Programación 2

La previa: Estructuras Arborescentes (parte 1)

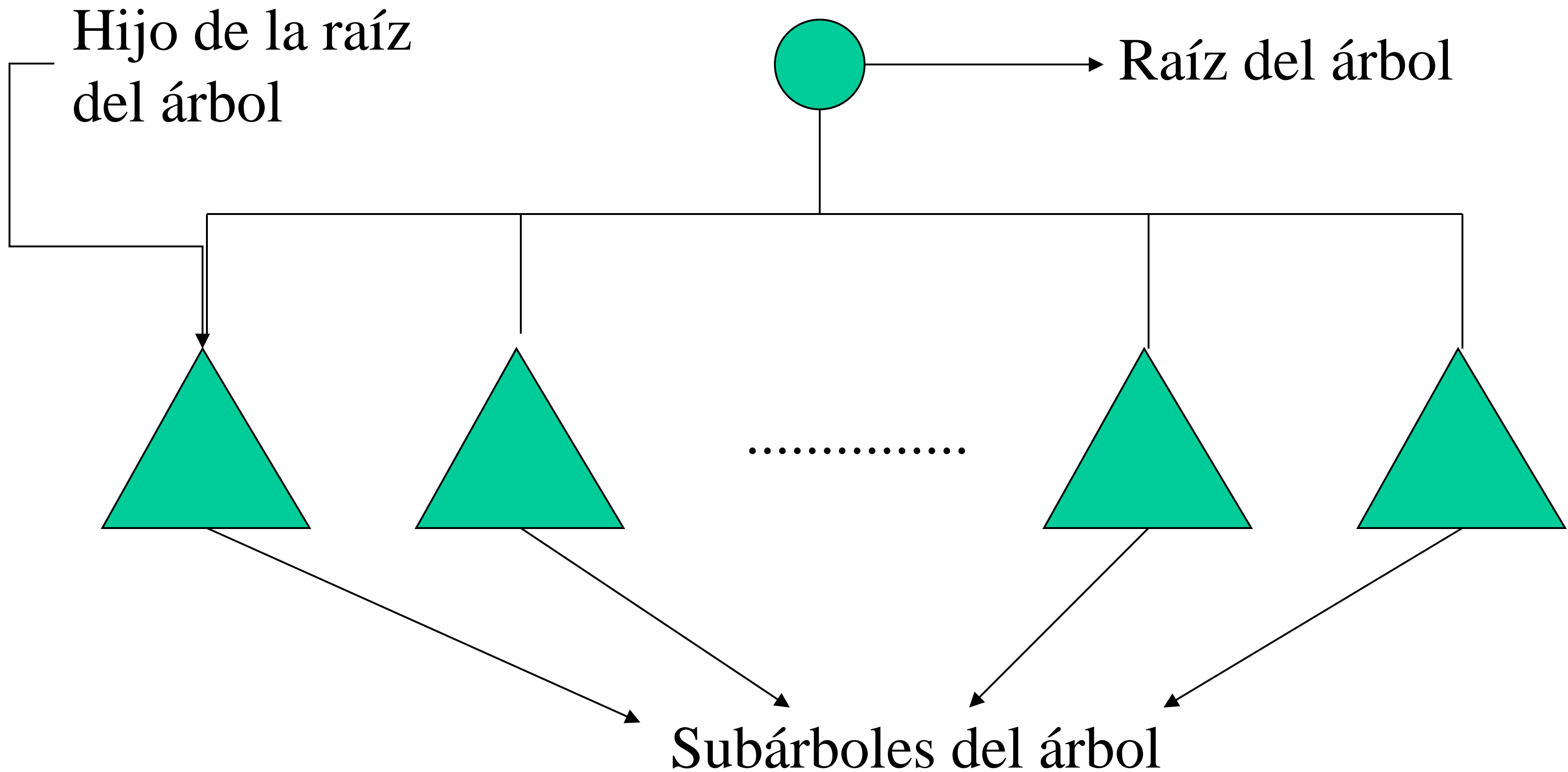
Definición

La recursión puede ser utilizada para la definición de estructuras realmente sofisticadas.

Una estructura *árbol* (*árbol general o finitario*) con tipo base T es,

1. O bien la estructura vacía
2. O bien un elemento de tipo T junto con un número finito de estructuras de árbol, de tipo base T, disjuntas, llamadas *subárboles*

Conceptos básicos (cont.)



Los elementos se ubican en *nodos* del árbol.

Arboles n-arios y binarios

La descripción de la noción de árbol dada antes es casi una definición inductiva.

Falta precisar qué se quiere decir con "un número finito ..." en la segunda cláusula de construcción de árboles.

