

Pontificia Universidad Javeriana

Taller 03: Árboles (TAD, Funcionamiento y Prueba)



Tomas Alejandro Silva Correal
Juan Guillermo Pabon Vargas

Estructuras de datos

25 de Marzo de 2025

Índice:

| | |
|----------------------------------|-----------|
| Objetivos..... | 3 |
| Introducción..... | 3 |
| TAD Implementados..... | 3 |
| Árbol..... | 3 |
| Arbol binario..... | 4 |
| Árbol binario ordenado..... | 5 |
| Árbol AVL..... | 6 |
| Árbol KD..... | 7 |
| Árbol Expresión..... | 8 |
| Árbol Quad-Tree..... | 9 |
| Árbol rojo-negro..... | 10 |
| Pruebas de algoritmo..... | 11 |
| Árbol..... | 11 |
| Arbol binario..... | 12 |
| Árbol binario ordenado..... | 12 |
| Árbol AVL..... | 13 |
| Árbol KD..... | 14 |
| Árbol Expresión..... | 15 |
| Árbol Quad-Tree..... | 15 |
| Árbol rojo-negro..... | 16 |
| Conclusiones..... | 17 |

Objetivos

El objetivo de este documento y con ello, de este taller, es conocer a fondo el funcionamiento de los diferentes tipos de árbol y su comportamiento general, al igual que conocer o implementar estos conocimientos a no solo un entorno de programación, sino que además a su planeación y diseño originales (diseño en base a TAD), junto a sus pruebas correspondientes de funcionamiento.

El siguiente documento mostrará la planeación, diseño (por medio de TAD), desarrollo y pruebas correspondientes a cada una de las implementaciones dadas en la asignación para cada tipo de árbol.

Introducción

La asignación trae con sí una serie de entregables correspondientes al siguiente documento, entre estas una versión en código (lenguaje c++) de cada una de las versiones o tipos de árbol correspondientes, junto a algunos diseños (TAD) para algunos de los tipos de árbol. Inicialmente se trabajarán los TAD de cada tipo de árbol en la entrega, tanto algunos ya dados como otros que se deberán de crear, estos se diferenciarán más adelante. Posterior a esto, se mostrarán pruebas de algoritmo para cada tipo de árbol con el caso creado dentro de su main, y valores predeterminados y preparados con antelación.

TAD Implementados

A continuación, se mostraran los TAD tantos dados como creados para cada versión de árbol correspondiente a la entrega, dando este un total de 7 (árbol, árbol binario, árbol binario ordenado, AVL, KD, de expresión y quad-tree).

Árbol

TAD Árbol:

Datos Mínimos:

- raiz, señala el nodo que corresponde a la raíz del árbol.

Comportamiento:

- Arbol(): crea un arbol con raiz nula.
- Arbol(val): crea una arbol con raiz de valor val.
- ~Arbol()
- esVacio(): booleano, devuelve verdadero si la raiz del arbol es nula.

//Get y set

- obtenerRaiz(), Nodo, retorna un apuntador a la raiz del arbol.
- fijarRaiz(nraiz), recibe un apuntador al Nodo y lo asigna a la raiz del arbol.
- fijarDato(val, valNodo), asigna el valor de un nodo (valNodo) por val.

//otros

- insertarNodo(padre, n), booleano, recibe el valor del padre, el cual se busca, y se le ingresa un Nodo hijo con valor n, si se inserta retorna verdadero.

- eliminarNodo(n), booleano ,elimina un nodo de valor n, retorna verdadero si se elimina.
- buscarNodo(val), retorna un apuntador al nodo de valor val.
- altura(), entero, retorna la altura del arbol.
- tamano(), entero, retorna el número de nodos en el arbol
- preOrden(), imprime en preorden el arbol.
- posOrden(), imprime en postorden el arbol.
- inOrden(), imprime en inorden el árbol.
- nivelOrden(), imprime en nivel orden el arbol.

