Informe de Documentación del Trabajo Práctico N°2: Conectando Localidades

Comisión: 02 (Mañana)

Alumnos:

-Trejo Fernando (43.986.607)

-García Matías (41.385.966)

-Gordillo Fabricio (42.948.240)

Objetivos:

Se nos encargó realizar una aplicación con la cual el usuario pueda planificar conexiones telefónicas entre localidades ubicadas en regiones despobladas. Dado un conjunto de localidades debíamos proporcionar un árbol generador mínimo entre ellas e informar el costo del mismo.

Dificultades:

En esta re-entrega nos pudimos dar cuenta de que habíamos entendido mal la consigna ya que se pedía un AGM de todas las localidades y habíamos hecho un camino mínimo entre localidades.

Lo que más nos costó a la hora de volverlo a hacer fue el hecho de que habíamos trabajado en otra cosa totalmente distinta y tuvimos que empezar de 0 con el algoritmo de primera.

Siguiendo las indicaciones de Daniel:

- Construimos la clase ConectorLocalidades para poder unir 2 localidades

- Cambiamos el nombre de las clases que estaban en plural a singular esto debido al hecho de hacer buenas practicas

-Agregamos para que el usuario pueda poner los costos por km, más de 300k y por provincias.

-La clase verificaciones la eliminamos y agregamos esos métodos al Menu ya que eran más verificaciones de la interfaz que otra cosa

-Agregamos test para todas las clases

-Revisamos la clase que no existía que ahora sí, que era la clase Main

Sin dudas lo que nos llevamos de este trabajo es a prestar un poco más atención y consultar si creemos lo que estamos haciendo, estamos bien encaminados.

Implementación:

Clase Main: Clase principal de la aplicación.

Inicializa y crea un objeto Menú

Clase Menu: Clase de la interfaz gráfica de la aplicación.

Atributos:

- JFrame frame: Objeto de la interfaz.

- JTextField textFieldMatrizTamanio: Objeto de la interfaz.

- JTextField textFieldNombre: Objeto de la interfaz.

- JTextField textFieldProvincia: Objeto de la interfaz.

- JTextField textFieldLatitud: Objeto de la interfaz.

- JTextField textFieldLongitud: Objeto de la interfaz.

- Verificaciones verificar: Objeto de la clase Verificaciones que utilizamos en distintos casos para aprovechar algún método creado en esta clase.

- JMapViewer mapa: Objeto del tipo mapa usado por la

interfaz. - Color colorFondo: Determina el color del fondo.

- Color colorLetra: Determina el color de la letra.

- Color colorVerde: Deja guardado una variante de color verde. - Color

colorAmarilloClaro: Deja guardado una variante de color amarillo. - Font

fuente: Determina la fuente utilizada.

-Administracion administracion: Va a inicializar el objeto sistema

- JTextField CostoPorKm: Objeto de la interfaz.

- JTextField Costo300Km: Objeto de la interfaz.

- JTextField CostoPorvDistintas: Objeto de la interfaz. distintas

-JLabel labelPrecio: Objeto de la interfaz

Métodos:

- void labelPreguntaLocalidades(): Pregunta al usuario la cantidad de localidades a utilizar.

- void textFieldNumeroDeLocalidades(): Casilla en la que el usuario escribe la cantidad de localidades.

- void limpiarPantalla(): Limpia la pantalla de la interfaz.

- void crearBotonListo(): Crea el botón “Listo” y configura sus acciones. -

void nuevoPanelParaMapa(): Crea un nuevo panel para colocar el mapa.

- JMapViewer crearMapa(): Crea el mapa que se colocará en el panel.

- void enfocarArgentina(JMapViewer): Le da al mapa las coordenadas de Argentina para que se abra en esa ubicación desde el principio.

-void agregarLocalidadAlMapa(double x, double y, String localidad):Crea un marcador en el mapa en las coordenadas de la localidad.

-void agregarPoligonoLocalidades(Float x1,Float t1, Float x2, Float y2, String peso): Crea un poligono y lo conecta con los marcadores añadidos pasados por parametros

void agregarAGM(Grafo grafo) Conecta :

- void limpiarFormulario(): Limpia el formulario que el usuario utilizó para agregar una localidad.

- void labelNombre(): Pide el nombre de la localidad.

- void textFieldNombre(): Casilla donde se escribe el nombre de la localidad.

- void labelProvincia(): Pide el nombre de la provincia donde se ubica la localidad.

- void textFieldProvincia(): Casilla donde se escribe el nombre de la provincia.

- void labelLatitud(): Pide la latitud de la localidad.

- void textFieldLatitud(): Casilla donde se escribe la latitud de la

localidad. - void labelLongitud(): Pide la longitud de la localidad.

- void textFieldLongitud(): Casilla donde se escribe la longitud de la localidad.

- void crearBotonCrearLocalidad(): Crea el botón “Crear localidad” y configura sus acciones.

- void panelActualizado(): Actualiza el panel donde se encuentra el mapa.

- void crearSeleccion(): Llama a una funcion que calcula el precio y lo muestra por pantalla el costo del AGM

-void calcularPrecioYMostrarArbol(): Metodo que devuelta el costo del AGM

- Menu(): Constructor de la clase.

- boolean verificarNumero(JTextField): Verifica que el número de localidades a ingresar sea válido.

- boolean verificarStringVacios(JTextField): Verifica que el String no esté vacío.

- boolean verificarLongitud(JTextField): Verifica que la longitud ingresada sea válida.

