Estructuras de Datos y Algoritmos 1

5 - Estructuras Arborescentes
Arboles (introducción)

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*

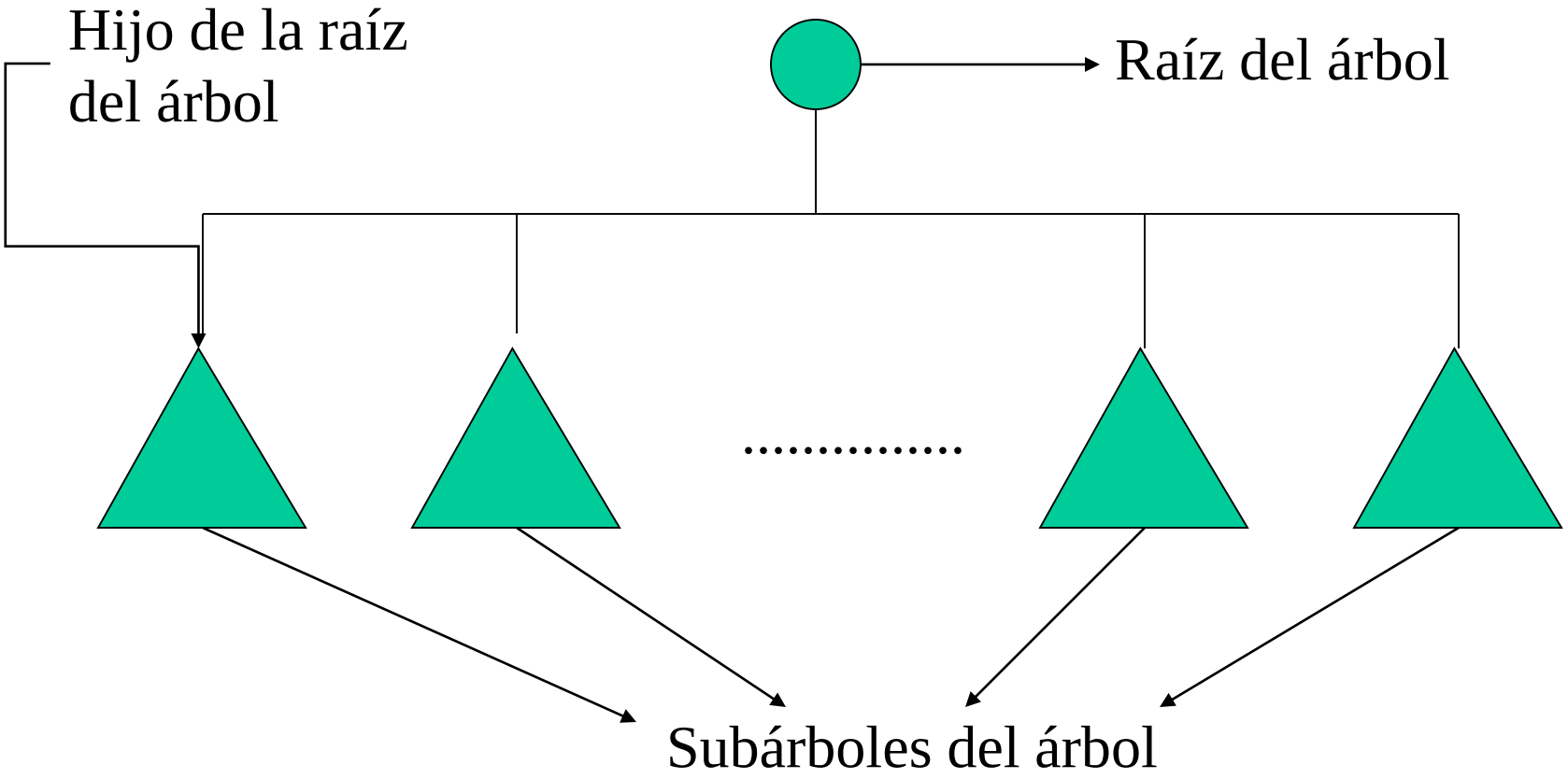
Conceptos básicos

Resulta evidente, a partir de la definición de secuencias (listas) y estructura árbol, que la lista es una estructura árbol en la cual cada nodo tiene a lo sumo un subárbol.

La lista es por ello también conocida por el nombre de árbol degenerado.

Veremos a continuación un poco de nomenclatura:

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.

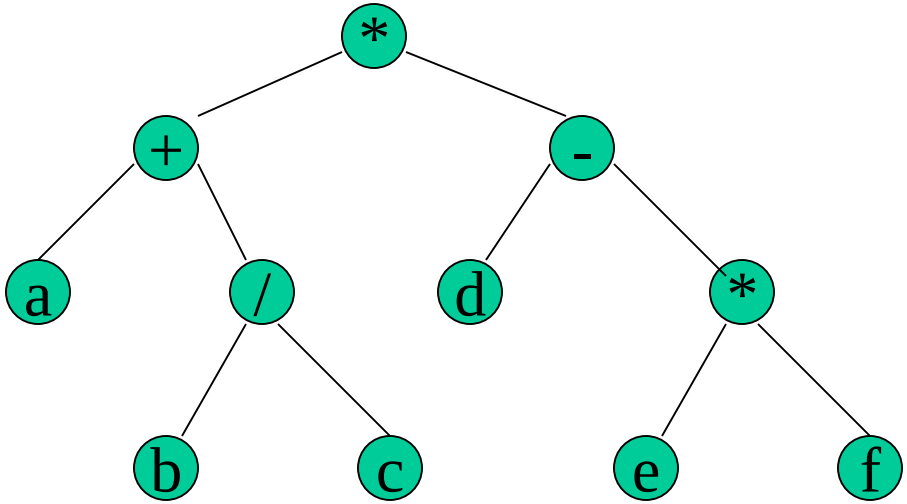
Arboles binarios

NOTA:

Comenzaremos primero a trabajar con árboles binarios, ya que al final de este módulo veremos que es posible representar árboles generales usando árboles binarios (con una semántica particular).

Ejemplo: árbol de expresiones
Sintaxis concreta vs. sintaxis abstracta

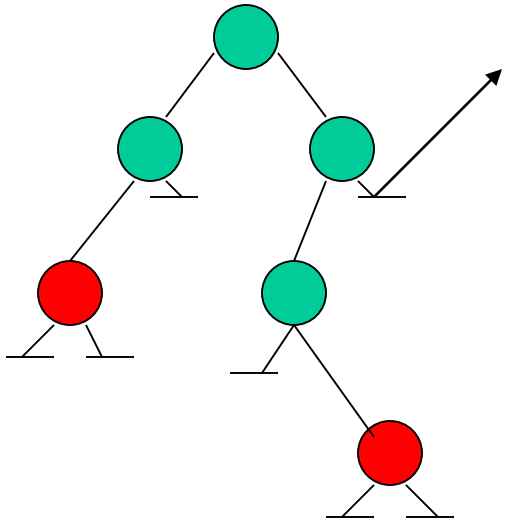
Representación no ambigua de expresiones aritméticas.



Representación arborescente de la fórmula:

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

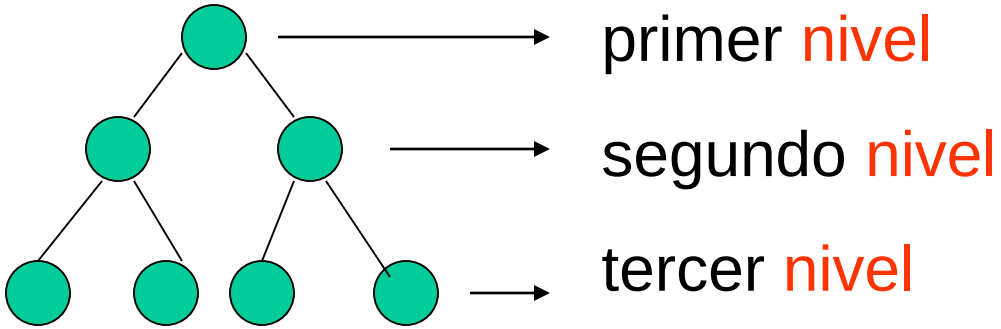
Hojas



Subárbol vacío

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

Niveles y altura

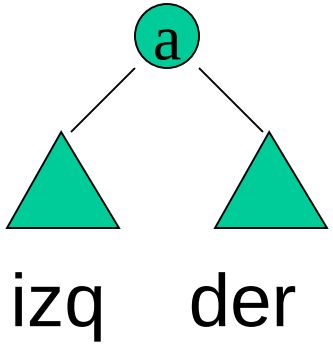


Def.
La *altura* de un árbol es:
la cantidad de niveles que tiene, o
la cantidad de nodos en el camino más largo de la raíz a una hoja.

La altura del árbol binario vacío es 0.

Altura (cont.)

La altura de un árbol de la forma:



es 1 + max(p_i, p_d), donde p_i es la altura de *izq* y p_d es la altura de *der*.

Más formalmente:

Altura (cont.)

