Estructura de Datos y Algoritmos

Grados de la Facultad de Informática de la UCM (Grupos C y F). Curso 2016-2017
Examen parcial 2º cuatr. de septiembre Tiempo: 3 horas

Ejercicio 1 [2.75 puntos]

Apartado a) [2.5 puntos] Extiende el TAD `Cola` visto en clase (`Queue.h`) con una nueva operación interna y pública cuya cabecera en `C++` es

```
void cuela(const T& a, const T& b);
```

que mueve al elemento `b` de su posición a la posición inmediatamente detrás del elemento `a`. En caso de haber múltiples apariciones de los elementos `a` y/o `b` se considerará: la primera aparición de `a`, y la primera aparición de `b` tras la primera aparición de `a`. Si alguno de los elementos no se encuentra en la cola, o bien, si `b` no aparece detrás de `a`, la operación no tendrá efecto. Indica y justifica la complejidad de la operación implementada. *Requisitos:* No se puede crear ni destruir memoria dinámica, ni tampoco modificar los valores almacenados en la cola.

Entrada	Salida
1 2 3 4 -1 1 3	1 3 2 4
1 2 3 4 -1 1 4	1 4 2 3
1 2 3 4 -1 2 1	1 2 3 4
3 1 2 1 3 4 -1 1 3	3 1 3 2 1 4

La función principal proporcionada para hacer pruebas lee la cola de enteros (secuencia de enteros no negativos acabando la lectura con un -1 que no se incluye en la cola), después los enteros `a` y `b`, llama a la función pedida, y muestra por pantalla la cola resultante (ver ejemplos). El proceso se repite hasta introducir una cola vacía (es decir, un -1).

Apartado b) [0.25 puntos] Indica y justifica la complejidad en tiempo y en espacio que tendría la operación del apartado a) si se implementase como una función externa al TAD `Queue`.

Ejercicio 2 [2.75 puntos]

Apartado a) [1.5 puntos] Extiende el TAD `TreeMap` visto en clase con una nueva operación interna y pública de nombre `keyInBounds`, que recibe una clave `k` y devuelve un booleano que será `true` si y solo si en la tabla existe alguna clave menor o igual que `k` y también alguna mayor o igual. Justifica la complejidad de la operación implementada.

Apartado b) [1 punto] Implementa una operación con igual nombre y comportamiento a la del apartado a) pero como función externa al TAD `TreeMap`, que por tanto recibe como parámetros un `TreeMap<K,V>` y una clave, y devuelve un booleano. Justifica la complejidad de la operación implementada.

Entrada	Salida
5 2 8 -1 3	si
5 2 8 -1 9	no
8 2 5 -1 7	si
8 2 5 -1 9	no

La función principal proporcionada para hacer pruebas en ambos apartados hace lo siguiente: construye un `TreeMap<int,Nada>` a partir de una secuencia de enteros no negativos que se van insertando, acabando la lectura con un -1 que no se inserta, después lee la clave `k` (un entero), llama al método o a la función pedida, y muestra por pantalla el resultado obtenido en el formato "si" o "no" (ver ejemplos). El proceso se repite hasta introducir una secuencia vacía (es decir, un -1).

Apartado c) [0.25 puntos] Indica y justifica las complejidades que tendrían la operación y la función de los apartados a) y b) para el caso de un `HashMap` en lugar de un `TreeMap`.

Ejercicio 3 [4.5 puntos]

Nos han encargado implementar un sistema para la gestión de la admisión en el Servicio de Urgencias de un Hospital. Cuando un paciente llega al servicio, se le toman los datos, se le asigna un código de identificación único (un número entero no negativo), y también se le asigna un nivel de urgencia, dependiendo de su estado (leve, normal, o grave). A partir de ahí, el paciente espera a ser atendido. El orden de atención da prioridad a los pacientes graves sobre los normales, y a los normales sobre los leves. Una vez atendido un paciente, sus datos se eliminan del sistema. Así mismo, en cualquier momento un paciente puede desistir de ser atendido. En este caso, en el control de salida se le solicita su número de identificación, y se elimina todo rastro de él del sistema. La implementación del sistema se deberá realizar como un TAD `GestionUrgencias` con las siguientes operaciones:

- `crea()`: Operación constructora que crea un sistema de gestión de urgencias vacío.
- `anadirPaciente(codigo, nombre, edad, sintomas, gravedad)`: Añade al sistema un nuevo paciente con código de identificación `codigo`, con nombre `nombre`, con edad `edad`, con una descripción de síntomas `sintomas`, y con código de gravedad `gravedad`. En caso de que el código ya se encontrase en el sistema la operación lanza una excepción con el mensaje "Paciente duplicado".
- `infoPaciente(codigo, nombre, edad, sintomas)`: Devuelve en `nombre`, `edad` y `sintomas` la información correspondiente del paciente con código `codigo`. En caso de que el código no exista, se lanza una excepción con el mensaje "Paciente inexistente".
- `siguiente(codigo, gravedad)`: Devuelve en `codigo` y `gravedad`, respectivamente, el código y la gravedad del siguiente paciente a ser atendido. Como se ha indicado antes, se atiende primero a los pacientes graves, después a los de nivel de gravedad normal, y por último a los leves. Dentro de cada nivel, los pacientes se atienden por orden de llegada. En caso de que no haya más pacientes se lanza una excepción con el mensaje "No hay pacientes".
- `hayPacientes()`. Devuelve `true` si hay más pacientes en espera, y `false` en otro caso.
- `elimina(codigo)`: Elimina del sistema todo el rastro del paciente con código `codigo`. Si no existe tal paciente, la operación no tiene efecto.

Se pide implementar al completo el TAD `GestionUrgencias` incluyendo todas las operaciones indicadas así como justificar la complejidad de cada operación. Debes elegir una representación adecuada para el TAD de manera que las operaciones sean eficientes.

Instrucciones

- Entrega un fichero .cpp para cada ejercicio, nombrado ejerX.cpp siendo X el número del ejercicio, excepto para los apartados a) de los ejercicios 1 y 2 en los que se entregará el Queue.h y TreeMap.h resp.
- Al principio de cada fichero debe aparecer, en un comentario, vuestro nombre y apellidos, dni y puesto de laboratorio. También debéis incluir unas líneas explicando qué habéis conseguido hacer y qué no.
- Todo lo que no sea código C++ (explicaciones, especificaciones, invariantes, etc.) debe ir en los propios ficheros en comentarios debidamente indicados.
- Los TADs vistos y las plantillas para poder probar vuestras soluciones se obtienen pulsando en el icono del Escritorio "Publicacion docente ...", después en "Alumno recogida docente", y en el programa que se abre, abriendo en la parte derecha la carpeta EDA-F, arrastrando los ficheros a hlocal (en la izqda).
- La entrega se realiza pulsando en el icono del escritorio "Examenes en Labs ...", y posteriormente, utilizando el programa que se abre, colocando los ficheros a entregar en la carpeta de vuestro puesto (en el lado derecho).