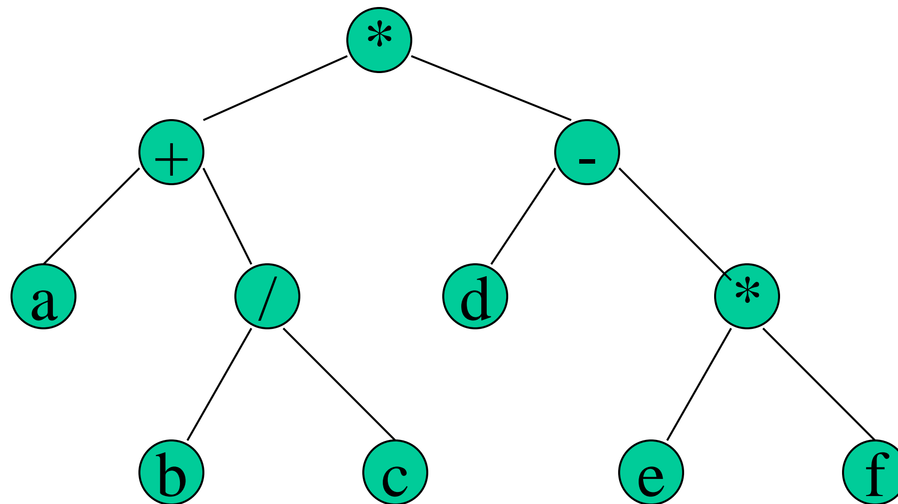
Existe un caso particular de árbol en que esta cláusula se hace precisa fácilmente:

"... junto con exactamente 2 (dos) subárboles binarios."

Este es un caso particular de **árboles n-arios**, que son a su vez un caso particular de árboles generales o finitarios.

Ejemplo: árbol de expresiones Sintaxis concreta vs. sintaxis abstracta

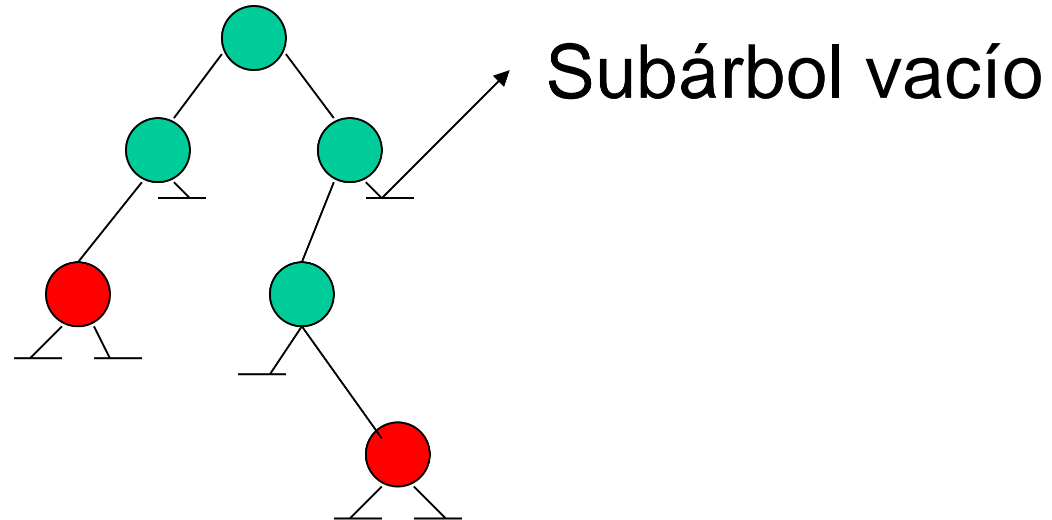
Representación no ambigua de expresiones aritméticas.



Representación arborescente de la fórmula:

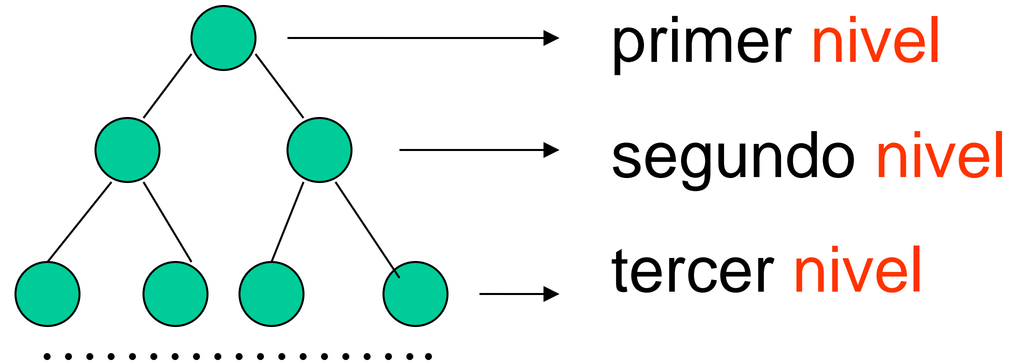
(a + b / c) * (d - e * f)

Hojas



Def: *Hojas* son los nodos cuyos (ambos) subárboles son vacíos

Niveles y altura



Def.

La *altura* de un árbol es:

la cantidad de niveles que tiene, o

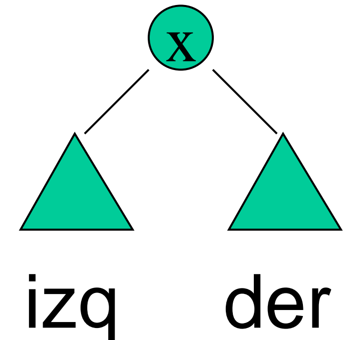
la cantidad de nodos en el camino más largo de la raíz a una hoja.

La altura del árbol binario vacío es 0.

Altura (cont.)

