

Estructuras de datos

Programa informático con árboles binarios

Resultado de aprendizaje

TID42M

Walter González

Aldo Barrera

Docente Milton Batres



23 nov 2024

Contenido

Objetivo:.....	3
Material:	3
Desarrollo:.....	4
Problema 1.....	4
Planteamiento del problema	4
Solución de problemática.....	4
Problema 2.....	5
Planteamiento del problema	5
Solución de problemática.....	5
Implementación en main.....	12
Conclusiones:	14

REPORTE DE PRÁCTICA

Objetivo:

El objetivo de este resultado de aprendizaje es resolver diferentes problemáticas usando la estructura de arboles binarios.

Material:

Internet

IntelliJ

NeoVim (el comeback)

Desarrollo:

Primero, se aborda el planteamiento y solución de cada uno de los problemas planteados, posteriormente, se prueban todas las soluciones en el método main.

Problema 1

Planteamiento del problema

Escribir una función recursiva que encuentre el número de nodos de un árbol binario.

Solución de problemática

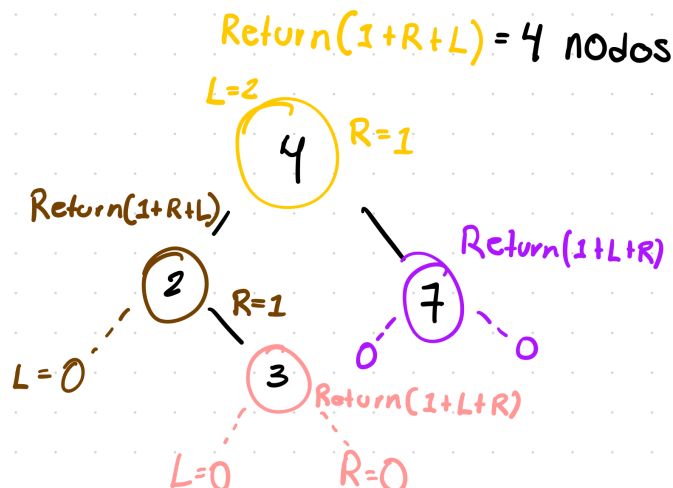
Para este problema se creó una nueva clase **TreeOperations<T>**, donde se implementó la función recursiva **countNodes(Nodo <T> root)**

```
public int countNodes(Nodo<T> root) {
    if (root == null) {
        return 0;
    }
    int l = countNodes(root.getLeft());
    int r = countNodes(root.getRight());

    return 1 + l + r;}

```

La función countNodes es recursiva y cuenta el número de nodos en un árbol binario. Si el nodo raíz (root) es null, retorna 0, manejando así el caso de un árbol vacío o un nodo hoja. Para nodos no nulos, la función llama recursivamente a sí misma para contar los nodos en los subárboles izquierdo y derecho. Luego, suma estos resultados más uno para incluir el nodo actual, devolviendo así el total de nodos en el árbol. Por ejemplo, en el árbol siguiente, esta función devolvería 4.



Problema 2

Planteamiento del problema

Escribir una función recursiva que encuentre la altura de un árbol binario.

Solución de problemática

Siguiendo la metodología de la función anterior, se tiene que analizar todo el árbol y ahora llevar un conteo de la cantidad de niveles que se desciende en cada camino del árbol.

