Estructuras de Datos y de la Información

Proyecto de Programación

1ª Parte

Por:
Profesores de EDI

Índice general

1. Introducción

Este documento describe los requerimientos y funcionalidad básica para el desarrollo de la primera parte del proyecto de programación de la asignatura *Estructuras de Datos y de la Información* del curso 2018-19.

En este proyecto vamos a colaborar con el Ayuntamiento de Cáceres en el desarrollo de una aplicación que procese los datos disponibles en el portal *Open Data Cáceres* [1] relativos a *barrios* y *calles* de la ciudad. La aplicación consistirá en una serie de algoritmos que extraerán información a partir de los datos previamente representados en nuestra aplicación.

Nuestro objetivo será desarrollar una aplicación con las siguientes características generales:

- *Eficiencia*: debido a la gran cantidad de datos y al esperado crecimiento de los mismos en el futuro, debemos encontrar la forma de almacenar y procesar los datos de la forma más eficiente posible. Además, deberemos realizar un análisis de la eficiencia y rendimiento de nuestra aplicación para cumplir este requisito.
- *Extensibilidad*: el diseño de la aplicación en cuanto a entidades y estructuras de datos debe ser extensible, para facilitar el desarrollo de futuras versiones de la aplicación y ampliar su funcionalidad.
- *Modularidad*: la organización de las entidades que forman nuestra aplicación debe ser modular, de forma que las modificaciones y ampliaciones de la misma se puedan realizar de la forma más sencilla posible, afectando a la menor cantidad de código.

Para cumplir con los requisitos anteriores se seguirá una metodología de desarrollo basada en el paradigma de *Programación Orientada a Objetos* y se utilizará el lenguaje de programación *C++*.

Este documento se organiza de la siguiente forma. En la sección 2 se discuten de forma general los pasos que debemos seguir para el desarrollo de la aplicación y se ofrecen detalles de implementación, incluida la interfaz para el contenedor de datos que utilizaremos, que denominaremos *Conjunto*. En la sección 3 se enumeran los algoritmos básicos que debemos implementar y en la sección 4 se discuten los mecanismos de análisis y tests que debemos aplicar sobre nuestra aplicación para evaluarla, y asegurar que cumple los requisitos, y lo hace de forma eficiente. Finalmente, la sección 5 establece los criterios de evaluación de la calidad del desarrollo y su calificación, así como las fechas de entrega.

2. Metodología e Implementación

En esta sección se describe la funcionalidad que implementará nuestra aplicación, y se establecen una serie de pasos generales que pueden servir de guía para el desarrollo de la misma.

En el desarrollo, se partirá de una plantilla de código, previamente disponible, a la que el programador debe ajustarse. Esta estructura estará formada por una clase principal en la que podemos incluir los algoritmos a implementar como nuevas operaciones.

2.1. Datos

Los datos ofrecidos por el Ayuntamiento han sido pre-tratados y están a disposición del programador en el Campus Virtual de la asignatura[1].

Los archivos que contienen los datos son de tipo texto, y se enumeran en el Cuadro 1. Los campos de cada archivo son autoexplicativos, es labor del programador extraer el tipo de cada dato para representarlo adecuadamente.

Nombre	**Descripción**
`Barrio.csv`	Archivo que contiene los *barrios* de Cáceres.
`Via.csv`	Archivo que contiene las *calles* de la ciudad.

Cuadro 1: Ficheros de datos para la aplicación.

Algunas de las calles en el archivo aparecen duplicadas, debido a que pertenecen a distintos barrios. Este es un detalle importante a la hora de elegir una clave adecuada para almacenar las calles *sin repeticiones* en nuestras estructuras de datos (véase sección 2.3).

2.2. Implementación de las clases *base*

En la plantilla proporcionada como esqueleto de la aplicación, una vez incorporada a un proyecto en el entorno de programación *Eclipse*, incluiremos dos clases que

[1]Se deben utilizar los datos en los ficheros proporcionados sin modificaciones. Cualquier modificación supondrá que los resultados de los algoritmos no serán los esperados, y como consecuencia se podrán considerar incorrectos.

contendrán la descripción de cada uno de los tipos de datos en nuestra aplicación: `Barrio` y `Via`. La elección de los atributos de cada clase y su representación (tipos de datos) es responsabilidad del programador, y se deben ajustar a los datos en los archivos. Las operaciones a implementar, inicialmente en cada una de las clases base, son (1) *setters/getters* para cada atributo, (2) constructores, (3) destructor y una operación para (4) mostrar la información en pantalla. A estas operaciones básicas se podrán añadir otras si es necesario.

