TECNOLÓGICO NACIONAL DE MÉXICO
Campus Colima

Departamento de Sistemas y Computación

INGENIERÍA EN SISTEMAS COMPUTACIONALES

"Programación Lógica y Funcional"

A 2.8 Construcción e implementación de funciones para árboles

Que presenta:

Barragán Flores Luis Francisco

Villa de Álvarez, Colima, México a 29 abril de 2022

Árbol A

```
arbola = Nodo 10 (Nodo 15
                         (Nodo 24 Hoja Hoja)
                         (Nodo 27 Hoja Hoja))
                  (Nodo 18
                          Hoja (Nodo 24 Hoja Hoja))
```

1. raizB

El objetivo de esta función es determinar la raíz de un árbol binario, la cual seria el primer elemento que se encuentra en dicho árbol.

Usaremos el siguiente código:

```
--Raiz de un árbol
raizB Hoja      = error "Arbol vacio"
raizB (Nodo x _ _) = x
```

```
Prelude> :l 18460194_A2.8.hs
[1 of 1] Compiling Main             ( 18460194_A2.8.hs, interpreted )
Ok, one module loaded.
*Main> raizB arbola
10
```

2. nHojas

El objetivo de esta función es determinar el número de hojas que tiene un árbol binario.

Código:

```
-- Número de hojas de un árbol
nHojas :: Arbol a -> Int
nHojas Hoja = 1
nHojas (Nodo x i d) = nHojas i + nHojas d
```

En este caso nos indica que el árbol a tiene 7 hojas.

```
*Main> nHojas arbola
7
```

3. tamañoB

El objetivo de esta función es retornar el número de nodos que tiene el árbol.

Código:

```
tamañoB :: Arbol a -> Integer
tamañoB Hoja = 0
tamañoB (Nodo x i d) = 1 + tamañoB i + tamañoB d
```

En este caso podemos ver que el árbol a posee solo 6 nodos.

```
arbola = Nodo 10 (Nodo 15
                      (Nodo 24 Hoja Hoja)
                      (Nodo 27 Hoja Hoja))
                  (Nodo 18
                        Hoja (Nodo 24 Hoja Hoja))
```

```
*Main> tamañoB arbola
6
```

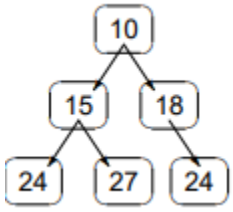
4. profundidadB

El objetivo de esta función consiste en retornar el número de niveles del árbol, considerando que la raíz es el nivel 1.

Código:

```
-- Retorna el número de niveles de un árbol
profundidadB :: Arbol a -> Int
profundidadB Hoja = 0
profundidadB (Nodo x i d) = 1 + max (profundidadB i) (profundidadB d)
```

Como se puede observar el árbol a tiene 3 niveles:




```
*Main> profundidadB arbola
3
```

5. enOrdenB

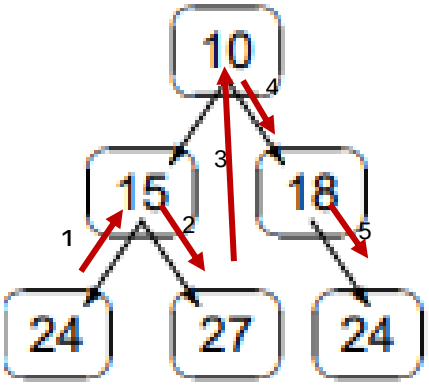
La siguiente función consiste en retornar el recorrido en entre orden de un árbol.

Código:

```
enOrdenB :: Arbol a -> [a]
enOrdenB Hoja = []
enOrdenB (Nodo x i d) = enOrdenB i ++ [x] ++ enOrdenB d
```

Caso podemos ver que el recorrido de este árbol inicia en el nodo 24, después sube al nodo 15 y en este caso como hay otro nodo hijo del 15 se dirige de nuevo a él para

después subir al nodo de arriba y bajar al nodo 18 y como el nodo 18 solo tiene un hijo y este hijo no tiene mas nodos ahí acaba el recorrido.




```
*Main> enOrdenB arbola
[24,15,27,10,18,24]
```

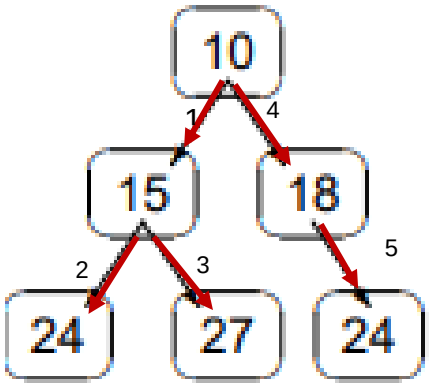
6. preOrdenB

Esta función al igual que la nos va a retornar un recorrido del árbol, pero este recorrido va a ser en pre orden.

Código:

```
preOrdenB :: Arbol a -> [a]
preOrdenB Hoja = []
preOrdenB (Nodo x i d) = x : (preOrdenB i ++ preOrdenB d)
```

El recorrido en pre Orden se recorre de la siguiente manera: raíz, subárbol izquierdo, subárbol derecho, como lo podemos ver a continuación.




```
*Main> preOrdenB arbola
[10,15,24,27,18,24]
```

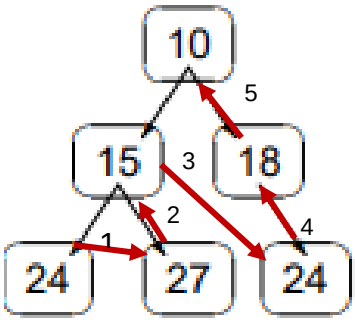
7. postOrdenB

Esta función retorna el recorrido post orden que sigue el árbol.

