Paradigmas de Programação

Fabrício Olivetti de França

21 de Agosto de 2018

Atividade 02 [1,0 pto]

Crie um projeto chamado `factcheck` e acrescente a seguinte linha no arquivo cabal:

```
build-depends:       base >= 4.7 && < 5, http-conduit, bytestring , async, tagsoup, text
```

Complete o código do arquivo *Main.hs* conforme instruções e submeta utilizando o formulário em: .

```haskell
module Main where

import Network.HTTP.Conduit
import Text.HTML.TagSoup

import qualified Data.ByteString as B
import qualified Data.ByteString.Lazy as L
import qualified Data.Text as T
import qualified Data.Text.Encoding as E
import qualified Data.Text.IO as TI

import Data.Char
import Data.List

import Control.Concurrent
import Control.Concurrent.Async

-- pega conteúdo de url
getURL :: String -> IO B.ByteString
getURL url = L.toStrict <$> simpleHttp url

-- substitui caracteres acentuados
substAcento :: Char -> Char
substAcento 'ç' = 'c'
substAcento 'á' = 'a'
```

```haskell
substAcento 'é' = 'e'
substAcento 'í' = 'i'
substAcento 'ó' = 'o'
substAcento 'ú' = 'u'
substAcento 'à' = 'a'
substAcento 'ã' = 'a'
substAcento 'õ' = 'o'
substAcento c = c

-- transforma texto em minúsculo, sem acentos e somente caracteres, números e espaço
normalizaTexto :: T.Text -> T.Text
normalizaTexto = T.filter (c -> isAlpha c || isSpace c) . T.map substAcento . T.toLower

-- similaridade entre dois textos
-- Exercicio: utilizando a função `words` divida cada texto
-- em uma lista de palavras e calcule a similaridade de Jaccard
calcSim :: T.Text -> T.Text -> Double
calcSim t1 t2 = ???

-- similaridade de Jaccard
-- a similaridade de Jaccard é a divisão entre
-- o tamanho da interseção das listas pelo
-- tamanho da união das listas
jaccard :: (Eq a) => [a] -> [a] -> Double
jaccard xs ys = ???

-- Pega o título da matéria
getTitle :: Tag B.ByteString -> B.ByteString
getTitle (TagOpen s as) = snd $ head $ drop 1 as
getTitle _ = B.empty

-- pega apenas os resultados da busca
resultadosBusca :: String -> IO [B.ByteString] -- IO ()
resultadosBusca url = do
    tags <- parseTags <$> getURL url
    let articles   = sections (~== "<div class="article-content clearfix">") tags
        links      = map (head . drop 1 . filter (~== "<a>")) articles
        filtered   = map (getTitle) links
    return filtered

baseUrl :: T.Text
baseUrl  = T.pack "http://www.boatos.org/?s="

-- retorna uma lista de tuplas com os n textos mais similares
-- em relação a query de busca.
-- utilize a função calcSim para determinar a similaridade
```

```haskell
topN :: Int -> T.Text -> [T.Text] -> [(Double, T.Text)]
topN n q = ???

main :: IO ()
main = do
  -- faz busca assíncrona por cada palavra digitada
  entrada <- B.getLine
  let query    = normalizaTexto $ E.decodeUtf8 entrada
      palavras = T.words query
      urls     = map (w -> T.unpack $ T.append baseUrl w) palavras

  -- aplique a função resultadosBusca assincronamente em cada url
  -- espere pelo resultado e concatene tudo em uma única lista
  as         <- ???
  resultados <- ???

  -- recupera os top 5 resultados mais similares
  let normalizado = map (normalizaTexto . E.decodeUtf8) resultados
      similares   = topN 10 query normalizado
  print similares
```