TAD Nodo:

Datos Mínimos:

- T dato: contenido del nodo
- hijos: contenedor, contiene un conjunto de apuntadores a nodos.

Comportamiento:

- Nodo(): crea un nodo vacío.
- Nodo(const T& val): crea un nodo a partir de un valor
//Get y set
- obtenerDato(): retorna el dato que es contenido por el nodo
- fijarDato(val): asigna el valor de un nodo (valNodo) por val.
//Otro
- adicionarDesc(nval): adiciona un Nodo creado a partir de val al conjunto de descendientes
- limpiarLista(): elimina la lista de descendientes.
- buscar(val): apuntador,retorna un apuntador al nodo de valor val de alguno de los subarboles.
- altura(): entero,retorna la altura del árbol
- tamano(): entero,retorna el número de nodos del subárbol que tiene como raíz el nodo
- preOrden(): imprime el arbol que tiene como raíz el nodo en PreOrden.
- posOrden(): imprime el arbol que tiene como raíz el nodo en PosORden
- inOrden(): imprime el arbol que tiene como raíz el nodo en inOrden.
- nivelOrden(cola): imprime el arbol que tiene como raíz el nodo en NivelOrden.

Arbol binario

TAD ArbolBinario

Datos Mínimos:

- raiz,NodoBinario: señala al primer nodo del árbol binario.

Comportamiento:

- ArbolBinario(): Crea un árbol binario vacío
//Get
- getRaiz(): NodoBinario, retorna el nodoBinario correspondiente a la raíz del árbol.
//otro
- esVacio(): retorna verdadero si el árbol no posee nodo raíz.
- datoRaiz(): retorna el contenido de la raíz.

- altura(): entero, retorna la altura del árbol.
- tamaño(): entero, retorna el número de nodos que posee el árbol.
- insertar(valor): inserta un valor en el árbol siguiendo los parámetros de orden.
- eliminar(valor): busca un valor en el árbol y lo elimina.
- buscar(valor): busca un valor en el árbol y retorna el apuntador si lo encuentra.
- preOrden(subarbol): realiza la impresión preorden de un subárbol
- inOrden(subarbol): realiza la impresión Inorden de un subárbol
- posOrden(subarbol): realiza la impresión postOrden de un subárbol
- nivelOrden(subarbol): realiza la impresion Nivel Orden de un subárbol

TAD NodoBinario

Datos Mínimos:

- dato, T, posee el contenido del nodo
- Izq, NodoBinario, apuntador hacia el hijo izquierdo
- Dere, NodoBinario, apuntador hacia el hijo derecho

Comportamiento:

- NodoBinario(dato): crea un nodo binario e inicializa su dato con el dato enviado por parámetro.
- NodoBinario(): crea un nodo binario vacío.
- //Get y set
- obtenerDato()
- fijarDato(val)
- obtenerHijoIzq()
- obtenerHijoDer()
- fijarHijoIzq(izq)
- fijarHijoDer(der)

Árbol binario ordenado

TAD ArbolBinario

Datos Mínimos:

- raiz, NodoBinario: señala al primer nodo del árbol binario.

Comportamiento:

- ArbolBinario(),
//Get
- getRaiz(), NodoBinario, retorna el nodoBinario correspondiente a la raíz del Árbol.
//otro
- esVacio(), retorna verdadero si el árbol no posee nodo raíz.
- datoRaiz(), ?, retorna el contenido de la raíz.
- altura(), entero, retorna la altura del árbol.
- int tamaño(), entero, retorna el número de nodos que posee el árbol.
- insertar(valor), inserta un valor en el árbol siguiendo los parámetros de orden.
- altura(subarbol), entero, retorna la altura de un subárbol que es enviado por parámetro.
- tamaño(subarbol), entero, retorna el número de nodos que posee un subárbol que es enviado por parámetro.