Definición Inductiva de Árboles Binarios

$$\frac{}{() : \text{ArbBin}} \qquad \frac{\text{izq} : \text{ArbBin} \quad x : T \quad \text{der} : \text{ArbBin}}{(\text{izq}\ ,\ t\ ,\ \text{der}) : \text{ArbBin}}$$




```
altura (()) = 0
altura ((izq,a,der)) =
            1 + max(altura(izq), altura(der))
```

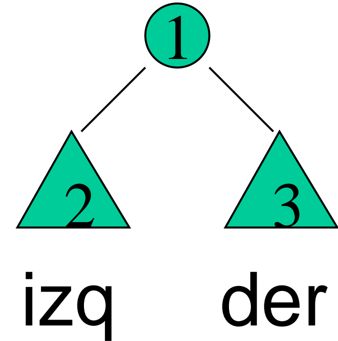
Recorridas de árboles binarios

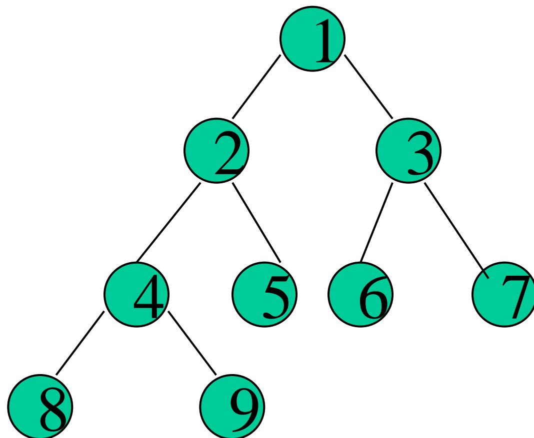
Para recorrer un árbol no vacío hay tres órdenes naturales, según la raíz sea visitada:

- antes que los subárboles
  (PreOrden - preorder)
- entre las recorridas de los subárboles
  (EnOrden - inorder)
- después de recorrer los subárboles
  (PostOrden - postorder)

Recorridas de árboles binarios (cont.)

Antes que los subárboles (preorder)

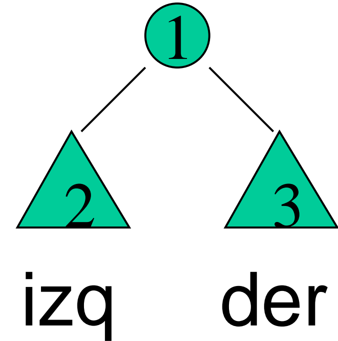


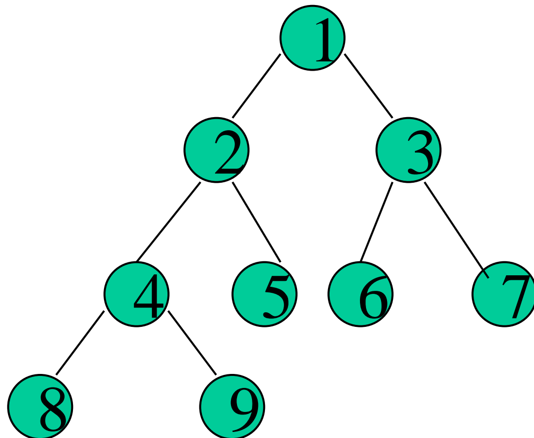


Preorder: …

Recorridas de árboles binarios (cont.)

Antes que los subárboles (preorder)

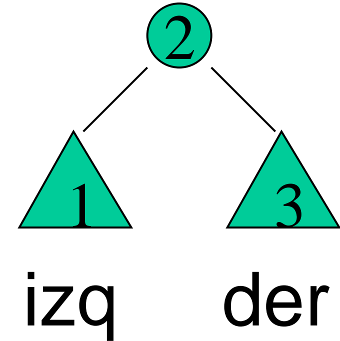


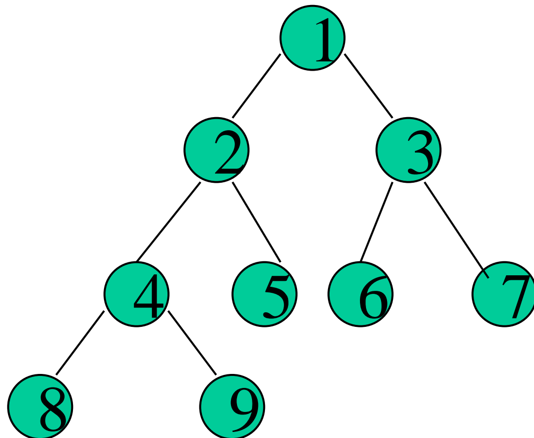


Preorder: 1, 2, 4, 8, 9, 5, 3, 6, 7

Recorridas de árboles binarios (cont.)

Antes que los subárboles (inorder)

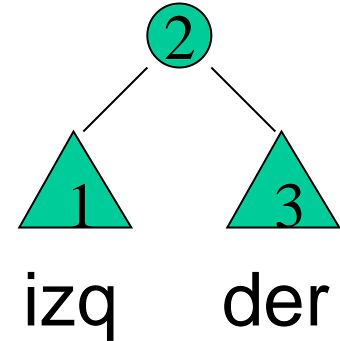


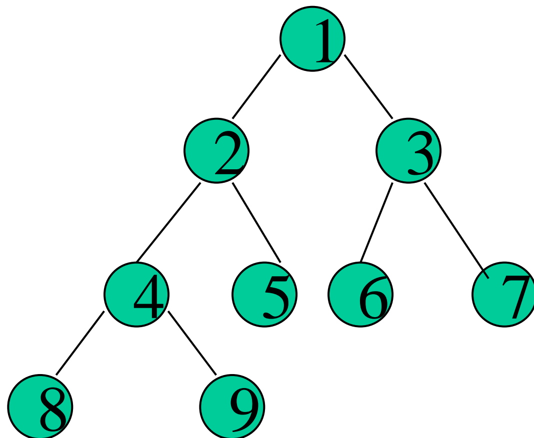


Inorder: …

Recorridas de árboles binarios (cont.)

