INSTITUTO TECNOLOGICO SUPERIOR DE CHICONTEPEC.

INGENIERIA EN SISTEMAS COMPUTACIONALES.

ASIGNATURA:

Programación Lógica.

ALUMNO:

Flor Hernández Cruz.

DOCENTE:

Ingeniero Efrén Flores Cruz.

UNIDAD 2:

Modelo de programación funcional.

TRABAJO:

Reporte de unidad 2.

SEMESTRE:

8°

FECHA DE ENTREGA:

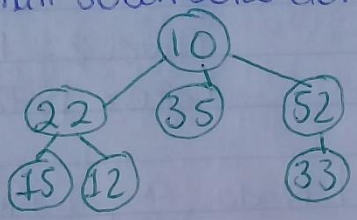
31 de marzo 2020

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 $\{V_1, V_2 .. V_n\}$ tales que:

Si $n = \emptyset$ el árbol se dice vacío.

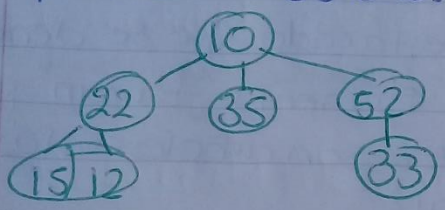
En otro caso, existen un valor destacado que se denomina raíz (p.e, V_1) los demas elementos forman parte de colecciones disjuntas que a su vez son árboles. Estos árboles se llaman subárboles del raíz



Estructuras de tipo árbol se usan para representar datos con una relación jerarquica entre sus elementos, como árboles genealogi
La terminología árbol proviene del árbol genealogico: Padre, hijo.

- Nodo son los elementos del árbol
- Raíz del árbol = todos los árboles que no están vacios 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
- Cada nodo tiene asociado un antesesor o ascediente 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.



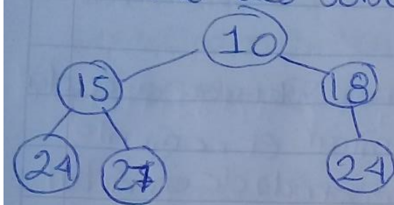
Ejemplo:
Raíz: 10
Nodos: 10, 22, 35, 52, 15, 12, 33
Tamaño: 7
Nivel ∅ = 10
Nivel 1: 22, 35, 52
Nivel 2: 15, 12, 33
Altura o profundidad: 3
Hojas: 15, 12, 35, 33

```
data Árbol a = Vacio | Nodo a [Árbol a] deriving
                       raíz   hijos
a1 :: Arbol 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 :: Arbol a → a
raíz Vacio = error "raíz de árbol Vacio"
raíz (Nodo x _) = x
tamaño :: Árbol a → Integer
tamaño Vacio = ∅
tamaño (Nodo _ xs) = 1 + sum (map tamaño xs)
```

```
Profundidad :: Árbol a -> Integer
Profundidad vacio = 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

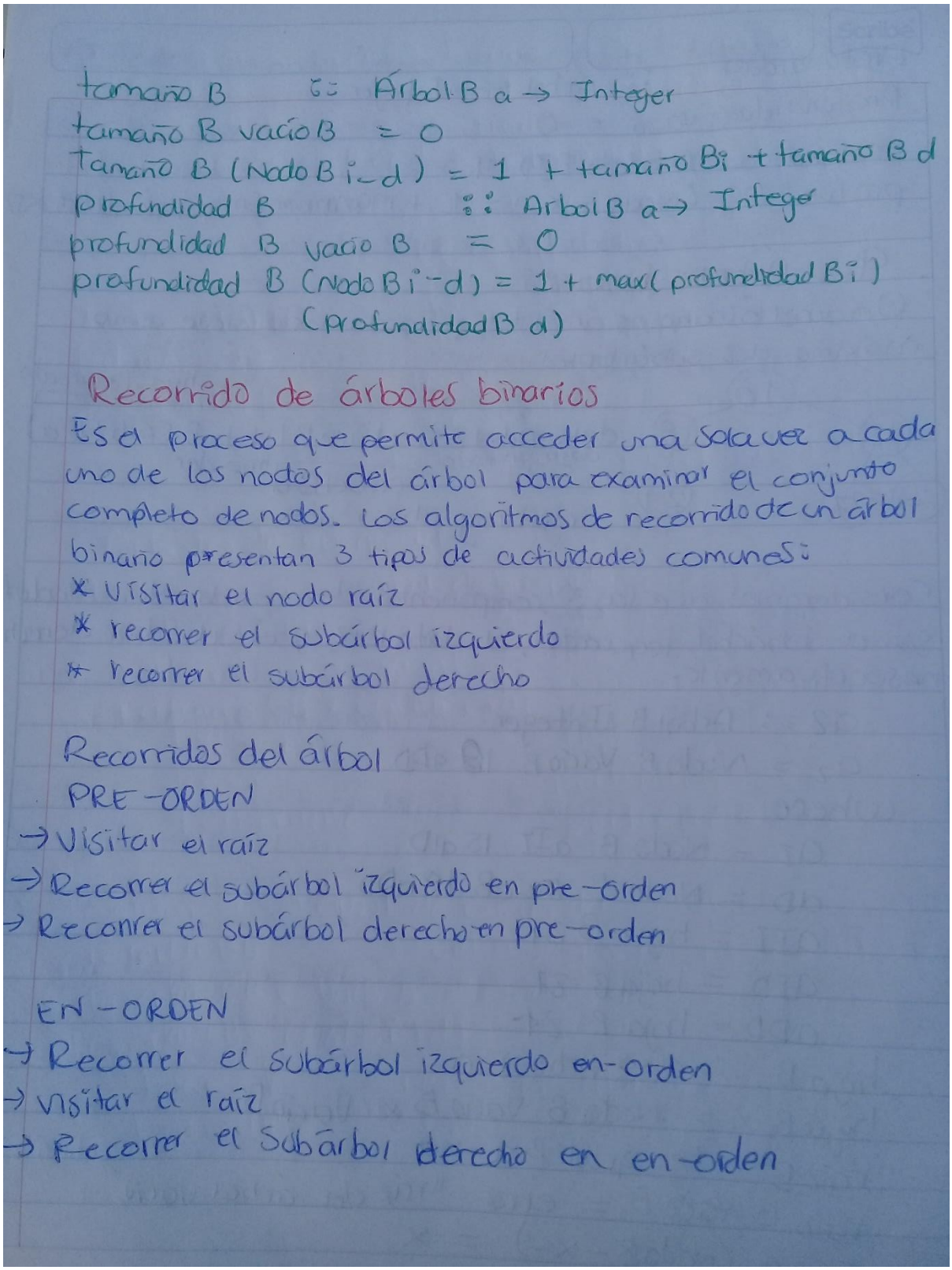



