PEC1 – Primera prueba de evaluación continua

Presentación

A continuación, os presentamos el enunciado de la primera prueba de evaluación continuada del curso. Tened en cuenta que la PEC debe resolverse individualmente.

Competencias

Las competencias que trabajaréis a la PEC son las siguientes:

Específicas

- Capacidad para analizar un problema en el nivel de abstracción adecuado a cada situación y aplicar las habilidades y conocimientos adquiridos para resolverlo.
- Capacidad para diseñar y construir aplicaciones informáticas mediante técnicas de desarrollo, integración y reutilización.
- Capacidad para proponer y evaluar diferentes alternativas tecnológicas para resolver un problema concreto.

Transversales

- Uso y aplicación de las TIC en el ámbito académico y profesional.
- Capacidad para innovar y generar nuevas ideas.

Objetivos

Los objetivos que se persiguen en el desarrollo de la PEC son los siguientes:

- Entender el concepto de TAD y saber hacer la especificación.
- Conocer la biblioteca de TADs de la asignatura y saber utilizarlos para diseñar e implementar nuevas estructuras de datos.
- Saber calcular la eficiencia espacial y temporal de una estructura de datos y los algoritmos asociados para comparar diferentes alternativas y poder elegir la mejor en términos de eficiencia (temporal o espacial).
- Ser capaz de identificar la estructura de datos utilizada en un programa y entender su funcionamiento.

- Entender el funcionamiento de los contenedores secuenciales y los árboles presentados en la asignatura. Saber cuándo y cómo utilizar estos contenedores.

Descripción de la PEC a realizar

La PEC consta de 4 ejercicios, algunos más teóricos y otros más prácticos, en los que el podréis en práctica los conocimientos adquiridos en el estudio de los tres primeros módulos de la asignatura.

Recursos

Los recursos necesarios para desarrollar la PEC son los siguientes:

Básicos (material didáctico de la asignatura)

- Módulo 1
- Módulo 2
- Módulo 3

Complementarios

No se requieren materiales complementarios.

Criterios de valoración

La puntuación global de la PEC es la suma de las puntuaciones individuales obtenidas en cada uno de los ejercicios que la componen. La puntuación individual de cada ejercicio se especifica en cada uno de ellos.

Formato y fecha de entrega

Fecha de publicación

7 de octubre de 2016

Fecha de entrega

28 de octubre de 2016

Formato de entrega

La entrega debe hacerse a través del espacio de Entrega y registro de EC con un único fichero preferentemente en formato PDF o, si no es posible, Word o OpenOffice. Este fichero contendrá la solución (incluyendo las clases Java). Por favor, no copiéis el enunciado, haced

constar vuestro nombre en cada página (por ejemplo, con un pie de página) y numerad las páginas.

Enunciado

Se quiere diseñar una estructura de datos para la DGT que gestione las infracciones de tráfico y el carnet por puntos. En esta PEC haremos una prueba piloto de una parte del sistema (un conjunto reducido de funcionalidades) y trabajaremos con volúmenes de información pequeños para que os familiarizáis con el problema a resolver y practiquéis el diseño y la composición de estructuras de datos para dar forma y solución al problema planteado. Concretamente, para la realización del ejercicio considerad:

- El número de vehículos V será grande y constante.
- El número de conductores C será grande y creciente.
- El número de vehículos de un conductor VC es pequeño y limitado a 10.
- El número de infracciones de un conductor IC será muy variable y en constante aumento.
- El número de tipos de infracción TI será pequeño y limitado, de centenares.