Antes que los subárboles (inorder)

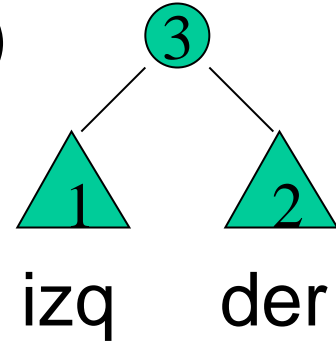


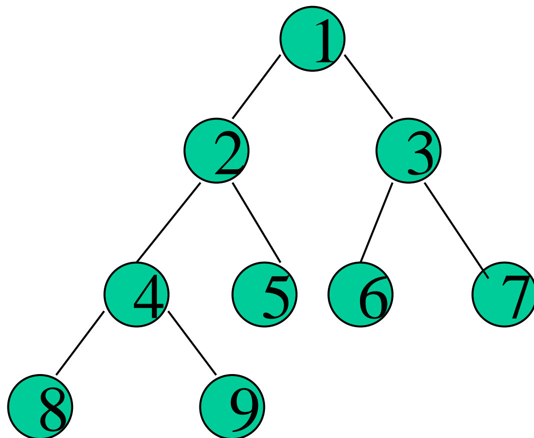


Inorder: 8, 4, 9, 2, 5, 1, 6, 3, 7

Recorridas de árboles binarios (cont.)

<u>Antes</u> que los subárboles (postorder)





<u>Postorder</u>: 8, 9, 4, 5, 2, 6, 7, 3, 1

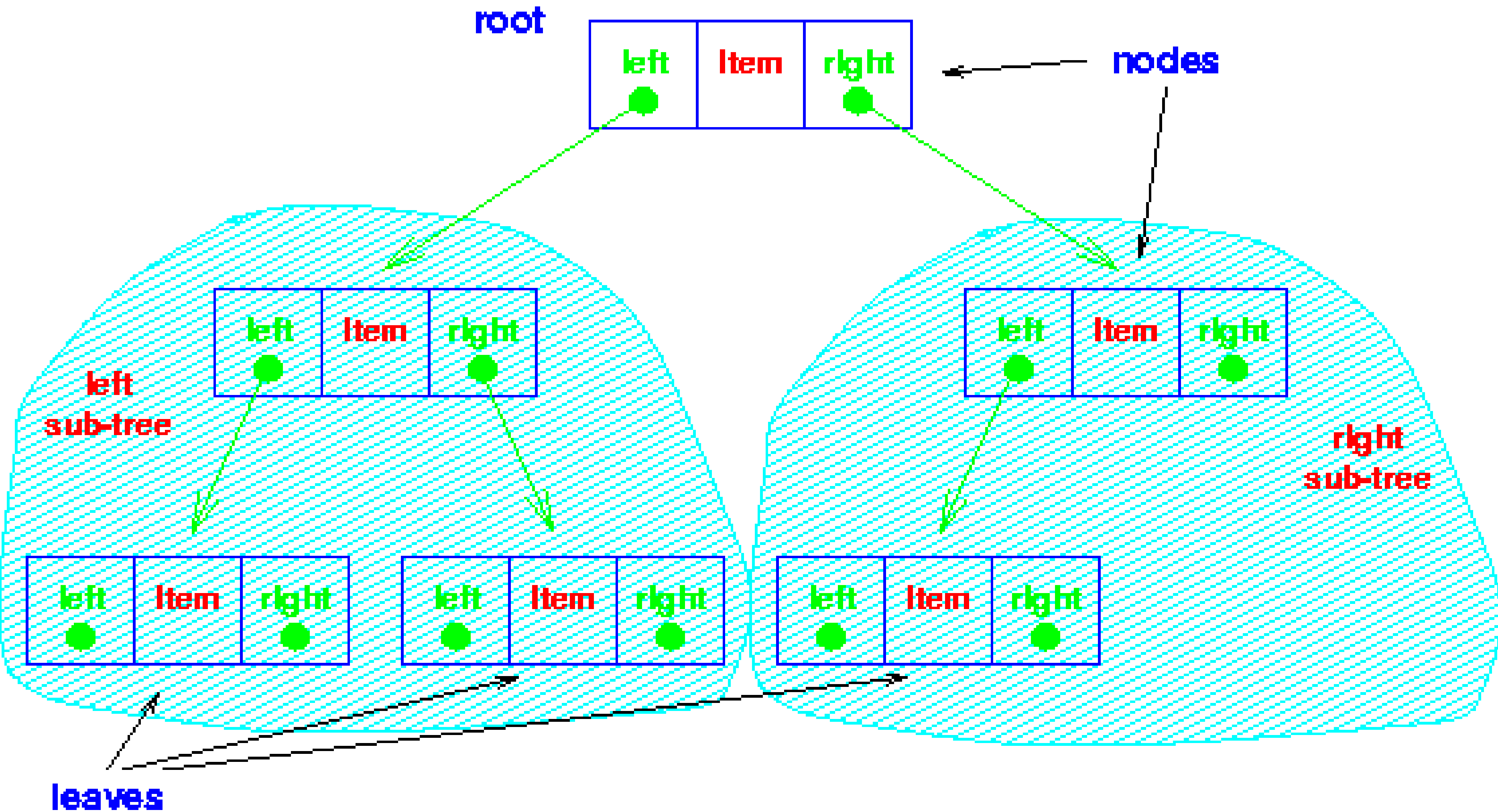
Implementación en C y C++

Ahora presentaremos una posible implementación del tipo de dato árbol binario (ArbBin) en C++. Los elementos del árbol son de tipo T en la siguiente definición:

```
typedef NodoAB* AB;

struct NodoAB{
   T item;
   AB left, right;
};
```

