FACULTAD
DE INGENIERIA
Universidad de Buenos Aires

TDA n2 — Arboles Binarios de búsqueda

[7541/9515] Algoritmos y Programación II

Segundo cuatrimestre de 2021

Alumno:	Vallejos, Matias
Número de padrón:	107924
Email:	matiavallejo@gmail.com
Email Institucional:	mvallejos@fi.uba.ar

1. Introducción

El objetivo de este TDA es implementar una solución simple al problema de la representación de jerarquía en estructuras de datos, en este caso vamos a utilizar **Arboles Binarios de búsqueda (ABB)** y así lograr una estructura de datos útil para su utilización y a su vez lograr optimizar la búsqueda lineal de este.

2. Teoría

Respuestas a las preguntas teóricas

1. ¿Qué es un árbol, árbol binario y árbol binario de búsqueda?

- Un **árbol** es una colección de nodos que a su vez están conectados a otros múltiples nodos. Este consiste en un nodo principal normalmente llamado **raíz** y la cantidad necesaria de **subárboles** los cuales son nodos que se formar a partir de la raíz. Su estructura se basa en la recursividad ya que esta cuenta con una condición de corte y llamadas a la función. Hay muchos tipos de arboles como, por ejemplo: Arboles binarios, AVL, árbol rojo-negro, etc. Para la implementación de este trabajo utilizaremos el tipo de árbol binario de búsqueda.
- Un **árbol binario** es un tipo de árbol que busca relacionarse con el método de búsqueda binaria, esto lo logra con la característica de que el nodo raíz esta solamente conectado a dos subárboles

permitiéndonos así definir la noción de izquierda y derecha. Este posee las siguientes operaciones: creación, destrucción, insertar, borrar, buscar, chequear si esta vacío, recorrer.

- Un **árbol binario de búsqueda** es un tipo de árbol binario el cual posee un orden en específico, para lograr este orden especifico tenemos que comparar los elementos de modo que cada árbol posea un valor o clave única. Gracias a este orden es que logramos optimizar la búsqueda lineal. Las características de este árbol que utilizaremos en este proyecto son: Las claves mayores se insertaran en los subárboles de la derecha, las menores en los de la izquierda. Estos subárboles también son arboles de búsqueda binaria.
- Es importante diferenciar cada concepto ya que no todos los árboles son binarios ni todos los árboles binarios tienen un ordenamiento especifico.

3. Detalles de implementación

Detalles específicos de la implementación, como compilar, como ejecutar

1. `nodo_insertar` Recibe un árbol, un elemento y un nodo, si el nodo recibido no existe lo crea. Inserta un nuevo nodo en el árbol binario utilizando el comparador es gracias a este que logra compararlo, si en la comparación el elemento a insertar es menor se inserta a la izquierda, sino a la derecha. En caso de ser iguales lo inserta a la izquierda. Devuelve el nodo insertado.
2. `buscar_nodo_mayor` Busca el nodo de elemento mayor de un árbol mediante recursividad y lo devuelve. Es una función auxiliar de `buscar_nodo_mayor_subarbol_izquierdo`
3. `buscar_nodo_mayor_subarbol_izquierdo` Busca el nodo mayor del subárbol izquierdo de la raíz.
4. `nodo_quitar` Recibe un nodo y lo quita del árbol. Si el nodo es hoja (No tiene hijos) hace que el padre apunte a NULL (Ver diagrama 3). Si el nodo tiene un solo hijo hace que el padre apunte al hijo del nodo a remover y finalmente remueve el nodo correspondiente (Ver diagrama 4). Si el nodo tiene dos hijos busca el **Sucesor inmediato**(el menor elemento del subárbol derecho) lo reemplaza por el nodo a remover y si el sucesor inmediato tuviera un hijo este lo une con el antiguo padre del sucesor inmediato (Ver diagrama 5). Devuelve el nodo que removió.
5. `buscar_nodo` Recorre el árbol buscando el elemento recibido. Devuelve el nodo que contiene el mismo elemento que estoy buscando.
6. `liberar_nodo` Recorre el árbol destruyendo todo, o sea aplicando la función destructora y liberando los nodos.
7. `abb_con_cada_elemento_x` es el iterador interno, recorre el árbol con el método indicado en el nombre de la función ya sea INORDER, PREORDER, POSTORDER. Utiliza la función recibida por argumento y si esta devuelve false deja de iterar, aumenta el contador recibido por parámetro hasta que deja de iterar. (Ver explicación métodos de recorrido y diagrama 2).
8. `abb_recorrer_x` es el iterador externo, recorre el árbol con el método indicado en el nombre de la función. Recibe un array por parámetro y su tamaño y va agregando los elementos leídos a ese array y aumentando su tamaño. también recibe un contador y lo va a aumentando por cada elemento insertado en el array. (Ver explicación métodos de recorrido y diagrama 2)

