**Escuela Superior Politécnica del Litoral**

**C:\Users\José Luis\Desktop\logo_fiec.jpg**

**Facultad de Ingeniería en**

**Electricidad y Computación**

**Lenguajes de Programación**

**Documentación Proyecto Haskell**

**Realizado por:**

* **Monar José**
* **Sayay Fabián**

**Paralelo:**

* **1**

**Profesor:**

* **Ing. Javier Tibau**

**I Término Académico 2014**

**GUAYAQUIL ECUADOR**

INDICE

[ALCANCE](#_Toc359531727) 3

[DESCRIPCIÓN DEL CÓDIGO](#_Toc359531739) 4

[DISTRIBUCIÓN DE ACTIVIDADES](#_Toc359531739) 9

[OBSERVACIONES Y CONCLUSIONES](#_Toc359531739) 10

# ALCANCE

Las acciones que nuestro proyecto es capaz de realizar son la de recibir por teclado una ciudad de origen y una ciudad destino, para lo cual, el usuario podrá decidir consultar la ruta más corta para llegar del origen al destino según el factor deseado, entre los disponibles en nuestro proyecto está el de consultar ruta más corta según la distancia o según el precio del vuelo, de modo que el programa devolverá una lista con las diversas ciudades por la que el usuario tendrá que recorrer para llegar a su destino, además de la distancia total que va a recorrer o costo total, según el factor deseado.

Para llevar a cabo la implementación de nuestro proyecto, primeramente se requirió el uso del lenguaje Perl para parsear la información de la data base de vuelos que obtuvimos, la cual consistió en una lista de vuelos de los Estados Unidos. Con la información de los vuelos ya parseada, se creó el módulo llamado “LoadGraph.hs ” en haskell, el cual nos permitió leer el archivo de vuelos (parseado con perl) para convertirlo en una lista de triplas y luego poder transformarlo en un Grafo.

A continuación se implementó el modulo “Dijkstra.hs”, sección más importante de nuestro proyecto, el cual permite hallar la solución dijkstra de un nodo y hallar la ruta más corta de un nodo a otro, una de las funciones que contiene este módulo es el llamado “transformGraph” el cual recibe como parámetro una lista de triplas para posteriormente transformarlo en un grafo. Luego implementamos el modulo “Peso.hs”, el cual nos permite encontrar la suma total de los pesos de la ruta más corta obtenida por la solución de dijkstra.

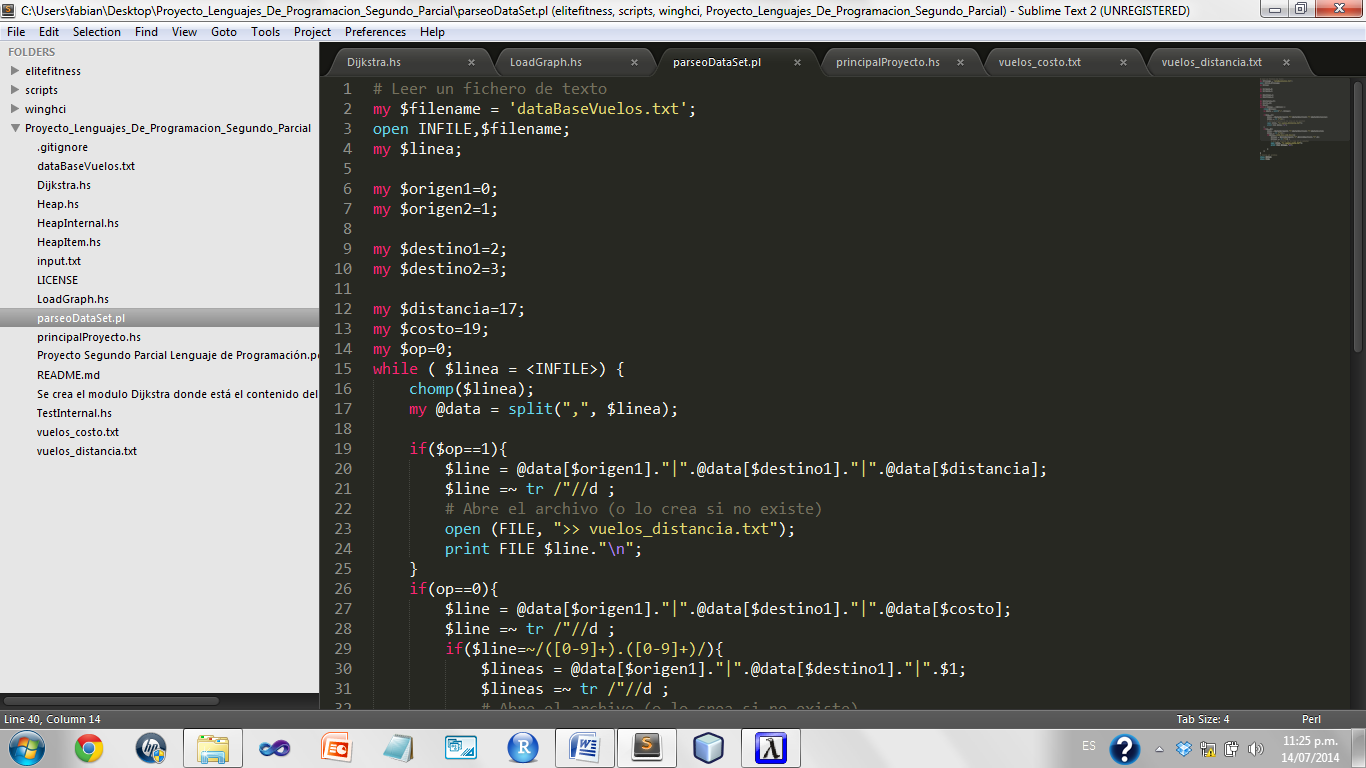
Finalmente se creó el archivo “principalProyecto.hs”, dentro del cual está implementado el “main” principal de nuestro programa, el cual será necesario ser compilado para iniciar nuestro proyecto.

# DESCRIPCIÓN DE CÓDIGO

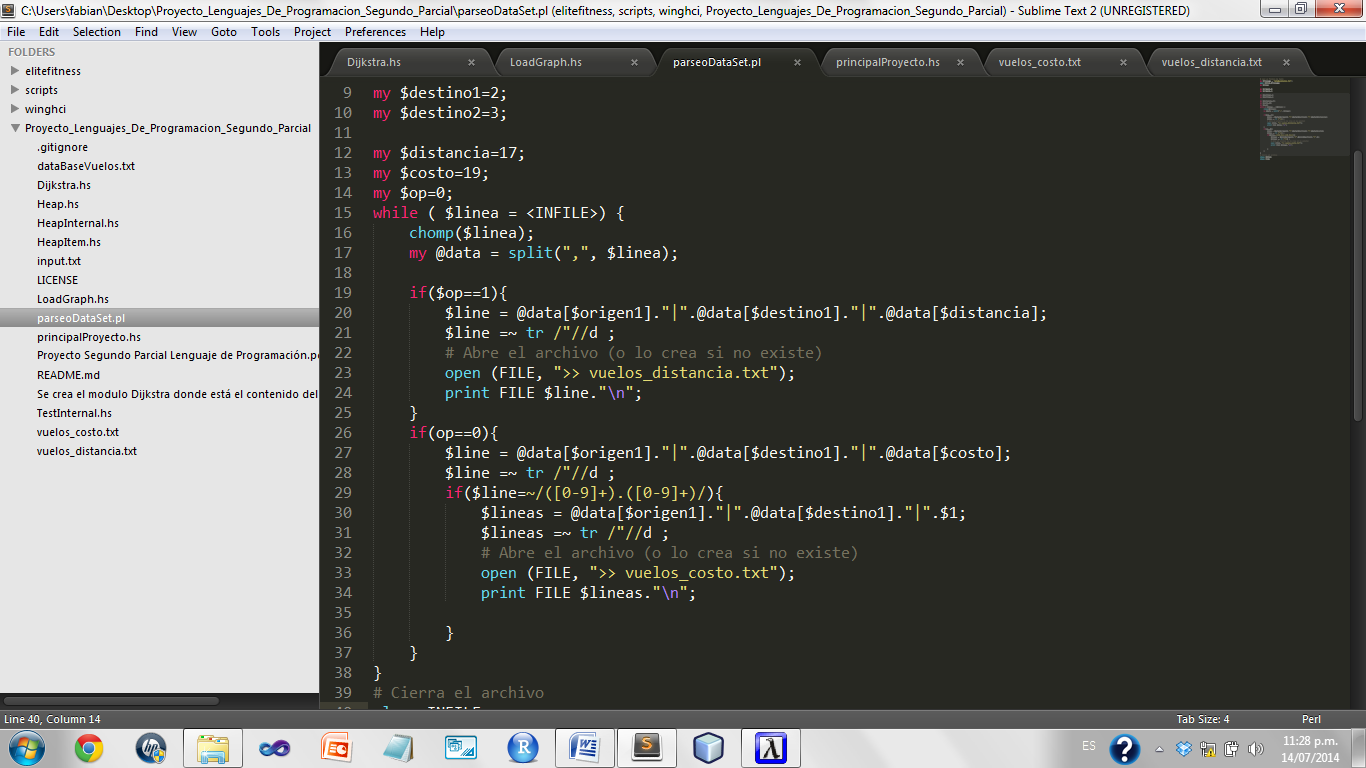
# Parseo de los datos

Para hacer el parseo de los datos que se van a usar en este proyecto se implemento un pequeño script en PERL ya que resulta más fácil trabajar con archivos en esté lenguaje.

