TDA ABB

[7541/9515] Algoritmos y Programación II

Primer cuatrimestre de 2022

Alumno:	Subero, Andrés
Número de padrón:	109114
Email:	asubero@fi.uba.ar

1. Introducción

El trabajo práctico tiene como finalidad poner en práctica el Tipo de Dato Abstracto ABB y sus primitivas. Para poder realizar las conexiones entre los distintos bloques, la estructura árbol cuenta con un puntero al nodo raíz, un entero para saber el número de elementos o nodos que contiene el árbol. La estructura nodo cuenta con un puntero a cada hijo y un puntero al elemento que contienen.

Para recorrer el árbol se tienen tres formas, INORDEN, PREORDEN Y POSTORDEN. El recorrido INORDEN es visitar hijo izquierdo, nodo actual y luego hijo derecho; el PREORDEN es visitar nodo actual primero, luego el izquierdo y por último el derecho; finalmente el recorrido POSTORDEN es visitar hijo izquierdo, hijo derecho y por último nodo actual.

Para la implementación se tomaron algunas consideraciones, a la hora de eliminar un nodo, se reemplaza con el predecesor inorden. A la hora de insertar, si el elemento a insertar es igual al elemento actual, se toma como si fuese menor.

Para confirmar la correcta implementación del TDA, se realizaron pruebas unitarias, las cuales simulan la mayor cantidad de casos posibles (en este caso son muy importante los casos bordes) y con cada prueba se asegura que ese caso está cubierto y no fallará en el futuro.Para la realización de las pruebas, se intentó realizar primero la prueba y luego implementar la mínima solución (TDD).

2. Teoría

2.1) ÁRBOL

Un árbol es una colección de nodos y se caracteriza porque uno de esos nodos es el nodo raíz, el cual puede tener o no nodos hijos, en el caso de tenerlos se puede tener con cada uno un subárbol, entonces esos nodos hijos pasan a ser la raíz de cada subárbol. Esto facilita el manejo de grandes cantidades de datos ya que el tiempo de búsqueda en general es O(log n).

