Algoritmos y Estructuras de Datos (VJ1215) - Universitat Jaume I
Evaluación continua - 2022/2023

24 de octubre de 2022

Nombre:

La duración de esta prueba es de 80 minutos. La prueba es individual. No puedes consultar libros, apuntes ni dispositivos electrónicos. Al finalizar entrega tu solución junto con el enunciado (no es necesario que entregues todas las hojas). Pon tu nombre en todo lo que entregues. En todos los ejercicios:

- Escribe tu solución empleando el lenguaje C++.
- No puedes utilizar ninguna clase de las bibliotecas de C++ (`vector`, `stack`, `queue`, `priority_queue`, `map`, etc.). Lo que necesites, lo has de implementar.
- No puedes utilizar otros métodos o funciones si no los implementas también.
- Asegúrate de tratar bien todos los casos.

Ejercicio 1 — 4 puntos

Estamos utilizando una lista simplemente enlazada para realizar una implementación del Tipo Abstracto de Datos **Conjunto** a la que se ha añadido la posibilidad de consultar y eliminar el mínimo eficientemente. Para ello, tenemos los siguientes atributos, y no puedes añadir otros atributos en `Conjunto` ni en `Conjunto::Nodo` (los puntos suspensivos corresponden a posibles declaraciones de métodos):

```
class Conjunto {
   struct Nodo {
      int dato;
      Nodo * siguiente;
      ...
   };
   Nodo * primero;
   ...
 public:
   ...
};
```

Los datos se guardan de modo tal que el coste temporal en el peor caso de la operación `consultarMinimo` es $O(1)$, el de `eliminarMinimo` es $O(1)$, el de `insertar` es $O(n)$, el de `eliminar` es $O(n)$ y el de `buscar` es $O(n)$, siendo n la talla del conjunto. No se guardan datos repetidos.

Piensa cómo conseguir que esas operaciones tengan esos costes. Teniendo en cuenta eso, añade a la clase `Conjunto` un método `Conjunto interseccion(const Conjunto &) const`, de modo tal que el resultado que devuelva `c1.interseccion(c2)` sea un nuevo conjunto que contenga solamente los datos que tienen en común `c1` y `c2`. El resultado será un conjunto vacío si no hay ningún dato que esté en ambos conjuntos. No se deben modificar `c1` ni `c2`. Para que tu solución sea válida, debe ser eficiente. Implementa también los constructores de `Conjunto` y de `Conjunto::Nodo` necesarios para que tu solución funcione correctamente.

Indica en esta hoja cuál es el coste temporal en el peor caso de tu solución en función de a y b, siendo a la talla del primer conjunto y b la talla del segundo. No es necesario que lo justifiques.

Coste temporal en el peor caso	$O($ $)$

Ejercicio 2 — 3 puntos

Estamos utilizando un árbol binario de búsqueda para realizar una implementación del Tipo Abstracto de Datos **Conjunto** con datos de tipo real. No se guardan datos repetidos. Tenemos los siguientes atributos, y no puedes añadir otros atributos en `Conjunto` ni en `Conjunto::Nodo` (los puntos suspensivos corresponden a posibles declaraciones de métodos):

```
class Conjunto {
   struct Nodo {
      float dato;
      Nodo * izquierdo;
      Nodo * derecho;
      ...
   };
   Nodo * raiz;
   ...
 public:
   ...
};
```

La profundidad de un nodo mide la distancia desde la raíz hasta él. Si existen, el nodo raíz está a profundidad 0, los hijos de la raíz están a profundidad 1, los nietos de la raíz están a profundidad 2, etc.

Implementa un método `int consultarProfundidad(float dato) const` que devuelva la profundidad a la que se encuentra el dato, si se encuentra en el árbol. Si el dato no está en el árbol, debe devolver -1. Tu solución debe ser recursiva, no puedes utilizar ningún bucle.

Indica en esta hoja cuáles son los siguientes costes de tu solución en función de n, siendo n la talla del conjunto (es decir, la cantidad de nodos del árbol), y cuáles serían los costes si el árbol fuese AVL. No es necesario que lo justifiques.

	sin ser AVL	**si fuese AVL**
Coste temporal en el mejor caso	$O($ $)$	$O($ $)$
Coste temporal en el peor caso	$O($ $)$	$O($ $)$
Coste espacial en el peor caso sin contar el propio árbol	$O($ $)$	$O($ $)$

Ejercicio 3 — 3 puntos

Añade a la clase `Conjunto` del ejercicio 2 el método `void eliminarMaximo()` que elimine el máximo de los elementos del conjunto. Si el conjunto está vacío, no debe cambiar nada. Tu solución debe ser recursiva, no puedes utilizar ningún bucle.

Indica en esta hoja cuáles son los siguientes costes de tu solución en función de n, siendo n la talla del conjunto (es decir, la cantidad de nodos del árbol), y cuáles serían los costes si el árbol fuese AVL. No es necesario que lo justifiques.

	sin ser AVL	**si fuese AVL**
Coste temporal en el mejor caso	$O($ $)$	$O($ $)$
Coste temporal en el peor caso	$O($ $)$	$O($ $)$