Definición Inductiva de Árboles Binarios

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

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

Recorridas de árboles binarios

- En general, son procedimientos que visitan todos los nodos de un árbol binario efectuando cierta acción sobre cada uno de ellos
- La forma de estos procedimientos depende del orden que se elija para visitar los nodos
- Obviamente recorrer un árbol vacío es trivial

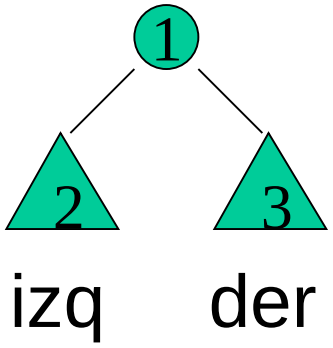
Recorridas de árboles binarios (cont.)

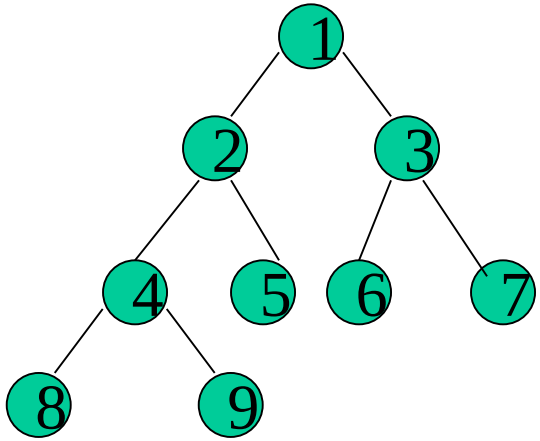
Para recorrer un árbol no vacío hay tres órdenes naturales, según la raíz sea visitada:

- 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)

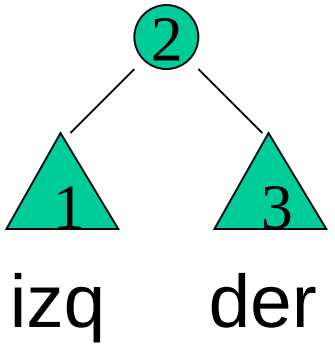


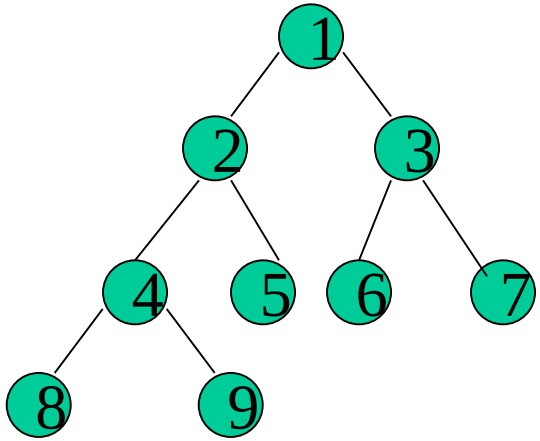


Preorder: 1, 2, 4, 8, 9, 5, 3, 6, 7

Recorridas de árboles binarios (cont.)

Antes que los subárboles (inorder)

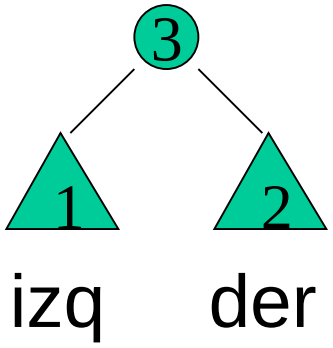


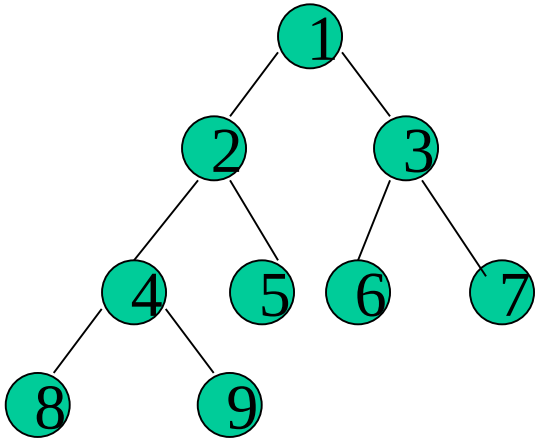


Inorder: 8, 4, 9, 2, 5, 1, 6, 3, 7

Recorridas de árboles binarios (cont.)

Antes que los subárboles (postorder)





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

Ejemplo

Generar la lista de elementos de un árbol binario que se obtiene al recorrerlo en orden:

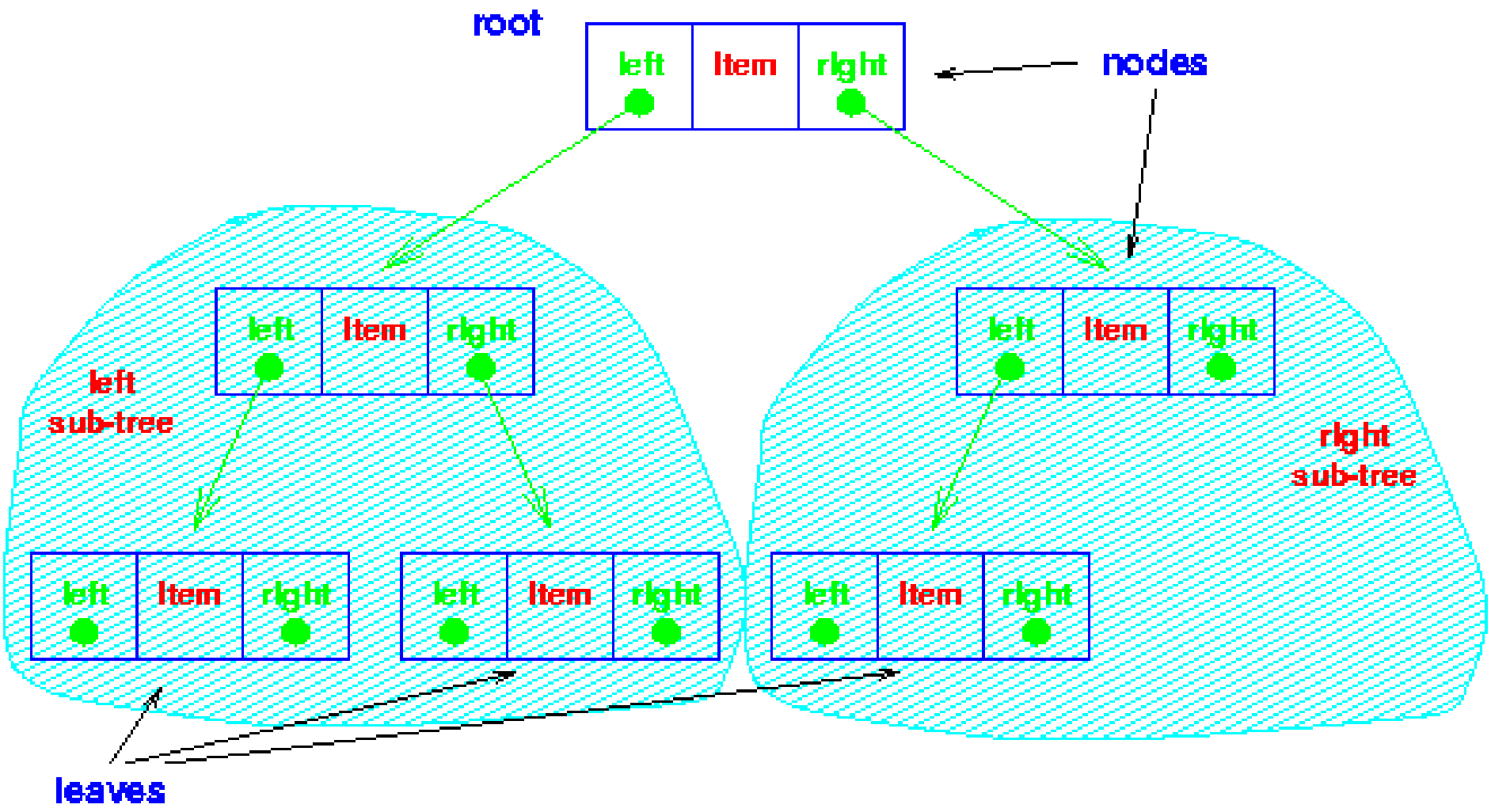
```
enOrden (()) = []
enOrden ((izq,a,der))
    = enOrden(izq) ++ (a . enOrden(der))

    = enOrden(izq) ++ [a] ++ enOrden(der)
```

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)



Recorridas

Formularemos ahora los 3 métodos de recorrida de árbol previamente presentados como procedimientos C++.

Usaremos un parámetro explícito t, que denotará el árbol sobre el cual se efectuará la recorrida.

Usaremos también un parámetro implícito P, que denotará la operación a ser aplicada sobre los elementos del árbol.

