Arboles

Introducción a la programación II

Arboles

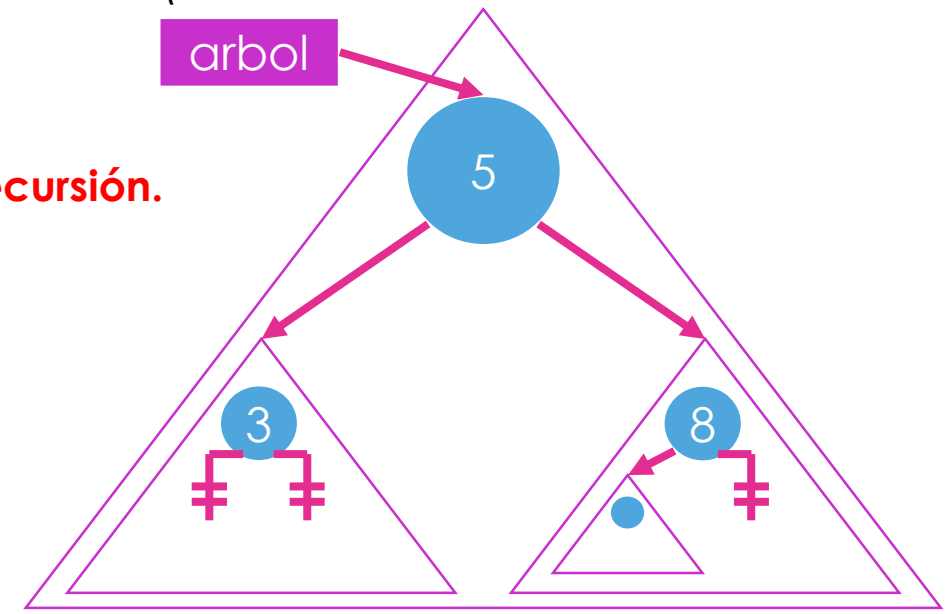
Si bien las listas son flexibles para muchos problemas, hay situaciones donde se requiere recorridos más eficientes. Por ejemplo la búsqueda de un elemento.

Los árboles permiten representar enlaces jerárquicos, que cuando mantienen un criterio de orden, permiten simplificar o hacer más eficientes los recorridos de búsqueda.

Cada árbol puede representarse por un nodo y subárboles (solo se verán arboles binarios, dos subárboles por nodo).
El nodo de mayor jerarquía se denomina **raíz**.

Los recorridos en árbol solo se pueden hacer con recursión.

```
type      punteronodo=^nodo;
          nodo=record
                    campo1;
                    ...
                    campoN;
                    izq, der:punteronodo;
          end;
```



Arboles

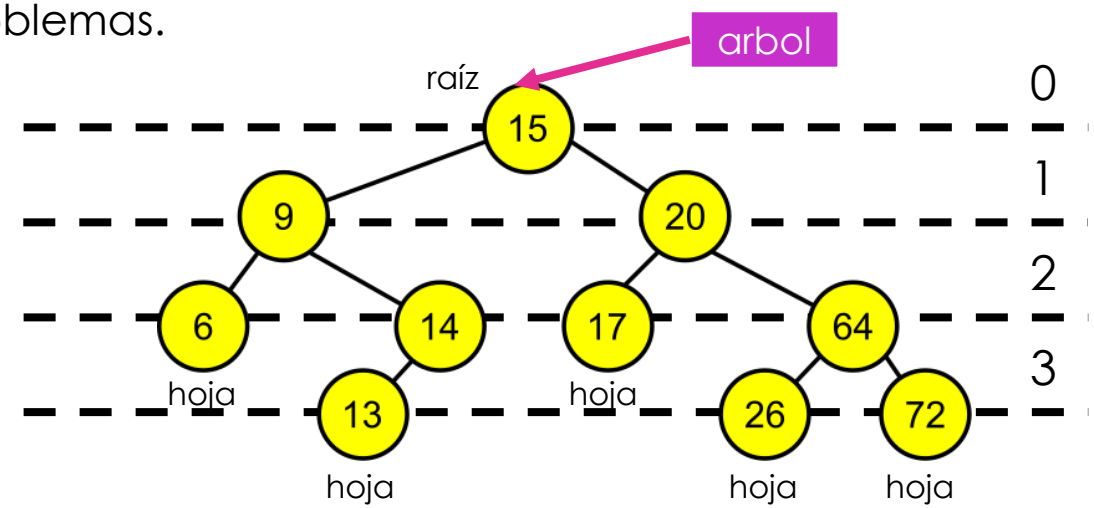
A un nodo que no tiene hijos se le conoce como **hoja**.

