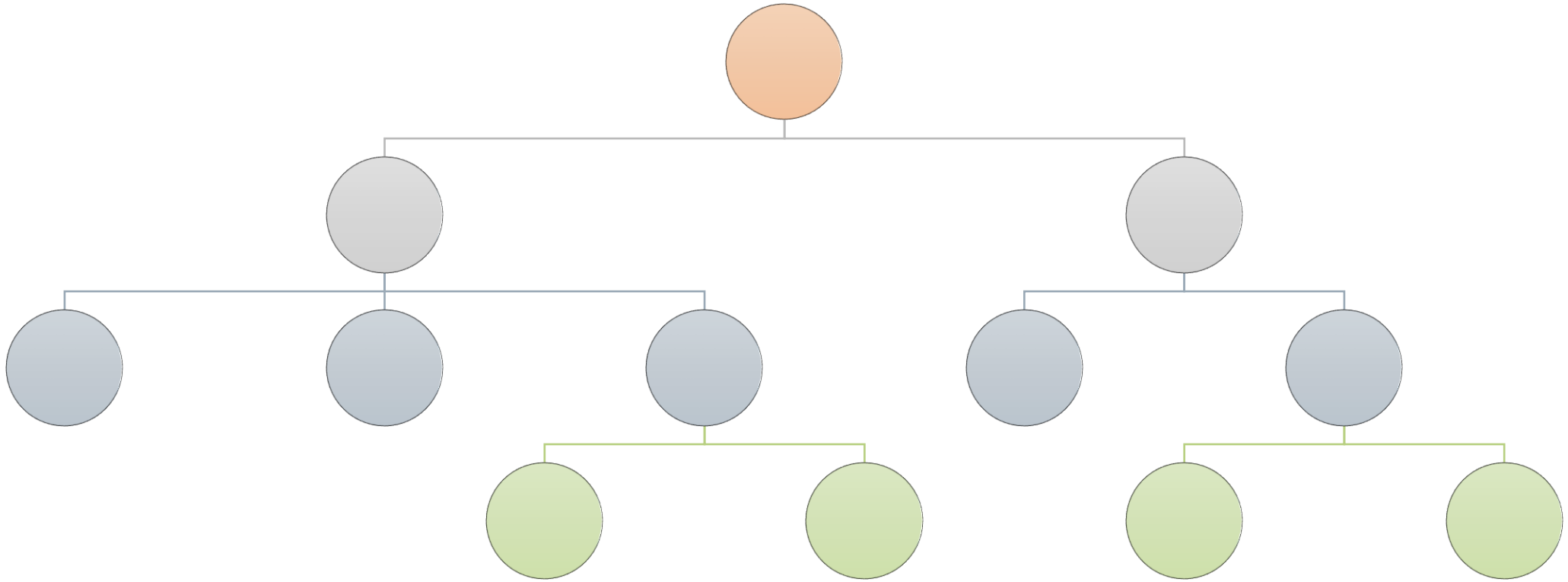
Árboles

recorrido, búsqueda, actualización – tipos

Árbol



Árboles

- Cuando crece la cantidad de nodos, en estructuras lineales como las **listas**, **el tiempo de acceso a un determinado espacio se hace ineficiente**.

