UNIVERSIDAD NACIONAL DE GENERAL SARMIENTO

Trabajo Práctico 2

Integrante:

Ezequiel Tomas Ramallo

Legajo: 36827376/2017

E-mail: ezeramallo@gmail.com

Profesores:

Patricia Bagnes

Javier Marenco

Asignatura: Programación III – Primer Cuatrimestre 2020

Comisión: COM-1

Fecha de entrega: 12/05/2020

Introducción

El presente informe tiene como objetivo exponer, de forma breve, el desarrollo de una aplicación que permitirá planificar conexiones telefónicas entre localidades despobladas, la cual consiste en calcular, luego del ingreso de las coordenadas de varias localidades, el costo mínimo de una red de conexiones. Este costo se calculara bajo las siguientes condiciones: relaciones entre costo-km, porcentaje de aumento entre una conexión si la distancia entre la localidades de esta conexión supera los 300km y un costo fijo agregado si la conexión si establece entre localidades de dos provincias distintas.

Al iniciar la app ,el usuario deberá ingresar los costos para los cuales se va a calcular la red mínima de una conexión de localidades y, evidentemente , las localidades que establecerán las conexiones. Las localidades se ingresaran en un apartado accedido mediante el botón “Actualizar Localidades” ubicado en el esquina inferior derecha de la ventana principal. Estas Localidades se identificaran con 3 valores: el nombre de la localidad, el provincia y sus coordenadas de ubicación geográficas. Las coordenadas asignadas a estas localidades tendrán un rango de elección de números en representación de punto flotante que ira de -90 a 90 para latitud y de -180 a 180 para longitud, ya que estas son los rango de representación cartesiana para estas medidas. Ademas, vale agregar, que así como es posible que el usuario ingrese una localidad, también este podrá eliminar alguna de ellas de ser necesario.

Una vez ingresados todos los valores necesarios para realizar el calculo, este se llamara a través del frame principal mediante el botón “calcular costo mínimo”;ademas , el frame principal, brindara la opción de visualizar la conexión de localidades en el mapa mediante el uso del botón “Dibujar conexiones”

Implementación

Para llevar a cabo implementación de esta aplicación decidimos utilizar el patrón MVC (modelo-vista-controlador), el cual nos permite separar el código de negocio del código de la interfaz. Para llevar a cabo la implementación de este patrón, decidimos dividir el proyecto en 5 package: codNegocio, Grafo (auxiliar del package codNegocio), Controlador, Vista y MVCPrincipal.

- ## Paquete Grafo (paquete con clases que utilizaremos en el package codNegocio)
  - **Clase "Coordenadas"**: En esta clase encontraremos la representación en Double de las coordenadas Latitud y Longitud que serán asignadas a las localidades creadas.
  - **Clase "UnionFind"**: clase auxiliar creada para reducir la complejidad del algoritmo kruskal, el cual sera utilizado para calcular el árbol generador mínimo entre localidades.
  - **Clase "vértice" :** clase padre de la clase localidad.
  - **Clase "arista" :** esta clase representa la conexión entre 2 instancia de La clase localidad. Los atributos de esta clase serán ,obviamente, estas 2 localidades que representa los extremos de la conexión y una instancia BigDecimal que representara el peso de la conexión calculado bajo los parámetros de valores pedidos el frame principal (el calculo del valores del peso lo hace un método de una clase del package del codNegocio).
  - **Clase "Grafo":** clase padre de la clase RedTelefonica.
  - **Clase "Localidad":** Esta clase contendrá los siguientes atributos: dos string (nombre de localidad y nombre de provincia), una instancia de la clase coordenadas que nos permitirá identificar la ubicación de esta localidad en el mapa y, por ultimo, un identificador numérico (entero) que nos permitirá realizar algoritmos necesarios para realizar la aplicación (p. Ej. Kruskal).
  - **Clase "RedTelefonica":** Esta clase contiene la mayoría de los métodos que nos podemos encontrar en un clase Grafo (agregar o eliminar una arista o vértice, o bien, preguntar si alguno de los dos esta contenido en el grafo), con la salvedad de que en esta clase, esos métodos, serán adaptados a los atributos de la clase localidad, en otras palabras, la clase Localidad tomara el lugar de la clase Vértice a la hora de la implementación de las funciones. Ademas de estos métodos en común con su clase padre, esta clase contendrá los siguientes métodos importantes:
    - ListaAristasOrdenadasPorPeso: método utilizado para ordenar de menor a mayor el peso de la aristas a fin de realizar el algoritmo de kruskal
    - Kruskal: Este método, el cual fue adaptado a los atributos de la clase Localidad, nos permite crear un árbol generador mínimo entre las conexiones de las localidades contenidas en la red telefónica.

- ## Paquete CodNegocio
  - **Clase “DistanciaGeografica”**: Esta clase contiene un solo método estático que nos permite calcular la distancia en km a partir de coordenadas geográficas.
  - **Clase “Negocio”:** Clase encargada de la inteligencia de la aplicación, en ella utilizaremos 4 variables de instancia, 3 de estas variable son instancias de BigDecimal, las cuales se encargara de almacenar los valores con los cuales se calcularan los pesos de las conexiones, y la variable restante es un instancia de la clase RedTelefonica, la cual se encargara de almacenar las localidades y sus conexiones, ademas de ayudarnos a
    - agregarLocalidad: ingresa una localidad a la RedTelefonica, luego de chequear que los parámetro pasados para su ingreso cumplan con las siguientes requisitos:
      - Los string que representan los nombre de localidad y provincial solo pueden contener letras y espacios
      - Los parámetros no pueden contener una cadena vaciá
      - Los string que representan la latitud y longitud deben ser números capaces de parsear a double, y respetar los siguientes rangos:
        - [-90,90] para latitud
        - [-180,180] para longitud.
      - La coordenadas no pueden ser igual a una ya ingresada, pese a que el nombre de la localidad y/o provincia sean diferentes
    - setEstablecerValores: Este método se encarga de asignarle los valores recibidos por parámetro a las 3 variables de instancia BigDecimal. Antes de esta asignación, este método chequea lo siguiente:
      - No debe haber ningún parámetro vació.
      - Todo los parámetros deben ser positivos y de valores numérico
    - calculoConexion: Este método calcula el valor del peso de una conexión entre dos localidades en base al valor de las 3 variables de instancia BigDecimal. Este método actuá como auxiliar en el método establecer_conexiones.
    - establecer_conexiones: establece todas las conexiones posibles entre las localidades ingresadas, es decir, arma un grafo completo. Para calcular el peso en el armado de conexiones utiliza el método calculoConexion.
    - camino_mínimo: devuelve la lista de las conexiones del árbol generador mínimo producido por el grafo completo generado en el método establecer_conexiones

