Universidad Nacional de General Sarmiento

Licenciatura en Sistemas
Tecnicatura Superior en Informática

Trabajo Práctico N°3
Distribución Golosa

Programación 3
(2º semestre 2020)

INTEGRANTES:
Sanchez Juan Ignacio - 39160465 - juani.scz95@gmail.com
Schmidt Maximiliano - 40664272 - maxischmidt2011@gmail.com
Sosa Martín Leonel - 36289472 - martinleonelarce@gmail.com

DOCENTES:
Alejandro Nelis
Fernando Torres

1.Introducción y Consignas

El trabajo práctico consiste en implementar un algoritmo goloso para una variante del problema de facility location.

Tenemos un conjunto C de clientes que debemos atender, cada uno geolocalizado con su latitud y longitud. Tenemos también un conjunto D de puntos donde podemos ubicar centros de distribución para los clientes. Estos puntos tambi´en est´an geolocalizados con su latitud y longitud. Finalmente, tenemos una cantidad máxima de centros de distribución que podemos abrir. Estos k centros de distribución deben ubicarse en k puntos del conjunto D.

Cada cliente será atendido desde el centro de distribución más cercano, y el costo de atenderlo es igual a la distancia en línea recta entre el cliente y su centro de distribución 1. El problema consiste en determinar qué subconjunto de k puntos de D se deben seleccionar para abrir centros de distribución, de modo tal que el costo total sea el menor posible.

Implementar una aplicación que contenga la siguiente funcionalidad:

• Leer los datos de latitud y longitud de los clientes desde un archivo (el formato del archivo queda a criterio del grupo).

• Leer los datos de latitud y longitud de los posibles centros de distribución desde un archivo (el formato del archivo queda a criterio del grupo).

• Resolver el problema de determinar qué centros de distribución abrir, por medio de un algoritmo goloso.

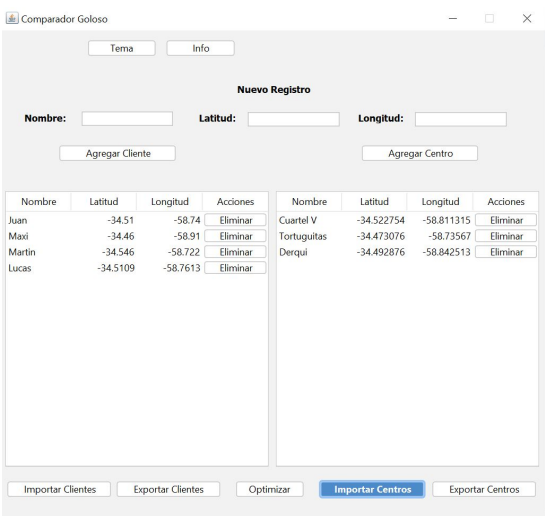
• Mostrarle al usuario los centros de distribución que el algoritmo propone abrir, y el costo total de esta solución. Como objetivos opcionales no obligatorios, se pueden contemplar los siguientes elementos:

1. Mostrar a los clientes y los posibles centros de distribución sobre un mapa.
2. Escribir la solución (que centros de distribución abrir) en un archivo.
3. Mostrar estadísticas de la solución, como la distancia promedio entre los clientes y los centros de distribución, máximo y mínimo número de clientes por centro de distribución, etc.

2. Separación de Capas y Denominación de Clases

Nuestra aplicación será capaz de:

1. Agregar clientes y centros de distribución, siempre y cuando no sean iguales (que comparten nombre y/o ubicación).
2. Mostrar a todos los clientes y centros de distribución en dos tablas, donde se indicará los atributos correspondientes.
3. Importar y exportar listas de clientes y de centros de distribución que se almacenarán en un archivo JSON.



4. A partir de una lista de clientes y de centros de distribución se podrá optimizar la relación de los centros de distribución con sus clientes correspondientes.
5. Cuando se genere la optimización se mostrará la información en otra ventana, pudiendo seleccionar cada centro de distribución para ver sus clientes respectivamente, también se podrá visualizar los centros de distribución y los clientes en una mapa.

Para la implementación de la aplicación se decidió separar en tres capas :

I. **Interfaz**
II. **Modelo**
III. **Negocio**
IV. **Vista**

Cada capa es responsable de un aspecto de la aplicación, la capa de "**Negocio**" se encarga de la lógica y la "**Interfaz**" provee una interface de ILugar (véase en Interfaz), "**Vista**" se encargan de toda la interfaz gráfica. Como clase principal de la capa de negocio se implementa la clase Negocio y como principales de la capa de Interfaces Panel, como ventana principal. En el caso del "**Modelo"** solo se encuentra Persistencia que se encarga de guardar y cargar en archivos los conjuntos de clientes y centros de distribución.

