FACULTAD
DE INGENIERIA
Universidad de Buenos Aires

TDA ABB

Lucas Franciulli

[7541/9515] Algoritmos y Programación II

Segundo cuatrimestre 2021

Nombre:	Franciulli, Lucas
Padrón:	107059
Email:	lfranciulli@fi.uba.ar

Índice

1. Introducción 3

2. Teoría 3

ARBOLES 3

ARBOL BINARIO 4

ARBOL BINARIO DE BUSQUEDA 4

¿POR QUÉ ES IMPORTANTE LA DISTINCIÓN DE CADA UNO DE ESTOS CONCEPTOS? 5

3. Funciones 6

3.1 abb_insertar () 6

3.2 abb_quitar () 7

3.3 abb_con_cada_elemento () 11

1. Introducción

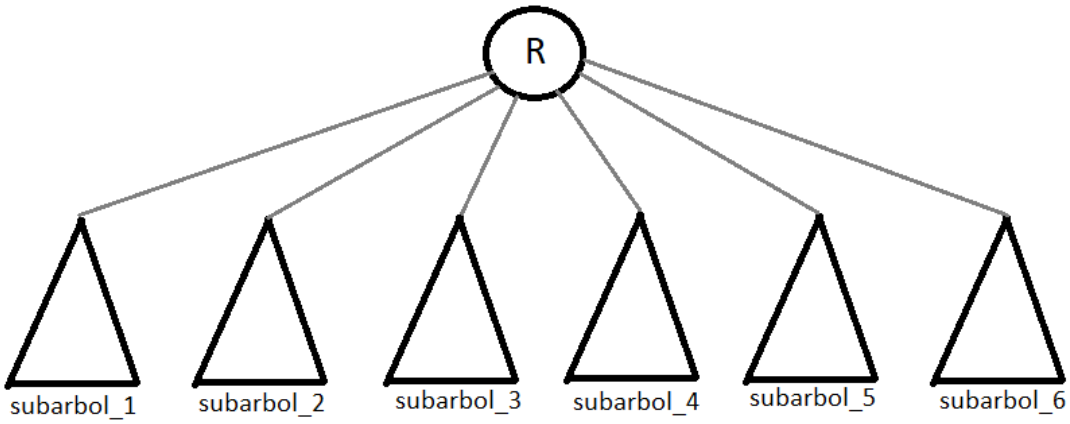
La idea de este informe es responder ciertas preguntas teóricas sobre árboles, arboles binarios y arboles binarios de búsqueda. Además, se intentará explicar de forma clara la lógica usada en algunas funciones del archivo **abb.c** para la implementación del *tda abb*.

2. Teoría

ARBOLES:

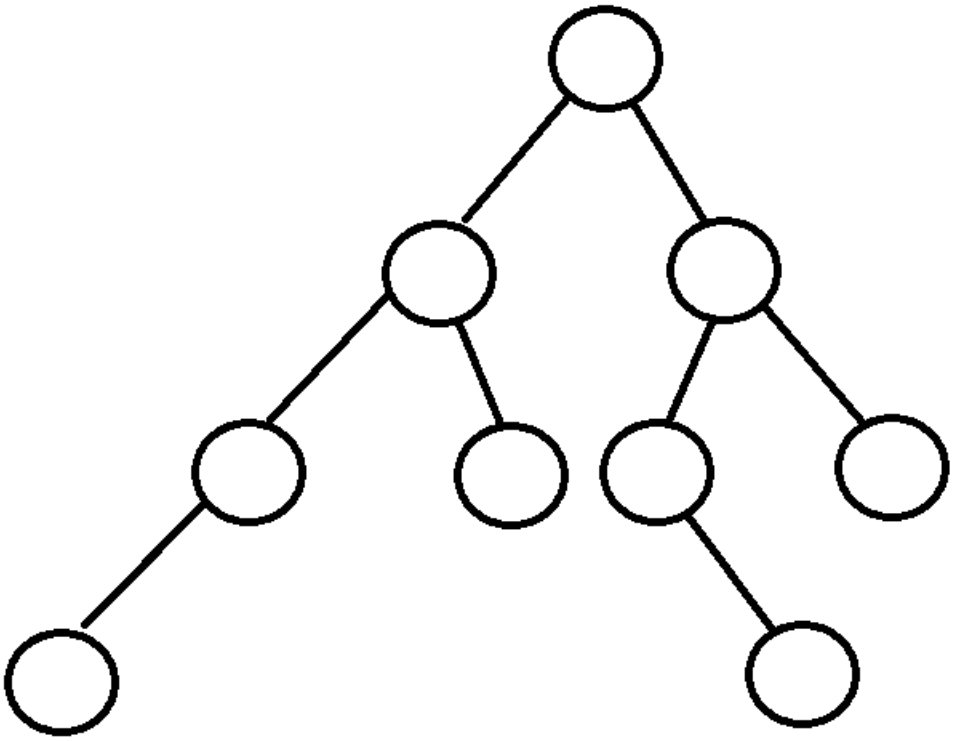
Un árbol es un tipo de estructura de datos la cual nos permite reducir el tiempo de acceso a los mismos. Tienen una naturaleza recursiva ya sea en sus diversas definiciones e implementaciones.