- boolean verificarLatitud(JTextField): Verifica que la latitud ingresada sea válida.

- boolean verificarLocalidadesMaximas(int, int): Verifica que la cantidad de localidades ingresadas sean la cantidad antes dada por el usuario.

Clase AGM: Clase que va a crear el Arbol Generador Minimo

Atributos:

Grafo arbolMinimoGenerado: Se define un Grafo
LinkedList<Localidad> localidadesVisitadas: Se define una lista de Localidad
PriorityQueue<ConectorLocalidades> conexionesPrioritarias: Se define una lista de prioridad sobre ConectorLocalidades

Metodos:

Grafo generarPorPrim(Grafo grafo): Usa el algoritmo de prim para generar el

AGM

void actualizarConexionesPrioritarias(Localidad localidad, Grafo grafo): Se agreganlos elementos a una cola de prioridad

Localidad obtenerNoVisitada(ConectorLocalidades conexion): Verifica que la localidad no haya sido visitada

Clase BFS: Clase que va a aplicar el algoritmo de BFS
Atributos:

List<Localidad> localidadesPendientes: Lista de localidad

Metodos:

boolean esConexo: Verifica que sea conexo el Grafo

Set<Localidad> obtenerAlcanzables(Grafo grafo, Localidad origen): Lista de Set de Localidad

void agregarVecinosNoVisitados(Set<Localidad> localidadesVisitadas, List<Localidad> vecinos) : Set de Localidad y Lista de Localidad de vecinos

Clase Conector Localidades: Clase que va a unir dos localidades

Atributos:

Localidad localidad1:Objeto Localidad
Localidad localidad2. Objeto Localidad
Double distanciaEnKm: Distancia en km de una localidad a otra
Double costo: El costo de esta conexion

Metodos:

ConectorLocalidades(Localidad local1, Localidad local2, double distancia, double costo): Constructor

boolean equals(Object conectorLocalidad): Verifica que la conexion que se esta haciendo no sea igual a otra

Clase Grafo: Clase donde se implementa la creacion de los grafos

Atributos:

LinkedList<ConectorLocalidades> aristas: Lista de objetos de conectorLocalidades

Map<Localidad, LinkedList<Localidad>> vecinos: Diccionario de localidad, localidades

double costoTotal: costo total

double costoPorKm: costo por km entre localidades

double costoPorProvincia: costo por distintas provincias

double mayor300k: costo por km si supera los 300km

Métodos:

Grafo(double costo, double costoPorProvincia, double porcentajeSupera300km, Localidad... localidades): Constructor

Grafo(LinkedList<Localidad> listaVertices, double costo, double costoPorProvincia, double porcentajeSupera300km): Constructor

void generarTodasLasAristas(): Se generan las aristas

void agregarVertice(): Se generan los vertices

void agregarArista(Localidad local1, Localidad local2): Agrega las aristas que serian las localidades que se le pasan por parametro

sumarCosto(): Calcula el costo que se le pasa por parametro

boolean existeArista(Localidad local1, Localidad local2): Verifica que exista la la arits con las localidades que se le pasan por parametro y la crea a conexion

boolean existeVertice(Localidad local1): Verifica que el vertice exista

boolean verificarVertices(Localidad local1, Localidad local2): Verifica que no existan los vertices que se van a añadir

Clase Localidad: Clase que maneja las localidades.

Atributos:

- String nombre: Nombre de la localidad.
- String provincia: Nombre de la provincia donde se encuentra la localidad.
- double latitud: Latitud de la localidad.
- double longitud: Longitud de la localidad.

Métodos:

- Localidad (Integer, String, String, double, double): Constructor de clase.

-boolean equals(Object otraLocalidad): Verifica que la localidad que se le pasa por parametro sea igual a la que estamos comparando

Clase Costo: Clase donde se manejan los costos de las conexiones.

Atributos:

double RadioTierra = 6371: El radio que tiene la tierra

Métodos:

- double calcularDistancia(Localidad origen, Localidad destino): Calcula la distancia entre una localidad y la otra

-calcularHaversine(double restaLatitud, double restaLongitud, double latitudOrigen, double latitudDestino): Es el calculo que te permite hacer la cuenta de radianes para ver las distintas

-double obtenerCosto(Localidad origen, Localidad destino, double costoPorKm, double porcentajeExtra, double costoExtraProvincias): Calcula los costos totales de las distancias entre localidades

Clase Administracion: Clase que administra las demas clases y la conexion con la interfaz

Atributos:

Grafo grafoAGM: Clase grafo

Grafo grafo: Clase grafo

Set<Localidad> localidadesCargadas: Set de localidades

Set<ConectorLocalidades> conexionesEstablecidas: Set de localidades conectadas

double costoPorKm: costo por km entre localidades

double porcentajeSupera300Km: costo por km si supera los 300km

double costoProvinciasDistintas costo por distintas provincias

Metodos:

Administracion(): Constructor

void finalizarCargaLocalidades() Crea una lista con todas las localidades cargadas

boolean crearGrafoAGM() Si se pudo crear el AGM, devuelve un booleano

Grafo obtenerGrafoAGM() Va a llamar a 2 metodos, finalizarCargaLocalidades y crearGrafoAGM

Grafo obtenerGrafo() Retorna el grafo

boolean cargarLocalidad(String nombre, String provincia, Float latitud, Float longitud): Va a agregar a la lista de localidades la localidad que le ingresemos

double obtenerCostoTotalAGM(): Devuelve el costo del AGM