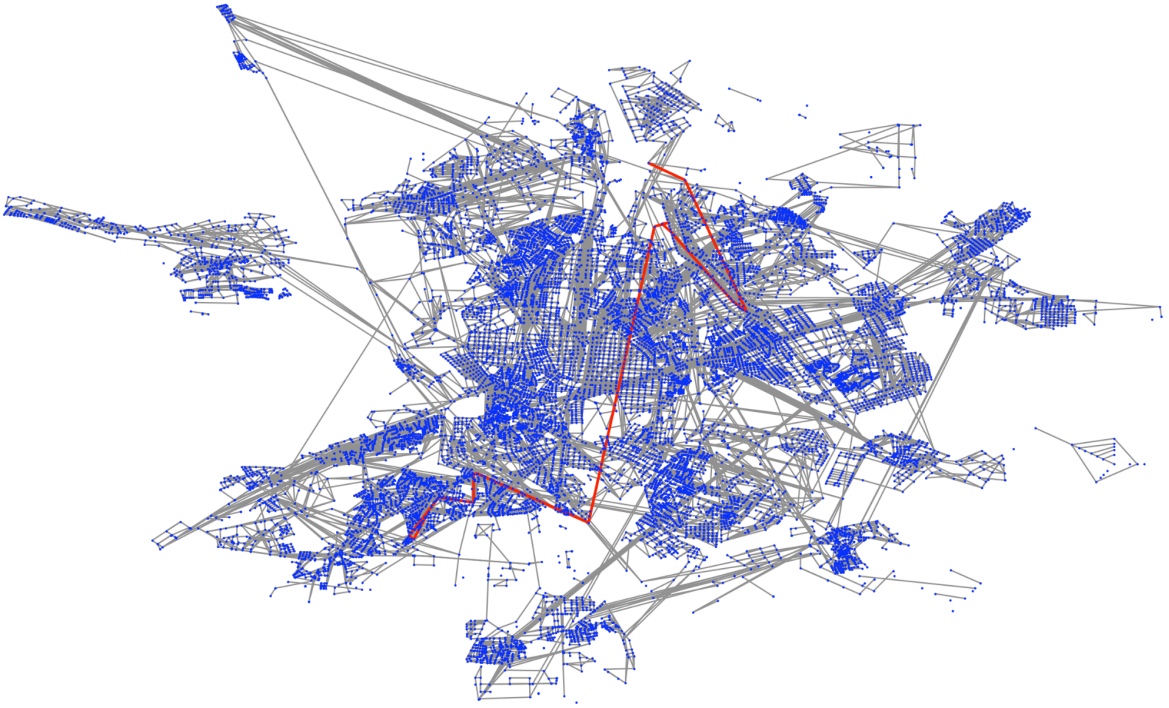
Práctica 3

Navegador GPS



Integrantes: Jorge Ibinarriaga Robles y Miguel Ángel Huamani Salinas

Grupo: GP02B

Introducción

Esta práctica consiste en un navegador del callejero de Madrid, que usa algoritmos de Recorridos de Grafos, como el Algoritmo de Dijkstra, para encontrar el camino mínimo entre dos direcciones de Madrid.

Reconstrucción del grafo a partir del dataset (callejero.py)

La librería **callejero.py** se encarga de procesar los datos de los datasets de las direcciones y los cruces de callejero de Madrid para posteriormente crear el grafo del callejero.

Primeramente, se unifican todos los cruces que tengan coordenadas muy próximas, como por ejemplo, una glorieta. Esto se hace calculando la distancia desde cada cruce al resto de cruces y comprobando que esta distancia es inferior a un radio determinado R. En caso de que si sea inferior, los unifica en la lista cruces_cercanos y elimina todos aquellos cruces que en realidad son el mismo cruce. A este cruce restante se le asigna como coordenadas la media de las coordenadas de todos los cruces unificados.

Para procesar los datos, se crean dos clases Cruce y Calle que contienen la información necesaria para construir el grafo.

Cada clase Cruce tiene como atributo una lista de todas las calles que contiene y sus coordenadas y la clase Calle contiene todos los cruces que pasan por dicha calle asicomo el nombre de la calle.

Los cruces se almacenan en un diccionario cuyas claves son las coordenadas del cruce y el valor es el objeto cruce.
Las calles se almacenan en otro diccionario con clave el código de vía, que identifica de forma única a cada calle y como valor el objeto calle.

Para construir el grafo, se itera sobre todas las calles que tengan más de un cruce y se va agregando cada arista como sucesión de cada 2 cruces de la calle.