Los árboles están formados por nodos. Estos son elementos del árbol los cuales pueden estar vacíos o almacenando algún tipo de dato. Cada árbol tiene un nodo raíz y cero o varios subárboles no vacíos los cuales tienen a sus raíces conectadas al nodo raíz.



ARBOL BINARIO:

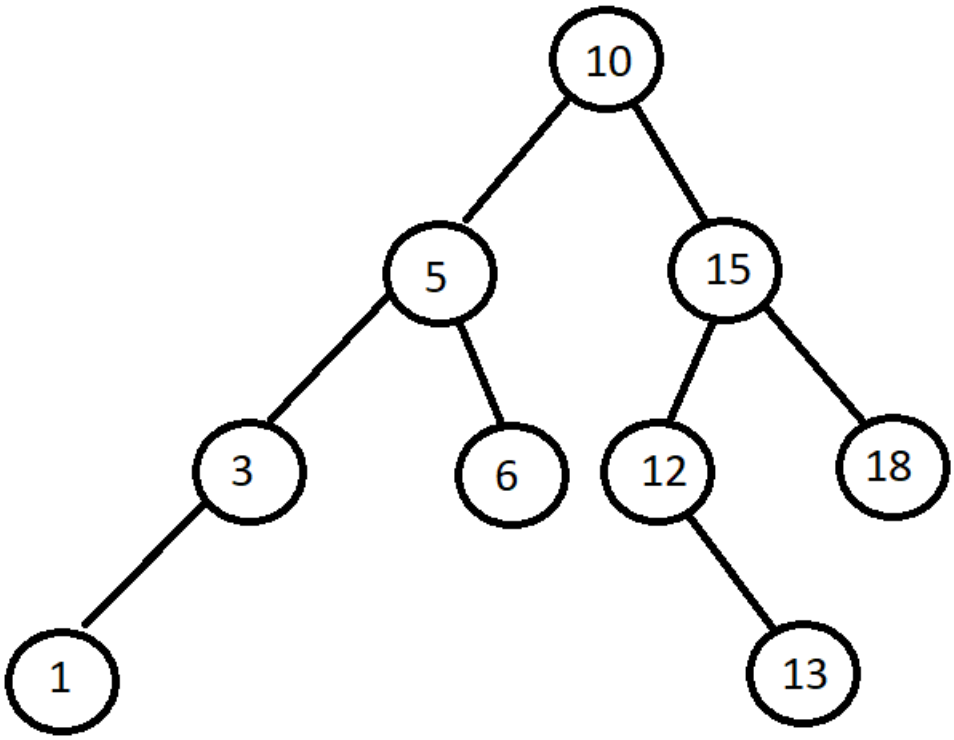
La idea de un árbol binario es que este se conforma de un nodo raíz y dos subárboles. Así mismo, cada nodo está compuesto por el valor que almacena y dos punteros a sus dos hijos. Ósea que cada nodo puede tener 0 ,1 o 2 hijos como máximo.



ARBOL BINARIO DE BUSQUEDA:

Un árbol binario de búsqueda es un árbol binario el cual se le suma una condición a los hijos de cada nodo. Esta es que el hijo izquierdo debe ser menor al nodo en el que estamos parados y que el nodo derecho sea mayor a este.

Gracias a este criterio es muchísimo más fácil buscar un elemento por el árbol.



¿POR QUÉ ES IMPORTANTE LA DISTINCIÓN DE CADA UNO DE ESTOS CONCEPTOS?

