EI1022/MT1022 - Algoritmia

Entregable 2 - Fecha de entrega: lunes 23 de noviembre de 2020

1. El problema: El viajante de comercio euclídeo

Dado un conjunto de puntos en el plano que interpretamos como los vértices de un grafo euclídeo totalmente conectado, queremos obtener ciclos hamiltonianos de longitud mínima.

Como el problema es NP-Completo, utilizaremos algoritmos voraces heurísticos. Deberás implementar los dos algoritmos que te pedimos más adelante y aplicarlos a una serie de instancias.

Se pide que implementéis un programa de línea de órdenes (`entregable2.py`) con las siguientes características:

a) El programa recibirá por la **entrada estándar** un fichero de texto con el siguiente formato:

- La primera línea contendrá un entero con el número de puntos del grafo.
- Cada una de las líneas siguientes tendrá dos números flotantes separados por un blanco, representando las coordenadas de un punto en el plano (x, y). En tu salida deberás identificar los puntos por el índice que tendrían si se almacenaran en una lista en el orden en que se leen del fichero. Por ejemplo, el punto 2 en la lista de puntos [(0.5, 50.5), (11.1, 10.6), (1.4, 3.5), (7.1, 9.2)] es el punto (1.4, 3.5).

  RECOMENDACIÓN: Es posible que utilices la estructura UndirectedGraph para construir el grafo de puntos. Te recomendamos que los vértices no sean directamente los puntos sino los índices de dichos puntos. Es decir, los vértices serán enteros (p. ej. con la lista anterior, te aconsejamos que guardes la arista ((0.5, 50.5), (1.4, 3.5)) como (0, 2)).

b) El programa deberá obtener, para los puntos de su entrada, los ciclos hamiltonianos que obtienen los dos algoritmos voraces que se especifican más adelante.

c) El programa deberá mostrar por la **salida estándar** cuatro líneas de texto, dos para cada algoritmo (primero irá el algoritmo 1). La primera línea tendrá la longitud del ciclo obtenido por el algoritmo. La segunda línea tendrá la lista, en notación de Python, de los índices de los vértices del ciclo. El primer elemento será el 0 y no se repetirá al final.

Como ejemplo, veamos la solución de la instancia `iberia.ins`:

```
12523.615361168399
[0, 1, 3, 7, 18, 35, 44, 54, 72, 81, 88, 98, 111, 101, 99, ..., 11, 6, 4]
12530.292586532008
[0, 4, 6, 11, 15, 9, 12, 32, 43, 53, 44, 35, 18, 54, 67, 65, ..., 7, 3, 1]
```

d) El tiempo de ejecución en `lynx.uji.es` del programa para una instancia de hasta **1000 puntos** no debe ser superior a **cuatro segundos**. Si lo sobrepasa, se considerará que dicha instancia no ha sido superada.

Como respuesta a la tarea correspondiente en el aula virtual deberéis subir un fichero comprimido con el nombre `alxxxxx_alyyyyy_e2.zip` (modificando los `al?????` por los de los dos miembros del grupo). Dicho fichero comprimido contendrá el fichero `entregable2.py` junto con los ficheros adicionales (propios) que sean necesarios para su funcionamiento (no debéis incluir ni la biblioteca `algoritmia`, ni `easycanvas`, ni ninguno de sus ficheros).

2. Algoritmos solicitados

Nuestros dos algoritmos voraces se basan en modificar el proceso de construcción de un árbol de recubrimiento mínimo. La modificación consiste en añadir la restricción de que cada vértice sólo puede aparecer, como máximo, en dos aristas. Cuando termine el proceso de construcción todos los vértices aparecerán en dos aristas, excepto dos, que aparecerán sólo en una. Ambos serán el primer y el último vértice de un camino que pasa por todos los vértices. Podemos convertir dicha secuencia en un ciclo hamiltoniano si conectamos ambos vértices con una arista.

Algoritmo 1: Siguiendo la idea anterior, modifica el algoritmo de **Kruskal** para encontrar un ciclo hamiltoniano.

Algoritmo 2: Siguiendo la idea anterior, modifica el algoritmo de **Prim** para encontrar un ciclo hamiltoniano. Utiliza el vértice 0 como vértice inicial en el algoritmo de Prim. En este caso, las diferencias con el algoritmo original son tantas que es mejor entender primero el funcionamiento de Prim con la restricción, y después implementar el algoritmo desde cero. Para entender lo que hace la versión modificada, os recomendamos hacer una traza visual sobre un grafo sencillo.

