```haskell
data Tree a = T a [Tree a]

--Aufgabe 1

size :: Tree a -> Int
size (T n []) = 1
size (T n (x:xs)) = size x + size (T n xs)

height :: Tree a -> Int
height (T n []) = 0
height (T n (x:xs)) = 1 + maximum (map (height) (x:xs))

loadOfInner :: Tree Int -> Int
loadOfInner (T n []) = n
loadOfInner (T n (x:xs)) = n + sum (map (loadOfInner) (x:xs))

maxLoadPath :: Tree Int -> Int
maxLoadPath (T n []) = 1
maxLoadPath (T n (x:xs)) = n + maximum (map (height) (x:xs))

postOrderTrav :: Tree a-> [a]
postOrderTrav (T n []) = [n]
postOrderTrav (T n (x:xs)) = postOrderTrav (x) ++ postOrderTrav (T n xs)

preOrderTrav :: Tree a -> [a]
preOrderTrav (T n []) = [n]
preOrderTrav (T n [x]) = n : (preOrderTrav x)
preOrderTrav (T n (x:y:xs)) = preOrderTrav (T n xs) ++ preOrderTrav (y) ++ preOrderTrav (x)

main = do
    print (maxLoadPath (T 1 [T 2 [T 4 [], T 5 [T 8 [], T 9 []]], T 3 [T 6 [], T 7 []]]))

--Aufgabe 3

data Mobile = Kugel Float | Stab (Mobile) (Mobile)
instance Eq where
                |length (Stab a) == length (Stab b) = Stab a == Stab b

masse :: Mobile -> Float
masse Kugel n = 1
masse (Stab Kugel Kugel) = masse n + masse n

--main = do
--  print (masse (Stab (Stab (Stab (Kugel 1), (Kugel 2)), (Kugel 3)), (Kugel 2)))

-- laenge,lL,lR,xL,x_R :: Mobile -> Float

```