TDA ABB

[7541/9515] Algoritmos y Programación II
Primer cuatrimestre de 2022

Alumno:	PELLEGRINO, Dafne
Número de padrón:	104368
Email:	dpellegrino@fi.uba.ar

Índice

1. Introducción

Se pide implementar una Árbol Binario de Búsqueda (ABB) en el lenguaje de programación C. Para ello se brindan las firmas de las funciones publicas a implementar y se deja a criterio del alumno la creación de las funciones privadas del TDA para el correcto funcionamiento del ABB cumpliendo con las buenas practicas de programación. Adicionalmente se pide la creación de un iterador interno que sea capaz de realizar diferentes recorridos en el árbol y una funcione que guarda la información almacenada en el árbol en un vector. El TDA entregado deberá compilar y pasar las pruebas dispuestas por la cátedra sin errores, adicionalmente estas pruebas deberán ser ejecutadas sin pérdida de memoria.

2. Teoría

Un árbol es una estructura formada por nodos. Los nodos son los .Elementos"del árbol. Existe un nodo que se denomina Raíz, que es el inicial del árbol y de donde se puede decir que çuelgan"los sub-arboles que componen un árbol. Suele llamárseles a los nodos que tienen nodos 'colgando' de si, padres, y a sus inferiores, hijos. Los nodos que no tienen hijos, adicionalmente, suelen ser denominados 'nodos hoja', siguiendo con los términos referentes a un árbol, los nodos hoja son los que se encuentran más abajo en sus ramas.

Para los árboles generales, las operaciones de creación, verificar si está vacío y ver su tamaño (con una implementación similar a la que se utiliza en este trabajo) seria O(1). Luego para destruir, insertar, eliminar, buscar, es muy fácil que ocurra el peor de los casos, es decir O(n) si no están ordenados en base a algún parámetro y son facilmente recorribles, como los arboles binarios.

Un árbol binario es un tipo particular de árbol cuyos nodos tienen como máximo 2 hijos (o 2 elementos colgando de sus nodos). La estructura sería gráficamente algo como lo siguiente:

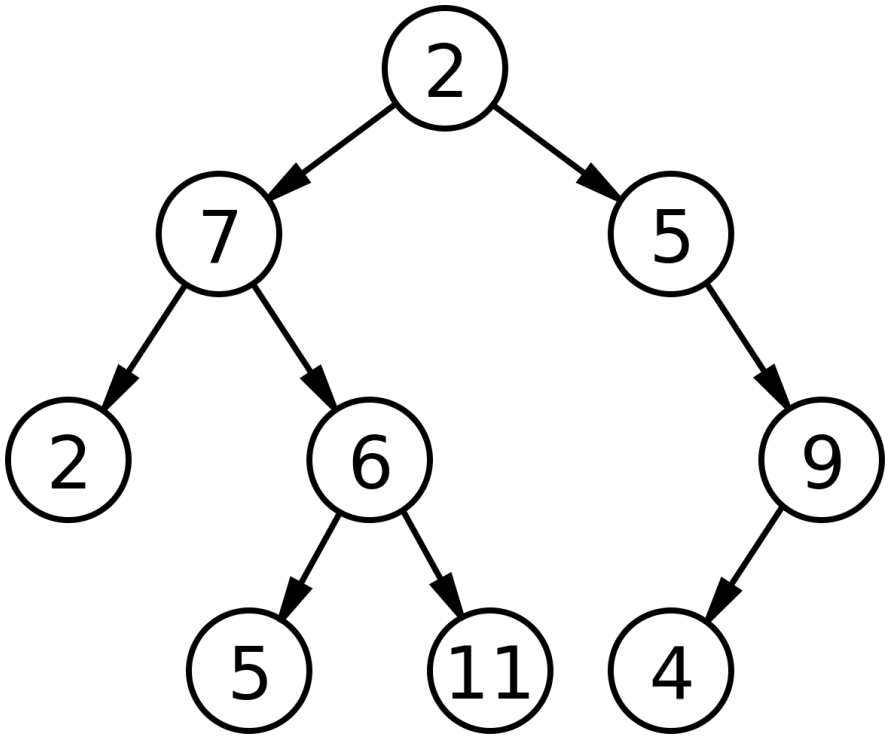

Figura 1: Diagrama 1: Estructura Árbol Binario

Los árboles pueden o no ser binarios, de hecho los arboles binarios solo son un caso particular

de los arboles n-arios. El árbol binario tiene un beneficio en cuanto a la simplicidad para recorrerlo, ya que las ramas que se van bifurcando pueden asociarse con las direcciones izquierda y derecha, para una más fácil identificación.

Particularmente dentro de los arboles binarios encontramos al árbol binario de búsqueda (abb). Dicho árbol es un Tipo de dato abstracto definido que permite realizar búsquedas mediante el concepto de que los elementos pueden estar a la derecha o izquierda de su elemento raíz o padre. Generalmente se implementan teniendo en cuenta algún criterio de comparación con los elementos a insertar y ubicando los elementos mas 'grandes' hacia la derecha y los mas 'chicos' hacia la izquierda.