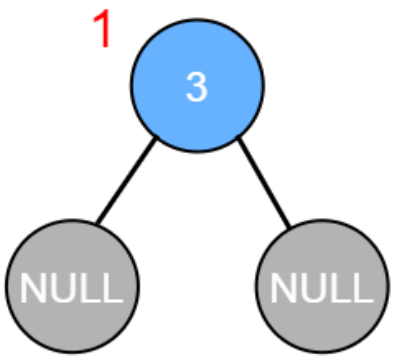
Métodos de recorridos

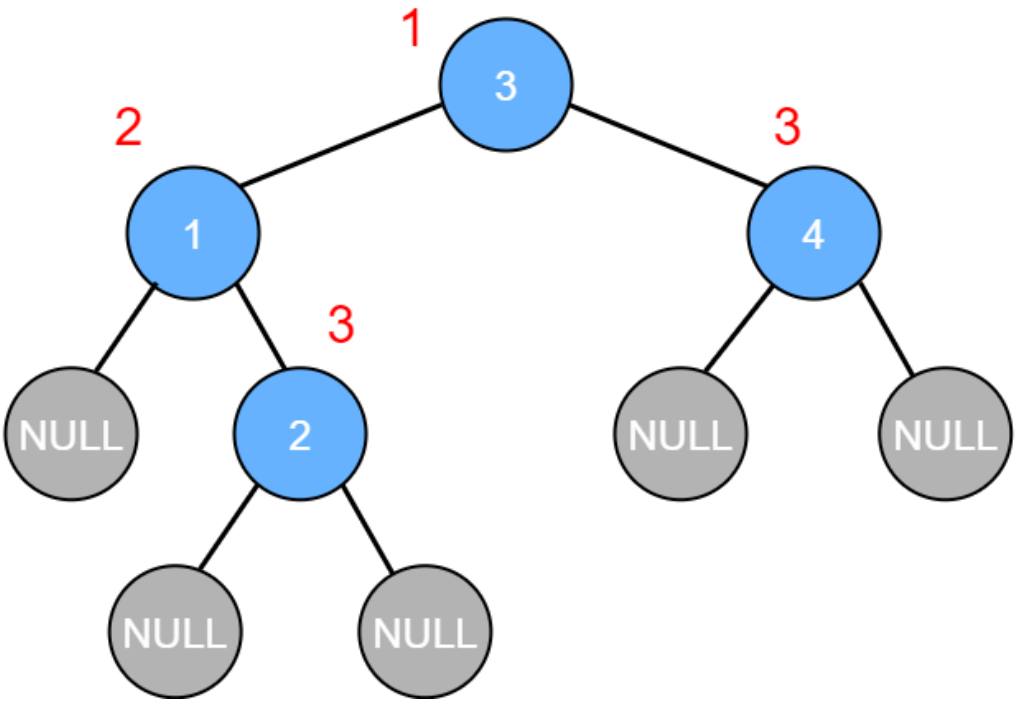
Existen 3 métodos de recorridos:

- **Inorder** Este es el más común, se utiliza para recorrer de manera ordenada el árbol ya sea numérica o alfabéticamente, consiste en primero visitar los subárboles izquierdo hasta el nodo final, luego el nodo actual (el anterior al nodo del subárbol izquierdo) y finalmente el subárbol derecho.
- **Preorder** Se utiliza para **clonar el árbol** primero se visita el nodo actual partiendo de la raíz, luego el subárbol izquierdo y finalmente el subárbol derecho.
- **Postorder** Se utiliza para **Destruir el árbol** primero se visita el subárbol izquierdo, luego el subárbol derecho y finalmente el nodo actual.

