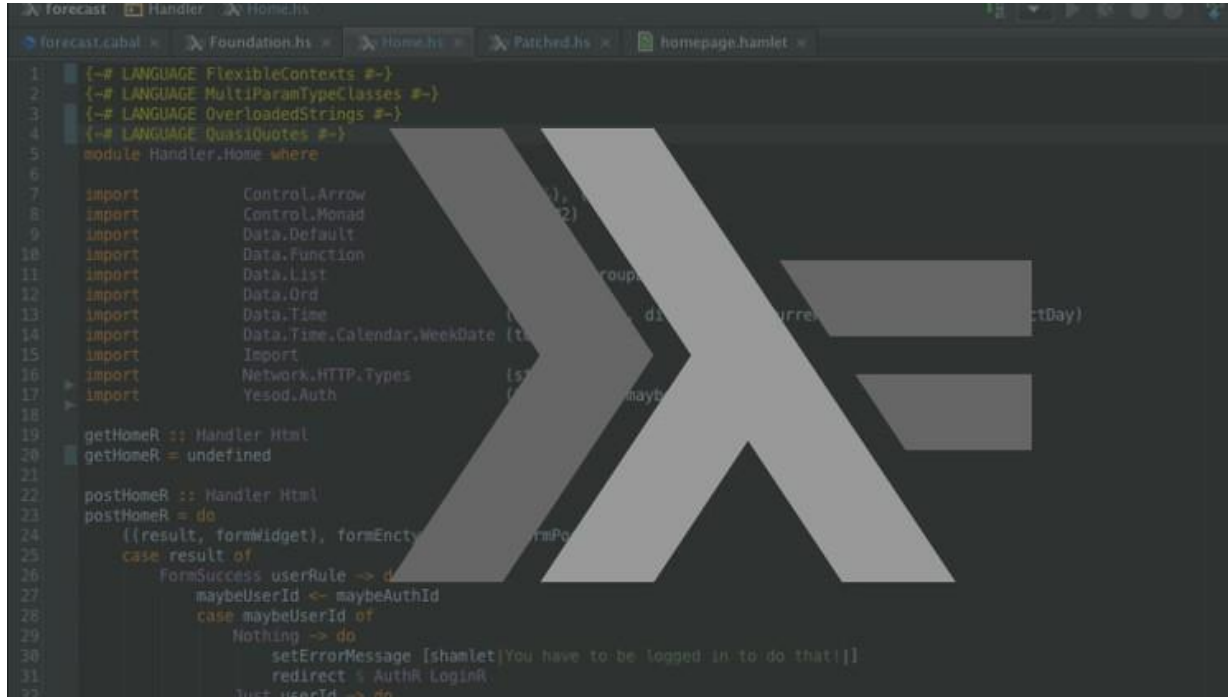


# Multiway Tree Problems



The image shows a screenshot of a Haskell code editor with a dark theme. The editor has several tabs at the top: 'forecast.cabal', 'Foundation.hs', 'Home.hs', 'Patched.hs', and 'homepage.hamlet'. The 'Home.hs' tab is active, displaying Haskell code. The code includes several language extensions at the top, followed by a module declaration 'module Handler.Home where'. It then lists various imports from libraries like 'Control.Arrow', 'Control.Monad', 'Data.Default', 'Data.Function', 'Data.List', 'Data.Ord', 'Data.Time', 'Data.Time.Calendar.WeekDate', 'Network.HTTP.Types', and 'Yesod.Auth'. Below the imports, there are function definitions for 'getHomeR' and 'postHomeR'. The 'postHomeR' function is a 'do' block that uses 'formEncrypted' and 'case result of' to handle different outcomes, including setting an error message and redirecting to a login page. A large, semi-transparent 'X' watermark is overlaid on the right side of the code.

```
1 {-# LANGUAGE FlexibleContexts #-}
2 {-# LANGUAGE MultiParamTypeClasses #-}
3 {-# LANGUAGE OverloadedStrings #-}
4 {-# LANGUAGE QuasiQuotes #-}
5 module Handler.Home where
6
7 import Control.Arrow
8 import Control.Monad
9 import Data.Default
10 import Data.Function
11 import Data.List
12 import Data.Ord
13 import Data.Time
14 import Data.Time.Calendar.WeekDate
15 import Network.HTTP.Types
16 import Yesod.Auth
17
18 getHomeR :: Handler Html
19 getHomeR = undefined
20
21 postHomeR :: Handler Html
22 postHomeR = do
23   ((result, formWidget), formEncrypted) <- formPost
24   case result of
25     FormSuccess userRule -> do
26       maybeUserId <- maybeAuthId
27       case maybeUserId of
28         Nothing -> do
29           setErrorMessage [shamlet|You have to be logged in to do that!|]
30           redirect % AuthR.LoginR
31       Just userId -> do
```