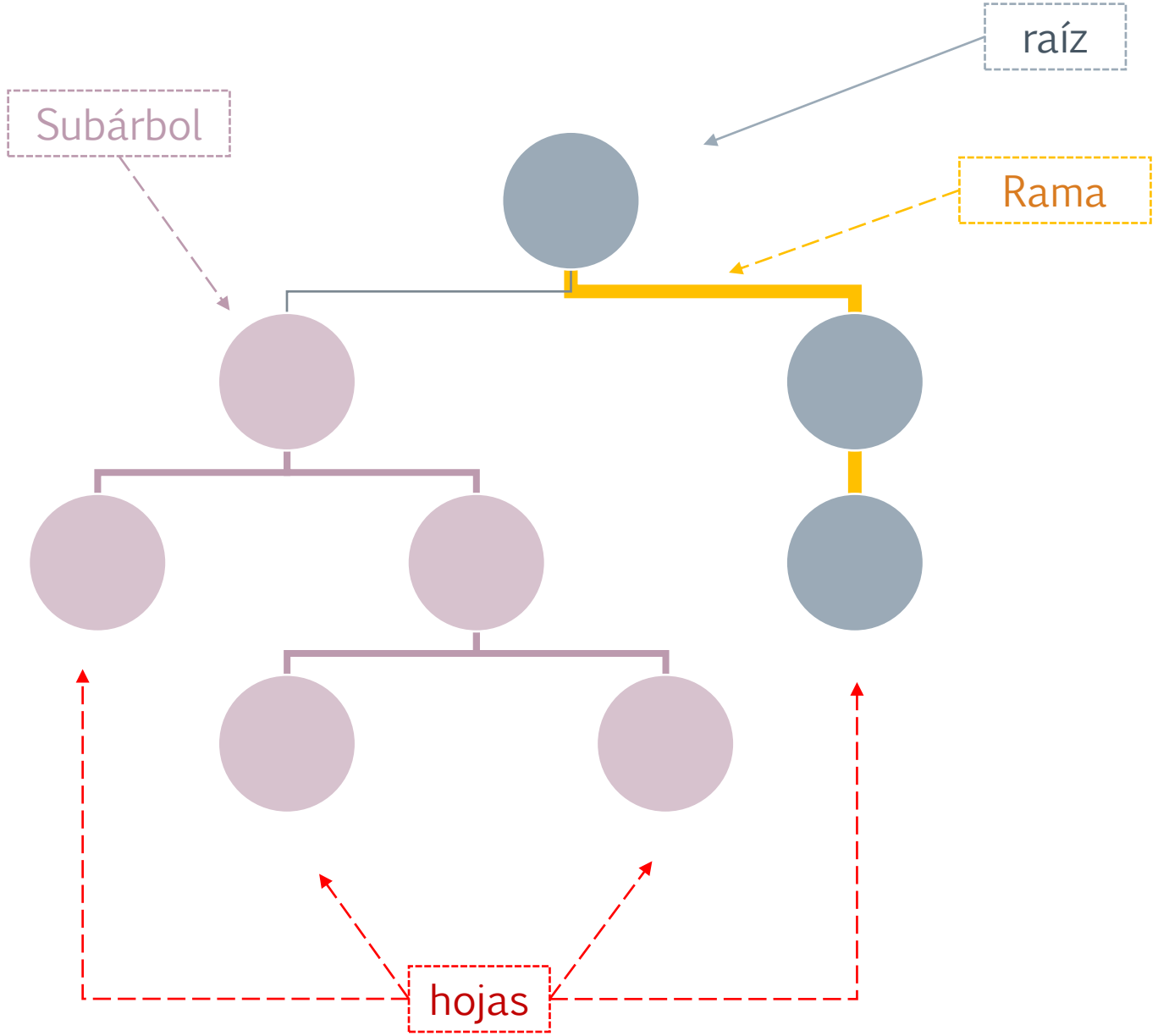
- En este capítulo veremos un tipo de estructura, **no lineal**, que nos permitirá obtener en promedio, para las distintas operaciones, un tiempo de ejecución $O(\log_2 n)$

Árboles - características

- Son estructuras de datos no lineales y jerárquicas.
- Elementos homogéneos (nodos).
- Cada elemento se relaciona con cero o más nodos (hijos).
- Sus elementos se pueden recorrer en distintas formas
- Sus muy útiles para realizar búsquedas y recuperar información.
- Si el árbol no está vacío hay un único elemento llamado raíz que no tiene padre(antecesor).
- Todo elemento posee un único padre y es descendiente de la raíz.

