

TDA 2 — ABB

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

Alumno:	Castillo, Carlos E.
Número de padrón:	108535
Email:	ccastillo@fi.uba.ar

Índice

1. Introducción	2
2. Teoría	2
2.1. TDA Árbol	2
2.1.1. Árbol Binario de Búsqueda (ABB)	2
3. Detalles de implementación	3
3.1. Inserción de elementos	3
3.2. Eliminación de elementos	4
3.3. Búsqueda de un elemento	4
3.4. Iterador	4
3.5. Getters	4
3.6. Destrucción del ABB	4
4. Diagramas	4

1. Introducción

Muchos lenguajes de programación permiten al usuario extender los tipos de datos nativos del lenguaje con para facilitar la implementación de características y funcionalidades más complejas. De esta forma, el programador tiene la capacidad de combinar diferentes primitivas del lenguaje y así generar nuevos tipos compuestos para crear interfaces abstractas que faciliten el manejo y organización de la información mediante diferentes estructuras de datos.

El objetivo de este trabajo práctico es implementar una de las estructuras de datos más populares, el árbol binario de búsqueda.

2. Teoría

2.1. TDA Árbol

Un árbol es una colección de datos en la que la información mantiene una estructura jerárquica. Esta estructura puede ser definida de manera recursiva utilizando nodos enlazados, en la que cada nodo es una estructura que contiene el valor almacenado y referencias a los nodos siguientes, conocidos como **hijos**. Cada árbol tienen un nodo en particular que es inicio o el punto de entrada al árbol, es el único nodo sin padres, y se le conoce como **raíz**. Por otra parte, a aquellos nodos que no tienen ningún hijo, se les conoce como **hojas**.

Existen muchas variantes de esta estructura de datos, cada una con diferentes propiedades y utilidades, por ejemplo:

- Árbol Binario
- Árbol Binario de búsqueda
- Árbol AVL
- Árbol B
- Árbol Rojo Negro

Sin embargo la más popular entre estas implementaciones es el **árbol binario**, que consiste en un árbol cuyos nodos pueden tener un máximo de dos hijos cada uno (esto no impide que un nodo pueda estar vacío o tener solo un solo hijo). Para facilitar la diferenciación entre los hijos de un nodo, se les suele llamar "nodo izquierdo" y "nodo derecho" respectivamente, en función de la ubicación relativa de cada uno de los nodos con su nodo padre.

2.1.1. Árbol Binario de Búsqueda (ABB)

Este tipo de árbol es un caso particular de un árbol binario. Como su nombre bien lo indica, es utilizado principalmente para operaciones de búsqueda entre los datos almacenados. Consiste en un tipo de árbol binario en el que se tiene un criterio de comparación definido para ser utilizado al momento de insertar nuevos nodos al árbol. Esto permite realizar búsquedas de manera más sencilla, ya que al contar con este patrón de ordenamiento, se pueden realizar operaciones de búsqueda binaria, recortando el tiempo de búsqueda en comparación con las listas enlazadas u otras estructuras de datos lineales.

Generalmente se plantea un método de comparación que permite distinguir entre valores considerados menores o mayores al nodo actual, así por ejemplo, si se le asigna la posición izquierda a los elementos cuyo valor es menor al valor del elemento del nodo actual, entonces se sabe que todos los nodos a la izquierda de cualquier nodo van a tener un valor menor al de sus respectivos padres, y todos los nodos la derecha, tendrán un valor mayor al de sus padres.

Esto facilita la navegación o **recorrido** del árbol, ya que al saber si el elemento buscado es menor o mayor que el nodo actual, se puede determinar en qué dirección seguir el recorrido y descartar todas aquellas ramas que no satisfagan la comparación.

3. Detalles de implementación

La implementación de esta estructura de datos fue escrita en el lenguaje de programación C, siguiendo la especificación del lenguaje detallada en el estándar C99. Los archivos principales se encuentran dentro del directorio **src** ubicada en la raíz del repositorio.