El **nivel** de un nodo en un árbol binario se define del modo siguiente:

- La raíz del árbol tiene el nivel 0.
- El nivel de cualquier otro nodo en el árbol es uno más que el nivel de su padre.

La profundidad o **altura** de un árbol binario es el máximo nivel de cualquier hoja en el árbol.

Los árboles ordenados son los más utilizados. Los arboles desordenados suelen usarse como información para resolver problemas.

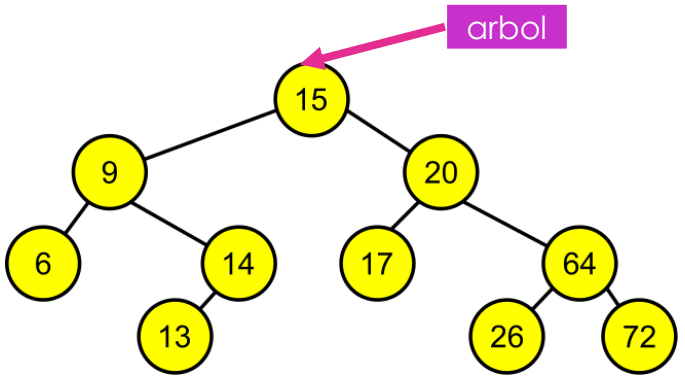


Recorridos de árboles binarios

```
procedure preorden(arbol:parbol);
begin
if (arbol <> NIL) then begin
                tratar(arbol);
                preorden(arbol^.izq);
                preorden(arbol^.der);
end;
end;
```

```
procedure inorden(arbol:parbol);
begin
if (arbol <> NIL) then begin
                inorden(arbol^.izq);
                tratar(arbol);
                inorden(arbol^.der);
end;
end;
```

Orden en que se resuelven
15, 9, 6, 14, 13, 20, 17, 64, 26, 72
6, 13, 14, 9, 17, 26, 72, 64, 20, 15
6, 9, 13, 14, 15, 17, 20, 26, 64, 72




```
procedure postorden(arbol:parbol);
begin
if (arbol <> NIL) then begin
                postorden(arbol^.izq);
                postorden(arbol^.der);
                tratar(arbol);
end;
end;
```

Observación: el orden ↕ en que se invoca **arbol^.izq** y **arbol^.der** modifica el resultado.

Arboles binarios

Insertar nodo en un árbol ordenado de forma ascendente:

```
Procedure insertar_nodo_árbol(var árbol:parbol; nodo:parbol);
Begin
        if árbol=NIL then //CORTE
                árbol:=nodo
        else  if árbol^.dato>nodo^.dato then
                insertar_nodo_árbol(árbol^.izq, nodo) //tiene orden ascendente de izq a der
        else
                insertar_nodo_árbol(árbol^.der, nodo)
End;
```

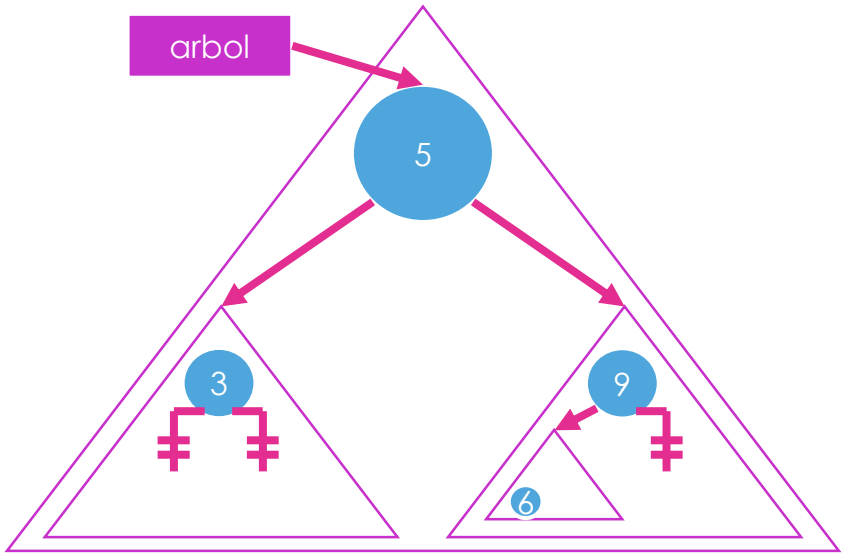
Imprimir un árbol ordenado de forma ascendente **en orden inverso** (de forma descendente):

```
Procedure imprimir_árbol_inv(árbol:parbol);
Begin
        //árbol = NIL es el CORTE, no se coloca porque no haría nada en ese caso
        if árbol<>NIL then begin
                imprimir_árbol_inv(árbol^.der);//VOY A DERECHA EN EL AVANCE DE RECUSIÓN
                write(árbol^.dato);
                imprimir_árbol_inv(árbol^.izq);//VOY A IZQUIERDA EN REGRESO DE RECURSIÓN
        end;
End;
```

