```
{- *
   * Find all of the words in the dictionary that have two or more
   * consecutive pairs of letters.
-}

import System.IO    -- readFile
import Data.Char    -- toLower

-- | replace runs of identical things by a count and the the thing.
-- /e.g.,/
-- > runs [1,1,3,2,2,2,2]
-- is
-- > [(2,1),(1,3),(4,2)]
runs :: Eq a => [a] -> [(Int,a)]
runs [] = []
runs (x:xs) = runs' (1,x) xs where
  runs' a [] = [a]
  runs' (n,x) (y:ys)
    | x==y          = runs' (n+1,x) ys
    | otherwise     = (n,x) : runs' (1,y) ys

-- |  count adjacent twos
-- /e.g.,/
-- > successivePairsCounts [2,1,2,2,2,3,2]
-- is
-- > [1,3,1]
successivePairsCounts :: [Int] -> [Int]
successivePairsCounts (2:xs) = let
  (ys,zs) = span (==2) xs
  in 1+length ys: successivePairsCounts zs
successivePairsCounts (_:xs) =
  successivePairsCounts . dropWhile (/= 2) $ xs
successivePairsCounts [] = []

-- | find the maximum number of consecutive pairs in a list
maxPairCounts :: Eq a => [a] -> Int
maxPairCounts xs = maximum . (0:) . successivePairsCounts . map fst . runs $ xs

{-
maxPairCounts xs = let
  counts = map fst (runs xs)
  successive = successivePairsCounts counts
  in maximum (0:successive)
-}


bookkeeperIsh :: String -> Bool
bookkeeperIsh = (> 1) . maxPairCounts
-- bookkeeperIsh string = (maxPairCounts string) > 1


dictFileName :: String
dictFileName = "/usr/share/dict/web2"

main :: IO ()
main = do
  fileContents <- readFile dictFileName
  putStr . unlines . filter bookkeeperIsh . lines . map toLower $ fileContents
```