Tema 9. Árboles

9.1 Árboles generales

9.2 Árboles binarios

9.3 Árboles de búsqueda

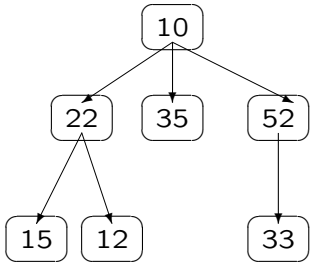
9.1 Árboles generales

✓ Un árbol es una estructura no lineal acíclica utilizada para organizar información de forma eficiente.

✓ La definición es recursiva:

✓ Un árbol es una colección de valores $\{v_1, v_2, \ldots v_n\}$ tales que

- ◇ Si $n = 0$ el árbol se dice vacío.
- ◇ En otro caso, existe un valor destacado que se denomina *raíz* (p.e. v_1), y los demás elementos forman parte de colecciones disjuntas que a su vez son árboles. Estos árboles se llaman subárboles del raíz.



✓ Las estructuras tipo árbol se usan principalmente para representar datos con una relación jerárquica entre sus elementos, como árboles genealógicos, tablas, etc.

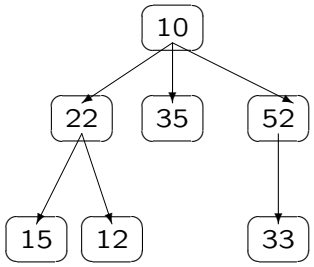
✓ La terminología de los árboles se realiza con las típicas notaciones de las relaciones familiares en los árboles genealógicos: padre, hijo, hermano, ascendente, descendente, etc.

Algunas definiciones

- ✓ Nodo, son los elementos del árbol.
- ✓ Raíz del árbol: todos los árboles que no están vacíos tienen un único nodo raíz. Todos los demás elementos o nodos se derivan o descienden de él.
- ✓ Nodo hoja es aquel nodo que no contiene ningún subárbol.
- ✓ Tamaño de un árbol es su número de nodos.
- ✓ A cada nodo que no es hoja se le asocia uno o varios subárboles llamados descendientes o hijos.
- ✓ De igual forma, cada nodo tiene asociado un antecesor o ascendiente llamado padre.
- ✓ Todos los nodos tienen un solo padre excepto el raíz que no tiene padre.
- ✓ Cada nodo tiene asociado un número de nivel que se determina por la longitud del camino desde el raíz al nodo específico.
- ✓ La altura o profundidad de un árbol es el nivel más profundo más uno.

Ejemplo

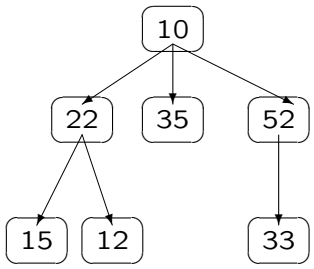
✓ Para ilustrar las definiciones se considera el siguiente árbol general:



✓ Por ejemplo, en la figura:

- ◇ Raíz: 10
- ◇ Nodos: 10, 22, 35, 52, 15, 12, 33
- ◇ Tamaño: 7
- ◇ Nivel 0: 10
- ◇ Nivel 1: 22, 35, 52
- ◇ Nivel 2: 15, 12, 33
- ◇ Altura o profundidad: 3
- ◇ Hojas: 15, 12, 35, 33