Ejercicio 1 [2’5 puntos]

Especificad un TAD ***GestorInfracciones*** que permita las operaciones:

- Añadir un nuevo conductor al sistema. De cada conductor sabremos su DNI que lo identificará, su nombre y los apellidos. Si ya existe un usuario con este DNI, actualizamos sus datos.
- Añadir un nuevo vehículo al sistema. De cada vehículo sabremos la matrícula que lo identificará, la marca, el modelo y el DNI del conductor habitual. Si el vehículo ya existe devuelve un error. Si el DNI del conductor habitual no existe, devuelve un error. Tened en cuenta que un vehículo sólo puede tener un conductor habitual pero que un conductor puede ser conductor habitual de hasta 10 vehículos. Si el conductor habitual a vincular con el vehículo ya es conductor habitual de 10 vehículos devuelve un error.
- Añadir un nuevo tipo de infracción al sistema. De cada tipo de infracción sabremos un identificador, el artículo del código de circulación infringido, una descripción y el número de puntos que descuenta. Si el tipo de infracción ya existe actualizamos los datos (no hace falta recalcular los puntos a descontar por los usuarios que hayan cometido esta infracción)
- Registrar una infracción cometida por un vehículo. De cada infracción cometida sabremos el vehículo, la fecha y hora y un texto descriptivo. La infracción se asignará al conductor habitual del vehículo y le descontará automáticamente los puntos asociados al tipo de infracción. Si el vehículo o el tipo de infracción no existen, devuelve un error. Los pasos a realizar por esta operación sueño los siguientes:

    - Si el infractor tiene 0 puntos disponibles antes de la infracción, se debe indicar una un mensaje de aviso ("El infractor ya tenía retirado el carnet") y registrar la infracción.
    - Se debe realizar el descuento de puntos (en caso de que el descuento sea un número negativo el infractor se quedará con 0 puntos)
    - Si como resultado de la sanción el infractor tiene 0 puntos disponibles se debe indicar el mensaje de aviso: "El infractor tiene retirado el carnet"
    - Si como resultado de la sanción el infractor todavía tiene puntos disponibles también se debe indicar el mensaje de aviso: "Al infractor le quedan X puntos"
- Consultar el historial de infracciones de un conductor. Considerad que el conductor debe existir. El historial tiene que mostrar todas las infracciones cometidas por el conductor con todos los vehículos que es conductor el habitual por orden cronológico.
- Consultar los puntos del conductor habitual de un vehículo. Considerad que el vehículo debe y que sólo se proporciona la matrícula del vehículo. Si el conductor habitual del vehículo tiene 0 puntos disponibles, se debe indicar un mensaje: "El infractor tiene retirado el carnet
- Consultar los 5 tipos de infracción más habituales ordenados de mayor a menor. En caso de empate no importa el orden.

Las operaciones consultoras tienen que ser lo más eficientes posible.

Apartado a) [1 punto]

Especificad la firma del TAD ***GestorInfracciones***. Es decir, indica el nombre que darías a las operaciones encargadas de cada funcionalidad requerida. Indica, también, los parámetros de entrada y de salida cuando sean necesarios.

Solución

- addDriver(dni, name, surname)

- addCar(carPlate, brand, model, usualDriverId)

- addTrafficViolationType(id, article, description, points)

- registerTrafficViolation(carPlate, idTrafficViolation, date, time, description)

- getDriverTrafficViolations(dni)

- getDriverPoints(carPlate)

- topTrafficViolations()

Apartado b) [1,5 puntos]

Haced la especificación contractual de las operaciones del TAD ***GestorInfracciones***. En la redacción de la especificación puedes usar, si se requiere, cualquiera de las operaciones del TAD. Toma como modelo la especificación del apartado 1.2.3 del Módulo 1 de los materiales docentes. Se valorará especialmente la concisión (ausencia de elementos redundantes o

innecesarios), precisión (definición correcta del resultado de las operaciones), completitud (consideración de todos los casos posibles en que se puede ejecutar cada operación) y carencia de ambigüedades (conocimiento exacto de cómo se comporta cada operación en todos los casos posibles) de la solución. Es importante responder este apartado usando una descripción condicional y no procedimental. La experiencia nos demuestra que no siempre resulta fácil distinguir entre las dos descripciones, es por eso que hacemos especial énfasis insistiendo que pongáis mucha atención en vuestras definiciones.

