Universidad de Castilla-La Mancha

Álgebra y Matemática Discreta

Practica Incremental 2021-22 (Grupo Español)

Sistema de navegación

Profesores de la asignatura:
José Ángel Martín Baos (joseangel.martin@uclm.es)
José Luis Espinosa Aranda (josel.espinosa@uclm.es)
Juan Romero Cañas(juan.romero2@alu.uclm.es)
Ricardo García Ródenas [Teoría] (ricardo.garcia@uclm.es)
José Ángel López Mateos [Teoría]

A tener en cuenta:

1. La realización de la práctica es individual.
2. Esta práctica se evalúa sobre 2 puntos del global de la asignatura y vale aproximadamente un $57\,\%$ de la parte de prácticas de la asignatura.
3. No es necesario realizar todos los hitos para que la práctica pueda ser entregada.
4. Para la evaluación de cada uno de los apartados se tendrán en cuenta tanto la claridad del código como los comentarios incluidos en éste.
5. Para la corrección de la práctica se utilizará un programa de detección de copia. En caso de que la similitud entre dos o más prácticas se encuentren fuera de lo permisible, todas ellas serán calificadas con un 0 y aparecerá suspenso en la asignatura tanto en la convocatoria ordinaria como extraordinaria.
6. Fecha límite de entrega: Hasta el 22 de Mayo. Se habilitará una tarea en Campus Virtual para poder subirla.
7. Los códigos deben entregarse comprimidos en un único fichero .zip con el nombre del alumno. No olvides adjuntar todos los ficheros necesarios para su correcta ejecución, incluidos los ficheros proporcionados en Campus Virtual.

Objetivos de la práctica:

Un sistema de navegación es un sistema informático cuyo objetivo es asistir en la navegación (ya sea de vehículos terrestres, marítimos o aéreos). Estos sistemas de navegación pueden encontrarse completamente instalados dentro del vehículo que controlan o pueden estar ubicados en otro lugar diferente y hacer uso de la transmisión de radio u otro tipo de comunicación para enviar los comandos de control al vehículo.

Dentro de las posibilidades de estos sistemas de navegación, destacan en el uso cotidiano el caso de los dispositivos de navegación por satélite. Los dispositivos de navegación modernos usan la señal de la constelación de satélites GPS (Estados Unidos), GLONASS (Rusia), BDS (China) o Galileo (Europa) entre otros, para determinar la ubicación de un vehículo y dar indicaciones sobre la ruta a seguir.

En esta práctica nos centraremos en el estudio y diseño de una pequeña parte de un dispositivo de navegación GPS. Más concretamente, nos centraremos en el problema de representar en un ordenador la red de calles de una ciudad y el diseño de un programa para obtener la ruta óptima que debe seguir un vehículo para llegar desde un punto A hasta otro punto B.

Los dispositivos modernos cuentan con conexión a internet, lo que les permite conocer el tráfico en una ciudad en tiempo real. Por este motivo, en esta práctica simularemos el transito de vehículos entre los distintos puntos de una ciudad. Usando este tráfico simulado, podremos realizar un análisis de intervenciones en la red viaria, respondiendo así preguntas como: ¿qué calle debo cortar si tenemos una obra?, ¿es conveniente pasar una calle a doble sentido? ó ¿es conveniente invertir el sentido de otra calle? Finalmente, mejoraremos el diseño del programa GPS desarrollado anteriormente para obtener una ruta óptima entre dos puntos, teniendo en cuenta el tráfico dentro de la ciudad.

1. Reconstruir la red de calles en una ciudad

El primer paso a la hora de construir un dispositivo de navegación para una ciudad es el de almacenar la red de calles de dicha ciudad en el dispositivo. Para ello, se debe estudiar como se puede representar el mapa de la red de calles en el sistema informático. La forma más común de trabajar con estas redes de calles y carreteras es mediante el uso de **grafos**.

Recordemos que un grafo es un objeto matemático formado por un conjunto de objetos llamados vértices (o nodos) que se encuentran unidos por enlaces denominados aristas (o arcos). Estas aristas permiten representar las relaciones binarias entre distintos elementos del conjunto de vértices. En nuestro caso, las aristas representan las carreteras, mientras que los vértices son las intersecciones, es decir, todos los puntos donde es posible elegir qué camino tomar. La Figura 1 muestra un ejemplo de la representación de un mapa utilizando un grafo. En esta figura los círculos rojos representan los vértices y las lineas marrones representan las aristas.

Figura 1: Ejemplo de la representación de un mapa usando un grafo. (Fuente: OpenStreetMap)

En esta práctica utilizaremos como base los mapas de OpenStreetMap, un proyecto colaborativo para crear mapas editables y libres que se distribuyen como datos vectoriales bajo licencia abierta. Para ello, podemos ir a la página del proyecto https://www.openstreetmap.org desde donde descargar tanto la imagen del mapa deseado (en formato JPG, PNG, SVG) como exportar los datos del mapa en un formato propio de OpenStreetMap (OSM). Este formato OSM contiene toda la información sobre el mapa descargado (calles, carreteras, edificios, servicios, aparcamientos, transporte público, parques, etc).

