



**unab**

**UNIVERSIDAD NACIONAL  
GUILLERMO BROWN**

# **CLASE 8 - Unidad 4**

## **Árboles Generales**

**ESTRUCTURAS DE DATOS (271)**

**Clase N. 8. Unidad 4.**

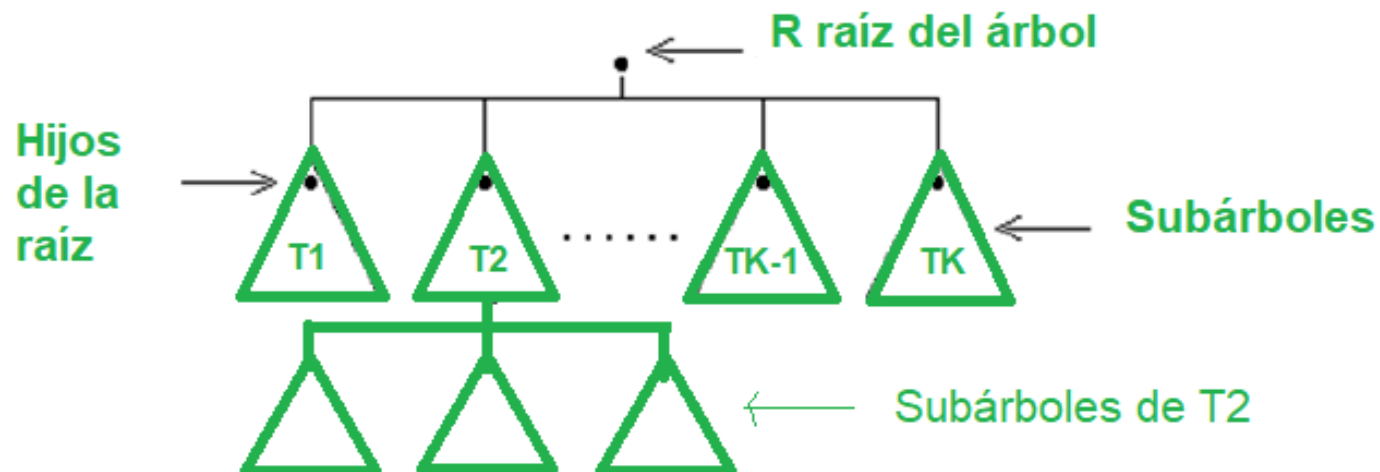
## AGENDA

- **Temario:**
  - Árboles generales, implementaciones.
  - Recorridos ordenados (InOrden, PostOrden, PreOrden). Búsquedas.
  - Actualización: inserción y borrado.
  - Análisis de la eficiencia de cada algoritmo. Aplicaciones.
- **Ejemplos en Lenguajes Python**
- **Temas relacionados y links de interés**
- **Práctica**
- **Cierre de la clase**

## Árboles Generales:

Un **árbol general** es una estructura ramificada, tal que :

