Practica 3 : Algoritmos Voraces

El Problema del Viajante de comercio

Algoritmica - 2015/2016

**Alejandro Campoy Nieves**

**David Criado Ramón**

**Nour Eddine El Alaoui**

**Luis Gallego Quero**

**Índice :**

**1 - Descripción del Problema**

**2 - Heurística 1 : El vecino más cercano**

**2.1 - El main y el código usado por los tres algoritmos**

**2.2 - Código del Algoritmo**

**2.3 - Ejemplo de demostración**

**3 - Heurística 2 : Algoritmo de inserción**

**3.1 - Código del Algoritmo**

**3.2 - Ejemplo de demostración**

**4 - Heurística 3 : Algoritmo basado en aristas**

**4.1 - Código**

**4.2 - Ejemplo de demostración**

**1 - Descripción del problema :**

Básicamente el problema trata de un conjunto de ciudades y una matriz con las distancias entre todas ellas, tenemos un viajante que debe recorrer todas las ciudades solo una vez, y regresar al punto de partida, tal que la distancia recorrida sea la mínima.

Más formalmente, dado un grafo G, conexo y ponderado, se trata de hallar el ciclo hamiltoniano de mínimo peso de ese grafo.

Para resolver este problema tenemos que usar tres algoritmos distintos :

1. - Algoritmo del vecino más cercano.
2. - Algoritmo de inserción.
3. - Algoritmo basado en aristas.

**2 - Heurística 1 : El vecino más cercano**

* **Enfoque Greedy**
* Conjunto de candidatos: Ciudades a visitar
* Conjunto de seleccionados: Ciudades visitadas
* Función solución: Si hemos recorrido todas las ciudades
* Función de factibilidad: Que la ciudad no haya sido visitada
* Función de selección: Seleccionamos la ciudad más cercana
* Función objetivo: Recorrer todas las ciudades y volver a la ciudad inicial

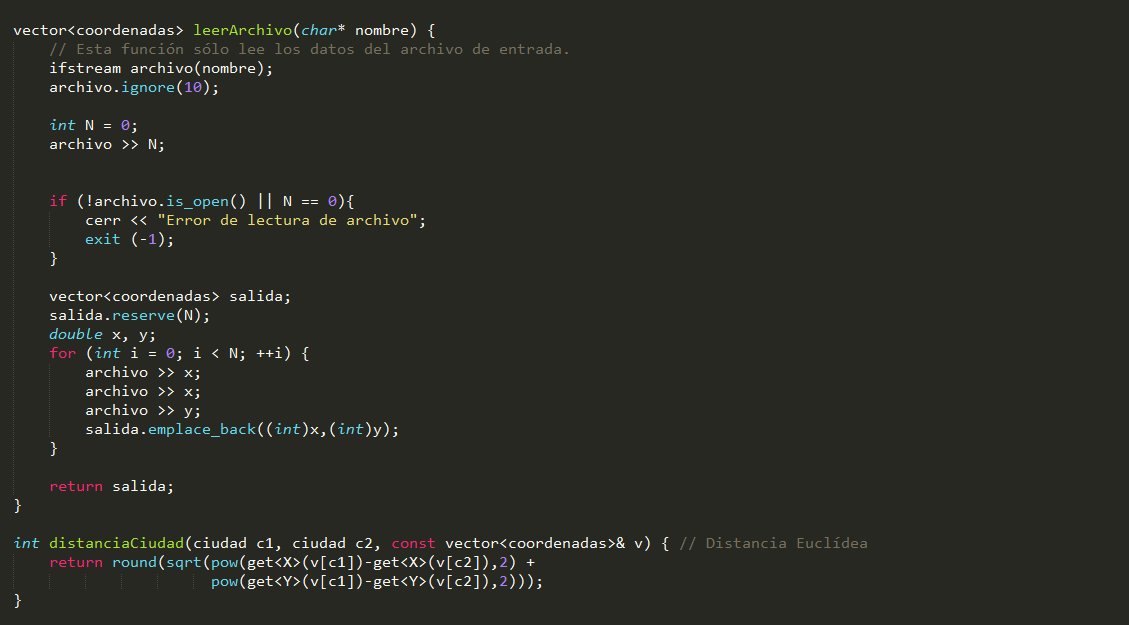
Primero creamos una matriz con la distancias entre las ciudades, utilizando el algoritmo de Euclides.