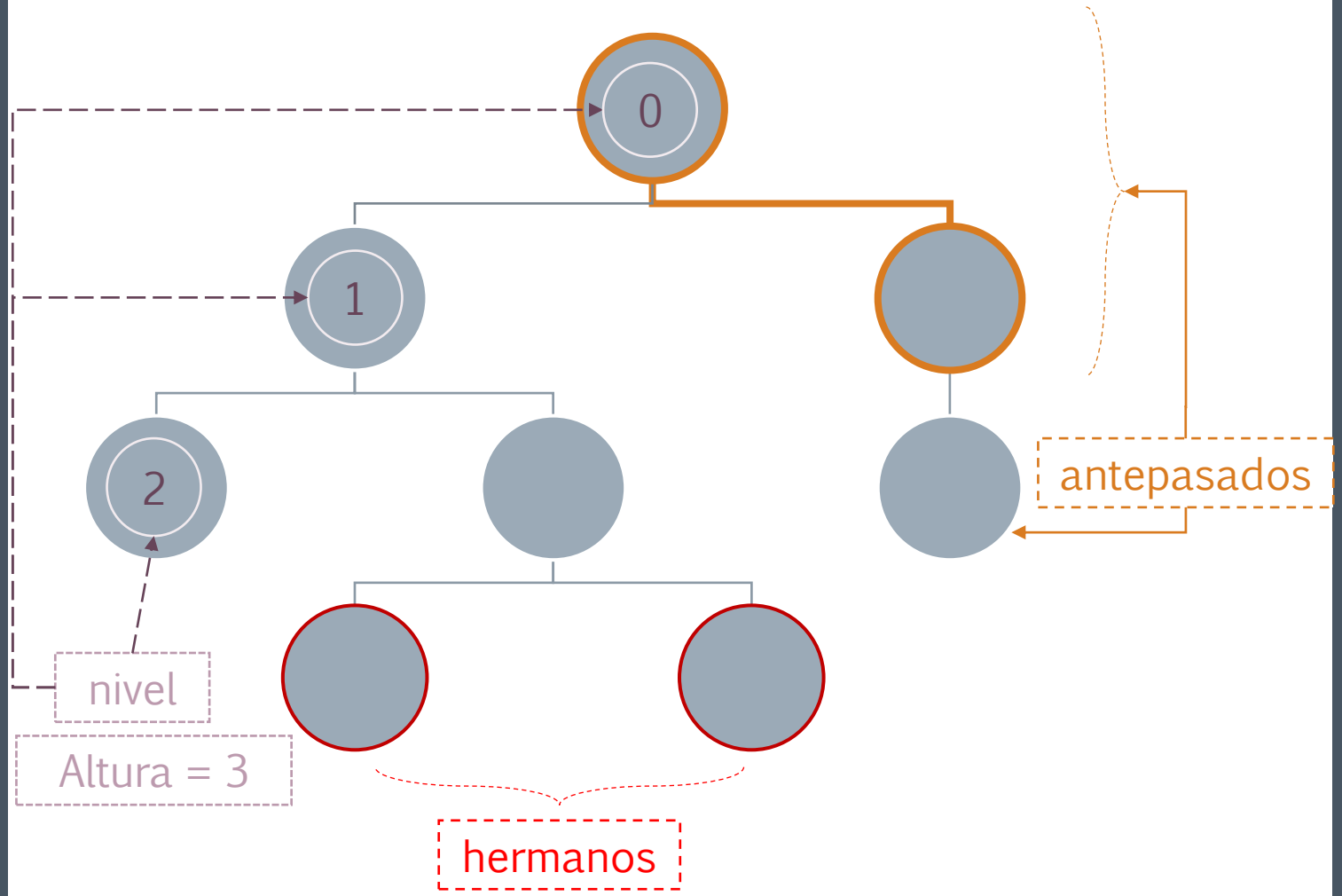
DEFINICIONES

- **Hojas**, son los nodos que no poseen hijos.
- **Camino** es la secuencia lineal de nodos, donde cada uno de ellos es el padre del próximo.
- **Rama** es el camino que existe desde la raíz hasta una hoja.
- **Subárbol** es un nodo con todos sus descendientes.



DEFINICIONES

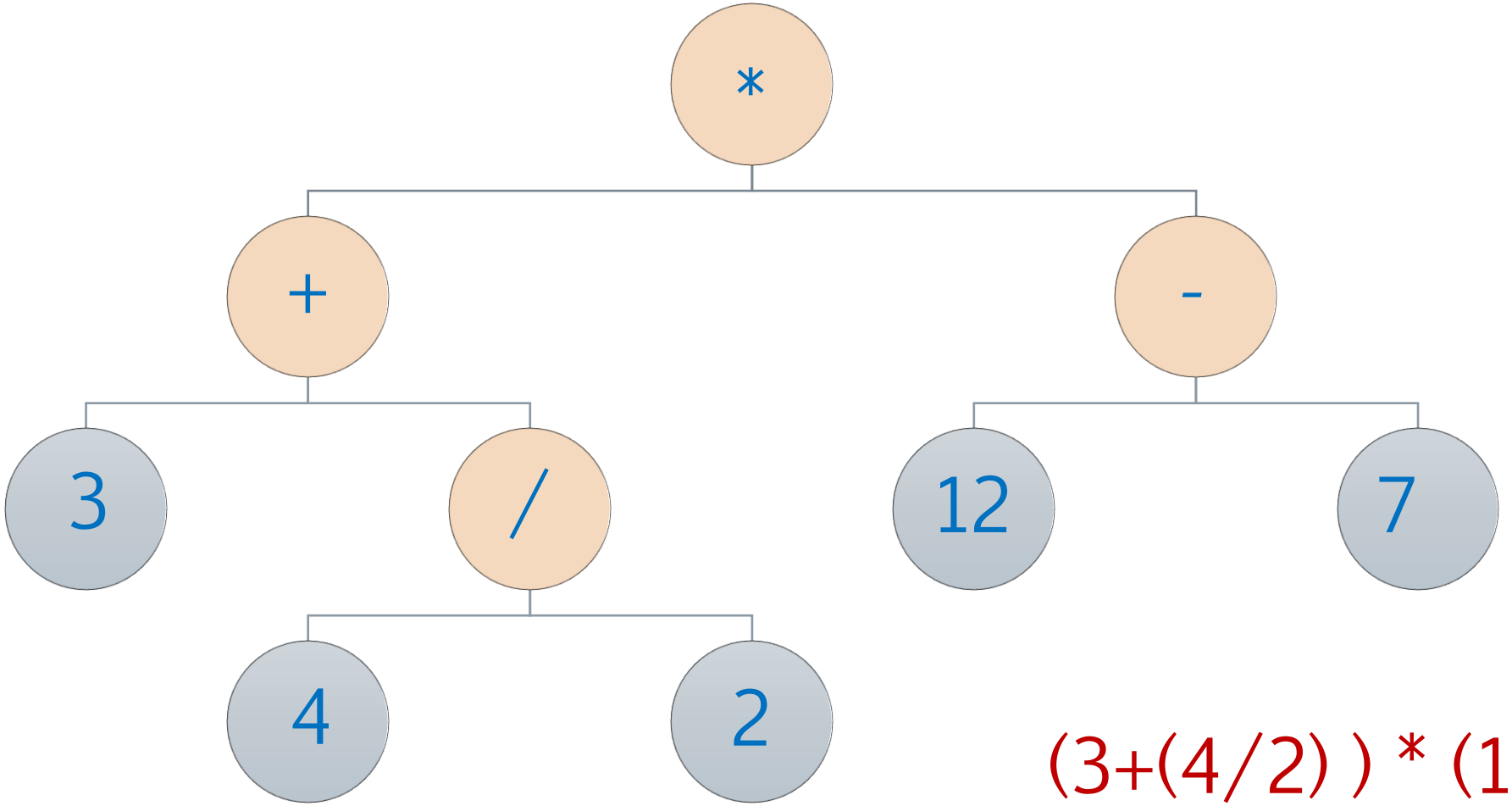
- Nivel de un nodo, es el número de nodos que hay sobre el camino que los une a la raíz.
- Altura del árbol es el máximo entre los niveles.
- Hermanos son los que tienen el mismo padre.
- Antepasado es su padre y todos los antepasados de este.
- Descendientes son sus hijos y los descendientes de estos.