A partir de los datos exportados en formato OSM es posible filtrar únicamente las calles e inferir cuales serán los vértices y cuales las aristas en nuestro grafo. Esta es posiblemente la parte más complicada y por este motivo se da resuelta en esta práctica. No obstante, si el alumno está interesado en saber como filtrar estos datos, se ha utilizado el paquete de Python

OsmToRoadGraph[1]. Este paquete permite, dado como entrada un mapa en formato OSM, obtener dos ficheros de salida: uno con los distintos vértices y aristas y otro fichero con el nombre de las calles por las que transcurren las aristas (esto se explicará más adelante). El comando exacto que se ha utilizado para obtener los datos de esta práctica es:

```
$ python3 run.py -f mapa-descargado.osm -n c
```

donde `mapa-descargado.osm` es el mapa que se ha descargado en formato OSM. Este proceso **ya se ha realizado** y **no debe** realizarse por el alumno y se detalla únicamente para explicar el proceso completo seguido para obtener el mapa.

Realizado el proceso anterior, para cada mapa se obtienen cuatro archivos que son proporcionados como material para el alumno. Todos estos ficheros se encuentran en la carpeta `/data` del fichero ZIP proporcionado junto al enunciado de la práctica en Campus Virtual. Por ejemplo, para el mapa llamado **ESI**, se proporcionan los ficheros:

1. `ESI.osm`. Es el fichero OSM original del cuál se han extraído todos los datos necesarios para la práctica. Se proporciona únicamente para curiosidad del alumno, y **no es necesario para el desarrollo de la práctica**.
2. `ESI.png`. Este fichero contiene el mapa anterior en formato de imagen PNG. Se utilizará como fondo cuando se vaya a realizar el grafo, proporcionando así contexto visual a las calles dibujadas.
3. `ESI.pycgr`. Este fichero se ha obtenido del paquete Python anterior y contiene los vértices y aristas del grafo.
4. `ESI.pycgr_names`. Este fichero contiene el nombre de las calles por las que transita cada arista.

A continuación, explicaremos la estructura de los ficheros `.pycgr`. Las primeras 7 líneas de estos archivos comienzan con una almohadilla (#) y contiene un resumen de la estructura del archivo. La primera línea sin el signo # será un entero que contendrá el número de vértices (a este número lo denominaremos V). La siguiente línea contiene el número de aristas del fichero (denominamos a este valor E). A continuación tenemos V líneas con la estructura

```
<id> <lat> <lon>
```

donde cada línea representa cada uno de los vértices del grafo y contiene 3 valores numéricos: `<id>` es un valor entero que contiene el identificador del vértice; y `<lat>` y `<lon>` son dos valores decimales (en punto flotante) que contienen la latitud y la longitud del vértice, respectivamente, es decir, su posición sobre en el mapa. Finalmente, tenemos E líneas con la estructura

```
<source_node_id> <target_node_id> <length> <street_type> <
    max_speed> <bidirectional>
```

donde cada línea representa cada uno de las aristas del grafo y contiene 6 valores numéricos: `<source_node_id>` y `<target_node_id>` son valores enteros que contienen los identificadores del vértice de origen y destino, respectivamente; `<length>` es un valor decimal que contiene la longitud de la arista en metros; `<street_type>` es un identificador del tipo

[1] https://github.com/AndGem/OsmToRoadGraph

de carretera[2]; <max_speed> indica la máxima velocidad permitida en la calle (en km/h); y <bidirectional> es un valor que será 0 si la calle solo tiene un sentido (del vértice de origen al vértice de destino) o un 1 si la calle es de doble sentido.

En el caso de los ficheros .pycgr_names, estos contienen E líneas en los que la línea e contiene el nombre de la calle por la que transita la arista e.

Se proporciona la función load_pycgr que el alumno puede utilizar para cargar los ficheros anteriores en MATLAB. Esta función recibe como parámetros de entrada el directorio en el que están contenidos los datos (por ejemplo: 'data/') y el nombre del fichero que contiene el mapa, sin extensión (por ejemplo: 'ESI' o 'RondaCiudadReal'). Esta función devolverá 4 variables (obsérvese su implementación en el fichero .m): el número de vértices, una estructura con todos los vértices, el número de aristas y una estructura con todas las aristas.

A modo de ejemplo, suponiendo que se han almacenado los vértices en una estructura denominada edges, si ejecutamos:

```
>> edges.length
```

obtendremos un vector con la longitud de cada uno de los vértices, dónde por ejemplo:

```
>> edges.length(40)
```

contendrá la longitud de la arista 40.

