Facultad de Ciencias UNAM
Lógica Computacional
Pruebas para práctica 1

Profesor: Francisco Hernández Quiroz
Ayudante: Valeria Garcia Landa
Ayudante de laboratorio: Sara Doris Montes Incin

1 Ejercicios

1. `mayorQue :: Natural -> Natural -> Bool`

   - Main> mayorQue (Suc(Suc(Suc Cero))) (Suc(Suc Cero))
     True
   - Main> mayorQue (Suc Cero) (Suc (Suc Cero))
     False

2. `restaNat :: Natural -> Natural -> Natural`

   - Main > restaNat (Suc(Suc Cero)) (Suc(Suc(Suc(Suc(Suc Cero)))))

     Cero
   - Main> restaNat (Suc(Suc(Suc(Suc Cero)))) (Suc(Suc(Suc Cero)))

     Suc Cero

3. `mulNat :: Natural -> Natural -> Natural`

   - Main> mulNat (Suc(Suc(Suc Cero))) (Suc(Suc(Suc(Suc(Suc Cero)))))

     Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc Cero))))))))))))))

   - Main > mulNat (Suc(Suc(Suc(Suc(Suc(Suc Cero)))))) (Suc Cero)

     Suc (Suc (Suc (Suc (Suc (Suc Cero)))))

4. `concatNat :: ListaNat -> ListaNat -> ListaNat`

   - Main > concatena (Cons (Suc Cero) Nil) (Cons Cero (Cons (Suc (Suc Cero)) Nil))

     Cons (Suc Cero) (Cons Cero (Cons (Suc (Suc Cero)) Nil))
   - Main > concatena (Cons (Suc(Suc(Suc Cero))) (Cons (Suc Cero) (Cons (Suc(Suc(Suc(Suc Cero)))) Nil))) (Cons Cero (Cons (Suc Cero) Nil))

     Cons (Suc (Suc (Suc Cero))) (Cons (Suc Cero) (Cons (Suc (Suc (Suc (Suc Cero)))) (Cons Cero (Cons (Suc Cero) Nil))))

5. `reversa :: ListaNat -> ListaNat`

   - Main > reversa (Cons Cero (Cons (Suc (Suc Cero)) (Cons (Suc Cero) Nil)))

     Cons (Suc Cero) (Cons (Suc (Suc Cero)) (Cons Cero Nil))
   - Main > reversa (Cons (Suc(Suc(Suc(Suc Cero)))) (Cons (Suc(Suc Cero)) (Cons (Suc Cero) Nil)))

     Cons (Suc Cero) (Cons (Suc (Suc Cero)) (Cons (Suc (Suc (Suc (Suc Cero)))) Nil))

6. `perteneceNat :: Natural -> ListaNat -> Bool`

   - Main> perteneceNat (Suc(Suc Cero)) (Cons Cero (Cons (Suc Cero) (Cons (Suc(Suc Cero)) Nil)))
     True
   - Main > perteneceNat (Suc Cero) (Cons Cero Nil)
     False

7. `inOrden :: BTree a -> [a]`

   - Main > inOrden (Node (Node Void 2 Void) 4 (Node(Node Void 6 Void) 7 Void))

     [2,4,6,7]
   - Main > inOrden (Node (Node (Node Void 4 Void) 2 (Node Void 5 Void)) 1 (Node (Node Void 6 Void) 3 (Node Void 7 Void)))

     [4,2,5,1,6,3,7]

8. `agregaOrden :: (Ord a) => a -> (BTree a) -> (BTree a)`

   - Main > agregaOrden 5 (Node (Node Void 2 Void) 4 (Node (Node Void 6 Void) 7 Void))

     Node (Node Void 2 Void) 4 (Node (Node (Node Void 5 Void) 6 Void) 7 Void)
   - Main > agregaOrden 8 (Node (Node Void 2 Void) 3 (Node Void 4 Void))

     Node (Node Void 2 Void) 3 (Node Void 4 (Node Void 8 Void))

9. `tailSnoc :: ListaSnoc a -> ListaSnoc a`

   - Main> tailSnoc (Snoc (Snoc (Snoc (Snoc (Snoc Empty 'a') 'b') 'c') 'd') 'e')

     Snoc (Snoc (Snoc (Snoc Empty 'b') 'c') 'd') 'e'
   - Main> tailSnoc (Snoc (Snoc (Snoc (Snoc Empty 1) 2) 3) 4)

     Snoc (Snoc (Snoc Empty 2) 3) 4

10. `mapSnoc :: (a -> b) -> ListaSnoc a -> ListaSnoc b`

    - Main> mapSnoc (*10) (Snoc (Snoc (Snoc (Snoc Empty 1) 2) 3) 4)

      Snoc (Snoc (Snoc (Snoc Empty 10) 20) 30) 40
    - Main> mapSnoc (+25) (Snoc (Snoc (Snoc Empty 10) 15) 20)

      Snoc (Snoc (Snoc Empty 35) 40) 45

2 Puntos Extra

1. `longitud :: Int -> Int`

   - Main> longitud 20
     2
   - Main> longitud 1997
     4

2. `tribonaccies :: Int -> [Int]`

- Main> tribonaccies 5
  [0,1,1,2,4,7]
- Main> tribonaccies 10
  [0,1,1,2,4,7,13,24,44,81,149]