TALF II - Práctica 1
Introducción a la tratabilidad

7 de octubre y 14 de octubre - Fecha de entrega 21 de octubre

Problema (simétrico) del viajante (STSP – Symmetric Traveling Salesman Problem)

DESCRIPCIÓN:

Dado un conjunto de *n* ciudades y las distancias entre ellas, calcular la **ruta CIRCULAR MÍNIMA que visite TODAS las ciudades SIN REPETIR NINGUNA** (salvo la ciudad de origen que será también la de destino). Cualquier método que se utilice debe garantizar que la ruta resultado es la de menor coste de todas las posibles.

Al tratarse de la variante simétrica del problema del viajante, la distancia de la ciudad *i* a la ciudad *j* coincide con la distancia de la ciudad *j* a la ciudad *i*. Para calcular la ruta mínima la solución básica consiste en realizar una búsqueda exhaustiva que mida las distancias de todas las rutas posibles, con todos los nodos posibles como nodo inicial, y compare sus distancias para localizar la ruta mínima. Sobre este algoritmo inicial se pueden realizar mejoras que poden el conjunto de rutas a explorar. Por ejemplo:

- si el coste de la ruta *R* que se está calculando ya supera al coste mínimo *C* encontrado hasta el momento no es necesario seguir explorando *R* ya que cualquier ruta que tenga a *R* como sub-ruta tendrá un coste superior a *C*.
- si sabemos que la ruta *c1 c2 c3 ... c(n-2) c(n-1) cn* no es la ruta mínima, no es necesario calcular el coste de la ruta *cn c(n-1) c(n-2) ... c3 c2 c1* ya que se trata de la misma ruta.

ESPECIFICACIONES:

Entrada:

Fichero de datos:

- La lista de ciudades y sus costes viene definida en un fichero. Este fichero tiene una parte de **especificación** (con información sobre el propio fichero y el formato de sus contenidos) y otra de **datos** (los contenidos en sí). Todas las líneas de la parte de especificación tienen el formato <*keyword*> *:* <*value*>. Las más significativas son las siguientes:

```
NAME : <nombre>
COMMENT : <comentario>
DIMENSION : n
EDGE_WEIGHT_TYPE : <tipo de cálculo de pesos>
```

Donde:

NAME: Nombre del fichero de ciudades.
COMMENT: Algún comentario acerca del fichero.
DIMENSION n es el número de ciudades de que consta el mapa .

EDGE_WEIGHT_TYPE: define el tipo de cálculo necesario para calcular la distancia entre ciudades. Todos los tipos de cálculo están definidos en el fichero DOC.pdf. Los ejemplos que usaremos en esta práctica son los siguientes:

2.1 Euclidean distance (L_2-metric)

For edge weight type `EUC_2D` and `EUC_3D`, floating point coordinates must be specified for each node. Let `x[i]`, `y[i]`, and `z[i]` be the coordinates of node i.
In the 2-dimensional case the distance between two points i and j is computed as follows:

```
xd = x[i] - x[j];
yd = y[i] - y[j];
dij = nint( sqrt( xd*xd + yd*yd) );
```

2.5 Pseudo-Euclidean distance

The edge weight type `ATT` corresponds to a special "pseudo-Euclidean" distance function. Let `x[i]` and `y[i]` be the coordinates of node i. The distance between two points i and j is computed as follows:

```
xd = x[i] - x[j];
yd = y[i] - y[j];
rij = sqrt( (xd*xd + yd*yd) / 10.0 );
tij = nint( rij );
if (tij<rij) dij = tij + 1;
else dij = tij;
```

***NOTA**: nint() redondea al entero más cercano, al entero más alto en caso de equidistancia.*

La parte de datos, definida al final del fichero, comienza con una palabra clave (*NODE_COORD_SECTION* en nuestro caso) seguida, a partir de la siguiente línea, por los datos en cuestión para terminar en el marcador de final de fichero *EOF*:

```
NODE_COORD_SECTION
1 x1 y1
2 x2 y2
3 x3 y3
...
n xn yn
EOF
```

Donde:

NODE_COORD_SECTION: marca el comienzo de la sección en dónde se especifican las coordenadas de cada nodo.

xi e ***yi*** son las coordenadas de la ciudad ***i***.

EOF: marca el final del fichero.

***NOTA**: Además de estos pares <keyword> : <value> el fichero puede contener otros que deben ser convenientemente gestionados e ignorados Más detalles en el apartado 1 del fichero* DOC.pdf

Parámetros

El programa deberá aceptar dos parámetros

```
programa <número_de_ciudades> <archivo_de_mapa>
```

Donde:

número_de_ciudades es el número de ciudades sobre las que se quiere calcular la ruta.
archivo_de_mapa es el archivo que contiene la lista de ciudades y sus costes/coordenadas

NOTA: El número de ciudades del problema puede ser menor que el número de ciudades del fichero. En tal caso, se calcula la ruta para las <número_de_ciudades> primeras ciudades del fichero.

Salida:

El programa deberá escribir en un fichero y en salida estándar:

```
NAME : <nombre_del_fichero>
NUM_CITIES : m
DIMENSION : n
TOTAL_COST : n
TOUR_SECTION
c1 c2 c3… cn
EOF
```

Donde:

NAME: nombre del fichero de ciudades
NUM_CITIES: número de sobre las que se ha calculado la ruta
DIMENSION: número de ciudades del fichero de entrada
TOTAL_COST: coste total de la ruta
TOUR_SECTION: comienzo de la sección en la que se especifica la ruta
ci: índice de la ciudad i-ésima de la ruta
EOF: final de la ruta

NOTA: no hay que repetir la ciudad origen como ciudad destino en la TOUR_SECTION aunque sí que se debe computar el coste de vuelta en el coste total

SE PIDE:

Apartado 1:

- Desarrollar un programa que resuelva el problema para las 7 ciudades del archivo p1ap1.dat
- Realizar una breve memoria (máximo dos caras) en la que se explique el algoritmo utilizado, las mejoras aplicadas y, brevemente, los módulos de la práctica.
- Desarrollar un validador que, dado un fichero solución y un fichero problema como parámetros, compruebe que la suma de los costes de las ciudades de la ruta (incluyendo la vuelta de la ciudad final a la inicial) se corresponde con el valor de *TOTAL_COST*, que no hay ninguna ciudad repetida y que el número de ciudades de la ruta se corresponde con el valor de *NUM_CITIES*

Apartado 2:

- Realizar una gráfica que compare los costes computacionales medios de cada algoritmo (i.e. "sin podas" "con una poda" o "con varias) en función del número de ciudades del problema y **comentar brevemente qué implica, a qué se debe y cómo se puede mejorar**. Para ello se usará el fichero *p1ap2.dat* y se medirán los costes para 4, 5, 6, 7, 8, 9, 10... ciudades.
- Optativo: calcular el coste medio, caso peor y caso mejor del algoritmo implementado

IMPORTANTE: Todos los programas deben haber sido probados con **Valgrind** para obtener un código libre de fallos de memoria.

ENTREGA:

Al **comienzo de la clase** del 21 de octubre se entregará:

- Un archivo de nombre **p1pY.tgz (**donde X es el número de grupo e Y el de la pareja) que contendrá:
  - **Código** C de la práctica
  - Archivo **Léeme.txt** con instrucciones de compilación, breve descripción de lo que hace el programa, entrada, salida...
  - **Memoria** doc o pdf
  - **Makefile**
- La memoria impresa

Aparte de la utilidad que tendrá en la segunda práctica, se valorará la limpieza, correcta modularización y comentado del código.