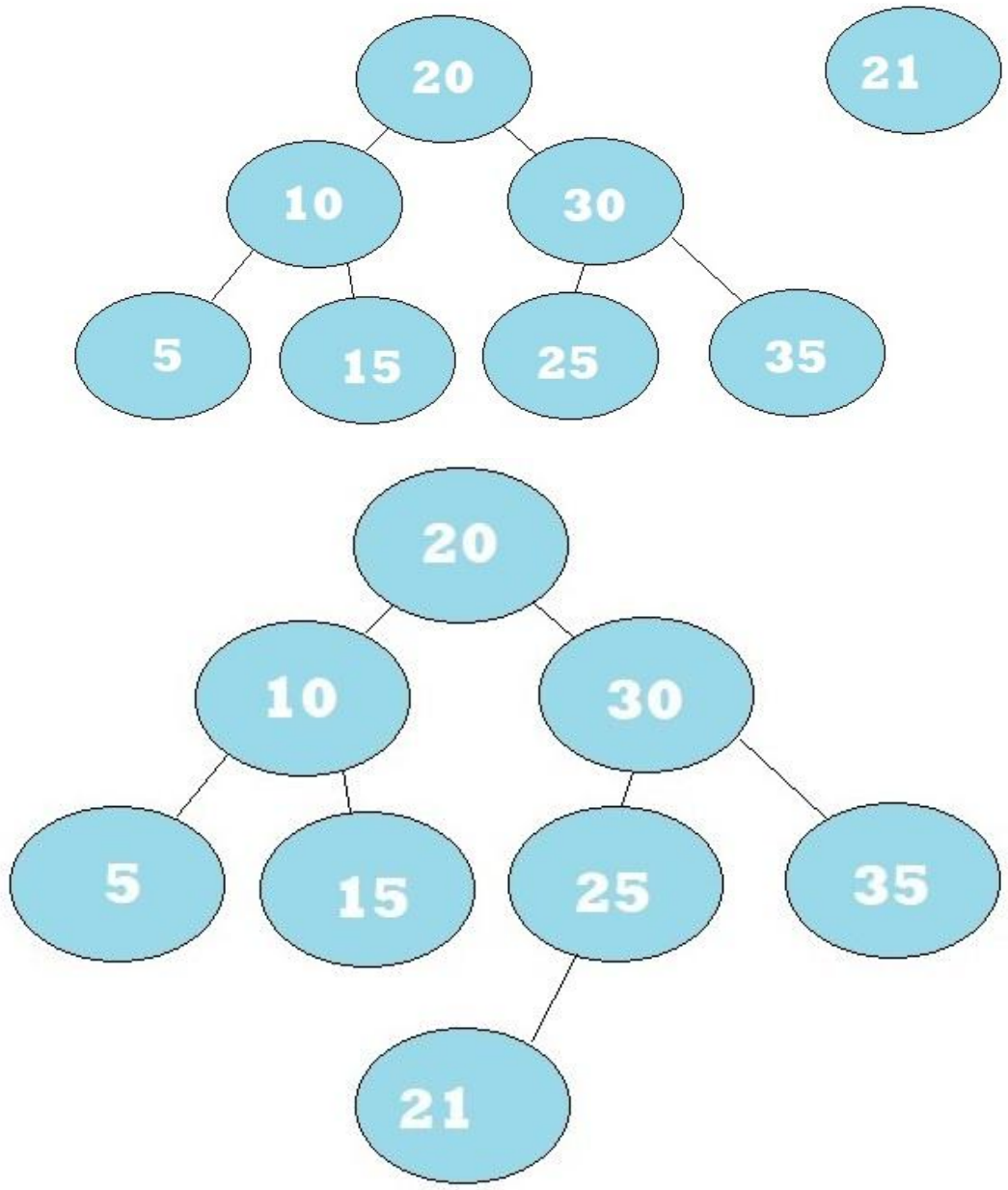
2.2) ÁRBOL BINARIO

Los árboles binarios son aquellos cuyos nodos o raíces pueden tener 0, 1 o 2 hijos. Con este tipo de árbol se tienen los conceptos de “izquierda” y “derecha”, cada nodo puede o no tener un hijo “izquierdo” y uno “derecho”. Este tipo de dato de por si no es muy útil ya que no se tiene una forma de comparar y saber que camino seguir de acuerdo a la comparación.

2.3) ÁRBOL BINARIO DE BÚSQUEDA

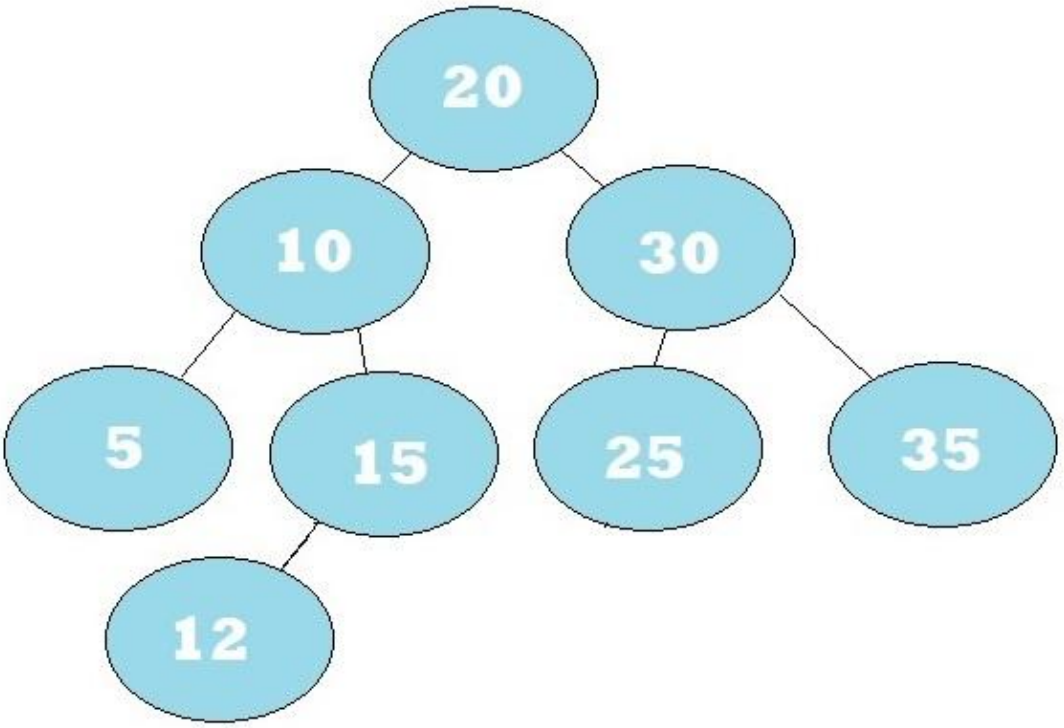
Teniendo el concepto de árbol binario, que introduce la noción de hijo izquierdo y derecho, se puede implementar una comparación entre los hijos y el padre, definiendo si uno es mayor, menor o igual que el otro. Los árboles binarios de búsqueda son árboles binarios que cumplen con lo siguiente: el hijo izquierdo es menor que el padre, el hijo derecho es mayor que el padre. Con menor o mayor se refiere a que si establecemos un parámetro para medir, se sabe que el subárbol tiene valores más grandes y en el subárbol izquierdo se tendrá valores menores. Con esto, se tiene la ventaja de la velocidad de una búsqueda binaria y la ventaja de la búsqueda logarítmica del árbol.