Además se incluye un archivo **pruebas.c** en el que se encuentran diferentes pruebas automatizadas para detectar errores en la ejecución del programa o en la asignación, manejo y liberación de memoria dinámica. Para esto se utiliza **gcc** como compilador y **valgrind** como herramienta de análisis de memoria.

La estructura de árbol binario de búsqueda propuesta para esta implementación es la siguiente:

```
typedef struct nodo_abb {  
    void* elemento;  
    struct nodo_abb* izquierda;  
    struct nodo_abb* derecha;  
} nodo_abb_t;
```

```
typedef struct abb{  
    nodo_abb_t* nodo_raiz;  
    abb_comparador comparador;  
    size_t tamanio;  
} abb_t;
```

Esta estructura no es meramente recursiva, ya que además de contener referencias a otros nodos (en este caso únicamente a la raíz del árbol), almacena el tamaño actual del árbol (la cantidad de nodos que contiene) y una función comparador que establece el criterio de comparación utilizado para el árbol.

Por otra parte, la estructura de cada uno de los nodos si es recursiva. Cada nodo puede ser considerado como un sub-árbol, y almacena el elemento a guardar así como también referencias a sus nodos hijos derecho e izquierdo.

3.1. Inserción de elementos

La función `abb_insertar` permite agregar nuevos elementos a un ABB. En el caso de que el ABB esté vacío, cuando se inserte el primer elemento, este pasará a ubicarse en el nodo raíz. En este caso la posición en la que se va a insertar el nodo que contiene al nuevo nodo se define a partir del criterio de comparación propio de cada árbol.

La operación de inserción consiste en una serie de 3 pasos:

1. Comparar el valor del elemento a insertar con el valor del nodo raíz. En base al resultado de esta comparación se sabrá si navegar hacia la rama izquierda del árbol (en el caso de que el valor comparado sea menor al de la raíz) o a la derecha (en caso de que el valor comparado se mayor).
2. Se repite el paso anterior con cada uno de los nodos que se vayan recorriendo hasta llegar al final de la rama.
3. Al llegar al nodo final se crea un nuevo nodo que contiene el elemento a insertar y se posiciona en la ubicación correspondiente (izquierda o derecha) de acuerdo con el resultado de la comparación anterior.

En este caso la implementación de la operación de inserción es recursiva. Para esto la función `abb_insertar` antes mencionada, utiliza una función auxiliar llamada `abb_insertar_recursivo_aux`. En la primera función solamente se hace la validación de los argumentos y se verifica que el nuevo elemento haya sido insertado con éxito. En caso afirmativo, se incrementa la cantidad de elementos del árbol y finalmente se retorna el nuevo árbol.

3.2. Eliminación de elementos

Para quitar elementos del ABB primero se tiene que realizar un recorrido del mismo para buscar el elemento que se desea eliminar. En el caso en que el elemento que se desea eliminar no esté en el árbol, simplemente se termina la búsqueda. En caso contrario, la operación de eliminación puede ser dividida en tres casos particulares:

- **Nodo hoja:** En el caso en el que el elemento que se quiere eliminar esté ubicado en un nodo hoja, el procedimiento para remover este elemento simplemente consiste en extraer el elemento que contiene ese nodo (para no perder la referencia al elemento) y después la liberación del nodo de la memoria.
- **Nodo con un hijo:** En este caso se tiene que identificar si nodo actual tiene hijo izquierdo o derecho. Luego se enlaza el hijo identificado con el padre del nodo a eliminar, de tal forma que el árbol conserve sus propiedades de ABB. Finalmente se extra el valor almacenado en el nodo a eliminar y se procede a la liberación del nodo de la memoria.
- **Nodo con dos hijos:** En este último caso se tiene que encontrar el antecesor o sucesor inmediato al nodo eliminar, es decir, para los nodos cuyo valor es menor al valor del nodo a eliminar, extraer el nodo más a la derecha de estos mismos, es decir, el nodo con mayor valor, en el caso de el sucesor inmediato es el caso contrario, se extraer el nodo con menor valor entre todos los nodos que están a la derecha del nodo a eliminar, es decir, entre los nodos cuyo valor es mayor al nodo a eliminar.