4. Diagramas

1. Diagrama1



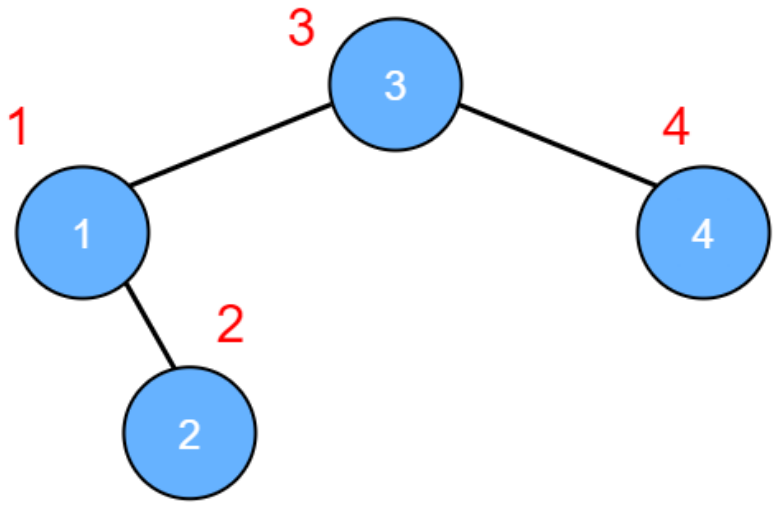


En este caso tenemos un árbol vacío y vamos a ingresar los elementos 3 - 1 - 2 - 4, Empezamos poniendo la raíz (3) y este nodo inicialmente va a tener un puntero a un nodo izquierda y derecha NULL, luego ingresamos los siguientes elementos, el 1 como es menor a 3 se va a insertar a la izquierda, luego el 2 como es menor a 3 va la izquierda y como es mayor a 2 va a la derecha y finalmente el 4 como es mayor a 3 va a la derecha.

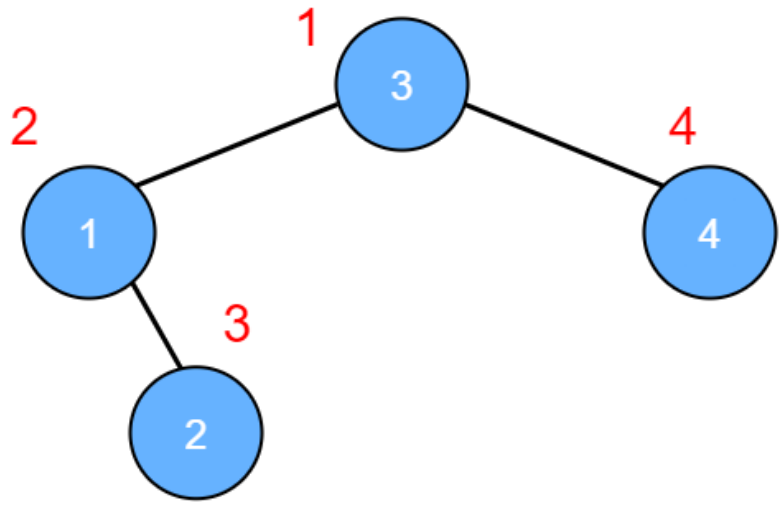
> Nota: Cada vez que se inserta un nodo nuevo este inicialmente en la izquierda y la derecha apunta a NULL hasta que se ingrese algo diferente.