- insertar(valor,subarbol),retorna un apuntador a la raíz de un subárbol al cual se le ingresa un valor enviado por parámetro.
- eliminar(valor),busca un valor en el árbol y lo elimina.
- buscar(valor), busca un valor en el árbol y retorna verdadero si lo encuentra.
- void preOrden(subarbol), realiza la impresión preorden de un subárbol
- void inOrden(subarbol), realiza la impresión Inorden de un subárbol
- void posOrden(subarbol), realiza la impresion posOrden de un subarbol
- void nivelOrden(subarbol), realiza la impresion NivelOrden de un subárbol

TAD NodoBinario

Datos Mínimos:

- dato,T, posee el contenido del nodo
- Izq, NodoBinario,apuntador hacia el hijo izquierdo
- Dere, NodoBinario,apuntador hacia el hijo derecho

Comportamiento:

- NodoBinario(dato), crea un nodo binario e inicializa su dato con el dato enviado por parámetro.
- NodoBinario(), crea un nodo binario vacío.
//Get y set
- obtenerDato()
- fijarDato(val)
- obtenerHijolzq()
- obtenerHijoDer()
- fijarHijolzq(izq)
- fijarHijoDer(der)

Árbol AVL

TAD ArbolBinarioAVL

Datos Mínimos:

- raíz, apuntador a NodoBinarioAVL que contiene el dato y sus respectivos hijos

Operaciones:

- ArbolBinarioAVL(), crea un nuevo árbol binario AVL
- esVacio(), revisa si el árbol está vacío
- datoRaiz(), retorna el dato en la raíz
- altura(inicio), retorna la altura del árbol
- tamano(inicio), retorna el tamaño del árbol
//Giros del árbol
- giroDerecha(inicio), Realiza el giro a derecha
- giroIzquierdaDerecha(padre), Realiza el giro Izquierda-Derecha
- giroIzquierda(inicio), Realiza el giro a izquierda
- giroDerechaIzquierda(padre), Realiza el giro Derecha-Izquierda
//Agregar, eliminar y buscar valores
- insertar(val), inserta un valor al árbol
- eliminar(val), elimina un valor del árbol
- buscar(val), busca un valor en el árbol

//Impresiones

- preOrden(), imprime en forma pre-order
- inOrden(), imprime en forma in-order
- posOrden(), imprime en forma pos-order
- nivelOrden(), imprime por nivel

TAD NodoBinarioAVL

Datos Mínimos:

- dato, posee el contenido del nodo
- hijoIzq, contiene la dirección al hijo derecho
- hijoDer, contiene la dirección al hijo izquierdo

Operaciones:

- NodoBinarioAVL(), crea un nuevo nodo binario
- //Get y set
- getDato()
- setDato(val)
- getHijoIzq()
- getHijoDer()
- setHijoIzq(izq)
- setHijoDer(der)

Árbol KD

TAD kdtree

Datos mínimos:

- raíz, contiene la raíz del árbol

Operaciones:

- kdtree(), crea un nuevo árbol KD
- esVacio(), booleano, define si el árbol KD está vacío o no
- datoRaiz(), retorna el dato en la raíz
- altura(), define la altura del árbol
- int tamano(), define el tamaño del árbol
- insertar(val), inserta un valor en el árbol
- eliminar(val), booleano, elimina un valor del árbol, retorna si se elimino correctamente
- buscar(val), busca y entrega el valor a buscar en el árbol
- //Impresiones
- preOrden(), imprime en forma pre-order
- inOrden(), imprime en forma in-order
- posOrden(), imprime en forma pos-order
- nivelOrden(), imprime por nivel
- maximo(maxi), obtiene el valor máximo del árbol
- minimo(mini), obtiene el valor mínimo del árbol

TAD kdnodo

Datos mínimos:

- datos, vector de datos del nodo
- hijoIzq, apuntador al hijo izquierdo

- hijoDer, apuntador al hijo derecho
- tag, etiqueta del nodo

Operaciones:

- kdnodo(), construye un nodo kd
//Get y set
- obtenerDato()
- fijarDato(vector < T > & val);
- obtenerHijolq();
- obtenerHijoDer();
- fijarHijolq(kdnodo<T> *izq);
- fijarHijoDer(kdnodo<T> *der);
- fijarTag(int value);
//otros
- altura(), define la altura del árbol
- tamano(), define el tamaño del árbol
- insertar(val), inserta el valor en el árbol
- buscar(val), busca el valor en el árbol
- maximo(maxi), obtiene el valor máximo del árbol
- minimo(mini), obtiene el valor mínimo del árbol
//Impresiones
- preOrden(), imprime en forma pre-order
- inOrden(), imprime en forma in-order
- posOrden(), imprime en forma pos-order
- nivelOrden(), imprime por nivel
- imprimir(), imprime valor resultado

Árbol Expresión

TAD ArbolExp

Datos Mínimos:

- raíz , de tipo NodoExpresion indica el inicio del arbol

Comportamiento:

- ArbolExpresion(), crea un arbolExpresion
- llenarDesdePrefija(expresion), recibe una expresión en Polaca y llena el árbol a partir de esta.
- llenarDesdePosfija (expresion), recibe una expresión en Polaca Inversa y llena el árbol a partir de esta.
- siOperando(car), booleano , verifica si una cadena de caracteres está en el conjunto de operadores válidos, si lo está retorna true.
//Get y set
- obtenerPrefija(), cadena de caracteres, retorna una cadena de caracteres que expresa el árbol en Polaca.
- obtenerInfija(), cadena de caracteres, retorna una cadena de caracteres que expresa el árbol en Infija.
- obtenerPosfija(), cadena de caracteres, retorna una cadena de caracteres que expresa el árbol en Polaca Inversa.
//Evaluar

- evaluar(nodi), entero, retorna el resultado de la expresión contenida en el árbol.

TAD NodoExp

Datos Mínimos:

- dato, char, posee el contenido del nodo
- hijoIzq, NodoExp, apuntador hacia el hijo izquierdo
- hijoDer, NodoExp, apuntador hacia el hijo derecho
- operando, booleano, determina si el contenido es un operador(true), o un número(false).

Comportamiento:

- NodoExpresion(dato), crea un nodo binario e inicializa su dato con el dato enviado por parámetro.
- NodoExpresion() crea un nodo binario vacío.
- //get y set
- getDato()
- setDato(val)
- getHijoIzq()
- getHijoDer()
- setHijoIzq(izq)
- setHijoDer(der)
- getOperando()
- setOperando(left)

Árbol Quad-Tree

TAD quadtree:

Datos mínimos:

raíz, define el valor de la raíz del árbol (un nodo)

Operaciones:

- Arbol(), crea un nuevo quad tree
- Arbol(val), crean un nuevo quad tree con el valor como raíz
- esVacio(), booleano, define si el árbol está vacío
- //Get y set
- obtenerRaiz()
- fijarRaiz(root)
- int altura(), define la altura del árbol
- int tamano(), define el tamaño del árbol
- void insertar(val), inserta valor en el árbol
- buscar(val), busca valor en el árbol
- //Impresión
- preOrden(), imprime en pre-order para el nodo
- posOrden(), imprime en pos-order para el nodo

TAD nodo:

Datos mínimos:

- dato, valor en el nodo
- NW, nodo noroeste
- NE, nodo noreste
- SW, nodo suroeste
- SE, nodo sureste

Operaciones:

- Nodo(), crea un nuevo nodo
- Nodo(val), crea un nuevo nodo y le asigna el valor val
- altura(), define la altura del árbol
- tamano(), define el tamaño del árbol
- //Get y set
- obtenerDato()
- fijarDato(pair<T,T> val);
- //Otros
- insertar(val), inserta el nodo al árbol o subárbol
- buscar(val), busca el valor en el árbol o subárbol
- preOrden(), imprime en pre-order
- posOrden(), imprime en pos-order

Árbol rojo-negro

TAD Node:

Datos mínimos:

- data, valor del nodo
- color, color del nodo (rojo o negro)
- left,right,parent, apuntadores a nodos hijos y al nodo padre

Operaciones

- Node(data), crea un nuevo nodo en blanco con el dato que se le da

TAD rbtree

Datos mínimos:

- root, nodo raíz del árbol

Operaciones:

- //Constructor
- RBTtree(), crea un nuevo arbol rojo-negro
- //Rotaciones y normas
- rotateLeft(), realiza giro a izquierda
- rotateRight(), realiza giro a derecha
- fixViolation(), arregla de forma correspondiente segun las necesidades del arbol rojo-negro
- insert(), agrega un nuevo nodo
- //Impresiones
- inorder(), Genera impresión de forma in-order
- preorder(), Genera impresión de forma pre-order
- postorder(), Genera impresión de forma post-order
- levelOrder(), Genera impresión de forma nivel-order

Pruebas de algoritmo

A continuación, se mostrarán las pruebas creadas para los 7 tipos de árboles definidos con tal de confirmar su respectivo funcionamiento según concierne. En general, cada plan de pruebas cuentan con 4 columnas: Algoritmo (el que define lo que se va a demostrar), valor (solo se llena si la prueba necesita un valor para hacerse), resultado esperado y resultado obtenido.

Árbol

Se definen los siguientes valores como prueba para el árbol:

5,6,7,8,9,10,11

Que se ve de la siguiente forma

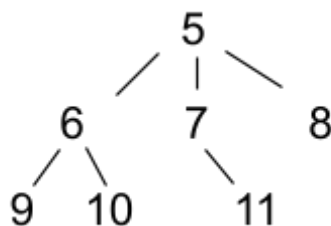


Imagen 1, árbol de prueba (no binario y no balanceado - complejidad lineal)

| Algoritmo | Valor | Resultado esperado | Resultado obtenido |
|------------------|----------------|--------------------|--------------------|
| Encontrar raíz | - | 5 | 5 |
| Encontrar altura | - | 3 | 3 |
| Encontrar tamaño | - | 7 | 7 |
| Buscar nodo | 8 | 8 | 8 |
| Eliminar nodo | 6 | 1 | 1 |
| Pre-Orden | con 12 y sin 6 | 5 7 11 12 8 9 10 | 5 7 11 12 8 9 10 |
| In-Orden | con 12 y sin 6 | 7 11 12 5 8 9 10 | 7 11 12 5 8 9 10 |
| Pos-Orden | con 12 y sin 6 | 11 12 7 9 10 8 5 | 11 12 7 9 10 8 5 |
| Nivel-Orden | con 12 y sin 6 | 5 6 7 8 9 10 11 12 | 5 6 7 8 9 10 11 12 |

Tabla 1, pruebas de árbol

Arbol binario

Se definen los siguientes valores para el árbol:

5,6,4,1,7,9,11

Que se ve de la siguiente forma:

| Algoritmo | Valor | Resultado esperado | Resultado obtenido |
|------------------|-------|--------------------|--------------------|
| Encontrar raiz | - | 5 | 5 |
| Encontrar altura | - | 6 | 6 |
| Encontrar tamaño | - | 7 | 7 |
| Buscar nodo | 11 | 11 | 11 |
| Eliminar nodo | 11 | 1 | 1 |
| Pre-Orden | - | 5,4,1,6,7,9,11 | 5,4,1,6,7,9,11 |
| In-Orden | - | 1,4,5,6,7,9,11 | 1,4,5,6,7,9,11 |
| Pos-Orden | - | 1,4,11,9,7,6,5 | 1,4,11,9,7,6,5 |
| Nivel-Orden | - | 5,4,6,1,7,9,11 | 5,4,6,1,7,9,11 |

Tabla 2, pruebas de BST

Árbol binario ordenado

Se definen los siguientes valores para el árbol:

5,6,4,1,7,9,11

Que se ve de la siguiente forma:

| Algoritmo | Valor | Resultado esperado | Resultado obtenido |
|------------------|-------|--------------------|--------------------|
| Encontrar raiz | - | 6 | 6 |
| Encontrar altura | - | 4 | 4 |
| Encontrar tamaño | - | 7 | 7 |
| Eliminar nodo | 11 | 11 | 11 |
| Pre-Orden | - | 5,1,4,6,7,9,11 | 5,1,4,6,7,9,11 |
| In-Orden | - | 1,4,5,6,7,9,11 | 1,4,5,6,7,9,11 |
| Pos-Orden | - | 1,4,6,7,9,11,5 | 1,4,6,7,9,11,5 |
| Nivel-Orden | - | 5,4,6,1,7,9,11 | 5,4,6,1,7,9,11 |

Tabla 3, pruebas de árbol BST ordenado

Árbol AVL

Se definen los siguientes valores para el árbol AVL

6,5,7,8,9,4,1,14,11,15

Que se ve de la siguiente forma:

| Algoritmo | Valor | Resultado esperado | Resultado obtenido |
|------------------|-------|---------------------------------|------------------------|
| Encontrar raíz | - | 6 | 6 |
| Encontrar altura | - | 5 | 5 |
| Encontrar tamaño | - | 10 | 10 |
| Buscar nodo | 11 | 1 | 1 |
| Eliminar nodo | 11 | 1 | 1 |
| Pre-Orden | - | 6, 4, 1, 5, 11, 8, 7, 9, 14, 15 | 7,5,4,1,6,9,8,14,11,15 |
| In-Orden | - | 1, 4, 5, 6, 7, 8, 9, 11, 14, 15 | 1,4,5,6,7,8,9,11,14,15 |
| Pos-Orden | - | 1, 5, 4, 7, 9, 8, 15, 14, 11, 6 | 1,4,6,5,8,11,15,14,9,7 |
| Nivel-Orden | - | 6, 4, 11, 1, 5, 8, 14, 7, 9, 15 | 7,5,9,4,6,8,14,1,11,15 |

Tabla 4, pruebas de árbol AVL

Árbol KD

Se definen los siguientes valores para el árbol KD con la siguiente forma:

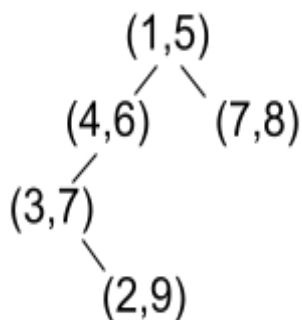


Imagen 5, árbol de prueba (Árbol KD)

| Algoritmo | Valor | Resultado esperado | Resultado obtenido |
|-----------|-------|--------------------|--------------------|
|-----------|-------|--------------------|--------------------|

| | | | |
|------------------|---|-------------------------------|-------------------------------|
| Encontrar raíz | - | (1,5) | (1,5) |
| Encontrar altura | - | 4 | 4 |
| Encontrar tamaño | - | 5 | 5 |
| Pre-Orden | - | (1,5),(4,6),(7,8),(3,7),(2,9) | (1,5),(4,6),(7,8),(3,7),(2,9) |
| In-Orden | - | (1,5),(4,6),(3,7),(2,9),(7,8) | (1,5),(4,6),(3,7),(2,9),(7,8) |
| Pos-Orden | - | (2,9),(3,7),(7,8),(4,6),(1,5) | (2,9),(3,7),(7,8),(4,6),(1,5) |
| Nivel-Orden | - | (1,5),(4,6),(7,8),(3,7),(2,9) | (1,5),(4,6),(7,8),(3,7),(2,9) |

Tabla 5, pruebas de árbol KD

Árbol Expresión

Como en este caso se debe probar la entrada, se generan dos ejemplos a comprobar:

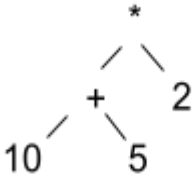
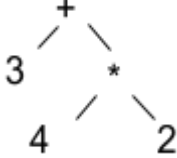
| Algoritmo | Valor | Resultado esperado | Resultado obtenido |
|----------------------|------------|---|--------------------|
| Llenar Desde Prefija | + 5 * 10 2 |  | 10 + 5 * 2 |
| Llenar Desde Posfija | 4 2 * 3 + |  | 4 * 2 + 3 |
| Operar | + 5 * 10 2 | 30 | 30 |

Tabla 6, pruebas de árbol Expresión

Árbol Quad-Tree

Se definen los siguientes valores para el árbol quad-tree vistos de la siguiente forma:

| Algoritmo | Valor | Resultado esperado | Resultado obtenido |
|------------------|-------|---|-------------------------|
| Encontrar raiz | - | 0 | 0 |
| Encontrar altura | - | 4 | 4 |
| Encontrar tamaño | - | 13 | 13 |
| Pre-Orden | - | (1,7), (6,7), (2,6), (7,7), (1,2), (6,2) | (1,7),(6,7),(2,6),(7,7) |
| Pos-Orden | - | (7,7), (2,6), (6,7), (1,2), (6,2), (1,7) | (6,7),(2,6),(7,7),(1,7) |

Tabla 7, pruebas de árbol Quad-tree

Árbol rojo-negro

Se definen los siguientes valores para el árbol rojo negro en el siguiente orden:

2,6,15,34,25,70,66,44

Lo que resulta en el siguiente árbol

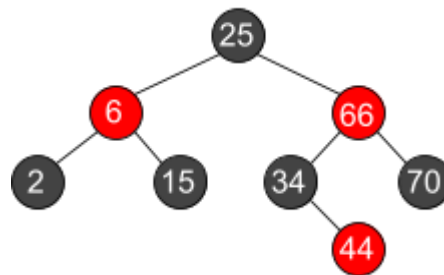


Imagen 7, árbol de prueba (Arbol red-black balanceado)

| Algoritmo | Valor | Resultado esperado | Resultado obtenido |
|-------------|-------|-----------------------|-----------------------|
| Pre-Orden | - | 25 2 6 15 34 44 66 70 | 25 2 6 15 34 44 66 70 |
| In-Orden | - | 2 6 15 25 34 44 66 70 | 2 6 15 25 34 44 66 70 |
| Post-Orden | - | 2 6 15 34 44 66 70 25 | 2 6 15 34 44 66 70 25 |
| Nivel-Orden | - | 25 6 66 2 15 34 70 44 | 25 6 66 2 15 34 70 44 |

Tabla 8, pruebas de árbol Red-Black

Cabe recalcar que en todas las tablas la prueba de inserción es omitida, esto dado que todos los nodos del árbol se insertan en un inicio mostrando así su funcionamiento, además, cabe recalcar que si la función de inserción no funcionase, nada más en el programa podría probarse, dado que se necesita insertar valores al árbol para poder hacer el resto de operaciones y que estas den algún resultado.

Conclusiones

En base a lo trabajado en el anterior documento se pueden concluir los trabajos realizados de forma tal: Se han trabajado, diseñado, planeado e implementado satisfactoriamente los algoritmos correspondientes a los diferentes tipos de árbol trabajados en la materia, no solo entendiendo con ello su funcionamiento, sino que además como se implementan o se muestran a nivel de lenguajes de programación (c++ en este escenario).

Se comprende que todos los tipos de árbol en la materia tienen propósitos y funcionamientos relativamente diferentes, a pesar de contar con características muy similares. No todos los árboles han de ser clasificados en la misma categoría (por ejemplo, no todos los árboles pueden clasificarse como binarios, pues algunos como el quad-tree rompen algunas de las características clave como lo es la existencia de máximo dos nodos hijos), sin embargo, todos cuentan con sus elementos principales, una raíz, nodos y aristas, por lo que todos estos se clasifican como árbol, y por tanto se crea la plantilla del primer tipo trabajado en el proyecto (Árbol).