4 Árboles

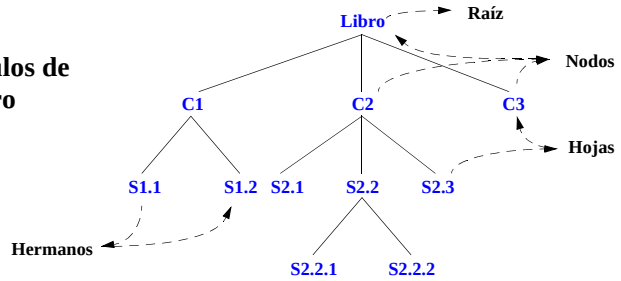
Un árbol es una estructura de datos jerarquizada

Cada dato reside en un nodo, y existen relaciones de parentesco entre nodos:

- **padre, hijo, hermano, ascendiente, descendiente, etc.**

Ejemplo:

Capítulos de un libro



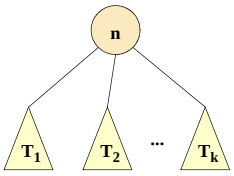
Definición recursiva de los árboles

Un nodo simple *n* constituye un árbol

- **se denomina la *raíz* del árbol**

Supongamos que *n* es un nodo y T_1, T_2, ..., T_k son árboles cuyas raíces son n_1, n_2, ..., n_k, respectivamente.

- **Podemos construir un nuevo árbol haciendo que *n* sea el padre de los nodos n_1, n_2, ..., n_k**
- **En el nuevo árbol *n* es la raíz y n_1, n_2, ..., n_k se denominan los hijos de *n***



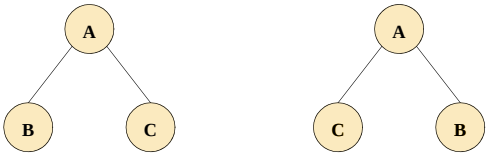
Terminología

- ***Camino***: **secuencia de nodos tales que cada uno es hijo del anterior**
- ***Longitud del camino***: **nº de nodos - 1 (la longitud del camino de un nodo a sí mismo es 0)**
- ***Ascendiente***: **un nodo es ascendiente de otro si hay un camino del primero al segundo**
- ***Descendiente***: **un nodo es descendiente de otro si hay un camino del segundo al primero**
- ***Subárbol*** **o** ***Rama***: **un nodo y todos sus descendientes**
- ***Altura*** **de un árbol: longitud de su camino más largo más 1**
- ***Profundidad*** **de un nodo: longitud del camino desde la raíz a ese nodo**
- ***Árbol n-ario***: **árbol en que cada nodo puede tener como máximo n hijos**

4.1 Ordenación y recorrido

Un árbol se considera ordenado si hay un orden definido para los hijos de cada nodo

En un árbol ordenado, los hijos de un nodo se ordenan de izquierda a derecha



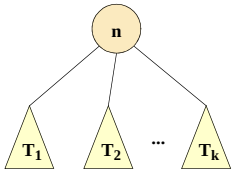
Dos árboles ordenados, distintos

Recorrido de un árbol

El recorrido de un árbol es una forma sistemática de visitar todos sus nodos

El recorrido de los nodos se suele hacer de 3 modos:

- ***Preorden*: la raíz n seguida de los nodos de T_1 en preorden, luego los de T_2 en preorden, ...**
- ***Postorden*: los nodos de T_1 en postorden, luego los de T_2 en postorden, y así hasta la lista de T_k en postorden, finalizando con el nodo raíz n**
- ***Inorden*: los nodos de T_1 en inorden, seguida de la raíz n, luego los subárboles T_2, ..., T_k en inorden**