3. Prueba del programa. Instancias de prueba.

En el aula virtual, junto al enunciado, puedes descargarte el paquete `test_e2.zip`. Este paquete contiene un conjunto de instancias de prueba (`*.ins`) junto con sus respectivas soluciones (`*.sol`). Ambos tipos de fichero son ficheros de texto.

En este entregable vuestras soluciones de las instancias de prueba deben coincidir con las suministradas, con un matiz: dado que trabajar con flotantes supone errores de redondeo, consideraremos que dos longitudes de ciclo hamiltoniano son iguales si difieren menos de 0.0001.

4. Probar el programa en lynx.uji.es

Nota: En el enunciado del entregable 1 os indicamos qué hay que hacer para poder probar vuestros programas en lynx.uji.es. Puedes consultarlo allí si lo necesitas.

El programa `entregable2.py` se probará en `lynx.uji.es` desde la línea de ordenes. Por ejemplo, para la instancia `instancia1.txt`:

```
time python3 entregable2.py < instancia1.txt > salida_instancia1.txt
```

Como resultado de la ejecución de la línea anterior, se guardará la salida en el fichero `salida_instancia1.txt` (que se utilizará en la corrección del entregable) y, además, se mostrará por pantalla las siguientes tres líneas, con información del tiempo de ejecución (las muestra la orden `time` que hemos añadido al lanzar la ejecución):

```
real    0m1.011s
user    0m1.001s
sys     0m0.003s
```

La línea `user` muestra el tiempo de CPU utilizado por el programa, que deberá ser menor de **cuatro segundos** para cada una de las instancias.

5. Visualizador de los ciclos hamiltonianos de las soluciones

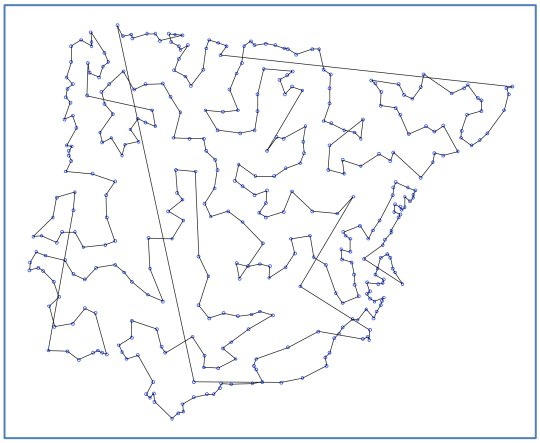
Para facilitar la depuración, junto al enunciado también se incluye el programa `e2_viewer.py` que puede mostrar cualquiera de los dos ciclos hamiltonianos de un fichero de solución. Sus parámetros son:

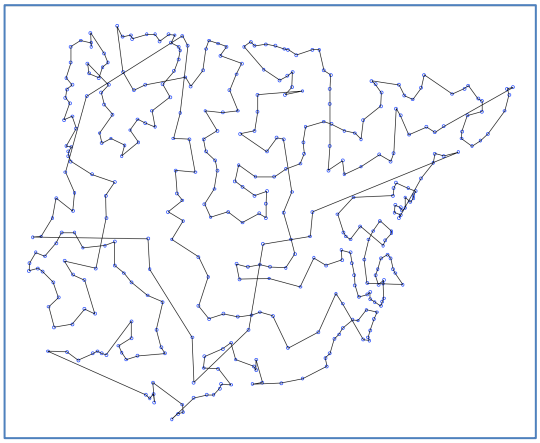
```
e2_viewer.py <-1|-2> <instance.ins> <instance.sol>
```

Por ejemplo, para mostrar el ciclo hamiltoniano que el algoritmo 2 obtiene para la instancia `iberia.ins` ejecutaríamos la orden:

```
python3 e2_viewer.py -2 iberia.ins iberia.sol
```

La instancia `iberia.ins` contiene las coordenadas 2D (puntos) de las 378 principales ciudades de la península ibérica. La figura de la izquierda muestra el ciclo hamiltoniano del algoritmo 1, basado en Kruskal, y que mide de 12,523.6 km. La figura de la derecha muestra el ciclo hamiltoniano del algoritmo 2, basado en Prim, y que mide 12,530.3 km:





Fecha de entrega en el aula virtual: lunes 23 de noviembre de 2019.

No os quedéis colgados. Recordad que hay tutorías.