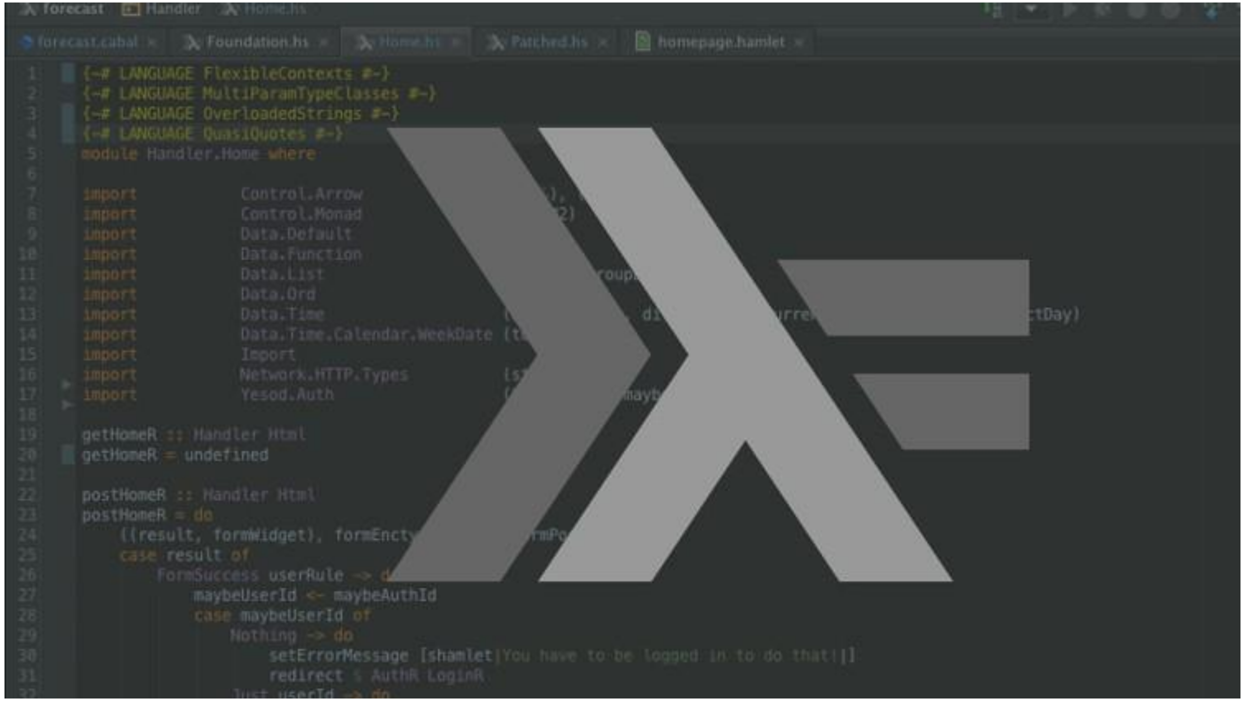
Graph Problems



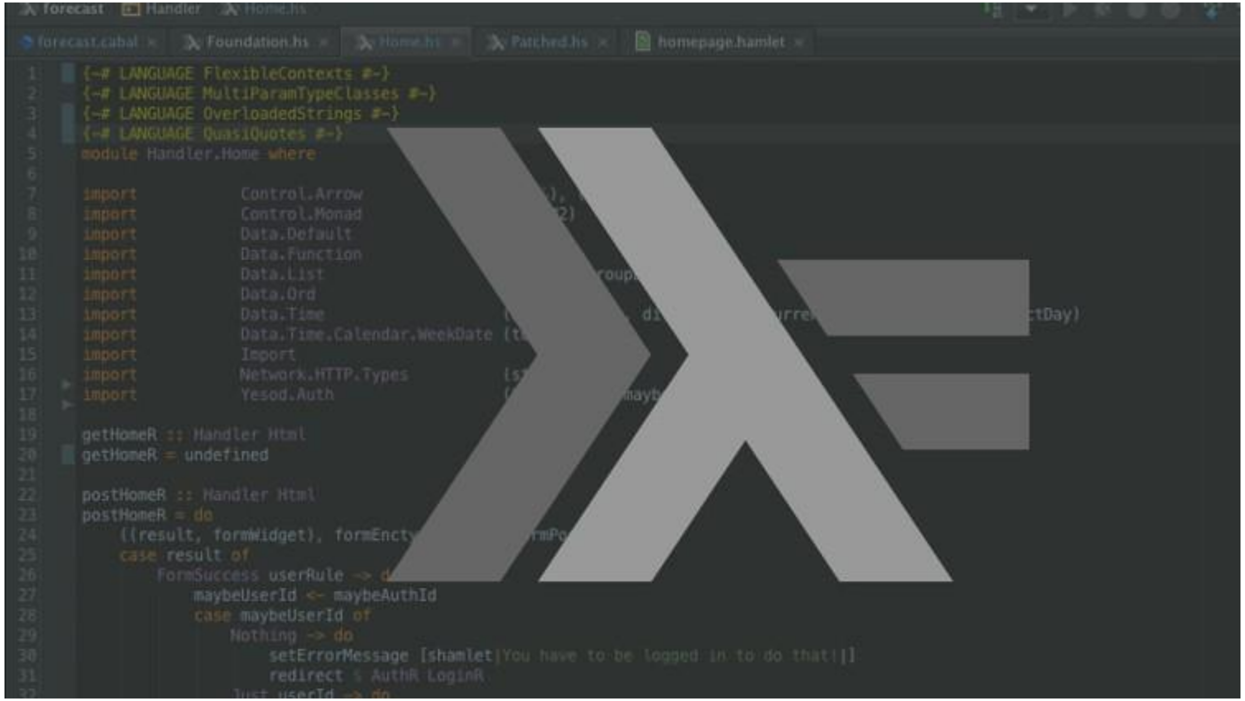
```
{-# LANGUAGE FlexibleContexts #-}
{-# LANGUAGE MultiParamTypeClasses #-}
{-# LANGUAGE OverloadedStrings #-}
{-# LANGUAGE QuasiQuotes #-}
module Handler.Home where

import          Control.Arrow                      ), (
import          Control.Monad                      2)
import          Data.Default
import          Data.Function
import          Data.List                            roup
import          Data.Ord
import          Data.Time                   (      , di      urre                  ctDay)
import          Data.Time.Calendar.WeekDate (t
import          Import
import          Network.HTTP.Types          (s
import          Yesod.Auth                  (           mayb

getHomeR :: Handler Html
getHomeR = undefined

postHomeR :: Handler Html
postHomeR = do
    ((result, formWidget), formEncty             rmPo
    case result of
        FormSuccess userRule -> d
            maybeUserId <- maybeAuthId
            case maybeUserId of
                Nothing -> do
                    setErrorMessage [shamlet|You have to be logged in to do that!|]
                    redirect $ AuthR LoginR
                Just userId -> do
```

Problem 1



```
{-# LANGUAGE FlexibleContexts #-}
{-# LANGUAGE MultiParamTypeClasses #-}
{-# LANGUAGE OverloadedStrings #-}
{-# LANGUAGE QuasiQuotes #-}
module Handler.Home where

import           Control.Arrow                      ), (
import           Control.Monad                      2)
import           Data.Default
import           Data.Function
import           Data.List                          roup
import           Data.Ord
import           Data.Time                  (       , di        urre                ctDay)
import           Data.Time.Calendar.WeekDate (t
import           Import
import           Network.HTTP.Types          (s
import           Yesod.Auth                  (          mayb

getHomeR :: Handler Html
getHomeR = undefined

postHomeR :: Handler Html
postHomeR = do
    ((result, formWidget), formEncty          rmPo
    case result of
        FormSuccess userRule -> d
            maybeUserId <- maybeAuthId
            case maybeUserId of
                Nothing -> do
                    setErrorMessage [shamlet|You have to be logged in to do that!|]
                    redirect $ AuthR LoginR
                Just userId -> do
```

