**Informe Trabajo Práctico**

**Programación III**

Integrantes:

Martin Velazquez (legajo: 39963713/2015

Facundo Cardenas (legajo: 42054625/2016)

Rodrigo Paz (legajo: 36788839/2014)

Profesores:

● Patricia Bagnes - pbagnes@ungs.edu.ar

● Javier Marenco - jmarenco@ungs.edu.ar

**Trabajo Práctico “ Clustering”**

I**ntroducción y desarrollo del proyecto:**

El trabajo propuesto por la materia consta en hacer una aplicación con Mapas que pueda generar un Clustering, es decir, grupo de caminos mínimos sobre un grafo en el mapa. En el transcurso del proyecto se verá aplicado el algoritmo de árbol generador mínimo que ayuda a resolver la cuestión de camino mínimo entre los vértices. Trabajaremos sobre el patrón de diseño MVC para definir el diseño del proyecto.

El trabajo se llevó a cabo de manera tal que podamos abrir la ventana de la aplicación, marcar puntos en cualquier parte del mapa y generar la cantidad de Cluster según la distancia que corresponda entre par de grupos de camino mínimo del grafo.

**Detalle sobre el funcionamiento del código:**

***Clases correspondientes al paquete de Interfaz/Controlador:***

Clases de Interfaz: Estas clases están compuestas sobre lo que se va a reflejar en la interfaz de la aplicación y parte de lo que vamos a utilizar de la clase de la lógica del mapa para que se visualice en pantalla, es decir, las clases de interfaz a su vez es el controlador a nivel diseño.

**Mapa** : Esta clase se encarga de inicializar la clase **InterfazGrafica** y que la vista de la Ventana pueda visualizarse.

**InterfazGrafica:** Clase donde se inicializan las clases restantes de la interfaz para que se pueda visualizar el mapa en la aplicación con sus componentes.

**Métodos:**

**bufferInterfaz():** genera un Jpanel donde se va a cargar el contenido que gestionan las demás clases de la interfaz como el mapa, botones, textos.

**InterfazMapa:** Esta clase inicializa los elementos del JmapViewer para que se pueda ver en la aplicación y el controlador del mapa.

**Métodos:**

**bufferMapa():** almacena datos al mapa para que pueda verse en las coordenadas indicadas.

**getLatitud(), getLongitud(), getZoom(), getJMap():** Obtiene los datos indicados**.**

**ControlMapa:** Clase encarga de que se puedan marcar los puntos mediante un marcador en el mapa por el MouseEvent (Cuando presiono el cursor sobre el mapa).

**Métodos:**

**bufferMouse():** MétodoEncargado de darle la accion del Mouse cuando se presiona el cursor sobre el mapa.

**InterfazBotones:** Clase encargada de crear los botones y agregarlos al JPanel donde van a ser utilizados.

**Métodos:**

**bufferBotones():** Agrega los botones instanciados al JPanel

**accionesBoton1():** Accion del boton “Información” donde refleja la información que se genera sobre el mapa, cantidad de clusters, cantidad de aristas,etc.

**accionesBoton2():** Accion del boton “ Generar Clusters” donde envían acciones al controlador que gestiona con la lógica dicha información sobre los clusters con AGM.

**InterfazTexto:** Clase encargada de crear los TextField que vamos a visualizar junto al mapa en la aplicación.

**Métodos:**

**bufferTexto():** se encarga de agregar los TextField donde va a estar la información en el JPanel encargado de gestionar los datos.

**accionesTexto():** 2 métodos que generan el set del texto de los TextFields donde esta la informacion que esta relacionado con la lógica

**InterfazLineas**: Clase encargada de dibujar las aristas en el Mapa con polígonos.

**Métodos:**

**bufferLineas()**: Metodo encargado de almacenar contenido en los polígonos de las aristas del Cluster. Esto tiene relacion con la lógica de mapa.

***Clases correspondientes al paquete donde se encuentra la Lógica del mapa:***

**Punto:** Clase encargada de crear un punto con coordenada latitud y longitud.

**Métodos:**

**Distancia():** calcula la distancia entre un punto instanciado y por el cual recibo por parámetro

**getLatitud() , getLongitud():** Obtengo latitud y longitud del punto.

**equals():** Devuelve true si los puntos son iguales. Método sobreescribido de Object.

**toString():** devuelve el String del punto con su latitud y longitud.

**Arista:** Clase encargada de representar la Arista del Grafo, con su peso y punto de origen-destino.

**Métodos:**

**getPeso() getOrigen() getDestino():** Obtengo peso de la arista(distancia), punto de origen y destino respectivamente dependiendo de quien se invoque.