Este acercamiento requiere que se retorne -1 en lugar de 0, ya que un nodo hoja tiene una altura de 0.

```
public int findAltura(Nodo<T> root){

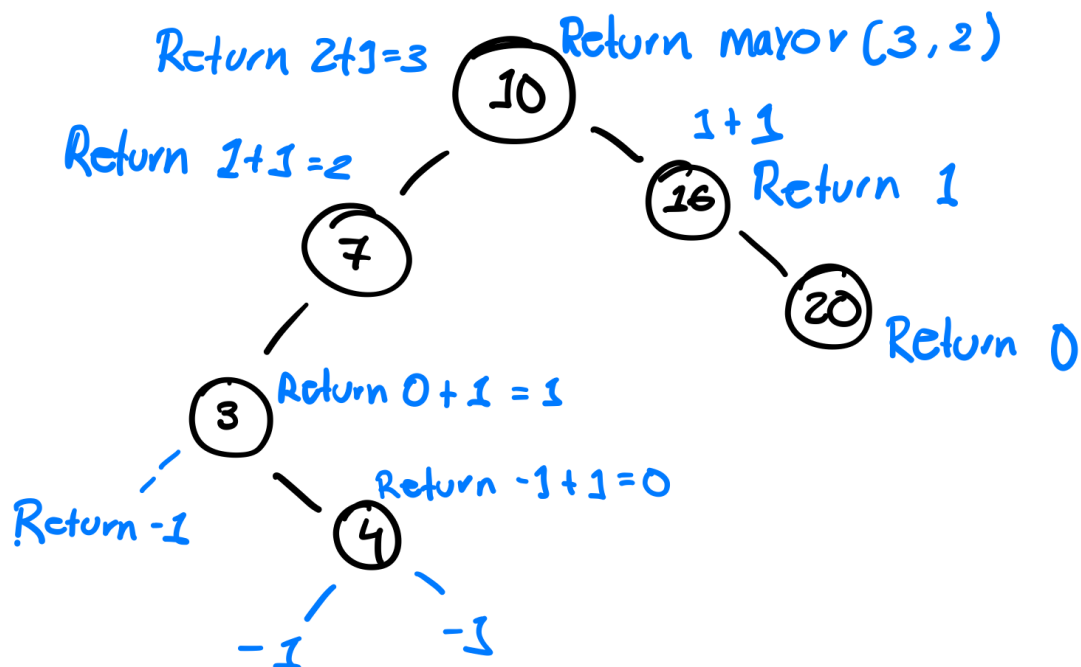
    if (root == null) return -1;

    int l = findAltura(root.getLeft());
    int r = findAltura(root.getRight());

    if (l < r){
        return r + 1;
    } else {
        return l + 1;
    }
    // Aquí si fue sentir que uno se avienta al vacío
    // confiando en la recursión
    // se sintió raro
}
```

La función **findAltura** calcula la altura de un árbol binario de manera recursiva. Si el nodo raíz (root) es null, la función retorna -1, lo que indica que un árbol vacío tiene una altura de -1. Para un nodo no nulo, la función realiza llamadas recursivas para calcular la altura de los subárboles izquierdo y derecho. Al alcanzar un nodo hoja, las llamadas a sus hijos retornan -1. La función luego compara las alturas de los subárboles izquierdo y derecho y devuelve el mayor de los dos + 1, para incluir el nodo actual en el conteo. Este proceso se repite hasta que se llega de nuevo al nodo raíz, donde se determina la altura total del árbol. La recursión permite explorar todo el árbol, confiando en que cada subárbol devuelve su altura correcta.

