.UBAfiuba
FACULTAD DE INGENIERÍA

TDA: Árbol Binario de Búsqueda

[7541/9515] Algoritmos y Programación II
Segundo cuatrimestre de 2021

Alumno:	de la Rosa, Agustín
Número de padrón:	102875
Email:	adelarosa@fi.uba.ar

Índice

1. Introducción

Se pide implementar una Arbol Binario de B´usqueda (ABB) en el lenguaje C y el uso e iteradores internos, a cada función variando el recorrido y poder cargar a un vector según un recorrido, los elementos conseguidos.

Aclaraciones de la implementación. Se toma de referencia el predecesor como nodo a reemplazar de el que se va a borrar (el mayor de los ménores)

2. Teoría

Respuestas a las preguntas teóricas

1. Explique qué es un árbol, árbol binario y árbol binario de búsqueda. Explique por qué es importante la distinción.

   Árbol binario: árbol donde cada nodo tiene hasta dos hojas.

   Árbol binario de busqueda: árbol donde cada nodo no es que tiene solo dos hojas, sino que sigue una regla donde el menor va hacia la izquierda y el mayor hacía la derecha. Se aplica una restricción y los nodos son comparables, para poder así reducir la complejidad algorítmica al realizar una búsqueda.

2. Explicación de la solución implementada.

   La solución implementada es la recursiva, excepto en la función buscar que consideré más legíble la versión iterativa.

3. Detalles de implementación

Para implementar el abb usé dos principales fuentes de testing, la carpeta abb interactivo que pasó Lucas y luego usé una modificación con mispropiasa pruebas del ejemplo.c.

3.1. Insertar

Función recurisva que recibe por parametro la estrúctura árbol y un elemento para ordenar. a través del comparador que también pasó usuario cuando se creo el abb, se busca que el nodo este en la posición correcta, se le reserva memoria al struct y las referencias recursivas se encargan de .^enganchar"los nodos.

3.2. Buscar

Función iterativa que, mientras haya un nodo, se va a actualizar nodo hasta que el comparador resulte 0, una vez el elemento encontrado no es más que devolverlo y terminar con la función.

3.3. Borrar

Para borrar el trabajo fue un poco más complicado, primero hay que analizar los casos que nos pueden aparecer.

Primero, un nodo sin hijos, el cual es solo liberar la memoria que está reservada y retornar NULL así la referencia de su anterior (stackeada recursivamente) apunté hacía NULL.

Segundo caso, el nodo tiene un hijo, pero esto a mí no me molesta, porque la forma en la que uso los condicionales, es que dependiendo que hijo sea NULL, devolvería el otro ya que asigno a temp su hijo, por lo tanto con o sin hijo, el procedimiento recursivo es el mismo. Solo me interesa que lado es el hijo no si tiene 1 o ningúno.

Ahora si, el caso con 2 hijos que ya es diferente al resto, pero no tanto. Puedo "forzar"que sean iguales, simplemente buscando el mayor de los menores, una vez tengo su valor, reemplazo

el puntero a el que está apuntando el nodo con dos hijos a ese elemento, y le doy ese elemento al nodo con o sin hijos que es el mayor de los menores. Entonces así logre cambiar la situación a una que ya tenía que es nodo con menos de dos hijos.

Para devolver el voidptr que se pide, me encargo de pasar un puntero a void y lo devuelvo.

4. Diagramas

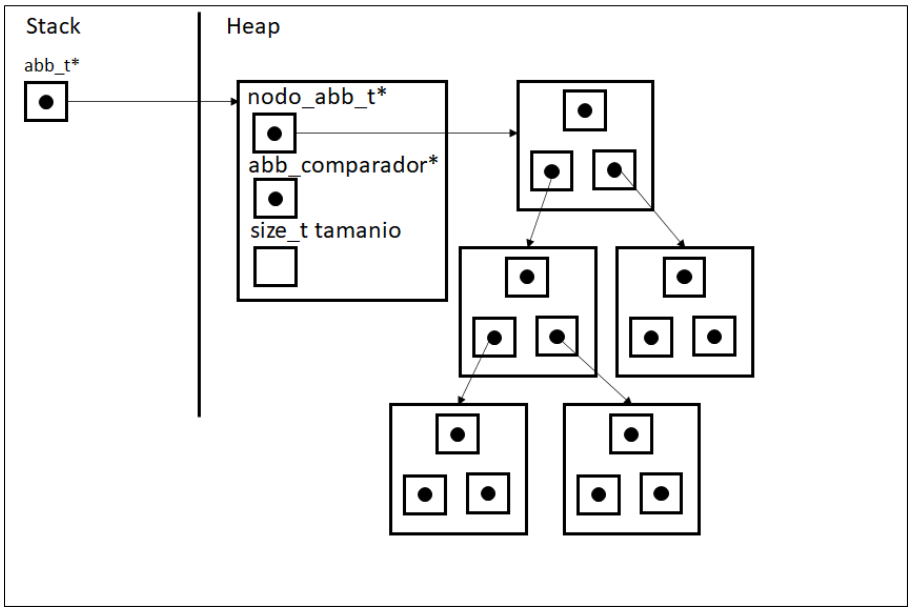

Figura 1: Ejemplo de la estructura empleaa para el arbol, cabve resaltar que los nodos con elemento menor van hacia la izquierda y los mayores hacia la derecha. Luego no marqué que el comparador es una función que nos pasa el usuario.

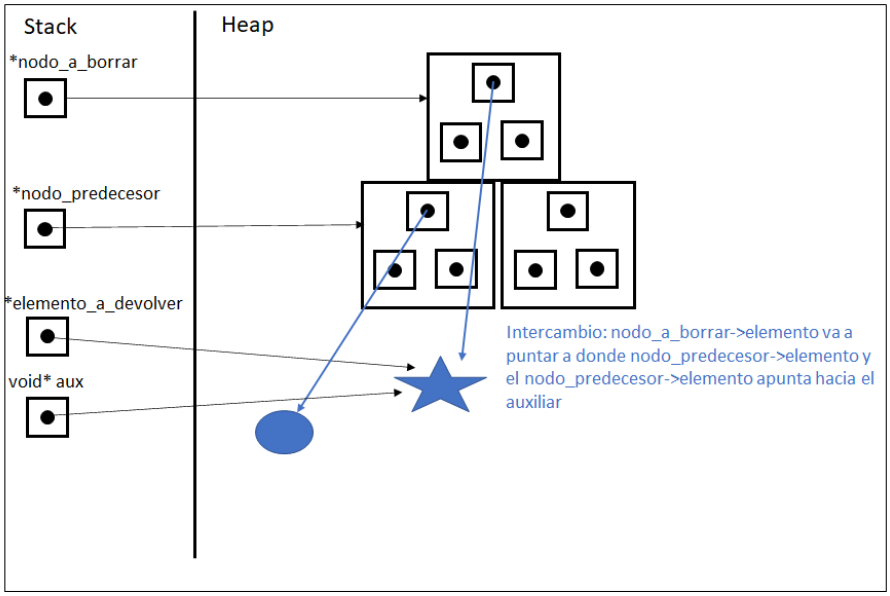

Figura 2: Ejemplo de la función borrado.