Árboles binarios

- Son de particular interés dada la distribución uniforme de sus elementos. En promedio la profundidad será de $O(\sqrt{N})$.
- Como máximo cada nodo tiene dos hijos.
- Por ejemplo se pueden utilizar para representar expresiones
  - Las hojas representan operandos.
  - Los otros nodos representan operadores.
- **Árboles de búsqueda.**
- **Árboles de expresión.**
- **Árboles de decisión.**

Árboles de expresión



(3+(4/2)) * (12-7)

Árboles Binarios de Búsqueda

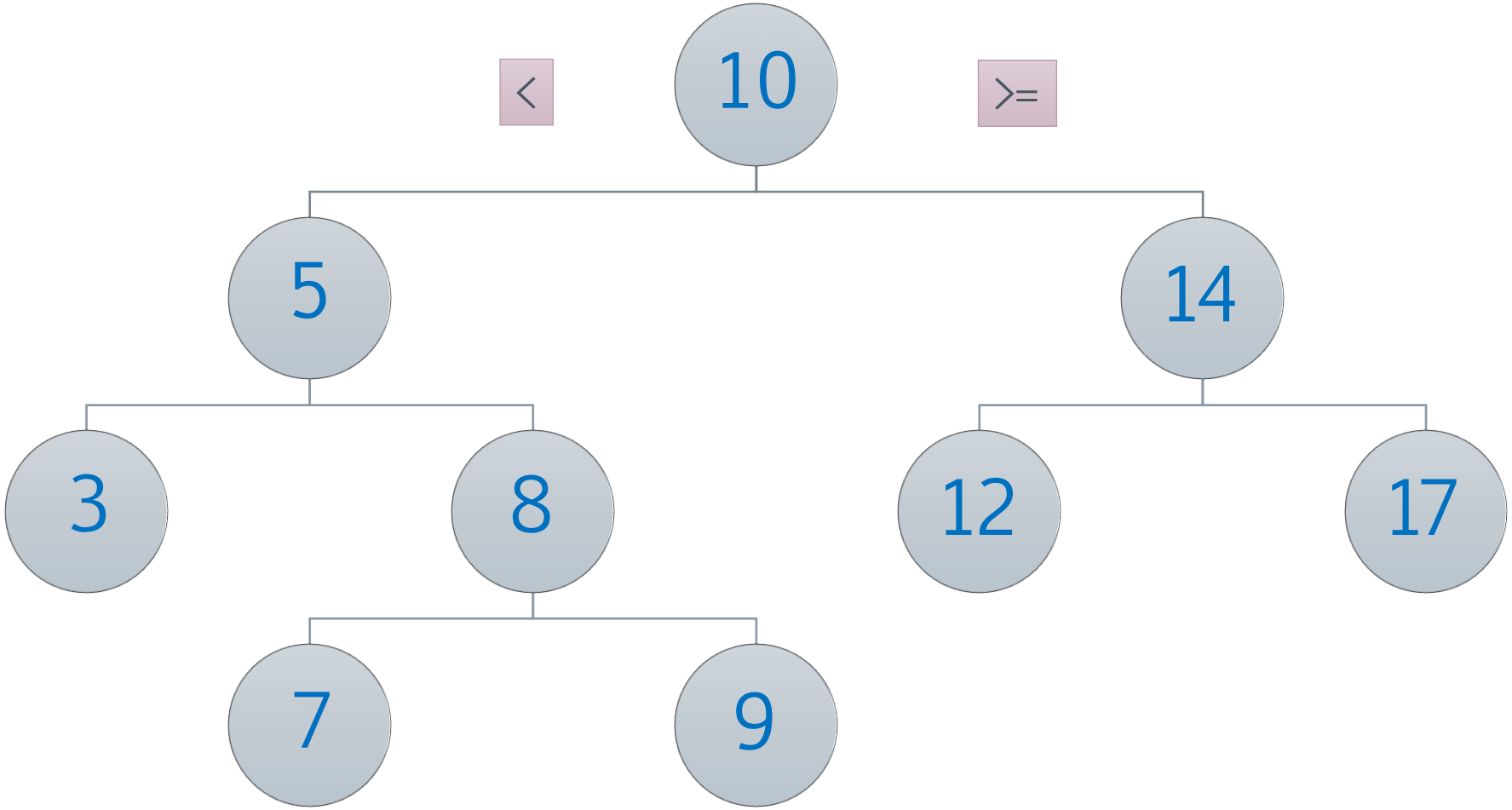


ABB - características

- Cada nodo tiene como **máximo 2 hijos**.
- El campo clave del **hijo izquierdo es menor** al de su padre.
- El campo clave del **hijo derecho es mayor** al de su padre.
- El máximo número de nodos de un nivel N es 2^N

ABB – Operaciones TDA

- Crear el árbol (vaciar)
- Saber si el árbol está vacío
- Insertar un nodo
- Buscar una clave
- Recorrer el árbol
- Borrar un nodo

ABB

insertar(arbol, 3);

insertar(arbol, 1);

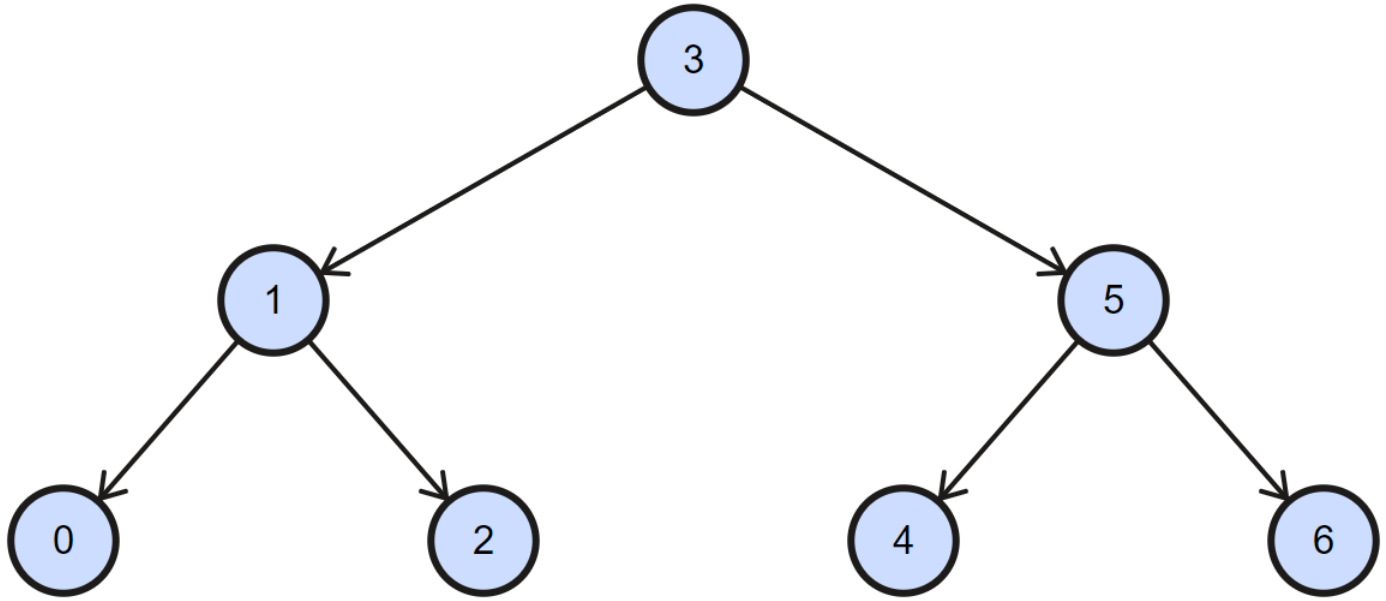
insertar(arbol, 5);

insertar(arbol, 2);

insertar(arbol, 6);

insertar(arbol, 4);

insertar(arbol, 0);



EJERCICIOS

Generar los árboles binarios de búsqueda que se generarán si se ingresan los datos:

1) DBFACEG

2) BADCGFE

3) GFEDCBA

```
procedure insertarNF(var arbol:TArbol; const e:Telemento);
var nodo,aux, ant:TArbol;
begin
       nuevo(nodo,e);

       if(vacio(arbol))then //esta vacío?
              arbol := nodo
       else begin //buscamos la posición
            aux := arbol;
            while(aux <> nil)do begin
                ant := aux;
                if(aux^.info > e)then
                     aux := aux^.izq
                else
                     aux := aux^.der;
                end;
            aux := nodo
            end;
end;
```

```
procedure insertarF(var arbol:TArbol; const e:Telemento);
var nodo,aux, ant:TArbol;
begin
       nuevo(nodo,e);
       if(vacio(arbol))then //esta vacío?
              arbol := nodo
       else begin //buscamos la posición
            aux := arbol;
            while(aux <> nil)do begin
                ant := aux;
                if(aux^.info > e)then
                     aux := aux^.izq
                else
                     aux := aux^.der;
                end;
            if(ant^.info > e)then //verificamos qué hijo es
                ant^.izq := nodo
            else
                ant^.der := nodo
            end;
end;
```

¿ RECURSIVO ?