Arboles binarios

Podar un árbol desde un nodo padre es eliminar todo el árbol desde dicho nodo (si el nodo es el raíz se eliminará el árbol completo).
Para ello avanza hasta las hojas y empieza a borrar cada nodo de forma ascendente sin importar el orden.

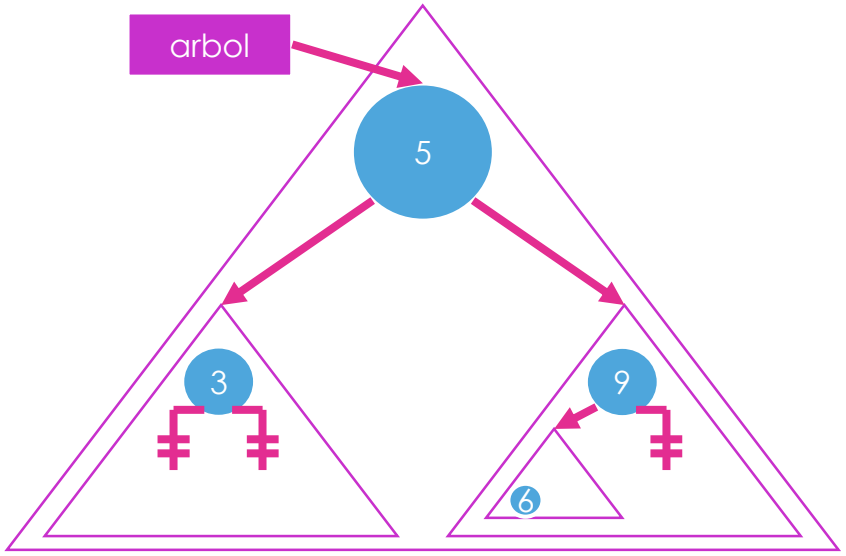
```
Procedure podar_árbol(var árbol:parbol);
Begin
        if árbol<>NIL then begin
                //avanzo hacia las hojas por izq y der, o der e izq
                podar_árbol(árbol^.izq);
                podar_árbol(árbol^.der);
                //cuando llego a una hoja la elimino
                dispose(arbol);
                arbol:=nil;
        end;
End;
```



Arboles binarios

Borrar un nodo de un arbol:

- El nodo no se encuentra.
- El nodo del arbol existe, opciones:
  - No tiene hijos subarboles (Ejemplo nodo con **3**, se elimina el nodo).
  - Tiene un hijo subarbol (Ejemplo nodo con **9**, se reemplaza el nodo por su hijo y se eliminar el nodo).
  - Tiene dos subarboles hijos (se reemplaza por el nodo más izquierdo del subarbol derecho o el más derecho del zubarbol izquierdo; por ejemplo nodo con **5**, se reemplaza por el nodo con **6** y se elimina).



Clases de listas y arboles descriptos

Para cada clase de lista y arbol mencionado se deberá tener en cuenta los siguientes métodos (procedimientos y funciones).

