Escuela Superior Politécnica del Litoral

# Facultad de Ingeniería Electricidad y Computación

Lenguajes de Programación

# Proyecto Haskell

## I Término 2014 - 2015

Búsqueda de Vuelos

### Realizado por: Marcelo Sánchez & Raquel Villón

# Introducción

En un entorno consistente por varios aeropuertos de una región en particular, se presenta la necesidad de optimizar las rutas entre los distintos aeropuertos de la región, para esto la mejor opción es convertir todo el entorno en un modelo matemático, un grafo y usar un método informático para procesar la información de todo el entorno y luego encontrar las mejores rutas a tomar. Este proyecto fue realizado en el lenguaje funcional, Haskell, procurando obtener dichas rutas.

# Descripción y Alcance

Se implementa un motor de búsquedas de vuelos que provee al usuario la información acerca de las diversas rutas de vuelo que le permitan llegar de un lugar a otro con el menor costo posible (monetario o de tiempo, pasando por X o Y ciudades).

Para poder implementar este motor se debe conseguir los datos legítimos de información de vuelos de un o algunos aeropuerto y trabajar con esa información.

Este motor debe de ser capaz de pedir el puerto de partida y destino, opcionalmente los puertos intermedios. Adicionalmente debe de existir una manera de seleccionar qué factor tiene prioridad (tiempo o costo).

# Objetivos Alcanzados

El motor de búsqueda que logramos implementar, obtiene la información de un documento de texto plano, el cual fue previamente depurado de un \*.xls encontrado en la direccion web (http://www.dot.gov/sites/dot.gov/files/docs/Domestic%20Airline%20Airfare%20Report%20Dec%202013%20Table1.xls) el cual cuenta con los datos reales de los aeropuertos de Estados Unidos, y es diferente al proporcionado en el grupo de la materia. Este documento, contiene la información referente al reporte aéreo de los aeropuertos de Estados Unidos, en el cuarto periodo del 2013 y es la más actualizada base de datos que logramos encontrar. El documento fue depurado, quitando información innecesaria por medio de excel y puesto en texto plano para su mejor mapeo en Haskell.

Nuestro motor de búsqueda no posee una interfaz gráfica, es manejado completamente por consola y es capaz de pedirle al usuario un aeropuerto de origen y un aeropuerto de destino para luego de procesar estos datos y mostrar la información referente a la ruta de vuelo entre ambos aeropuertos: optimizando el tiempo de vuelo, distancia recorrida en kilómetros o el costo monetario del pasaje.

# Descripción de Código Importante

La implementación está en una función main, aquí se abre el documento plano, se lo pone en un contenedor y se hace un split para separar los datos y ponerlos todos en una línea.

handle <- openFile "DatasetM.txt" ReadMode

contents <- hGetContents handle

let lineas = splitOneOf ";\n" contents

Se crea las tuplas (aristas del grafo) dependiendo del filtro con las siguientes funciones que trabajan directamente con lo obtenido del dataset.

tuplasGrafoDistancias:: [String] -> [(Int,Int,Int)]

tuplasGrafoDistancias [] =[]

tuplasGrafoDistancias l = [(ciudadToNum (l!!0), ciudadToNum (l!!1), ciudadToNum (l!!2))] ++tuplasGrafoDistancias(drop 4 l)

tuplasGrafoCostos :: [String] -> [(Int,Int,Int)]

tuplasGrafoCostos [] = []

tuplasGrafoCostos l = [(ciudadToNum (l!!0), ciudadToNum (l!!1), ciudadToNum (drop 2 (l!!3)))] ++tuplasGrafoCostos(drop 4 l)

Trabajamos con grafos dirigidos y con la librería Graph.