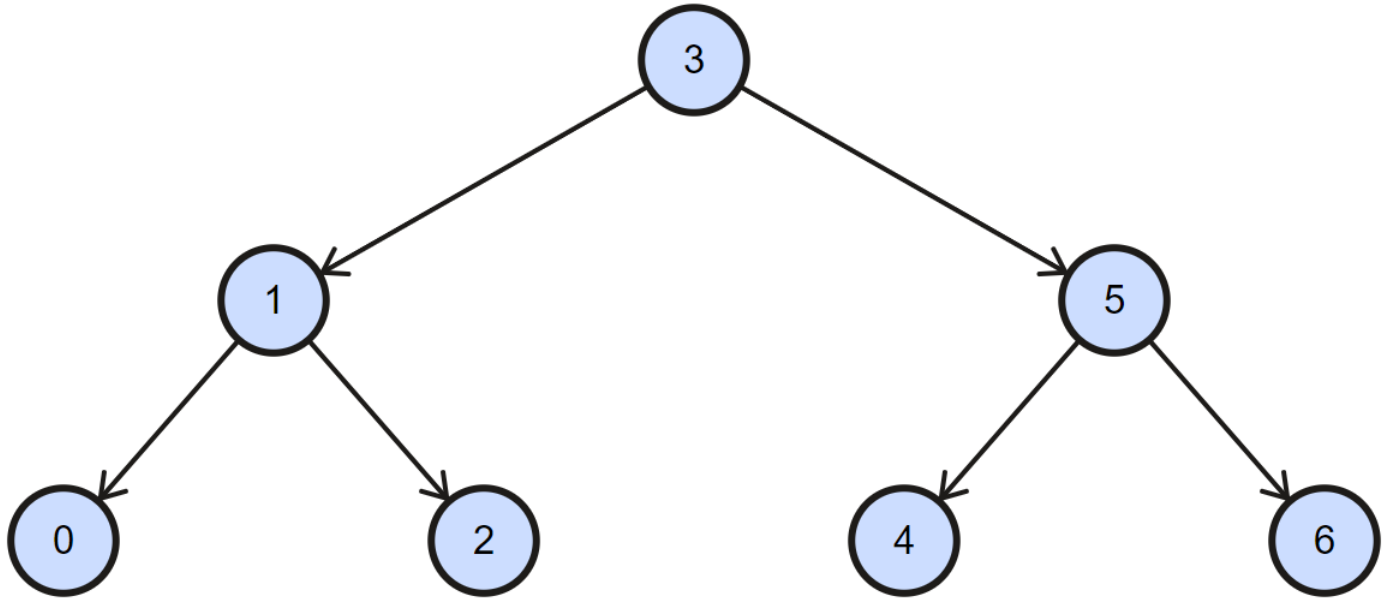
```
procedure insertar_recursivo(
          var arbol:TArbol; const e:Telemento);
begin
     if (arbol = nil) then
          nuevo(arbol,e)
     else if e < arbol^.info then
          insertar_recursivo(arbol^.izq, e)
     else
          insertar_recursivo(arbol^.der, e)
end;
```

Recorridos

- Recorrido en orden
- Recorrido en preorden
- Recorrido en postorden
- Recorrido por niveles