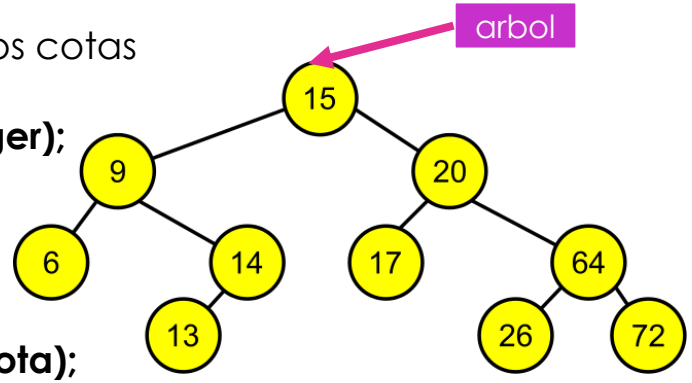
	Búsqueda, recorrido, etc.	Agregar nodo	Insertar nodo	Eliminar nodo	Eliminar completo	Ordenar
Lista simple	X	X		X	X	X
Lista simple ordenada	X		X	X	X	
Lista circular	X	X		X	X	X
Lista circular ordenada	X		X	X	X	
Lista doblemente vinculada	X	X		X	X	X
Listas doblemente vinculada ordenada	X		X	X	X	
Arbol simple	X				X	
Arbol ordenado	X		X	X	X	

Arboles binarios

Recorridos por rango: mayor a una cota, menor a una cota, entre dos cotas

```
Procedure recorrer_menor_a_cota_preorden(árbol:parbol; cota:integer);
Begin
if árbol<>NIL then
        if árbol^.dato<cota then begin
                Tratar(arbol);
                recorrer_menor_a_cota_preorden(árbol^.der, cota);
        end;
        recorrer_menor_a_cota_preorden(árbol^.izq, cota);
End;

Procedure recorrer_menor_a_cota_inorden(árbol:parbol; cota:integer);
Begin
if árbol<>NIL then
        if árbol^.dato<cota then begin
                recorrer_menor_a_cota_inorden(árbol^.der, cota);
                Tratar(arbol);
        end;
        recorrer_menor_a_cota_inorden(árbol^.izq, cota);
End;
```

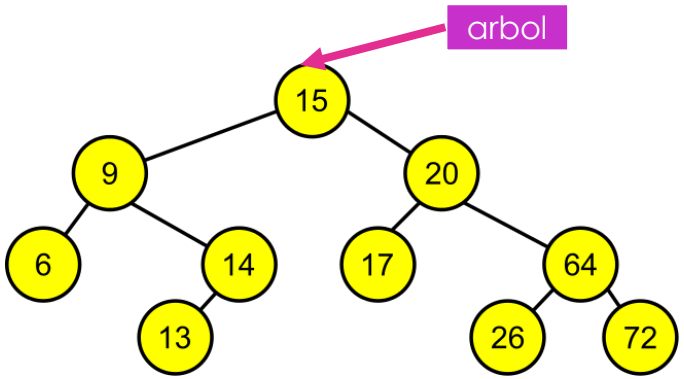


Cota = 20
15, 17, 9, 14, 13, 6
17, 15, 14, 13, 9, 6

Arboles binarios

Recorridos por rango: mayor a una cota, menor a una cota, entre dos cotas

```
Procedure recorrer_menor_a_cota_inorden(árbol:parbol; cota:integer);
Begin
if árbol<>NIL then
        recorrer_menor_a_cota_inorden(árbol^.izq, cota);
        if árbol^.dato<cota then begin
                Tratar(arbol);
                recorrer_menor_a_cota_inorden(árbol^.der, cota);
        end;
End;
```