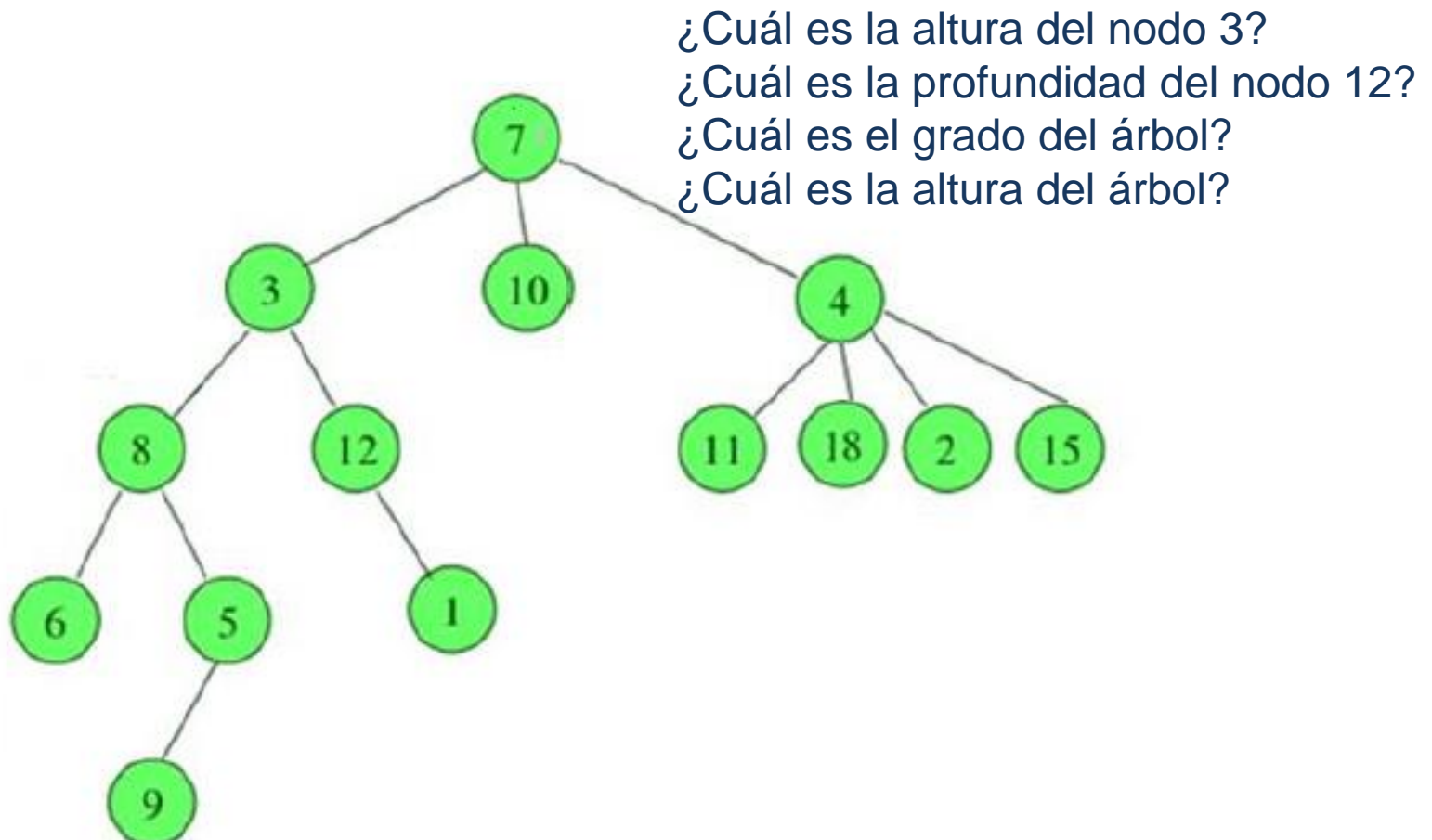
- puede estar vacío (árbol vacío)
- Puede estar formada por un nodo distinguido **R**, llamado **raíz** y un conjunto de árboles **T1, T2, ..., Tk**,  $k \geq 0$  (**subárboles**), donde la raíz de cada subárbol **Ti** está conectado a **R** por medio de una arista.



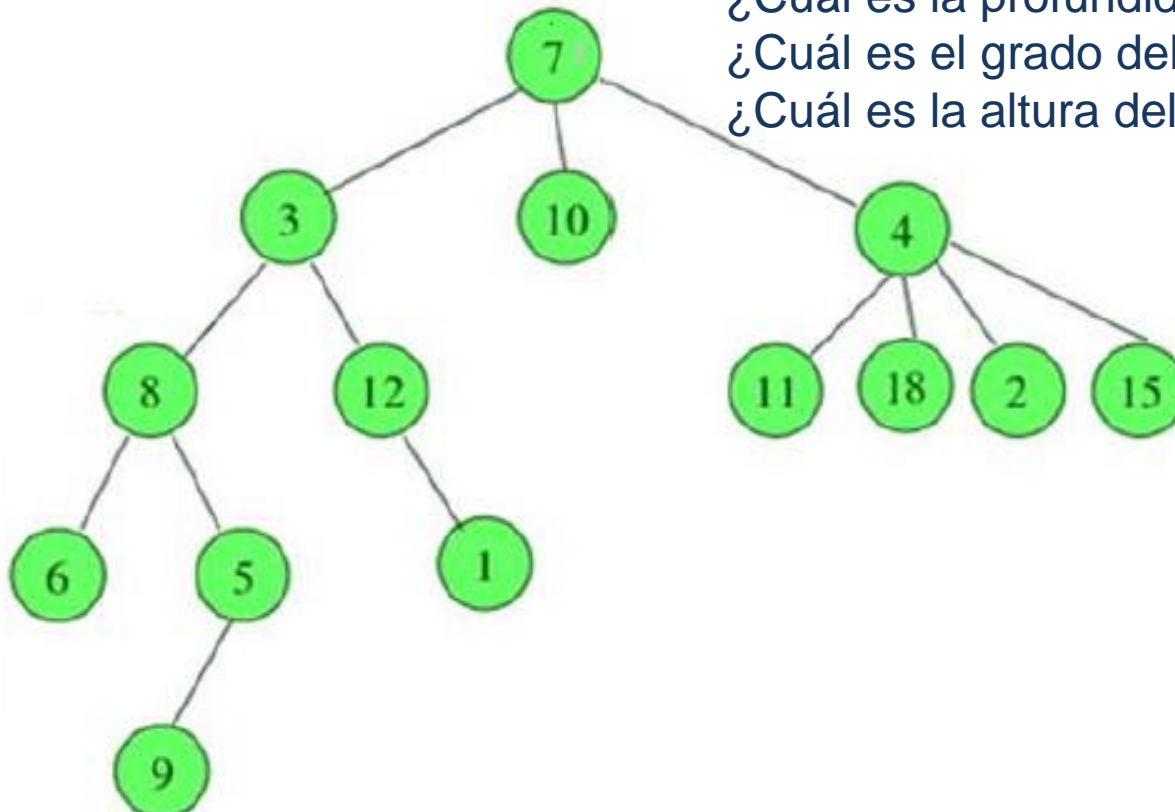
### Arboles Generales propiedades :

