Universidad Metropolitana

Departamento de Gestión de Proyectos y Sistemas.
Estructuras de datos (BPTSP06).
Trimestre 2425-1.

# Proyecto: Cobertura de sucursales

## *Preliminares*

En el siguiente proyecto, usted tendrá que realizar una investigación documental que le permita obtener información sobre el contexto del problema. En tal sentido, se sugiere que comience por realizar la siguiente lectura relativa a grafos, pero tome en cuenta que es solo un recurso inicial que debe ser complementado con la búsqueda autónoma de información por parte de los integrantes del equipo de trabajo:

https://drive.google.com/file/d/1ml9sZlFie0aLTqYit9Pc6LEQGTBFPPut/view?usp=drive_link

## *El problema*

Una reconocida cadena de supermercados busca entrar en Sudamérica, pero necesita un programa que le ayude a determinar la mejor ubicación de sus sucursales. Como política estratégica de la empresa, una sucursal se ubicará dentro de un radio de no más de 100 metros de una parada del sistema de transporte urbano de la ciudad. Además, la Dirección de Operaciones de la empresa ha generado un modelo geográfico de uso interno en el que cada parada se considera como el centro geográfico alrededor del cual y en un radio equivalente a máximo *t* paradas, se define una zona comercial. Cada zona es denominada como la parada elegida como su centro y que está próxima a la sucursal. Se considera que una sucursal cubre comercialmente una zona si la parada (centro de la zona) está a no más de *t* paradas de una parada con sucursal (donde *t* es un número que puede variar por ciudad).

El programa debe permitirle al usuario seleccionar paradas para colocar sucursales y luego indicar la cobertura comercial, según la configuración creada. De esta forma, el programa ayudará al usuario a determinar si se logra cubrir las necesidades de la población que hace vida, al menos alrededor de las rutas de transporte urbano. La meta es cubrir enteramente la ciudad.

Se le pide estudiar dos casos:

- Caracas, en donde se toman las paradas del Metro con $t = 3$
- Bogotá, en donde se toman las paradas del Transmilenio con $t = 10$.

La información de las redes de transporte (Metro de Caracas, Transmilenio de Bogotá) se darán en archivos que su programa debe ser capaz de cargar. El programa debe poder cargar y trabajar con otras redes no contempladas, en el mismo formato.

## *Requerimientos Funcionales*

Debe realizar un programa en Java utilizando la librería de interfaces gráficas de Java Swing que permita:

1. **Cargar una nueva Red de Transporte desde un archivo:** El programa debe, en cualquier momento, poder cargar un archivo con una red de transporte. Si ya hay una red cargada, el programa deberá reemplazar la que se cargó anteriormente con la que se encuentra en el archivo.
2. **Mostrar grafo:** El programa deberá mostrar el grafo correspondiente. Se sugiere utilizar GraphStream (https://graphstream-project.org/).
3. **Establecer *t*:** Debe ser posible, en cualquier momento, cambiar el valor de *t* para poder evaluar escenarios alternos (por ejemplo, Caracas con $t = 2$). Todas las revisiones de cobertura sucesivas deben realizarse con el nuevo valor de *t* hasta que vuelva a ser cambiada.
4. **Colocar sucursal:** Debe ser posible seleccionar cualquiera de las paradas de la red de transporte y establecerla como la ubicación de una futura sucursal.
    a. Debe ser posible des-seleccionar la parada.
5. **Ver cobertura de sucursal:** Debe ser posible seleccionar una sucursal colocada y ver cuáles paradas son alcanzables desde la misma dentro del límite *t*.
    a. Se debe poder especificar que el programa realice esta revisión con Búsqueda en Profundidad (DFS por sus siglas en inglés)
    b. Se debe poder especificar que el programa realice esta revisión con Búsqueda en Amplitud (BFS por sus siglas en inglés)
6. **Revisar cobertura total:** Cada vez que se coloque una sucursal, el programa deberá revisar si, con este cambio, el total de las sucursales colocadas cubre totalmente la ciudad.
    a. Si las sucursales colocadas cubren totalmente la ciudad, el programa debe indicarlo.
    b. Si la cobertura no es total, el programa debe sugerir, de entre las paradas no cubiertas por ninguna sucursal, dónde colocar las sucursales para lograr la cobertura total.
    c. Los requerimientos **5.a** y **5.b** solo deben realizarse cuando hay al menos una sucursal seleccionada y debe indicarse en el grafo dibujado.
7. **Agregar línea:** Debido a que las redes están en constante expansión, el programa debe permitir agregar líneas nuevas a la red para evaluar si, cuando las mismas estén completadas, se logrará la cobertura completa (por ejemplo, la propuesta Línea 6 que conectaría las estaciones Zoológico y La Rinconada—mostrada en rojo en la siguiente figura—puede hacer innecesario que ambas tengan una sucursal). Tome en cuenta que agregar una línea es en realidad agregar una *secuencia* de paradas.