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

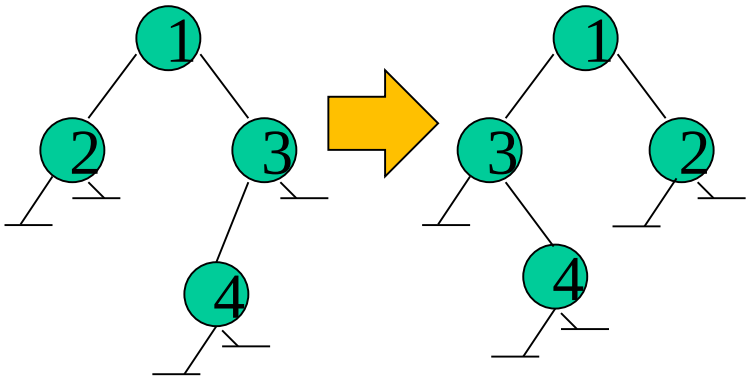
```
cantNodos (()) = 0
cantNodos ((izq,a,der)) =
      1 + cantNodos(izq) + cantNodos(der)
```

```
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

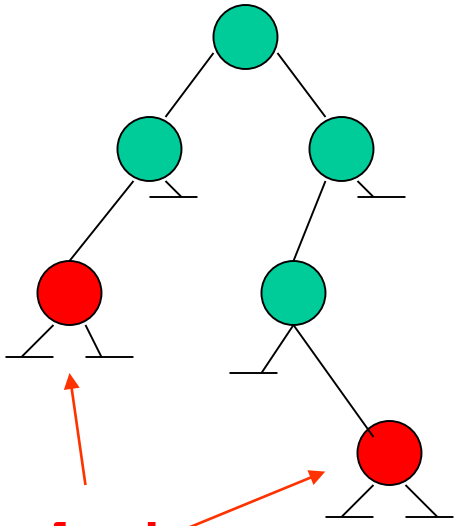
```
espejo (()) = ()
espejo ((izq,a,der)) =
  (espejo(der),a,espejo(izq))
```




```
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?

Hojas de un árbol




```
hojas (()) = []

hojas ((),a,())) = [a]

hojas ((izq,a,der)) =
                  hojas(izq) ++ hojas(der),
                  Si izq≠() o der≠()
```

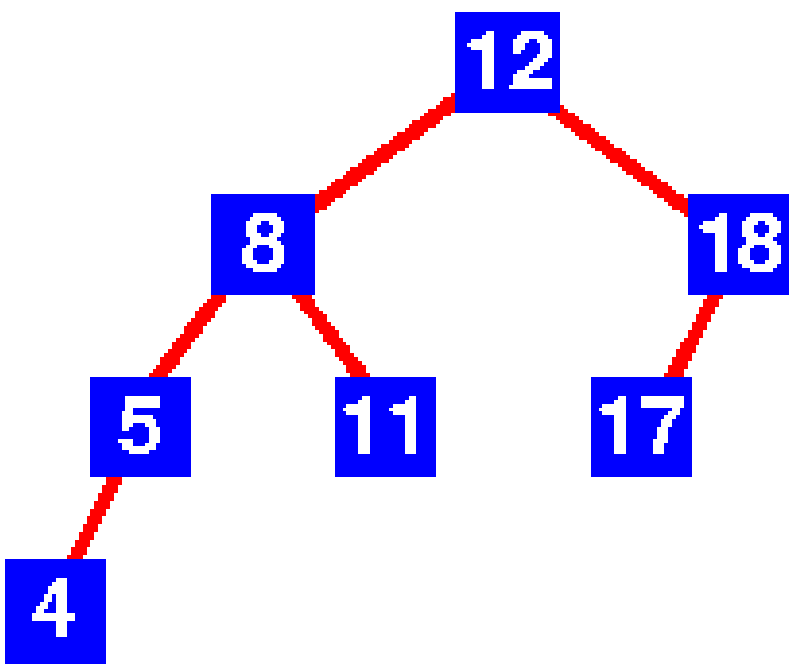
```
Lista hojas (AB t){
 Lista p;
 if (t == NULL) return NULL;
 else
    if ((t -> left == NULL)
         && (t -> right == NULL))
        { p = new nodoLista;
          p -> info = t -> item;
          p -> sig  = NULL;
          return p;
        }
    else return Concat(hojas(t -> left),
                       hojas(t -> right));
} // Nota: Concat es aquí una función (no un proced.)
```

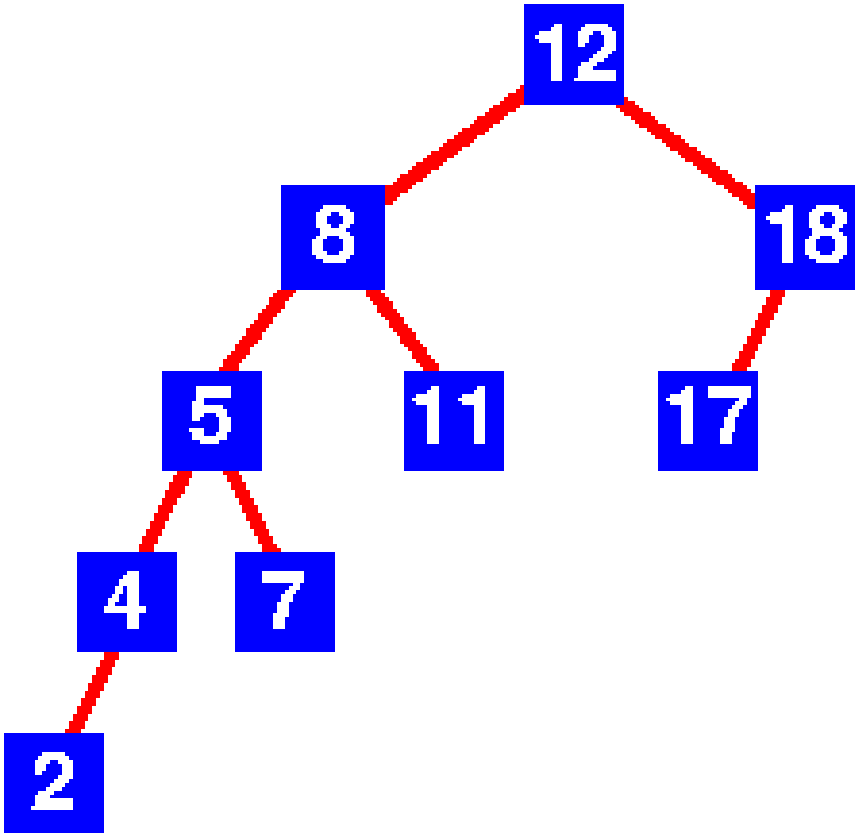
Arbol binario de búsqueda (ABB)

Arboles binarios son frecuentemente usados para representar conjuntos de datos cuyos elementos pueden ser recuperables (y por lo tanto identificables) por medio de una clave única.

Si un árbol está organizado de forma tal que para cada nodo n_i, todas las claves en el subárbol izquierdo de n_i son menores que la clave de n_i, y todas las claves en el subárbol derecho de n_i son mayores que la clave de n_i, entonces este árbol es un *ABB*.

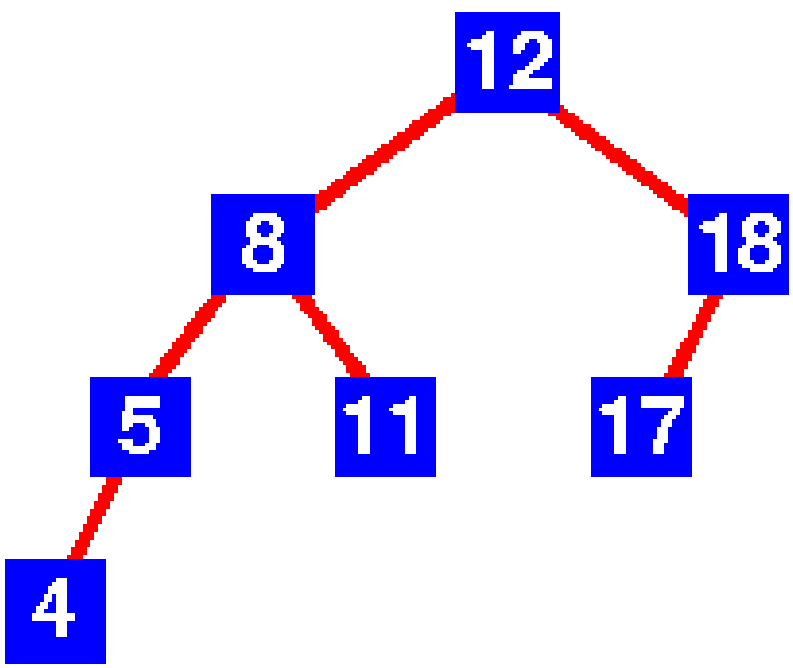
Arbol binario de búsqueda (ABB): Ejemplos