REPORTE DE PRÁCTICA



Problema 3

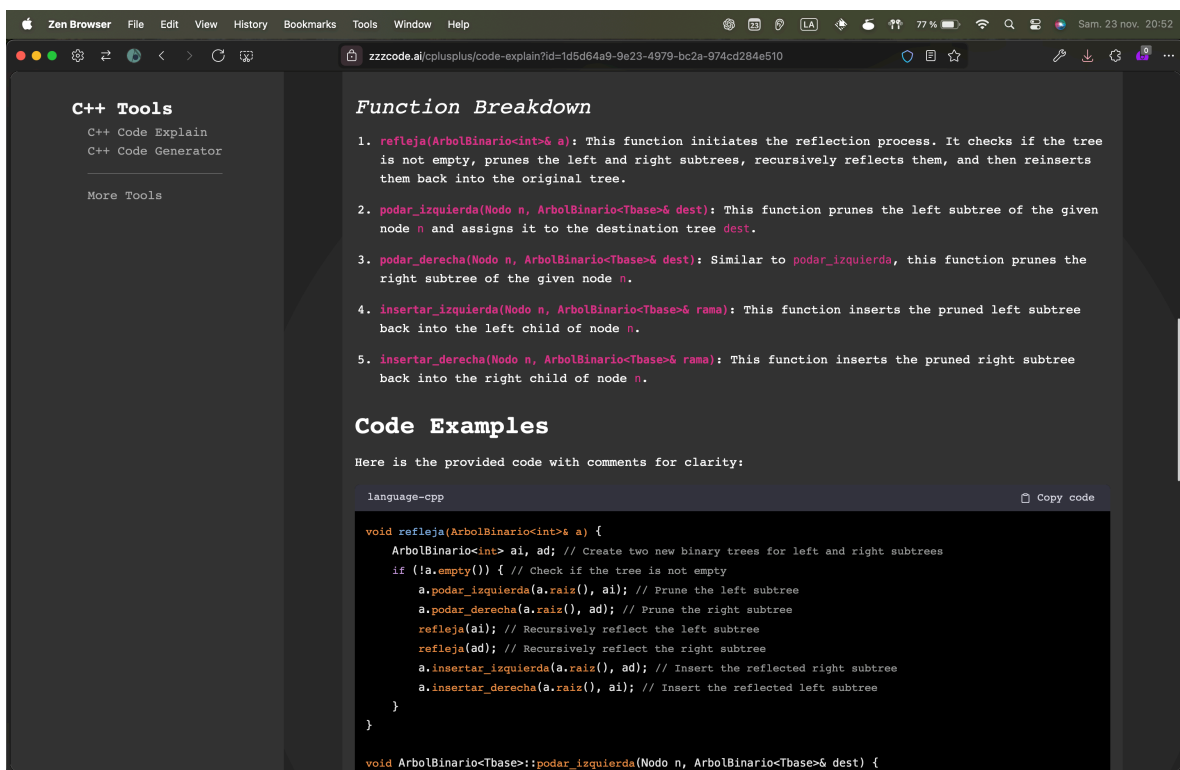
Planteamiento del problema

Implementar una función que refleje al árbol binario, traduciendo su implementación en C++.

El árbol resultante debe ser un reflejo del original (como si se reflejara en un espejo).

Solución de problemática

Para realizar este ejercicio, se utilizó la herramienta C++ Code Explain (<https://zzzcode.ai/cplusplus/code-explain>). Se cargó el código en la plataforma y analicé su contenido para comprender los conceptos específicos de C++ y planificar su adaptación a Java.



The screenshot shows the C++ Code Explain website interface. On the left, there's a sidebar with 'C++ Tools' including 'C++ Code Explain' and 'C++ Code Generator'. The main content area is titled 'Function Breakdown' and lists five steps: 1. `refleja(ArbolBinario<int>& a)`: This function initiates the reflection process. It checks if the tree is not empty, prunes the left and right subtrees, recursively reflects them, and then reinserts them back into the original tree. 2. `podar_izquierda(Nodo n, ArbolBinario<Tbase>& dest)`: This function prunes the left subtree of the given node `n` and assigns it to the destination tree `dest`. 3. `podar_derecha(Nodo n, ArbolBinario<Tbase>& dest)`: Similar to `podar_izquierda`, this function prunes the right subtree of the given node `n`. 4. `insertar_izquierda(Nodo n, ArbolBinario<Tbase>& rama)`: This function inserts the pruned left subtree back into the left child of node `n`. 5. `insertar_derecha(Nodo n, ArbolBinario<Tbase>& rama)`: This function inserts the pruned right subtree back into the right child of node `n`. Below this, there's a 'Code Examples' section with the text 'Here is the provided code with comments for clarity:'. It shows a C++ code snippet for the `refleja` function, which creates two new binary trees for left and right subtrees, checks if the tree is not empty, prunes the left and right subtrees, recursively reflects them, and then reinserts them back into the original tree. The code is in C++ and uses `ArbolBinario<int>` and `Nodo` types.

Posteriormente, se tradujo cada función a Java.