A título de ejemplo indicaremos que la descripción condicional (la correcta a utilizar en el contrato) de llenar un vaso vacío con agua sería:

> @pre el vaso se encuentra vacío.
> @post el vaso está lleno de agua.

En cambio, una descripción procedimental (incorrecta para utilizar en el contrato) tendría una forma parecida a:

> @pre el vaso tendría que encontrarse vacío, si no se encontrase vacío se tendría que vaciar.
> @post se acerca el vaso al grifo y se echa agua hasta que esté lleno.

También debéis tener en cuenta el invariante si este es necesario para describir el TAD.

Solución

@pre cierto.

@post si el código de conductor es nuevo, los conductores serán los mismos más un nuevo conductor con los datos indicados. Sino los datos del conductor se habrán actualizado con los nuevos.

- addDriver(dni, name, surname)

@pre cierto.

@post los vehículos serán los mismos más uno nuevo con los datos indicados y los vehículos del conductor serán los mismos más el vehículo indicado. Si existe un vehículo con matrícula *carPlate*, no existe un conductor con identificador *usualDriverId* o bien el conductor ya tiene 10 coches devuelve error.

- addCar(carPlate, brand, model, usualDriverId)

@pre No se superará el máximo de tipos de infracción.

@post si el código del tipo de infracción es nuevo, los tipos de infracción serán los mismos más un nuevo tipo con los datos indicados. Sino los datos del tipo de infracción se habrán actualizado con los nuevos. Si cambian los puntos no se actualizarán las infracciones de este tipo ya cometidas por los conductores.

- addTrafficViolationType(id, article, description, points)

@pre cierto.

@post las infracciones del conductor serán las mismas más una nueva con los datos indicados. El número de puntos del conductor serán los mismos menos los puntos que resta la infracción cometida excepto en el caso en que el resultado sea menor que 0, en cuyo caso el número de puntos del conductor será 0 y se debe notificar. Si no existen el vehículo o el tipo de infracción devuelve error.

- registerTrafficViolation(carPlate, idTrafficViolation, date, time, description)

@pre existe un conductor con identificador *dni.*

@post devuelve un iterador para recorrer las infracciones del conductor por orden cronológico.

- getDriverTrafficViolations(dni)

@pre existe un vehiculo con matrícula *carPlate*.

@post devuelve las infracciones cometidas por el conductor habitual del vehiculo.

- getDriverPoints(carPlate)

@pre cierto

@post devuelve un iterador para recorrer las 5 infracciones más habituales ordenadas según el número total de infracciones cometidas..

- topTrafficViolations()

Ejercicio 2 [3,5 puntos]

En el ejercicio 1 habéis definido la especificación de un nuevo TAD, el TAD ***GestorInfracciones***. Ahora os pedimos que hagáis el diseño de las estructuras de datos que formarán este TAD. Diseñad, pues, el sistema para que sea el máximo de eficiente posible, tanto a nivel de eficiencia espacial como temporal, teniendo en cuenta los volúmenes de información y las restricciones especificadas en el enunciado.

Para la realización de la PEC1 os tendréis que limitar a los TAD's del módulo 3 de los apuntes. Quedan descartados, por tanto, los árboles y las tablas de dispersión.

Tened en cuenta sólo las operaciones que se piden en el enunciado al hacer este diseño.

Apartado a) [0,5 puntos]

Dudamos entre utilizar un vector, un vector ordenado o una lista encadenada ordenada para almacenar los **conductores**. Justificad cuál creéis que es la mejor opción.

Solución

El enunciado nos dice que el número de conductores C será grande y creciente y, por lo tanto, necesitamos un contenedor no acotado. Los vectores, ordenados o no, son estructuras acotadas que no nos serán útiles en este caso. Por eliminación nos quedamos, pues, con la **lista encadenada**.

Apartado b) [0,5 puntos]

Dudamos entre utilizar un vector, un vector ordenado o una lista encadenada ordenada para almacenar los **vehículos**. Justificad cuál creéis que es la mejor opción.