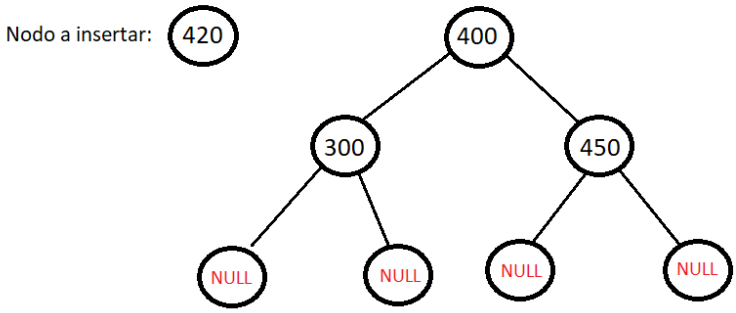
Es importante la distinción de cada uno de estos conceptos ya que a pesar de que un árbol binario de búsqueda sea un tipo especifico de árbol binario y que árbol binario sea un tipo de árbol, esto no significa que sean iguales. Por ejemplo, en el caso de los árboles existen muchísimos tipos: AVL, Arboles Rojo-Negro, etc. Y también con los árboles binarios existen otros además del ABB como por ejemplo los árboles binarios equilibrados.

3. Funciones

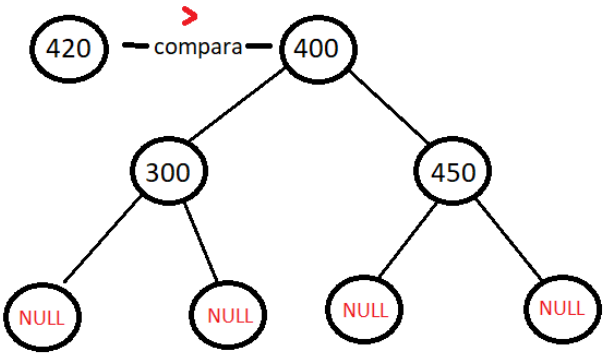
3.1 abb_insertar ()

La función ***abb_insertar()*** recibe por parámetro el árbol en el cual se quiere insertar el elemento y también recibe el elemento en cuestión. Al recibir ambos lo que hace es crear un nuevo nodo y en el apartado de elemento lo iguala al elemento que se pasó por parámetro. Una vez hecho esto le función se fija si el árbol tiene un ***nodo_raiz*** o no. Si no lo tiene entonces se encarga de insertarlo directamente ahí. Si lo tiene entonces llama a otra función llamada ***insertar_elemento()***. Esta se encarga de recorrer el árbol de forma tal que se para en el nodo vacío que tendría que ser insertado el nuevo nodo. Una vez insertado simplemente aumenta el ***arbol->tamanio*** por uno y devuelve el árbol.