Utilizando estas estructuras y los conceptos aprendidos en las sesiones de grafos, podemos construir un **grafo no dirigido** que utilizaremos para visualizar las calles. Para ello, los pasos serán los siguientes: Primero construir el grafo no dirigido; segundo crear una figura en MATLAB y un objeto *axes* a partir de esta figura; después mostraremos en la figura recién creada el mapa de la ciudad (en formato PNG) utilizando para ello la función show_map proporcionada en Campus Virtual; finalmente dibujaremos el grafo generado encima del mapa y posicionaremos cada uno de los vértices utilizando la longitud y la latitud.

Obsérvese que los vértices y aristas dibujados deben quedar en los lugares correctos del mapa, de no ser así, posiblemente se esté cometiendo algún error a la hora de establecer las coordenadas de los vértices.

Por último, observemos que la red de calles en una ciudad es en realidad un grafo dirigido, donde la dirección de las aristas representa el sentido de circulación por cada una de las calles. La construcción del grafo no dirigido es una simplificación que nos permitirá dibujarlo correctamente en MATLAB, no obstante, debemos definir un segundo grafo (el cuál si será dirigido) y se utilizará en los hitos posteriores para el cálculo de rutas y flujos de tráfico. Si no hacemos esto, corremos el riesgo de enviar vehículos por calles en sentido prohibido.

Para realizar el **grafo dirigido**, nos centraremos en el campo bidirectional de la estructura que contiene las aristas. Si este campo tiene un valor 0, nos indica que la arista es de un solo sentido, y el sentido de esta será del vértice de origen al vértice de destino. Por lo tanto, podríamos crear directamente el digrafo en MATLAB especificando cuales son los vértices de origen y de destino. Sin embargo, también tenemos aristas con el valor del campo bidirectional en 1, lo que representa calles de doble sentido. Por ello, antes de crear el digrafo, tendremos que obtener las aristas de doble sentido y para cada una de ellas añadir una nueva arista en sentido contrario. Para ello, podemos ir añadiendo nuevas componentes a los vectores que hemos obtenido de la estructura de aristas que cargamos del fichero .pycgr,

[2] https://wiki.openstreetmap.org/wiki/Key:highway

en las cuales copiaremos los mismos valores que las aristas originales pero intercambiando los vértices de origen y destino.

Nota: Es necesario almacenar ambos grafos: el grafo no dirigido se utilizará a lo largo de los distintos hitos para realizar las visualizaciones, mientras que el grafo dirigido se utilizará para ejecutar todos los algoritmos.

1.1. Hito 1 (0.25 puntos)

a) Utilizando como base el archivo `Hito1.m` proporcionado, se pide implementar un script de MATLAB llamado `Hito1.m` que permita generar la red de calles en una ciudad y representarlas sobre un mapa. Para ello, se debe crear un grafo no dirigido (llámalo `G_visual`) y otro dirigido (llámalo `G`) siguiendo los pasos explicados en esta sección. Finalmente, se utilizará el grafo no dirigido para realizar la visualización de las calles encima del mapa de la ciudad.

b) Utilizando el script `Hito1.m`, visualiza los tres mapas proporcionados en la carpeta `data/` (**ESI**, **RondaCiudadReal** y **CiudadReal**).

2. Diseño de un sencillo GPS

En el Hito 1 hemos visto como podemos representar matemáticamente la red de calles de una ciudad, y como podemos visualizarla en el ordenador. A continuación, vamos a realizar un sencillo programa que nos permitirá calcular la ruta óptima entre dos lugares del mapa.

Cuando tenemos que elegir entre dos caminos posibles, podemos utilizar varias métricas que nos ayuden a tomar una decisión: podemos elegir el camino más rápido hasta el destino, el más corto, el que contenga menos intersecciones, el que tenga menor coste (por ejemplo, imagine tener que pagar un peaje por utilizar un camino determinado), etc. Normalmente, solemos tener en cuenta mayoritariamente el camino que nos lleva primero al destino. Los factores que entran en juego son muchos: la velocidad máxima, el tamaño de la carretera, la presencia de semáforos, el tráfico, etc. Podemos resumirlos todos en un solo valor, el tiempo (estimado) de viaje en esa sección. Esta información se integra en las aristas de nuestro grafo en forma de pesos y, en consecuencia, tendremos un grafo ponderado (o etiquetado). Por ejemplo, si se escribe un "4" en la arista que conecta el vértice X y el vértice Y, esto indica que para ir de la intersección X a la intersección Y estimamos que se necesitan 4 minutos.