```
data ÁrbolB a = VacioB | NodoB (ÁrbolB a)
  deriving show
```

(dato en nodo: a; árbolB a: hijo der; (ÁrbolB a): hijo izquierdo)

Consideramos que las 3 componentes del constructor NodoB son el subárbol izquierdo, el dato raíz y el subárbol derecho respectivamente.

```
a2 :: ÁrbolB Integer
a2 = NodoB VacioB 10 aDD
where
    aI = NodoB aII 15 aID
    aD = NodoB vacioB 18 aDD
    aII = hojaB 24
    aID = hojaB 27
    aDD = hojaB 24
hojaB :: a -> Árbol B a
hojaB x = NodoB VacioB x VacioB
raízB :: Árbol B a -> a
raíz B vacioB = error "ráiz de árbol vacio"
raízB (NodoB - x -) = x
```



tamaño B :: Árbol B a → Integer
tamaño B vacío B = 0
Tamaño B (Nodo B i_d) = 1 + tamaño Bi + tamaño B d
profundidad B :: Arbol B a → Integer
profundidad B vacío B = 0
profundidad B (Nodo B i -d) = 1 + max(profundidad Bi)
(Profundidad B d)

Recorrido de árboles binarios

Es el 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 3 tipos de actividades comunes:

* Visitar el nodo raíz
* recorrer el subárbol izquierdo
* recorrer el subárbol derecho

Recorridos del árbol

PRE-ORDEN

→ Visitar el raíz
→ Recorrer el subárbol izquierdo en pre-orden
→ Recorrer el subárbol derecho en pre-orden

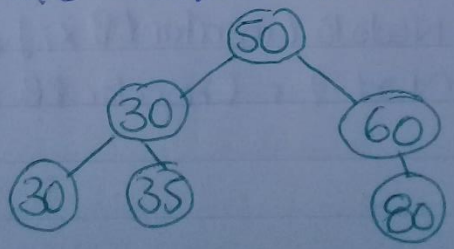
EN-ORDEN

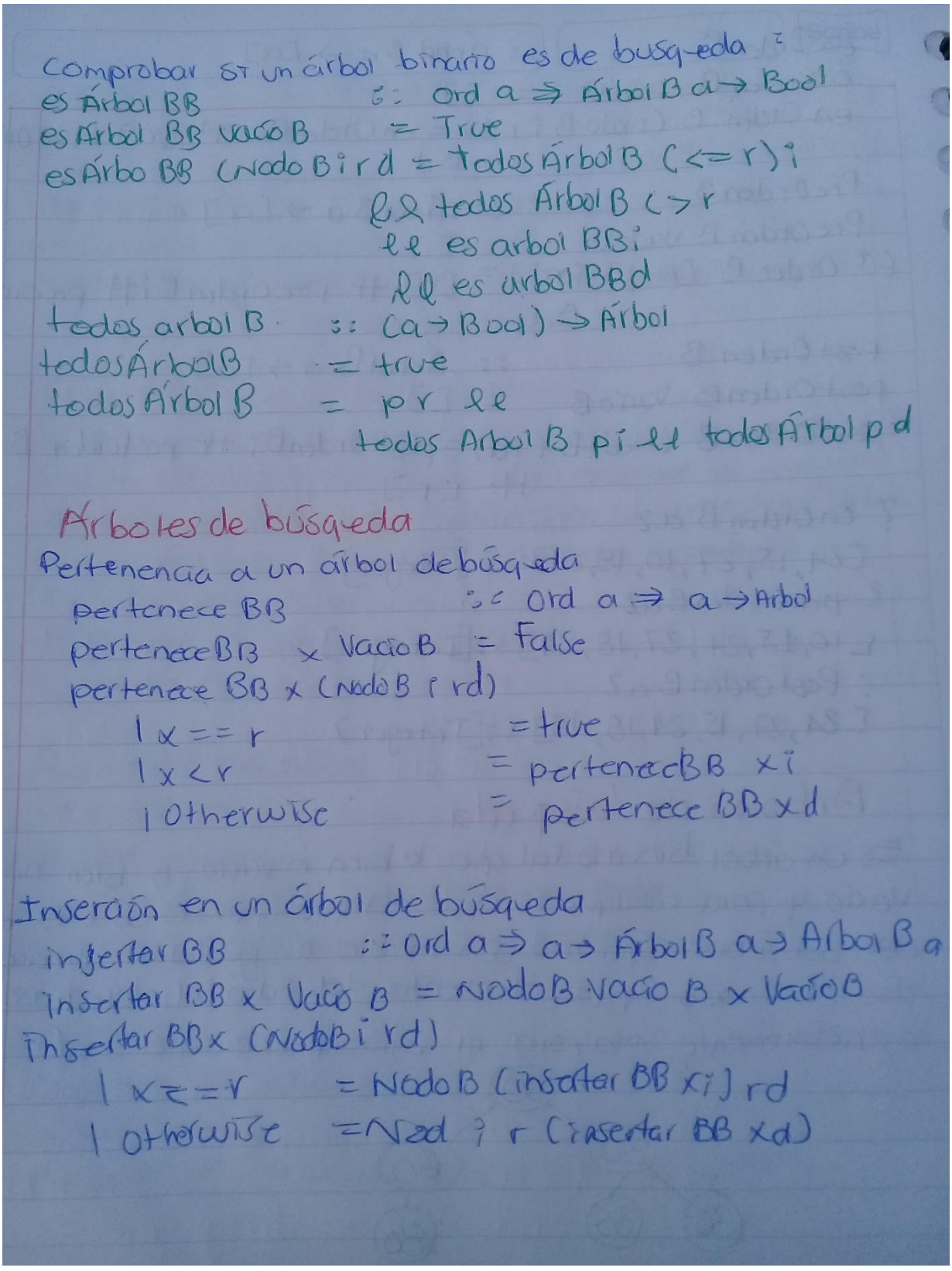
→ Recorrer el subárbol izquierdo en-orden
→ visitar el raíz
→ Recorrer el Subárbol derecho en en-orden

```
enOrdenB          :: ÁrbolBa → [a]
enOrdenB VacioB   = []
enOrden B (NodoB i r d) = enOrdenB i ++ [r] ++ enOrden
                          B d

PreOrdenB         :: ÁrbolBa → [a]
PreOrdenB VacioB  = []