2. Diagrama2

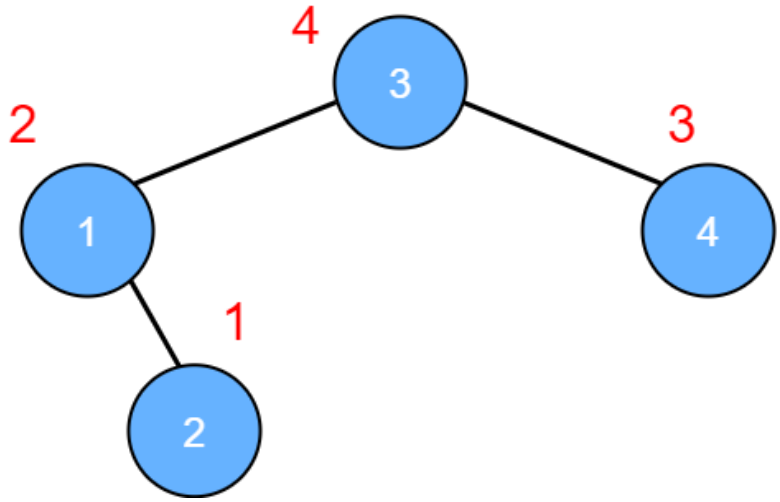
Recorrido Inorden



Recorrido Preorden



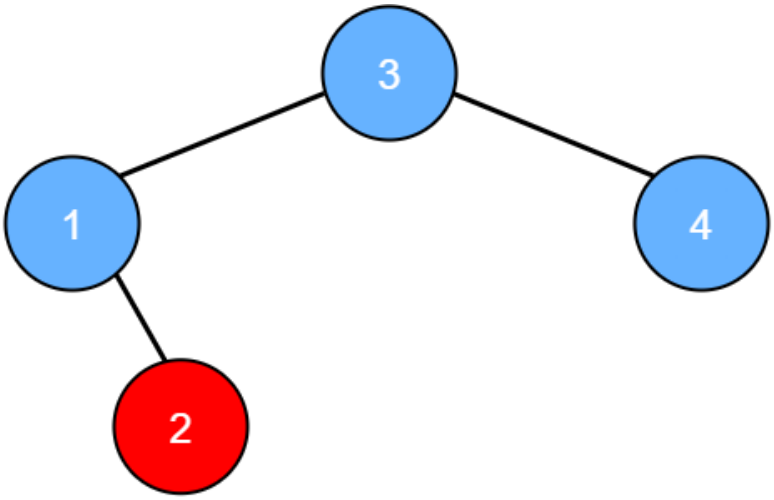
Recorrido Postorden

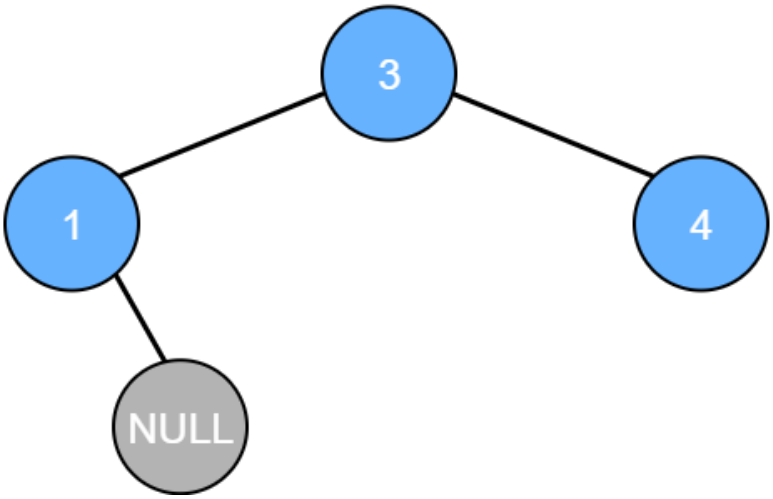


En este caso podemos ver como se recorre el árbol mediante sus diferentes métodos. (Ver explicación de métodos de recorridos). Dado que son bastante similares explicare el inorden, este si vamos a la explicación vemos que recorre de manera subárbol izquierdo -> Nodo actual -> subárbol derecho. es por eso por lo que partimos del 3 vamos al izquierdo y nos paramos en

1, como el 1 no tiene izquierdo anotamos el 1, luego vamos al árbol derecho el 2, como este no tiene ni izquierda ni derecha anotamos al 2, ya que finalizamos este subárbol izquierdo volvemos a la raíz, como es el nodo actual lo anoto, me voy al subárbol de la derecha y como este no tiene ni izquierda ni derecha lo anoto.