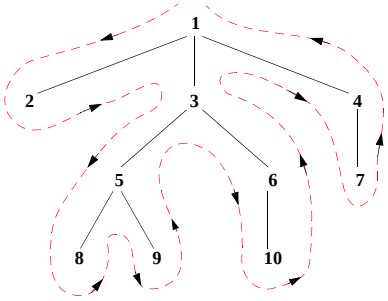
Preorden: n, T_1, T_2, ..., T_k

Postorden: T_1, T_2, ..., T_k, n

Inorden: T_1, n, T_2, ..., T_k

Método para producir las ordenaciones a mano:

- ***Preorden*: se lista cada nodo la primera vez que se pasa por él**
- ***Postorden*: se lista cada nodo la última vez que se pasa por él**
- ***Inorden*: se listan las hojas la primera vez que se pasa por ellas, pero los nodos interiores la segunda**



```
método Preorden (N : Nodo; A : Arbol)
  visita N;
  para cada hijo H de N, y empezando por la izquierda
  hacer
    Preorden(H,A);
  fpara;
fmétodo;
```

```
método Postorden (N : Nodo; A : Arbol)
  para cada hijo H de N, y empezando por la izquierda
  hacer
    Postorden(H,A);
  fpara;
  visita N;
fmétodo;
```

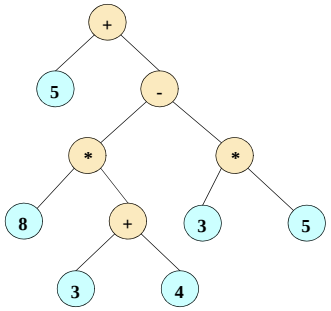


```
método Inorden (N : Nodo; A : Arbol)
  si N es una hoja entonces
    visita N;
  si no
    Inorden(hijo más a la izquierda de N,A);
    visita N;
    para cada hijo H de N, excepto el más a la
        izquierda, y empezando por la izquierda
    hacer
      Inorden(H,A);
    fpara;
  fsi;
fmétodo;
```


Ejemplo de ordenación de expresiones aritméticas

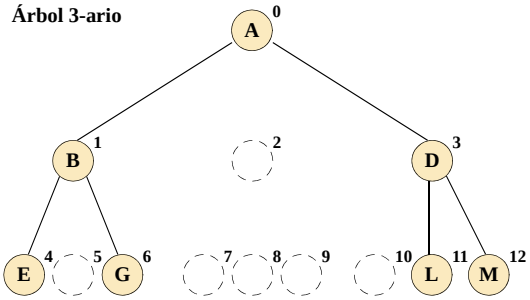
Expresión: 5+8*(3+4)-3*5:

- ***preorden*: +5-*8+3,4*3,5**
- ***inorden*: 5+(8*(3+4)-(3*5)) es la expresión en notación matemática normal**
- ***postorden*: 5,8,3,4+*3,5*-+ es la expresión en Notación Polaca Inversa (RPN)**