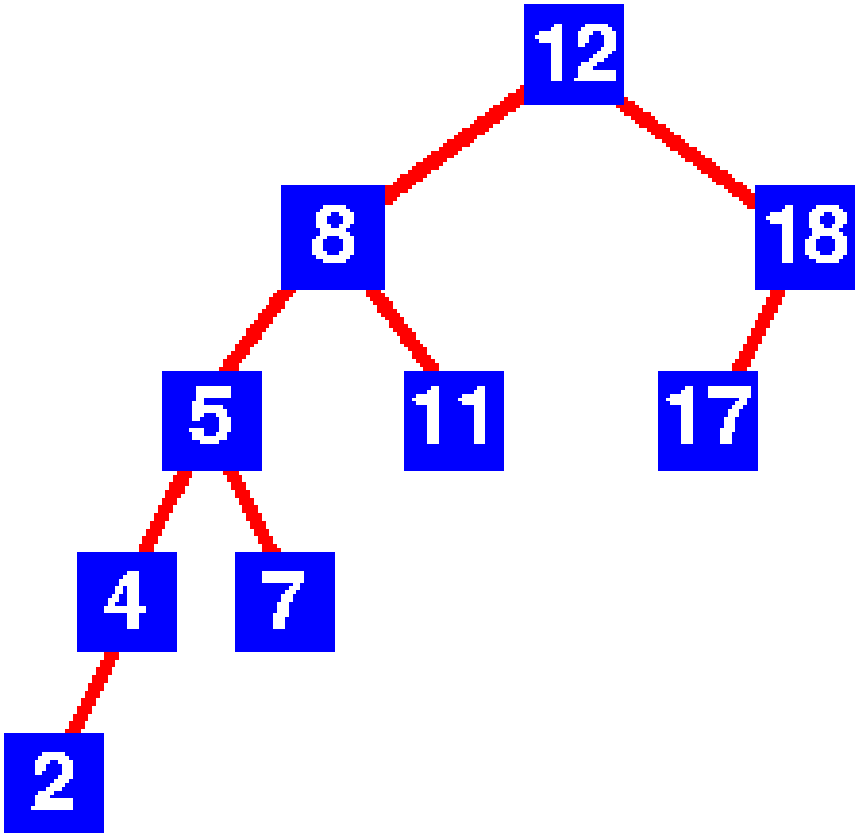




Analizar la relación con la búsqueda binaria sobre un arreglo ordenado (y sobre una lista).

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;
};
```

Búsqueda binaria

En un ABB es posible localizar (encontrar) un nodo de clave arbitraria comenzando desde la raíz del árbol y recorriendo un camino de búsqueda orientado hacia el subárbol izquierdo o derecho de cada nodo del camino dependiendo solamente del valor de la clave del nodo.

Como esta búsqueda sigue un único camino desde la raíz, puede ser fácilmente implementada en forma iterativa, como veremos a continuación:

Buscar iterativo

```
ABB buscarIterativo(Ord x, ABB t){
  bool esta;
  esta = false;
  while ((t != NULL) && !esta){
    if (t -> key == x)
       esta = true;
    else
        if (t -> key > x)
           t = t -> left;
        else t = t -> right;
  }
  return t;
} // ¿hace falta la variable "esta"?
```

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;
} // Sin la variable "esta" de la diapositiva previa
```

Buscar recursivo

```
ABB buscarRecursivo(Ord x, ABB t){
  ABB res;
  if (t == NULL)
   res = NULL;
  else
   if (x == t->key)
     res = t;
   else
     if (x < t->key)
           res = buscarRecursivo (x,t->left);
     else
           res = buscarRecursivo (x,t->right);
  return res;
}
```

¿Usaría esta versión recursiva?

Pertenencia

Hemos dicho que ABB se usan para representar conjuntos, o colección de elementos que son identificados únicamente por su clave.

Otra de las funciones características definidas sobre este tipo de árboles es la función que determina si existe un nodo del árbol cuya clave sea igual a una arbitraria dada (si el elemento pertenece al conjunto).

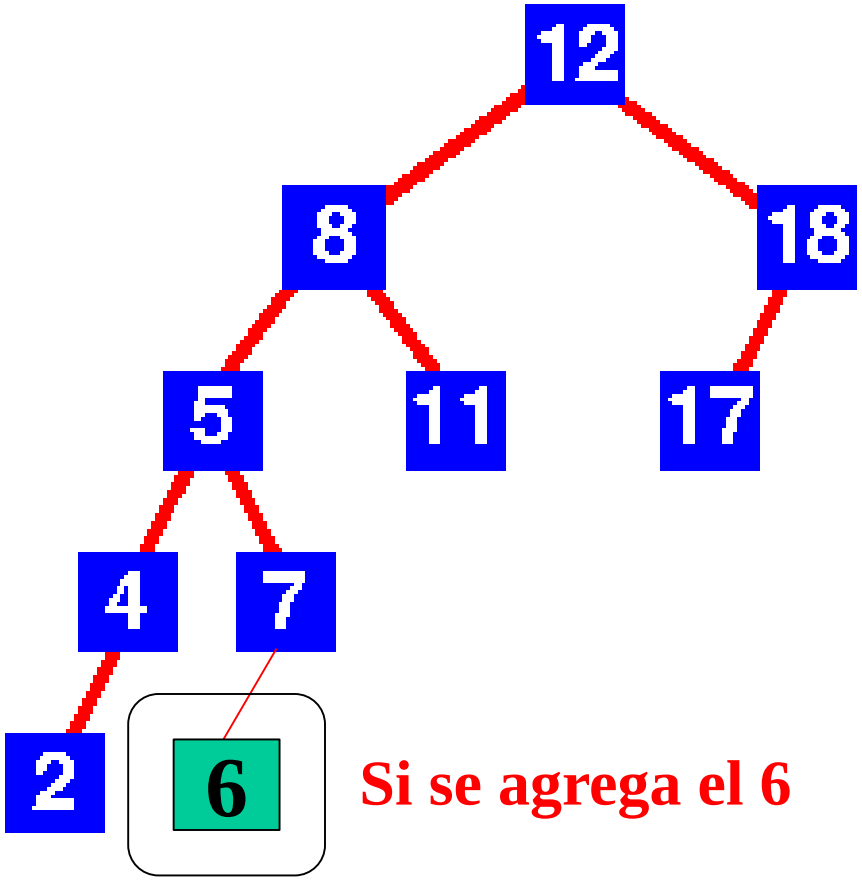
La función Miembro

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

¿Es eficiente la función *miembro*?

¿Podría optimizarse?

Inserción en un ABB

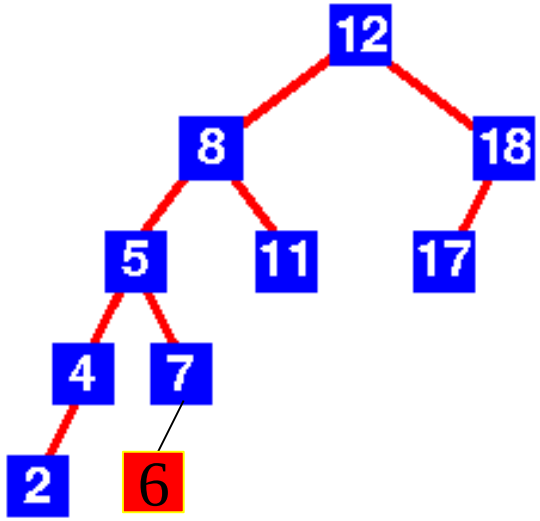


¿Siempre se agrega (eventualmente) un elemento como una hoja?

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

Inserción en un ABB

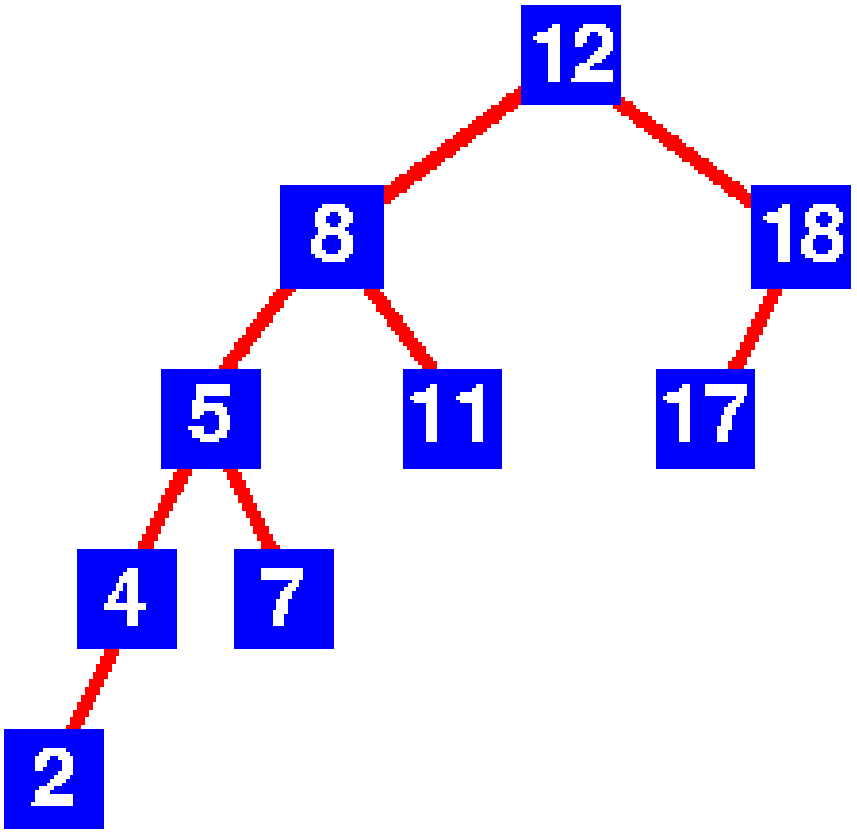
```
void insABB (Ord clave, T dato, ABB & t){
  if (t == NULL){
     t = new NodoABB ;
     t->key = clave;
     t->info = dato;
     t->left = t->right = NULL;
  }
  else if (clave < t->key)
      insABB (clave, dato, t->left);
  else if (clave > t->key)
      insABB (clave, dato, t->right);
}
```