- **Grado de  $n_i$**  es el número de hijos del nodo  $n_i$ .
- *Grado del árbol es el grado del nodo con mayor grado*
- **Altura de  $n_i$**  es la longitud del camino más largo desde  $n_i$  hasta una hoja.
- *La altura de un árbol es la altura del nodo raíz.*
- *Las hojas tienen altura cero.*
- **Profundidad/Nivel de  $n_i$** : es la longitud del único camino desde la raíz hasta  $n_i$ .
- *La raíz tiene profundidad o nivel cero.*

## Arboles Generales ejercitación :



## Arboles Generales ejercitación :



¿Cuál es la altura del nodo 3? **3**

¿Cuál es la profundidad del nodo 12? **2**

¿Cuál es el grado del árbol? **3**

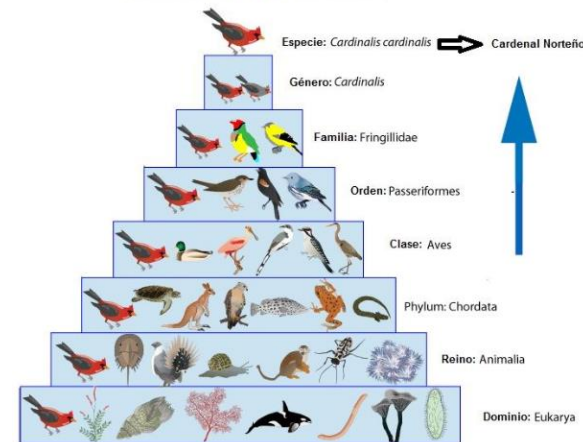
¿Cuál es la altura del árbol? **4**

## Arboles Generales ejemplos :

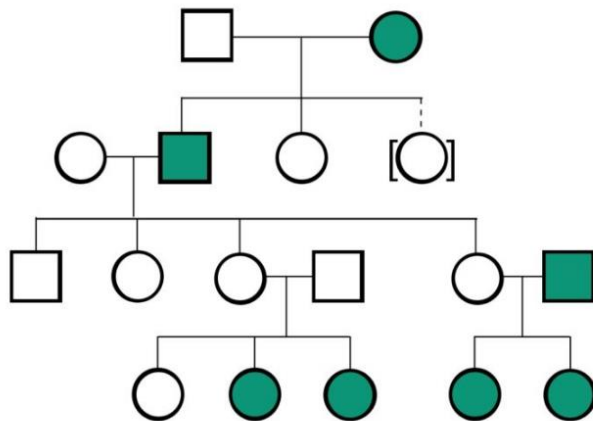
- Organigrama de una empresa
- Árboles genealógicos
- Taxonomía que clasifica organismos
- Sistemas de archivos
- Organización de un libro por capítulos y secciones



### TAXONOMIA EJEMPLO




**Taxonomía = Clasificar a los Seres Vivos**




I  
II  
III  
IV

### Representación:

- Lista de hijos:

Cada Nodo tiene  información propia del Nodo  
lista de todos sus hijos

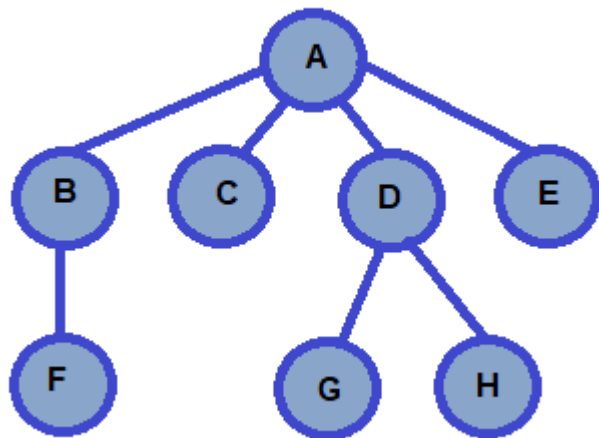
- Hijo mas izquierdo y hermano derecho

Cada Nodo tiene  información propia del Nodo  
referencia al hijo mas izquierdo  
referencia al hermano derecho



## Implementación:

- Lista de hijos implementada con listas enlazadas:



arbol\_general  
(ag)

Arbol\_General instance

hijos	•
info	"A"
sig	None

Arbol\_General instance

hijos	•
info	"B"
sig	•

Arbol\_General instance

hijos	None
info	"C"
sig	•

Arbol\_General instance

hijos	None
info	"D"
sig	•

Arbol\_General instance

hijos	None
info	"E"
sig	None

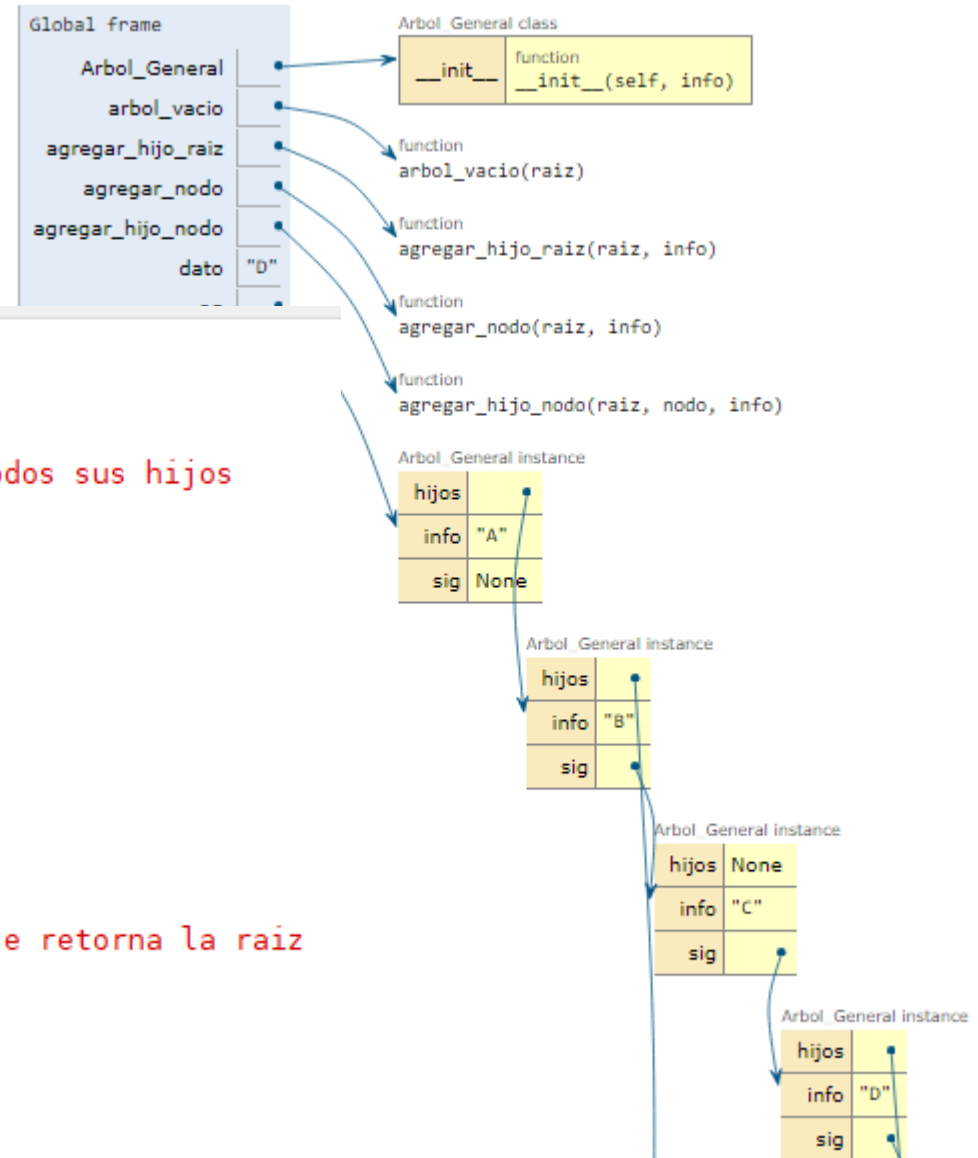
Arbol\_General instance

hijos	None
info	"F"