Google Maps (y el resto de herramientas GPS) se basa en un algoritmo muy sencillo pero increíblemente eficaz: el algoritmo de Dijkstra. Este algoritmo toma el nombre de su inventor, Edsger Dijkstra, uno de los pioneros fundadores de la informática moderna. Pero, ¿qué hace el algoritmo de Dijkstra? Dado un grafo ponderado, un vértice de partida y un vértice de destino, el algoritmo encuentra el "camino mínimo" que conecta los dos puntos, es decir, la secuencia de aristas que minimiza la suma de los pesos (positivos) y, por tanto, en el caso de los mapas, minimiza el tiempo de viaje estimado.

Se ha demostrado matemáticamente que Dijkstra siempre encuentra el camino más corto, siempre que haya al menos una ruta posible hasta él. Este tiene una cara negativa, de hecho seguro que te ha ocurrido alguna vez que al utilizar el navegador de Google Maps, este te ha enviado por caminos "alternativos" en contra del sentido común. Esto se debe a que Google

elige la ruta más corta aunque esta apenas ahorre unos segundos de viaje, ignorando si esta es más complicada para el conductor. Sin embargo, por otro lado, es extremadamente versátil ya que siempre es capaz de encontrar la ruta más rápida.

Por lo tanto, volviendo a nuestro grafo, el primer paso será añadir pesos a cada una de estas aristas. Recordemos que para cada arista disponemos de su longitud, del tipo de vía, la velocidad máxima, si es de doble sentido y su nombre. Utilizando la distancia y la velocidad máxima, podemos inferir el tiempo que un vehículo tardaría en recorrer la arista en condiciones óptimas (suponiendo una densidad de tráfico baja, que el coche circula a la velocidad máxima permitida, semáforos en verde, ausencia de atascos, etc). Para ello, solo debemos operar con los vectores de longitud y velocidad. **Recuerda** que la longitud está expresada en metros y la velocidad en kilómetros por hora. Asumiendo que se viaja constantemente a un $90\,\%$ de la velocidad máxima, **se debe calcular** el tiempo estimado para cada arista **expresado en minutos**. Una vez calculado el tiempo de viaje de cada arista, construiremos un nuevo digrafo (similar al construido en el Hito 1) pero usando esta magnitud como peso para las aristas.

Si observamos la estructura del digrafo recién creado, vemos que este contiene dos tablas, una de aristas (`Edges`) y otra de vértices (`Nodes`). La tabla de aristas solamente contiene los vértices de origen y destino de cada arista y su peso asociado (recordar que el peso lo hemos definido como el tiempo de viaje en minutos circulando al $90\,\%$ de la velocidad máxima). Lo que haremos ahora, será añadir a esta tabla de aristas del grafo los valores de **longitud** (`length`), **velocidad máxima** (`maxspeed`) y **nombre de la calle** (`name`), los cuales nos serán de utilidad más adelante. Sin embargo, hay que tener cuidado ya que el orden de las aristas en esta tabla no es el mismo que en la estructura de aristas que cargamos del fichero, ya que al crear el digrafo MATLAB reordena las aristas.

Para solventar esto, utilizaremos la función de MATLAB `findedge(G, source, target)`[3] dónde, dados un vértice (o un vector de vértices) de origen (`source`) y de destino (`target`) en un grafo (`G`), obtenemos el índice (o vector de índices) de la tabla en los que se encuentran estas aristas, es decir, la fila en la que se encuentran las aristas que van desde `source` hasta `target`. Utilizando estos índices, podemos buscar cada una de las aristas cargadas del fichero .pycgr en la tabla de aristas de digrafo y asociarles los valores de longitud, velocidad máxima y nombre de la calle que le correspondan (creando para ello nuevas columnas en la estructura `G.Edges`).

Nota: A modo de ejemplo, si queremos asignar a la arista con índice número 5 (es decir, la situada en la fila 5) valor `length`$= 2{,}25$, usaremos:

```
1 >> G.Edges.length(5) = 2.25
```

También se puede trabajar con vectores en vez de con valores individuales (evitandote así utilizar bucles).

Construido el grafo, solo nos faltará localizar el índice de los vértices de origen y destino y posteriormente aplicar el algoritmo de Dijkstra[4] para obtener el camino mínimo entre ambos vértices. **Cuidado**, como estamos trabajando con un grafo dirigido, el camino mínimo será distinto si permutamos los vértices de origen y destino.

Obtenido el camino mínimo, utilizaremos un **grafo no dirigido** para construir una figura que muestre el mapa de la ciudad y sus calles (como se hizo en el Hito 1), **resaltando claramente** cual es **vértice de origen**, el **vértice de destino** y el **camino mínimo** seguido.

