```
-----------------------Q1-------------------------------------

data Edge t = Edge Int Int deriving (Eq,Show,Ord) -- vai para / Custo
data Node t = Node Int t [(Edge t)]  deriving (Eq,Show,Ord)


gtest:: [Node Int]
gtest = [Node 0 10 [(Edge 2 1),(Edge 3 1)] ,
         Node 1  123 [(Edge 0 2)] ,
         Node 2  14 [(Edge 3 2),(Edge 0 2)] ,
         Node 3  43 [] ]

gtest1:: [Node Int]
gtest1 = [Node 0 10 [(Edge 2 1),(Edge 3 1)] ,
         Node 3  43 [],
         Node 1  123 [(Edge 0 2)] ,
         Node 2  14 [(Edge 3 2),(Edge 0 2)] ]

preOrder:: [Node t]->[Node t]
preOrder [] = []
preOrder ((Node i v ls):xs) = (Node i v (qs ls)):(preOrder xs)

divide:: (Ord t)=>[t]->t->([t],[t])->([t],[t])
divide [] n p= p
divide (x:xs) n (me, ma)    |  x>n = (divide xs n (me,x:ma))
                            | otherwise = divide xs n ((x:me),ma)

qs :: (Ord t)=>[t]->[t]
qs [] = []
qs p = (qs me )++ (head p):(qs ma)
        where (me,ma) = (divide (tail p) (head p) ([],[]))

equal::(Ord t) => [Node t]->[Node t]-> Bool
equal a b = (qs$preOrder$ a) == (qs$preOrder$ b)

-----------------------Q2-------------------------------------------------
dfs :: (Eq t, Ord t) => [Node t] -> t -> Bool
dfs nodes k = buscaProfundidade (qs nodes) k

buscaProfundidade:: (Eq t) => [Node t] -> t -> Bool
buscaProfundidade [] _ = False
buscaProfundidade ((Node i n []):cs) k = (n == k)
buscaProfundidade (Node i n (b:bs):cs) k | (k == n) = True
                                         | otherwise = fazerBusca [i] nodes [] k
                                         where
                                            nodes = (Node i n (b:bs):cs)

fazerBusca :: (Eq t) => [Int] -> [Node t] -> [Int] -> t -> Bool
fazerBusca _ [] _ _ = False
fazerBusca _ [(Node i n [])] _ k = (n == k)
fazerBusca [] _ _ _ = False
fazerBusca (a:as) ((Node i n []):bs) _ k = (k == n)
fazerBusca (a:as) grafo vis k | k == getVal (grafo!!a) = True
                              | otherwise = fazerBusca (pilhaNos ++ as) grafo visinhos k
                              where
                                nodesAT = (take a grafo) ++ (drop (a+1) grafo)
                                pilhaNos = elimina (retornaVisinhos (grafo!!a)) visinhos
                                visinhos = (vis ++ [a])

getVal:: Node t -> t
getVal (Node i n _) = n

elimina::[Int]->[Int]->[Int]
elimina [] _ = []
elimina [x] [] = [x]
elimina [x] (y:ys) | x==y = []
                   | otherwise = elimina [x] ys
elimina (x:xs) []  = x:xs
elimina (x:xs) (y:ys) | x==y = (elimina xs (y:ys))
                      | otherwise = (elimina [x] ys)++(elimina xs (y:ys))
```

```
retornaVisinhos :: Node t -> [Int]
retornaVisinhos (Node i k []) = []
retornaVisinhos (Node i k ((Edge b c):as)) = [b] ++ retornaVisinhos (Node i k as)


gtest2:: [Node Int]
gtest2 = [ Node 0 10 [(Edge 3 10), (Edge 1 2)],
           Node 1 30 [(Edge 2 6)],
           Node 2 20 [(Edge 0 4)],
           Node 3 50 []]
```