# 99 questions/Solutions/81

## From HaskellWiki

< 99 questions | Solutions

(**) Path from one node to another one

Write a function that, given two nodes a and b in a graph, returns all the acyclic paths from a to b.

```
import List (elem)

paths :: Eq a => a -> a -> [(a,a)] -> [[a]]
paths a b g = paths1 a b g []

paths1 :: Eq a => a -> a -> [(a,a)] -> [a] -> [[a]]
paths1 a b g current = paths2 a b g current [ y | (x,y) <- g, x == a ]

paths2 :: Eq a => a -> a -> [(a,a)] -> [a] -> [a] -> [[a]]
paths2 a b g current [] | a == b = [current++[b]]
                        | otherwise = []
paths2 a b g current (x:xs) | a == b = [current++[b]]
                            | elem a current = []
                            | otherwise = (paths1 x b g (current++[a])) ++ (paths2 a b g curr
```

This solution uses a representation of a (directed) graph as a list of arcs (a,b).

---

Here is another implementation using List's monadic behavior

```
import Data.List (partition)

pathsImpl :: Eq a => [a] -> a -> a -> [(a, a)] -> [[a]]
pathsImpl trail src dest clauses
    | src == dest = [src:trail]
    | otherwise = do
        let (nexts, rest) = partition ((==src) . fst) clauses
        next <- nexts
        pathsImpl (src:trail) (snd next) dest rest

paths :: Eq a => a -> a -> [(a, a)] -> [[a]]
paths src dest clauses = map reverse (pathsImpl [] src dest clauses)
```

---

Here is another recursive implementation

```
paths :: Eq a =>a -> a -> [(a,a)] -> [[a]]
paths source sink edges
    | source == sink = [[sink]]
    | otherwise = [
        [source] ++ path | edge<-edges, (fst edge) == source,
        path<-(paths (snd edge) sink [e|e<-edges, e/=edge])
```

```
    ];
```