99 questions/Solutions/85

From HaskellWiki

< 99 questions | Solutions

(**) Graph isomorphism

Two graphs G1(N1,E1) and G2(N2,E2) are isomorphic if there is a bijection f: N1 -> N2 such that for any nodes X,Y of N1, X and Y are adjacent if and only if f(X) and f(Y) are adjacent.

Write a predicate that determines whether two graphs are isomorphic.

This solution compares the canonical forms of the two graphs to determine whether they are isomorphic.

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

data Adjacency a = Adj [(a, [a])]
                   deriving (Show, Eq)

graphG1 = Graph [1, 2, 3, 4, 5, 6, 7, 8]
                [(1, 5), (1, 6), (1, 7), (2, 5), (2, 6), (2, 8),
                 (3, 5), (3, 7), (3, 8), (4, 6), (4, 7), (4, 8)]

graphH1 = Graph [1, 2, 3, 4, 5, 6, 7, 8]
                [(1, 2), (1, 4), (1, 5), (6, 2), (6, 5), (6, 7),
                 (8, 4), (8, 5), (8, 7), (3, 2), (3, 4), (3, 7)]

graphToAdj :: (Eq a) => Graph a -> Adjacency a
graphToAdj (Graph [] _)      = Adj []
graphToAdj (Graph (x:xs) ys) = Adj ((x, ys >>= f) : zs)
   where
      f (a, b)
         | a == x = [b]
         | b == x = [a]
         | otherwise = []
      Adj zs = graphToAdj (Graph xs ys)

iso :: (Ord a, Enum a, Ord b, Enum b) => Graph a -> Graph b -> Bool
iso g@(Graph xs ys) h@(Graph xs' ys') = length xs == length xs' &&
                                        length ys == length ys' &&
                                        canon g == canon h

canon :: (Ord a, Enum a) => Graph a -> String
canon g = minimum $ map f $ perm $ length a
   where
      Adj a = graphToAdj g
      v = map fst a
      perm n = foldr (\x xs -> [i : s | i <- [1..n], s <- xs, i `notElem` s]) [[]] [1..n]
      f p = let n = zip v p
```

```
        in show [(snd x,
                  sort id $ map (\x ->
                     snd $ head $ snd $ break ((==) x . fst) n) $ snd $ find a x)
                | x <- sort snd n]
    sort f n = foldr (\x xs -> let (lt, gt) = break ((<) (f x) . f) xs
                               in lt ++ [x] ++ gt) [] n
    find a x = let (xs, ys) = break ((==) (fst x) . fst) a in head ys
```