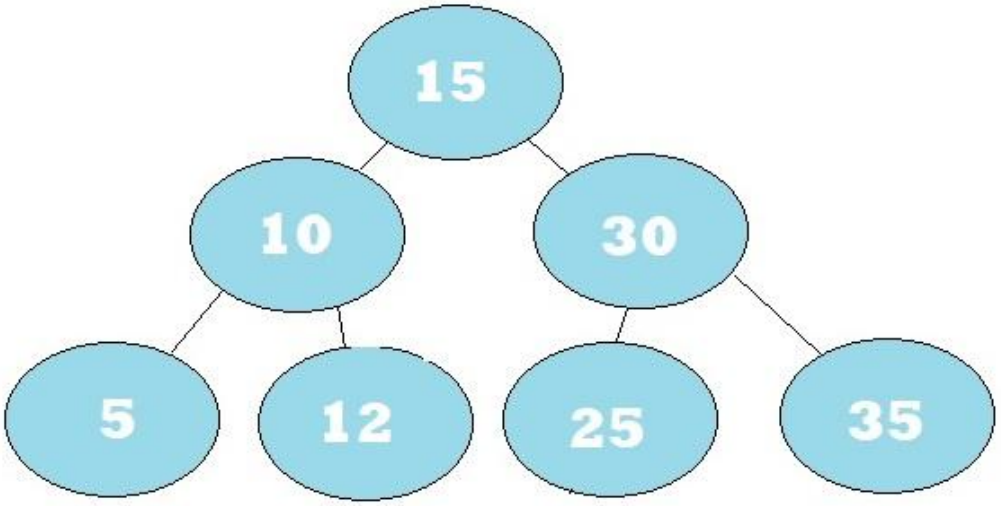
- *Inserción:* Al insertar un nodo en un árbol binario de búsqueda, se va realizando la comparación entre el elemento que se desea insertar con el elemento del nodo actual. Entonces comenzando con el nodo raíz, se compara, ¿ es menor ? se toma como camino para la siguiente comparación el subárbol izquierdo, ¿ es mayor ? se toma como camino el subárbol derecho. Cuando se encuentre que no hay nodo actual, es momento de insertar, ya que se llegó al “final” de la rama que se estuvo recorriendo. Ejemplo: insertar 21.