VISTA:

Package Vista

PanelDialog: Es el panel donde se muestran los grupos de personas con sus respectivos promedios. Contiene TablaCabecera y TablaDetalle.

TablaCabecera: Es la tabla que muestra los diferentes centros optimizados que se obtienen.

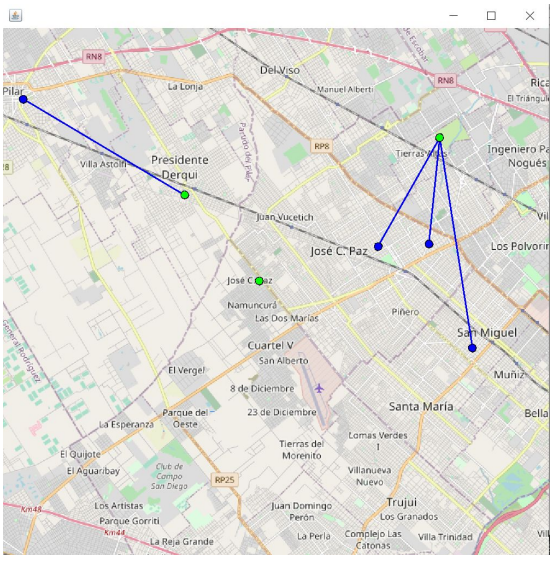
TablaDetalle: Es la tabla que muestra a los clientes contenidos en cada centro de distribución optimizado.

BottonColumn: Representa los botones de "Eliminar" en la columna de "Acciones", al utilizar este botón se elimina a la persona de la clase Negocio y modifica la Tabla de la interfaz principal.

Temas: Posee los temas para la aplicación.

Botonera: Son los botones que se encuentran dentro de la ventana del programa principal: Tema, Info, Optimizar, Exportar Centros, Exportar Clientes, Importar Centros e Importar Clientes.

Panel: Es la ventana principal de la aplicación. Su función principal es recibir los datos de los clientes y de los centros de distribución, y poseer los botones para generar la optimización de los centros, exportar, importar, etc.



CampoRegistro: Son todos los elementos correspondientes para recibir los datos de un cliente o centro de distribución. El textField para ingresar el nombres y los comboBox para ingresar latitud y longitud.

Tabla: Es la encargada de almacenar y mostrar a cada cliente y centro de distribución de cada conjunto.

Mapa: Es el mapa donde se grafica los centros de distribución y los clientes.

PanelMapa: Contiene al Mapa donde se grafican los centros de distribución y clientes.

NEGOCIO:

Package Negocio

Centro: Representa a un centro de distribución con su nombre, ubicación, clientes asociados, la utilidad del centro y si se encuentra activo o no.

Cliente: Representa a un cliente con su nombre, ubicación, centro al que se encuentra asociado y la distancia a la que se encuentra del mismo.

Lugar: Representa a un lugar con su nombre y ubicación.

Ubicacion: Representa a una ubicación que posee latitud y longitud.

ComparadorGoloso: Es el encargado de ordenar a los clientes asociados a un centro de distribución, teniendo en cuenta: la cantidad de clientes asociados y después la utilidad del centro.

ConjuntoDeLugares: Representa a un grupo de lugares en una lista.

DatosOptimizados: Representa a los centros de distribución optimizados y activos, también a los clientes asociados a los centros activos.

Negocio: Es el encargado de administrar y de llevar a cabo las tareas más importantes y nucleares del programa.

Heuristica: En esta clase se encuentran los métodos para optimizar, distribuir y redistribuir a los clientes en los centros de distribución

MODELO:

Package Modelo

Persistencia: Se encarga de guardar y cargar el conjunto de clientes y centros de distribución en dos archivos de extensión JSON.

INTERFAZ:

Package Interfaz

ILugar: Es una interfaz que representa a un lugar, que posee nombre y ubicación.

