Estructuras de Datos y Algoritmos

Grados en Ingeniería Informática

Examen Septiembre, Segundo Cuatrimestre, 12 de septiembre de 2017

Nombre: _____ **Grupo:** _____

Laboratorio: _____ **Puesto:** _____ **Usuario de DOMjudge:** _____

1. **(2.5 puntos)** Extiende la clase `Queue` con una nueva operación cuya cabecera en `C++` es

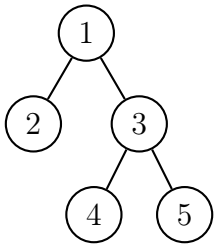
```
void llevarAlPrincipio(unsigned int pos);
```

que lleva el elemento que está en la posición *pos* de la cola a la primera posición de la misma. Ten en cuenta que $pos = 1$ es el primer elemento de la cola; $pos = 2$ es el segundo; y así sucesivamente. Si la posición de la cola no existe, deberá lanzarse una excepción `ExcepcionTAD` con mensaje `Posicion inexistente`.

Aparte de implementar esta operación, debes indicar la complejidad de la misma. Para resolver este ejercicio no se puede crear ni destruir memoria dinámica, ni tampoco modificar los valores almacenados en la cola.

Copia el fichero `Queue.h` en `Queue_modificado.h` y haz las modificaciones en este fichero. Escribe al principio un comentario indicando las modificaciones que has hecho, justificando tu solución. El código en el fichero `main1.cpp` prueba la nueva función. Este código **no** puede ser modificado.

2. **(2.5 puntos)** Implementa una función `hojas` que, dado un árbol binario de enteros y un número entero no negativo k, determine el número de hojas cuya profundidad es mayor que k. Por ejemplo, para el siguiente árbol



la función devolvería 3 si $k = 0$ o $k = 1$, devolvería 2 si $k = 2$ y devolvería 0 si $k \geq 3$.

Aparte de implementar esta función, debes indicar la complejidad de la misma.

El fichero `main2.cpp` contiene ya el código necesario para este ejercicio, salvo la función `hojas`.

3. **(5 puntos)** La DGT nos ha pedido ayuda para gestionar el *carnet por puntos*. Los conductores están identificados de manera unívoca por su DNI y la cantidad de puntos de un conductor está entre 0 y 15 puntos inclusives. La implementación del sistema se deberá realizar como un TAD `carnet_puntos` con las siguientes operaciones:

   - `nuevo(dni)`: Añade un nuevo conductor identificado por su `dni` (un `string`), con 15 puntos. En caso de que el dni esté duplicado, la operación lanza una excepción `ExcepcionTAD` con mensaje `Conductor duplicado`.

- `quitar(dni, puntos)`: Le resta puntos a un conductor tras una infracción. Si a un conductor se le quitan más puntos de los que tiene, se quedará con 0 puntos. Si los puntos resultantes de esta operación son los mismos que los que tiene el conductor actualmente, entonces la operación debe ignorarse. En caso de que el conductor no exista, lanza una excepción `ExcepcionTAD` con mensaje `Conductor inexistente`.
- `recuperar(dni, puntos)`: Le añade puntos a un conductor enmendado. Si debido a una recuperación un conductor supera los 15 puntos, se quedará con 15 puntos. Si los puntos resultantes de esta operación son los mismos que los que tiene el conductor actualmente, entonces la operación debe ignorarse En caso de que el conductor no exista, lanza una excepción `ExcepcionTAD` con mensaje `Conductor inexistente`.
- `consultar(dni)`: Devuelve los puntos actuales de un conductor. En caso de que el conductor no exista, lanza una excepción `ExcepcionTAD` con mensaje `Conductor inexistente`.
- `cuantos_con_puntos(puntos)`: Devuelve cuántos conductores tienen un determinado número de puntos. En caso de que el número de puntos no esté entre 0 y 15 lanza una excepción `ExcepcionTAD` con mensaje `Puntos no validos`.
- `lista_por_puntos(puntos)`: Produce una lista con los DNI de los conductores que poseen un número determinado de puntos. La lista estará ordenada por el momento en el que el conductor pasó a tener esos puntos, primero el que menos tiempo lleva con esos puntos. En caso de que el número de puntos no esté entre 0 y 15 lanza una excepción `ExcepcionTAD` con mensaje `Puntos no validos`.

La implementación de las operaciones debe ser lo más eficiente posible. Por tanto, debes elegir una representación adecuada para el TAD, implementar las operaciones y justificar la complejidad resultante.

El programa principal que prueba el nuevo TAD se encuentra en el fichero `main3.cpp`, que **no** puede ser modificado.

Normas de realización del examen

1. Debes programar soluciones para cada uno de los ejercicios, probarlas y entregarlas en el juez automático accesible en la dirección `http://exacrc/domjudge/team`.

2. Escribe comentarios que expliquen tu solución, justifiquen por qué se ha hecho así y ayuden a entenderla. Calcula la complejidad de todas las funciones que implementes.

3. En el juez te identificarás con el nombre de usuario y contraseña que has recibido al comienzo del examen. El nombre de usuario y contraseña que has estado utilizando durante la evaluación continua **no** son válidos.

4. Escribe tu **nombre y apellidos** en un comentario en la primera línea de cada fichero que subas al juez.

5. Descarga el fichero `http://exacrc/DocumA_SEP.zip` que contiene material que debes utilizar para la realización del examen (implementación de las estructuras de datos, ficheros con código fuente y ficheros de texto con algunos casos de prueba de cada ejercicio del enunciado).

6. Tus soluciones serán evaluadas por el profesor independientemente del veredicto del juez automático. Para ello, el profesor tendrá en cuenta **exclusivamente** el último envío que hayas realizado de cada ejercicio.