- *Eliminación:* Como se dijo anteriormente, se tomó como convención el reemplazar por el menor inorden a la hora de eliminar un nodo. Para la eliminación se realiza la comparación de la misma manera que en la inserción solo que ahora se detiene al encontrar el elemento a eliminar, se revisa si tiene hijos, en el caso de no tener hijos (nodos hojas) simplemente se libera la memoria del nodo.
  En el caso de tener un hijo, se apunta con un auxiliar, se libera la memoria del nodo que se va a eliminar y se retorna el auxiliar para conectar al hijo con el padre del nodo eliminado.
  En el caso de tener dos hijos, se busca el menor predecesor INORDEN. Para ello hay que tomar ciertas consideraciones. Primero se verifica si el hijo izquierdo tiene hijo derecho, en el caso de no tenerlo, se reemplaza el nodo a eliminar por su hijo izquierdo, y el hijo derecho del nodo a eliminar pasa a ser el hijo derecho del nodo que lo reemplaza.

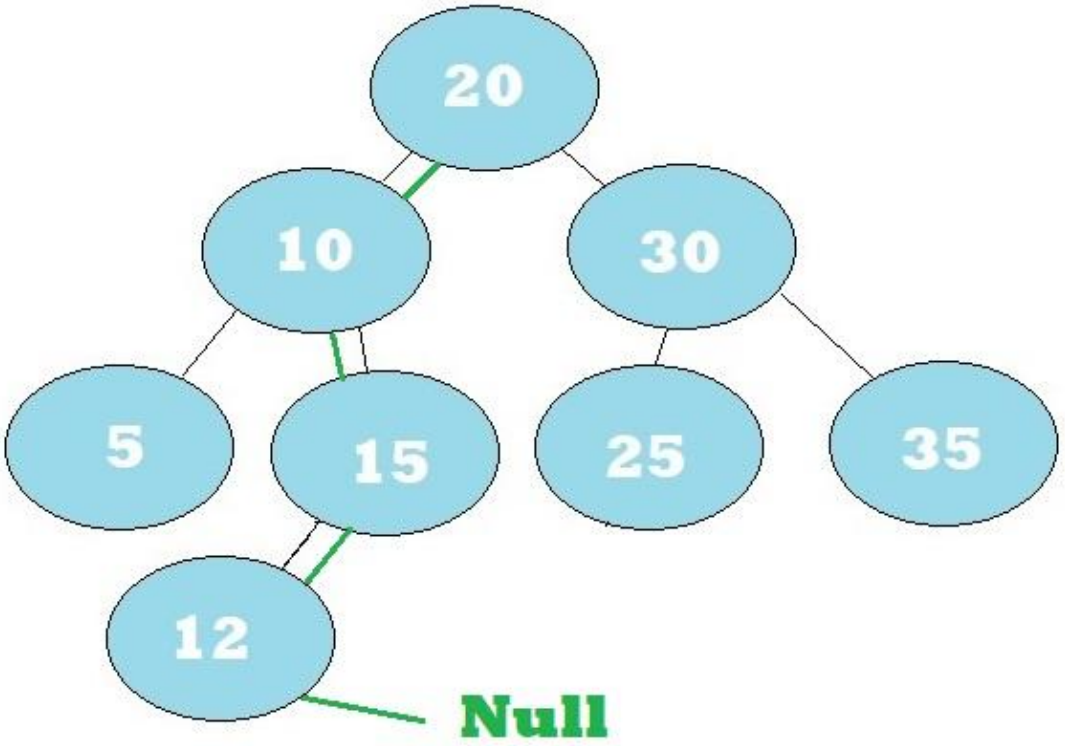
En el caso de que el hijo izquierdo tenga hijo derecho, se busca el último nodo siempre tomando el camino de la derecha para encontrar el menor más cercano. Al encontrarlo, lo apuntamos con un auxiliar (predecesor inorden) el hijo izquierdo del predecesor inorden es asignado como hijo derecho del padre del predecesor inorden. Luego al predecesor inorden se le asignan los hijos del nodo a eliminar, se libera la memoria del nodo a eliminar y se retorna el reemplazo para que sea asignado como hijo del nodo que llamó a la función eliminar. Ejemplo: eliminar el 20.



