```haskell
-- Aufgabe 1
data GTree a = Leaf a | GNode a [GTree a]
  deriving (Eq, Ord, Show, Read)

baum :: GTree Int
baum = GNode 1 [(GNode 2 [(GNode 5 [Leaf 7]), Leaf 6]),
   Leaf 3, Leaf 4]
-- a)
countLeaf :: GTree a -> Int
countLeaf (Leaf a) = 1
countLeaf (GNode a []) = 0
countLeaf (GNode a (x:xs)) = countLeaf x + countLeaf (GNode
    a xs)

-- b)
summe :: Num a => GTree a -> a
summe (Leaf a) = a
summe (GNode a []) = a
summe (GNode a (x:xs)) = summe x + summe (GNode a xs)

-- c)
hoehe :: GTree a -> Int
hoehe (Leaf a) = 0
hoehe (GNode a []) = 0
hoehe (GNode a (x:xs)) = max (1 + hoehe x) (1 + hoehe (
   GNode a xs))

-- d)
search :: Eq a => a -> (GTree a) -> Bool
search x (Leaf a) = x == a
search x (GNode a []) = x == a
search x (GNode a (y:ys)) = x == a || search x y || search
   x (GNode a ys)

-- e)

mapGTree :: (a->b) -> GTree a -> GTree b
mapGTree f (Leaf a) = Leaf (f a)
mapGTree f (GNode a (x:[])) = GNode (f a) [(mapGTree f x)]
mapGTree f (GNode a (x:xs)) = GNode (f a) ((mapGTree f x)
```

```
    :[(mapGTree f (GNode a xs))])
-- fehlerhaft

-- f)
collapse :: GTree a -> [a]
collapse (Leaf a) = [a]
collapse (GNode a []) = [a]
collapse (GNode a (x:xs)) = collapse x ++ collapse (GNode a
    xs)

-- Aufgabe 2
process :: [Int] -> Int -> Int -> Int
process xs n m
  | length xs < n || length xs < m = 0
  | otherwise = xs !! (n-1) + xs !! (m-1)
```