Solución

El enunciado nos dice que el número de vehículos es grande pero constante. Esto nos desaconseja utilizar una estructura dinámica como la lista encadenada ordenada puesto que malgastamos espacio.

Si usamos un vector tendremos que hacer búsquedas lineales para buscar los vehículos y, si usamos un vector ordenado, podríamos mejorar esta eficiencia en logarítmica. Cómo que el número de vehículos es grande y hay algunas operaciones consultoras que tienen que buscar por identificador de vehículo elegimos un vector ordenado puesto que las operaciones de consulta se verán muy favorecidas.

Apartado c) [0,5 puntos]

Dudamos entre utilizar un vector, un vector ordenado o una lista encadenada para almacenar los **vehículos de un conductor**. Justificad cuál creéis que es la mejor opción.

Solución

Por el volumen de información a guardar cualquier de las tres opciones sería válida. Dado que no hay ninguna operación que requiera una busca por los vehículos de un conductor y el enunciado nos dice que el número de vehículos de un conductor está limitado a 10 nos decantamos por un simple vector de 10 posiciones.

Apartado d) [1 punto]

Justificad el resto de las estructuras de datos para hacer el diseño del TAD. La justificación de cada una de las estructuras de datos tiene que ser del estilo:
"Para guardar XXX elegimos una lista encadenada ordenada puesto que el número de elementos no es muy grande, necesitamos acceso directo y recorridos ordenados."