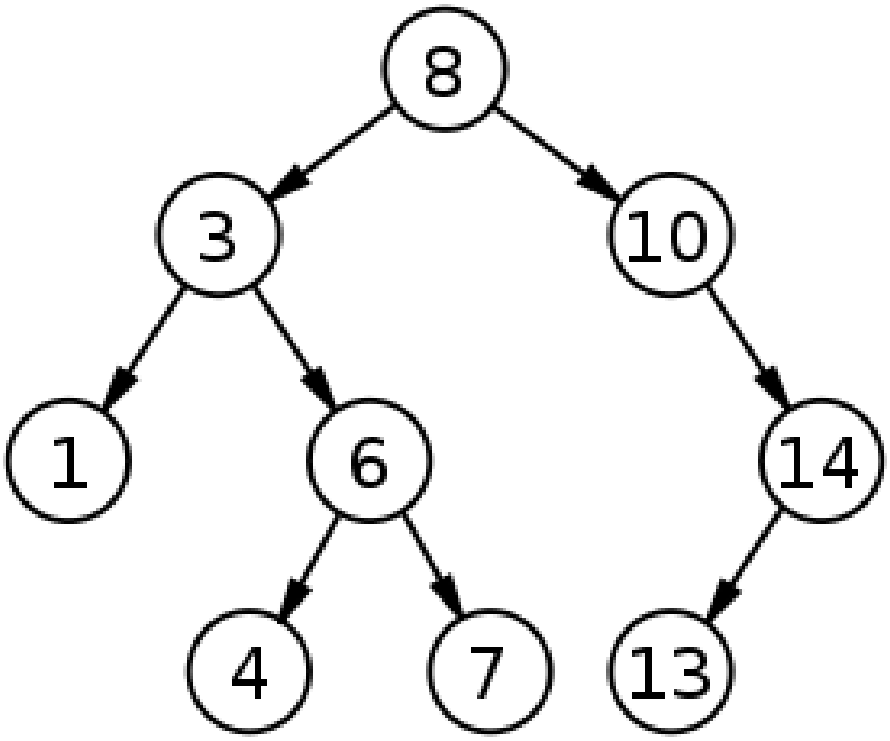

Figura 2: Diagrama 2: Estructura ÁBB

Es importante tener en cuenta que todos los arboles binarios de búsqueda son arboles binarios, pero no todos los arboles binarios son de búsqueda. De la misma forma pueden existir árboles ternarios, cuaternarios y hasta n-arios que cumplen características singulares para cada caso definido además de la premisa de cantidad de hijos máximos por nodo.

Para los árboles binarios, la complejidad media en las operaciones de búsqueda, inserción y eliminación es bastante optimista, siempre que no degenere a lista, se considera O(log(n)). Para el recorrido, ya que hay que recorrer todos los elementos, la complejidad sería O(n)

3. Detalles de implementación

Para compilar el programa se puede utilizar el atajo de makefile "make pruebasz luego para ejecutar el programa corriéndolo en valgrind para chequeo de memoria puede sencillamente ejecutar el comando definido en el makefile con "make valgrind-pruebas".

Para la implementación de abb el tipo de dato está definido desde el enunciado. El abb almacena un puntero al nodo raiz, una cantidad de elementos y un comparador de datos que vamos a ingresarle.

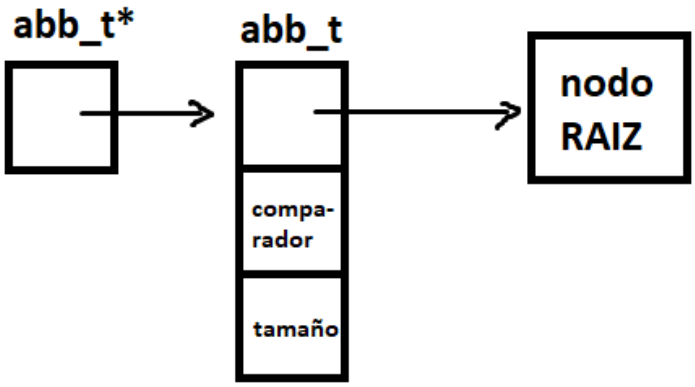

Figura 3: Diagrama 3: Estructura abb_t

Cada nodo almacena tres punteros, uno void hacia el elemento correspondiente y luego dos hacia otros nodos, derecha o izquierda.

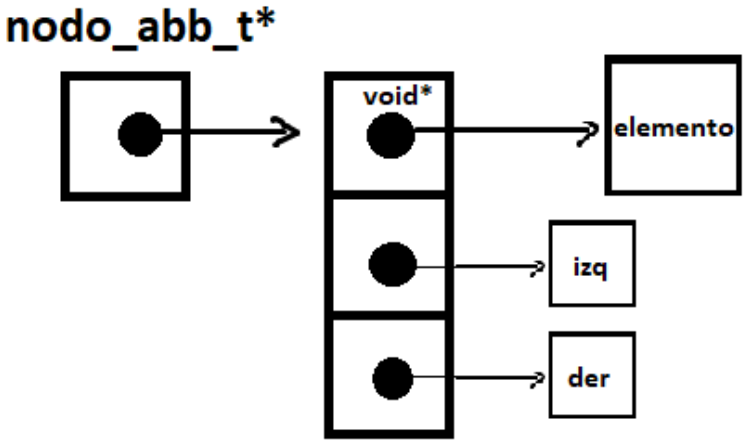

Figura 4: Diagrama 4: Estructura Nodo

Al insertar un elemento en el abb es importante ir comparando para insertarlo en la posición correcta, eso se logra almacenando en la estructura del abb una función que realice la comparación. Los elementos menores se ubican a la izquierda y los mayores a la derecha del elemento con el cual se está comparando. Esta implementación admite repetidos, los cuales se colocan a la izquierda.