**toString() :** devuelve el String de la arista con su peso, punto de origen y destino.

**perteneceElPuntoAlaArista():** Devuelve true si el punto recibido por parámetro pertenece al punto de origen o destino.

**Grafo:** Clase encargada de utilizar la estructura del grafo para operar sobre el mapa, contiene las aristas y puntos que conectan a la misma.

**Métodos:**

**addPunto():** Agrega un punto recibido por parámetros a los puntos de grafo.

**addArista():** Agrega una arista entre los puntos (origen y destino)

**cantidadDeVertices():** Devuelve la cantidad de vértices del grafo.

**cantidadDeAristas():** Devuelve la cantidad de aristas del grafo.

**getPuntos():** Devuelve todos los puntos.

**getAristas():** Devuelve todas las aristas.

**toString():** Devuelve el String con la cantidad de aristas, puntos y las aristas del grafo.

**AGM:** Clase encargada de generar el árbol generador mínimo que se muestra en el Mapa partiendo desde el Grafo.

**Métodos:**

**bufferAgm()**: Almacena el contenido del grafo en el AGM.

**crearAgm()**: Crea el AGM sobre un array auxiliar para reemplazarlo sobre el Grafo.

**reemplaceGrafoToAgm()**: Reemplaza el grafo agregando a las aristas lo que contiene el agm recibido por parametro.

**cantidadDeAristas()**: Devuelve la cantidad de aristas

**getPuntos()**: Obtengo los puntos del AGM

**getAristas()**: Obtengo las aristas del AGM

**getCantidadVertices()**: Obtengo la cantidad de vértices del AGM

**getCantidadAristas()**: Obtengo la cantidad de aristas del AGM.

**Cluster**: Clase encargada de generar los Clusters a partir del AGM.

**Métodos:**

**bufferCluster()**: Almacena la información del AGM en el Cluster.

**promedio()**: Se calcula el promedio de las distancias.

**crearClusters()**: Crea el cluster en base al promedio de las distancias y elimina la arista para dividir el AGM.

**getCantidadCluster()**: Obtengo la cantidad de Clusters.

**getPuntos()**: Obtengo los puntos de los Clusters.

**getAristas()**: Obtengo las aristas de los Clusters

**getCantidadVertices()**: Obtengo la cantidad de vértices de los Clusters.

**getCantidadAristas()**: Otengo la cantidad de aristas de los Clusters.

***Clase correspondiente al paquete que contiene Test Unitario:***

**testLogicaMapa**: Clase encargada de testear el funcionamiento de la aplicación y estructuras de datos utilizadas.

**Métodos:**

**testDistanciaEntreDosPuntos()**: Verifica que la distancia entre dos puntos sean iguales.

**testDistanciaEntreDosPuntosDiferente():** Verifica si dos puntos son diferentes

**testGrafoCantidadVertices():** Verifica la cantidad de vértices del Grafo**.**

**testGrafoCantidadAristas():** Verifica que la cantidad de vértices del Grafo.

**testAGMcantidadAristas():** Verifica la cantidad de aristas del AGM.

**testAGMcantidadVertices():** Verifica la cantidad de vértices del AGM.

**testCantidadClusters():** Verifica la cantidad de Clusters

**testClusterCantidadVertices():** Verifica la cantidad de vértices del Cluster

**testClusterCantidadAristas():** Verifica la cantidad de aristas del Cluster.

**Problemática y Conclusiones Finales:**

En el trabajo propuesto tuvimos algunos inconvenientes, siendo uno de los mábajosos el trabajo sobre el Cluster y AGM. Sobretodo en Cluster tuvimos inconvenientes para definir los criterios a la hora de dividir los AGM lo cual resultó sencillo a la hora de utilizar el promedio de las distancias.

Por otro lado continuamos utilizando nuestro patrón de diseño MVC que nos ayuda a ordenar la información y estructurarlo para identificar errores dentro de nuestro código, lo cual permite que si surgen errores en trabajos posteriores podamos identificar con mayor facilidad.