Empezamos seleccionando la cuidad inicial, y vamos mirando por orden creciente de distancia desde la ciudad actual, se inserta la ciudad actual y la longitud del trayecto,y se vuelve a hacer el mismo procedimiento, haciendo uso de un filtro para asegurar que las cuidades introducidas no están repetidas.

Una vez hecho esto, insertamos de nuevo la primera ciudad, que es la única que se puede repetir.

**2.1 El main y el código usado por los tres algoritmos**

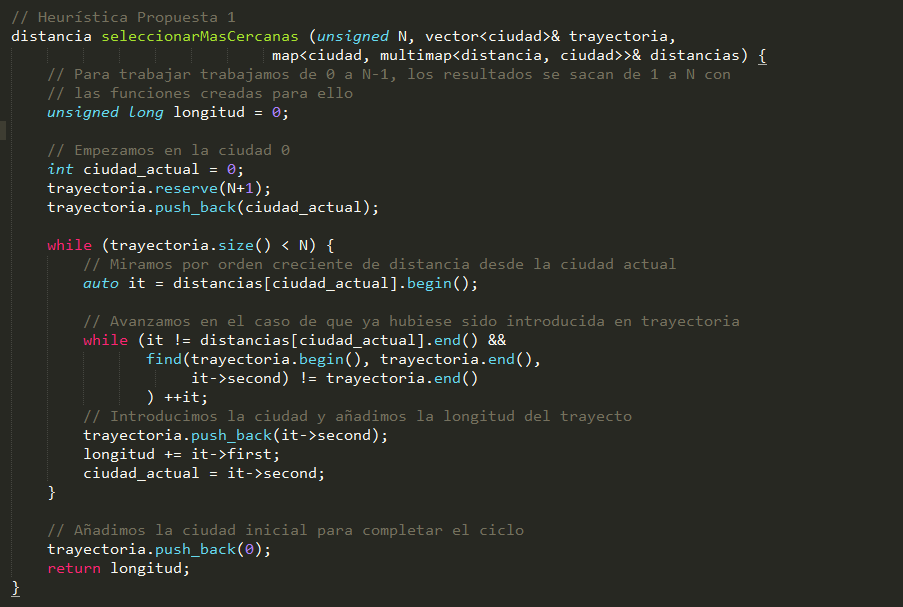
* *Para las tres estrategias, hemos hecho uso de :*
* Una función *LeerArchivo*, para leer los datos del archivo de entrada
* otra función *DistanciaCuidad*, para calcular la distancia Euclídea



* y por último la función *MostrarRecorrido* para imprimir el recorrido de cada estrategia :