# Problem 3



```
forecast.cabal x Handler x Home.hs
forecast.cabal x Foundation.hs x Home.hs x Patched.hs x homepage.hamlet x

1 {-# LANGUAGE FlexibleContexts #-}
2 {-# LANGUAGE MultiParamTypeClasses #-}
3 {-# LANGUAGE OverloadedStrings #-}
4 {-# LANGUAGE QuasiQuotes #-}
5 module Handler.Home where
6
7 import Control.Arrow
8 import Control.Monad
9 import Data.Default
10 import Data.Function
11 import Data.List
12 import Data.Ord
13 import Data.Time
14 import Data.Time.Calendar.WeekDate (toDayOfYear, dayToWeek)
15 import Import
16 import Network.HTTP.Types
17 import Yesod.Auth
18
19 getHomeR :: Handler Html
20 getHomeR = undefined
21
22 postHomeR :: Handler Html
23 postHomeR = do
24   ((result, formWidget), formEnctype, formPost) <- runFormPost
25   case result of
26     FormSuccess userRule -> do
27       maybeUserId <- maybeAuthId
28       case maybeUserId of
29         Nothing -> do
30           setErrorMessage [shamlet|You have to be logged in to do that!|]
31           redirect % AuthR.LoginR
32         Just userId -> do
```

# Problem 3

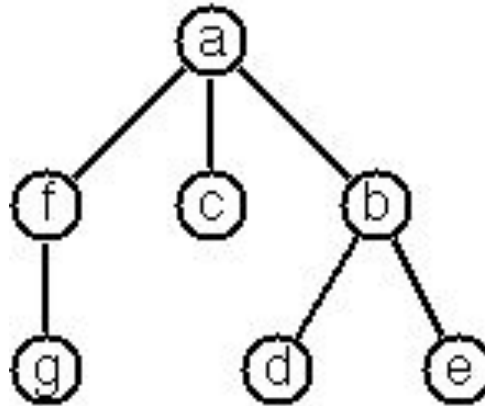


Write a function *pathLength* :: *Tree a* -> *Int* that, given a multiway tree, it returns the path length of the tree.

We define the path length of a multiway tree as the total sum of the path lengths from the root to all nodes of the tree.

Example below:

Path Length = 9



$$2 + 1 + 1 + 1 + 2 + 2 = 9$$

# Problem 3



Examples:

```
pathLength (Node 'a' [Node 'f' [Node 'g' []],Node 'c' [],Node 'b' [Node 'd' []], Node 'e' []])
```

-> **9**

```
pathLength (Node 'a' [Node 'b' [Node 'c' []]])
```

-> **3**

# Haskell Multiway Trees



In Haskell, we define multiway trees as a datatype:

```
data Tree a = Node a [Tree a]
              deriving (Eq, Show)
```

# Example



```
tree1 = Node 'a' [  
    Node 'f' [Node 'g' []],  
    Node 'c' [],  
    Node 'b' [Node 'd' [], Node 'e' []]  
]
```