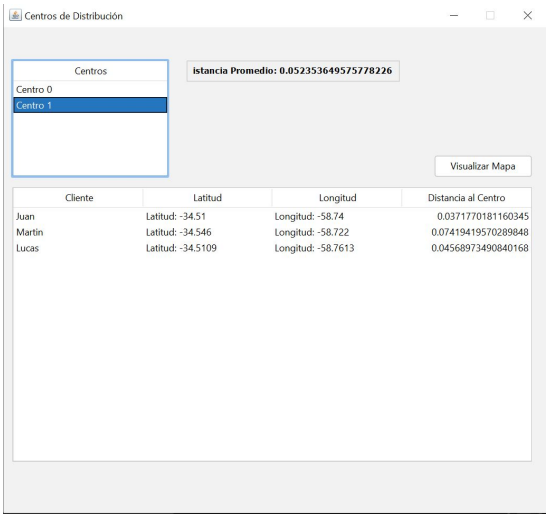
Desarrollo y decisiones de implementación:

La implementación inicial solo contaba con las clases: Negocio, CentrosDeDistribucion, Cliente, Heuristica, ConjuntoDeClientes y ConjuntoDeCentrosDeDistribuicion. Pero nos dimos cuenta que las clases ConjuntoDeClientes y ConjuntoDeCentrosDeDistribuicion estaban muy estrechamente vinculadas a la nueva clase que creamos ConjuntoDeLugares, así que decidimos integrar estas dos clases a ConjuntoDeLugares, y crear una interfaz de lugar llamada ILugar.

Otra decisión tomada, fue la de mostrar a los clientes generados al optimizar en tablas en una ventana aparte, en vez de utilizar la misma ventana, lo mismo en el caso de que se desee visualizar a los clientes y los centros de distribución en el mapa.

También optamos por cambiar el nombre de la class CentrosDeDistribucion por solamente Centro.

Originalmente existían las clases ConjuntoDeClientes y ConjuntoDeCentrosDeDistribuicion, ahora utilizamos simplemente ArrayList para almacenar Clientes y Centros.



Al igual que en el trabajo práctico anterior decidimos “desmontar” toda la parte de la interfaz gráfica en pequeñas partes las cuales están controladas por la clase Panel.

Algoritmos:

Algoritmo encargado de optimizar, activar y desactivar a los centros de distribución, también asigna a los clientes a cada centro activo.

```
private void optimizar() {
            asignarClienteCentroSegunDistancia();
            calcularUtilidadCentroReconocerClientes();
            Collections.sort(centrosOptimizados, new ComparadorGoloso());
            reasignarClientesHuerfanos();
    }
}
```

```
private void asignarClienteCentroSegunDistancia() {
```

```
		for (Cliente cliente : clientesOptimizados) {
			cliente.setDistanciaAlCentro(Double.MAX_VALUE);
			for (Centro centro : centrosOptimizados) {
				double dist = calcularDistancia(centro, cliente);
				if (cliente.getDistanciaAlCentro() > dist) {
					cliente.setCentroAsociado(centro);
					cliente.setDistanciaAlCentro(dist);
				}
			}
		}
	}

	private void calcularUtilidadCentroReconocerClientes() {
		for (Centro centro : centrosOptimizados) {
			double distanciaAcum = 0;
			for (Cliente cliente : clientesOptimizados) {
				if (cliente.getCentroAsociado().equals(centro)) {
					centro.getClientesAsociados().add(cliente);
					distanciaAcum += cliente.getDistanciaAlCentro();
				}
			}
			if (centro.getClientesAsociados().size() != 0)
				centro.setActivo(true);

			centro.setUtilidad(distanciaAcum /
centro.getClientesAsociados().size());
		}
	}

	private void reasignarClientesHuerfanos() {
		ArrayList<Cliente> clientesHuerfanos;
		int index = 0;
		if (cantActivos() < cantidad)
			return;
		while (cantActivos() > cantidad) {
			index = activoDeMenorUtilidad();
			clientesHuerfanos =
centrosOptimizados.get(index).getClientesAsociados();
			centrosOptimizados.get(index).setActivo(false);
			Centro centroIdeal = centrosOptimizados.get(0);
			for (Cliente cliente : clientesHuerfanos) {
				for (Centro centro : centrosOptimizados) {
					double dist = calcularDistancia(centro, cliente);
					if (cliente.getDistanciaAlCentro() > dist &&
centro.getActivo()) {
						cliente.setCentroAsociado(centro);
						cliente.setDistanciaAlCentro(dist);
						centroIdeal = centro;
					} else
```

```
cliente.setDistanciaAlCentro(Double.MAX_VALUE);
                                    }
                                    centroIdeal.getClientesAsociados().add(cliente);

//centrosOptimizados.get(centrosOptimizados.indexOf((cliente.getCentroAsociado()))).getCli
entesAsociados().add(cliente);
                                }
                        }
                }
```