3.3. Búsqueda de un elemento

3.4. Iterador

3.5. Getters

3.6. Destrucción del ABB

4. Diagramas

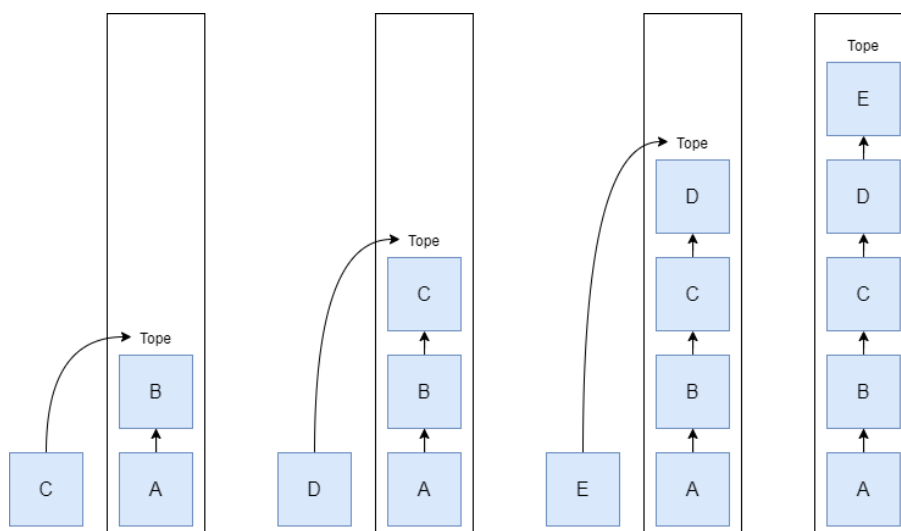


Figura 1: Apilado en pila.

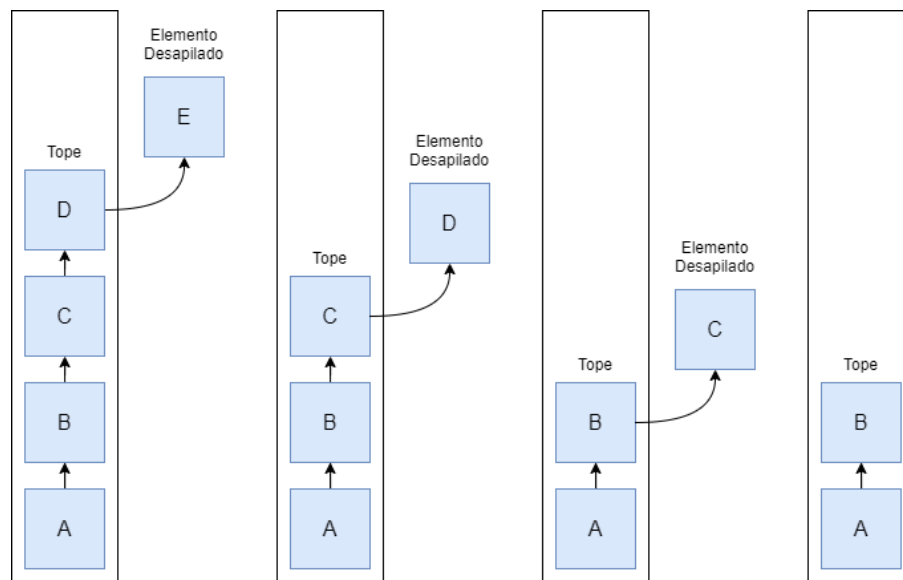


Figura 2: Desapilado en pila.