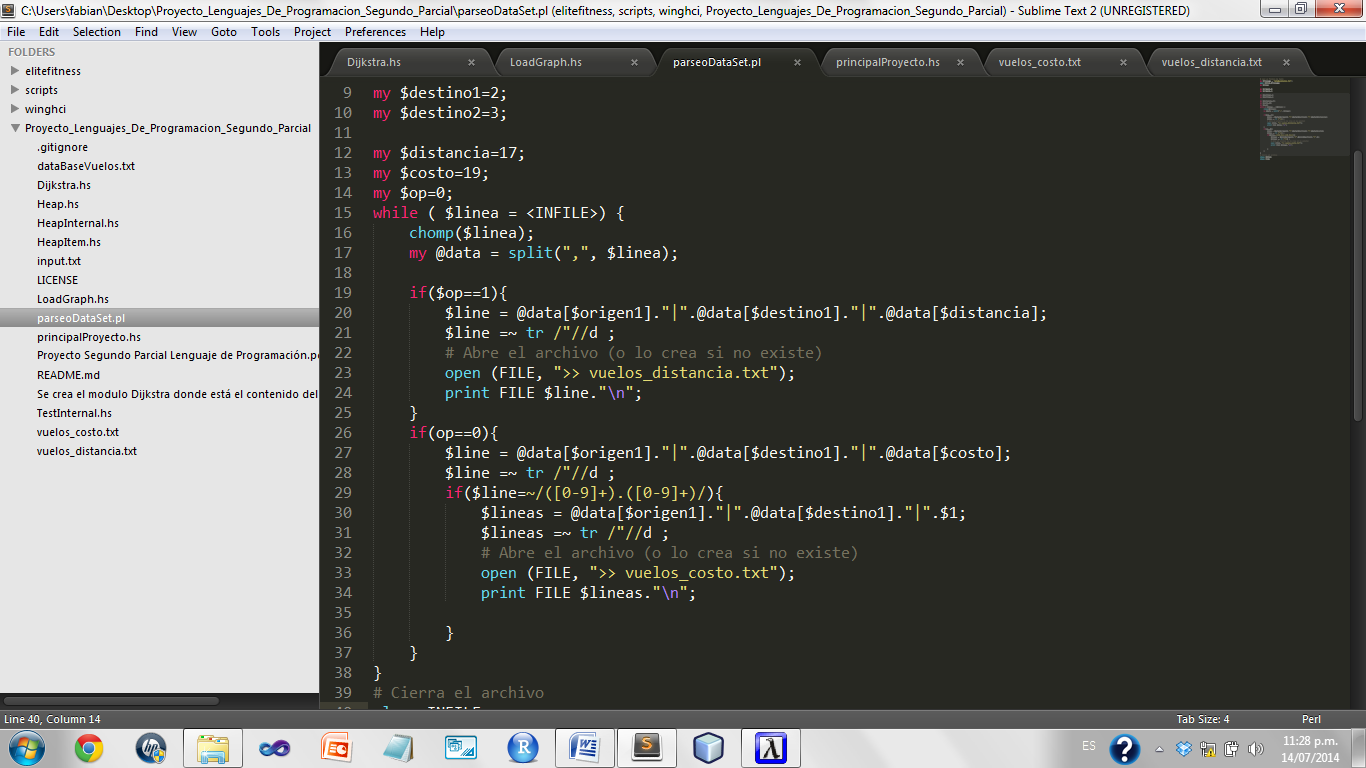
El siguiente código hace la apertura del archivo dataBaseVuelos.txt para poderlo separar en dos archivos .txt que se necesitan en el proyecto de haskell.



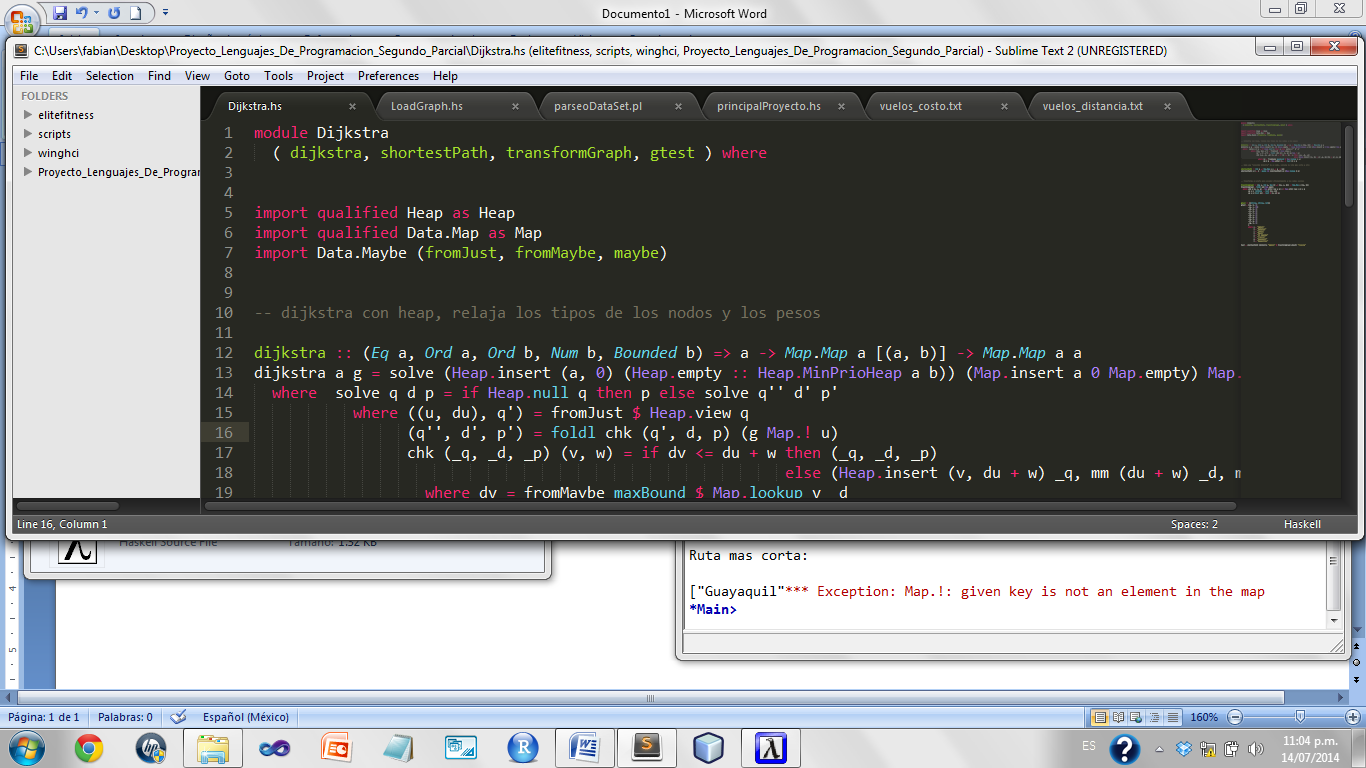
En las siguientes líneas de código, se guarda en el archivo vuelos\_distancia.txt solo datos que serán utilizados en el proyecto de haskell como son: ciudad origen, ciudad destino y la distancia.