Recorrido en orden

Hijo izquierdo – Nodo – Hijo derecho



0 – 1 – 2 – 3 – 4 – 5 – 6

Recorrido en orden – implementación

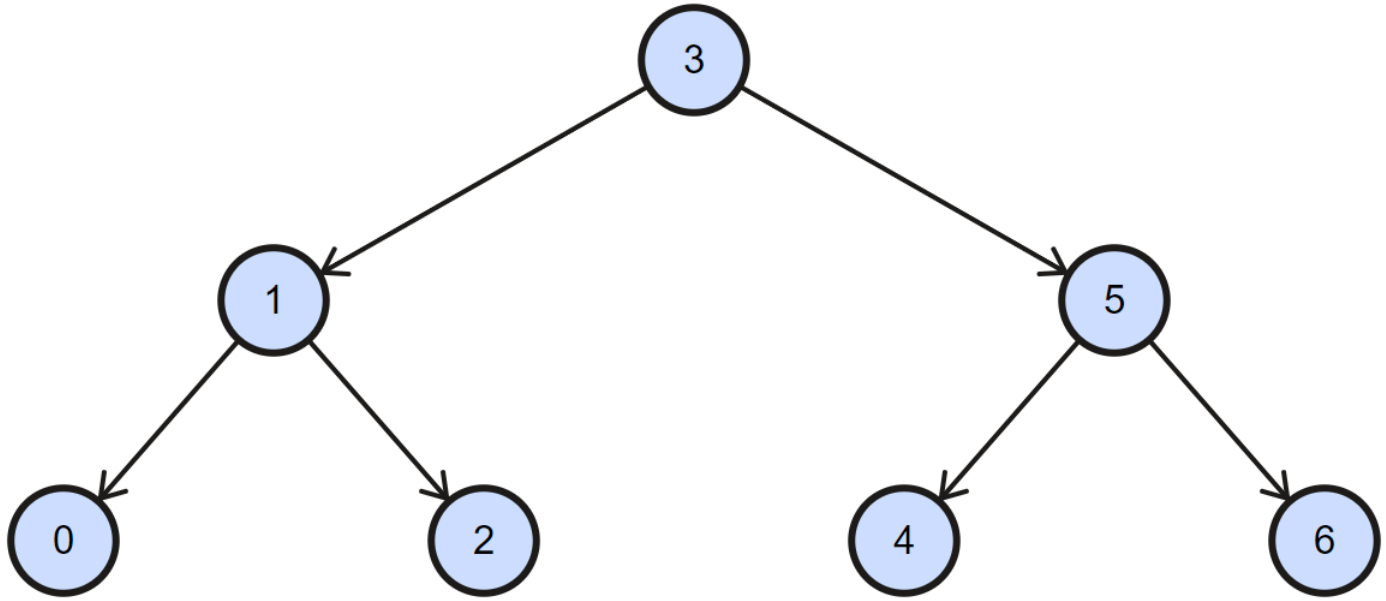
```
procedure listarEnOrden(arbol:TArbol);
begin
     if(arbol<>NIL) then
          begin
          listarEnOrden(arbol^.izq);
          writeln(arbol^.info);
          listarEnOrden(arbol^.der);
          end;
end;
```

¿ ITERATIVO ?

```
procedure listarEnOrdenIterativo(arbol:TArbol);
var pila : TPila; nodo : TArbol;
begin
        uPilaArbol.crear(pila);
        nodo  := arbol;
    Repeat
        while nodo  <>  nil Do
            begin
            uPilaArbol.meter(pila, nodo);
            nodo  := nodo^.Izq;
            end;
        if not uPilaArbol.vacia(pila) then
            begin
            uPilaArbol.sacar(pila, nodo);
            writeln(nodo^.info);
            nodo  := nodo^.der;
            end;
    until (nodo  = nil) and uPilaArbol.vacia(pila);
end;
```