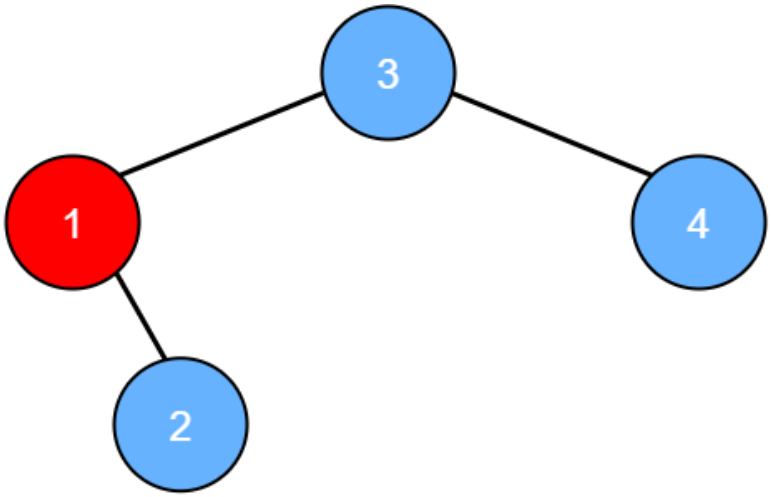
3. Diagrama3

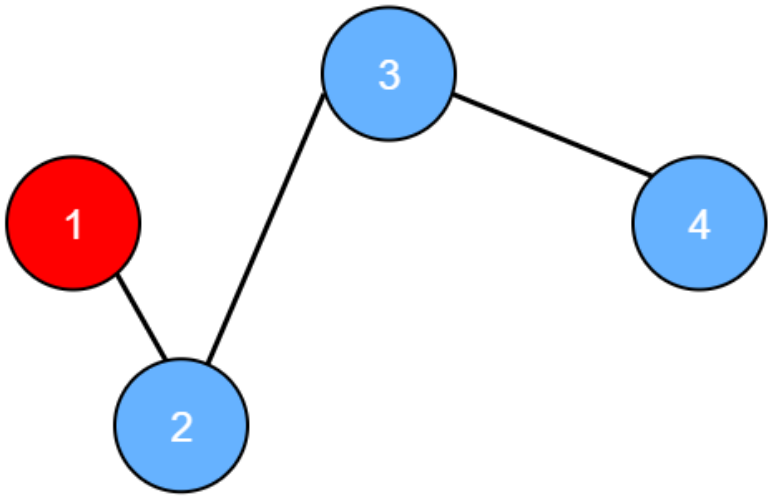


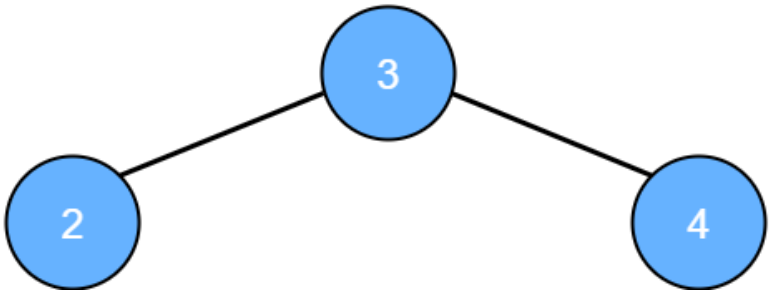


Método de eliminación nodo hoja En este caso vamos a remover el elemento 2, recorremos hasta encontrar el 2 en el árbol, una vez lo hallamos encontrado podemos notar que es un nodo hoja (no tiene hijos) por lo tanto su eliminación es sencilla, al no tener hijos solo liberamos ese nodo y finalmente el 1 va a apuntar hacia la derecha y izquierda a NULL.