Estructura general del programa “gps.py”

El módulo gps.py alberga las distintas funciones que permiten el correcto comportamiento del navegador GPS, cuyo objetivo principal es el de calcular y guiar una ruta para el usuario del programa, desde un punto de origen hasta un punto de destino en un sistema de calles interconectadas. Las funciones que componen gps.py son las siguientes:

- **distancia_entre_nodos**: Para la obtención de la distancia entre dos intersecciones, mediante el uso de las coordenadas cartesianas de estas.

- **crear_grafo_distancia**: crea un grafo ponderado donde los vértices representan las intersecciones (nodos) y las aristas representan las calles entre estas intersecciones. Las aristas tienen pesos que representan la distancia entre las intersecciones correspondientes.

- **crear_grafo_tiempo**: Similar a crear_grafo_distancia(), pero en este caso, las aristas tienen pesos que representan el tiempo de viaje estimado entre intersecciones, basado en la velocidad máxima permitida en las calles.

- **dibujar_grafo**, **dibujar_ruta**: funciones que usan la librería NetworkX para visualizar gráficamente el grafo y la ruta calculada. Muestran los nodos (intersecciones), las aristas (calles) y resaltan la ruta específica desde el origen hasta el destino.

- **cargar_direcciones**: Crea un diccionario de direcciones con nombres de calles como claves y coordenadas como valores.

- **obtener_informacion_direccion**, **encontrar_cruce_mas_cercano**: Estas funciones trabajan juntas para determinar el cruce más cercano a una dirección dada, usando las coordenadas de las direcciones y las intersecciones.

- **encontrar_ruta_minima**: Calcula la ruta mínima (ya sea en términos de distancia o tiempo) entre dos intersecciones (nodos), dependiendo del modo especificado.

- **rotonda**: Verifica si una intersección es una rotonda basándose en la cantidad de calles conectadas.

- **angulo_entre_vectores**: función que calcula el ángulo entre dos vectores con el uso de la arcotangente.

- **obtener_calle_arista**, **determinar_sentido_giro**: Ayudan a determinar la calle de una arista y el sentido de un giro en la ruta.

- **dirigir_ruta**: Esta es la función principal que se explicará más detalladamente en el apartado de “Recuperación de la ruta óptima y de las direcciones de navegación”.

Estructura general del programa “grafo.py”

La implementación principal del módulo grafo.py, es definir la estructura del grafo mediante la clase Grafo, la cual cuenta con atributos como si este se trata de un grafo dirigido o no, o una lista de la adyacencia del grafo (conjunto de pares arista y peso asociado a estas). Además el grafo cuenta con múltiples métodos como los siguientes:

- **Operaciones básicas**: como agregar_vertice, agregar_arista (junto con su respectivo peso), eliminar_vertice, eliminar_arista, obtener_arista (método que devuelve datos y peso), lista_vertices (lista con todos los vértices del grafo), lista_adyacencia (que dado un vértice del grafo devuelve su lista de adyacencia de este).

- **Grados de los vértices**: grado, grado_saliente y grado_entrante (para obtener el respectivo grado dado un vértice del grafo). En nuestro caso al tratarse de un grafo NO dirigido, los tipos de grados coinciden.

- **Algoritmos**: la clase Grafo también cuenta con los siguientes algoritmos:
    - ***dijkstra***: que proporciona el árbol de caminos mínimos a partir de un vértice de origen del grafo (se tiene en cuenta los pesos de la aristas). Lo que el método devuelve es un diccionario con pares clave-valor, donde el valor es el nodo padre de la clave.

    - ***camino_minimo***: devuelve una lista con los vértices ordenados que forman parte del camino más corto entre un origen y un destino determinados. Hace uso del árbol de caminos mínimos obtenido con el algoritmo de Dijkstra.

    - ***prim***: proporciona el árbol abarcador mínimo para el grafo mediante el algoritmo de Prim. También se devuelve en el formato diccionario.

    - ***kruskal***: también proporciona un árbol abarcador mínimo, mediante el algoritmo de Kruskal, sin embargo, el navegador gps no hace uso ni de este ni del algoritmo de Primo, sino que se centra en el uso del algoritmo de Dijkstra para buscar la ruta más corta al destino.

