

# Haskell-feb-2016.pdf



Juanma21\_



Estructuras de Datos



2º Grado en Ingeniería Informática



Escuela Técnica Superior de Ingeniería Informática  
Universidad de Málaga

**WUOLAH + BBVA**

## Te regalamos

# 15€



**1/6**

Este número es indicativo del riesgo del producto, siendo 1/6 indicativo de menor riesgo y 6/6 de mayor riesgo.

BBVA está adherido al Fondo de Garantía de Depósitos de Entidades de Crédito de España. La cantidad máxima garantizada es de 100.000 euros por la totalidad de los depósitos constituidos en BBVA por persona.

**1**

Abre tu Cuenta Online sin comisiones ni condiciones

**2**

Haz una compra igual o superior a 15€ con tu nueva tarjeta

**3**

BBVA te devuelve un máximo de 15€



1/6

Este número es indicativo del riesgo del producto, siendo 1/6 indicativo de menor riesgo y 6/6 de mayor riesgo.

BBVA está adherido al Fondo de Garantía de Depósitos de Entidades de Crédito de España. La cantidad máxima garantizada es de 100.000 euros por la totalidad de los depósitos constituidos en BBVA por persona.

1

Abre tu Cuenta Online sin comisiones ni condiciones

2

Haz una compra igual o superior a 15€ con tu nueva tarjeta

3

BBVA te devuelve un máximo de 15€

15/1/23, 17:34

EulerianCycle.hs

```

1 -----
2 -- Student's name:
3 -- Student's group:
4 --
5 -- Data Structures. Grado en Informática. UMA.
6 -----
7
8 module DataStructures.Graph.EulerianCycle(isEulerian, eulerianCycle) where
9
10 import DataStructures.Graph.Graph
11 import Data.List
12
13 --H.1)
14 isEulerian :: Eq a => Graph a -> Bool
15 isEulerian g
16   | isEmpty g || length (vertices g) == 1 = True
17   | all even (map (degree g) (vertices g)) = True
18   | (length (vertices g) == 2) && all (>2) (map (degree g) (vertices g)) = True
19   | otherwise = False
20
21 -- H.2)
22 remove :: (Eq a) => Graph a -> (a,a) -> Graph a
23 remove g (v,u) = remove' (vertices g') g'
24   where
25     g' = deleteEdge g (v, u)
26     remove' [] g = g
27     remove' (x:xs) g
28       | (degree g x) == 0 = remove' xs (deleteVertex g x)
29       | otherwise = remove' xs g
30
31
32
33 -- H.3)
34 extractCycle :: (Eq a) => Graph a -> a -> (Graph a, Path a)
35 extractCycle g v0 = extractCycle' g [v0] v0 (head (successors g v0))
36   where
37     extractCycle' g cycle v suc
38       | suc == v0 = (newGraph, cycle ++ [v0])
39       | otherwise = extractCycle' newGraph (cycle ++ [suc]) suc newSuc
40     where
41       newGraph = (remove g (v,suc))
42       newSuc = head (successors newGraph suc)
43
44
45
46 -- H.4)
47 connectCycles :: (Eq a) => Path a -> Path a -> Path a
48 connectCycles x [] = x
49 connectCycles [] x = x
50 connectCycles (x:xs) (y:ys)
51   | null (x:xs) = (y:ys)
52   | x == y = (y:ys) ++ (connectCycles xs [])
53   | otherwise = [x] ++ (connectCycles xs (y:ys))
54
55 -- H.5)
56 vertexInCommon :: Eq a => Graph a -> Path a -> a
57 vertexInCommon g cycle = vertexInCommon' (vertices g) cycle
58   where
59     vertexInCommon' z (x:xs)
60       | elem x z = x
61       | otherwise = vertexInCommon' z xs
62
63
64 -- H.6) Cortesía de mi compa @Ferre18 del pazifico, zin tonteria
65 eulerianCycle :: Eq a => Graph a -> Path a
66 eulerianCycle g
67   | not (isEulerian g) = error "El grafo no es euleriano"
68   | isEmpty g = []
69   | length (vertices g) == 1 = [head (vertices g)]
70   | otherwise = eulerianCycle' g (head (vertices g))
71   where
72     eulerianCycle' g v
73       | isEmpty g = []
74       | otherwise = connectCycles cycle (eulerianCycle' newGraph newVertex)
75     where
76       cycle = snd (extractCycle g v)
77       newGraph = fst (extractCycle g v)

```

```
newVertex = vertexInCommon newGraph cycle
```