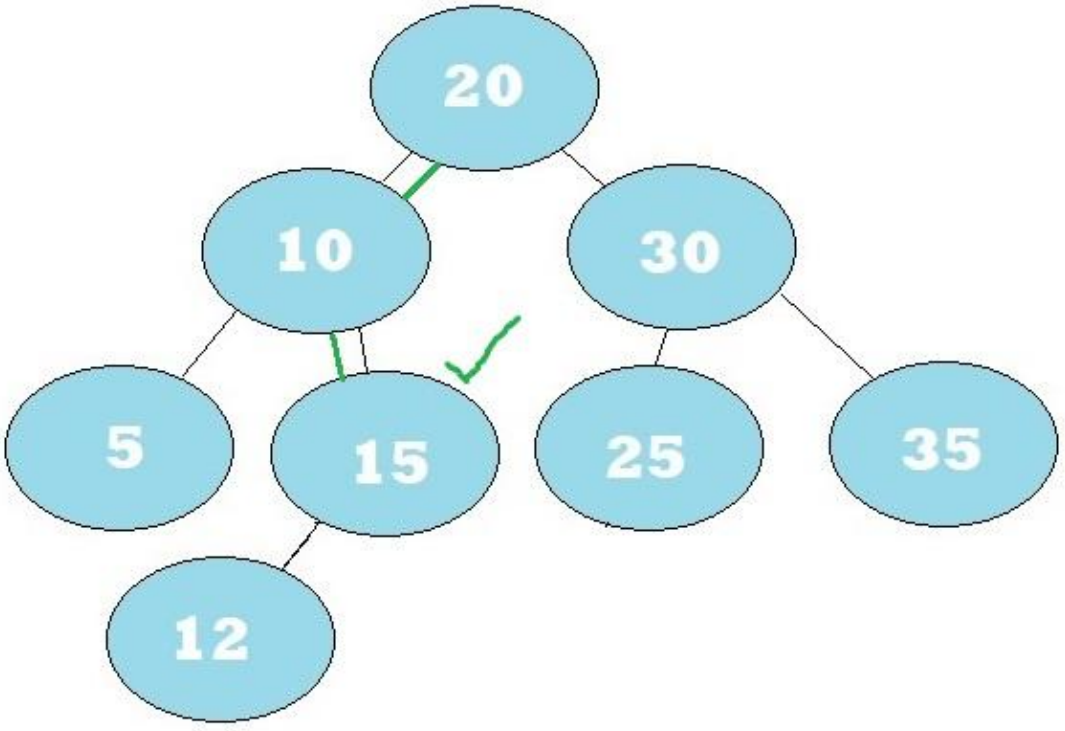


- Búsqueda: La búsqueda de un elemento es bastante sencilla, se realiza la comparación como en insertar o eliminar solo que al encontrar el elemento, se retorna.

Ejemplo: busca el número 13, no lo encuentra retorna NULL.



Ejemplo: busca el número 15.



- Con cada elemento: Se recorre el árbol según el orden que establezca el usuario. Con cada elemento aplica la función comparándolo con el elemento pasado como contexto. Si la función retorna false, entonces no se debe seguir recorriendo. Para detener el recorrido se tiene varios condicionales que verifican si el la variable seguir recorriendo es true o false, así no se siguen con las llamadas recursivas. Esta variable es pasada por referencia en cada llamado a la función, así el cambio de valor se puede ver reflejado en cualquiera de los llamados.

- Recorrer: Esta función es muy parecida a con cada elemento, solo que esta vez en lugar de tener una variable booleana, se tiene el tamaño del vector por valor y el tope del vector por referencia, mientras el tope sea menor que el tamaño del vector, se hacen las llamadas recursivas y se guarda el elemento actual dentro del vector.

Es importante entender la diferencia entre los distintos tipos de árboles porque dependiendo de lo que se quiera hacer, se podrá escoger el que más se adapte a la situación. Un árbol binario sin ningún tipo de comparador para poder definir qué camino tomar no es la manera más óptima para realizar una búsqueda por ejemplo.

En el caso de las funciones en el árbol binario de búsqueda, tienen complejidad O(log n), menos el destruir todo, destruir y recorrer que tienen complejidad O(n) porque se recorre todo el árbol.