preOrden B (NodB i rd) = [r] ++ preOrdenB i ++ preorden
                         Bd

PostOrdenB        :: ÁrbolBa → [a]
postOrdenB VacioB = []
postOrdenB (NodoB i rd) = postOrdenB i ++ postOrden B d
                          ++ [r]

? enOrdenB a2
[24,15,27,10,18,24] :: [Integer]
? preOrdenBa2
[10,15,24,27,18,24] :: [Integer]
? PostOrdenB a2
[24,27,15,24,18,10] :: [Integer]
```

Árboles de búsqueda

Es un árbol binario tal que o bien es vacío, o bien no es vacío y para cualquier nodo se cumple que: O 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.





```
Comprobar si un árbol binario es de busqueda :
esArbolBB          :: Ord a => ArbolB a -> Bool
esArbolBB vacioB    = True
esArboBB (NodoB i r d) = todosArbolB (<= r) i
                         && todosArbolB (> r)
                         && esarbolBBi
                         && esarbolBBd

todosarbolB        :: (a -> Bool) -> Arbol
todosArbolB         = true
todosArbolB         = p r le
                      todosArbolB pi && todosArbolpd
```

Arboles de búsqueda

Pertenencia a un árbol de búsqueda

```
pertenece BB                :: Ord a => a -> Arbol
perteneceBB x vacioB         = False
perteneceBB x (NodoB i r d)
    | x == r                 = true
    | x < r                  = perteneceBB x i
    | otherwise              = pertenece BB x d
```

Inserción en un árbol de búsqueda

```
insertar BB          :: Ord a => a -> ArbolB a -> ArbolB a
insertar BB x vacio B = NodoB vacio B x vacioB
insertar BB x (NodoB i r d)
    | x == r          = NodoB (insertar BB x i) r d
    | otherwise       = Nod i r (insertar BB x d)
```