Arbol Binario (AB)



Procedimiento preOrden

```
void preOrden (AB t){
      if (t != NULL){
            P(t -> item);
            preOrden(t -> left);
            preOrden(t -> right);
      }
}
```

Procedimiento enOrden

```
void enOrden (AB t){
      if (t != NULL){
            enOrden(t -> left);
            P(t -> item);
            enOrden(t -> right);
      }
}
```

Procedimiento postOrden

```
void postOrden (AB t){
      if (t != NULL){
            postOrden(t -> left);
            postOrden(t -> right);
            P(t -> item);
      }
}
```

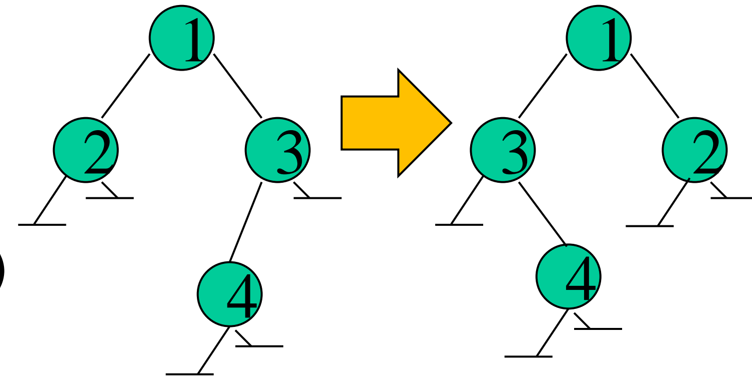
Cantidad de nodos de un árbol

```
cantNodos (()) = 0
cantNodos ((izq,a,der)) =
     1 + cantNodos(izq) + cantNodos(der)
```

```
int cantNodos (AB t){
     if (t == NULL) return 0;
     else
          return 1 + cantNodos(t->left)
                   + cantNodos(t->right);
}
```

¿Cuál es la diferencia con la función altura?

Espejo

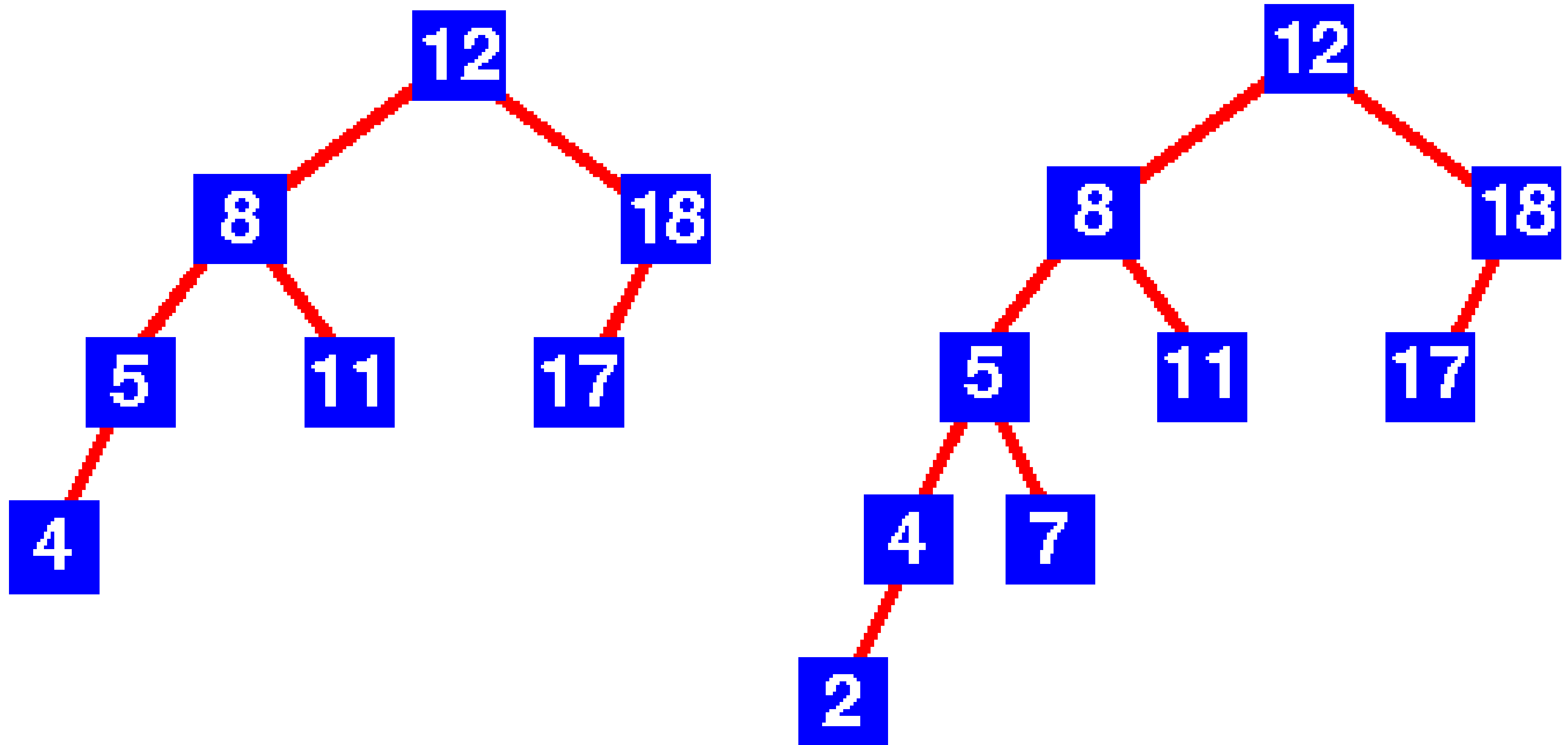



```
espejo (()) = ()
espejo ((izq,a,der)) =
   (espejo(der),a,espejo(izq))
```

```
AB espejo (AB t){
     if (t == NULL) return NULL;
     else
        { AB rt = new NodoAB;
          rt -> item = t -> item;
          rt -> left = espejo (t -> right);
          rt -> right = espejo (t -> left);
          return rt;
        }
}
```

