99 questions/Solutions/89

From HaskellWiki

< 99 questions | Solutions

```
import Data.List

type Node = Int
type Edge = (Node,Node)
type Graph = ([Node],[Edge])

dfsbipartite :: Graph -> [(Node, Int)] -> [Node] -> [Node] -> Bool
dfsbipartite ([],_) _ _ _ = True
dfsbipartite (_,_) [] _ _ = True
dfsbipartite (v,e) ((nv, 0):stack) odd even
    | [x|x<-v,x==nv] == [] = dfsbipartite (v, e) stack odd even
    | [] == intersect adjacent even = dfsbipartite (newv, e) ([(x,1)|x<-adjacent] ++ stack) o
    | otherwise = False
    where
        adjacent = [x | (x,y)<-e,y==nv] ++ [x | (y,x)<-e,y==nv]
        newv = [x|x<-v,x/=nv]
dfsbipartite (v,e) ((nv, 1):stack) odd even
    | [x|x<-v,x==nv] == [] = dfsbipartite (v, e) stack odd even
    | [] == intersect adjacent odd = dfsbipartite (newv, e) ([(x,0)|x<-adjacent] ++ stack) (n
    | otherwise = False
    where
        adjacent = [x | (x,y)<-e,y==nv] ++ [x | (y,x)<-e,y==nv]
        newv = [x|x<-v,x/=nv]

bipartite :: Graph -> Bool
bipartite ([],_) = True
bipartite (top:v,e) = dfsbipartite (top:v, e) [(top,0)] [] []
```

You can call it:

```
bipartite ([1,2,3,4,5],[(1,2),(2,3),(1,4),(3,4),(5,2),(5,4)])
```