4. Diagrama4

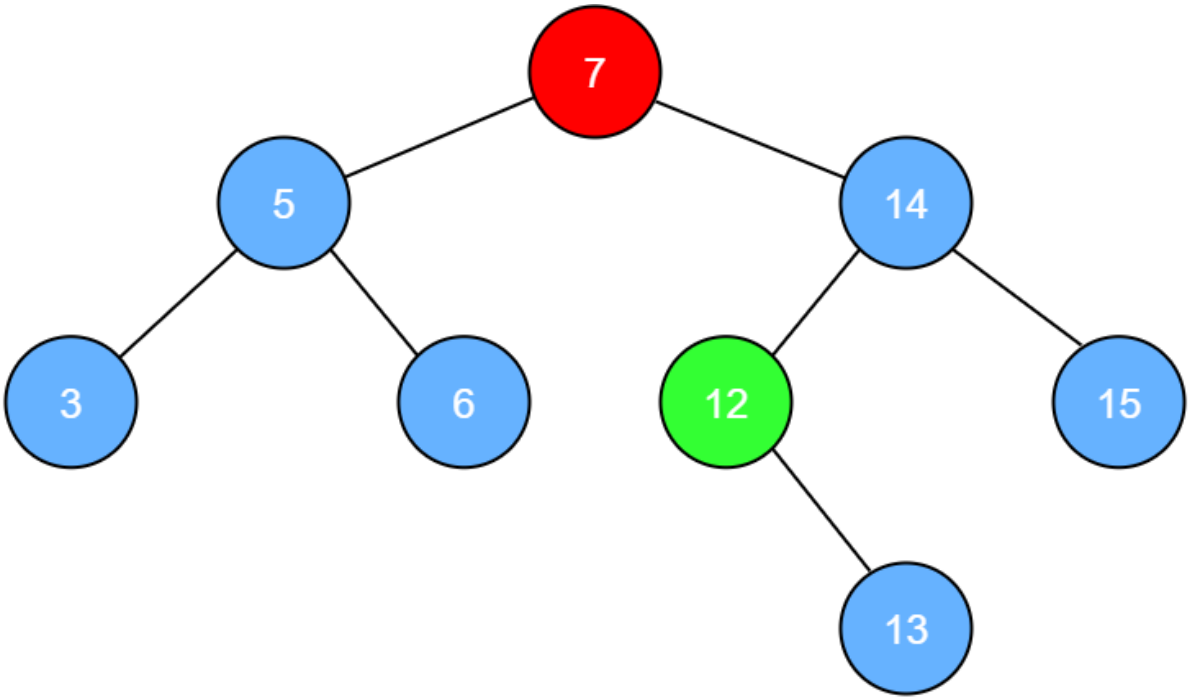


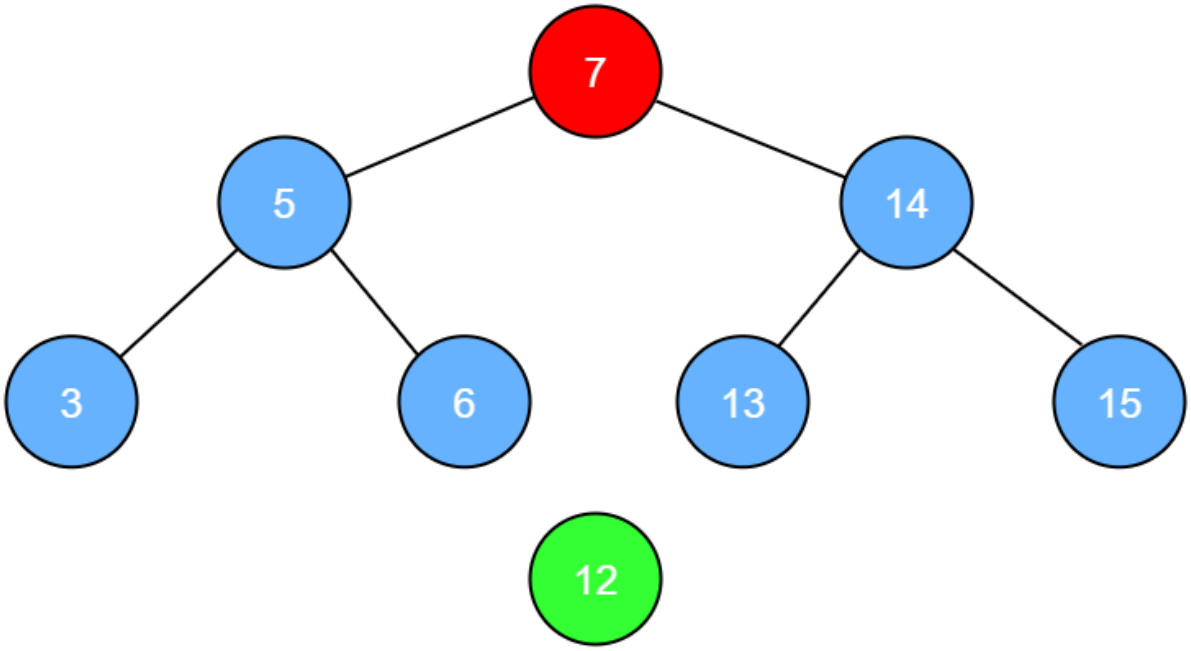


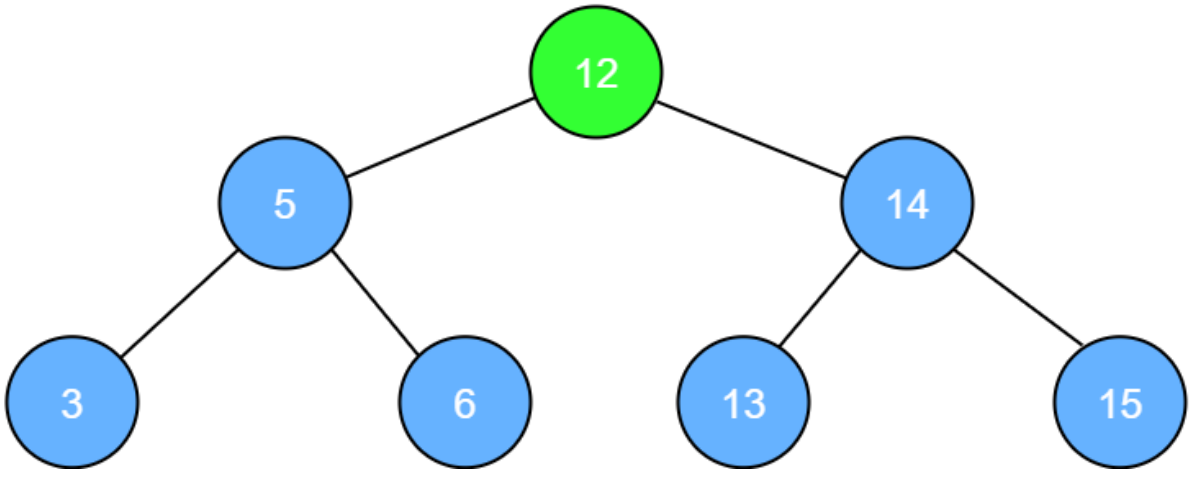


Método de eliminación nodo con un hijo En este caso vamos a remover el elemento 1, recorro hasta encontrar el 1 en el árbol, una vez lo hallo puedo notar que este posee 1 solo hijo por lo tanto no puedo removerlo directamente ya que si no pierdo la referencia a su hijo (el elemento 2), es por eso que lo que hago es unir el padre del nodo a remover a su hijo, en este caso uno el 1 al 2, una vez hecho esto ya puedo remover el elemento ya que no pierdo la referencia al 1.

5. Diagrama5







Método de eliminación nodo con dos hijos En este caso vamos a remover el elemento raíz 7,

recorro solo 1 vez hasta llegar al 7 ya que este es raíz, como tiene dos nodos hijos voy a buscar el **sucesor inmediato** o sea el menor elemento del subarbol de la derecha, en este caso es el 12, en caso de que el 12 no tenga hijos simplemente lo reemplazo por el 7 en este caso la raiz, sin embargo en nuestro diagrama tiene un hijo 13 por lo tanto lo que tengo que hacer es asignar al padre del sucesor inmediato(14) al hijo del sucesor inmediato(13). Gracias a este paso que hice no perdi la referencia al hijo del sucesor inmediato y finalmente puedo reemplazar el sucesor por el 7 raíz.

> Nota: Cada vez que un nodo de dos hijos se reemplaza con el sucesor inmediato este sucesor inmediato debe tener a la izquierda y derecha lo que tenía el elemento que iba a remover.