REPORTE DE PRÁCTICA

C++	Java
<pre>void refleja (ArbolBinario<int>& a) { ArbolBinario<int> ai,ad; if (!a.empty()) { a.podar_izquierda(a.raiz(),ai); a.podar_derecha(a.raiz(),ad); refleja(ai); refleja(ad); a.insertar_izquierda(a.raiz(),ad); a.insertar_derecha(a.raiz(),ai); } }</pre>	<pre>public Btree<Integer> refleja(Btree<Integer> copia) { if (copia.getRoot() == null) { return null; } Btree<Integer> ai = new Btree<>(); Btree<Integer> ad = new Btree<>(); ai = copia.podarIzq(copia.getRoot(), ai); ad = copia.podarDer(copia.getRoot(), ad); ai = refleja(ai); ad = refleja(ad); copia.insertarIzq(copia.getRoot(), ad); copia.insertarDer(copia.getRoot(), ai); return copia; }</pre>
<pre>void ArbolBinario<Tbase>::podar_izquierda(Nodo n, ArbolBinario<Tbase>& dest) { assert(n!=0); destruir(dest.laraiz); dest.laraiz=n->izqda; if (dest.laraiz!=0) { dest.laraiz->padre=0; n->izqda=0; } }</pre>	<pre>public Btree<T> podarIzq(Nodo<T> n, Btree<T> dest) { if (n != null) { dest.root = null; dest.root = n.getLeft(); if (dest.root != null) { n.setLeft(null); } } return dest; }</pre>
<pre>void ArbolBinario<Tbase>::podar_derecha(Nodo n, ArbolBinario<Tbase>& dest) { assert(n!=0); destruir(dest.laraiz); dest.laraiz=n->drcha; if (dest.laraiz!=0) { dest.laraiz->padre=0; n->drcha=0; } }</pre>	<pre>public Btree<T> podarDer(Nodo<T> n, Btree<T> dest) { if (n != null) { dest.root = null; dest.root = n.getRight(); if (dest.root != null) { n.setRight(null); } } return dest; }</pre>
<pre>void arbolBinario<Tbase>::insertar_izquierda(Nodo n, ArbolBinario<Tbase>& rama) { assert(n!=0); destruir(n->izqda); n->izqda=rama.laraiz; if (n->izqda!=0) { n->izqda->padre= n; rama.laraiz=0; } }</pre>	<pre>public Btree<T> insertarIzq(Nodo<T> n, Btree<T> rama) { if (rama == null) { return null; } if (n != null) { n.setLeft(null); n.setLeft(rama.root); if (n.getLeft() != null) { rama.root = null; } } return rama; }</pre>

REPORTE DE PRÁCTICA

```
void ArbolBinario<Tbase>::insertar_derecha (Nodo
n,      ArbolBinario<Tbase>& rama)
{
    assert(n!=0);
    destruir(n->drcha);
    n->drcha=rama.laraiz;
    if (n->drcha!=0)
    {
        n->drcha->padre= n;
        rama.laraiz=0;
    }
}
```

```
public Btree<T> insertarDer(Nodo<T> n, Btree<T>
rama) {
    if (rama == null) {
        return null;
    }
    if (n != null) {
        n.setRight(null);
        n.setRight(rama.root);
        if (n.getRight() != null) {
            rama.root = null;
        }
    }
    return rama;
}
```

Cuando se ejecuta la funcion refleja, se recibe un arbol binario y nos encontramos con el caso base de la llamada recursiva.

```
public Btree<Integer> refleja(Btree<Integer> copia) {
    if (copia.getRoot() == null) {
        return null;
    }

    Btree<Integer> ai = new Btree<>();
    Btree<Integer> ad = new Btree<>();
    ai = copia.podarIzq(copia.getRoot(), ai);
    ad = copia.podarDer(copia.getRoot(), ad);

    ai = refleja(ai);
    ad = refleja(ad);

    copia.insertarIzq(copia.getRoot(), ad);
    copia.insertarDer(copia.getRoot(), ai);

    return copia;
}
```

Vamos a detener esta recursión cuando la función refleja se ejecute sobre un nodo nulo y se devolverá nulo.

Se crean dos arboles nuevos, ai (inteligencia artificia... digo, arbol izquierdo) y ad (arbol derecho).

REPORTE DE PRÁCTICA

En estos nuevos arboles se llama a la función podar respectiva de cada lado (podarIzq(root, ai) o podarDer(nodo, ad)), enviando el nodo raíz actual y el nuevo objeto arbol.

```
public Btree<T> podarIzq(Nodo<T> n, Btree<T> dest) {  
    if (n != null) {  
        dest.root = null;  
        dest.root = n.getLeft();  
        if (dest.root != null) {  
            n.setLeft(null);  
        }  
    }  
    return dest;  
}
```

Esta función revisa si el nodo recibido no es nulo, despues, limpia el contenido del arbol destino (en este caso de ai); ahora, con el arbol destino limpio, se encarga de cambiar el root y le asigna el hijo izquierdo del nodo recibido. Posteriormente hace una comprobación para despejar la memoria del nodo utilizado (mañas de C++).

Ahora, el arbol destino almacena todo el subarbol izquierdo del nodo padre recibido, el cual es devuelto.