[3] Revisa https://es.mathworks.com/help/matlab/ref/graph.findedge.html
[4] Revisa https://es.mathworks.com/help/matlab/ref/graph.shortestpath.html

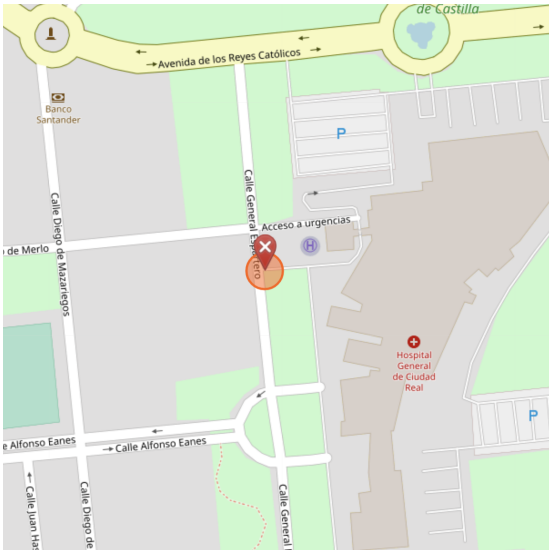
Para terminar, se pide mostrar por consola el tiempo de viaje estimado (se corresponde con el valor del peso total del camino obtenido utilizando Dijkstra) y la distancia aproximada de este camino, para lo cual se deberá sumar la distancia de cada una de las aristas atravesadas en este camino mínimo.

2.1. Hito 2 (0.5 puntos)

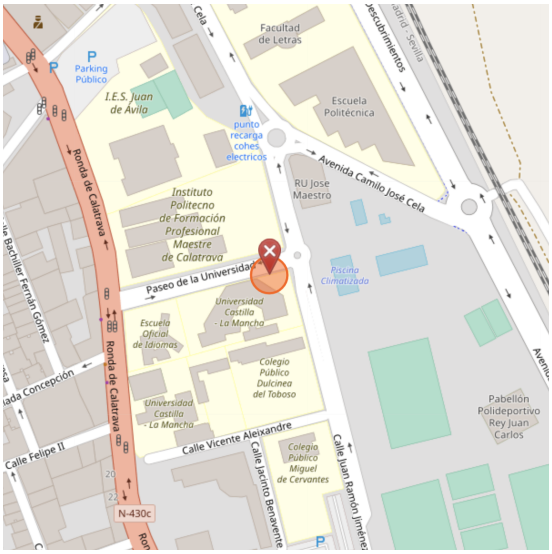
a) Utilizando como base el archivo `Hito1.m` que creaste anteriormente, crea una copia llamada `Hito2.m`. Este script debe buscar la ruta óptima entre dos puntos (en base a lo explicado en esta sección), así como su representación gráfica. Además, se debe mostrar por consola de comandos la distancia de la ruta óptima (en Km) y el tiempo estimado (en minutos) de viaje.

b) El script `Hito2.m` deberá obtener las siguientes rutas óptimas en el mapa denominado **CiudadReal**:

1. Ruta desde la puerta de urgencias del Hospital General Universitario de Ciudad Real hasta la entrada principal de la Escuela Superior de Informática (ESI). Véase las Figuras 2a y 2b para ayudarte a identificar los vértices de origen y destino.



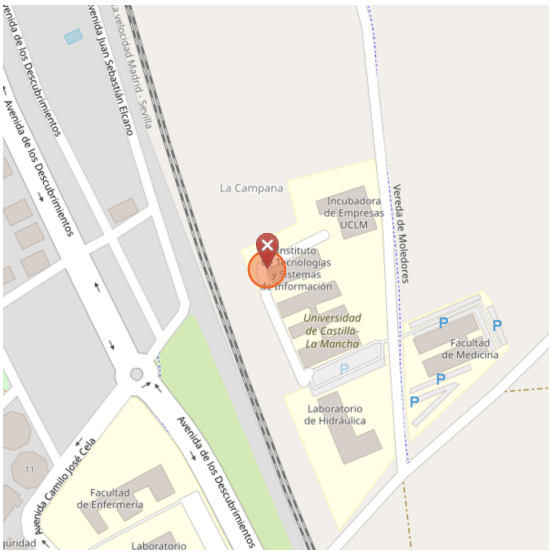
(a) Urgencias del Hospital



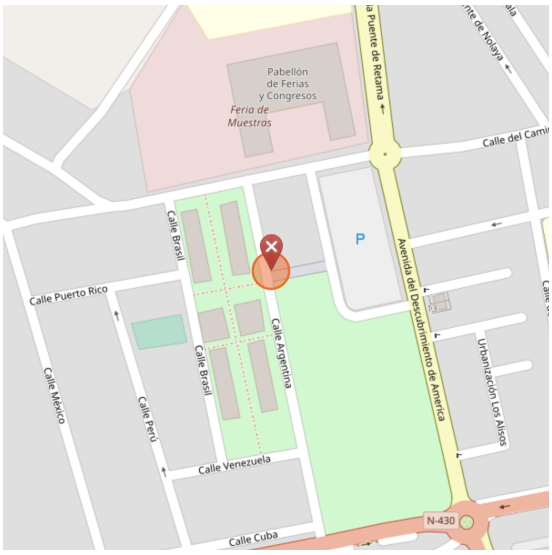
(b) ESI

2. Obtén la ruta desde el Instituto de Tecnologías y Sistemas de Información (ITSI) hasta el Auditorio de la Granja. Véase las Figuras 3a y 3b para ayudarte a identificar los vértices de origen y destino.
3. Obtén la ruta inversa a la anterior, es decir del auditorio al ITSI.