## *Formato de Archivo*

Los archivos se presentan en un archivo JSON con en el siguiente formato:

```
{
    <Nombre de la Red>
        {
            <Nombre de Línea₁>:[
               <Nombre Parada₁>,
               ...
               {<Nombre Paradaᵢ> : <Nombre Paradaⱼ>},
               ...
               <Nombre Paradaₘ>
            ]
        },
        ...
        {
            <Nombre de Lineaₙ>:[
               <Nombre Parada₁>,
               ...
               <Nombre Paradaₘ>
            ]
        }
}
```

Donde los nombres de las líneas y paradas son únicos y siempre van encerrados en comillas. Si una misma parada aparece en dos líneas, esto indica que es una estación de transferencia entre esas dos líneas. Si una parada está entre llaves “({ })” unida a otra con dos puntos (:) indica que existe una conexión peatonal entre ellas, permitiendo que sean contadas como la misma estación para efectos de *t*. Por ejemplo, la parada **`{"Capitolio":"El Silencio"}`** de la línea 1 y la parada **`{"El Silencio":"Capitolio"}`** de la línea 2 se pueden considerar como una única parada.

Los archivos de entrada para los casos descritos pueden encontrarse en los siguientes enlaces:

- Caracas.json

- Bogota.json

## *Requerimientos técnicos*

1. Debe tener una clase grafo, la cual se debe representar la red de transporte y debe ser utilizada para responder a los requerimientos funcionales. El grafo debe ser implementada con una **matriz de adyacencia** o una l**ista de adyacencia**.

2. Puede utilizar cualquier otra estructura auxiliar de ser necesario para mejorar los tiempos de respuesta del programa. Sin embargo, **NO podrá utilizar librerías para la implementación del tipo de dato abstracto**.

3. La aplicación debe ofrecer una interfaz gráfica al usuario. No se permite la entrada ni la salida de información por consola. Debe investigar independientemente cómo usar la librería Java Swing.

4. El programa debe poder cargar un archivo de texto para la lectura de datos. Para ello, es requerido el uso del componente **JFileChooser**.

5. Debe documentar el proyecto con **Javadoc**.

6. Junto al programa, cada equipo deberá presentar un **Diagrama de clases** *(arquitectura detallada)* que explique la solución obtenida. El diagrama de clases se deberá incluir como un archivo PDF en el repositorio de código

7. Los equipos de trabajo deberán utilizar **GitHub** para el control de versiones y facilitar el trabajo en equipo de manera asíncrona. De esta forma, podrán comenzar a crear su portafolio de trabajos, elemento que puede ser importante a la hora de buscar trabajo. En el registro se deberá reflejar la participación activa y significativa de los integrantes.

## *Consideraciones*

- Los equipos deben ser de **3 personas.**

- Los proyectos que no tengan interfaz gráfica, serán calificados con **0 (cero)**.

- Los proyectos que no tengan documentación alguna, serán calificados con **0 (cero)**.

- Los proyectos que implementen el tipo de dato abstracto usando librerías como java.util, serán calificados con **0 (cero)**.

- Los proyectos que sean iguales o parecidos, serán calificados con **0 (cero)**.

- Los programas que “no compilen” o “no corran”, serán calificados con **0 (cero).** Tenga en cuenta que el hecho de que el programa muestre una interfaz gráfica de usuario sin funcionalidad alguna, será considerado como que ***no*** corre.

- Los registros de Github que tengan todos los commits de un solo integrante o en un solo día, serán considerados como ***no válidos***, pues no reflejan la participación activa y

significativa de todos los integrantes del equipo (punto 7 del apartado de requerimientos técnicos).

- Los proyectos **podrán ser sometidos a defensa**, es decir, el facilitador convocará al equipo para una revisión si lo considera necesario.

## *Criterios de evaluación*

- *Funcionalidad:* Capacidad para proporcionar las funcionalidades que satisfacen las necesidades explícitas e implícitas bajo unas ciertas condiciones: **60%**
    - *Adecuación:* El programa ofrece todas funcionalidades que respondan a las necesidades, tanto explícitas (contenidas en el documento descriptivo del proyecto) como implícitas; entendiendo como necesidades implícitas, aquellas que, no estando descritas en el documento, surgen como resultado de un concienzudo análisis del problema planteado y que aseguran el correcto funcionamiento del programa.
    - *Exactitud*: El programa genera los resultados o efectos correctos o acordados, con el grado necesario de precisión.
- *Fiabilidad*: Capacidad para mantener un nivel especificado de prestaciones cuando se usa bajo ciertas condiciones: **20%**
    - *Madurez*: El programa no presenta fallas originadas por errores de programación, análisis o diseño. **(10%)**
    - *Tolerancia a fallos*: El programa responde adecuadamente al manejo inadecuado del usuario; es decir, mantiene su correcto funcionamiento aun cuando el usuario introduzca datos erróneos o manipule inadecuadamente las interfaces de usuario. **(10%)**
- *Usabilidad*: Capacidad del proyecto para ser entendido, aprendido, usado y al mismo tiempo, ser atractivo para el usuario, cuando se usa bajo condiciones específicas: **10%**
    - *Comprensibilidad*: El programa ofrece una interfaz de fácil comprensión, facilitando su aprendizaje y correcta utilización. El programa emite mensajes de alerta cuando se introducen valores erróneos. Existen elementos informativos que indican al usuario como operar el programa. **(5%)**
    - *Capacidad de ser atractivo*: El diseño de la interfaz de usuario, esto es: disposición de controles, esquema de colores, utilización de cajas de diálogo y demás elementos gráficos; es atractivo para el usuario. **(5%)**
- *Eficiencia*: Capacidad para proporcionar prestaciones apropiadas, relativas a la cantidad de recursos utilizados, bajo condiciones determinadas: **10%**

- *Estructuras de datos*: Utiliza eficientemente las estructuras de datos para la solución del problema.