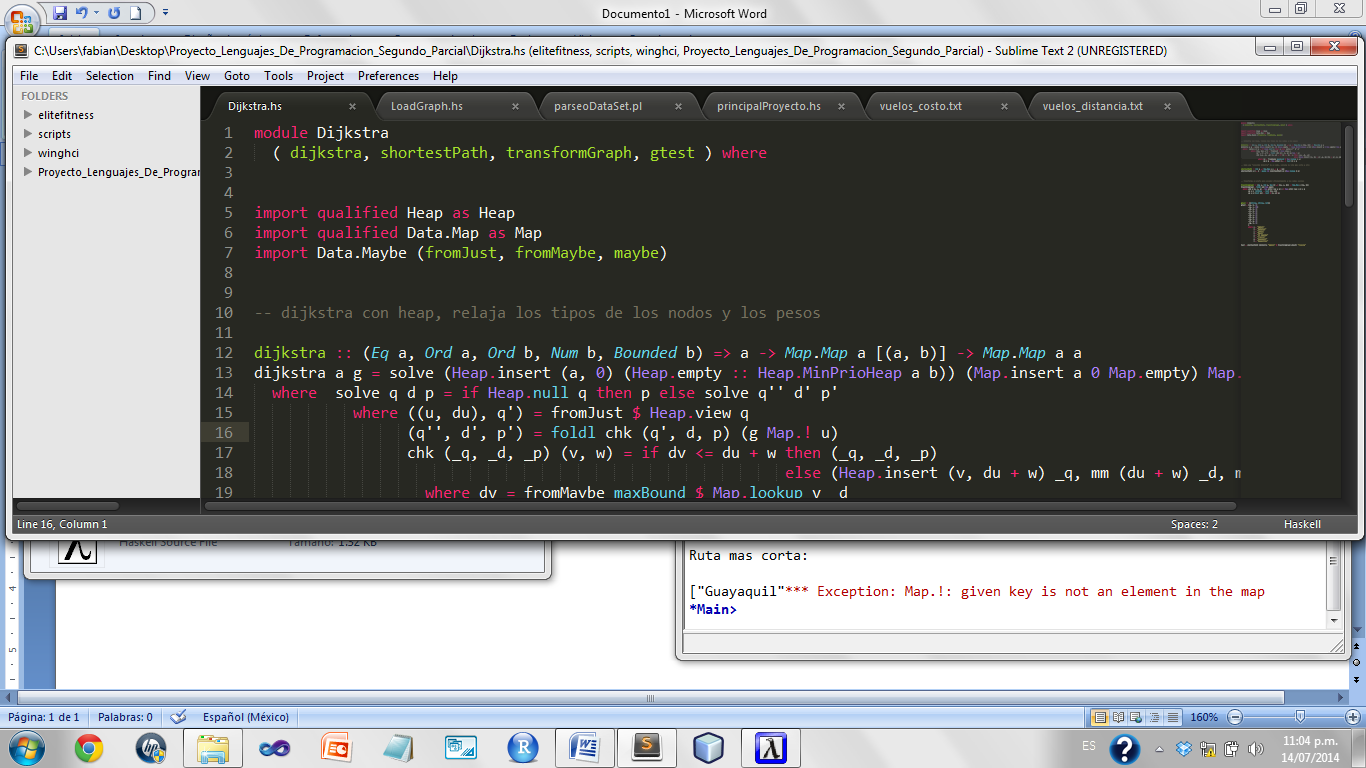
Mientras que en estas líneas de código, primero se guarda ciudad origen, ciudad destino y el costo, que serán necesarios para encontrar el costo mínimo de viajar de una ciudad a otra.

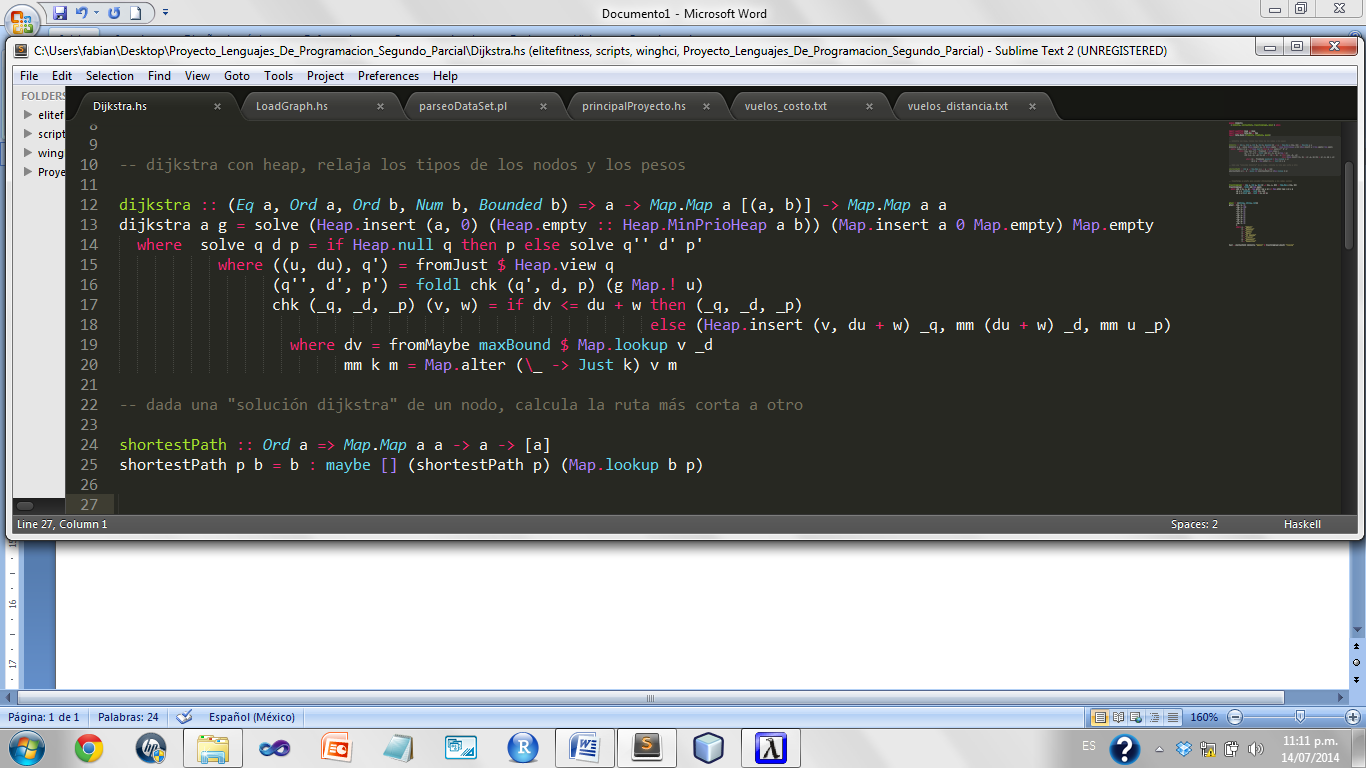


# Módulo Dijkstra.hs

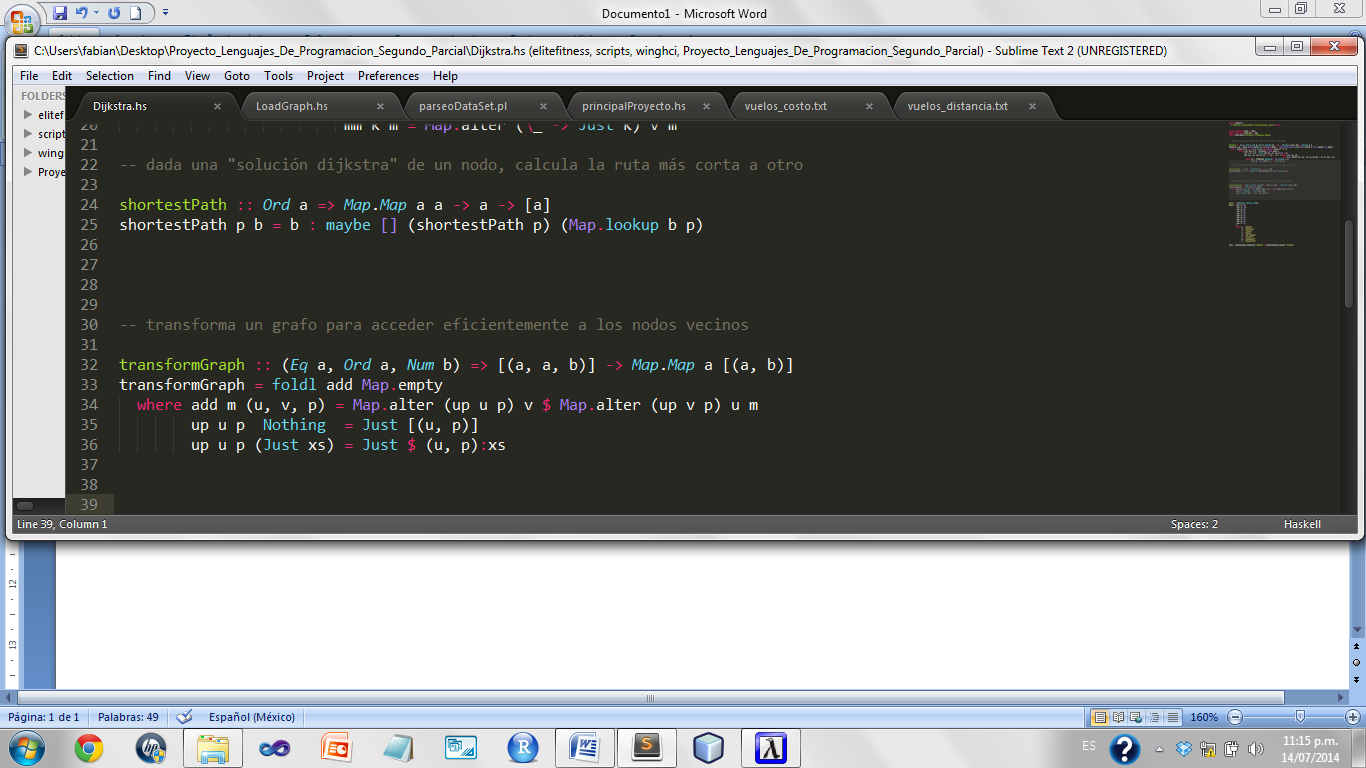
Se crea el modulo Dijkstra, donde está el contenido del algoritmo de dijkstra, para poderlo utilizar desde la parte principal del proyecto.