Problem 1

Write a function *acyclicPaths :: Eq a => a -> a -> [(a,a)] -> [[a]]* that given two nodes a and b in a graph, returns all the acyclic paths from a to b.

```
Examples

acyclicPaths 1 4 [(1,2),(2,3),(1,3),(3,4),(4,2),(5,6)]

-> [[1,2,3,4],[1,3,4]]

acyclicPaths 2 6 [(1,2),(2,3),(1,3),(3,4),(4,2),(5,6)]

-> []
```

Problem 1

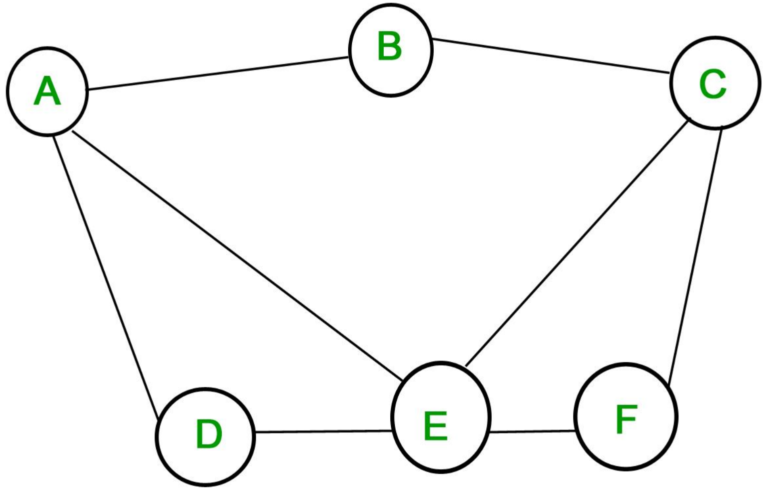
Note that the edges have directions.

In this problem edge (1,2) is **not** equal to edge (2,1)

As one example.

Cycle

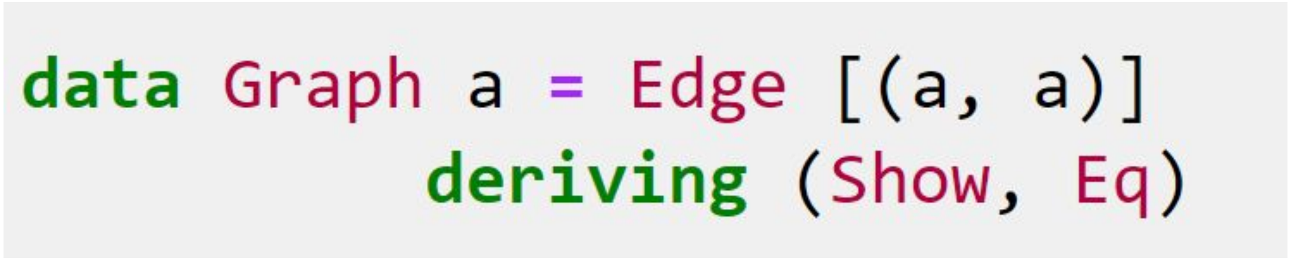
In graph theory, a cycle in a graph is a non-empty trail in which only the first and last vertices are equal.



B-C-E-A-B is a cycle example

Edge Notation

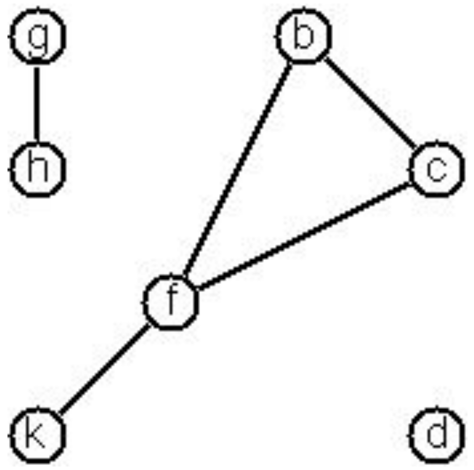
We represent the graph by its edges.



```haskell
data Graph a = Edge [(a, a)]
               deriving (Show, Eq)
```

Isolated nodes cannot be represented.

Example



```
Edge [(g,h), (k,f), (f,b), (f,c), (b,c)]
```