La función que retorna una copia idéntica de un árbol, sin compartir memoria, ¿se parece a la función espejo?

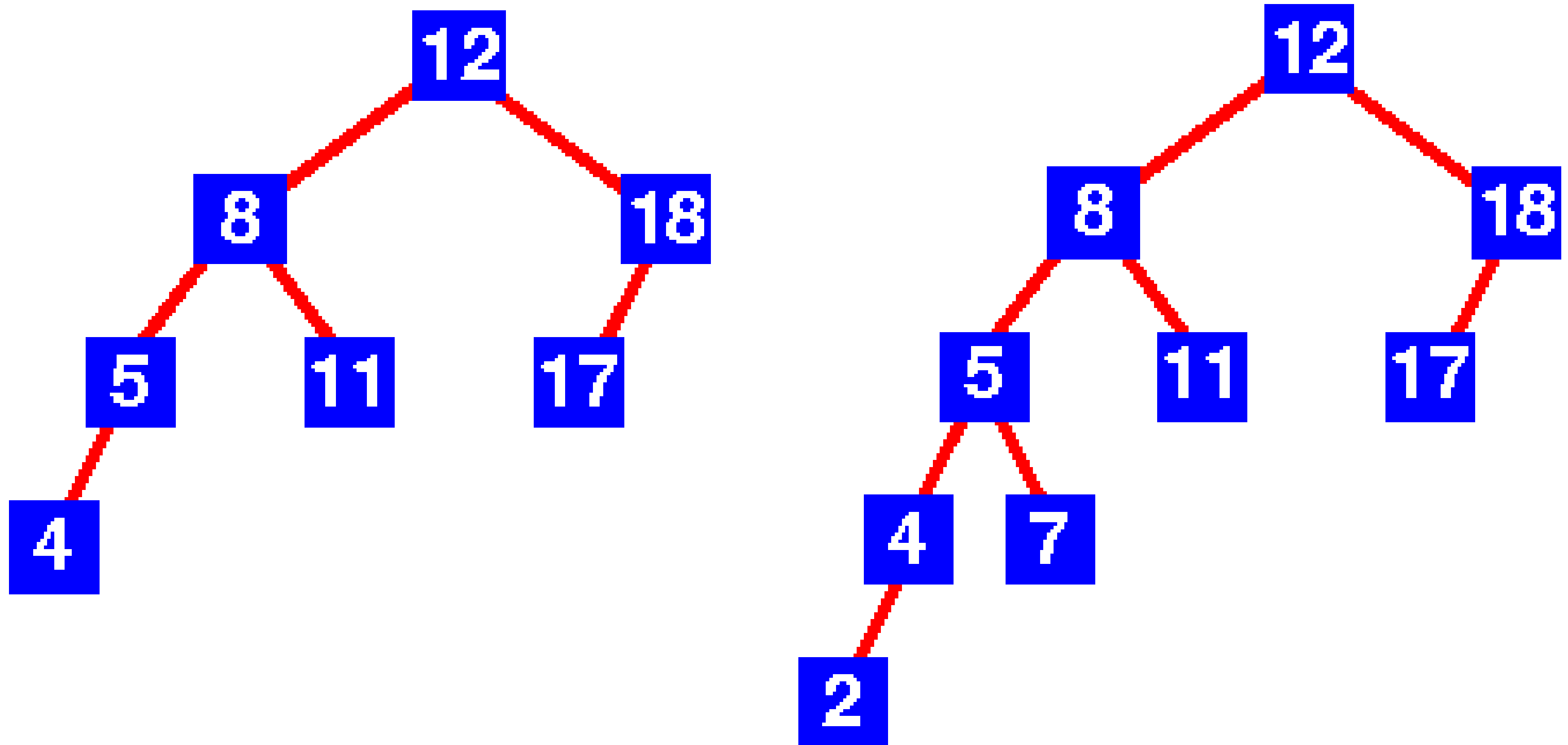
Arbol binario de búsqueda (ABB): Ejemplos



Analizar la relación con la búsqueda binaria sobre un arreglo ordenado (y sobre una lista).

3	5	8	10	14	22	29	45	77

Arbol binario de búsqueda (ABB): Ejemplos



¿Dónde está el mínimo y el máximo en un ABB?