Se importan los módulos necesarios para el correcto funcionamiento del algoritmo.

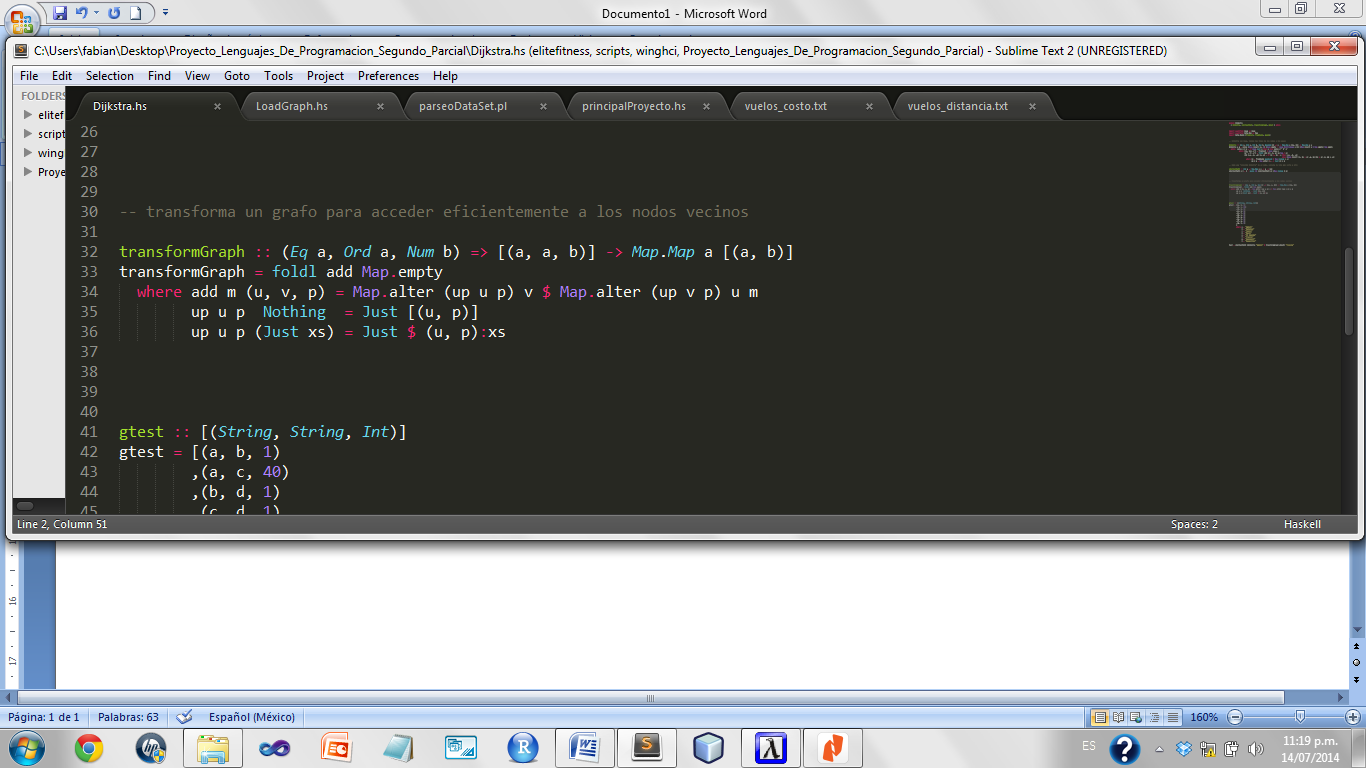


La función del algoritmo de dijkstra con Heap, Heap es un modulo que era necesario para la implementación del algoritmo. Relaja los tipos de los nodos y los pesos.

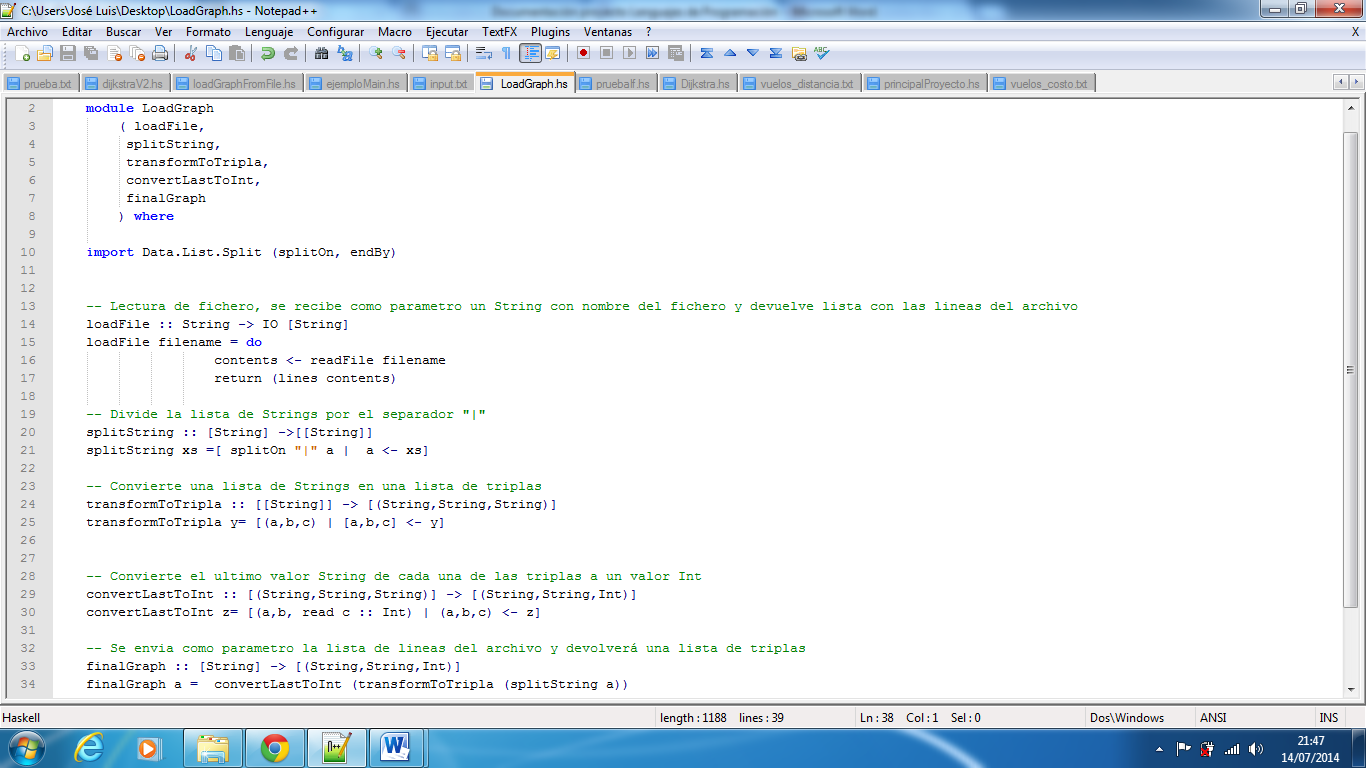
Función shortespath: Lo que hace está función es que dada una "solución dijkstra" de un nodo, calcula la ruta más corta a otro.



Función transformGraph: Trasforma un grafo para acceder eficientemente a ella.

