Algoritmos y Estructuras de Datos (VJ1215) - Universitat Jaume I

Examen final - 2024/2025 - Primera parte
Recuperación del primer examen parcial

5 de junio de 2025

Nombre:

El examen final consta de dos partes, cada una con una duración de dos horas. La primera parte está destinada a aquellos estudiantes que han solicitado intentar mejorar la calificación obtenida en cualquiera de los exámenes parciales, renunciando a la misma. Este primer ejercicio de la primera parte está dirigido a quienes buscan mejorar la nota del primer examen parcial.

La prueba es individual. No puedes consultar libros, apuntes ni dispositivos electrónicos. Escribe tu solución empleando el lenguaje C++, y no olvides los costes que se piden. Al finalizar entrega tu solución junto con el enunciado (no es necesario que entregues todas las hojas). Pon tu nombre en cada hoja que entregues.

Ejercicio 1 — 1 punto

Estamos utilizando una lista enlazada para implementar el Tipo Abstracto de Datos **Cola de Prioridad de Doble Fin**. Por simplicidad, supondremos que el único dato almacenado en cada elemento de la cola es su prioridad, pudiendo existir múltiples elementos con la misma prioridad.

La implementación debe soportar las siguientes operaciones con los tiempos de ejecución en el peor caso que se indican, siendo n la talla de la cola de prioridad:

```
class ColaDePrioridadDeDobleFin {
   struct Nodo {
      float prioridad;
      ...
   };
   ...
 public:
   ColaDePrioridadDeDobleFin();    // O(1)
   void  insertar(float);          // O(n)
   float consultarMinimo() const;  // O(1)
   float consultarMaximo() const;  // O(1)
   void  eliminarMinimo();         // O(1)
   void  eliminarMaximo();         // O(1)
};
```

a) **Diseño de la estructura de datos**

Explica brevemente cómo almacenarías los datos para garantizar que las operaciones tengan los costes indicados. Especifica qué atributos incluirías en los puntos suspensivos de la declaración de la clase.

b) **Implementación**

Añade a la clase `ColaDePrioridadDeDobleFin` el siguiente método:

```
void mezclar(const ColaDePrioridadDeDobleFin &)
```

Este método debe permitir que `c1.mezclar(c2)` añada a `c1` todos los datos que aparecen en `c2`, sin modificar `c2`.

- Si un dato inicialmente aparece n veces en `c1` y m veces en `c2`, al finalizar aparecerá $n + m$ veces en `c1`.
- Puede darse el caso de que `c1` y/o `c2` estén vacías.

- La solución debe funcionar correctamente en todos los casos.
- No se debe lanzar ninguna excepción.
- Para que tu solución sea válida, debe ser eficiente.
- Se valorará que la solución no siga recorriendo ninguna lista enlazada innecesariamente.

Además, implementa los constructores de `ColaDePrioridadDeDobleFin` y de `Nodo` necesarios para que tu solución funcione correctamente

Si utilizas otras funciones, debes implementarlas también.

c) **Análisis de costes**

Indica cuáles son los siguientes costes de tu solución en función de a y b, donde a es la talla de la primera cola de prioridad y b la talla de la segunda. No es necesario justificarlos.

Coste temporal en el peor caso	$O($ $)$
Coste espacial en el peor caso sin contar las listas enlazadas	$O($ $)$