Eliminación en un ABB

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

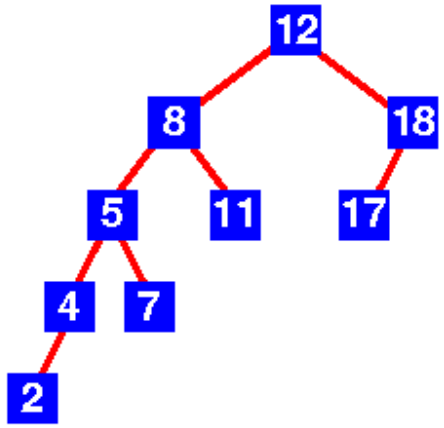
```
void elimABB (Ord clave, ABB & t){ ... }
```



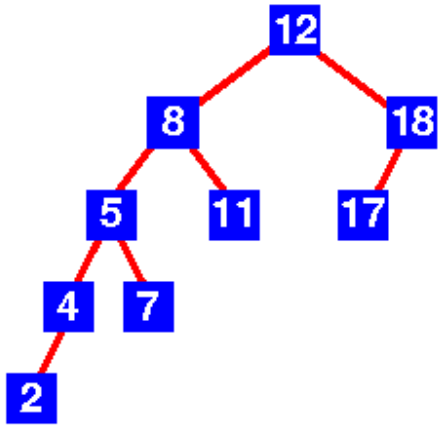
Eliminación en un ABB

Completar los casos

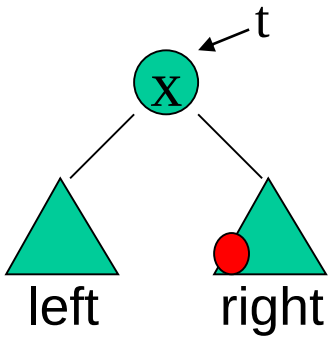
```
void elimABB (Ord clave, ABB & t){
   if (t!=NULL){
        if (clave < t->key)
              elimABB(…);
        else if (clave > t->key)
              elimABB(…);
        else { \\ clave == t->key
              if (t->right == NULL){
              …
              }
              else … // ver la diapositiva siguiente
```



Eliminación en un ABB (cont.)




```
            else if (t->left == NULL){
            …
            }
            else {
                  ABB min_t_der = minimo(t->right);
                  t->key = …
                  t->info = …
                  elimABB(…);
            }
      }
   }
}
```




```
\\ retorna un puntero al nodo con el mínimo de t; NULL si es vacío.
ABB minimo (ABB t)
```

Sobre el orden de tiempo de ejecución del peor caso y del caso promedio en ABBs

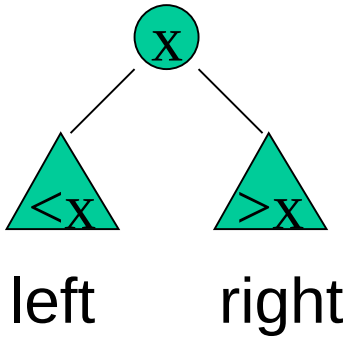
Sobre un ABB de *n* nodos:

- Si un algoritmo recorre todo el árbol, haciendo cosas de tiempo constante, tiene O(*n*) para el peor caso.
- En particular, si un algoritmo recorre solo un camino del árbol, su orden es *n* en el peor caso (árbol degenerado; como una lista).
- No obstante, si un algoritmo recorre solo un camino del árbol, su orden es log(*n*) en el caso promedio (asumiendo que todos los árboles son igualmente probables).

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 *aplanar* tenga O(*n*) peor caso, siendo *n* la cantidad de elementos del ABB:

```
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 de la diapositiva 17.

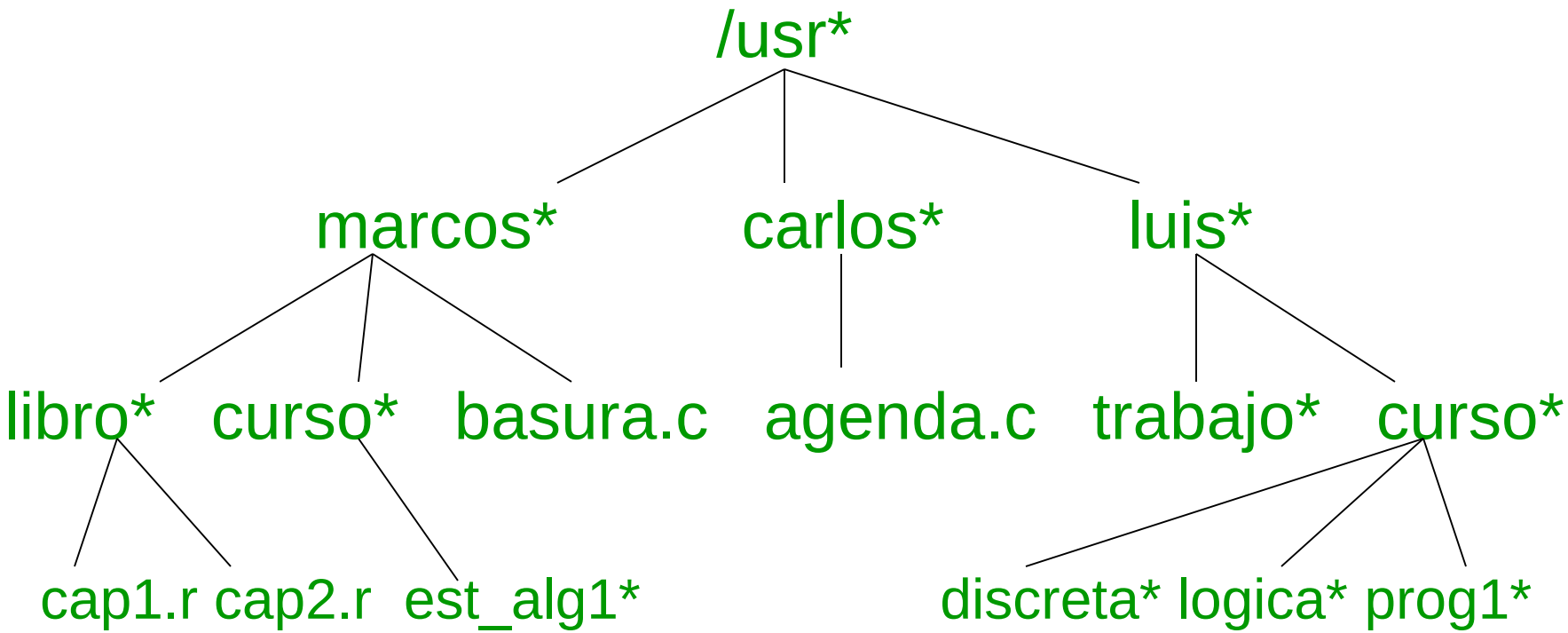
Arboles Generales.
Cómo implementarlos?

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

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

¿cómo representamos árboles generales?

Un ejemplo de árbol general: “estructura de directorios”



Una aplicación: listar un directorio (recorridos de árboles)

¿Cómo representamos árboles generales?

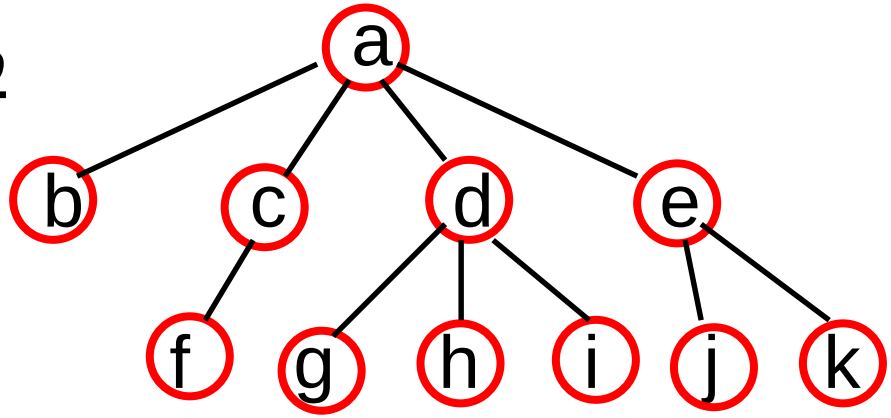
En un árbol general (o finitario) el número de hijos por nodo puede variar. Luego, una idea de representación consiste en pensar que cada nodo tiene una "**lista**" de árboles asociados (sus subárboles).

Una manera de hacer esto es considerando que cada nodo se relaciona con su "primer hijo (pH)" y con su "siguiente hermano (sH)", conformando una estructura de árbol binario.

Esta forma de representación establece una **equivalencia entre árboles generales y árboles binarios**: todo árbol general puede representarse como uno binario y, todo árbol binario corresponde a un determinado árbol general.

Arbol general - Arbol binario

Cada nodo tiene un núnero *finitio* de subárboles

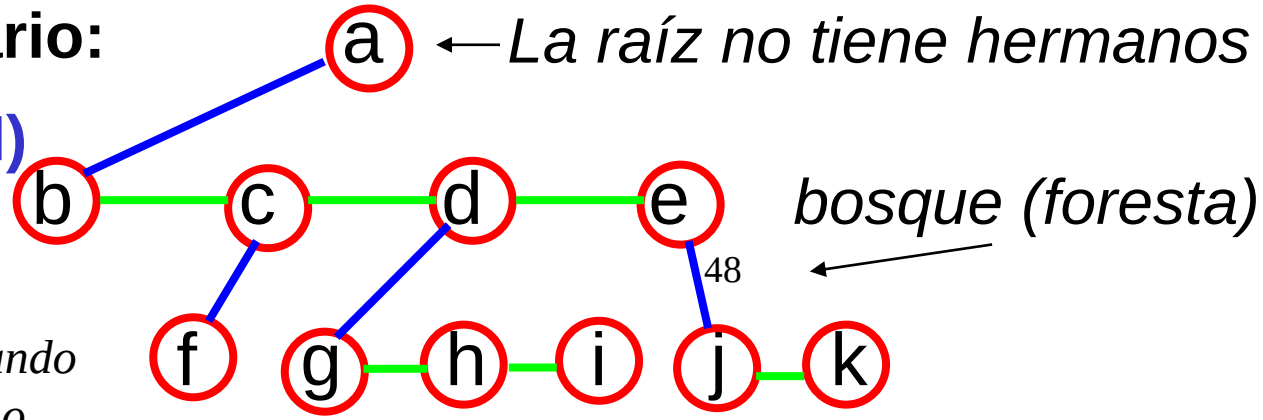


. . . . . .

Representación como árbol binario:

*** primer hijo (pH)**

*** siguiente hermano (sH)**



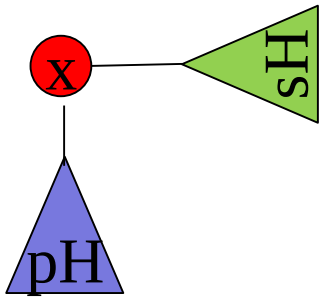
. . . . . .

Notar que estamos representando tanto bosques de árboles como árboles (bosques de un sólo árbol; la raíz).

Ejemplo 1

- La función contar nodos de un árbol general (recordar que la raíz no tiene hermanos).

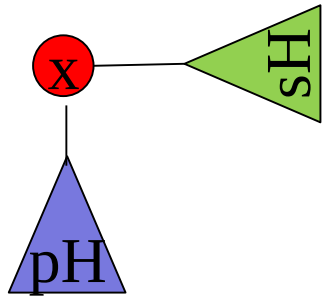
```
typedef NodoAG* AG;
struct NodoAG { T item; AG pH; AG sH; };