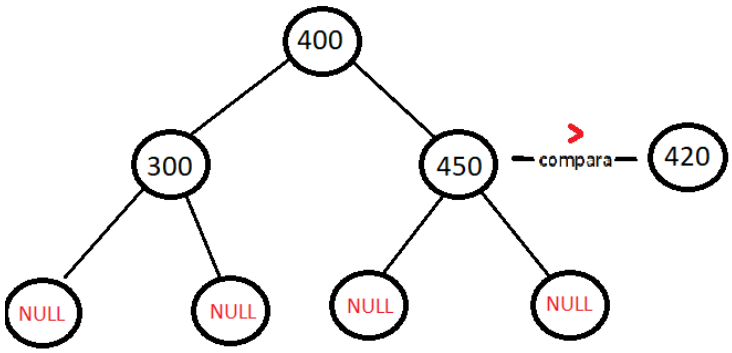
1



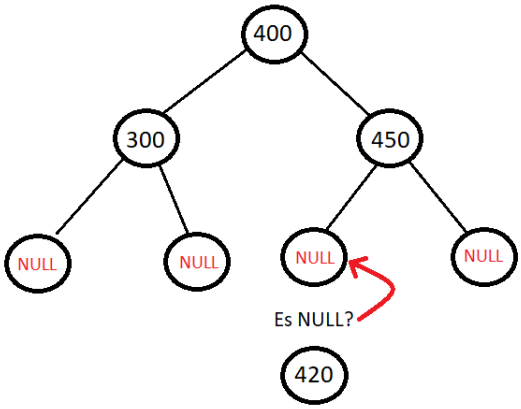
2



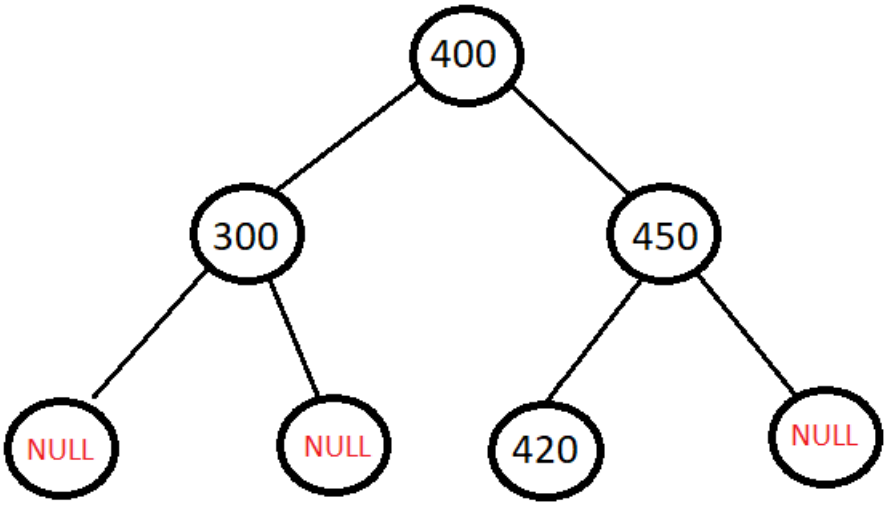
3



4



5

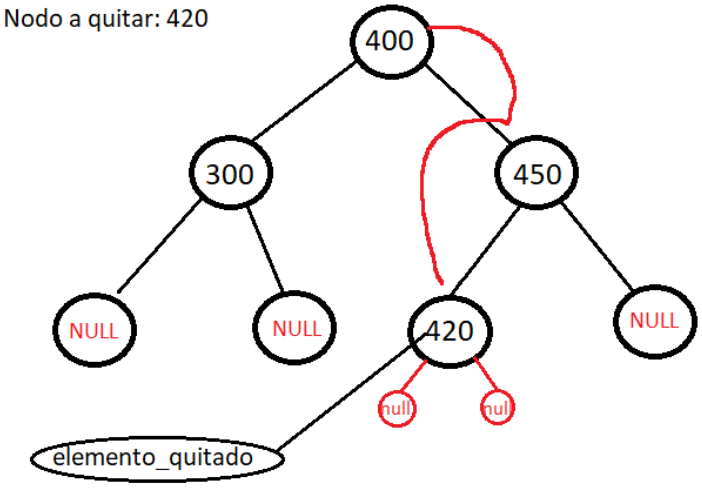


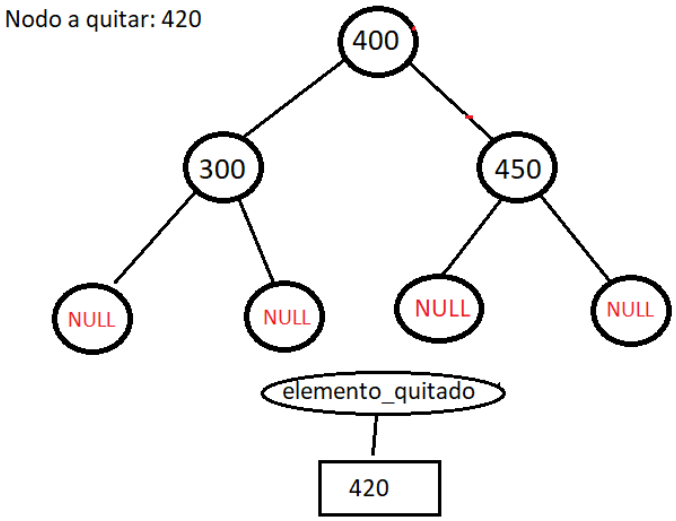
3.2 abb_quitar ()

La funcion **abb_quitar()** recibe por parametro el arbol del cual se quiere quitar cierto nodo y el elemento del nodo a quitar. Primero de todo esta funcion revisa que el arbol no este vacio y que este sea valido. Una vez hecho esto llama a la funcion **quitar_elemento()** la cual se va a encargar de hacer la mayoria.

