Objetivo

El objetivo que nos planteamos resolver en este trabajo practico consiste en implementar una aplicación que muestre la formación de distintos clusters representados como coordenadas en un mapa, separando el código responsable de la lógica del programa de aquel que se encarga de modelar la interfaz (implementada conSwing y JMapMaker).

Responsabilidad de clases

El proyecto cuenta con 5 paquetes en los que están distribuidas las clases según su funcionalidad:

. Controlador: dentro se encuentran las clases encargadas del funcionamiento de los distintos componentes que pertenecen a la interfaz visual.

. Modelo: dentro se encuentran las clases encargadas de poner en funcionamiento la lógica interna del programa.

. Vista: dentro se encuentran las clases visuales, que contienen el código de interfaz correspondiente al MainFrame y los distintos paneles.

. Test: dentro se encuentra la suite de test correspondiente al funcionamiento interno.

. Iconos: contiene imágenes que son utilizadas por los distintos componentes visuales.

. Principal: contiene la clase principal que se encarga de inicializar las clases visuales y desencadenar el funcionamiento del programa.

A continuación, se prestará detalle a aquellas clases encargadas de la lógica interna, describiendo sus variables de clase, sus métodos y su rol dentro del programa.

Grafo:

Variables de clase

. double[][] D representa la matriz de distancias interna del grafo.

Métodos de clase

A continuación, veremos la descripción de los métodos más importantes de esta clase:

. void agregarArista(int i, int j, double distancia) se encarga de agregar una arista entre los vertices i y j colocando en esa posición de la matriz la distancia entre ambos.

. void borrarArista(int i, int j) se encarga de eliminar la arista entre los vértices i,j volviendo nula la distancia entre ellos.

. boolean existeArista(int i, int j) verifica si existe una arista entre los vértices i,j es decir si la posición de la matriz es mayor a cero.

. set<Integer> vecinos(int i) retorna una colección con aquellos vértices que tengan una arista en común con el vértice i.

Arista:

Variables de clase

. int vertice1, vertice2 representan los vértices que se unen mediante una arista.

. double peso representa el valor de la distancia entre ambos vértices.

Los métodos de esta clase no son relevantes ya que devuelven información de la Arista en sí misma.

AGM:

Esta clase implementa una versión del famoso algoritmo de Prim para obtener un AGM.

Variables de clase

. Grafo grafoAGM representa al grafo que se obtiene como entrada y se modificara para obtener el árbol generador mínimo.

. boolean marcados[] contendrá a los vértices marcados es decir aquellos que hayan sido verificados.

. ArrayList<Arista> aristas contendrá las aristas finales del grafo modificado.

Métodos de clase

. void verificarTamanio() comprueba que el grafo no este vacío, es decir que contenga al menos un vértice.

. void generarAGM(Grafo grafo) se podria explicar con imágenes o con esquemas creo.

. boolean todosMarcados() verifica si todos los vértices están marcados es decir si fueron verificados.

. Grafo getGrafoAGM() retorna el grafo final con la cantidad mínima de aristas que conectan a todos los vértices.