Recuperación de la ruta óptima y de las direcciones de navegación

En el contexto de este proyecto la ruta óptima entre un inicio y un destino es aquella que cuente con el conjunto de caminos más cortos entre ambos puntos. Los caminos más cortos son los de menor distancia, o equivalente a aquellos de menor peso en el grafo, por lo que el uso de la función def camino_mínimo para obtener el camino desde un origen hasta un destino, la cual a su vez usa el algoritmo de Dijkstra (def dijkstra) es vital. Ambas de estas funciones se encuentran en el módulo grafo.py, donde def dijkstra, primeramente nos devuelve el árbol de caminos mínimos (teniendo en cuenta el coste del camino) como diccionario, y def camino_minimo una lista con los vértices (cruces) que componen el camino. A partir de esta lista podemos ya proporcionar al usuario del programa una serie de direcciones de navegación para guiarle a su destino.

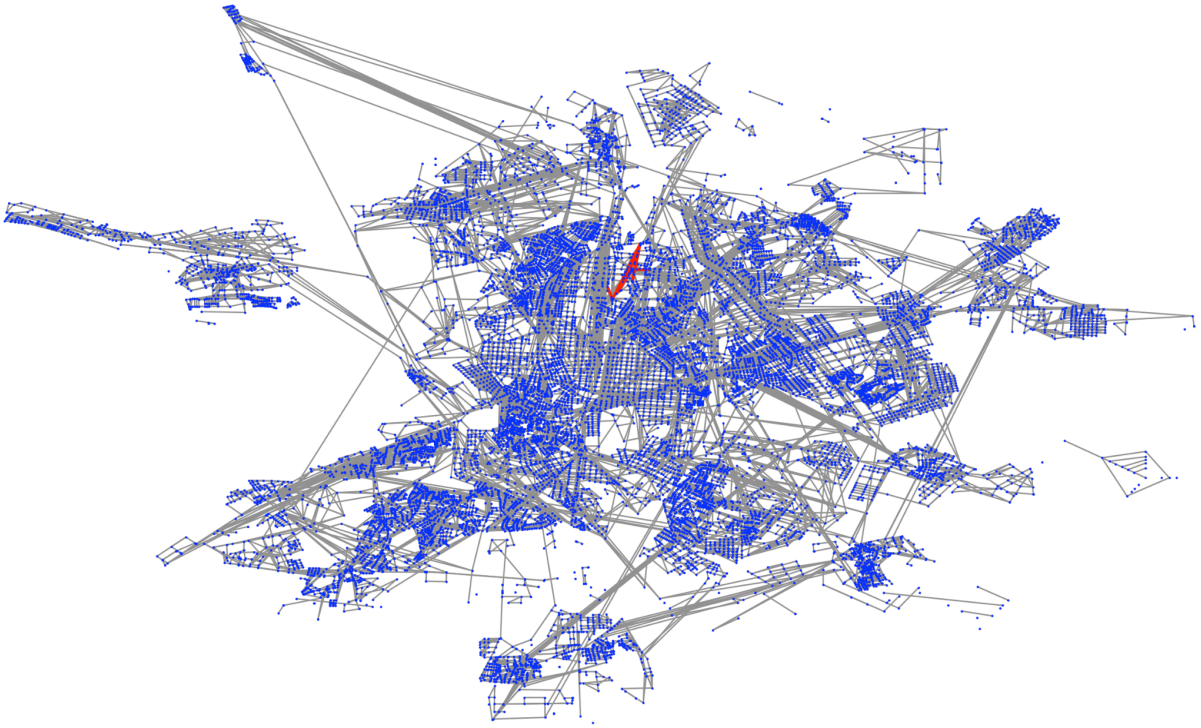
De esto último se encarga la función def dirigir_ruta del módulo gps.py. La función recibe como argumentos la dirección de origen, la dirección de destino y el modo, proporcionado por el usuario del programa. Primero se obtiene la información de origen y el cruce más cercano a este, junto con la distancia, a la vez que almacena el cruce más cercano a la dirección de destino y la distancia a este. Se calcula la ruta mínima con los algoritmos mencionados previamente, y para cada cruce de la ruta mínima se verifica si se trata del último nodo del camino, y si no, se obtienen tanto calle como distancia entre el cruce actual y el siguiente a este en la ruta. Se establece el caso en el que un cruce se trate de una rotonda, y si se da el caso se indica por que calle salir de esta. En caso contrario, se verifica si se debe continuar en linea recta o realizar un giro. Tras llegar al nodo final (destino), se indica la última dirección del navegador y se muestra por pantalla que el usuario ha llegado a su destino. La función devuelve tanto el camino como el grafo de este.