4.2 Implementación de árboles

Implementación con vectores de árboles n-arios

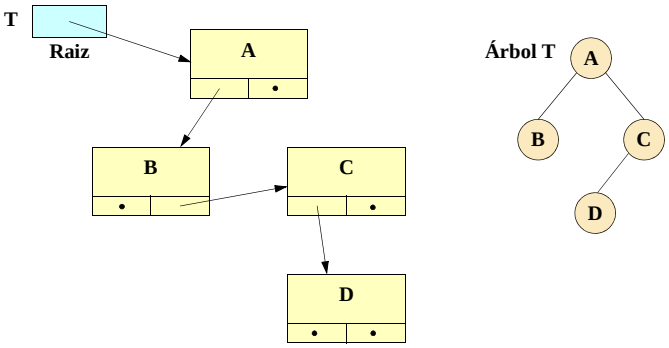


$hijo_i(x)=n*pos(x)+i \ (i=1..n)$

$padre(x)=(pos(x)-1) \ div \ n$

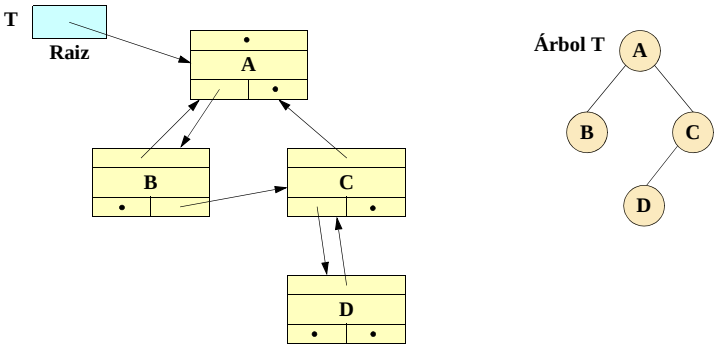
- \+ Navegación por el árbol sencilla
- – Desperdicia memoria cuando el árbol no está lleno
- – Operaciones de unión poco eficientes

Implementación primer-hijo/hermano-derecho



- \+ Facilita las operaciones de tipo unión
- \+ Utiliza la memoria justa
- – Navegación por el árbol (muy) complicada

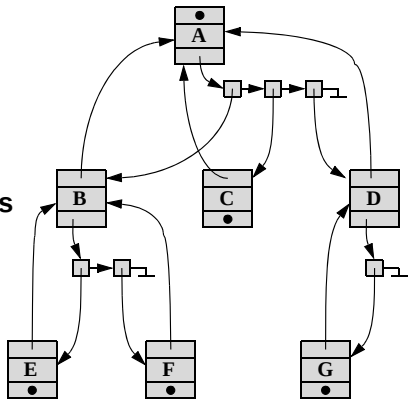
Implementación padre/primer-hijo/hermano-derecho



- \+ Facilita las operaciones de tipo unión
- \+ Utiliza la memoria justa
- – Navegación por el árbol complicada

Implementación padre/lista-hijos

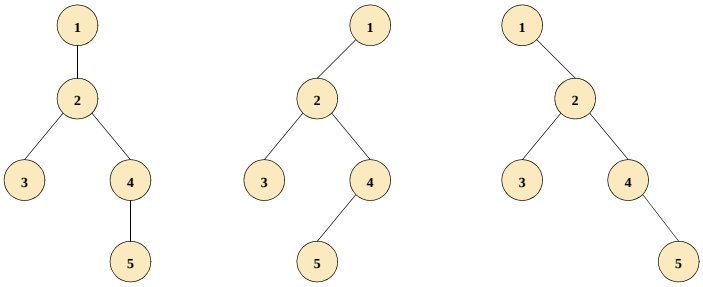
- + Facilita las operaciones de tipo unión
- + Utiliza la memoria justa
- + Navegación por el árbol menos complicada que con otras implementaciones con punteros
- – Estructura más compleja



4.3 Árboles binarios

Un árbol binario: árbol ordenado de aridad 2

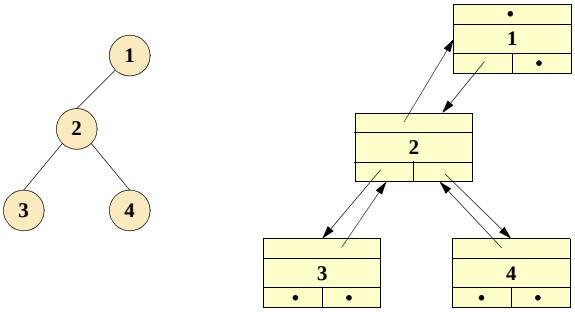
- cada nodo puede tener como máximo dos hijos (izquierdo y derecho)
- en la ordenación de los hijos, el izquierdo precede al derecho



Un árbol ordinario **Dos árboles binarios**

Implementación de árboles binarios

Punteros al padre, al hijo izquierdo, y al hijo derecho:



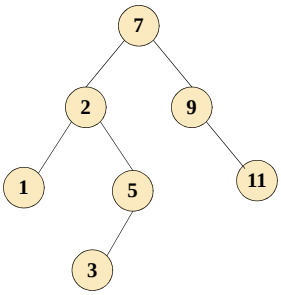
También se puede implementar sólo con punteros a los hijos