Nota: Crea una figura nueva para cada ruta. No olvides resaltar el recorrido óptimo, así como los puntos de origen y destino. Por último, muestra por consola la distancia y el tiempo de viaje estimado para cada una de estas rutas. Puedes obtener el índice de los vértices de origen y destino situando el cursor del ratón sobre estos hasta que aparezca.



(a) ITSI



(b) Auditorio de la Granja

3. Simular el tráfico en una ciudad

En el Hito 2 hemos construido un sencillo navegador GPS. Aunque su funcionamiento es la base de sistemas más complejos como por ejemplo Google Maps, no deja de ser una simplificación de estos sistemas. En este hito, añadiremos una nueva dimensión que añadirá realismo a nuestro programa.

Se puede añadir información adicional, como la información sobre el tráfico o la presencia de atascos, simplemente modificando los pesos en el grafo. Pero, ¿cómo conoce Google la información sobre el tráfico? Para ello, utiliza los datos recogidos por otros usuarios que están utilizando la aplicación en tiempo real. De hecho, cada vez que usas Google Maps, o circulas por una ciudad con un dispositivo Android, Google está recolectando tu posición, así como la velocidad a la que te estás desplazando, y envía de manera anonimizada estos datos a sus servidores, los cuales utiliza en tiempo real para determinar la congestión en cada punto de la ciudad. Dado que no podemos acceder a esta información, nosotros vamos a simular el tráfico de la ciudad.

El nivel de congestión en una red de tráfico es el resultado de la interacción entre la red viaria (oferta) y la demanda de la misma. En los modelos de planificación de tráfico se representa la demanda a través de la llamada *matriz origen-destino (O-D)*. Esta matriz se construye en un proceso de varias etapas. En la primera etapa (generación/atracción) se zonifica la ciudad y se mide el potencial de cada una de estas zonas para atraer o generar viajes. Estos valores dependerá de la población que reside, el número de empleos localizados en la zona, plazas escolares, etc. Tras evaluar estas características se estima el número de viajes que son capaces de generar y de atraer cada una de estas zonas. Estas zonas son abstraídas mediante un conjunto de vértices *centroides* que actúan como fuentes/sumideros que inyecta/reciben flujo de tráfico de la red. En el fichero adjunto **matrizOD.xlsx** se ha recogido esta información para el mapa **CiudadReal**. La primera columna contiene el id del vértice que actúa como centroide, la segunda columna su capacidad atractora de viajes y en la tercera el número de viajes que genera.

En una segunda etapa (distribución) se busca en saber cuantos viajes se realizan de una zona determinada a otra y en la última etapa (partición modal) se determina los modos de

transporte para realizar estos viajes. En esta práctica consideraremos que la matriz origen-destino para los viajes realizados en vehículo privado se obtienen mediante la expresión:

$$d_{ij} = \frac{G_i A_j}{D} \text{ con } i \neq j \tag{1}$$

donde d_{ij} es la demanda de la zona i a la zona j, G_i es el número de viajes generados en la zona i, A_j es el número de viajes atraídos por la zona j y D es la demanda total, esto es $D = \sum_i G_i = \sum_j A_j$. Notad que se han eliminado los viajes intrazonales d_{ii} (viajes con el mismo origen y destino). La expresión (1) equivale a que los viajes se realizan aleatoriamente de una zona a otra.

La superposición de todos los caminos empleados en la red determinarán el nivel de uso de las aristas de la red. Esta operación la denominamos la asignación de la matriz $O - D$ a la red. Suponemos que **los vehículos viajan por el camino mínimo** (respecto al tiempo de viaje) y que para cada par (i, j) de la matriz $O - D$ se ha calculado este camino mínimo p_{ij}. Definimos el parámetro

$$\delta_{a,p_{ij}} = \begin{cases} 1 & \text{si la arista } a \text{ está en el camino } p_{ij} \\ 0 & \text{en caso contrario} \end{cases}$$

entonces el número de vehículos que pasan por la arista a viene expreEsado por

$$f_a = \sum_{(i,j)} \delta_{a,p_{ij}} d_{ij} \text{ para toda arista } a \tag{2}$$

La expresión (2) indica que el flujo en la arista a es el flujo de todos los caminos que pasan por a. Si el camino no pasa por la arista sumamos 0 pero si el camino mínimo pasa por la arista sumamos su flujo d_{ij}. La mejor forma es crear una nueva columna en el campo `Edges` del digrafo G que se llame `Flow` y contenga el valor f_a.