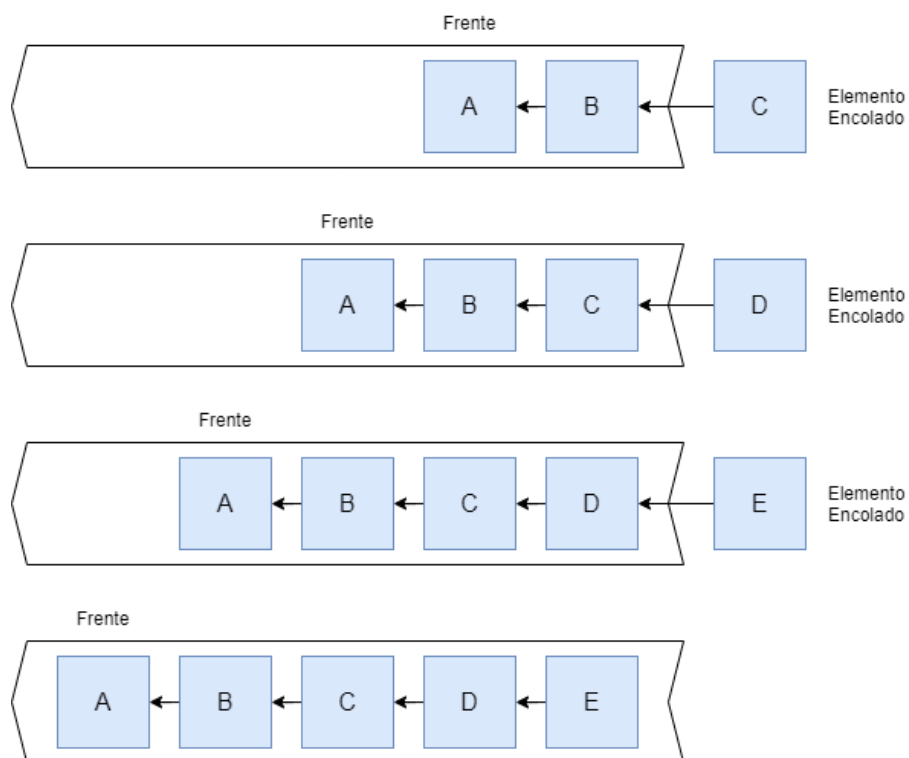


Figura 3: Encolado en cola.

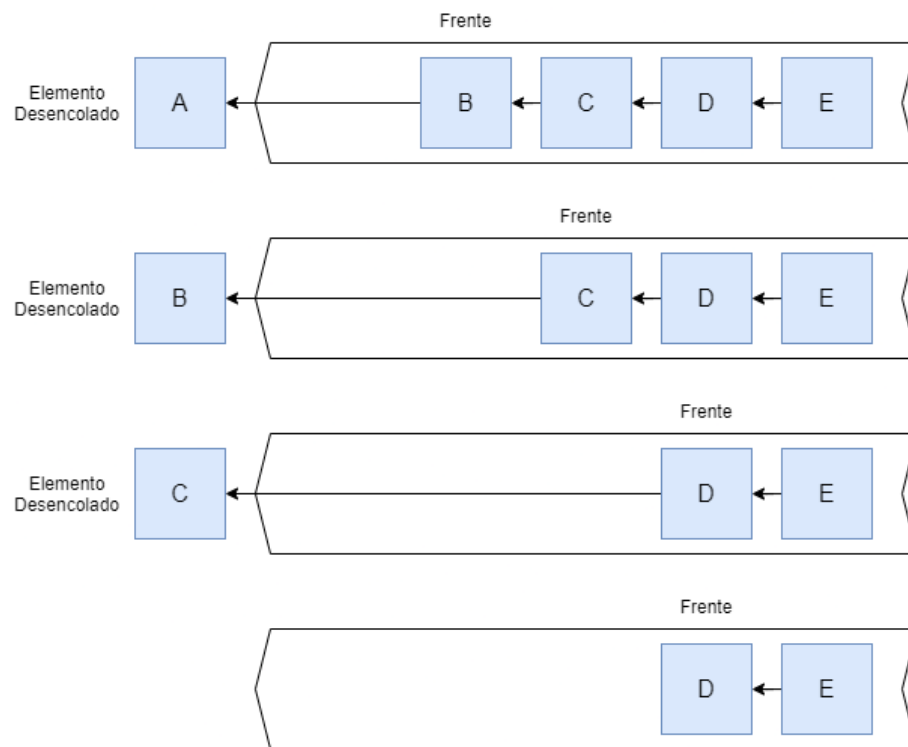


Figura 4: Desencolado en cola.

