Estructuras de Datos y de la Información

Proyecto de Programación 2019/20

2ª Parte

Por:
Profesores de EDI

Índice general

1. Introducción

Este documento describe los requerimientos y funcionalidad básica para el desarrollo de la segunda parte del proyecto de programación de la asignatura *Estructuras de Datos y de la Información* del curso 2019-20. Esta segunda parte será una ampliación de la primera, y por tanto contendrá todo el código y los algoritmos de la primera parte (con las mejoras solicitadas como retroalimentación) y añadirá lo que se especifique en este documento.

Como en la primera parte del proyecto, en esta 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, vías y ocio de la ciudad.

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, incluido los contenedores de datos que usaremos en esta parte, la *Lista con Punto de Interés* y el ABB, *Árbol Binario de Búsqueda*. 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

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

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 de esta segunda parte, se partirá del código desarrollado en la primera parte, que usaba una plantilla de código, previamente disponible. Esta estructura estaba formada por una clase principal (Algoritmos) en la que se deben 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[2].

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 *vís* de la ciudad.
`Bar.csv`	Archivo que contiene los *bares y restaurantes* de la ciudad.

Cuadro 1: Ficheros de datos para la aplicación.

Algunas de las vías en el archivo aparecen duplicadas, debido a que pertenecen a distintos barrios. Este es un detalle importante a la hora de almacenarlas en las estructuras adecuadas (véase sección 2.3). Asímismo, la longitud de algunas vías está en blanco, con lo que se puede tomar como 0 m.

[2]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.

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 tres clases que contendrán la descripción de cada uno de los tipos de datos en nuestra aplicación: `Barrio`, `Via` y `Bar`. 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, (4) operadores sobrecargados necesarios, y (5) una operación para mostrar la información en pantalla. A estas operaciones básicas se podrán añadir otras si es necesario. Estas clases deben haber sido programadas y probadas en la primera entrega de este proyecto.

2.3. Contenedores de datos

Las estructuras de datos básicas que vamos a utilizar en nuestra aplicación (sección 1) son la *Lista con Punto de Interés* y *Árbol Binario de Búsqueda*. La implementación de dichos contenedores será proporcionada por los profesores de la asignatura, y se deben usar sus interfaces sin modificaciones.

Las clases `ListaPI` y `Arbol` se utilizarán para almacenar los datos en la aplicación. Será necesario utilizarlas correctamente siguiendo la sintaxis basada en *templates* de C++. Además, las *Precondiciones* y *Postcondiciones* para cada una de las operaciones se deben respetar por parte del programador[3].

Los criterios de ordenación (si los hubiera) de los datos en las estructuras de datos de la aplicación debe quedar claramente definido.

2.4. Relaciones entre clases

Como se ha mencionado anteriormente, los datos se almacenarán en contenedores de tipo *ListaPI* y *Arbol*. Las estructuras implementadas en la primera parte se deben mantener. Además, se añadirá una única estructura de datos árbol para gestionar todos los bares. El programador debe tener en cuenta que los objetos en la aplicación no pueden estar duplicados (se recomienda el uso de punteros).

La estructura interna de clases y contenedores debe ser la siguiente:

[3]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.

1. Los barrios se almacenarán en una ListaPI.
2. Cada barrio contendrá una lista con únicamente las vías que pertenecen a ese barrio.
3. Los bares se almacenarán en una lista independiente y también en un árbol binario de búsqueda. La relación de los bares con las vías viene establecida por el código de la vía.

3. Algoritmos

A continuación se enumeran los algoritmos a implementar en la segunda parte de nuestro proyecto. Tened en cuenta que **(1) no se deben utilizar estructuras auxiliares a menos que se indique explícitamente, y (2) siempre que sea posible se debe implementar el algoritmo sobre la estructura de datos árbol**.

1. Escribir un algoritmo iterativo sobre la estructura de datos árbol que genere un archivo de tipo texto con los nombres de los bares en orden alfabético inverso.
2. Generar un fichero de texto con todos los bares de una determinada vía dado su nombre completo (tipo y nombre).
3. Dado el nombre de un bar, devolver sus datos completos y el nombre de la vía en la que se encuentra. El algoritmo se debe implementar sobre el árbol.
4. Devolver una lista ordenada por capacidad de todos los bares cuyo nombre comience por una determinada subcadena. El algoritmo debe ser lo más eficiente posible y ser implementado sobre la estructura de datos árbol.
5. Procesar un fichero con distintas operaciones sobre bares: (A) altas, (B) bajas y (M) modificaciones. Una modificación no conlleva la modificación del nombre del bar. Las estructuras de datos que gestionan bares deben quedar consistentes. Después del tipo de operación se indica el número de bares sobre el que hay que aplicar la modificación. Por ejemplo:

```
A#5
La Gambita;30;SI;TAPAS;668402756;10004;4034;2;9.00;24.00
Casa Pepa;20;SI;TAPAS;668300744;10004;239;21;9.00;24.00
El Bodegón;20;SI;TAPAS;;10004;2468;12;9.00;24.00
Varadero;30;SI;TAPAS;;10004;3052;21;7.00;24.00
Los Templarios;50;SI;TAPAS;606770995;10001;977;6;11.00;24.00
M#2
Homarus;100;SI;MEDITERRÁNEA;927235582;10005;3609;3;12.00;01.00
El Corral de las Cigüeñas;90;NO;;647758245;10003;773;6;13.00;1.00
B#2
Vadepizza
Hornos 25
```

6. Devuelve una lista con los bares que sirvan un determinado tipo de comida y cuya capacidad sea superior a una capacidad dada. El algoritmo se implementará sobre el árbol.

7. Devolver el barrio que más bares tiene, y escribir sus bares en un fichero de texto.

Todos los algoritmos deben ser implementados 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. 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 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.

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

5. Evaluación

La evaluación de la implementación tomará en cuenta los criterios generales que se especifican a continuación:

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 (*eficiencia*) 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 cumpla 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[4] en el desarrollo, ni cualquier otra biblioteca similar.
7. Uso correcto y eficiente de parámetros en la invocación de operaciones.
8. Uso de constructores y destructores para las clases de forma coherente y adecuada.
9. Siempre que sea posible, no se duplicarán objetos en la aplicación, y se evitarán copias y duplicados de los mismos.
10. El proyecto deberá incluir obligatoriamente documentación externa siguiendo la plantilla proporcionada por los profesores.

La evaluación final y calificación del proyecto se llevará a cabo mediante una rúbrica (a modo de lista de comprobación) que contendrá varios items basados en los puntos

[4]Standard Template Library

anteriores. Dicha rúbrica estará a disposición del alumno con antelación suficiente.

Además, las siguientes normas se deben seguir en la realización del proyecto:

- El proyecto puede 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 `ApellidosAlumno1_ApellidosAlumno2.zip`.
- El fichero debe contener: todo el código fuente generado para las dos partes del proyecto, los test implementados para probar las funcionalidades (solo para la parte de la primera entrega) y la documentación en formato PDF.
- Cualquier intento de copia, detección de entrega de código no 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 todos los alumnos implicados), y puesta en conocimiento de la autoridad del centro.

La fecha límite de entrega del proyecto es el día **2 de Junio a las 23:55h** 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.