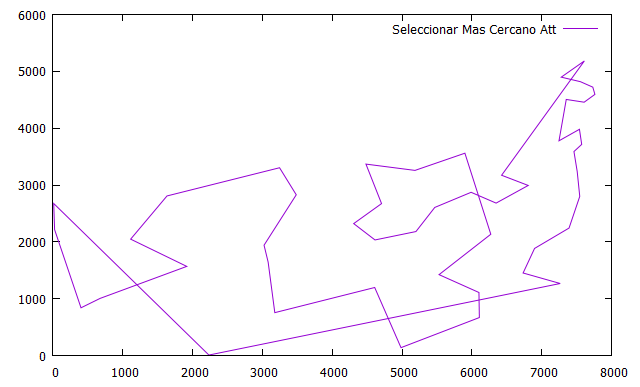


**2.2 Código del Algoritmo :**

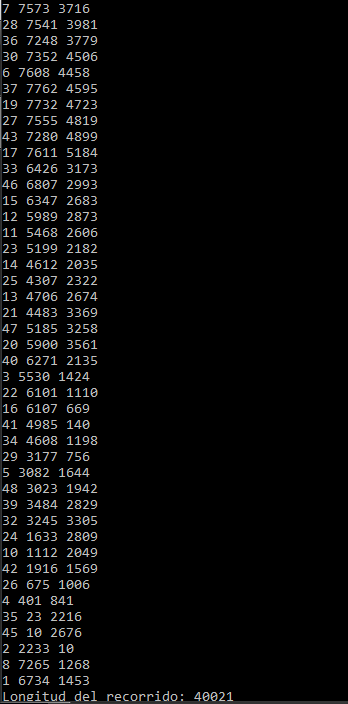


**2.3 - Ejemplos de demostración :**

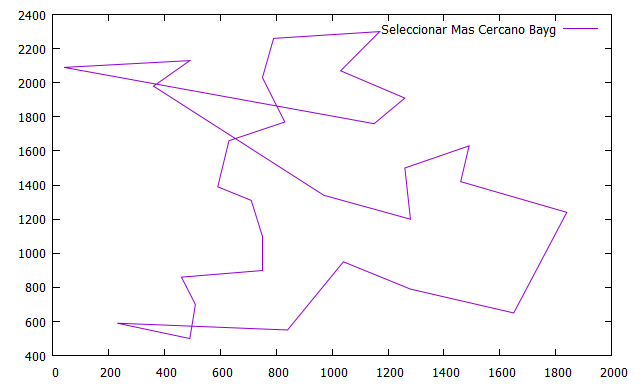
* ***Ejemplo : Att48***



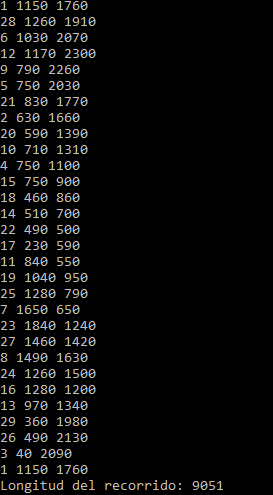
***Figura 1 : demostración para la ciudad Att48***



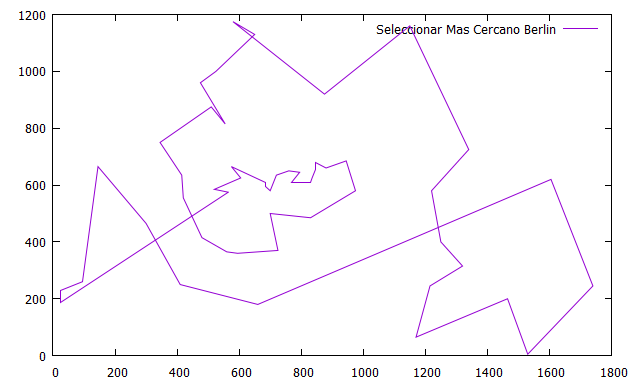
* ***Ejemplo : Bayg29***



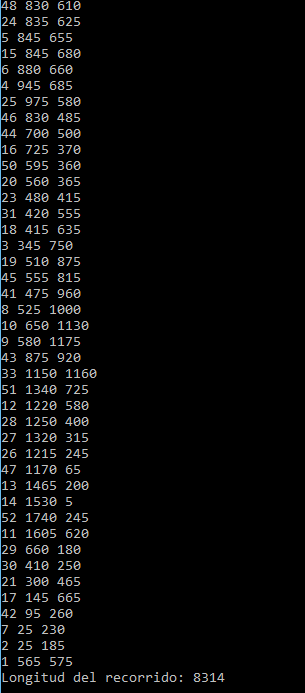
***Figura 2 : demostración para la ciudad Bayg29***



