	-	-	-	< 2,3,4,5 >
	2		-	-	-	-	-	-	-	-	-	< 3,4,5 >
	3		-	-	-	-	-	-	-	-	-	< 4,5 >
		2	-	-	-	-	-	-	-	-	-	—
		6	-	-	-	-	-	6	-	-	-	< 4,5,6 >
		8	-	-	-	-	-	-	-	7	-	< 4,5,6,8 >
	4		-	-	-	-	-	-	-	-	-	< 5,6,8 >
		3	-	-	-	-	-	-	-	-	-	—
		8	-	-	-	-	-	-	-	-	-	—
	5		-	-	-	-	-	-	-	-	-	< 6,8 >
	6		-	-	-	-	-	-	-	-	-	< 8 >
		7	-	-	-	-	-	-	8	-	-	< 8,7 >
	8		-	-	-	-	-	-	-	-	-	< 7 >
		6	-	-	-	-	-	-	-	-	-	—
	7		-	-	-	-	-	-	-	-	-	<>
9			-	-	-	-	-	-	-	-	9	< 9 >
	9		-	-	-	-	-	-	-	-	-	<>
	9	5	-	-	-	-	-	-	-	-	-	—
	9	8	-	-	-	-	-	-	-	-	-	—
			1	2	3	4	5	6	8	7	9	

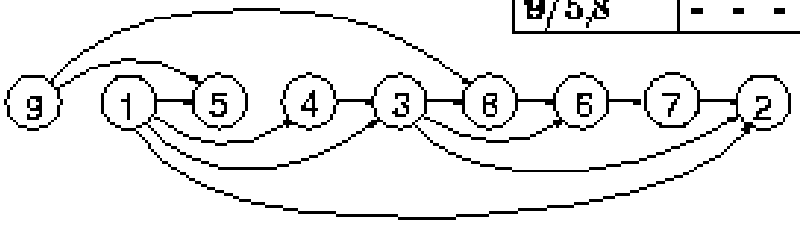
ORDEN TOPOLOGICO DE GRAFO ACICLICO

```
tipo vector_sal= vector [1..n] de enteros
var global R:vector_sal;
                n: entero;
                P:pila de enteros
función OTP (G:grafo) devuelve pila de enteros;
n:=0; crearpila(P);
para v:=1 hasta nvertices hacer R[v]:=0  fpara; /*∀v∈ V R[v]:=0 */
para v:=1 hasta nvertices hacer
        si  R[v]:=0 entonces DFS(v) fsi
fpara
devuelveP;
fRec_profundidad

algoritmo DFS(v:entero);  /*Depth first search */
n:=n+1; R[v]:=n;
∀w∈ adyacentes(v) hacer
        si R[w]=0 entonces DFS(w) fsi
f∀
apilar(P,w);
fDFS
```

Orden Topológico en Grafos Acíclicos: Ejemplo



ñodos	H									
v/w	1	2	3	4	5	6	7	8	9	P
——	0	0	0	0	0	0	0	0	0	< >
1/2,3,4,5	1	-	-	-	-	-	-	-	-	< >
2	-	2	-	-	-	-	-	-	-	< 2 >
3/2,6,8	-	-	3	-	-	-	-	-	-	< 2 >
6/7	-	-	-	-	-	4	-	-	-	< 2 >
7/2	-	-	-	-	-	-	5	-	-	< 7, 2 >
——	-	-	-	-	-	-	-	-	-	< 6, 7, 2 >
8/6	-	-	-	-	-	-	-	6	-	< 8, 6, 7, 2 >
——	-	-	-	-	-	-	-	-	-	< 3, 8, 6, 7, 2 >
4/3,8	-	-	-	7	-	-	-	-	-	< 4, 38, 6, 7, 2 >
5/–	-	-	-	-	8	-	-	-	-	< 5, 4, 3, 8, 6, 7, 2 >
——	-	-	-	-	-	-	-	-	-	< 1, 5, 4, 3, 8, 6, 7, 2 >
9/5,8	-	-	-	-	-	-	-	-	9	< 9, 1, 5, 4, 3, 8, 6, 7, 2 >



Grafo ordenado Topológicamente