La funcion **quitar_elemento()** se podria decir que esta dividida en 2 partes principales. La primera es la de recorrer el arbol hasta encontrar el nodo que debe ser quitado. La segunda se encarga de quitarlo. Luego esta segunda parte se subdivide en 4 casos:

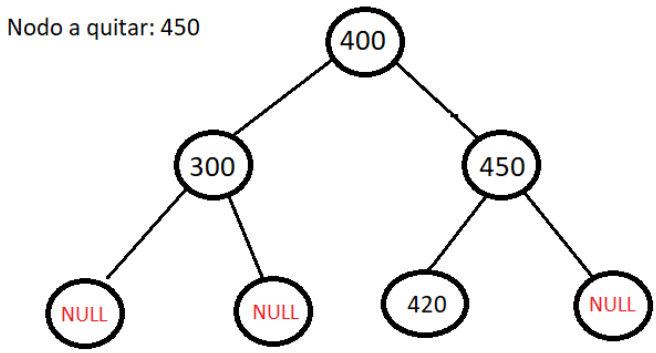
El primer caso es si el nodo a quitar no tiene ningun hijo. Entonces lo unico que debe hacer es guardar el elemento del nodo en el puntero *elemento_quitado*, liberar la memoria de ese nodo e igualar el puntero de este a *NULL* asi se sabe que esta vacio.



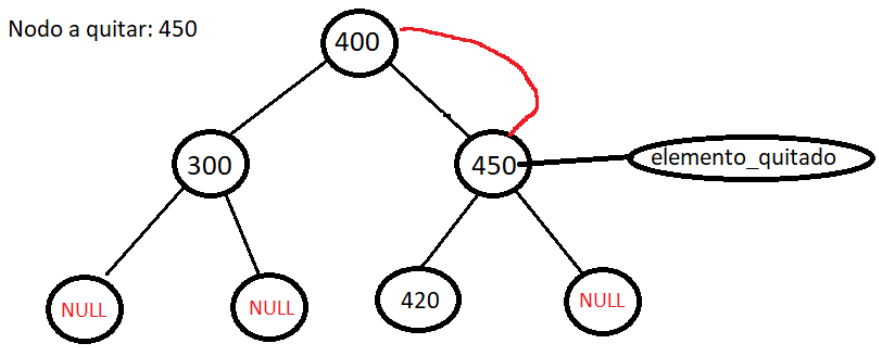


El segundo caso es si el nodo a quitar tiene un hijo, especificamente el hijo izquierdo. De ser asi entonces la funcion guarda un puntero al nodo, guarda el elemento del nodo en el puntero *elemento_quitado*, iguala el puntero del nodo al hijo izquierdo asi lo “hereda” y por ultimo libera el nodo a quitar.