* ***Ejemplo : Berlin52 :***



***Figura 3 : demostración para la ciudad Berlin52***



**3. Heurística 2 : Algoritmo de inserción**

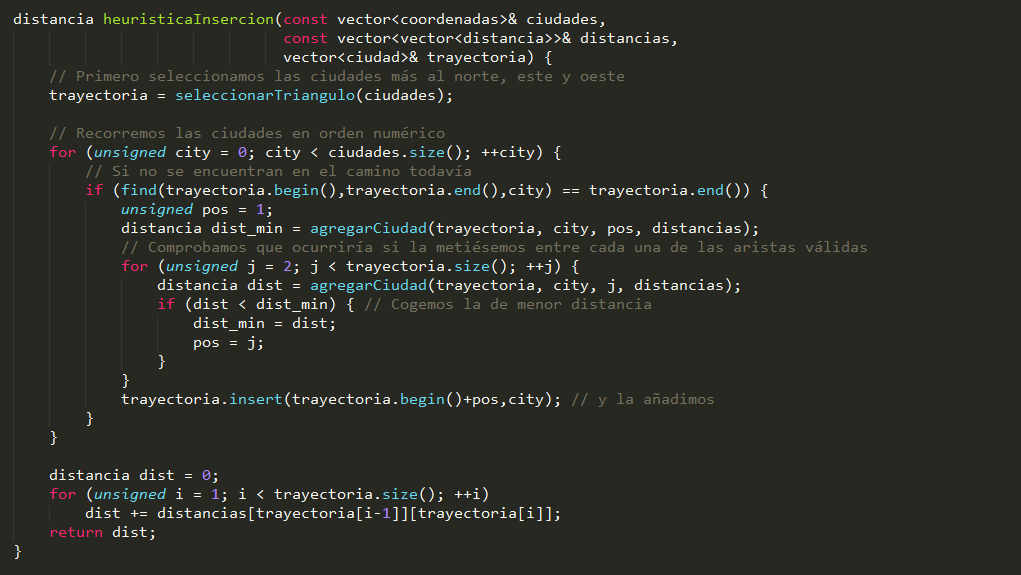
* **Enfoque Greedy :**
* Conjunto de candidatos: Ciudades a visitar
* Conjunto de seleccionados: Ciudades visitadas
* Función solución: Si hemos recorrido todas las ciudades
* Función de factibilidad: Que la ciudad no haya sido visitada
* Función de selección: Seleccionamos de entre todas las ciudades cual tendría mejor distancia,comprobandolo en todas las ubicaciones posibles.
* Función objetivo: Recorrer todas las ciudades y volver a la primera

La idea de esa heurística es crear un recorrido inicial, por ejemplo nos fijamos en la tres ciudades que forman el triángulo más grande (la ciudad que está más al norte, la más al este y la más al oeste) y a partir de este recorrido, vamos insertando las ciudades que faltan.

Empezamos recorriendo el vector de ciudades no seleccionadas e insertando cada una de ellas en cada posible posición, luego cogemos la de menor distancia y la añadimos.

