99 questions/Solutions/80

From HaskellWiki

< 99 questions | Solutions

(***) Conversions

Write predicates to convert between the different graph representations.

Here is a working solution for the graph-term, adjacency-list, and edge-clause / human-friendly forms for undirected, unweighted graphs.

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

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

data Friendly a = Edge [(a, a)]
                  deriving (Show, Eq)

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)

adjToGraph :: (Eq a) => Adjacency a -> Graph a
adjToGraph (Adj [])          = Graph [] []
adjToGraph (Adj ((v, a):vs)) = Graph (v : xs) ((a >>= f) ++ ys)
   where
      f x = if (v, x) `elem` ys || (x, v) `elem` ys
            then []
            else [(v, x)]
      Graph xs ys = adjToGraph (Adj vs)

graphToFri :: (Eq a) => Graph a -> Friendly a
graphToFri (Graph [] _)  = Edge []
graphToFri (Graph xs ys) = Edge (ys ++ zip g g)
   where
      g = filter (\x -> all (\(a, b) -> x /= a && x /= b) ys) xs

friToGraph :: (Eq a) => Friendly a -> Graph a
friToGraph (Edge []) = Graph [] []
friToGraph (Edge vs) = Graph xs ys
   where
      xs = foldr acc [] $ concat $ map (\(a, b) -> [a, b]) vs
      ys = filter (uncurry (/=)) vs
      acc x xs = if x `elem` xs then xs else x : xs
```

```haskell
adjToFri :: (Eq a) => Adjacency a -> Friendly a
adjToFri = graphToFri . adjToGraph

friToAdj :: (Eq a) => Friendly a -> Adjacency a
friToAdj = graphToAdj . friToGraph
```