Al quitar un elemento se busca que la estructura no pierda su orden por lo que en el caso de ser elementos con solo un hijo, este se mueve a la posición del padre. En el caso de ser un nodo con dos hijos buscamos el elemento mas grande dentro de la rama de los pequeños, también llamado predecesor inorden, y lo colocamos en la posición del nodo eliminado.

Para recorrerlo podemos usar 3 tipos de recorrido: inorden, preorden y postorden. Cada uno de ellos se caracteriza por determinar el orden de recorrido del árbol y entonces de los nodos izquierdo, derecho y raíz en diferentes ordenes. Inorden recorre el árbol primero por el nodo o subárbol izquierdo, luego la raíz y luego el nodo o subárbol derecho. Postorden recorre el árbol primero por el nodo o subárbol izquierdo, luego el derecho y luego la raíz. Preorden recorre el árbol primero por la raíz, luego la izquierda y luego a la derecha.

Es sencillo buscar elementos en este tipo de estructura ya que está diseñada para facilitar este proceso. Para buscar un elemento solo hace falta tener el dato mediante el cual se compara (depende de la función de comparación que facilita el usuario) y luego ir descendiendo hacia la derecha o izquierda dependiendo del resultado de dicha comparación.

Para destruir el abb puedo optar por destruir también los elementos que insertamos o solo los nodos. Lo primero se hace enviando una función que destruya lo que insertamos.

3.1. Detalle crear e insertar

Para crear se reserva memoria para un abb (tipo de dato abb_ t) y se inicializa con el comparador que se recibe. No se puede crear un abb si se recibe un comparador nulo. La función de reserva de memoria calloc se encarga de inicializar el tamaño en 0 y el nodo raíz en NULL.

Para insertar el elemento solamente se llama a la función insertar recursiva, dentro de la función de inserción, con los datos necesarios, el nodo raíz, el elemento a insertar y el comparador. Elegí la forma recursiva, ya que facilita el recorrer la mitad del árbol en la que se insertará el elemento, descartando la otra mitad, de una forma sencilla.

La función insertar recursivamente no puede insertar si se recibe un comparador nulo. Si recibe un nodo nulo significa que llego a la posición de inserción y reserva la memoria correspondiente y coloca el elemento. Si el nodo recibido es válido, en cambio, se realiza la comparación para saber para que lado habrá que insertar el elemento y se vuelve a llamar a esta función.

Al salir de la función recursiva se actualiza el dato de la cantidad de elementos.

3.2. Detalle de quitar

La función quitar chequea primeramente si el elemento a eliminar es la raíz, en ese caso lo elimina y según si tiene 2, 1 o 0 hijos realiza el reemplazo correspondiente. En el caso de 2 hijos busca su predecesor inorden para reemplazarlo. Para esto último y para facilitar la lectura y organización del código, llamo a una función que se encarga de eso. A la función que busca el predecesor inorden se le envía el nodo a la izquierda del que estamos parados y luego itera buscando el ultimo nodo derecho que haya, lo elimina y lo devuelve.

En caso que la comparación determine que es menor a la raíz llama a la función quitar recursiva con el nodo izquierdo, caso contrario con el derecho. La función recursiva realiza los mismos procedimientos que la anterior, con la salvedad de asignar cualquier nodo como raíz, ya que no tiene acceso al tda abb_t.

3.3. Detalle de recorrer y abb con cada elemento

Las funciones de recorrer y abb con cada elemento realizan un case switch con el tipo de recorrido recibido para identificar a que función deben llamar. La diferencia principal entre abb recorrer y la otra función es que abb con cada elemento recibe una función que debe ser aplicada a los elementos en el orden correspondiente con el tipo de recorrido recibido. Abb recorrer copia los elementos a un vector que se le debe indicar, en el orden correspondiente con el tipo de recorrido recibido.

Ambas funciones delegan la tarea a una función que recorre el árbol de la forma correcta, de forma recursiva, ya que eso facilita la lectura del código y muestra claramente cuando se opera con el nodo preciso y cuando se prosigue a izquierda o derecha.

Los tipos de recorrido posibles son los previamente expuestos.

3.4. Detalle de destruir

La función de destrucción de abb recibe un abb y libera la memoria reservada para los nodos existentes y para el abb en sí mediante la llamada a la función de destrucción recursiva. Adicionalmente con la función destruir todo y facilitando un destructor de los elementos también podemos liberar esa memoria