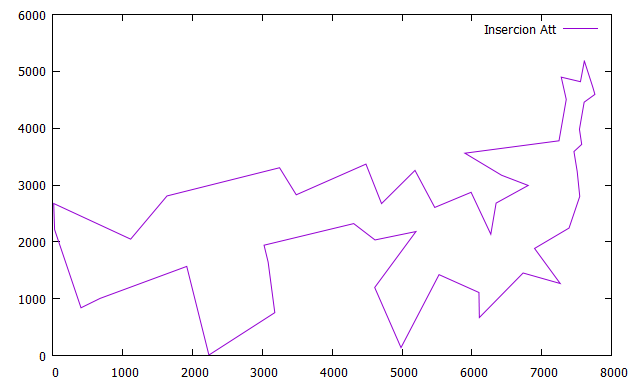
**3.1 Código del Algoritmo :**



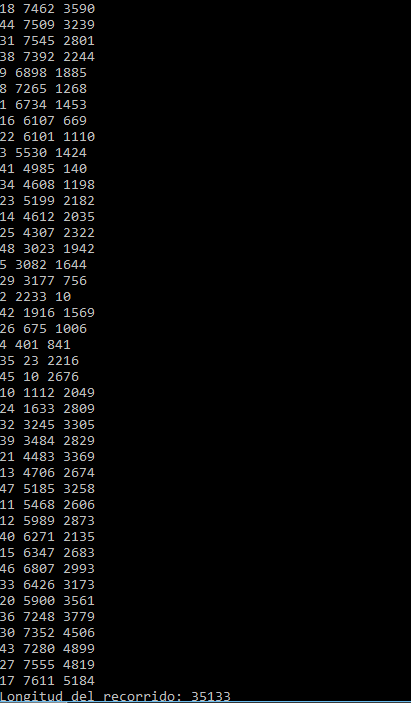


**3.2 - Ejemplos de demostración :**

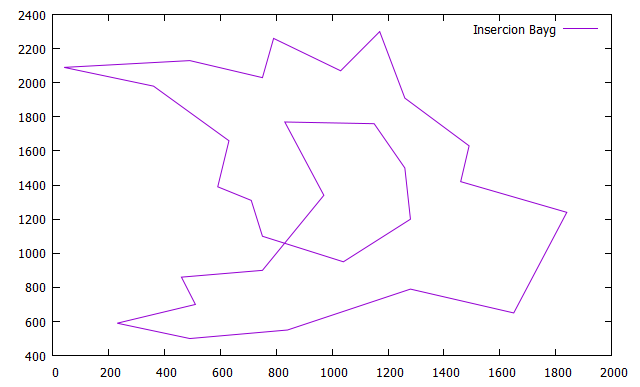
* ***Ejemplo : Att48***



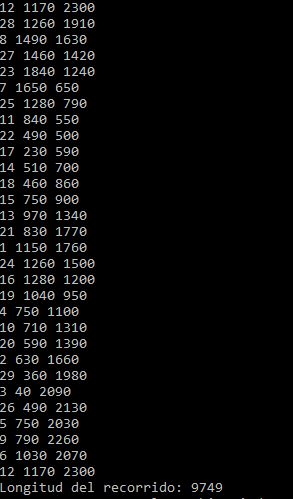
***Figura 4 : demostración para la ciudad Att48***



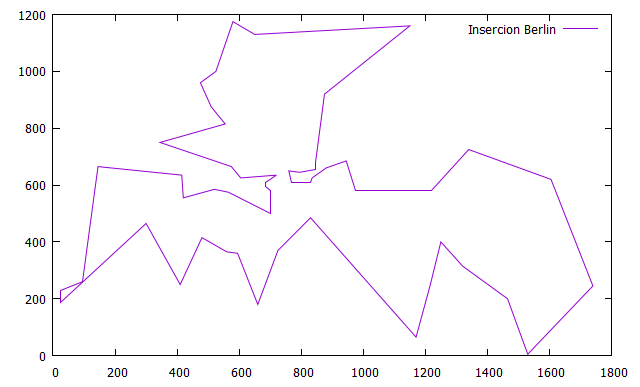
* ***Ejemplo : Bayg29***



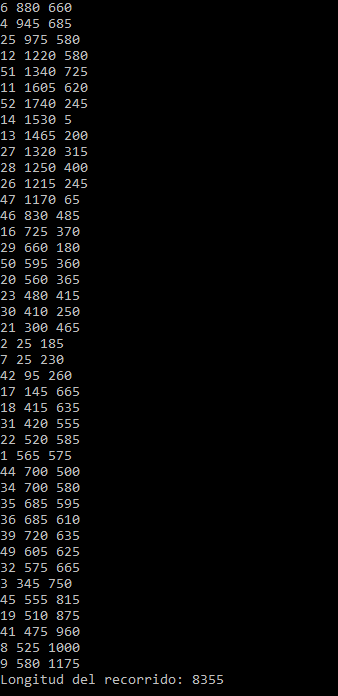
***Figura 5 : demostración para la ciudad Bayg29***



* ***Ejemplo : Berlin52***



***Figura 6 : demostración para la ciudad Berlin52***



**4. Heurística 3 : Algoritmo basado en aristas**

* **Enfoque Greedy :**
* Conjunto de candidatos: Ciudades a visitar
* Conjunto de seleccionados: Ciudades visitadas
* Función solución: Si hemos recorrido todas las ciudades
* Función de factibilidad: Que la ciudad no tenga más de 2 arista y no se cree un ciclo al añadirla
* Función de selección: Seleccionamos las ciudades con menor distancia entre ellas.
* Función objetivo: Recorrer todas las ciudades y volver a la primera.