CLASES de Negocio

public class Centro implements Serializable, ILugar {

```
        private static final long serialVersionUID = 1L;
        private ArrayList<Cliente> clientesAsociados;
        private double utilidad;
        private Boolean activo;
        private String nombre;
        private Ubicacion ubicacion;
```

public class Cliente implements Serializable, ILugar {

```
        private static final long serialVersionUID = 1L;
        private Centro centroAsociado;
        private double distanciaAlCentro;
        private String nombre;
        private Ubicacion ubicacion;
```

public class ComparadorGoloso implements Comparator<Object> {

public class ConjuntoDeLugares implements Serializable {

```
        private static final long serialVersionUID = 1L;
        private ArrayList<Lugar> lugares;
```

public class DatosOptimizados {

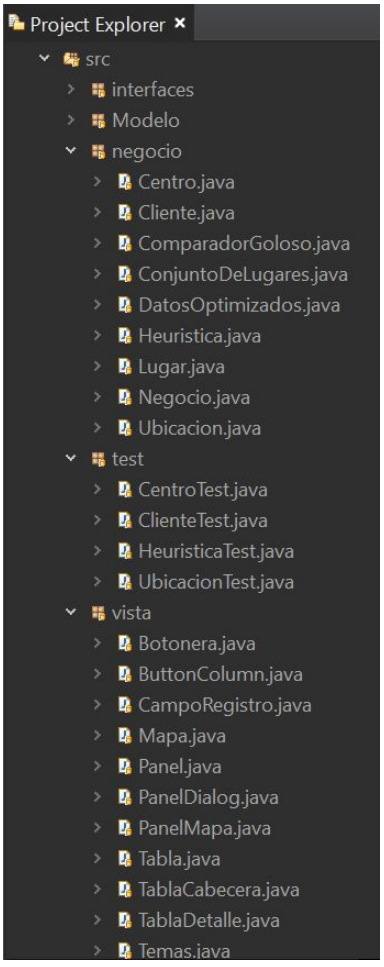
```
        ArrayList<Centro> centrosOptimizados;
        ArrayList<Cliente> clientesOptimizados;
        ArrayList<Centro> centrosActivos;
```

public class Heuristica {

```
        ArrayList<Centro> centrosOptimizados;
        ArrayList<Cliente> clientesOptimizados;
        int cantidad; //cantidad de centros pedidos por el usuario
```

public class Lugar implements Serializable {

```
        private static final long serialVersionUID = 1L;
```



```
    private String nombre;
    private Ubicacion ubicacion;

public class Negocio {
    private ArrayList<Centro> centrosDeDistribucion;
    private ArrayList<Cliente> clientes;

public class Ubicacion implements Serializable {
    private static final long serialVersionUID = 1L;
    private double latitud;
    private double longitud;
```

CLASES de Vista

```
public class Botonera extends JPanel {
    private static final long serialVersionUID = 1L;

public class Mapa extends JPanel {
    private static final long serialVersionUID = 1L;
    private JMapViewer mapa;

public class PanelMapa extends JFrame {
    private Mapa mapa;
    private static final long serialVersionUID = 1L;

public class Panel {
    private JFrame frame;
    private Botonera botonera;
    public Tabla tablaClientes;
    public Tabla tablaCentros;
    private Negocio negocio;
    private Temas temas;

public class CampoRegistro extends JPanel {
    private static final long serialVersionUID = 1L;
    private JTextField txt_nombre;
    private JTextField txt_longitud;
    private JTextField txt_latitud;

public class PanelDialog extends JFrame {
    private static final long serialVersionUID = 1L;
    private DatosOptimizados data;
    private JTextField txt_distanciaPromedio;

public class Temas
    int temaActual;
    private List<LookAndFeel> temasDisponibles;
```

public class ButtonColumn extends AbstractCellEditor implements TableCellRenderer, TableCellEditor, ActionListener, MouseListener

```
private static final long serialVersionUID = 1L;
private JTable table;
private Action action;
private int mnemonic;
private Border originalBorder;
private Border focusBorder;
private JButton renderButton;
private JButton editButton;
private Object editorValue;
private boolean isButtonColumnEditor;
```

public class Tabla extends JPanel {

```
private static final long serialVersionUID = 1L;
private JTable table;
private DefaultTableModel model;
private Consumer<Integer> funcionEliminarFila;
```

public class TablaCabecera extends JPanel {

```
private static final long serialVersionUID = 1L;
private JTable table;
private DefaultTableModel model;
```

public class TablaDetalle extends JPanel {

```
private static final long serialVersionUID = 1L;
private JTable table;
private DefaultTableModel model;
```

CLASES de Interfaz

public interface ILugar {

```
public String getNombre();
public Ubicacion getUbicacion();
```

CLASES de Modelo

public class Persistencia {

```
private String direccionClientes = "clientes.json";
private String direccionCentros = "centros.json";
```