sig	None

## Implementación:

- Listas enlazadas

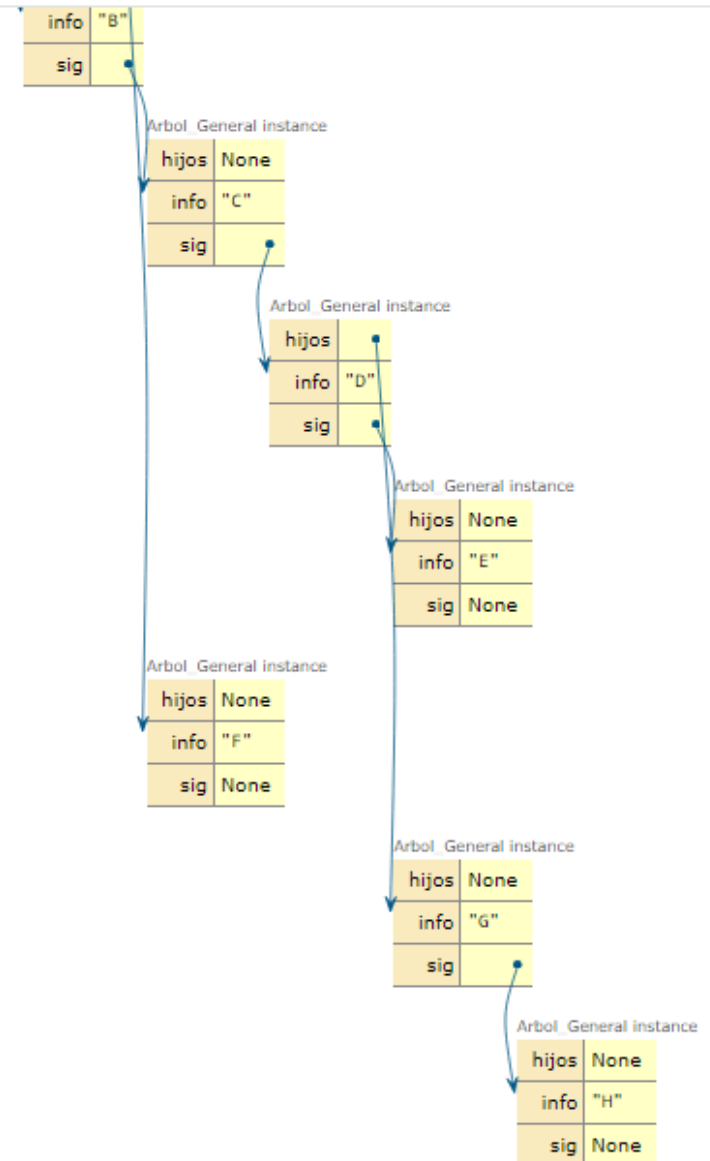
```
class Arbol_General(object):  
    #clase nodo arbol  
    def __init__(self, info):  
        self.info = info  
        self.sig = None #lista donde guarda todos sus hijos  
        self.hijos = None #sin hijos  
  
def arbol_vacio(raiz):  
    #retorna si el arbol esta vacio  
    return raiz == None  
  
def agregar_hijo_raiz(raiz, info):  
    #inserto el dato en el arbol  
    if arbol_vacio(raiz):  
        raiz = Arbol_General(info)  
        #referencio el comienzo de la lista  
    else:  
        raiz = agregar_nodo(raiz, info)  
    return raiz #una vez actualizado el dato se retorna la raiz
```



