Tinpro01-7 Practicumopdracht 2

W. Oele

11 april 2023

Inleiding

In deze opdracht ga je:

- Je eigen datastructuur schrijven: de binary tree.
- Je datastructuur wegschrijven naar een bestand.
- Je datastructuur teruglezen uit een bestand.

Uitleg: de binary tree

Zoek op internet uit wat een *binary tree* is:
`https://en.wikipedia.org/wiki/Binary_search_tree`
Lees ook over *tree traversal*:
`https://en.wikipedia.org/wiki/Tree_traversal`
Zorg dat je goed begrijpt wat wordt verstaan onder:

- pre-order traversal
- in-order traversal
- post-order traversal

Opdracht 1

Schrijf het datatype Bintree. Dit datatype:

- heeft in elke node een waarde.
- heeft geen aparte leafs; We gebruiken de waarde `Empty`.
- is generiek.

Opdracht 2

Schrijf de volgende functies:

- `push :: (Ord a) => (Bintree a) -> a -> (Bintree a)`
  Push een element van het type a op de binary tree.
- `pushlist :: (Ord a) => (Bintree a) -> [a] -> (Bintree a)`
  Push een lijst met daarin elementen van het type a op de binary tree.
- `maptree :: (a -> b) -> (Bintree a) -> (Bintree b)`
  Mapt een functie over alle elementen van de binary tree.
- `filtertree :: (a->Bool) ->(Bintree a) -> [a]`
  Filtert elementen uit de binary tree en geeft de betreffende elementen terug in een lijst.
- `preorder :: (Bintree a) -> [a]`
  Voert een preorder traversal uit op een tree.
- `postorder :: (Bintree a) -> [a]`
  Voert een postorder traversal uit op een tree.
- `inorder :: (Bintree a) -> [a]`
  Voert een inorder traversal uit op een tree.

Uitleg: serializatie

Het komt in objectgeoriënteerde programma' s nogal eens voor dat men een zich in het geheugen bevindend object (met alle waarden van de in dat object zittende fields) wenst weg te schrijven naar een bestand. Enige tijd later kan men het bestand teruglezen en verder werken met het object alsof er niets gebeurd is.

Het wegschrijven van een een object(Java,Python,etc.)/struct(C)/datastructuur naar een bestand om het daarna weer terug te lezen staat bekend als *serializatie*. In Java, bijvoorbeeld, bestaat hiervoor de `Serializable` interface.

Ook in Haskell zijn er uiteraard libraries te vinden waarmee men datastructuren, lijsten, etc. kan serializeren. We gaan in deze opdracht echter onze eigen manier van serializeren programmeren m.b.v. de `show` en `read` functies.

De Show typeclass

De `Show` typeclass bevat de `show` functie. Deze functie maakt een `String` representatie van elk datatype dat van `Show` is afgeleid. Je eigen datastructuur afleiden van `Show` is eenvoudig:

```
data ... = ...
         | ...
         deriving(Show)
```

Wanneer zich in de variabele `t` een complete binary tree bevindt, kunnen we een string representatie van onze tree maken middels `show t` en deze string wegschrijven naar een bestand met de `writeFile` functie.

De Read typeclass

De `Read` typeclass bevat de `read` functie. Deze doet het omgekeerde van wat `show` doet:

```
main = do
  let x = "123"
  let y = read x :: Integer
  let z = y + 1
  putStrLn (show z)
```

In de derde regel wordt de `String` "123" omgezet in een `Integer` middels de `read` functie. Deze functie heeft als typesignature:

```
read :: Read a => String -> a:
```

De functie heeft een `String` als parameter en geeft een waarde van het type `a` terug. Als we de `read` functie een willekeurige string geven, weet Haskell natuurlijk niet wat die string voorstelt. Daarom zetten we er `:: Integer` achter om Haskell duidelijk te maken dat de betreffende string als een `Integer` moet worden opgevat.

In Haskell kan een String m.b.v. `read` omgezet worden in elk datatype dat van de `Read` typeclass is afgeleid. Ons eigen datatype van `Read` afleiden is eenvoudig:

```
data ... = ...
         | ...
         deriving(Show,Read)
```

Opdracht 3

Plaats je eigen naam, achternaam en leeftijd en die van je groepsgenoot in een tekst bestand. Schrijf een programma dat de volgende stappen uitvoert:

- De namen en leeftijden worden gelezen uit het bestand en in één String geplaatst.
- M.b.v. `pushList` worden alle letters en cijfers in een binary tree geplaatst.
- M.b.v. `mapTree` worden alle letters en cijfers in de tree omgezet naar `Int`.
- De tree wordt weggeschreven naar het bestand `tree.txt`.
- `tree.txt` wordt ingelezen en middels `read` omgezet in een tree.
- M.b.v. `mapTree` worden alle letters en cijfers in de tree omgezet naar `Char`.
- Een in-order traversal wordt gebruikt om een `String` van de tree te maken.
- Deze String wordt op het scherm afgedrukt.
- `filterTree` gebruiken we om uitsluitend de cijfers uit de tree te halen.
- Deze cijfers worden op het scherm afgedrukt.