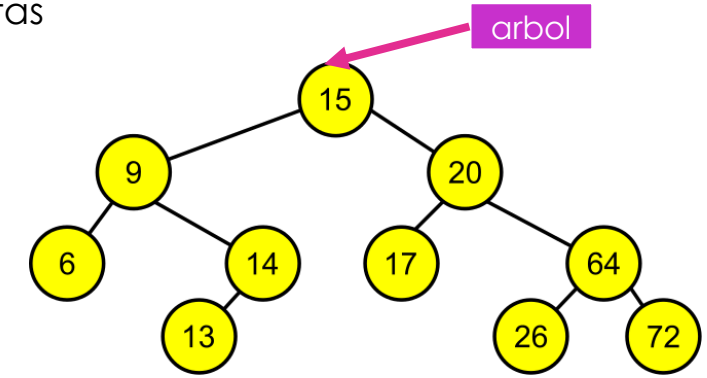
Cota = 20

6, 9, 13, 14, 15, 17

Arboles binarios

Recorridos por rango: mayor a una cota, menor a una cota, entre dos cotas

```
Procedure recorrer_entre_cotas(árbol:parbol; cotamin,cotamax:integer);
Begin
if árbol<>NIL then
        if árbol^.dato>cotamin then begin
                recorrer_entre_cotas(árbol^.izq, cotamin,cotamax);
                if árbol^.dato<cotamax then begin
                        Tratar(arbol);
                        recorrer_entre_cotas(árbol^.der, cotamin,cotamax);
                end;
        end
        else
        if árbol^.dato<cotamax then
                recorrer_entre_cotas(árbol^.der, cotamin,cotamax);
End;
```



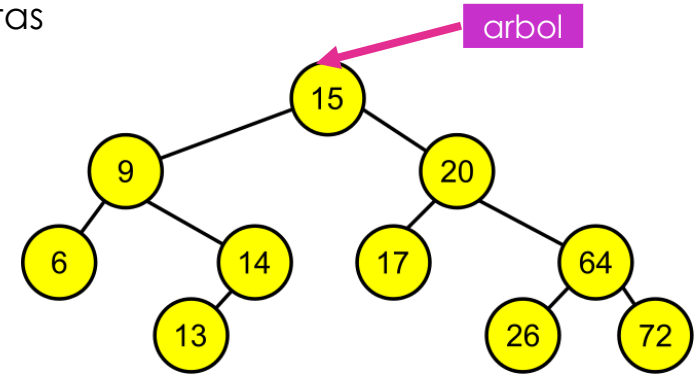
Cotamin = 12, Cotamax = 30

13, 14, 15, 17, 20, 26

Arboles binarios

Recorridos por rango: mayor a una cota, menor a una cota, entre dos cotas

```
Procedure recorrer_entre_cotas(árbol:parbol; cotamin,cotamax:integer);
Begin
if árbol<>NIL then
        if árbol^.dato<cotamax then begin
                recorrer_entre_cotas(árbol^.der, cotamin,cotamax);
                if árbol^.dato>cotamin then begin
                        Tratar(arbol);
                        recorrer_entre_cotas(árbol^.izq, cotamin,cotamax);
                end;
        end
        else
        if árbol^.dato>cotamin then
                recorrer_entre_cotas(árbol^.izq, cotamin,cotamax);
End;
```



Cotamin = 12, Cotamax = 30

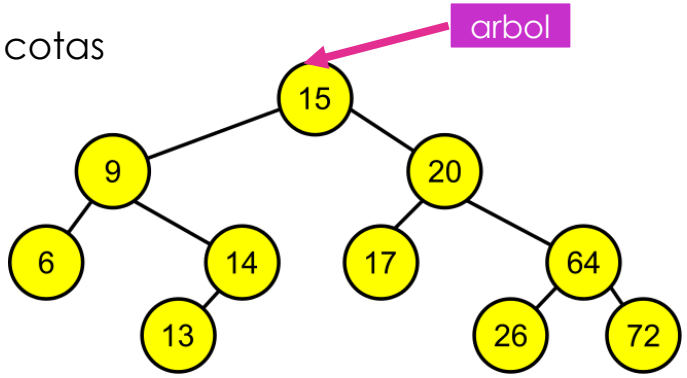
13, 14, 15, 17, 20, 26

Arboles binarios

Recorridos por nivel: mayor a una cota, menor a una cota, entre dos cotas

```
Procedure recorrer_menor_nivel(árbol:parbol; nivel:integer);
Begin
if árbol<>NIL then
        if 0<nivel then begin
                recorrer_menor_nivel(árbol^.der, nivel-1);
                Tratar(arbol);
                recorrer_menor_nivel(árbol^.izq, nivel-1);
        end;
End;

Procedure recorrer_mayor_nivel(árbol:parbol; nivel:integer);
Begin
if árbol<>NIL then begin
        if 0>nivel then
                Tratar(arbol);
        recorrer_mayor_nivel(árbol^.der, nivel-1);
        recorrer_mayor_nivel(árbol^.izq, nivel-1);
End;
End;
```



Cota = 2
20, 15, 9
72, 26, 13