2.3. Contenedores de datos

La estructura de datos básica que vamos a utilizar en nuestra aplicación (sección 1) es el *Conjunto*, entendido como *una colección ordenada de datos del mismo tipo y sin duplicados*. La implementación de dicho contenedor le corresponde al programador, sin embargo, la interfaz se describe en el Cuadro 2.

La clase `Conjunto` se utilizará para almacenar todos los datos en la aplicación. Si es necesario, se puede replicar para contener diferentes tipos de datos (por ejemplo, un conjunto de barrios y un conjunto de calles), pero manteniendo la interfaz. Además, a pesar de que la interfaz ha sido descrita, las *Precondiciones* y *Postcondiciones* para cada una de las operaciones se deben establecer por el programador[2].

La *clave* elegida para identificar los elementos de un conjunto debe ser única, para evitar duplicados[3]. Hay que tener en cuenta que la clave sirve para no duplicar elementos, pero el criterio de ordenación puede ser por la propia clave u otro diferente (por otro atributo).

Las operaciones que se consideren necesarias para crear y destruir objetos de la clase `Conjunto` se deben añadir convenientemente.

2.4. Relaciones entre clases

Todos los datos se almacenarán en contenedores de tipo `Conjunto`. Sin embargo, la decisión sobre la forma en la que se relacionan los datos de las diferentes clases corresponde al programador. Como ejemplo, se ofrecen dos alternativas a la implementación de *barrios* y *calles*:

[2]Se recuerda que las pre/post-condiciones son expresiones booleanas, y no texto descriptivo, y que una vez establecidas deben ser respetadas, es decir, el programador que utiliza la interfaz debe cumplir las precondiciones, y el programador que implementa la interfaz debe asegurar las postcondiciones.

[3]Puede ser útil en algún caso formar la clave como la unión de dos atributos de una clase, por ejemplo, el código de calle y barrio en la clase `Via`.

Valor Retorno	**Nombre**	**Parámetros**	**Descripción**
-	insertar	↓ elemento	Inserta un elemento en orden de clave.
-	borrar	↓ clave	Elimina un elemento de la estructura de datos (no de memoria).
-	obtener	↓ posición, ↑ elemento	Obtiene un elemento en una posición del conjunto.
-	obtener	↓ clave, ↑ elemento	Obtiene un elemento dada su clave.
booleano	existe	↓ clave	Devuelve *True* si el elemento existe, *False* en otro caso.
entero	cuantos	-	Devuelve el número de elementos en el conjunto.
booleano	vacío	-	Devuelve *True* si no hay elementos en el conjunto, *False* en otro caso.

Cuadro 2: Interfaz para la clase `Conjunto`.

1. Tener un único conjunto con todas las calles y un único conjunto con todos los barrios.
2. Tener un único conjunto de barrios, y cada barrio tendrá un conjunto únicamente con las calles que pertenezcan a ese barrio.

Las implicaciones de esta decisión de diseño son muchas e importantes. Entre ellas: (1) la forma en que se leen y cargan los datos desde los ficheros, (2) la forma en que se implementan las operaciones en cada clase y (3) la forma y complejidad de los algoritmos implementados. **El programador debe escoger la estructura en función de los criterios de calidad establecidos en la sección 1**.

El programador tiene libertad para implementar contenedores *genéricos* de datos de tipo *Conjunto*. Esta facilidad de los lenguajes de programación permite la implementación de las operaciones de la clase de forma independiente del tipo de datos que van a almacenar, lo que es muy útil para no duplicar implementaciones. El mecanismo utilizado en C++ para construir clases genéricas son los *Templates* [2].

3. Algoritmos

Los algoritmos a implementar en nuestra aplicación son:

1. Escribir una operación que genere un fichero con las calles que pertenezcan a un barrio dado. El barrio se especificará indicando su nombre.
2. Escribir una operación que devuelva los barrios con un mayor y un menor número de calles.
3. Mostrar en pantalla todos los barrios, junto con sus calles, que empiecen (ambos) por una subcadena.
4. Obtener la *Avenida* de mayor longitud de Cáceres, junto con su distrito. Si la avenida pertenece a varios distritos, el programador puede decidir cuál de ellos devolver.

Todos los algoritmos deben ser realizados sobre las estructuras de datos en memoria, es decir, no están permitidas las implementaciones de los algoritmos directamente sobre los ficheros de datos.

La interfaz de las operaciones debe ser diseñada y documentada correctamente conforme a la descripción[4].