Representación en Haskell

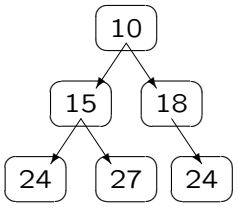


$$\textbf{data}\ \mathit{Árbol}\ a = \mathit{Vacío} \mid \mathit{Nodo}\ \underbrace{a}_{\mathit{raíz}}\ \underbrace{[\mathit{Árbol}\ a]}_{\mathit{hijos}}\ \textbf{deriving}\ \mathit{Show}$$

```
a1 :: Árbol Integer
a1 = Nodo 10 [a11, a12, a13]
  where
    a11 = Nodo 22 [hoja 15, hoja 12]
    a12 = hoja 35
    a13 = Nodo 52 [hoja 33]

hoja :: a → Árbol a
hoja x = Nodo x []

raíz :: Árbol a → a
raíz Vacío = error "raíz de árbol vacío"
raíz (Nodo x _) = x

tamaño :: Árbol a → Integer
tamaño Vacío = 0
tamaño (Nodo _ xs) = 1 + sum (map tamaño xs)

profundidad :: Árbol a → Integer
profundidad Vacío = 0
profundidad (Nodo _ []) = 1
profundidad (Nodo _ xs) = 1 + maximum (map profundidad xs)
```

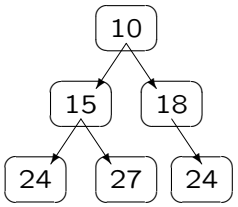
9.2 Árboles binarios

Un árbol binario es árbol tal que cada nodo tiene como máximo dos subárboles.



$$\textbf{data}\ \ \acute{A}rbolB\ a\ =\ \ Vac\acute{i}oB \mid NodoB\ \overbrace{(\acute{A}rbolB\ a)}^{hijo\ izq}\ \overbrace{a}^{dato\ en\ nodo}\ \overbrace{(\acute{A}rbolB\ a)}^{hijo\ der}\ \ \textbf{deriving}\ Show$$

Consideraremos que las tres componentes del constructor $NodoB$ son el subárbol izquierdo, el dato raíz y el subárbol derecho respectivamente.




```
a2 :: ÁrbolB Integer
a2 = NodoB aI 10 aD
  where
    aI  = NodoB aII 15 aID
    aD  = NodoB VacíoB 18 aDD
    aII = hojaB 24
    aID = hojaB 27
    aDD = hojaB 24

hojaB :: a → ÁrbolB a
hojaB x = NodoB VacíoB x VacíoB
```

Árboles binarios (II)

```
raízB                    :: ÁrbolB a → a
raízB VacíoB             =  error "raíz de árbol vacío"
raízB (NodoB _ x _)      =  x

tamañoB                  :: ÁrbolB a → Integer
tamañoB VacíoB           =  0
tamañoB (NodoB i _ d)    =  1 + tamañoB i + tamañoB d

profundidadB             :: ÁrbolB a → Integer
profundidadB VacíoB      =  0
profundidadB (NodoB i _ d) = 1 + max (profundidadB i) (profundidadB d)
```

Recorrido de árboles binarios (I)

- ✓ Se llama recorrido de un árbol al proceso que permite acceder una sola vez a cada uno de los nodos del árbol para examinar el conjunto completo de nodos.

- ✓ Los algoritmos de recorrido de un árbol binario presentan tres tipos de actividades comunes:
  - ◇ visitar el nodo raíz
  - ◇ recorrer el subárbol izquierdo
  - ◇ recorrer el subárbol derecho

- ✓ Estas tres acciones llevadas a cabo en distinto orden proporcionan los distintos recorridos del árbol.

- ✓ Recorrido en PRE-ORDEN:
  - ◇ Visitar el raíz
  - ◇ Recorrer el subárbol izquierdo en pre-orden
  - ◇ Recorrer el subárbol derecho en pre-orden

- ✓ Recorrido EN-ORDEN
  - ◇ Recorrer el subárbol izquierdo en en-orden
  - ◇ Visitar el raíz
  - ◇ Recorrer el subárbol derecho en en-orden

- ✓ Recorrido en POST-ORDEN
  - ◇ Recorrer el subárbol izquierdo en post-orden
  - ◇ Recorrer el subárbol derecho en post-orden
  - ◇ Visitar el raíz