¿Qué pasa con un recorrido en orden de un ABB?

Implementación de ABB

La implementación del tipo *ABB* es muy similar a la de ArbBin. La diferencia es que ahora diferenciamos un campo, **`key`**, cuyo tipo es un ordinal (**`Ord`**), del resto de la información del nodo:

```
typedef NodoABB* ABB;

struct NodoABB {
    Ord key;
    T info;
    ABB left, right;
};
```

Buscar iterativo

```
ABB buscarIterativo(Ord x, ABB t){
  while ((t != NULL) && (t -> key != x)){
        if (t -> key > x)
           t = t -> left;
        else t = t -> right;
  }
  return t;
}
```

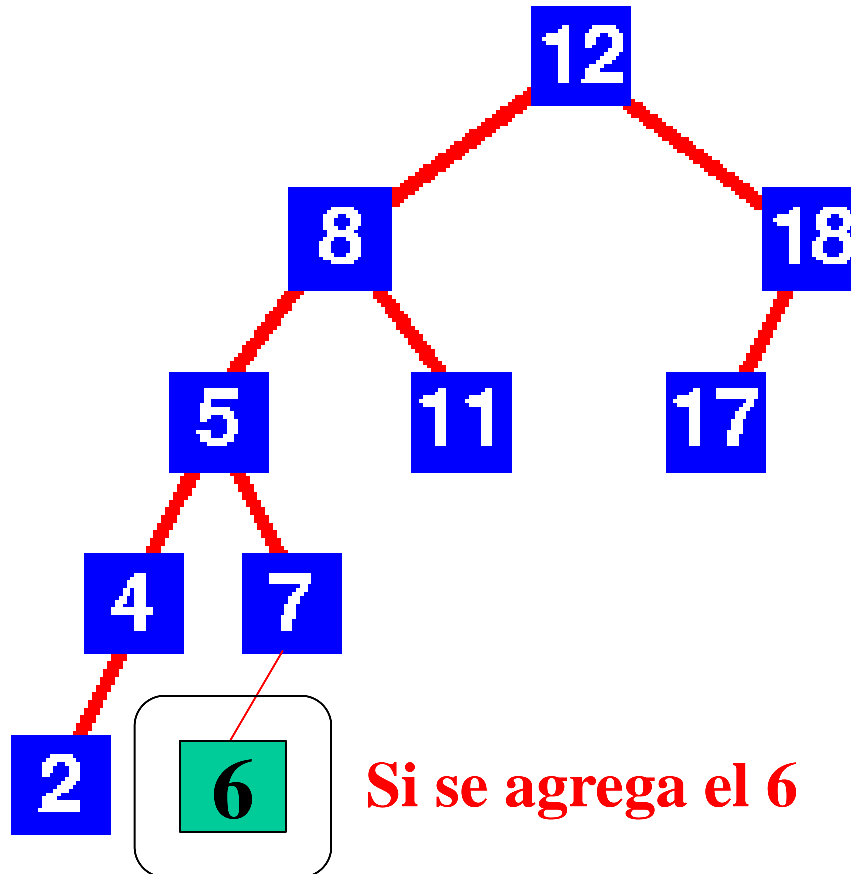
La función Pertenece

```
bool pertenece(Ord x, ABB t){
  if (t == NULL) return false;
  else
     if (x == t->key)
       return true;
     else
       return (pertenece(x,t->left)
            || pertenece(x,t->right));
}
```

¿Es eficiente la función *pertenece*?

¿Podría optimizarse?

Inserción en un ABB



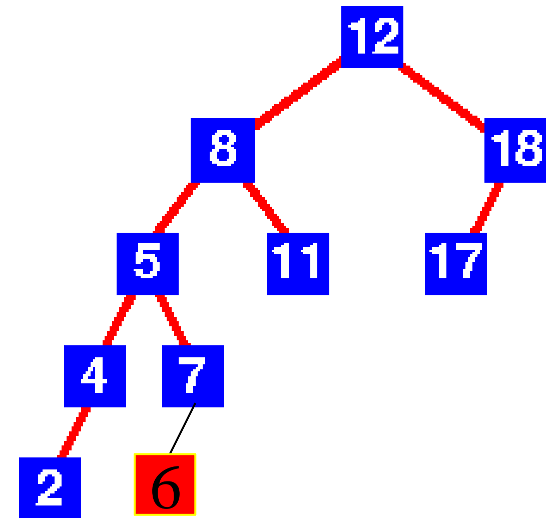
Si se agrega el 6

¿Siempre se agrega (eventualmente) un elemento como una hoja?

```
void insABB (Ord clave, T dato, ABB & t)
```