4. Análisis

La labor de un desarrollador de aplicaciones no es solamente hacer que su código funcione correctamente, sino asegurar que cumple con unos criterios de calidad. Por tanto, el programador tendrá que entregar, además de una versión completamente funcional y completa del código que cumpla con los requisitos expuestos en anteriores secciones, un análisis del mismo, así como algunas propuestas de mejoras futuras y posibles alternativas a la implementación realizada.

La plantilla de documentación a entregar estará disponible en el campus virtual de la asignatura, e incluirá los siguientes puntos:

1. Eficiencia y complejidad: cada algoritmo implementado, y cada operación de un contenedor debe tener asociada una complejidad que depende de la implementación y de la estructura de datos, así como de la estructura de los datos

[4]Por ejemplo, en la operación que devuelve los barrios con mayor y menor número de calles, la operación devolverá dos parámetros de salida, que serán presumiblemente punteros a objetos de la clase `Barrio`.

en la aplicación. Se debe proporcionar esta información como resultado de la implementación.

2. Alternativas a la implementación actual: se debe analizar alguna alternativa a la implementación y estructura de datos elegida, incluyendo las implicaciones que tendría en el rendimiento y complejidad dicha alternativa. Como ejemplo, podemos suponer las dos alternativas a la organización de los datos descritas en la sección 2.4.

3. Propuestas futuras y discusión del diseño: se deben proponer mejoras al diseño implementado, como mejorarlo con estrucuras de datos má avanzadas, y qué implicaciones tendrían estos cambios.

5. Evaluación

La evaluación de la implementación tomará en cuenta varios puntos, de los cuales algunos son de obligado cumplimiento, mientras otros son recomendaciones para mejorar la calidad de la implementación. Los siguientes puntos no van en detrimento de lo especificado en el resto del documento.

Obligatorios:

1. Uso de punteros en la implementación, tanto en variables como en estructuras de datos, evitando así los problemas derivados de mover datos en memoria y mantener distintas copias (consistencia).

2. Diseño correcto de interfaces, incluidas la documentación (pre/post-condiciones y descripción) y evaluación de la complejidad de las operaciones.

3. Organización correcta de los datos en memoria de forma que la implementación de los algoritmos cumplan con los requisitos de eficiencia y uso de memoria.

4. Corrección en el resultado de la ejecución de los algoritmos.

5. Utilización correcta de memoria, en cuanto a utilización de la necesaria y reserva/liberación de forma correcta.

6. El lenguaje de programación será C++ y se utilizará el entorno de programación Eclipse. No está permitido utilizar la STL[5] en el desarrollo, ni cualquier otra biblioteca similar.

Recomendados:

[5]Standard Template Library

1. Uso correcto y eficiente de parámetros en la invocación de operaciones.
2. Sobrecarga de operadores para operaciones con objetos, especialmente cuando supongan copias de objetos.
3. Uso de constructores y destructores para las clases de forma coherente y adecuada.
4. Siempre que sea posible, no se duplicarán objetos en la aplicación, y se evitarán copias y duplicados de los mismos.

La evaluación final y calificación del proyecto se llevará a cabo mediante una rúbrica que contendrá los puntos anteriores y que estará a disposición del alumno. Además, las siguientes normas se deben seguir en la realización del proyecto:

- El proyecto debe realizarse en grupos de dos personas pertenecientes a grupos del mismo profesor.
- La entrega del proyecto consistirá en un fichero `zip` cuyo nombre seguirá el formato `ÄpellidosAlumno1_ApellidosAlumno2.zip¨`.
- El fichero debe contener: todo el código fuente generado, los test implementados para probar las funcionalidades y la documentación.
- Cualquier intento de copia, detección de entrega un código que no es propio, o de que uno de los alumnos de la pareja no ha trabajado en el desarrollo, será motivo de suspenso (0) en la asignatura (para los dos alumnos), y puesta en conocimiento de la autoridad del centro.

La fecha límite de entrega del proyecto es el día **jueves 21 de marzo**, a través de la correspondiente tarea abierta en el campus virtual de la asignatura. No se tendrán en cuenta entregas realizadas por cualquier otro medio que no sea el establecido, ni con posterioridad a esa fecha. Los alumnos que no entreguen la primera parte del proyecto de programación en la fecha indicada, u obtengan un suspenso en la misma, tendrán la opción de entregar las tres partes en la convocatoria oficial final de la asignatura. Por tanto, no podrán entregar en la segunda parte del proyecto.

Bibliografía

[1] Ayuntamiento de Cáceres, Open Data Cáceres. Disponible en: http://opendata.caceres.es

[2] B. Stroustrup, *Programming principles and practice using C++*, 2nd ed. Pearson, 2014. Capítulo 19 (sección 3).