Recorrido de árboles binarios (II)

$$
\begin{array}{lcl}
enOrdenB & :: & ÁrbolB\ a \to [a] \\
enOrdenB\ Vacío B & = & [\,] \\
enOrdenB\ (NodoB\ i\ r\ d) & = & enOrdenB\ i +\!\!+ [r] +\!\!+ enOrdenB\ d
\end{array}
$$

$$
\begin{array}{lcl}
preOrdenB & :: & ÁrbolB\ a \to [a] \\
preOrdenB\ Vacío B & = & [\,] \\
preOrdenB\ (NodoB\ i\ r\ d) & = & [r] +\!\!+ preOrdenB\ i +\!\!+ preOrdenB\ d
\end{array}
$$

$$
\begin{array}{lcl}
postOrdenB & :: & ÁrbolB\ a \to [a] \\
postOrdenB\ Vacío B & = & [\,] \\
postOrdenB\ (NodoB\ i\ r\ d) & = & postOrdenB\ i +\!\!+ postOrdenB\ d +\!\!+ [r]
\end{array}
$$

? $enOrdenB\ a2$
$[24, 15, 27, 10, 18, 24] :: [Integer]$

? $preOrdenB\ a2$
$[10, 15, 24, 27, 18, 24] :: [Integer]$

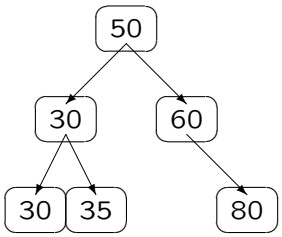
? $postOrdenB\ a2$
$[24, 27, 15, 24, 18, 10] :: [Integer]$

9.3 Árboles de búsqueda

✓ Un árbol de búsqueda es un árbol binario tal que

- ◇ O bien es vacío
- ◇ O no es vacío y para cualquier nodo se cumple que:
  - ○ los elementos del subárbol izquierdo son menores o iguales al almacenado en el nodo
  - ○ y los elementos del subárbol derecho son estrictamente mayores al almacenado en el nodo

Ejemplo



✓ La siguiente función puede ser utilizada para comprobar si un árbol binario es de búsqueda:

```
esÁrbolBB                 :: Ord a ⇒ ÁrbolB a → Bool
esÁrbolBB VacíoB          =  True
esÁrbolBB (NodoB i r d)   =  todosÁrbolB (<= r) i
                             && todosÁrbolB (>  r) d
                             && esÁrbolBB i
                             && esÁrbolBB d

todosÁrbolB               :: (a → Bool) → ÁrbolB a → Bool
todosÁrbolB p VacíoB      =  True
todosÁrbolB p (NodoB i r d) =  p r &&
                               todosÁrbolB p i && todosÁrbolB p d
```

Árboles de búsqueda (2)

✓ Pertenencia a un árbol de búsqueda

```
perteneceBB                  ::  Ord a ⇒ a → ÁrbolB a → Bool
perteneceBB x VacíoB         =   False
perteneceBB x (NodoB i r d)
  | x == r                   =   True
  | x < r                    =   perteneceBB x i
  | otherwise                =   perteneceBB x d
```

✓ Inserción en un árbol de búsqueda

```
insertarBB                   ::  Ord a ⇒ a → ÁrbolB a → ÁrbolB a
insertarBB x VacíoB          =   NodoB VacíoB x VacíoB
insertarBB x (NodoB i r d)
  | x <= r                   =   NodoB (insertarBB x i) r d
  | otherwise                =   NodoB i r (insertarBB x d)
```

✓ Construcción de un árbol de búsqueda a partir de una lista

```
listaAÁrbolBB  ::  Ord a ⇒ [a] → ÁrbolB a
listaAÁrbolBB  =   foldr insertarBB VacíoB
```

✓ El recorrido en orden genera una lista ordenada (*tree sort*)

```
treeSort  ::  Ord a ⇒ [a] → [a]
treeSort  =   enOrdenB . listaAÁrbolBB
```

? *treeSort* [4, 7, 1, 2, 9]
[1, 2, 4, 7, 9] :: [*Integer*]

Objetivos del tema

El alumno debe:

- ✓ Conocer el concepto de árbol y la terminología asociada
- ✓ Conocer las definiciones de tipo para representar árboles en Haskell
- ✓ Saber definir funciones sobre árboles
- ✓ Conocer la implementación de los árboles de búsqueda en Haskell