Ejemplo:
Origen: Calle Del Principe de Vergara 291
Destino: Calle Del Padre Damián 18

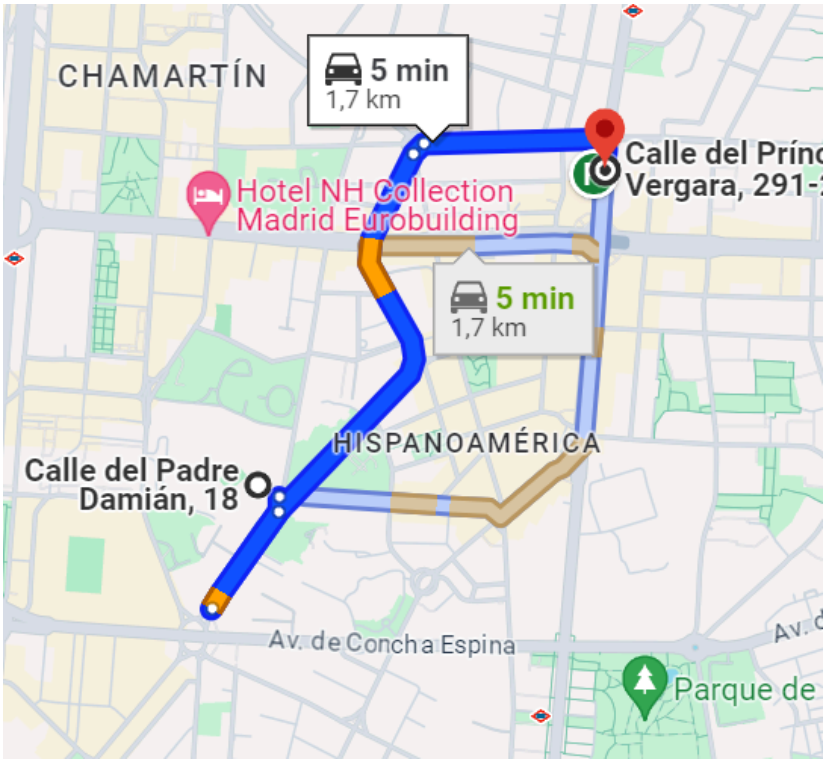
```
.....Bienvenido a GPS IMAT.....
Seleccione la dirección del origen: Calle Principe de Vergara
[?] Seleccione la dirección del origen: : CALLE DEL PRINCIPE DE VERGARA NUM000291
 > CALLE DEL PRINCIPE DE VERGARA NUM000291
   CALLE DEL PRINCIPE DE VERGARA NUM000289
   CALLE DEL PRINCIPE DE VERGARA NUM000287

Has seleccionado: CALLE DEL PRINCIPE DE VERGARA NUM000291
Seleccione la dirección del destino: Padre damian 18
[?] Seleccione la dirección del destino: : CALLE DEL PADRE DAMIAN NUM000018
 > CALLE DEL PADRE DAMIAN NUM000018
   AUTOVIA  A-1 KM.001000EN
   AUTOVIA  A-1 KM.001000SA

Has seleccionado: CALLE DEL PADRE DAMIAN NUM000018
Pulse S si desea encontrar la ruta más corta ó F si desea encontrar la ruta más rápida: F
Cargando ruta...
Continua por CALLE DEL PRINCIPE DE VERGARA 34.096922148487245 metros
En la rotonda, sal por AVENIDA DE ALFONSO XIII y continua 362.84001378017837 metros por AVENIDA DE ALFONSO XIII
En la rotonda, sal por PASEO DE LA HABANA y continua 838.2630862682669 metros por PASEO DE LA HABANA
Continua recto por PASEO DE LA HABANA 1320.9397797023148 metros
Continua recto por PASEO DE LA HABANA 881.0354127956492 metros
Continua recto por PASEO DE LA HABANA 1235.06409246646 metros
Continua recto por PASEO DE LA HABANA 1864.8904774811845 metros
En la rotonda, sal por CALLE DEL PADRE DAMIï¿½N y continua 415.8740012311421 metros por CALLE DEL PADRE DAMIï¿½N
Continua por CALLE DEL PADRE DAMIAN 37.887755277925876 metros
Ha llegado a su destino
```



Comparación con Google Maps:



Consideraciones

- Hemos tenido dificultades a la hora de codificar los datasets.
- Para la interfaz del programa se han usado las librerías difflib y inquirer para poder seleccionar una dirección de las direcciones del datasets de forma similar a un navegador real.

Bibliografía

- *La teoría de grafos: Mapas del metro y redes neuronales, Claudi Alsina, 2012.*
- *Cormen Leiserson y Rivest: "Introduction to Algorithms", MIT press, 1990.*
- *Aho, Hopcroft & Ullman. "Estructuras de Datos y Algoritmos". Pearson – Addison Wesley Longman, Primera Edición, 1998.*