Para visualizar el flujo de cada arista en el mapa debemos trasladar este campo `Flow` del digrafo al grafo no dirigido que utilizamos para las visualizaciones. Sin embargo, el orden de la tabla `Edges` **es distinto** en ambos grafos (ya que MATLAB reordena las aristas). Utilizando la función `findedge` podemos buscar que aristas del digrafo se corresponden con las aristas del grafo no dirigido, y así copiar estos valores del campo `Flow`. Se debe **tener cuidado** ya que habrá aristas en el grafo no dirigido que se correspondan con dos aristas en sentido opuesto en el digrafo (calles de doble sentido). En estos casos, el valor del campo `Flow` en el grafo no dirigido será la suma de ambas aristas en el digrafo.

3.1. Hito 3 (0.5 puntos)

a) Calcular la matriz origen-destino $\{d_{ij}\}$ empleando la expresión (1) y el fichero *matrizOD.xlsx* adjuntado. Recordar que la instrucción de Matlab para leer ficheros excel es *xlsread*.

b) Asignar esta matriz a la red del mapa **CiudadReal** y calcular el flujo en las aristas de la red (utilizando el grafo dirigido construido en los hitos anteriores).

c) Representar las aristas de la red de manera proporcional al flujo que soportan (a mayor flujo mayor tamaño del ancho de la arista).

Nota: Para modificar el tamaño del ancho de las aristas en una visualización de MATLAB se puede modificar el parámetro `LineWidth` del plot[5].

4. Análisis de intervenciones en la red

Utilizando los conceptos de los hitos anteriores, en esta sección vamos a utilizar nuestro sistema para evaluar una serie de alteraciones sobre la red de calles de Ciudad Real, analizando las consecuencias que estas tendrían.

El ayuntamiento de Ciudad Real está estudiando una serie de alternativas para reducir la contaminación producida por los vehículos que transitan el casco urbano. Actualmente se proponen 3 escenarios posibles:

1. Peatonalizar la "Calle Elisa Cendreros" y la "Calle de la Paloma".
2. Reducir la velocidad máxima de toda la "Ronda" a 30 Km/h.
3. Cambiar el sentido de circulación de la "Calle Toledo".

La Figura 4 muestra las calles afectadas por cada uno de estos escenarios: Escenario 1 (color red), escenario 2 (color magenta), escenario 3 (color verde).

Para implementar estos escenarios se deben obtener las aristas por las que transitan las calles afectadas y realizar los cambios necesarios sobre el grafo para implementar el escenario. Para obtener las aristas de las calles afectadas, se propone una posible alternativa que consiste en utilizar el campo `name` con el nombre de la calle que almacenamos en la estructura de aristas del digrafo en el Hito 2 para buscar las aristas que pertenecen a estas calles. Para localizar las calles "Calle Elisa Cendreros", "Calle de la Paloma" y "Calle Toledo" puede usarse el comando `find` y el operador ==. En el caso de la Ronda de Ciudad Real, como esta está compuesta por varias Rondas con distintos nombres, se puede utilizar el comando `startsWith` junto con `find` para localizar todas las calles que empiezan por la palabra "Ronda".

Para invertir la dirección de una arista una opción es usar `flipedge`, aunque también se pueden modificar las aristas manualmente permutando los vértices de origen y destino.

Cuidado: Crea un nuevo digrafo para cada cambio a implementar, de manera que cada cambio no afecte al digrafo original ni al resto de digrafos planteados para cada escenario.

4.1. Hito 4 (0.5 puntos)

a) Basándote en los hitos anteriores, crea un nuevo script denominado `Hito4.m` en el que implementes cada uno de estos nuevos escenarios.

b) El tiempo total de viaje en la red se calcula mediante la expresión:

$$T = \sum_a f_a t_a \tag{3}$$

[5] https://es.mathworks.com/help/matlab/ref/graph.plot.html#buzf5jn

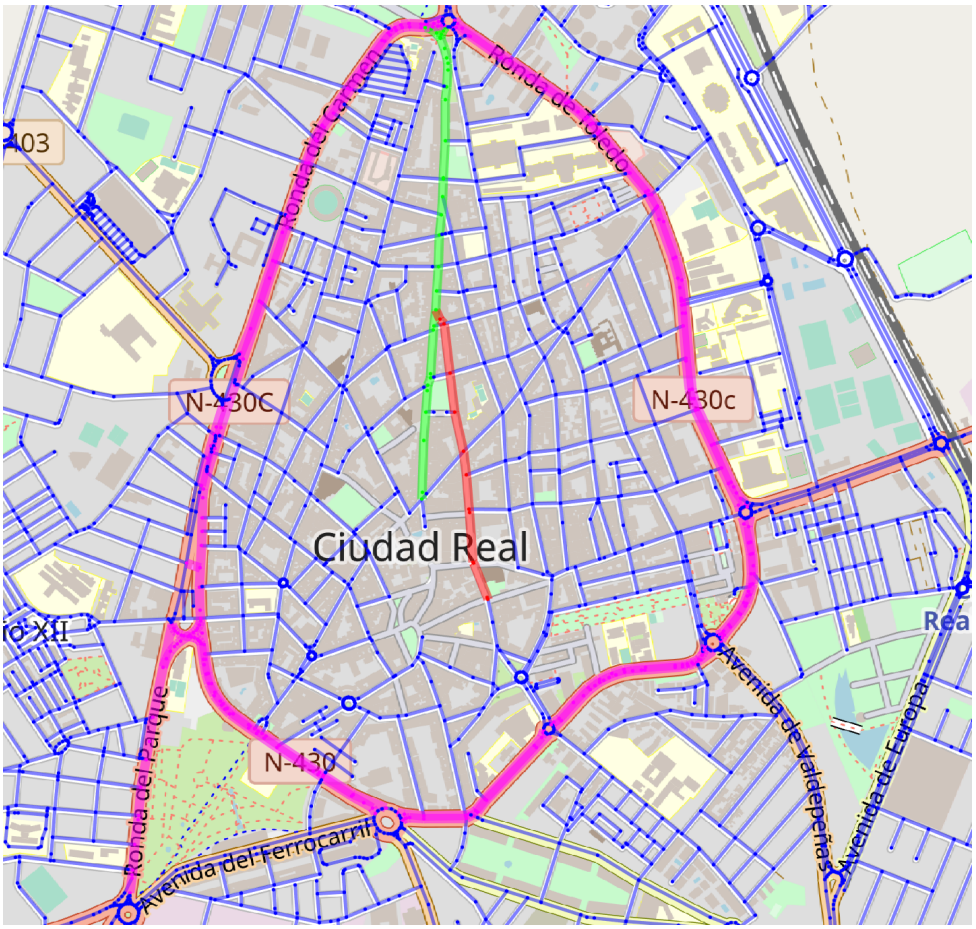


Figura 4: Calles afectadas por las medidas del Hito 4. (Fuente: OpenStreetMap)

donde t_a es el tiempo en atravesar la arista a (circulando al $90\,\%$ de la velocidad máxima) y f_a es el flujo de tráfico en la arista a (calculado usando la expresión 2). Las anteriores intervenciones modificarán el flujo y/o el tiempo de viaje en la red y por tanto el tiempo total. Utilizando MATLAB, rellena la siguiente tabla[6] con los tiempos totales de viaje para las diferentes intervenciones y la situación inicial y muéstrala por consola al ejecutar `Hito4.m`. Argumenta cual de los 3 escenarios propuestos tiene una menor repercusión en la red.

Escenario	T
Caso inicial	-?-
Escenario 1	-?-
Escenario 2	-?-
Escenario 3	-?-

5. Diseño de un GPS mejorado

El tiempo de viaje en las aristas depende del flujo que soportan. Cuando el flujo se aproxima la capacidad de la arista esta se empieza a congestionar incrementando los tiempos de viaje

[6] https://es.mathworks.com/help/matlab/ref/table.html

de la arista vacía. Introducimos la siguiente expresión para modelar este efecto y obtener el nuevo tiempo de viaje de la arista a:

$$t_a(f_a) = t_a^0 * \left(1 + 0{,}2 * \left(\frac{f_a}{k_a}\right)^4\right) \tag{4}$$

donde f_a es el flujo de la arista a, t_a^0 es el tiempo de viaje en la arista vacía y circulando al $90\,\%$ de la velocidad máxima (calculado en los hitos anteriores) y k_a es la capacidad de la arista. Asumimos que todas las aristas de la red tienen capacidad de $k_a = 500$ vehículos/hora.

El efecto de la congestión provoca que aparezcan como interesantes rutas que en un principio no eran los caminos mínimos. Supóngase un GPS que es capaz de estimar los flujos actuales en la ciudad. Se pide simular como cambiaría la propuesta de rutas con respecto a las obtenidas en el Hito 2.

5.1. Hito 5 (0.25 puntos)

a) Calcular los tiempos de viaje (pesos de las aristas) teniendo presente la congestión a través de la expresión (4).

b) Crea un nuevo script denominado `Hito5.m` dónde se repita el Hito 2 pero considerando estos nuevos tiempos de viaje. Dada la simulación del tráfico planteada en el Hito 3, ¿se produciría un cambio de ruta para un nuevo usuario que dese realizar estos itinerarios? ¿Qué rutas se ven alteradas y por qué? ¿Cambia el tiempo de viaje de las rutas no alteradas? ¿Por qué?