int nodos (AG t){
   if (t == NULL) return 0;
   else return nodos(t->pH)+nodos(t->sH)+1;
}
```



Notar que:

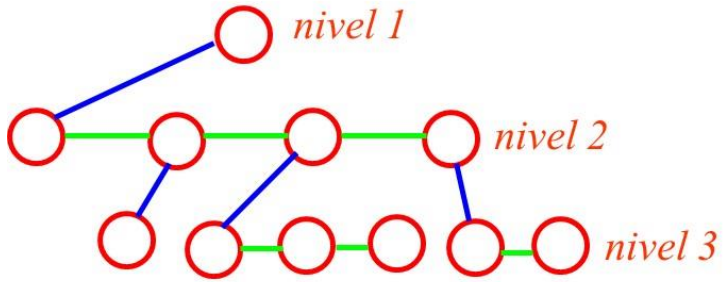
- **el código es idéntico al de contar nodos en un árbol binario tradicional.**
- **si se invoca a *nodos* con un bosque *t* (*t->sH!=NULL*), se tiene la cantidad de nodos del bosque.**

Ejemplo 2



- La función altura de un árbol general (recordar que la raíz no tiene hermanos).

```
int altura (AG t){
   if (t == NULL) return 0;
   else return MAX(1+altura(t->pH),
                   altura(t->sH));
}
```



Notar que el código NO es idéntico al de la altura de un árbol binario tradicional:

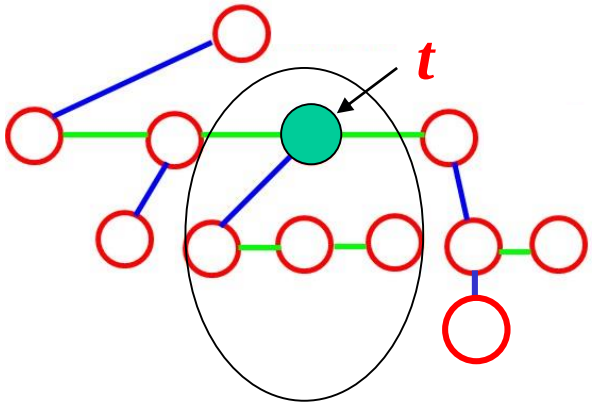
- "primer hijo" (**pH**) aumenta la altura.
- "siguiente hermano" (**sH**) no aumenta la altura, la mantiene (ver dibujo).

Si se invoca a la función *altura* con un bosque *t* (*t->sH!=NULL*), se tiene la altura del bosque.

Ejemplo 2 – Altura de un (sub)árbol

Para calcular la altura de un (sub)árbol con t como raíz en la representación pH-sH podemos hacer:

```
int alturaArbol(AG t){
   if (t == NULL) return 0;
   else return 1 + altura(t->pH);
}
```



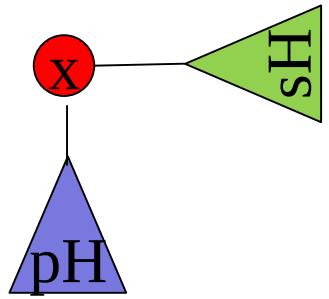
Notar que en el ejemplo:

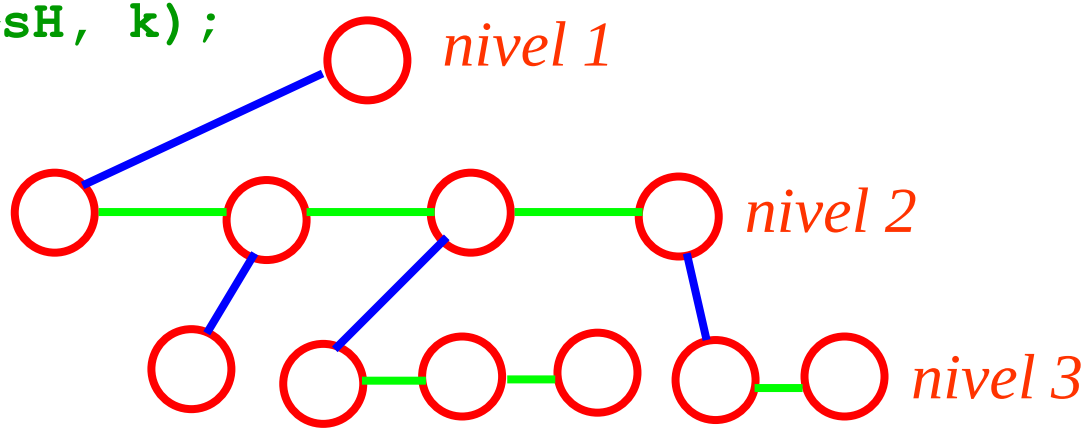
- alturaArbol(t) retorna 2.
- altura(t) retorna 3.

Ejemplo 3

- Imprime los elementos en el nivel k de un árbol general, asumiendo que la raíz de un árbol general no vacío está en el nivel 1 (recordar que la raíz no tiene hermanos).

```
void impNivel(AG t, int k){
   if (t != NULL && k>0){
       if (k == 1) cout << t->item;
           else impNivel(t->pH, k-1);
       impNivel(t->sH, k);
   }
}
```

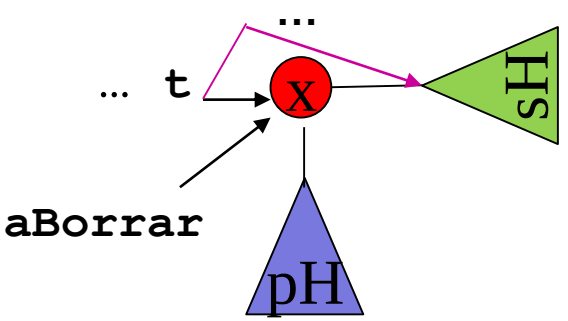




Ejemplo 4

- Elimina cada subárbol que tiene a x como raíz de un árbol general (recordar que en la representación *pH-sH*, el nodo raíz principal no tiene hermanos).