Recorrido preorden

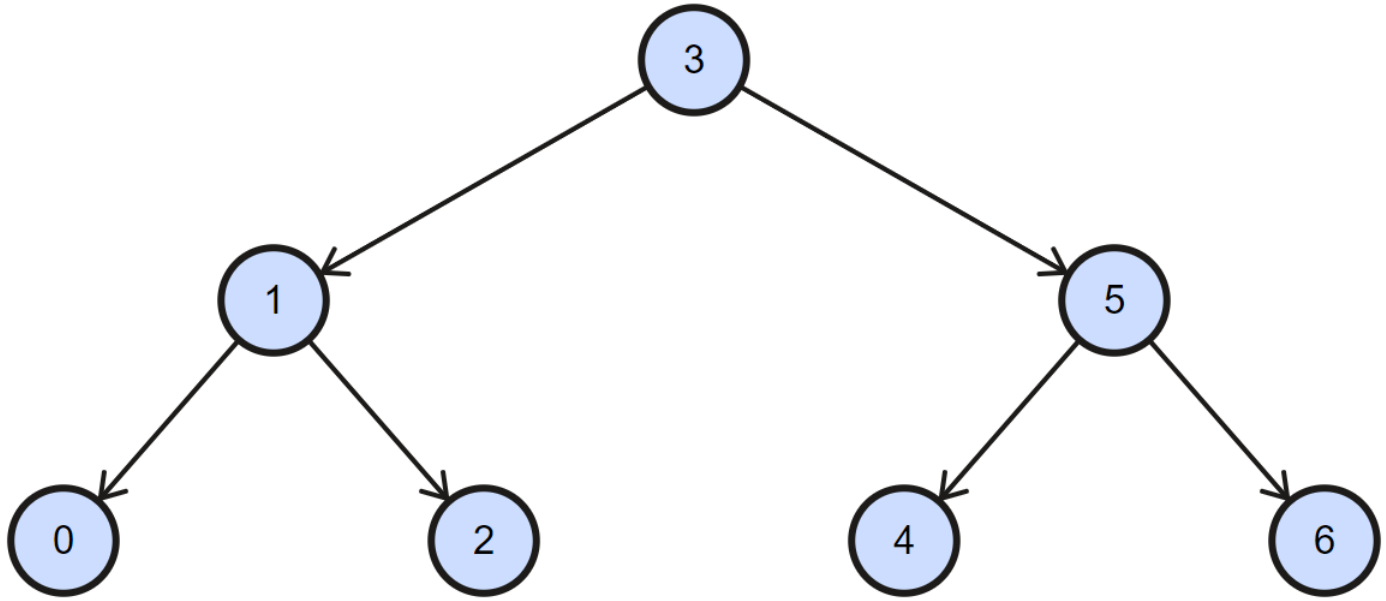
Nodo – Hijo izquierdo – Hijo derecho



3 – 1 – 0 – 2 – 5 – 4 – 6

Recorrido postorden

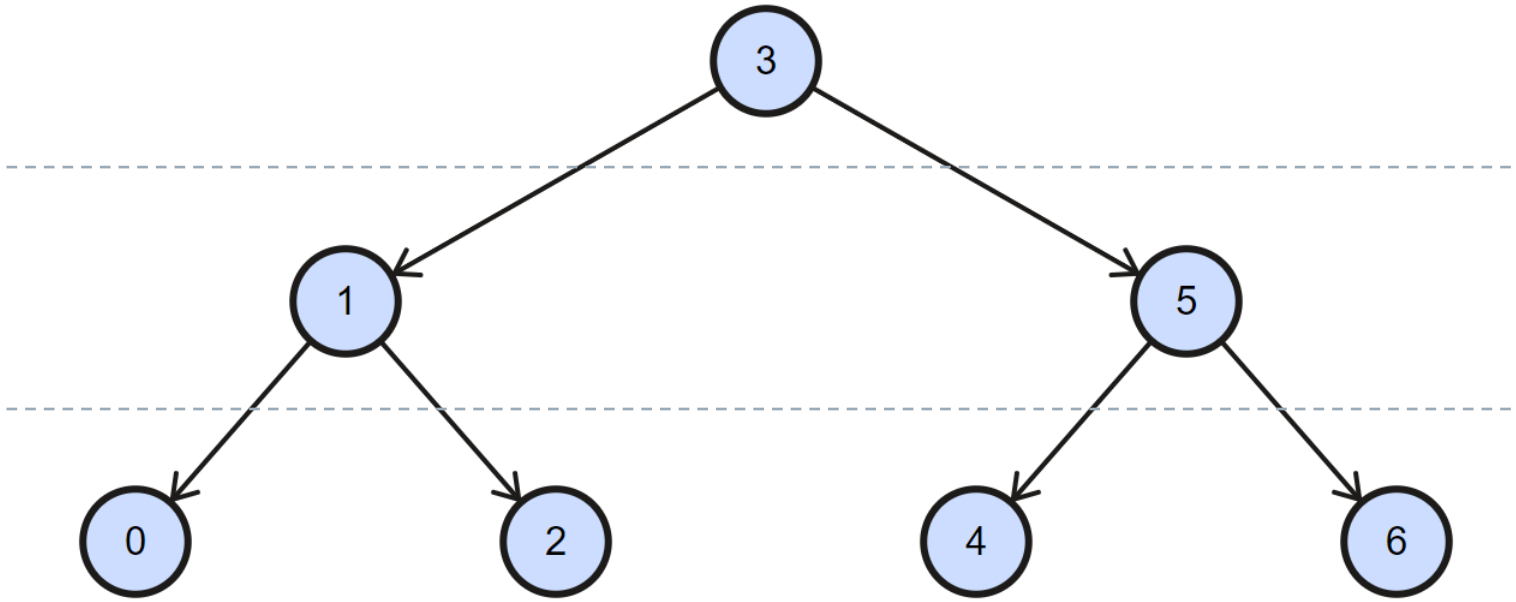
Hijo izquierdo – Hijo derecho – Nodo



0 – 2 – 1 – 4 – 6 – 5 – 3

Recorrido por niveles

Hijo izquierdo – Hijo derecho – Nodo



3 – 1 – 5 – 0 – 2 – 4 – 6

```
procedure listarPorNivel(nodo:TArbol);
var
      Cola :TCola;
begin
   uColaArbol.crear(cola);
   uColaArbol.meter(cola, nodo);
   while not uColaArbol.vacia(Cola) do
      begin
      uColaArbol.sacar(cola, nodo);
      writeln(nodo^.info);
      if nodo^.izq <> nil then
            uColaArbol.meter(cola, nodo^.izq);
      if nodo^.der <> nil then
            uColaArbol.meter(cola, nodo^.der);
      end;
end;
```

Ejercicio

- Realice una función que dado un árbol y una clave retorne el nodo que contiene dicha clave.

  Si el árbol no contiene la clave buscada deberá retornar nulo.

```
function buscar(arbol:TArbol; const e:TElemento):TArbol;
begin
    while (arbol<>nil) and (arbol^.info<>e) do
        if(arbol^.info>e)then
                arbol := arbol^.izq
        else
                arbol := arbol^.der;
    buscar := arbol;
end;
```

Bibliografía

- Data Structures. Nalle Dale.
- Estructuras de datos en C. A, Tenenbaum - Y, Langsam – M, Augenstein Un libro
- Estructuras de datos y algoritmos. M, Weiss

Herramientas:

- http://btv.melezinek.cz/binary-search-tree.html
- https://www.cs.usfca.edu/~galles/visualization/BST.html