## Implementación:

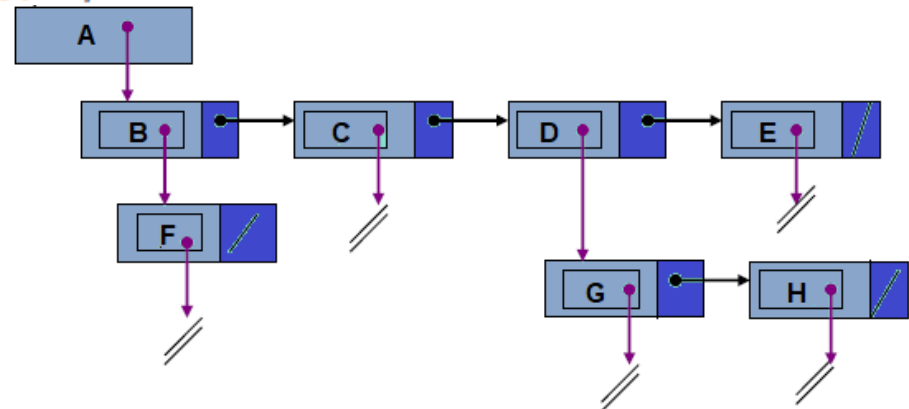
```
def agregar_nodo(raiz, info):
    nodo = Arbol_General(info)
    if (raiz.hijos is None):
        #no tiene hijos
        raiz.hijos = nodo
    else:
        anterior = raiz.hijos
        actual = raiz.hijos.sig
        while (actual is not None):
            anterior = anterior.sig #anterior = actual
            actual = actual.sig
        anterior.sig = nodo
        nodo.sig = actual
    return raiz

def agregar_hijo_nodo(raiz, nodo, info):
    if not arbol_vacio(raiz):
        anterior = raiz.hijos
        actual = raiz.hijos.sig
        ok = False
        while (actual is not None) and (not ok):
            if anterior.info == nodo:
                ok = True
            else:
                anterior = anterior.sig #anterior = actual
                actual = actual.sig
        if ok:
            nodo = agregar_nodo(anterior, info)
            anterior.sig = nodo
            nodo.sig = actual
    return raiz
```



## Implementación:

```
dato = input("Ingrese dato: ")
ag = agregar_hijo_raiz(None,dato)#raiz
for i in range(4):
    dato = input("Ingrese dato: ")
    ag = agregar_hijo_raiz(ag,dato)
dato = input("Ingrese dato del nodo Padre: ")
dato1 = input("Ingrese dato del nodo hijo: ")
ag = agregar_hijo_nodo(ag,dato,dato1)
dato = input("Ingrese dato del nodo Padre: ")
dato1 = input("Ingrese dato del nodo hijo: ")
ag = agregar_hijo_nodo(ag,dato,dato1)
dato = input("Ingrese dato del nodo Padre: ")
dato1 = input("Ingrese dato del nodo hijo: ")
ag = agregar_hijo_nodo(ag,dato,dato1)
```



### Operaciones:

- 1, **preorden(raíz)**. Se procesa primero la raíz y luego los hijos
2. **inorden(raíz)**. Se procesa el primer hijo, luego la raíz y por ultimo los hijos restantes
3. **postorden(raíz)**. Se procesa primero los hijos y luego la raíz
4. **por\_nivel(raíz)**. Se procesan los nodos teniendo en cuenta sus niveles, primero la raíz, luego los hijos,, los hijos de éstos, etc.

## Operaciones:

```
def preorden(raiz):
    if(raiz is not None):
        print(raiz.info)
        hijos = raiz.hijos
        while(hijos is not None):
            preorden(hijos)
            hijos = hijos.sig
def postorden(raiz):
    if(raiz is not None):
        hijos = raiz.hijos
        while(hijos is not None):
            postorden(hijos)
            hijos = hijos.sig
        print(raiz.info)
def por_niveles(raiz):
    #encolar(raiz)
    #mientras la cola no este vacia
    #desencolamos v
    #imprimimos dato de v
    #para cada hijo de v
    #encolar (hijo)
```

## #INVOCACIONES

```
dato = input("Ingrese dato: ")
ag = agregar_hijo_raiz(None,dato)#raiz
for i in range(4):
    dato = input("Ingrese dato: ")
    ag = agregar_hijo_raiz(ag,dato)
dato = input("Ingrese dato del nodo Padre: ")
dato1 = input("Ingrese dato del nodo hijo: ")
ag = agregar_hijo_nodo(ag,dato,dato1)
dato = input("Ingrese dato del nodo Padre: ")
dato1 = input("Ingrese dato del nodo hijo: ")
ag = agregar_hijo_nodo(ag,dato,dato1)
dato = input("Ingrese dato del nodo Padre: ")
dato1 = input("Ingrese dato del nodo hijo: ")
ag = agregar_hijo_nodo(ag,dato,dato1)
preorden(ag)
postorden(ag)
por_niveles(ag)
```