```
void elim(AG & t, T x){
   if (t != NULL){
        if (t->item == x){
                AG aBorrar = t;
                t = t->sH;
                elimTodo(aBorrar->pH);
                delete aBorrar;
                elim(t, x);
        }else{ elim(t->pH, x);
                elim(t->SH, x);
        }
   }
}
```




```
void elimTodo(AG & t){
  if (t != NULL){
    elimTodo(t->pH);
    elimTodo(t->sH);
    delete t;
    t = NULL;
  }
}
```

Arboles TRIE

Aplicación: representación de conjuntos grandes de palabras. Muchas palabras $\rightarrow$ Mucha memoria y operaciones lentas.

Idea: muchas palabras tienen prefijos comunes. Por ejemplo: operador, operando; encontrado, -a,

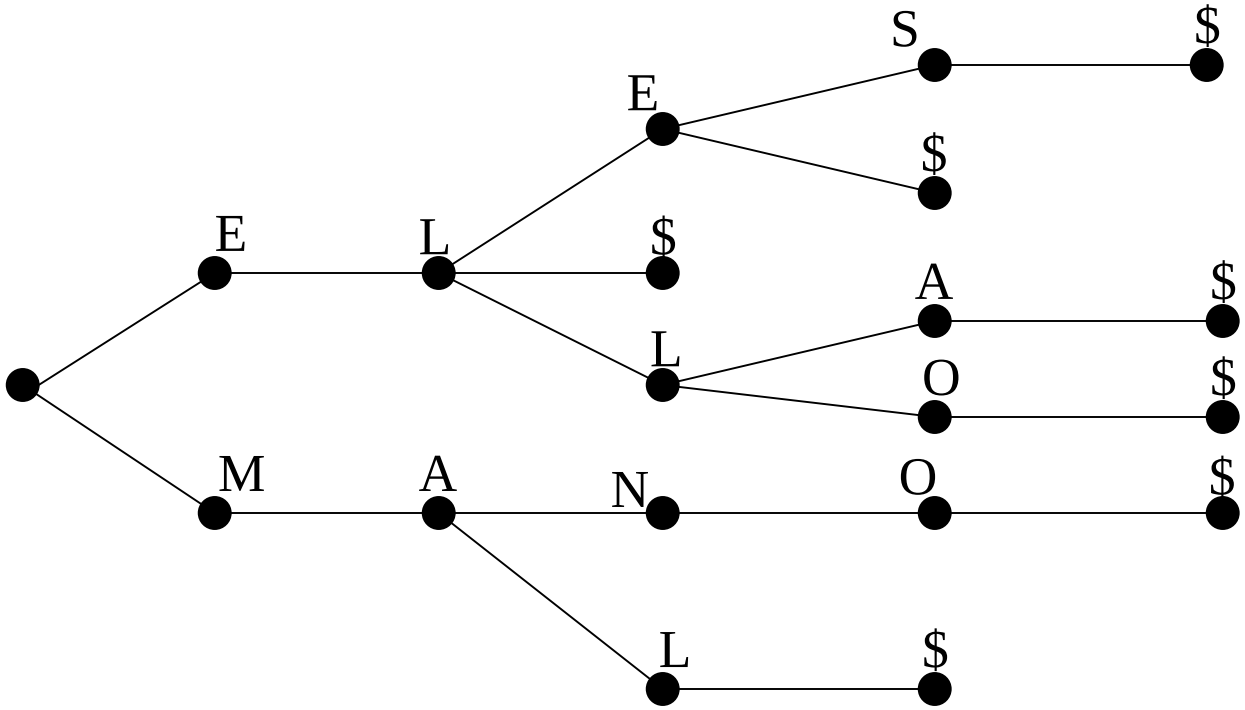
-os, -as...

Definición: un ***TRIE*** es una estructura de árbol en la que:

- Cada nodo (excepto la raíz) está etiquetado con un carácter (a, ..., z) o una marca de fin (símbolo $).
- Un camino de la raíz a una hoja (etiquetada con $) corresponde a una palabra del diccionario.
- Cada nodo (excepto la raíz y las hojas) es un prefijo del conjunto.

Arboles TRIE: ejemplo

Ejemplo: {EL, ELLA, ELLO, ELE, ELES, MANO, MAL}



Arboles TRIE: usos

- Esta estructura es adecuada cuando muchas palabras comparten prefijos comunes: Nº de prefijos << suma de longitudes de las palabras.
- Un nodo puede tener hasta 27 hijos (caracteres+marca de fin), aunque generalmente tendrá muchos menos.
- En general se pueden representar otros datos: cadenas de enteros, de reales, palabras, ...
- Ejemplo de uso de tries: corrector ortográfico.
  - Separar las palabras de un texto.
  - Buscarlas en un diccionario (conjunto).
  - Si no se encuentran mostrarlas por pantalla.

Ejercicios Propuestos

Ejercicio 1)

- Defina una estructura **ABChar** que represente el tipo de datos árboles binarios de caracteres.
- Defina una estructura **ListaChar** que represente el tipo de datos listas no acotadas de caracteres.
- Defina una función que, dado una árbol de tipo **ABChar**, retorne una lista de tipo ListaChar que contenga los elementos del árbol ordenados según su recorrido en Pre-orden.

Ejercicios Propuestos

Ejercicio 2)

Definir, usando punteros, el tipo **IntAB** de los árboles binarios de elementos de tipo **int**.

a) Escribir un subprograma recursivo, que dada la variable *ab* de tipo **IntAB** y un entero *i* retorna el árbol binario resultado de multiplicar cada uno de los elementos de *ab* por *i*.

b) Escribir un subprograma iterativo que dado el árbol binario de búsqueda *abb* de tipo **IntAB** y un entero *i*, retorna el nivel del árbol en el que se encuentra el elemento *i*. Si *i* no pertenece al árbol la función retornará el valor 0.

Ejercicios Propuestos

Ejercicio 3)

- Defina una estructura **ABBInt** que represente el tipo de datos árboles binarios de búsqueda de enteros.
- Defina una estructura **ListaInt** que represente el tipo de datos listas no acotadas de enteros.
- Defina una función que dados una árbol de tipo **ABBInt** y un entero n, retorne una lista de tipo **ListaInt** que contenga el camino en el árbol desde el elemento n (en la primera posición de la lista) hasta la raíz del árbol (en la última posición de la lista), si n está en el árbol. En caso contrario retorna la lista vacía.

Ejercicios Propuestos

Ejercicio 4)

- Defina una estructura **ABChar** que represente el tipo de datos árboles binarios de caracteres.
- Defina una estructura **ListaChar** que represente el tipo de datos listas no acotadas de caracteres.
- Defina una función que dados una árbol de tipo **ABChar** y un entero n, retorne una lista de tipo ListaChar que contenga los elementos del árbol que estén en un nivel menor a n. Recordar que el nivel de la raíz de un árbol (no vacío) es 1.

Ejercicios Propuestos

Ejercicio 5)

- Defina un procedimiento C++ **Insert** que dados un elemento t:T, su clave x:Ord y un ABB, inserte el elemento en el árbol, generando un ABB.
- Defina un procedimiento C++ **DeleteMin** que elimine el mínimo elemento de un ABB. El resultado debe ser un ABB.
- Piense un procedimiento C++ **Delete** que elimine un elemento dado de un ABB. El resultado debe ser un ABB. (Difícil)

Ejercicios Propuestos

Ejercicio 6)

Considere la definición de nodos de árboles binarios de enteros en memoria dinámica.

Defina un procedimiento *print_nivel* en C/C++ que dados un árbol binario de enteros y un entero positivo *i*, imprima todos los elementos del árbol que se encuentran en el nivel *i*. Recordar que la raíz en un árbol binario no vacío se encuentra en el nivel 1. Si el árbol es vacío o *i* es mayor que la cantidad de niveles del árbol, el procedimiento no debe imprimir nada.

Ejercicios Propuestos

Ejercicio 7)

Defina una función recursiva *MaxAB* en C/C++ que dado un árbol binario de enteros, retorne el máximo elemento del árbol. Asumimos como precondición de *MaxAB* que el árbol parámetro es *no vacío*.

Ejercicios Propuestos

Ejercicio 8)

Defina una función *CopiaAcotada* en C/C++ que dados un árbol binario de enteros y un entero no negativo *k*, retorne una copia completamente nueva del árbol parámetro, de profundidad a lo sumo *k*, que contenga la misma estructura y los mismos nodos hasta el nivel *k*. Si *k* es 0 *CopiaAcotada* retorna el árbol vacío y si *k* es mayor que la profundidad del árbol retorna una copia completamente nueva del árbol parámetro.

Ejercicios Propuestos

Ejercicio 9)

Considere la declaración, en C++, del tipo de los nodos de un árbol binario de enteros y la definición de la función f que siguen:

```
int f (NodoABPtr t, int x, int c) {
        if (t==NULL) return (c==0);
        else if (x==t->info) return f(t->izq,x,c-1);
                else return f(t->der,x,c); };
```

Ejercicios Propuestos

- ¿Qué retorna la función f ?
- Sea T(n) la cantidad de nodos visitados por la función f cuando recibe como parámetro un árbol binario de n nodos. Supongamos que árbol binario pasado como parámetro es aquel que hace a la función f visitar la mayor cantidad de nodos (“peor caso”). Plantear una recurrencia para T(n) en esta situación y resolverla.
- ¿Cuál es el orden de tiempo de ejecución de f en el peor caso?
- Consideremos ahora que se pasa como parámetro a la función f un árbol binario completo de altura h con $n=2^{h+1}-1$ nodos, con $h \geq 0$. ¿Cuál es el orden de tiempo de ejecución de f en el caso de árboles con estas características?. Justifique.

Ejercicios Propuestos

Ejercicio 10)

Defina un procedimiento *poda* en C/C++ que, dado un árbol binario de enteros, elimine todos sus nodos hojas.

Ejercicios Propuestos

Ejercicio 11)

Considere la declaración, en C++, del tipo de los nodos de un árbol binario de enteros y la definición de los nodos de una lista de enteros que siguen:

```
struct ABNode { int info; ABNode *left, *right; };
struct ListNode { int info; ListNode *sig; };
```

Se pide:

- Definir una función que dado un árbol binario de enteros devuelva en una lista el camino más largo entre la raíz y una hoja. En caso de haber más de un camino de igual longitud a la del camino más largo, retorna cualquiera de ellos. La lista que contiene al camino devuelto empieza con el dato de la raíz del árbol y termina con el dato de la hoja del árbol, siempre que el Árbol no sea vacío.
- Indicar el orden de tiempo de ejecución en el peor caso de su función. Justificar.

Ejercicios Propuestos

Ejercicio 12)

Considerar la declaración, en C++, del tipo de los nodos de un árbol binario de búsqueda de enteros:

Se pide:

- Definir una función F que dado un árbol binario de búsqueda de enteros retorne true si, y solamente si, el árbol tiene 2^{h+1}-1 nodos, con h la altura del árbol. La altura de un árbol vacío es –1 y de un árbol hoja es 0.
- Si A es un árbol binario de búsqueda de enteros tal que F(A) es true, ¿qué característica estructural tiene A?.
- ¿Cuál es el orden de tiempo de ejecución de F en el peor caso?. Justificar.

Ejercicios Propuestos

Ejercicio 13)

Definir una función que dados dos árboles binarios de búsqueda de enteros retorne true si, y solamente si, ambos árboles representan a un mismo conjunto. ¿Cuál es el tiempo de ejecución en el caso promedio y el orden del peor caso para su algoritmo?. ¿Podrían reducirse ambos estimativos?

Ejercicios Propuestos

Ejercicio 14)

Definir una función que dado un árbol binario de búsqueda de enteros y un entero k, retorne la lista de los elementos que se encuentran en el nivel k, ordenados de menor a mayor. La raíz de un árbol no vacío se encuentra en el nivel 1. Si no existe el nivel k, la lista a retornar debe ser vacía.

Ejercicio 15)

Considere la definición de la función f que sigue:

```
bool f (ABBNode *t, int k) {
        if (t==NULL) return (k==0);
        else return f(t->left,k-1) && f(t->right,k-1); };
```

¿Qué retorna la función f ?. ¿Cuántos nodos tiene el árbol si la función retorna true?. Justifique.

¿Cuál es el orden de tiempo de ejecución de f en el peor caso?

Ejercicios Propuestos

Ejercicio 16)

Considere la declaración, en C++, del tipo de los nodos de un árbol binario de enteros:

Se pide:

a) Defina una función *EsABB* que dado un árbol binario de enteros retorne true si y sólo si el árbol es binario de búsqueda.

b) Indique el orden de tiempo de ejecución en el peor caso de la función *EsABB*. Justifique.

Ejercicios Propuestos

Ejercicio 17)

Considere las declaraciones, en C++, del tipo de los nodos de un árbol binario de enteros y de una lista de enteros:

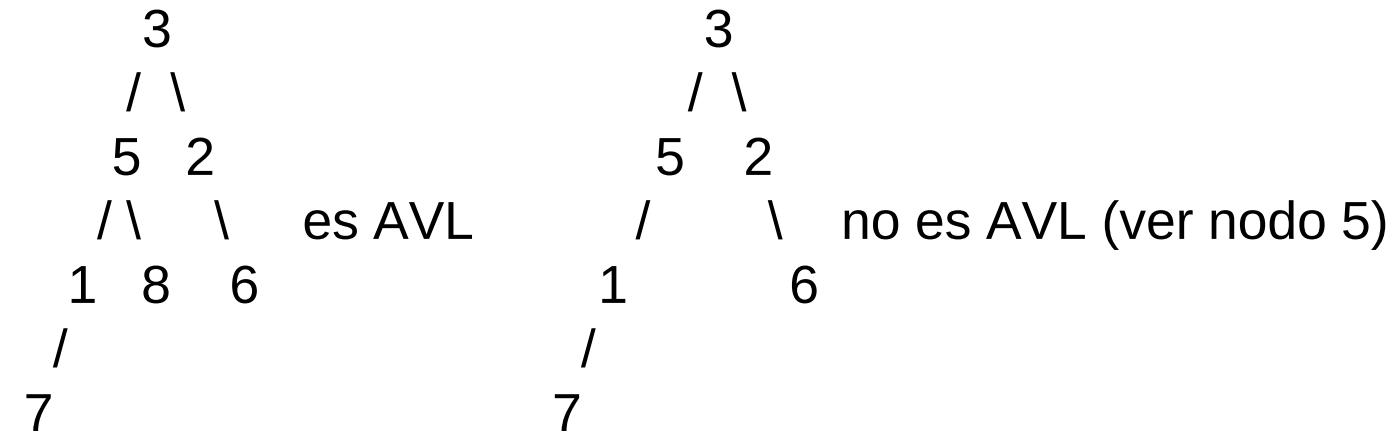
Defina una función *`EsCamino`* que dado un árbol binario de enteros y una lista de enteros con nodos de tipo *`ListNode`* retorne true si y sólo si la lista es un camino del árbol binario (desde la raíz a una hoja). La lista vacía es un camino (el único) del árbol vacío.

Ejercicios Propuestos

Ejercicio 18)

Considerar la declaración, en C++, del tipo de los nodos de un árbol binario de búsqueda de enteros.

Un árbol binario de búsqueda es un AVL si para coda nodo del árbol se cumple que las alturas de sus subárboles izquierdo y derecho difieren a lo sumo en uno. El árbol vacío es un AVL. Definir una función que dado un árbol binario de búsqueda de enteros retorne true si, y solamente si, el árbol es un AVL.

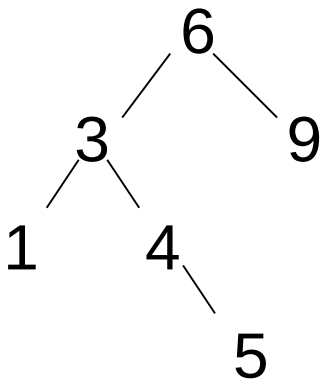


Ejercicios Propuestos

Ejercicio 19)

Escribir un procedimiento caminos en C++ que dado un árbol binario de búsqueda de números enteros, imprima todos los caminos desde la raíz hacia una hoja.

Por ejemplo, para:



imprime: 6, 3, 1

6, 3, 4, 5

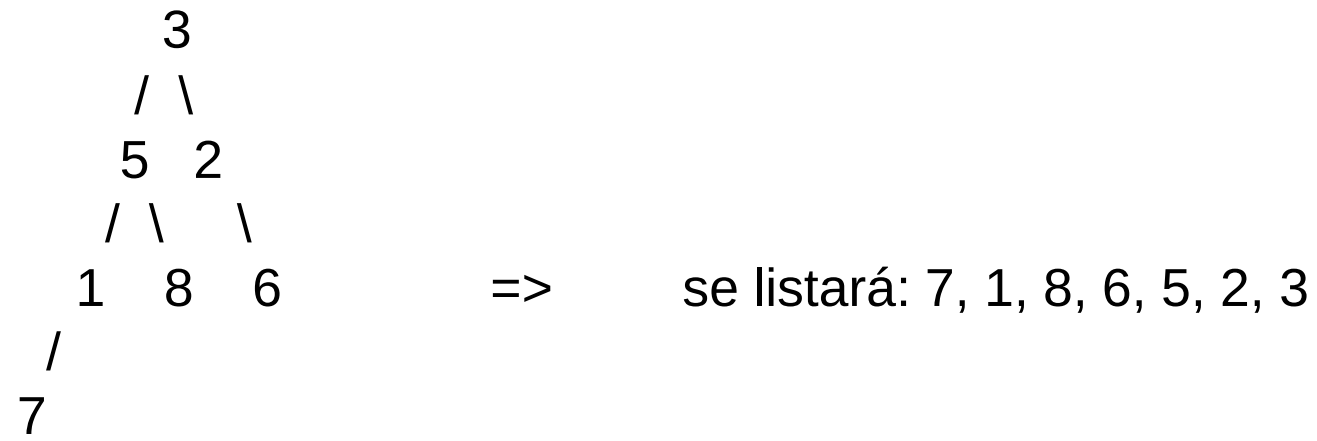
6, 9

Ejercicios Propuestos

Ejercicio 20)

Considere la declaración, en C++, del tipo de los nodos de un árbol binario de enteros:

Escribir un procedimiento que imprima los nodos de un árbol binario de enteros recorriéndolo por niveles, como sigue: los nodos de un mismo nivel deben aparecer listados de izquierda a derecha y después que los nodos de los niveles inferiores. Esto es, de abajo hacia arriba, por niveles y de izquierda a derecha. Ejemplo: el árbol



Bibliografía Recomendada

Como Programar en C/C++

H.M. Deitel & P.J. Deitel; Prentice Hall, 1994.

(o la Versión 2 del *Deitel*)

Todos los libros propuestos por la cátedra. *En particular el libro de Weiss*.