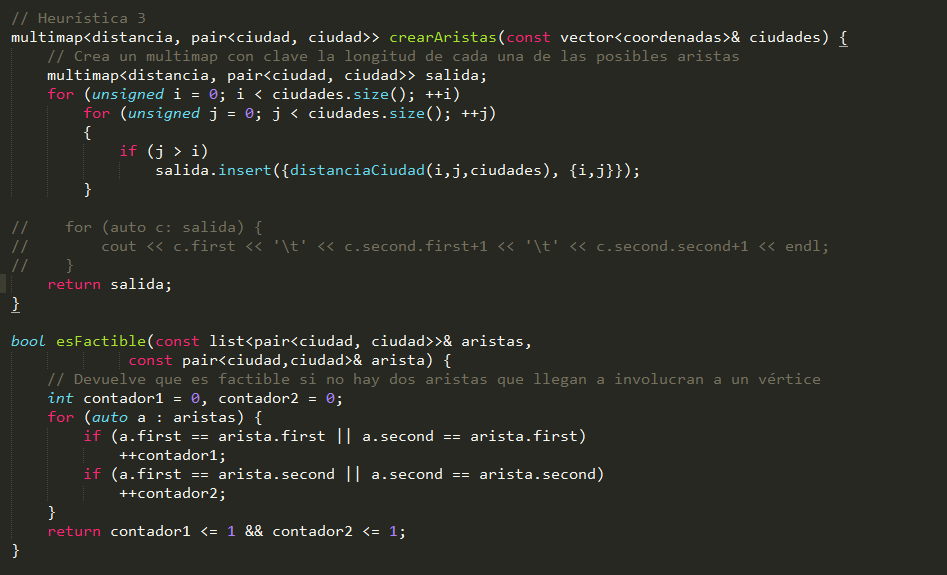
Para este algoritmo hemos creado un multimap ( multimap<distancia, pair<ciudad, ciudad>> crearAristas ).formado por la distancia entre dos ciudades cada entrada.

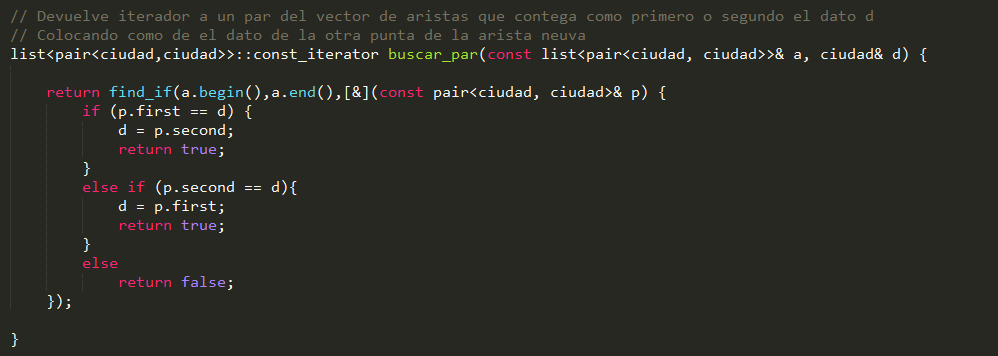
Vamos seleccionando las distancias menores entre dos ciudades y añadiendo estas al conjunto de seleccionados.

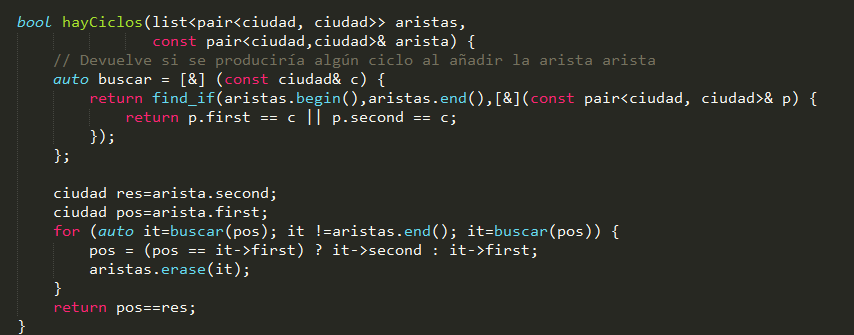
Para poder insertar una nueva ciudad, esta no puede tener más de 2 aristas ni crear un ciclo.