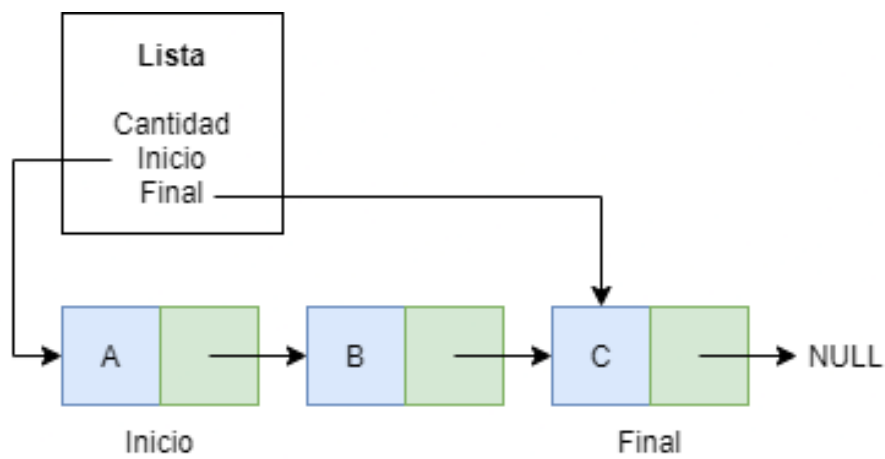


Figura 5: Implementación de lista enlazada.

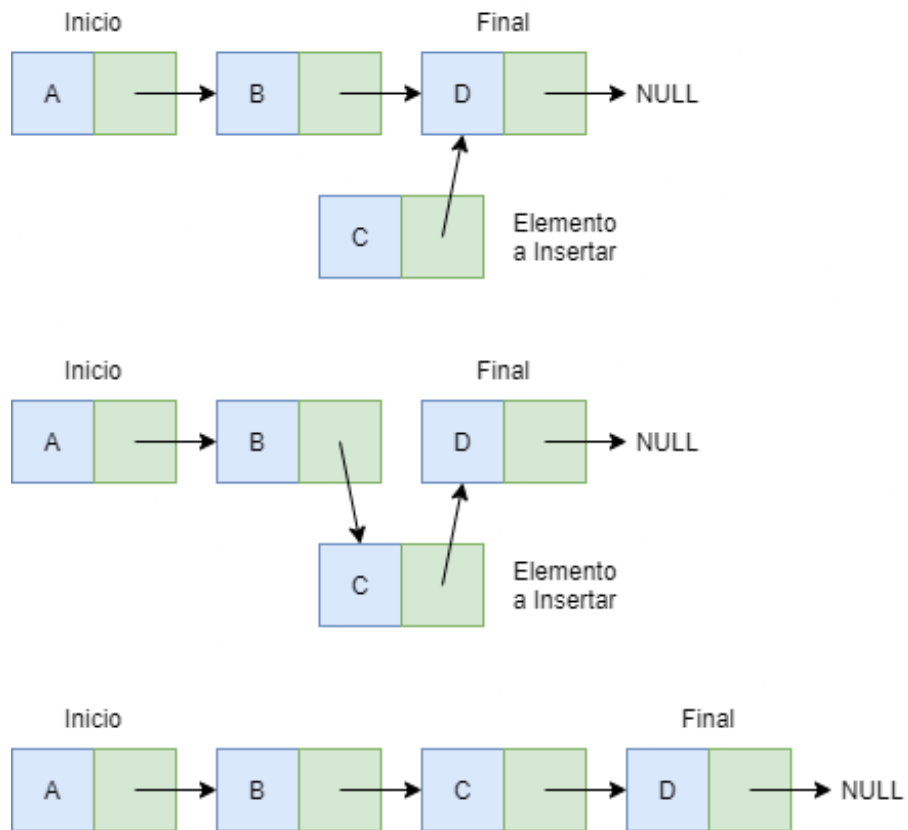


Figura 6: Insertado en lista.