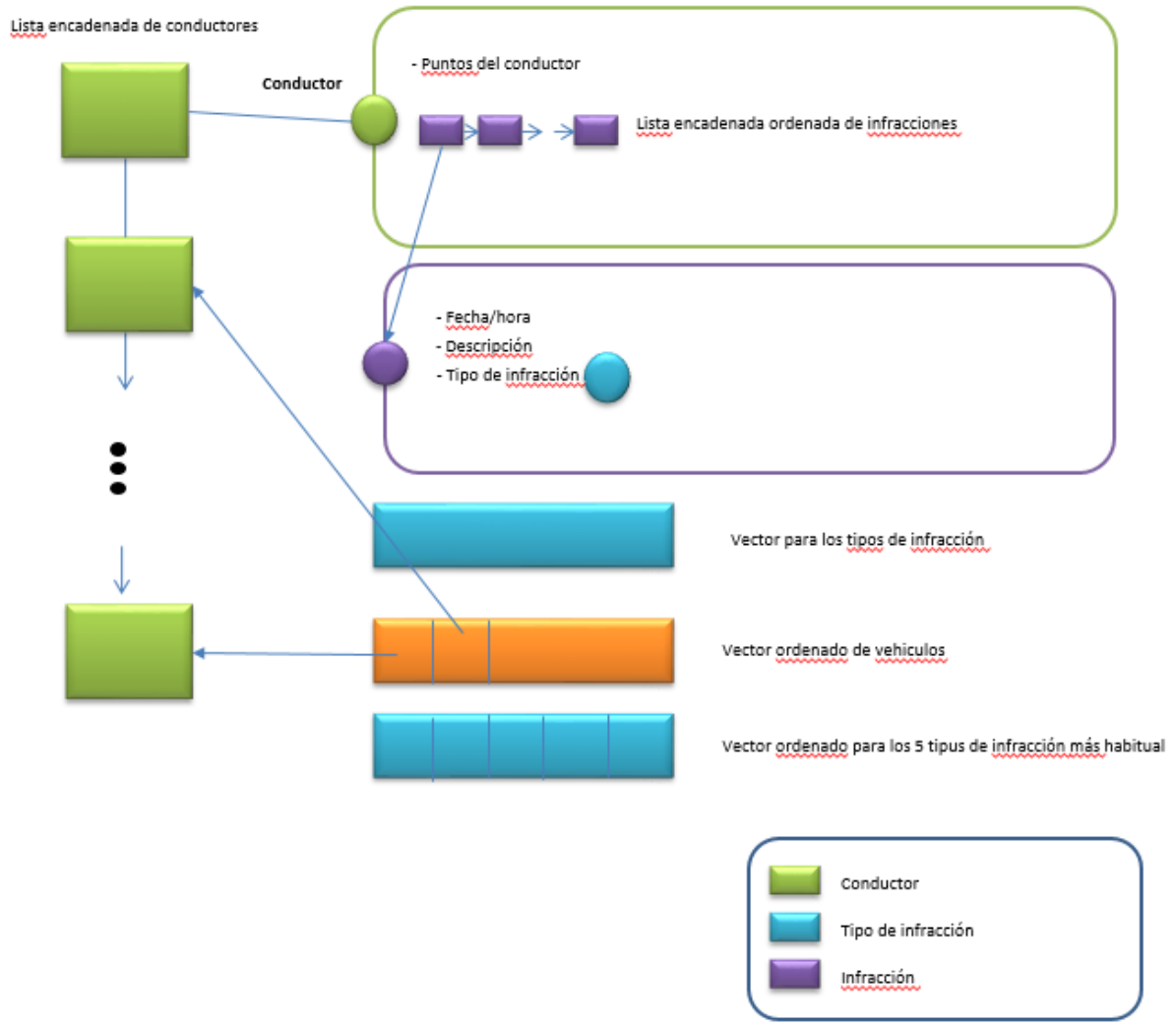
Solución

- Para guardar las infracciones de los conductores utilizaremos una **lista encadenada** ordenada puesto que IC será variable, en constante aumento y queremos la lista de infracciones por orden cronológico.
- Para guardar los 5 tipos de infracción más habitual utilizaremos un **vector ordenado** de 5 posiciones. Debido al número pequeño de elementos también sería válido un vector sin ordenación, pero en este caso haría falta una estructura auxiliar puesto que la operación de consultar los 5 tipos de infracción más habituales se tiene que devolver ordenados de mayor a menor.
- Para guardar los tipos de infracción utilizaremos un **vector** puesto que TI será pequeño y limitado, de centenares.

Apartado e) [1 punto]

Haced un dibujo de la estructura de datos global para el TAD ***GestorInfracciones*** donde se vean claramente las estructuras de datos que elegís para representar cada una de las partes y las relaciones entre ellas. Haced el dibujo de la estructura completa, con todas las estructuras que os permitan implementar las operaciones definidas en la especificación.

Solución



Ejercicio 3 [3 puntos]

En el ejercicio 1 habéis definido la especificación del TAD ***GestorInfracciones*** con sus operaciones y en el ejercicio 2 habéis elegido las estructuras de datos para cada parte del TAD. En este ejercicio os pedimos que os fijéis en los algoritmos que os servirán para implementar algunas de las operaciones especificadas y en el estudio de eficiencia de las mismas. Tened en cuenta que la implementación de las operaciones va estrechamente ligada a la elección de las estructuras de datos que hayáis hecho.

Apartado a) [1 puntos]

Describid y haced el estudio de eficiencia de la operación que habéis definido para registrar una infracción cometida por un vehículo. Para hacerlo tenéis que describir brevemente su comportamiento indicando los pasos que la componen (con frases como, por ejemplo: "insertar en el árbol AVL / borrar de la tabla de dispersión / consulta del pilón / ordenar el

vector..."), especificando la eficiencia asintótica de cada paso y dando la eficiencia total de la operación.

Solución

- Buscar el vehículo en el vector ordenado de vehículos => O(log V)
- Acceder a su conductor habitual => O(1)
- Añadir la infracción a la lista encadenada de infracciones del conductor => O(1)
- Actualizar el número de puntos restantes del conductor => O(1)
- Buscar el tipo de infracción en el vector de tipo de infracciones => O(TI)
- Incrementar el contador de número de infracciones de este tipo que se ha cometido => O(1)
- Actualizamos los 5 tipos de infracción mas habituales => O(1)
    - Si el vector de tipo de infracción más habituales está lleno:
        - Coger el elemento con menos infracciones al vector de tipo de infracción más habituales (denominado B) => O(1)
        - Comparar el número de infracciones de B con el de la infracción cometida y sustituirlo si hace falta => O(1)
- Si el vector de tipos de infracción más habituales no está lleno añadimos el tipo de infracción => O(1)

