Prácticas Árboles Binarios de Búsqueda

1. TAD Árbol Binario de Búsqueda

Utilizando la representación de árboles de la práctica del tema 1, haciendo las modificaciones oportunas para incluir el campo clave que nos permita ordenar los nodos, implementar las operaciones básicas de búsqueda, inserción y eliminación, teniendo en cuenta las siguientes especificaciones y prototipos:

int insertar(tipoÁrbol *a, tipoClave x,)

La operación insertar debe crear un nuevo nodo con clave *x* en el árbol con raíz *a*. Utilizaremos el campo información de cada nodo para almacenar el número de veces que se ha intentado insertar un nodo con clave *x* en el árbol *a*. La operación debe devolver cero si la inserción acaba correctamente y -1 si ha ocurrido algún fallo en el proceso.

int buscar(tipoClave x, tipoÁrbol a, tipoNodo **nodo)

Esta operación busca en el árbol con raíz *a* un nodo con valor *x* para la clave; si lo encuentra devuelve su dirección en el parámetro nodo. Además devolverá el número de veces que se ha insertado un nodo con clave x, es decir, si el nodo está en el árbol devuelve el contenido de la información del nodo y cero si no lo encuentra. Si ocurre algún error en el proceso devolverá -1.

int eliminar(tipoÁrbol *a, tipoClave x)

Operación que elimina del árbol con raíz *a* el nodo con clave *x* si existe. El algoritmo devolverá un valor mayor o igual que cero si la eliminación acaba correctamente y -1 si ocurre algún error en el proceso. El valor mayor o igual que cero que devuelve la función indicará el número de veces que se intentó insertar un nodo con clave *x* en el árbol *a*. **Modificar** el algoritmo de eliminación en árboles binarios de búsqueda visto en teoría de forma que el **criterio** para eliminar un nodo con dos hijos sea sustituirlo por el **nodo más a la izquierda del subárbol derecho.**

2. Aplicación TAD Árbol Binario de Búsqueda

Implementar un algoritmo que, utilizando el TAD de la práctica anterior, clasifique todas las palabras de un fichero de texto determinando cuantas veces aparece cada palabra. ¿Sería eficiente utilizar listas enlazadas en lugar de a.b.b.?

3. **Inserción y eliminación en Árboles Binarios de Búsqueda con claves duplicadas**

   Utilizando la representación de árboles del tema 1, haciendo las modificaciones oportunas para incluir el campo clave que nos permita ordenar los nodos, implementar las operaciones de inserción y eliminación, teniendo en cuenta los siguientes criterios

   1. Para todo nodo del a.b.b. se cumple que su clave es **mayor o igual** que todas las claves del subárbol izquierdo.
   2. Para todo nodo del a.b.b. se cumple que su clave es menor que todas las claves del subárbol derecho.
   3. Los nodos con claves duplicadas, si existen, deben ser nodos directamente enlazados

   Obsérvese que la definición admite claves duplicadas.

   El prototipo de la función insertar debe ser el siguiente:

   int insertarBB(tipoArbol *raíz, tipoClave clave, tipoInfo info)

   La función debe devolver 0 si la inserción acaba correctamente o -1 si se produce algún tipo de error en el proceso

   El prototipo de la función eliminar debe ser el siguiente:

   int eliminarBB(tipoÁrbol *raíz, tipoClave clave)

   La función debe devolver un número mayor o igual que cero indicando el número de nodos eliminados (todos los que tuvieran valor x para la clave) y un valor negativo (-1) si se produce algún tipo de error en el proceso

4. **Árboles Balanceados**

   Diseñar un algoritmo que recorra un a.b.b calculando el factor de equilibrio de todos sus nodos (hacer las modificaciones oportunas en tipoNodo) y devuelva verdadero o falso indicando si el árbol está balanceado o no.

Normas generales

En la realización de las prácticas se deben seguir los siguientes criterios:

1. Es **obligatorio** utilizar los tipos y prototipos siempre que se proporcionan en el fichero cabecera. Si no se proporcionan debe crearse el fichero cabecera correspondiente o añadir al proporcionado los prototipos de todas las funciones que se implementen.
2. La codificación de las funciones se realizará en un fichero fuente diferente al del programa principal.
3. Para visualizar el correcto funcionamiento de las operaciones implementadas se debe crear un fichero de prueba desde el cual se llamará a las funciones implementadas.
4. Se debe crear el correspondiente Makefile para la correcta creación del fichero ejecutable.