3. Detalles de implementación

- **abb insertar:** Esta función es la encargada de insertar nuevos elementos en el árbol, hace uso de la función auxiliar recursiva insertar elemento comparando. La función recursiva chequea si el nodo actual es nulo, en el caso de serlo reserva memoria para el nuevo nodo, setea los punteros de los hijos a null y le asigna el puntero al elemento nuevo. En el caso de no ser nulo el nodo actual, hace la comparación del elemento a insertar con el elemento del nodo actual para definir por cual subárbol seguir y finalmente llama así misma cambiando el nodo actual por nodo actual izquierda o derecha según corresponda. Cabe destacar que lo que retorna esta función debe ser asignado al nodo que se le pasa por parametro, ejemplo: ***hijo_izquierdo = funcion (hijo_izquierdo….);*** para poder guardar los cambios hechos en los hijos y seguir conectando los nodos.

- **abb quitar:** Para las llamadas recursivas se hace uso de dos funciones, quitar elemento comparando y buscar menor predecesor. Quitar elemento comparando se encarga de realizar la comparación, en el caso de no ser iguales, se llama así misma de acuerdo al resultado de la comparación. Aca lo mas complejo es en el caso de que la comparación de igual, es decir que llegamos al nodo que queremos eliminar.

En el caso de no tener hijos (nodos hojas) simplemente se libera la memoria del nodo. En el caso de tener un hijo, se apunta con un auxiliar, se libera la memoria del nodo que se va a eliminar y se retorna el auxiliar para conectar al hijo con el padre del nodo eliminado. }

En el caso de tener dos hijos, se hace uso de la función reemplazar con menor predecesor, esta función chequea si el hijo izquierdo tiene hijo derecho, en el caso de no tenerlo, se reemplaza el nodo a eliminar por su hijo izquierdo, y el hijo derecho del nodo a eliminar pasa a ser el hijo derecho del nodo que lo reemplaza. En el caso de que el hijo izquierdo tenga hijo derecho se hace uso de la función buscar menor predecesor, es recursiva, su caso de corte es al encontrar el nodo cuyo hijo derecho, hijo derecho de nuevo sea NULL. Se apunta con un auxiliar (predecesor inorden) el hijo izquierdo del predecesor inorden es asignado como hijo derecho del padre del predecesor inorden. Se retorna el predecesor inorden. Luego en la función reemplazar con menor predecesor es capturado el puntero al menor predecesor y se le asignan los hijos del nodo que se quiere eliminar. Se retorna de nuevo y se llega la función eliminar comparando, allí se captura de nuevo el puntero, se libera el nodo a eliminar y se retorna el menor predecesor para asignarlo como hijo al nodo que era padre del eliminado.

- **abb recorrer:** Esta función se encarga de guardar los elementos en el vector. Usa una función recursiva dependiendo del orden seleccionado. La función recursiva se encarga de guardar el elemento, para ello hace uso de la función guardar elemento vector. Antes de realizar las llamadas recursivas se chequea si el tope del vector es menor al tamaño del vector. Esto hace que si se alcanza el tamaño, no se sigan llamando recursivamente.
- **abb con cada elemento:** Es bastante parecida con abb recorrer, la diferencia es la función usada con los elementos y que el chequeo es si la variable seguir recorriendo es true, esta variable lo que hace es guardar el resultado de la función comparadora. Al cambiar a false, retorna (no retorna nada porque es void pero sale de la funcion) y como se llega a la función anterior si no se chequea se seguirá ejecutando en la línea que haya quedado, por eso es necesario el chequeo.
- **Funciones destructivas:** En las funciones destruir y destruir todo se usó un recorrido post orden ya que se tiene que liberar la memoria de los hijos antes que la del padre. Para ello se usó una función recursiva que va haciendo este recorrido y si recibe una función no nula la aplica al elemento, sino solamente libera el nodo. Esta función auxiliar recursiva es usada para ambas funciones, sólo cambia el sí recibe la función o recibe NULL.