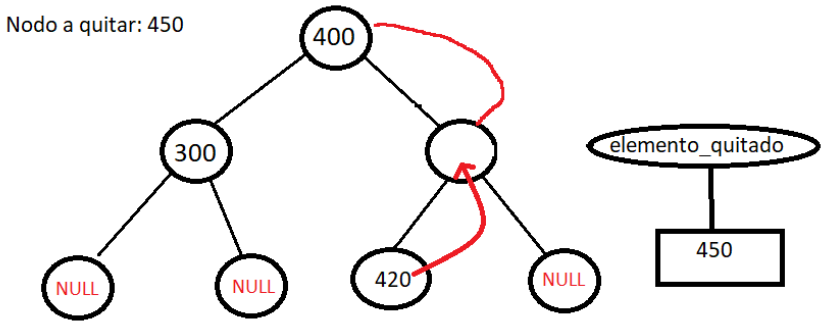
1



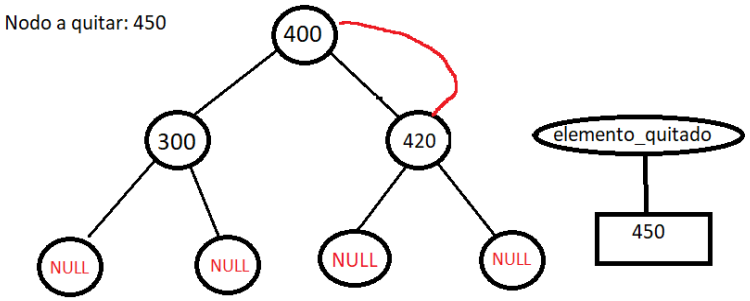
2



3



4

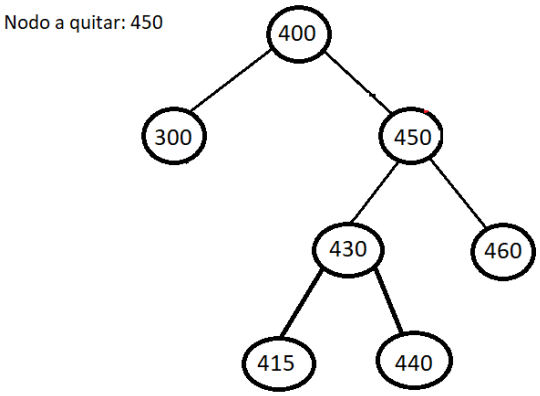


El tercer caso es igual al anterior solamente que se fija si tiene un hijo derecho y de asi serlo hace todo lo mismo que recien se mensiono pero en vez de heredad el nodo izquierda heredaria el nodo derecho, logicamente.

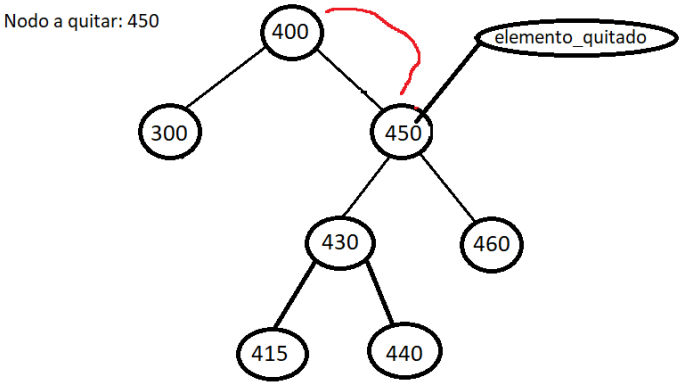
Y por último el caso de que el nodo a quitar tenga dos hijos. Aca la funcion guarda el elemento del nodo en el puntero *elemento_quitado*, encuentra un predecesor a este nodo llamando a la funcion **nodo_izquierdo_mas_derecho()**, la cual se encarga de pararse en el nodo

izquierdo del nodo a eliminar e ir al nodo mas derecho de este. Una vez hecho esto se iguala el elemento del nodo a quitar al del nodo izquierdo mas derecho y se llama otra vez a la funcion **quitar_elemento()** pero nos paramos en el hijo izquierdo del que era el nodo a quitar y se pide que se quite el que encontramos como predecesor ya que estaria ocupando el lugar del nodo a quitar y no queremos que este repetido.