Se realiza lo mismo con el arbol ad

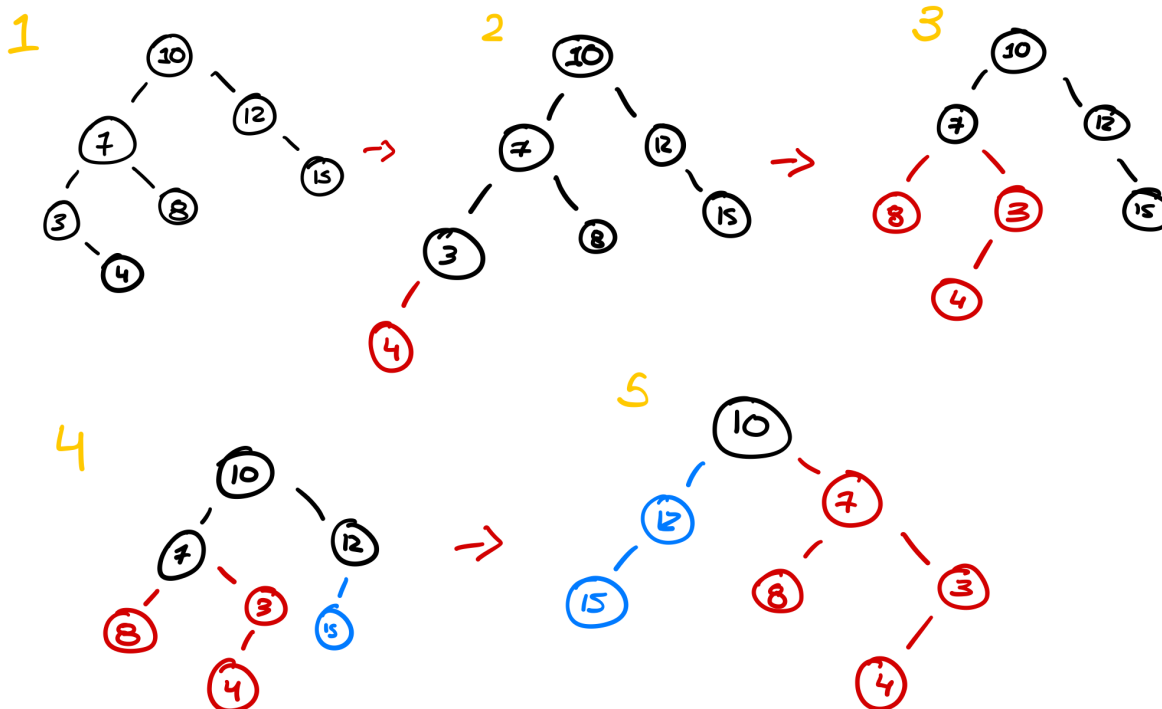
Ahora, ad y ai contienen los hijos del nodo root del arbol inicial.

Despues, ejecutamos los metodos insertarIzq e insertarDer al arbol inicial, con la finalidad de cambiar de posición los nodos.

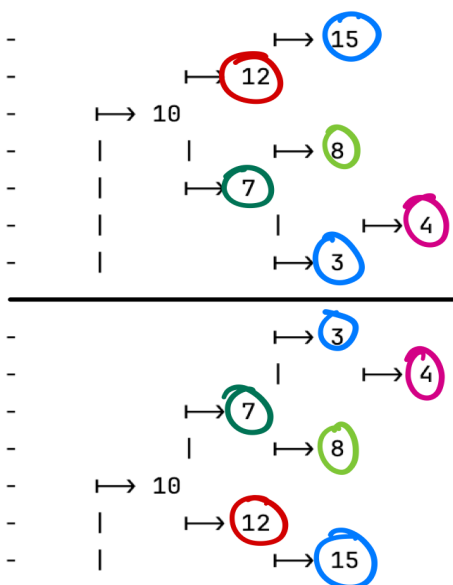
Esto sucede hasta alcanzar las hojas de cada rama, obteniendo así un arbol reflejado, donde los numeros a la derecha son menores y los numeros a la izquierda son mayores.