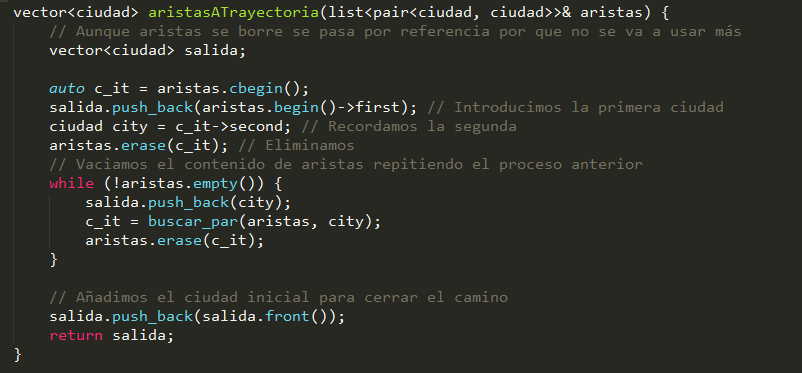
Cómo el camino resultante de este algoritmo no es cerrado, tenemos que recorrer el camino y añadir la distancia entre las 2 ciudades que únicamente tienen una arista.

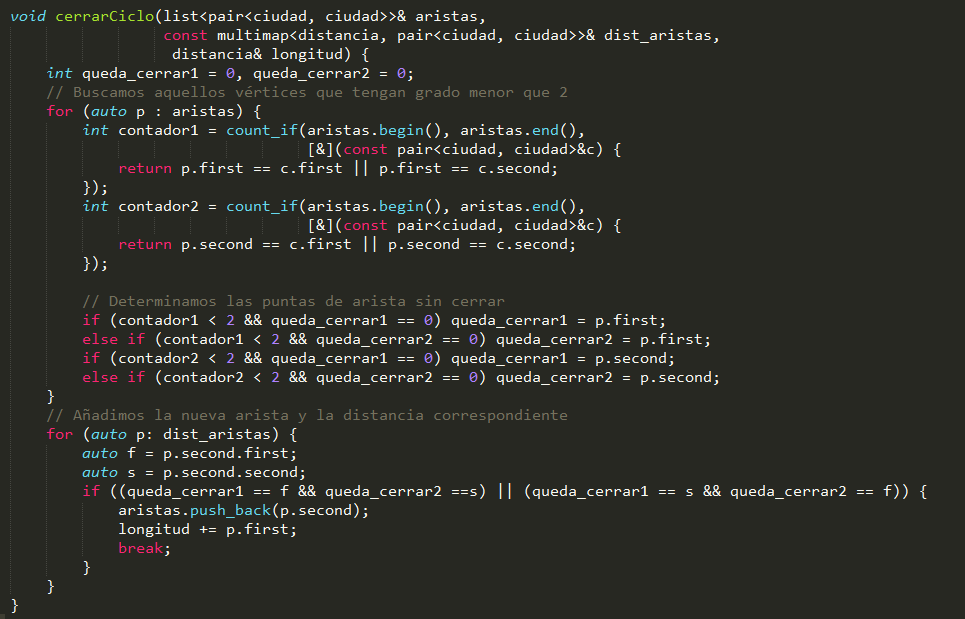
**4.1 Código del Algoritmo :**

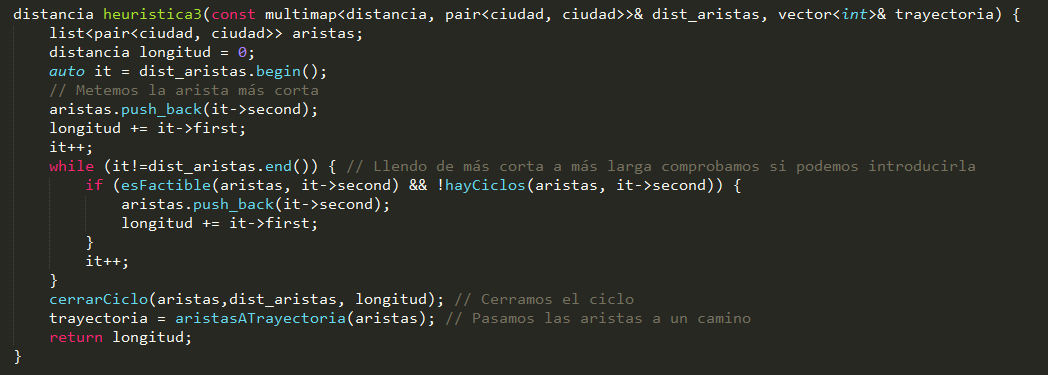






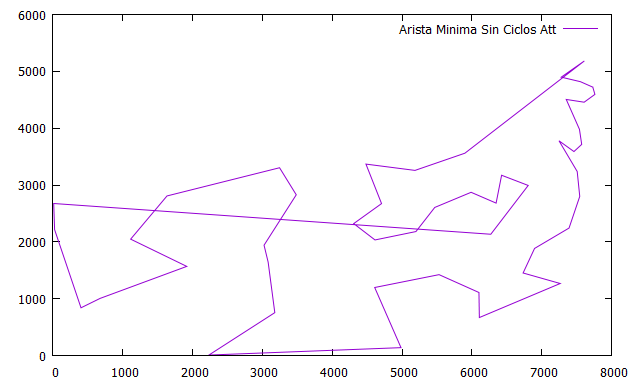




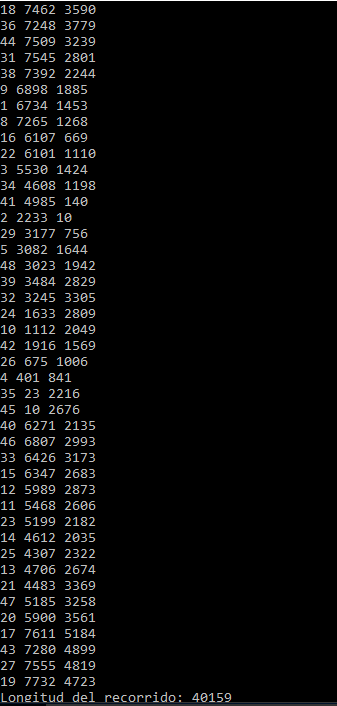