Código:

```
postOrdenB :: Arbol a -> [a]
postOrdenB Hoja = []
postOrdenB (Nodo x i d) = postOrdenB i ++ postOrdenB d ++ [x]
```

Se recorre de la siguiente manera: subárbol izquierdo, subárbol derecho, raíz.




```
*Main> postOrdenB arbola
[24,27,15,24,18,10]
```

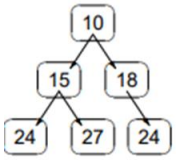
8. sumaArbolB

Esta función realiza la suma de todos los nodos del árbol utilizando funciones de orden superior.

Código:

```
sumaArbolB :: Arbol Integer -> Integer
sumaArbolB Hoja =0
sumaArbolB (Nodo x i d) = suma (sumaArbolB i) x (sumaArbolB d)
  where
     suma a b c = a + b + c
```

Como podemos ver suma de los valores del árbol a son 10 + 15 + 18 +24 +27+24 = 118




```
*Main> sumaArbolB arbola
118
```

Árbol B

```
arbolb = Nodo 50 (Nodo 30
                         (Nodo 30 Hoja Hoja)
                         (Nodo 35 Hoja Hoja))
                 (Nodo 60
                         Hoja (Nodo 80 Hoja Hoja))
```

9. esArbolBB

El objetivo de esta función es comprobar si un árbol binario es de búsqueda para esto retorna un true en caso que si sea un árbol de búsqueda y un False en caso contrario.

Código:

```
esArbolBB :: Ord a => Arbol a -> Bool
esArbolBB Hoja = True
esArbolBB (Nodo x i d) = todosArbolB(<= x) i
                         && todosArbolB (>x) d
                         && esArbolBB i
                         && esArbolBB d
```

Como podemos ver el árbol b si es un árbol de búsqueda.

```
*Main> esArbolBB arbolb
True
```

10. perteneceBB

El objetivo de esta función es indicar con true si el valor se encuentra dentro del árbol binario y false en caso de que este no esté.

Código:

```
perteneceBB n Hoja = False
perteneceBB n (Nodo x i d)
    | n == x = True
    | n < x = perteneceBB n i
    | otherwise = perteneceBB n d
```

Árbol b

```
arbolb = Nodo 50 (Nodo 30
                         (Nodo 30 Hoja Hoja)
                         (Nodo 35 Hoja Hoja))
                 (Nodo 60
                         Hoja (Nodo 80 Hoja Hoja))
```

Como podemos ver en el árbol b no tenemos ningún nodo con el número 24 por lo tanto nos marca false

```
*Main> perteneceBB 24 arbolb
False
```

En cambio, si ponemos 50 nos marca true ya que este si está y es el nodo raíz.

```
*Main> perteneceBB 50 arbolb
True
```

11. insertarBB

Esta función consiste en insertar el valor en el árbol binario de búsqueda.

Código:

```
insertarBB n Hoja = Nodo n Hoja Hoja
insertarBB n (Nodo x i d)
    | n <= x = Nodo x (insertarBB n i) d
    | otherwise = Nodo x i (insertarBB n d)
```

Como podemos ver aquí nos insertó el valor 24 que le pusimos.

```
*Main> insertarBB 24 arbolb
Nodo 50 (Nodo 30 (Nodo 30 (Nodo 24 Hoja Hoja) Hoja) (Nodo 35 Hoja Hoja)) (Nodo 60 Hoja (Nodo 80 Hoja Hoja))
```

12. listaAArbolBB

El objetivo de esta función es construir un árbol de búsqueda a partir de una lista de valores dada.

Código:

```
listaAArbolBB :: Ord a => [a] -> Arbol a
listaAArbolBB n = foldr insertarBB Hoja n
```

En este caso le vamos a dar esta lista de valores: [30,25,12,19,28,45,50], como podemos ver nos inserto los valores que indicamos iniciando como nodo raíz el 50, el cual es el valor que esta hasta al final de la fila.

```
*Main> listaAArbolBB [30,25,12,19,28,45,50]
Nodo 50 (Nodo 45 (Nodo 28 (Nodo 19 (Nodo 12 Hoja Hoja) (Nodo 25 Hoja Hoja)) (Nodo 30 Hoja Hoja)) Hoja) Hoja
```

13. arbolOrdenado

Esta función consiste en obtener una lista ordenada aplicando el recorrido de árboles.

Código:

```
arbolOrdenado :: Ord a => [a] ->[a]
arbolOrdenado = enOrdenB.listaAArbolBB
```

En este caso le vamos a dar esta lista de valores: [30,25,12,19,28,45,50]

```
*Main> arbolOrdenado [30,25,12,19,28,45,50]
[12,19,25,28,30,45,50]
```

Conclusión.

Esta actividad se me hizo muy interesante ya que me ayudo a recordar y reforzar un tema que ya había visto en los anteriores semestres, además de que se me hizo algo sencillo de desarrollar, sobre todo ya que las funciones venían en el pdf, entre las funciones que vi las que más interesantes y útiles que considero es la de listaAArboBB, ya que esta nos obtiene una lista que es dada por nosotros pero lo hace como si fuera un recorrido de árbol, además de esa esta la de los recorridos, esta función es especialmente útil para sacar los recorridos de algunos árbol grandes, además de esa esta la de sumar, si bien esta podría ser considera una función sencilla es bastante útil para sumar el contenido de los nodos de los árboles con muchos nodos.

Referencias

[1] P. Gallardo. [En línea]. Available: https://moodle.colima.tecnm.mx/pluginfile.php/19706/mod_assign/introattachment/0/10%20Arboles.pdf?forcedownload=1. [Último acceso: 29 04 2022].