UNIVERSIDAD DE GRANADA

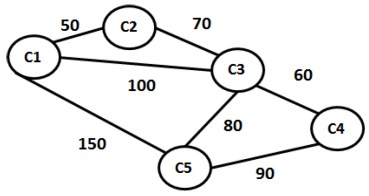
Metodología de la Programación

Grado de Ingeniería Informática

Práctica 4. Clases y sobrecarga de operadores

DECSAI

Se desea construir la clase `RedCiudades` para almacenar datos de un conjunto de ciudades, y las distancias de los caminos directos que las conectan. Si entre dos ciudades no existe un camino directo, se almacenará un cero. Se supone que la distancia de una ciudad consigo misma será cero y que las distancias son simétricas. Un ejemplo con 5 ciudades es:



	C1	C2	C3	C4	C5
C1	0	50	100	0	150
C2	50	0	70	0	0
C3	100	70	0	60	80
C4	0	0	60	0	90
C5	150	0	80	90	0

Se propone la siguiente representación para la clase:

```
class RedCiudades {
   private:
      int num_ciudades; // Numero de ciudades
      InfoCiudad * info; // info[i]: info de la ciudad i
      double ** distancia;// distancia entre i y j
   public:
       // ... interfaz publica de la clase
};

struct InfoCiudad {
      char * nombre; // Nombre
      int poblacion; // Numero habitantes
};
```

1. Implementar los siguientes métodos básicos para la clase `RedCiudades`:
   - Constructor por defecto y destructor. El constructor por defecto crea una red vacía (escriba también el método `EstaVacia` que indique si una red está vacía).
   - Constructor de copia y operador de asignación.
   - `NumCiudades`: devuelve el número de ciudades.
   - `Distancia`: devuelve la distancia entre dos ciudades.
   - `NombreCiudad`: devuelve el nombre de una ciudad (en realidad, la dirección de inicio).
   - `PoblacionCiudad`: devuelve el número de habitantes de una ciudad.
2. Implementar los siguientes operadores:
   - Operador de salida << para poder insertar en un flujo de salida el contenido de una red en formato texto. En concreto, deberá aparecer: un número entero que indica el número de ciudades y un salto de línea, una serie de líneas (tantas como ciudades) que contienen el índice de la ciudad (un número entero), su nombre (una serie de caracteres sin espacios intermedios) y su población (un número entero), una serie de líneas (tantas como conexiones entre ciudades) que contienen dos números enteros (números de las ciudades conectadas) y un número real (distancia entre ellas). Nota: Cada conexión entre ciudades se escribe una sóla vez (no importa cuál es la ciudad de origen o destino).

- Operador de entrada >> para poder obtener desde un flujo de entrada el contenido de un objeto. Se asume el mismo formato indicado en el item anterior.

3. Implementar los siguientes métodos de cálculo:

   - Un método `CiudadMejorConectada` que permita obtener la ciudad (su índice) con mayor número de conexiones directas. En el ejemplo, la ciudad mejor conectada es la ciudad $C3$, con 4 conexiones.
   - Un método `MejorEscalaEntre` para que, dadas dos ciudades i y j no conectadas directamente, devuelva aquella ciudad intermedia z que permita hacer el trayecto entre i y j de la forma más económica posible. Es decir, se trata de encontrar una ciudad z tal que $d(i, z) + d(z, j)$ sea mínima ($d(a, b)$ es la distancia entre las ciudades a y b). El método devuelve -1 si no existe dicha ciudad intermedia. Por ejemplo, si se desea viajar desde la ciudad $C2$ a la $C5$, hacerlo a través de la ciudad $C1$ tiene un coste de $50 + 150 = 200$ mientras que si se hace a través de la ciudad $C3$, el costo de $70 + 80 = 150$.

4. Escribe un programa completo que reciba desde consola la descripción de una red (como se indica en 2) y calcule, para todas las parejas de ciudades que no estén directamente conectadas, cuál es la mejor escala con una sola ciudad intermedia (su índice). Si no la tuviera, deberá indicarlo.

5. Divide el código en varios ficheros y comenta, en el formato de Doxygen, los ficheros de cabecera.