- costoMInimoTotal: devuelve el valor mínimo por el cual se puede generar de una red de conexiones telefónicas

• Paquete Controlador

- **clase “Controlador”**: esta clase implementa la interfaces de ActionListener, MouseListener y FocusListener, y se encarga de funcionar como una especie de intermediario entre el código de interfaz y el código negocio; es decir, mediante la reacción a un evento producido en el interfaz grafica, esta clase de encargar de actualizar y/o procesar el código de negocio para la posterior actualización de la parte grafica.. En ella tendremos tres variables, dos de ella harán referencia tanto a la clase encargada del código de inteligencia como a la clase del código encargado de la parte grafica; mientras que la restante, se encarga de habilitar el dibujar y eliminar de las conexiones en el mapa(alterna el dibujar y el eliminar). Las funcionalidades que cumplen serán:
  - pesoCostoMinimo: Este método se encargar de calcular el costo mínimo de una conexión tomando los valores ingresados en la interfaz grafica para procesarlos en el código de negocio.
  - DibujarConexiones: Dibuja en el mapa de la interfaz grafica la conexión mínima necesaria de una red telefónica tomando las conexiones generadas por el algoritmo de kruskal realizado en el código de negocio
  - EliminarConexiones: borra la conexión plasmada en mapa de la interfaz grafica.
  - agregarLocalidad: agrega una localidad al código de negocio y a una lista de la interfaz grafica, la cual nos permitirá chequear los valores ingresados y eliminarla en caso de ser necesario.
  - EliminarLocalidad: elimina una localidad ingresada, tanto en el código de negocio como en la parte grafica, mediante la selección de ella en la lista de la interfaz grafica
  - salirVntModal: cierra la ventada modal, por lo cual ingresamos o eliminamos Localidades.
  - actionPerformed: Este método ejecutara un funcionalidad diferente dependiente del tipo de parámetro que reciba. Los método que puede ejecutar, son los descriptos anteriormente (pesoCostoMinimo, DibujarConexiones, EliminarConexiones, agregarLocalidad, EliminarLocalidad, salirVntModal)
  - mouseClicked: Muestra el los labels de la ventana modal los datos de la localidad seleccionada en la lista ubicada en esa misma ventana.

        - focusLost: nos permite deseleccionar un elemento de la lista ubicada en la ventana modal (esto implica limpiar los labels de la ventana modal que muestran los datos de la localidad seleccionada en la lista).

- ## Paquete Vista
    - **clase "LocalidadListModel"**: esta clase, que extiende a AbstractListModel, fue creada para poder tomar y eliminar instancias de la clase Localidad de una lista.
    - **clase "GUI_Localidades"**: esta clase sera utilizada como ventana modal . En ella tendremos componentes como JtxtField, Jbutton, JLabel y Jlist que nos permitirán ingresar o eliminar localidades a fin de procesarlas en el código de negocio de la aplicación. Sus métodos mas importantes son:
        - VentanaEmergente: Nos permite mostrar un mensaje pasado por parámetro a través de un JoptionPane. Usamos esta clase para mostrar, en la ventana modal, los mensajes que especifican las excepciones generadas por el ingreso de algún valor incorrecto.
        - AgregarComponentes:Crea todos los componentes usados en la ventana modal.
    - **clase "GuiApp"**: En esta clase se encuentra implementada la ventana principal de la aplicación , la cual contendrá componentes como Jbutton, JtxtField,Jlabel y JmapViewer, los cuales nos permitirán acceder, a través de su interacción con el código de negocio, al costo mínimo de una red telefónica, a visualizar o no en el mapa la conexión mínima de una red telefónica e a ingresar a una ventana modal para ingresar o eliminar localidades. Los métodos mas importantes de esta clase son:
        - AgregarPaneles: inicializa los paneles a usar en el GUI.
        - AgregarComponentes: inicializar los componentes a usar en el GUI.
        - MostrarVentanaEmergente: Cumple la misma función que el método VentanaEmergente de la clase GUI_Localidades.
        - GUI: este método se encarga de distribuir y acoplar los paneles de componentes en el frame, así como también de activar la visualización de la aplicación.
        - CreaGUI: método encargado de inicial la aplicación

- ## Paquete MVCPrincipal
    - **Clase "Principal":** es la clase que contendrá el man utilizado para aunar las partes del modelo, vista y controlador, a fin de lanzar la aplicación.

Problemas durante el desarrollo y decisiones tomadas.

- Al igual que en el TP1, tuve inconvenientes al implementar el patrón MVC con la extensión windowBuilder (Imposibilita el setteo del código de negocio y de vista en controlador debido al main del “aplication window”) por lo que se decidió quitar el main predeterminado de la “aplication window” de la GuiApp e implementarlo en una clase aparte.