*creaGrafo:* recibe una serie de tuplas con los valores necesarios para crear un grafo y retorna un grafo dirigido

creaGrafo :: (Ix v, Num p) => Bool -> (v,v) -> [(v,v,p)] -> Grafo v p

creaGrafo d cs vs =

accumArray

(\xs x -> xs++[x]) [] cs

((if d then []

else [(x2,(x1,p))|(x1,x2,p) <- vs, x1 /= x2]) ++

[(x1,(x2,p)) | (x1,x2,p) <- vs])

*connect*: devuelve el camino más corto entre dos puntos del grafo

connect x y g = helper x y g [x]

where

helper a b g visited

| a == b = [[]]

| otherwise = [(a,c,d):path | (c,d) <- g!a, c `notElem` visited, path <- helper c b g (c:visited)]

Una vez obtenido los caminos se procede agregarle un tupla (0,0,valor) para cada camino, donde “*valor*” es la suma del peso de las aristas en dicho camino grafo, ya sea de costo o de distancia.

Este valor es procesado luego para poder ser comparado y obtener el mínimo con el la función *minimum* de Haskell.

let camino\_mas\_corto = minimum todos\_caminos

Luego se procede a presentar resultados.

Se utilizó un parseo de Strings a Int y viceversa para mapear el dataset, el cual se encuentra en el módulo Conversiones.hs

Las librerías que utilizamos fueron:

import System.IO -- permite leer el archivo .txt

import Data.List.Split -- permite dividir el contenido leido del txt segun unos carácteres

import Data.Graph -- tiene funciones para poder sacar el recorrido minimo

import Data.Array -- crea el tipo Grafo

import Conversiones -- módulo para conversiones de Ciudades a numeros

import GHC.Float -- conversiones a Floats

# Distribución de Actividades

El proceso de creación del presente proyecto ha sido un trabajo complementario, colaborativo, pues lo que no lo comprendía el uno, el otro si. Por eso, en definitiva fuimos por partes, en cascada, e intercalando lo que nos tocó.

Marcelo se encargó de:

* obtener el dataset
* procesamiento de la información (para crear el grafo)
* presentación final de resultados

Raquel se encargó de:

* leer del dataset
* elaborar funciones para manejar la información
* la búsqueda de las ciudades dentro del grafo

# Limitaciones y Suposiciones

## Limitaciones

El programa muestra únicamente una ruta óptima. Seria bueno que mostrara otras rutas alternativas, aunque estas no sean la óptima, pero no lo hace, solo presenta la mejor opción.

## Suposiciones

El usuario debe conocer el nombre correcto de las ciudades, tanto de origen como de destino.

# Observación y Conclusiones

El lenguaje Haskell provoca un choque en la forma de pensar de un programador de lenguajes imperativos. A los desarrolladores de este proyecto nos tomó mucho tiempo y esfuerzo comprender y adaptarnos a la forma en la Haskell trabaja, la sintaxis y semántica de sus funciones.

Creemos que el fuerte de este lenguaje son las funciones recursivas y el hecho de no poder recorrer las listas como normalmente se lo hace en un lenguaje imperativo es algo que tiene sus ventajas y desventajas. Una de las ventajas es que evita el engorroso trabajo de manejar los índices. Y una desventaja es que el retorno de los resultados no es explícito, esto puede crear una confusión al momento de trabajar con esos resultados luego de llamar a la función.

# Bibliografía

#### <http://hackage.haskell.org/package/split-0.2.1.1/docs/Data-List-Split.html>

#### <http://stackoverflow.com/questions/7867723/haskell-file-reading>

#### <http://ronnyml.wordpress.com/2007/12/08/funciones-utiles-en-listas-haskell/>

#### <http://www.lcc.uma.es/~blas/pfHaskell/gentle/goodies.html>

#### <http://uqbar-wiki.org/index.php?title=Recursividad_en_Haskell>

#### <https://www.youtube.com/watch?v=my-GMplwRZw>

* [www.dot.gov/office-policy/aviation-policy/domestic-airline-consumer-airfare-report-4th-quarter-2013](http://www.dot.gov/office-policy/aviation-policy/domestic-airline-consumer-airfare-report-4th-quarter-2013)
* <http://stackoverflow.com/questions/11168238/haskell-generating-all-paths-between-nodes>
* <http://www.glc.us.es/~jalonso/vestigium/el-tipo-abstracto-de-datos-de-los-grafos-en-haskell/>
* <http://zvon.org/comp/r/ref-Haskell.html#Functions~Prelude.mapM>