**4.2 - Ejemplos de demostración :**

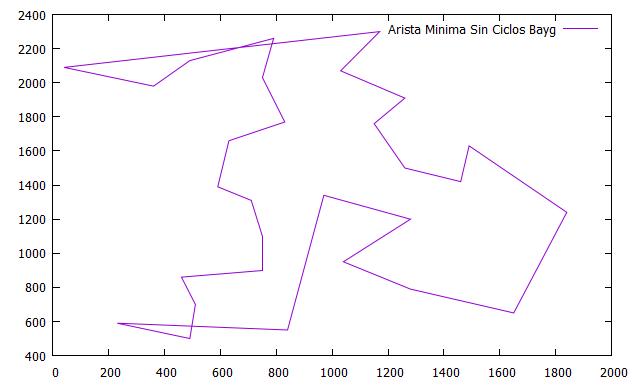
* ***Ejemplo : Att48***



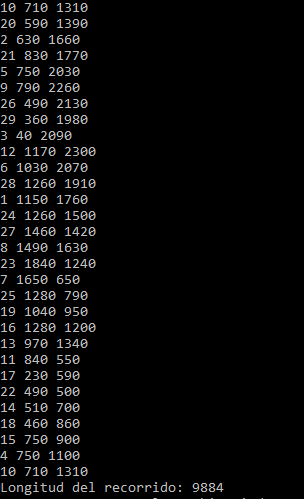
***Figura 7 : demostración para la ciudad Att48***



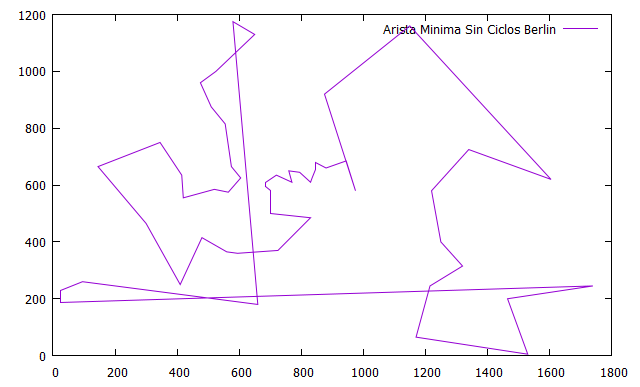
* ***Ejemplo : Bayg29***



***Figura 8 : demostración para la ciudad Bayg29***



* ***Ejemplo : Berlin52***



***Figura 9 : demostración para la ciudad Berlin52***