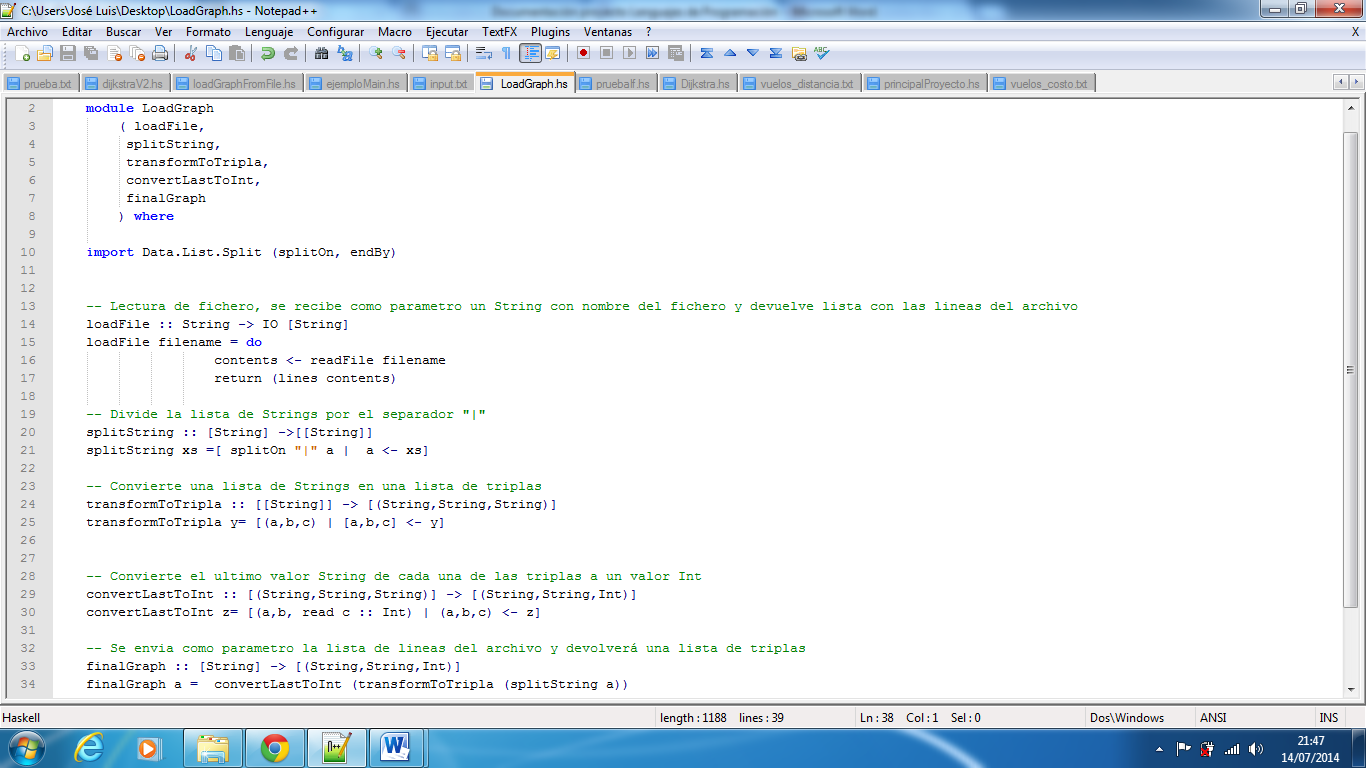


# Módulo LoadGraph.hs



La creación de este módulo fue necesaria para permitir la lectura del archivo que contenía la información de los vuelos, entre las funciones del siguiente módulo están:

**loadFile:**



El objetivo de esta función es la de leer archivo que contiene la información de vuelos y retornarla como una lista de Strings, donde cada string corresponderá a cada una de las líneas del archivo, cabe mencionar que el archivo que se va a leer (el cual fue parseado con perl) tiene la siguiente forma:

Nashville|Charlotte|328

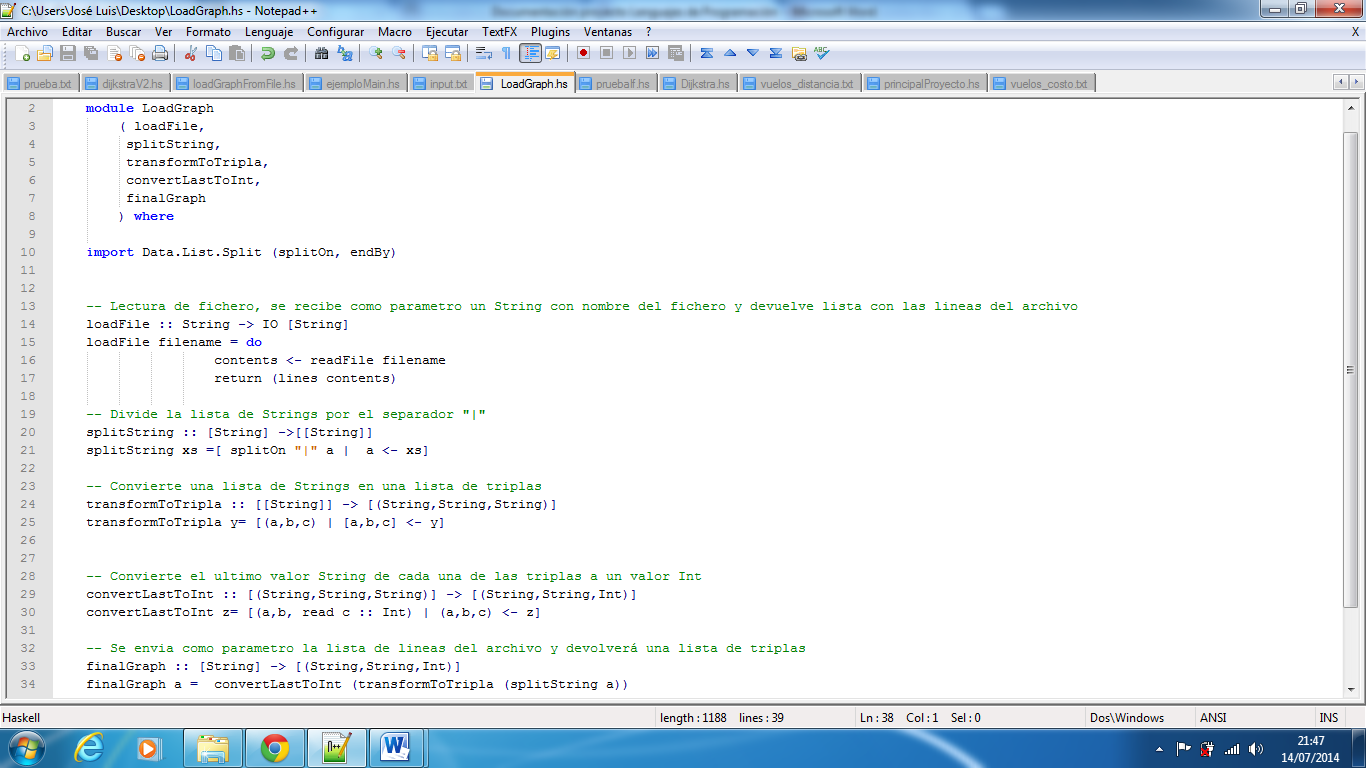
Cleveland|Washington|314

Atlanta|Greensboro/High Point|306

De modo que al usar la función loadFile, está devolverá una lista de Strings de la siguiente forma:

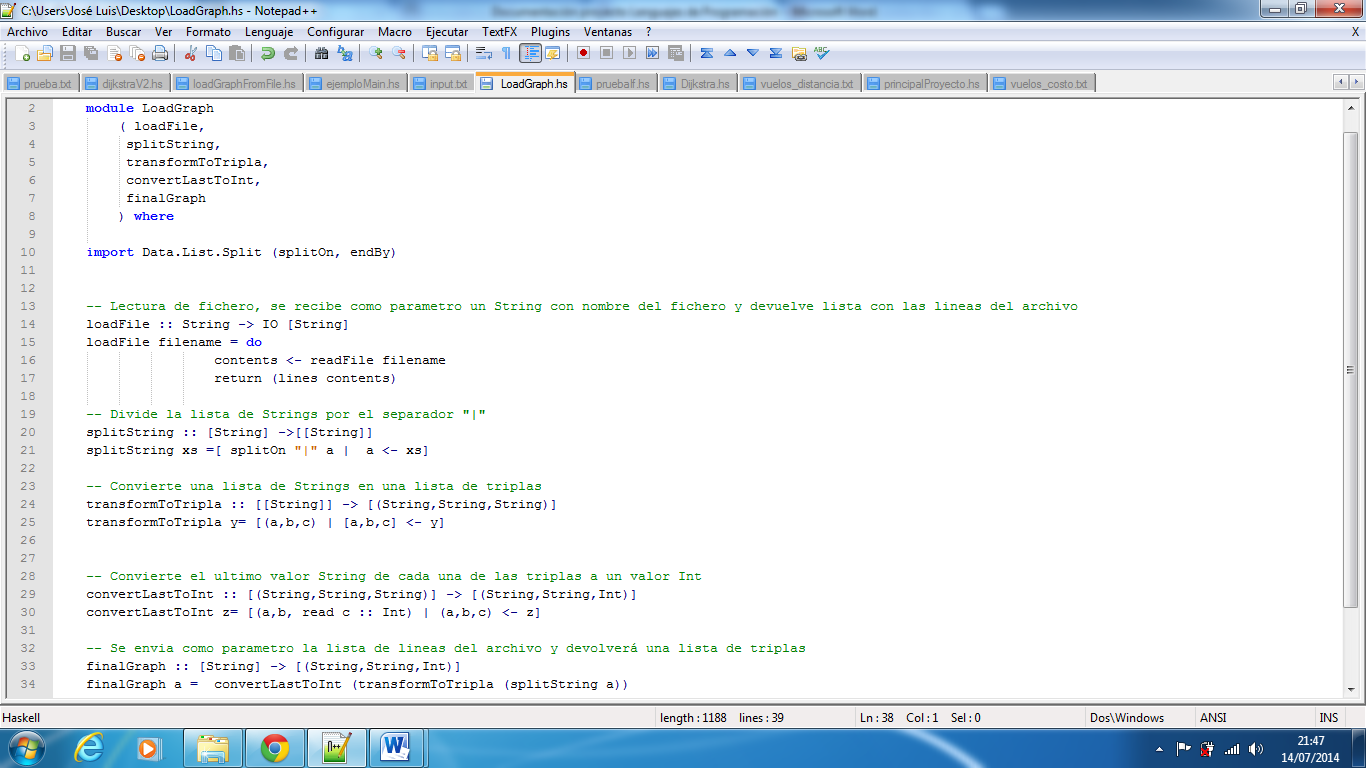
["Nashville|Charlotte|328","Cleveland|Washington|314","Atlanta|Greensboro/High Point|306"]