REPORTE DE PRÁCTICA

Ejemplos gráficos:



Ejecución



Implementación en main

```
import tree.TreePrinter;
import tree.Btree;

public class App {
    public static void main(String[] args) throws Exception {
```

Se crea un arbol binario, un objeto operador y un objeto printer.

```
Btree<Integer> tree = new Btree<Integer>();
TreeOperations<Integer> operator = new TreeOperations<Integer>();
TreePrinter printer = new TreePrinter<>();
```

Al arbol se le agregan los elementos siguientes:

```
tree.add(10);
tree.add(7);
tree.add(12);
tree.add(3);
tree.add(4);
tree.add(8);
tree.add(15);
```

Con el objeto operador, asignamos a count la cantidad de nodos en tree.

```
int count = operator.countNodes(tree.getRoot());
System.out.println("Cantidad de nodos: " + count);
```

Con el objeto operador, asignamos a count la altura maxima del arbol.

```
int height = operator.findAltura(tree.getRoot());
System.out.println("Altura del arbol: " + height);
```

Imprimimos el arbol normal y luego creamos un objeto arbol nuevo, construido a partir del reflejo del arbol original.

```
tree.treePrinter();

Btree<Integer> reflejo = tree.refleja(tree);

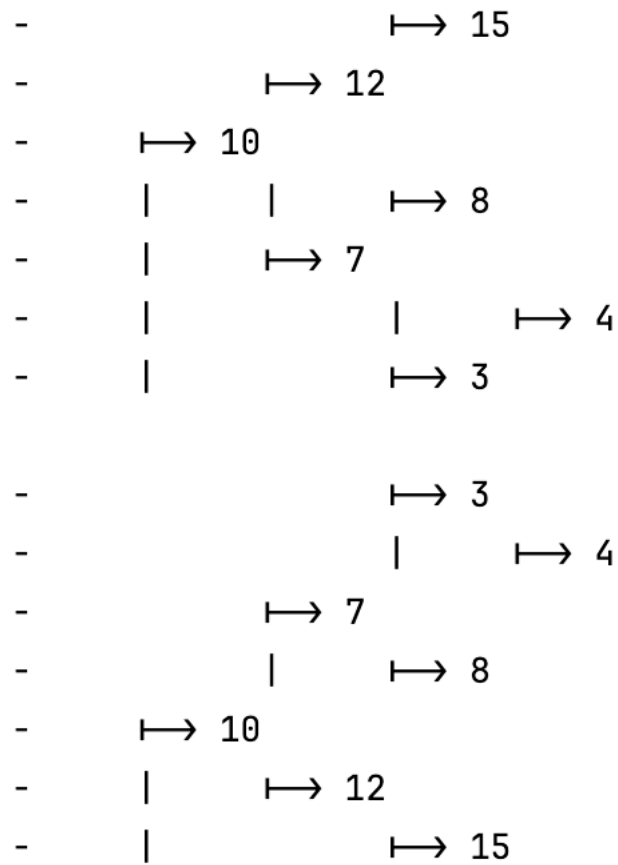
reflejo.treePrinter();

    }
}
```

REPORTE DE PRÁCTICA

Cantidad de nodos: 7

Altura del arbol: 3



Conclusiones:

En este resultado de aprendizaje se revisó bibliografía muy interesante, entre ellas varios problemas propuestos de leetcode y apoyos visuales para poder entender la logica que debe de seguir nuestro programa para responder a la problemática.

Lo más complicado de este ejercicio fue la comprensión de C++, al no tener ningún tipo de experiencia en este lenguaje me doy cuenta que Java es sumamente amigable, no tenía idea de lo (todavía más) verboso y complejo que era C++; aun asi, me gustó entender la logica detrás de los códigos y ver como varía el funcionamiento en cada lenguaje.

El intentar explicar cada función con dibujos ayudó mucho a la comprensión general de la recursión implicada y esperamos sea de apoyo para el lector de este reporte.

Para poder extender estos conocimientos, vamos a intentar implementar el arbol AVL, que es el que queremos lograr, pero por lo pronto, lo aprendido ha sido reforzado y entendido, una materia sumamente interesante y que nos deja con intención de seguir aprendiendo.

Referencias

FREE AI C++ Code Explainer: Explain C++ code online. (n.d.).

<https://zzzcode.ai/cplusplus/code-explain?id=771986d9-28c7-4532-9b6c-976dd4d6d0bd.com/watch?v=K0XXVSL4wUo>

Simplilearn. (2023, February 20). *An introduction to tree in Data Structure.* Simplilearn.com.

<https://www.simplilearn.com/tutorials/data-structure-tutorial/trees-in-data-structure>

Tree Data structure. (n.d.). <https://www.programiz.com/dsa/trees>

REPORTE DE PRÁCTICA

GeeksforGeeks. (2024a, August 21). *Binary Search Tree (BST) Traversals – Inorder, Preorder, Post Order*. GeeksforGeeks. <https://www.geeksforgeeks.org/binary-search-tree-traversal-inorder-preorder-post-order/>