- la navegación es un poco más difícil

4.4 Árboles binarios de búsqueda

Un árbol binario de búsqueda es un árbol binario

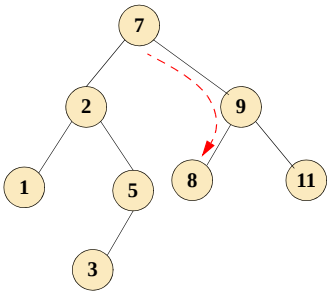
- **en el que entre sus elementos existe una relación de orden total**
- **para cada nodo, todos sus descendientes izquierdos son menores que él, y los derechos mayores**



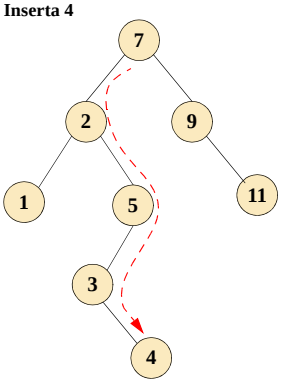
Inserción en un árbol binario de búsqueda

Bajar por el árbol hasta encontrar la posición adecuada

Inserta 8



Inserta 4

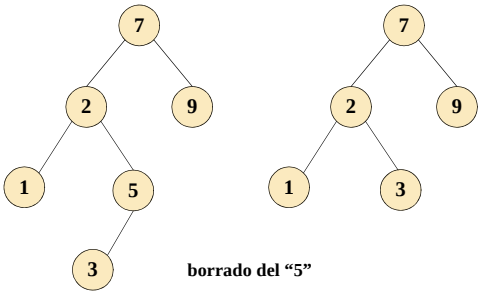


Eliminación en un árbol binario de búsqueda

Se consideren tres casos:

a) Si el nodo es una hoja, se puede borrar sin más

b) Si el nodo tiene un solo hijo se puede eliminar sustituyéndolo en el árbol por ese hijo (con toda su descendencia)

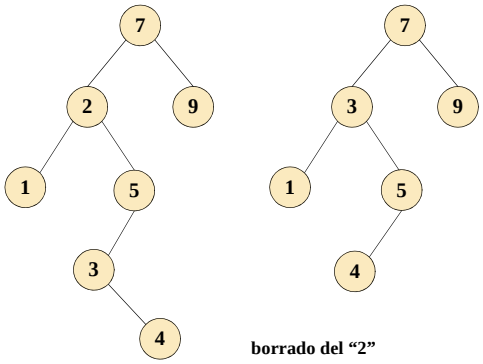


borrado del “5”

Eliminación en un árbol binario de búsqueda (cont.)

c) Si el nodo tiene dos hijos lo reemplazaremos por el menor elemento de su subárbol derecho, que a su vez eliminaremos

- **este menor elemento eliminado no tiene hijo izquierdo, por lo que se puede borrar con a) o b)**



borrado del "2"

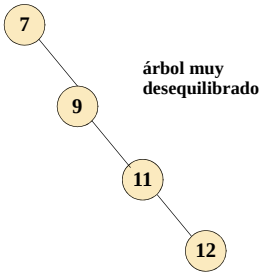
Eficiencia de las operaciones de un árbol binario de búsqueda

Inserción, búsqueda y eliminación: *O*(altura)

- **En promedio, para datos ordenados aleatoriamente, $altura \approx \log_2(n)$ (donde n es el número de nodos del árbol)**
- **luego la *eficiencia promedio* de las operaciones será *O(log n)***

Sin embargo, para entradas parcialmente ordenadas:

- **el árbol puede puede estar muy desequilibrado (altura≈n)**
- **en este caso puede interesar utilizar algoritmos que mantengan el árbol equilibrado**



árbol muy desequilibrado