**finalGraph:**

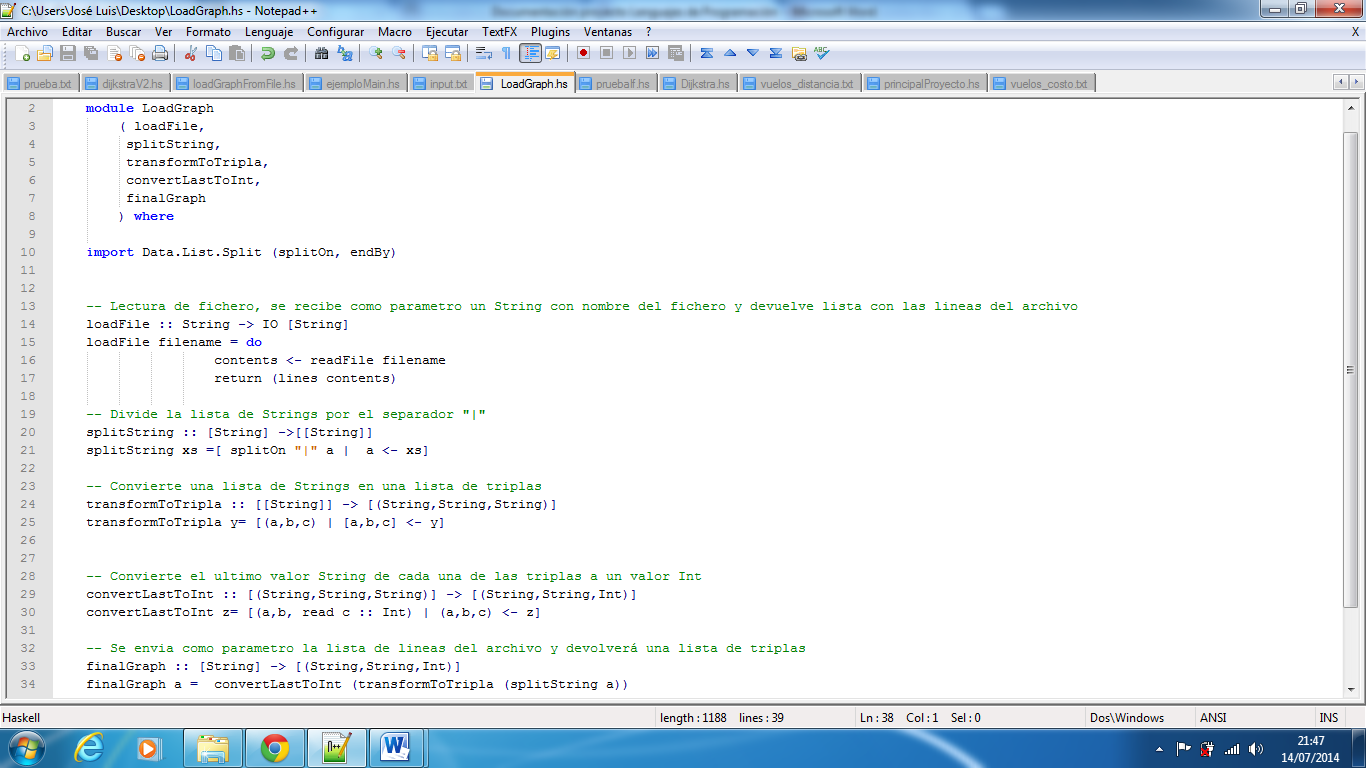


La siguiente función recibe como parámetro una lista de Strings (líneas del archivo leído) y devuelve una lista de triplas, esta conversión es necesaria para poder transformarlo a un grafo, esta debe tener la siguiente forma:

[(String, String, Int)]

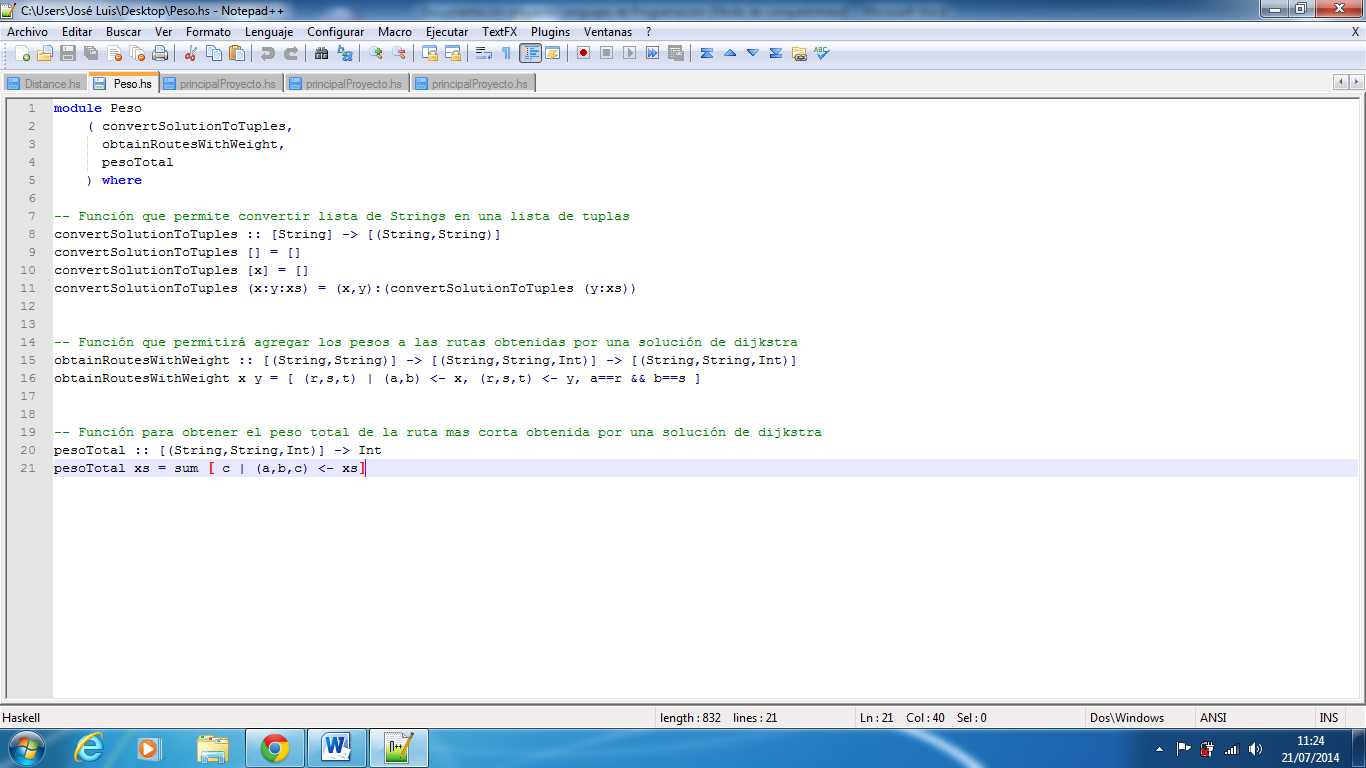


Para hacer la conversión se hace uso de la función “splitSpring” la cual recibe como parámetro una lista de Strings y devuelve una lista de listas de Strings, donde cada uno de estos strings serán debido a la división por el separador “|”. A continuación se utiliza la función “transformToTripla” para transformar la lista de listas de Strings a una lista de triplas de String, String y String.



