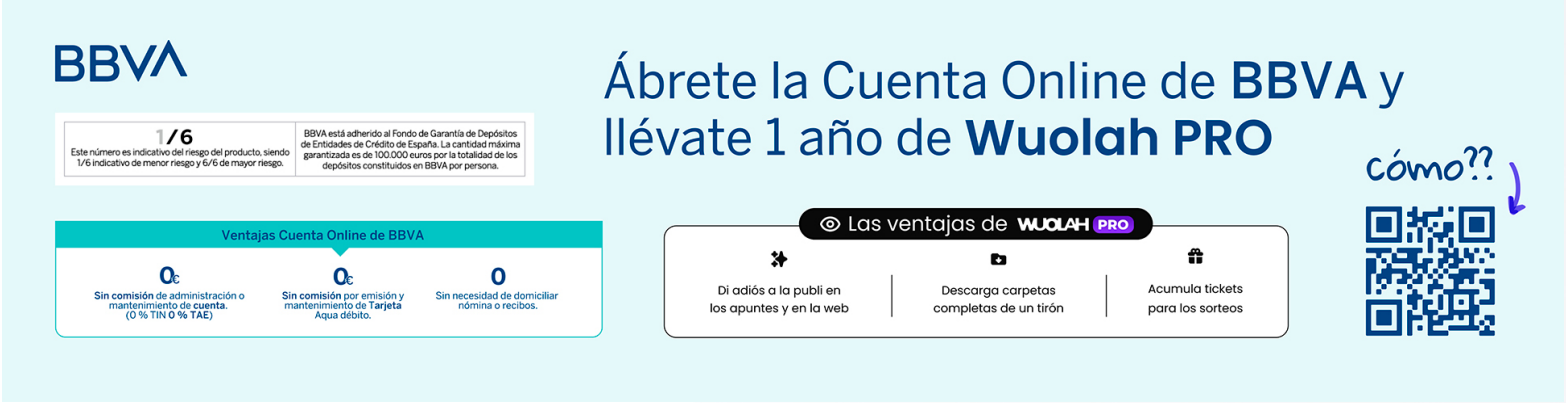
Solucion-febrero-2012-haskell.pdf

angelgg0700

Estructuras de Datos

2º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingeniería Informática
Universidad de Málaga



```
module BiPartite(
    biColored    -- :: Ord v  => Graph v -> Maybe (D.Dictionary v Color)
  , Color(..)
  )  where

import Graph
import Data.Maybe(isJust)
import qualified Dictionary as D
import qualified Stack as S


data Color = Red |  Blue deriving (Eq,Show,Ord)
nextColor Red  = Blue
nextColor Blue = Red

pushAll :: S.Stack a -> [a] -> S.Stack a
pushAll = foldr S.push

biColored :: Ord v => Graph v -> Maybe (D.Dictionary v Color)
biColored g
 | null vs   = Just D.empty -- empty graph is bipartite
 | otherwise = aux g D.empty (S.push (src ,Red) S.empty)
 where
  vs  = vertices g
  src = head vs -- initial vertex

aux :: Ord v => Graph v -> D.Dictionary v Color -> S.Stack (v, Color) ->
                Maybe (D.Dictionary v Color)
aux g dict stack
   |S.isEmpty stack       = Just dict
   |not(colored v) = aux g dict' (pushAll stack' newsuc)
   |c == c' = aux g dict stack'
   |otherwise = error "no bipartito"

--
-- ¡¡¡ completad el resto de guardas !!
--
  where
     colored  v = D.isDefinedAt v dict
     stack' = S.pop stack
     (v, c) = S.top stack
     dict' = D.insert v c dict
     c' = D.valueOf' v dict
     newsuc = [(v,nextColor c)| v <- successors g v, not(colored v)]

--
-- ¡¡¡ completad las variables locales necesarias !!!
--

---------------------
--- EXAMPLES --------
---------------------
```