TP numero II

Nombre y apellido: Javier Hernan Monzon.

DNI: 30066008.

Clases en el paquete main:

MainUI

En esta clase tenemos la ui, los datos de entrada. La cual contiene:

- JFrame, el contendor principal.
- Tres JTextField, que serán los datos de entrada, de la cantidad de vértices, aristas, y grafos.
- Diez JLabel, que mostrara información acerca de los componentes en la aplicación.
- Dos JButton, uno corresponde a la generación aleatoria, y el otro para calcular los resultados en las dos versiones de árbol generador minimo.

MainModel

public MainModel () = Inicializamos los componentes de la UI

private void mostrarResultados(long kruskalBFS, long kruskalUnionFind) = Mostramos los resultados calculados, en las dos versiones.

private ArrayList<Grafo> crearGrafosCantSeleccionada(int cantVertices, int cantAristas, int cantGrafos) = Creamos los grafos según la cantidad aleatoria que nos genere, con sus aristas, vértices, y grafos.

private long calcularKruskalBFS(ArrayList<Grafo> grafos) = Calculamos el tiempo que nos lleva hacer la primera versión BFS.

private long calcularKruskalUnionFind(ArrayList<Grafo> grafos) = Calculamos con el metodo Union find.

private int obtenerNumeroAleatorioEntero(int min, int max) = Generamos los datos aleatorios, tanto para los vértices, aristas y grafos.

KruskalBFS

arbolGeneradorMinimoBFS(Grafo g) = Solo contiene este método que nos devuelve el cálculo del árbol generador mínimo con BFS. Recibe el grafo y devuelve un árbol generador mínimo.

KruskalUnionFind

arbolGeneradorMinimoUnionFind(Grafo g) = Este método nos devuelve el cálculo del árbol generador mínimo con UnionFiend. Recibe el grafo y devuelve un árbol generador mínimo.

private static void unionFind(Grafo arbol) = Aquí implementamos el algoritmo unión find.

public static void union(Grafo arbol, Set<Integer> raices) = Aquí tenemos las siguientes parámetros de entrada:

- Grafo del árbol de kruskal que estamos generando
- Las raíces de nuestro componentes conexas
- Devolvemos el modificamos el árbol y el grafo por referencia

public static int raiz(Grafo g, int vertice) = Obtenemos las raiz de una componente conexa.

Grafo

Clase que realizamos en la cursada, solo con unos cambios, que son los siguientes:

- Se implementó la interfaz Cloneable, para tener una copia del grafo original y ver que aristas nos faltan para procesar.
- **agregarPesoArista(int i, int j, float peso)** = Agregamos el peso a una arista, donde lo iremos guardando en un HashMap, donde el valor como key es las dos vértices contatenados, y su peso.
- **public float obtenerPesoArista(int i, int j**) = Obtenemos el peso que posee una arista, en donde i+j, nos devuelve su peso.

UtilGrafos

Clase en la que tendremos los métodos y las devoluciones de objetos, que necesitamos para las dos versiones. Posee métodos staticos públicos, para facilitar su acceso, en las clases KruskalBFS y KruskalUnionFind.

public static boolean esConexo(Grafo g, int verticePartida) = Devuelve si un grafo es conexo, en donde también indicamos el vértice de partida.

public static HashMap<Integer, Set<Integer>> dameVecinosVertice(Grafo g) = Devuelve los vecinos de un un grafo, los alcanzables desde ese vértice.

public static HashMap<Integer, Integer> dameAristaDeMenorPesoBFS(Grafo g) = Nos da la arista de menor peso, según el grafo que le pasemos.

public static boolean formaCircuito(Grafo g, int verticePartida, int verticeFinal) = Nos indica si al agregar una arista esta puede o no formarse un circuito con nuestro árbol generador mínimo.

public static float obtenerPesoAleatorio() = Nos da un peso aleatorio entre 0 y 1, del tipo float.

public static Grafo obtenerGrafoAleatorio(int cantidadVertices, int cantidadAristas) = Nos genera un grafo aleatoreo según la cantidad de aristas y vértices.

private static int obtenerAristaAleatoriaValida(int verticeActual, int cantidadVertices) = Nos da un vértice aleatorio valido con lo que podremos generar una arista.

private static Grafo conexar(Grafo g) = Conexamos el Grafo con aristas aleatorias, y luego agregamos mas aristar, para de este modo crear un grafo conexo, en donde luego podemos generar el árbol generador minimo.

Clases en el paquete testMain:

Aquí tenemos nuestra Suit de tests, de la clase útil de grafos y las dos clases de kruskal en sus dos versiones.

Clases en el paquete testGrafos:

Clases realizadas en la cursada, nos será útil, ya que posee varios test para verificar los grafos que usaremos en la cursada. Los nombres de los tests son de igual nombre que los métodos de las clases que testean, agragando en sus métodos la palabra test.

KruskalBFSTest, contiene:

- public void generarArbolMinimoBFS()

KruskalUnionFindTest, contiene:

- public void generarArbolMinimoUnionFind()

KruskalUnionFindTest, contiene:

- public void esConexoTest()
- public void obtenerPesoArista()
- public void formaCircuitoTestBFS()
- public void dameAristaDeMenorPesoBFSTest()
- public void dameVecinosVerticeTest()
- public void obtenerGrafoAleatorioFaildTest()

Carpeta recursos:

- Una imagen que se usó de guía para guiarnos en la creación de los algoritmos.
- La documentación aquí escrita.