Universidad Nacional de General Sarmiento

Programación 3

Trabajo Práctico 3: Facility Location

Fernandez Gabriela - Luna Nicolas - Santillan Javier

Docentes:
Alejandro Nelis
Fernando Torres

Fecha de entrega:
08 de Diciembre del 2020

Introducción

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

Tenemos un conjunto de clientes y centros de distribución ubicados por coordenadas, también contamos con una cantidad máxima de centros de distribución que podemos abrir.

Los centros que se van a abrir se deben definir por algún criterio específico.

Funcionalidad

La aplicación es capaz de:

- Crear,modificar y leer los datos de los Clientes y Centros de distribución desde un archivo JSON.
- Resolver el problema de determinar qué centros de distribución abrir, por medio de un algoritmo goloso.
- Solucionar el problema por una distancia promedio o por categoría del Centro de distribución.
- Mostrar en un mapa los centros de distribución y los clientes.
- Mostrar en un mapa el resultado del algoritmo goloso.
- Guardar los resultados de la heurística en otro archivo JSON.

Implementaciones y separación de capas

La implementación de la aplicación se dividió en tres capas

Lógica:

- FacilityLocation
- Solver
- Solucion
- Distribucion
- Centro
- Cliente

JSON:

- ManejoDato
- Centro_JSON
- Cliente_JSON

Gráfica:

- Ventana

Desarrollo

Capa Lógica

Para la funcionalidad del proyecto se implementaron las siguientes clases:

Se implementó FacilityLocation, la más importante, que une las funciones de Distribución, Solución y de ManejoDato:

public void ResolverPorDistancia(int cantidadSoluciones) → posibles centros de distribución, dependiendo la distancia va a buscar los centros mas cercanos.

public void ResolverPorCategoria(int cantidadSoluciones) → recibe la cantidad de centros de distribución a abrir y resuelve el problema dependiendo la categoría.

public ArrayList<Centro> getCentrosACrear() → obtiene la lista de centros de distribución a crear.

public void TraerDatosDeJSON() → trae los datos de los JSON y los guarda en una lista de clientes y una lista de centros.

public HashMap<Coordinate, String> clientes() → Devuelve los nombres de los clientes y sus coordenadas.

public HashMap<Coordinate, String> centros() → devuelve hashmap con la coordenada como clave y como valor los nombres de los centros.

public HashMap<Coordinate, String> solucion() → devuelve hashmap con la coordenada como clave y como valor los nombres de los centros elegidos a abrir.

private Comparator<Centro> comparadorPorDistancia() → implementa clase anónima de un comparador de centros por distancia por promedio.

private Comparator<Centro> comparadorPorCategoria() → implementa clase anónima de un comparador de centros por categoría.

Se implementó la clase Solver, que se encarga de resolver problema:

public Solucion resolver() → devuelve una objeto Solucion , llama a definirDistancias(), centrosOrdenados() y agrega centros de acuerdo a la cantidad de centros que se quieren crear.

private void CentrosOrdenados() → ordena los centros de acuerdo al comparator implentado, puede ser por la distancia promedio o por categoría.

public void definirDistancias() → setea a cada centro la distancia promedio entre los clientes y ese centro.

public double distancia(Coordinate uno, Coordinate otro) → Devuelve un tipo double que es la distancia entre dos puntos de la tierra con sus respectivas latitudes y longitudes, utiliza el método semiverseno().

private double semiverseno(double arg) → es utilizada por la función distancia() devuelve un double que es el resultado de aplicar la fórmula del semiverseno a un argumento de tipo double.

Se implementó la clase Solución, que se encarga de contener una solución dejada por solver

public void agregarCentro (Centro c) → guarda a la lista los centro que se desean abrir con su coordenada.

public ArrayList<Centro> getCentros() → obtiene la lista de centros.

int cardinalidad () → devuelve el tamaño de la lista de centros.

public String getNombre(int i) → devuelve el nombre de un centro de distribución a partir de un número.

public void setDistPromedio(double distanciaPromedio) → setea la distancia promedio entre el centro y el cliente.

public double getDistProm() → obtiene la distancia promedio entre el cliente y el centro de distribución.

public int cantidadCentros() → obtiene la cantidad de centros que hay.

También se modelo la clase Distribución que contiene los centros, los clientes y la solucion

public void agregarCentro(Centro c) → agrega centros

public ArrayList<Centro> getCentros() → devuelve lista de centros.

public ArrayList<Cliente> getClientes() → devuelve lista de clientes.

public int cantidad Centros() → devuelve cantidad de centros.

public int cantidadClientes() → devuelve cantidad de clientes.

También modelamos la clase Centro que son los centros de distribución que existe para atender a los clientes.

public void setDistanciaPromedio(double promedio) → setea la distancia promedio entre el centro de distribución y el cliente.

public double getDistanciaPromedio() → obtiene la distancia promedio.

public int getCategoria() → obtiene los beneficios de ese centro.

public String getNombre() → obtiene el nombre del centro.

public Coordinate getCoord() → obtiene la latitud y longitud del centro.

Por último tenemos la clase Cliente que modela a un cliente y tiene la siguiente interfaz:

public String getNombre() → En el cual obtiene el nombre del cliente.

public Coordinate getCoord() → para obtener el lugar en donde está ubicado el cliente (obtiene la latitud y longitud).

JSON

Se usó la librería GSON para crear un archivo .json que contiene los clientes, esto con el objetivo de poder cargar los clientes y centros de distribución antes cargadas cuando se vuelve a iniciar la aplicación. Cuando se agregan clientes esta lista se actualiza.

- Cliente_JSON → guardan los clientes y genera el .json.
- Centro_JSON → guardan los centros de distribuciones y genera el .json.
- ManejoDato → Usa Clientes_JSON y Centros_JSON. Crea un .json específico donde se guardan los datos de los clientes y los centros, para luego guardar en datosSolucion.json sólo los centros de distribución que se decidieron abrir.

Gráfica

Ventana → Por una lado muestra un menú de presentación de la aplicación y da las opciones de los distintos tipos de soluciones (categoría o distancia), la cantidad de centros que se quieren abrir, un botón para mostrar los resultados el cual pinta de color amarillo los centros que se abrieron, y por ultimo un boton reset que despinta los centros abiertos por el boton mostrar resultados. A su vez en otro costado muestra un mapa con todos los centros en color azul con sus respectivos nombres y todos los clientes en color violeta.

Conclusión

Pudimos comprobar durante el desarrollo del proyecto que el uso de clases anónimas utilizadas para diferenciar las diferentes formas de beneficios fueron útiles ya que permiten que sin necesidad de implementar demasiadas clases poder cumplir nuestro cometido de inyectar distintos tipos de Comparaciones a la interfaz Comparator.
La implementación de los tests unitarios nos ayudaron rápidamente a identificar si los métodos de la capa de negocio cumplían bien su funcionamiento y a actualizar el código cuando teníamos que cambiarlo.