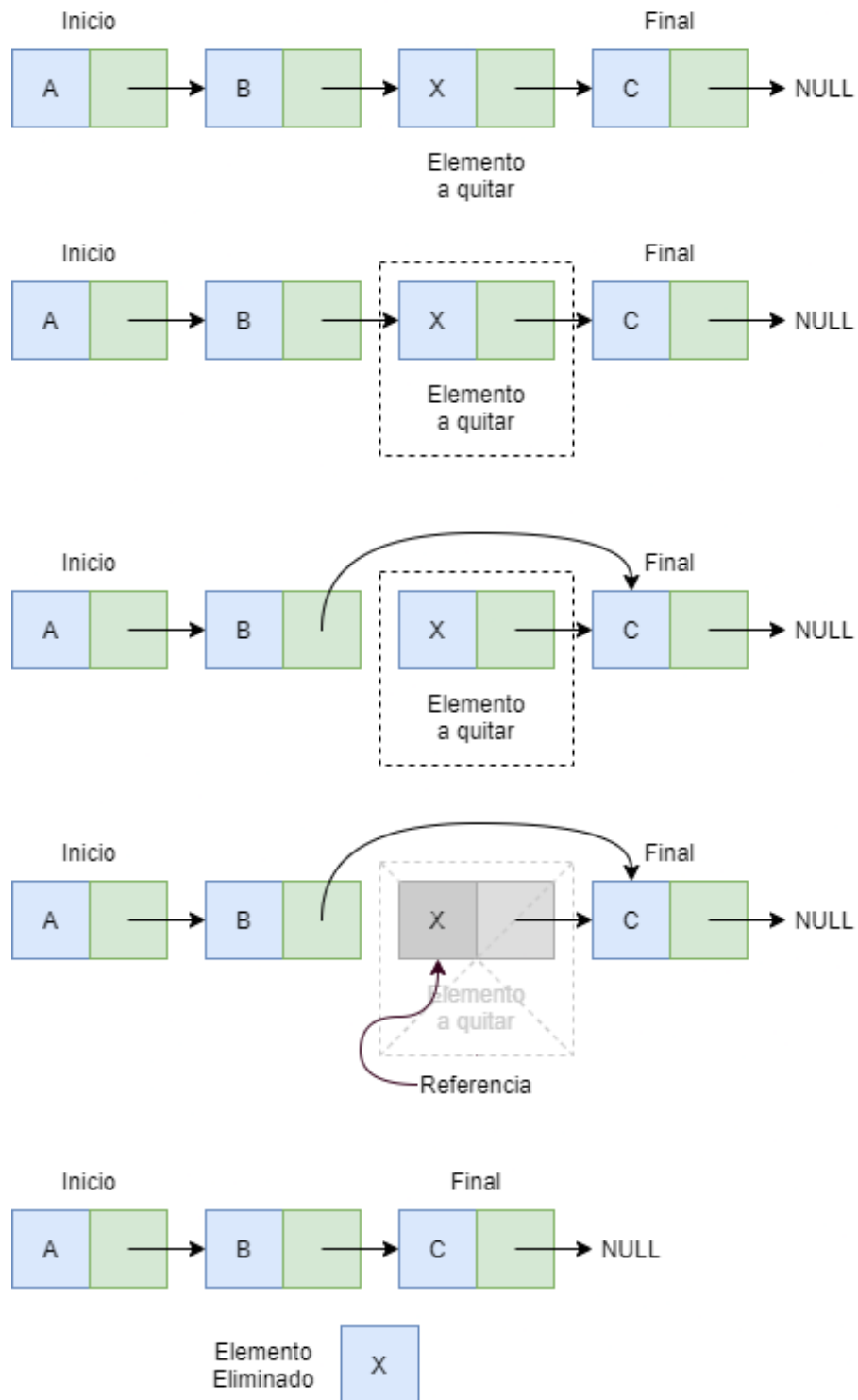


Figura 7: Eliminación en lista.