Inserción en un ABB

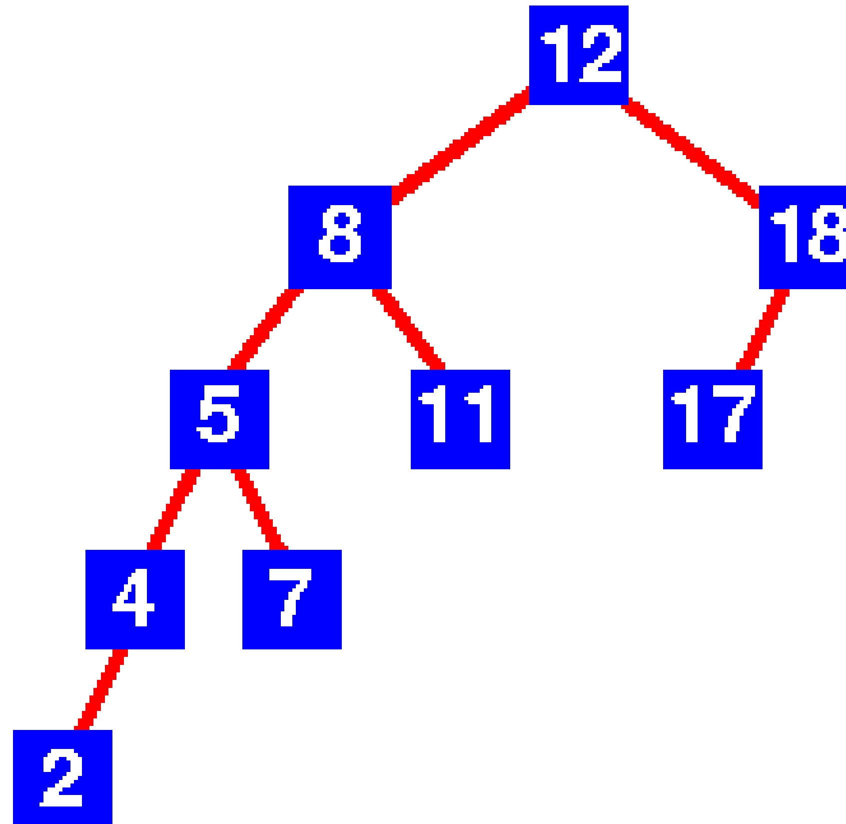
```
void insABB (Ord clave, T dato, ABB & t){
  if (t == NULL){
    t = new NodoABB ;
    t->key = clave;
    t->info = dato;
    t->left = t->right = NULL;
  }
  else if (clave < t->key)
     insABB (clave, dato, t->left);
  else if (clave > t->key)
     insABB (clave, dato, t->right);
}
```



Eliminación en un ABB

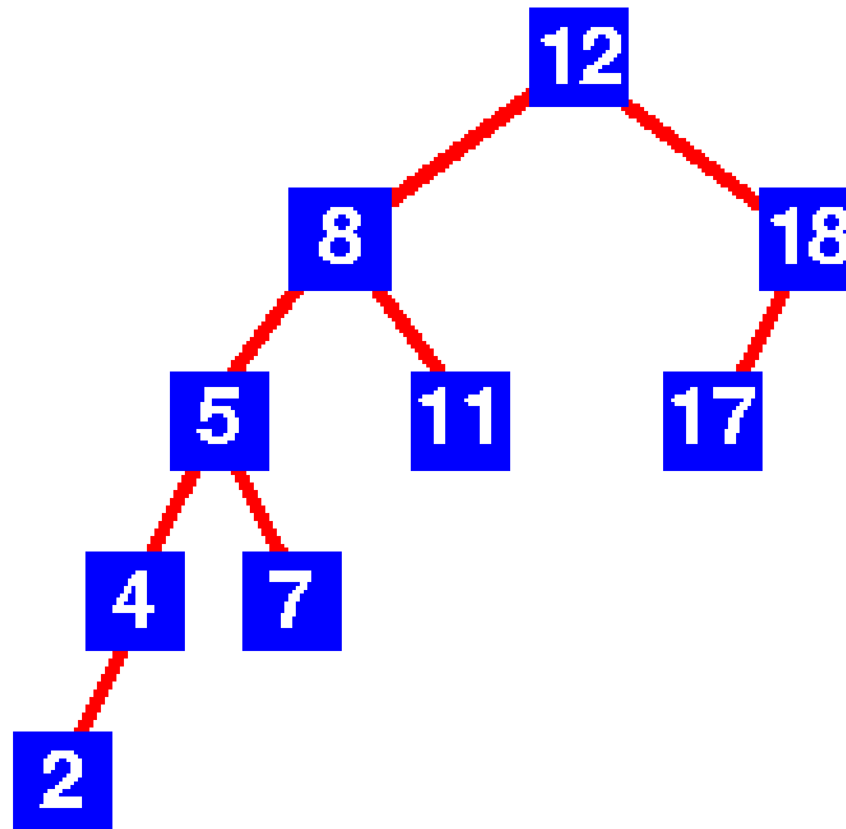
¿Cómo podría ser la eliminación de un elemento de un ABB?

```
void elimABB (Ord clave, ABB & t){ ... }
```



Eliminar los menores que k en un ABB

```
void elim (ABB & t, Ord k){ //pizarrón
...
}
```



Ejercicio: Aplanar eficientemente un ABB

Completar el siguiente código para obtener una lista ordenada con los elementos de un ABB, implementando *aplanarEnLista*, de tal manera que el árbol se recorra solo una vez y que la lista no se recorra al agregar cada elemento:

```
Lista aplanar (ABB t){
  Lista l = NULL;
  aplanarEnLista (t, l);
  return l;
}
```

Compare esta versión de *aplanar* con la que se obtiene usando la concatenación de listas.

Ejercicio: Aplanar eficientemente un ABB

```
Lista aplanar (ABB t){
   Lista l = NULL;
   aplanarEnLista (t, l);
   return l;
}


void aplanarEnLista (ABB t, Lista & l){
   if ( t!= NULL){
       aplanarEnLista(t->right,l);
       insComienzo(t->dato,l);
       aplanarEnLista(t->left,l);
   }
}
```