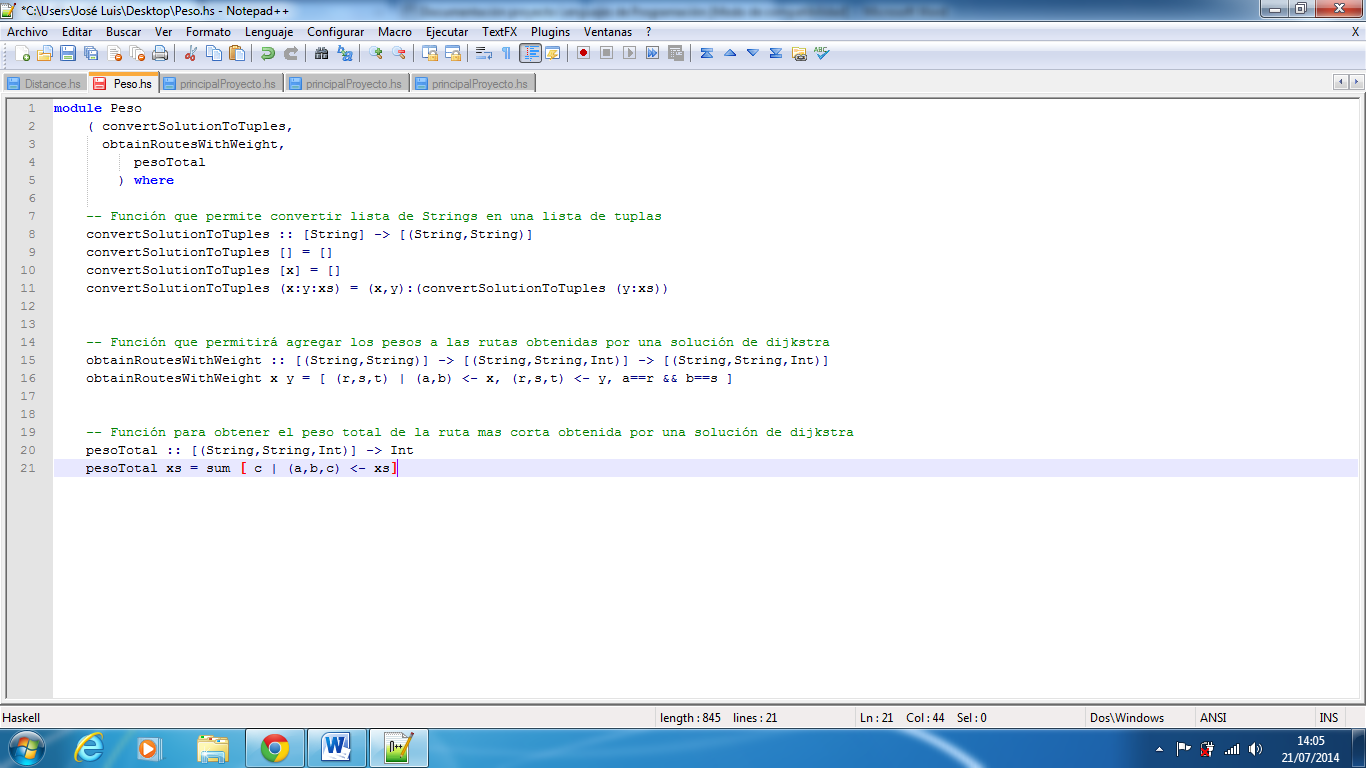
Finalmente, es necesario que el último valor de la tripla sea un Int, para lo cual se hace uso de la función “convertLastToInt”, el cual hace la conversión del último valor, de cada una de las triplas, de String a Int.

# Módulo Peso.hs



Se implementó el siguiente módulo, el cual contiene las siguientes funciones:

**convertSolutionToTuples:**



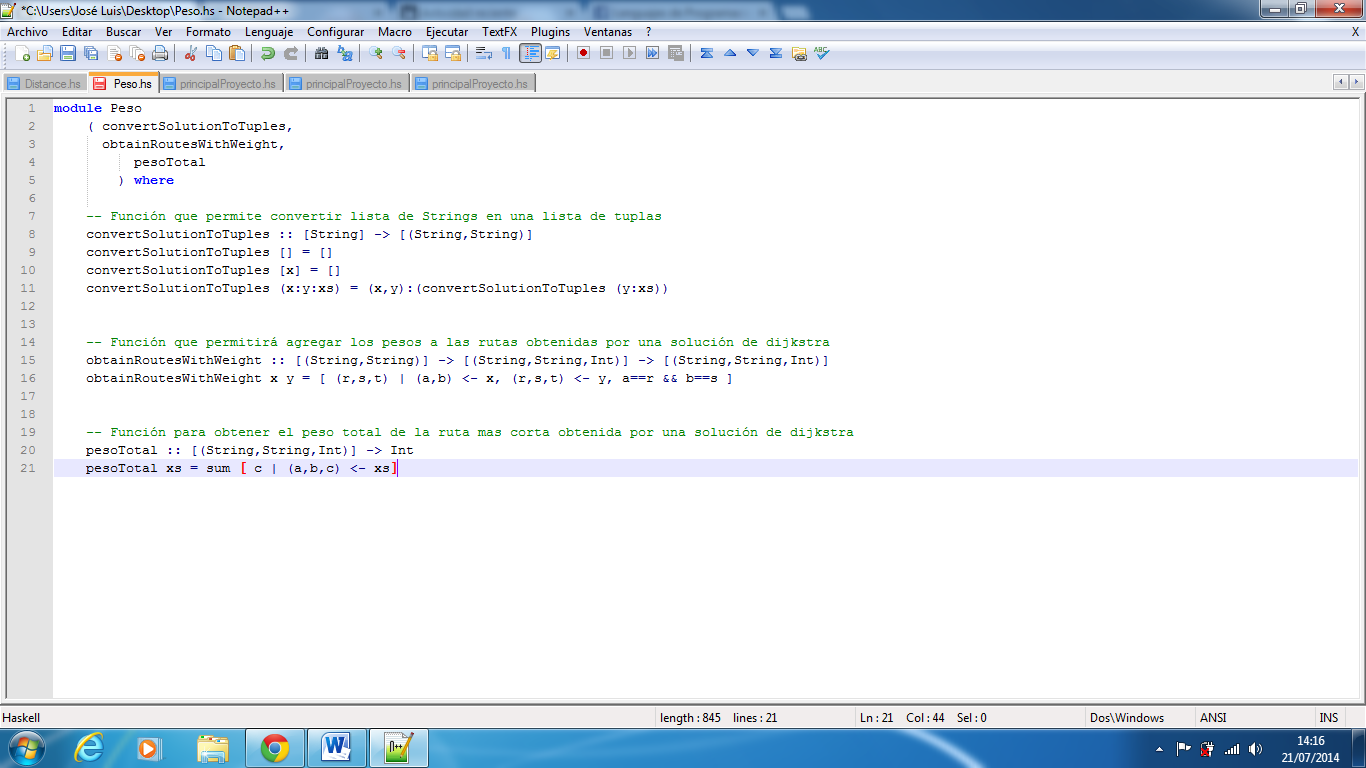
Esta función permite convertir una lista de Strings en una lista de tuplas, es la función que utilizaremos para convertir la lista de Strings (resultados) obtenidos por una solución dijkstra, a una lista de tuplas, la conversión sería de la siguiente manera:”"

Se tiene lista de strings [“A”,”B”,”C”,”D”,”E”] y el resultado sería una lista de la siguiente manera:

[(“A”,”B”),(“B”,”C”),(“C,”D”),(“D”,”E”)]

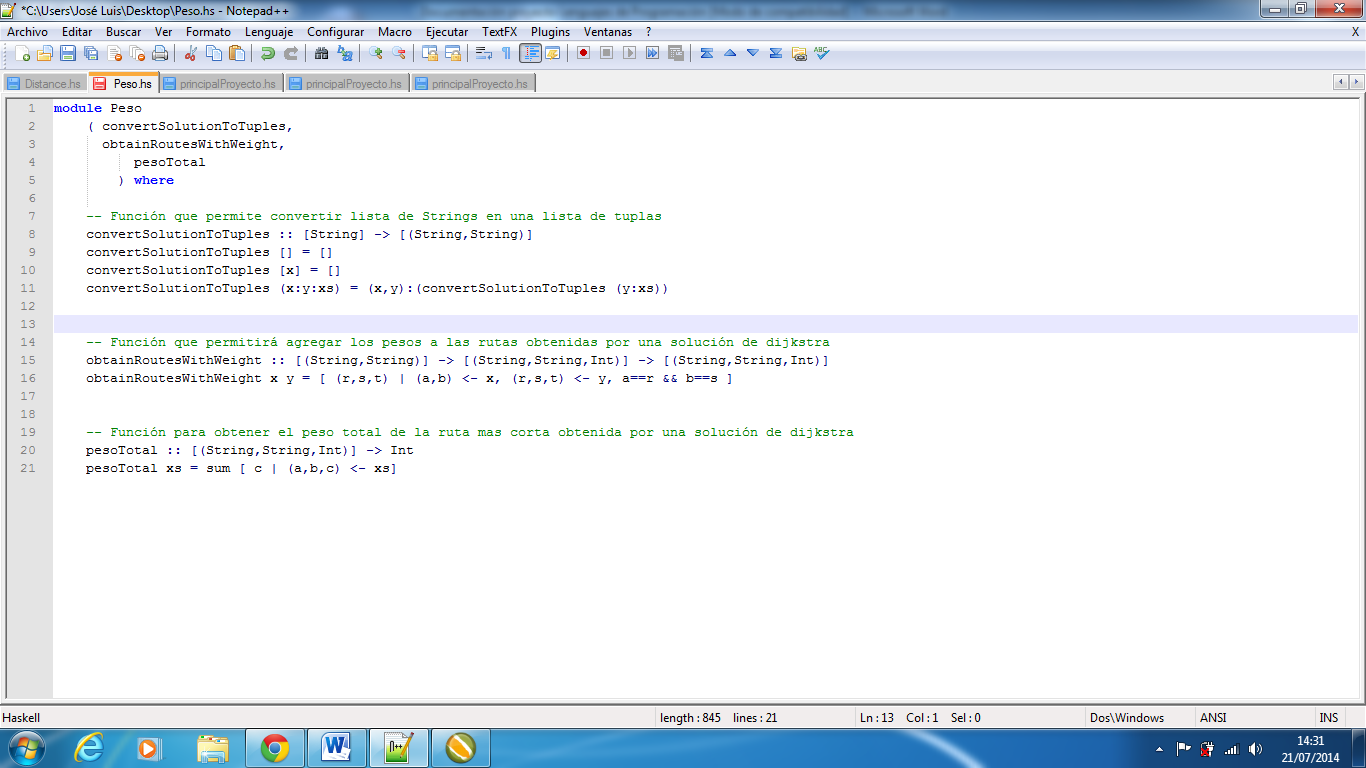
De esta forma obtendré el resultado de la solución de dijkstra como una lista de arcos, con origen y destino en cada tupla, sin embargo, aún hace falta tener el peso de cada arco, para ello usaremos la función que se detalla a continuación.