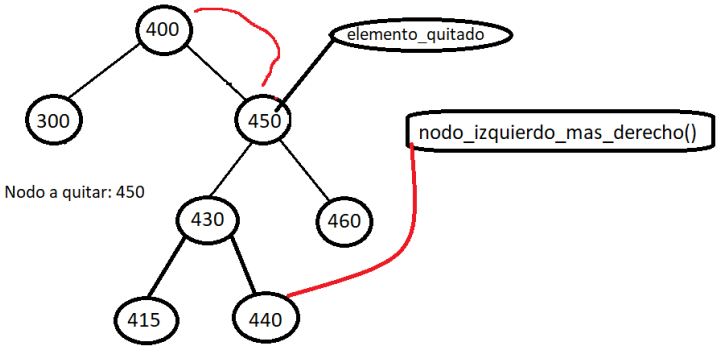
1



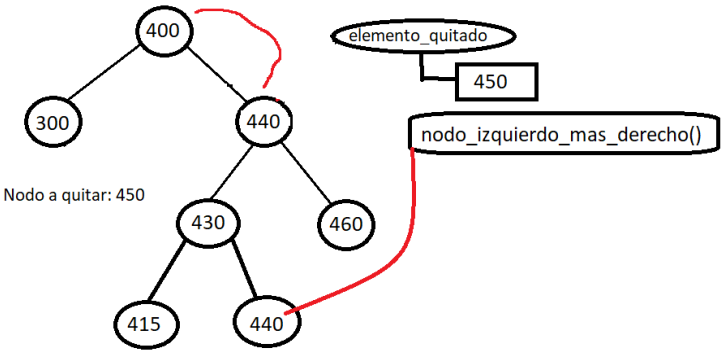
2

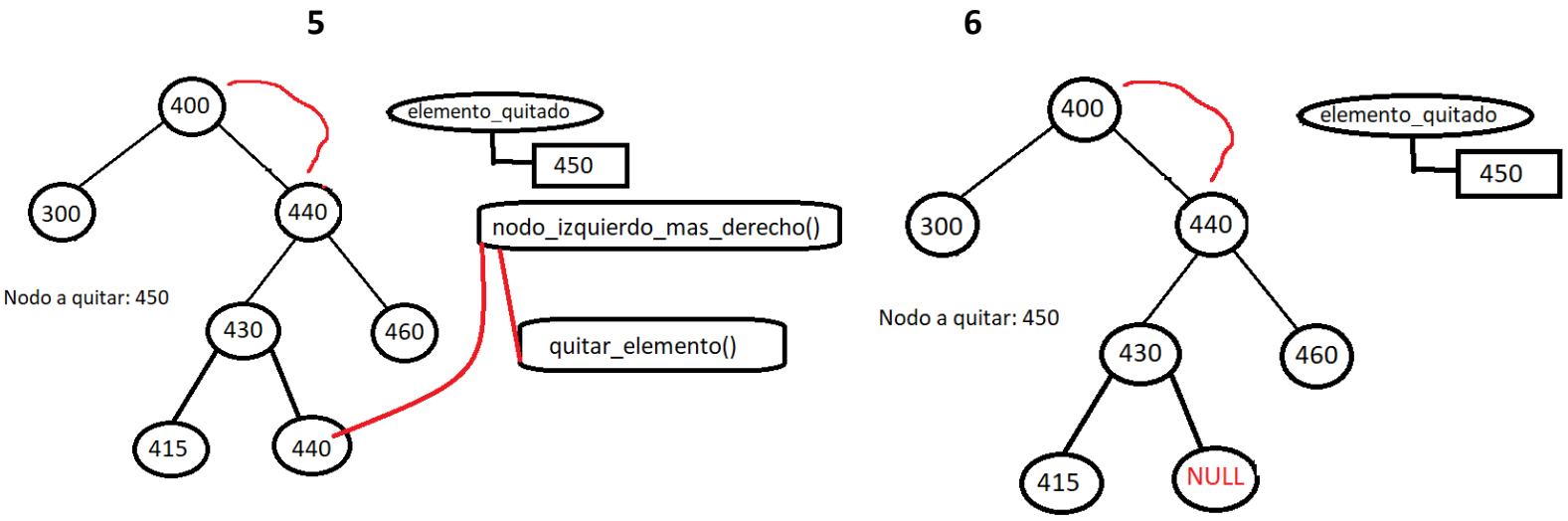


3



4





Una vez terminada la funcion **quitar_elemento()**, se fija que el *elemento_quitado* no sea nulo ya que significaria que no encontro el nodo que se pidio en el arbol. Luego disminuye la variable tamanio del arbol por uno. Despues verifica si el tamanio de este ahora es cero, ya que de serlo significa que no hay mas arbol y lo iguala a nulo. De no ser asi devuelve el *elemento_quitado*.

3.3 abb_con_cada_elemento ()

La funcion **abb_con_cada_elemento()** se encarga de recorrer el arbol en cierto orden e ir aplicando una funcion que fue pasada por parametro a cada elemento de los nodos. Esta primero verifica que sea valido el arbol y luego ve que tipo de recorrido fue pasado por parametro.

Si la funcion recibe el recorrido *inorden* esta llama a la funcion **abb_con_cada_elemento_inorden()**. En esta se hace un llamado recursivo con la funcion pasando por parametro primero al nodo izquierda del nodo actual, luego se aplica la funcion pasada por parametro al nodo y por ultimo se llama

recursivamente llamando a la funcion pasando por parametro al nodo derecho. Ademas de todo esto la funcion tiene una codicion de corte la cual es un booleano que de cambiar el valor a false para de avanzar la funcion y devuelve la cantidad de elementos recorridos hasta el momento.

Para los otros tipos de recorridos se sigue la misma logica solamente que se cambia de orden el llamamiento recursivo para el hijo izquierdo, aplicarle la funcion pasada por parametro al nodo actual y el llamamiento recursivo para el hijo derecho.

En el caso de si el recorrido es *preorden*, se llama a la funcion **abb_con_cada_elemento_preorden()** la cual aplica la funcion al nodo actual, luego hace el llamamiento recursivo con el hijo izquierdo y luego con el derecho.

El ultimo caso es el de *postorden*, el cual llama a la funcion **abb_con_cada_elemento_postorder()** que hace el llamamiento recursivo con el hijo izquierdo, luego con el derecho y por ultimo aplica la funcion al nodo actual.