Total: O(log V + TI) y, como TI está acotado a unos centenares (constante), podemos decir que la eficiencia asintótica es O(log V)

Apartado b) [1 punto]

Discutid si, sólo teniendo en cuenta las operaciones requeridas en el enunciado, os hace falta una estructura de datos para almacenar los vehículos asociados a un conductor. ¿Necesitáis una estructura de datos para almacenar los vehículos asociados a un conductor si os pidiéramos una operación que nos volviera el vehículo con el que un conductor ha cometido más infracciones y no quisiéramos que fuera lineal respeto el número de infracciones de un conductor?

Solución

Si sólo tenemos en cuenta las operaciones requeridas en el enunciado NO nos hace falta esta estructura puesto que no la utilizamos para nada.
Si necesitamos una operación que nos vuelva el vehículo con el que un conductor ha cometido más infracciones tampoco nos haría falta necesariamente una estructura de datos para almacenar los vehículos asociados a un conductor puesto que lo podríamos hacer simplemente recorriendo la lista de infracciones cada vez. Esto, pero, puede ser bastante ineficiente y, seguramente, es mejor guardar un atributo en la clase conductor que se vaya actualizando cada vez que el conductor comete una infracción.

Apartado c) [1 punto]

Suponed ahora que queremos ofrecer una funcionalidad que permita consultar el historial de infracciones de un vehículo y no queremos que sea lineal respecto al número de infracciones de un conductor. El historial tiene que mostrar todas las infracciones cometidas

por el vehículo por orden cronológico. Explicad qué cambios haríais en la estructura de datos y el algoritmo que utilizaríais.

Solución

Si no hacemos ningún cambio la única manera de obtener el historial de infracciones de un vehículo es consultando el vehículo en el vector ordenado de vehículos, acceder al conductor habitual y recorrer su lista de infracciones cogiendo sólo las del vehículo que nos interesa, añadirlas a una estructura auxiliar (una lista encadenada ordenada, por ejemplo) y devolver un iterador de esta nueva estructura. Cómo que hay que recorrer toda la lista de infracciones de un conductor la operación sería lineal respecto al número de infracciones de un conductor y esto no nos lo permite el enunciado.

En este caso, como que el enunciado nos especifica una cota de eficiencia temporal a cumplir, una manera de hacerlo sería sacrificado eficiencia espacial y añadir una lista encadenada ordenada a cada vehículo que contenga las infracciones del vehículo. Con esto sólo tendremos que consultar el vehículo al vector ordenado de vehículos O(log V) y retornar un iterador de la lista ordenada de infracciones de este vehículo.

Ejercicio 4 [1 punto]

Indicad cuál de los TADs de la biblioteca de TADs de la asignatura te parecen más adecuados para utilizarlos en la implementación de cada una de las estructuras de datos definidas para el TAD ***GestorInfracciones***.

Solución

Para guardar los **conductores** una ***ListaEncadenada***.

Para guardar las **infracciones de un conductor** utilizaremos una ***ListaEncadenadaOrdenada*** que habrá que implementar.

Para guardar los **vehículos** y los **tipos de infracción más habituales**, implementaremos una nueva clase que implemente un **vector ordenado** con operaciones para añadir ordenadamente y hacer consultas mediante búsqueda dicotómica. Para integrar esta clase en la biblioteca de clases, implementaremos las interfaces de ContenidorAcotado y de Diccionario.

Para guardar los **tipos de infracción**, implementaremos una nueva clase que implemente un **vector** con operaciones para añadir y hacer consultas mediante búsqueda lineal. Para integrar esta clase en la biblioteca de clases, implementaremos las interfaces de ContenidorAcotado y de Diccionario. Si pudiéramos utilizar las colecciones del jdk nos serviría un simple Vector.