**obtainRoutesWithWeight:**



Esta función permite encontrar el peso de cada arco resultante en el grafo, para ello le enviaremos como parámetro una lista de tuplas, la cual será la obtenida por la función “convertToTuples”, y una lista de triplas, la cual corresponderá al grafo. Finalmente la función retornará una lista de triplas, correspondiente a la lista de arcos con su peso incluido.

**pesoTotal:**

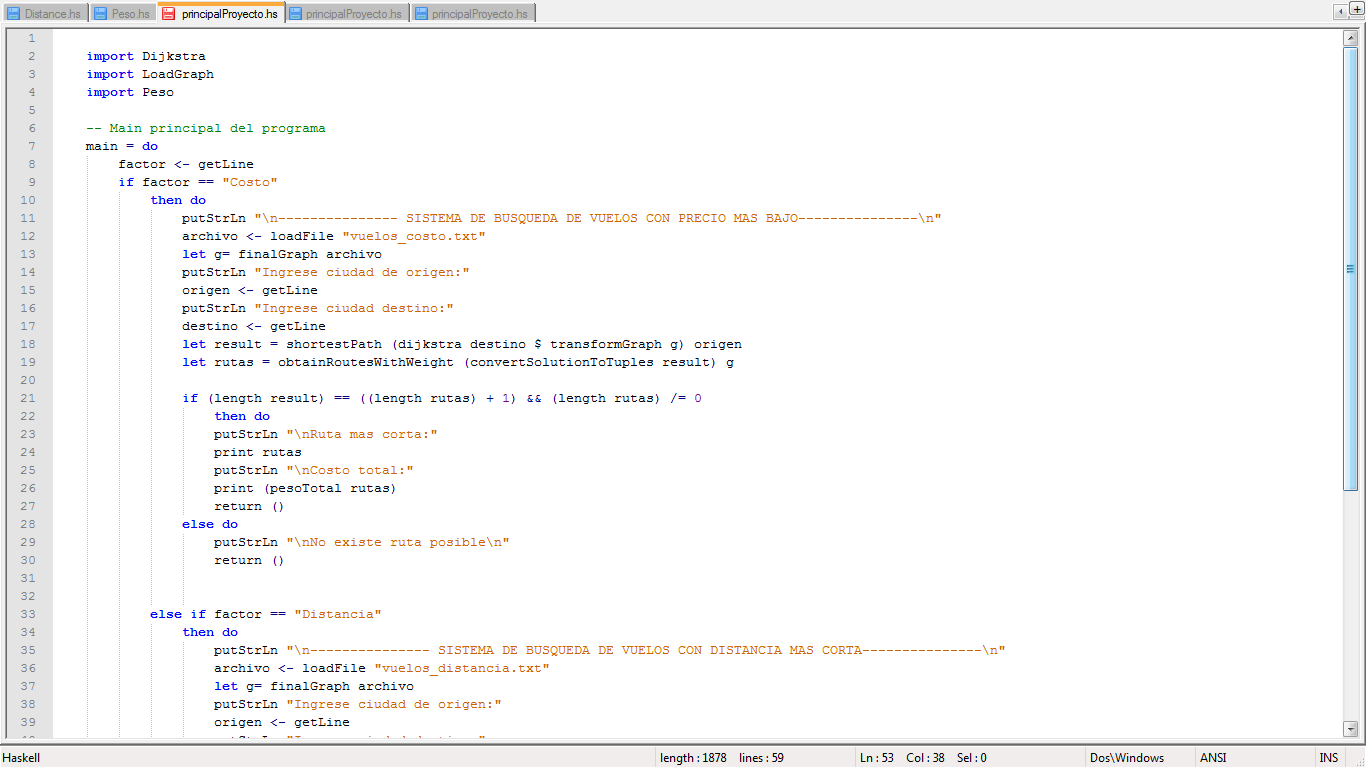


Finalmente el módulo contiene esta función la cual recibe como parámetro una lista de triplas, que corresponde a la lista de arcos con pesos resultantes, y devolverá la suma de los pesos, un ejemplo del uso de la función es el siguiente:

Se envía como parámetro la lista [(“A”,”B”,3),(“B”,”C”,7),(“C”,”D”,2)]

Se obtendrá como resultado la suma de los pesos: *12*

**Main principal del proyecto**



El siguiete archivo haskell será el que debe ser compilado para que funcione el programa, importa los modulos “Dijkstra.hs”, “LoadGraph” y “Peso.hs” para poder utilizar sus funciones.

El siguiente código lo que hace primero es pedir por teclado factor de búsqueda para hallar la ruta más corta, los factores pueden ser “Costo” o “Distancia”. Según la decisión del usuario, se seguirá el flujo de instrucciones, ingresando ya sea a “then do” o a “else do”, leyendo a continuación el archivo de datos correspondiente y guardándolo en “archivo”. Luego el programa pedirá por teclado una ciudad de origen y una ciudad de destino, lo valores ingresados serán almacenados en “origen” y “destino” y serán enviados como parámetros a la función “shortestPath” para hallar la ruta más corta, finalmente imprimirá una lista con los resultados incluyendo la suma total de distancia o precio, según sea el caso.

**Distribución de Actividades.**

El presente proyecto ha sido todo un reto para nosotros, ya que no es un lenguaje tan popular no existen muchos tutoriales, por lo que decidimos repartirnos el proyecto para poder avanzar lo más rápido posible. Nuestro primer reto fue encontrar un algoritmo implementado en haskell y que estuviese documentado pero no se encontraron muchos y los que se encontraron no tenían la documentación suficiente para poderlo entender.

Después de encontrar la implementación del algoritmo de dijkstra en esté lenguaje, tuvimos que leer el libro de haskell para poder entender lo que estaba haciendo en cada línea de código, después de largas noches de lectura ya sabíamos lo que necesitábamos para que funcione el algoritmo y como implementarlo en haskell, luego decidimos repartir el proyecto en dos partes, lo que muestra el algoritmo y lo que necesita para mostrarlo.

**Lo que muestra el algoritmo:** José Monar se encargó de ésta parte de la implementación en la que implemento el modulo LoadGraph.hs, Pesp.hs y principalProyecto.hs donde está como se presenta por pantalla.

**Lo que necesita para mostrarlo:** Fabián Sayay se encargó de implementar el modulo dijkstra.hs y parseoDataSet.pl que nos ayudo mucho para tener un data set solo con los datos que son necesarios para el proyecto.

**OBSERVACIONES Y CONCLUSIONES**

* Para iniciar el programa se debe compilar el archivo “principalProyecto.hs”, luego llamar al método main escribiendo “main” en la línea de comando, aparecerá una línea en blanco la cual nos indica que debemos ingresar un valor por teclado, los valores aceptados serán “Distancia” o “Costo”, si se ingresa un valor diferente entonces volverá a pedir que se ingrese un valor.
* Es importante tener los archivos “Heap.hs”, “HeapInternal.hs”, “HeapItem.hs” y “TestInternal.hs” en la misma carpeta donde se encuentra el módulo de Dijkstra para que el programa corra sin problemas.
* El grupo se valió de la página <http://aprendehaskell.es/> para aprender lo básico sobre el lenguaje Haskell, además de videos tutoriales que ayudaron a comprender mejor el uso de este lenguaje.