Estructuras de Datos Taller 3

Salomón Avila y Gabriel Jaramillo

April 2025

1 Resumen

El presente código ofrece una solución organizada y clara para implementar diferentes tipos de árboles en C++. Los árboles desarrollados y analizados son los siguientes:

- **Árbol General:** Es una estructura jerárquica en la que cada nodo puede tener un número arbitrario de hijos. Se usa en representaciones de jerarquías como sistemas de archivos, organigramas y taxonomías.
- **Árbol Binario:** Es un tipo de árbol en el que cada nodo tiene como máximo dos hijos, denominados hijo izquierdo e hijo derecho. Se emplea en estructuras de datos como montículos, árboles de expresión y algoritmos de búsqueda.
- **Árbol Binario Ordenado:** También llamado Árbol Binario de Búsqueda (BST, por sus siglas en inglés), es un árbol binario en el que los nodos se organizan de manera que los valores menores se ubican a la izquierda y los mayores a la derecha. Se utiliza en aplicaciones de búsqueda eficiente y almacenamiento ordenado de datos.
- **Árbol AVL:** Es un árbol binario de búsqueda autobalanceado en el que la diferencia de altura entre los subárboles izquierdo y derecho de cualquier nodo no puede ser mayor que uno. Se usa para garantizar tiempos de búsqueda eficientes, evitando la degeneración en listas enlazadas.
- **Árbol de Expresión:** Es una estructura utilizada para representar expresiones matemáticas y lógicas en notación prefija, infija o posfija. Se emplea en compiladores e intérpretes para evaluar y analizar expresiones algebraicas.
- **Árbol KD (K-Dimensional):** Es una estructura de datos para organizar puntos en un espacio k-dimensional. Se utiliza en búsqueda de vecinos más cercanos, visión por computadora y bases de datos espaciales.
- **Árbol QuadTree:** Es un árbol espacial en el que cada nodo tiene hasta cuatro hijos. Se usa en procesamiento de imágenes, compresión de datos y subdivisión espacial en simulaciones y videojuegos.

- **Árbol Red-Black (Rojo-Negro):** Es un árbol binario de búsqueda balanceado que mantiene un equilibrio mediante restricciones de coloración de los nodos. Se utiliza en bases de datos y sistemas de archivos para garantizar tiempos de búsqueda y modificación eficientes.

1.1 Planteamiento del Problema

El código en C++ tiene como objetivo analizar e implementar diferentes tipos de árboles para estudiar su estructura, comportamiento y eficiencia en diversas operaciones como inserción, eliminación, búsqueda y recorridos. Cada árbol implementado tiene sus propias características y aplicaciones específicas, lo que permite comparar su desempeño en distintos escenarios.

1.2 Propuesta de Solución

Para resolver el problema, el código implementa diversas estructuras de árboles, cada una organizada en dos Tipos Abstractos de Datos (TADs):

- **TAD Nodo**: Define la estructura base de cada nodo, incluyendo sus atributos y métodos esenciales, como acceso a datos, enlaces a otros nodos y operaciones auxiliares.
- **TAD Árbol**: Define la estructura y funcionalidad del árbol, proporcionando métodos para insertar, eliminar, buscar elementos y realizar recorridos en diferentes órdenes.

Los árboles implementados en el código son los siguientes:

- **Árbol General**: Se representa con un TAD para nodos y un TAD para la estructura del árbol. Permite modelar jerarquías sin restricciones en la cantidad de hijos por nodo.
- **Árbol Binario**: Utiliza un TAD para nodos con referencias a sus hijos izquierdo y derecho, y un TAD para la estructura del árbol, permitiendo operaciones básicas sobre árboles binarios.
- **Árbol Binario Ordenado**: Extiende el árbol binario con reglas de ordenación, asegurando que los valores menores se ubiquen a la izquierda y los mayores a la derecha.
- **Árbol AVL**: Añade balanceo automático para evitar degeneración en listas enlazadas, asegurando un rendimiento óptimo en búsquedas.
- **Árbol de Expresión**: Organiza expresiones matemáticas en un árbol binario, con operadores en los nodos internos y operandos en las hojas.
- **Árbol KD**: Estructura de datos para organizar puntos en espacios de múltiples dimensiones, optimizando búsquedas espaciales.

- **Árbol QuadTree**: Divide el espacio en cuadrantes, permitiendo la optimización de operaciones sobre datos espaciales.
- **Árbol Red-Black**: Es el único árbol que no sigue la estructura de dos TADs, ya que el nodo se define dentro del propio árbol. Se trata de un árbol binario de búsqueda balanceado que mantiene restricciones de coloración para garantizar eficiencia en las operaciones.

Cada estructura implementada permite evaluar su desempeño en distintos escenarios y comparar su eficiencia en operaciones clave.

1.3 Método de Prueba

Para probar la funcionalidad del programa, se ejecutan pruebas con diferentes conjuntos de datos y operaciones, evaluando:

1. Inserción de elementos en cada tipo de árbol.
2. Eliminación de elementos y su impacto en la estructura.
3. Búsqueda de elementos y comparación de tiempos de ejecución.
4. Recorridos en preorden, inorden, posorden y por niveles.

Se registra el tiempo de ejecución y la eficiencia de cada estructura en los distintos escenarios de prueba.

1.4 Resultados Esperados

Se espera que el programa:

- Implemente correctamente cada tipo de árbol con sus respectivas operaciones.
- Permita comparar el desempeño de los árboles en términos de eficiencia en búsqueda, inserción y eliminación.
- Genere recorridos correctos y estructurados según el tipo de árbol.
- Proporcione información clara sobre el comportamiento de cada estructura.

2 TADs

A continuación, se presentan los TADs creados para cada árbol.

2.1 TADs Arbol General

TAD Arbol:
Datos mínimos:

- raiz,senala el nodo que corresponde a la raiz del arbol.

Operaciones:

- Arbol(), crea un árbol con raíz nula.
- Arbol(val), crea un árbol con raíz de valor val.
- Arbol().
- esVacio(), booleano, devuelve verdadero si la raíz del árbol es nula.
- obtenerRaiz(), Nodo, retorna un apuntador a la raíz del árbol.
- fijarRaiz(nraiz), recibe un apuntador a Nodo y lo asigna a la raíz del árbol.
- 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 árbol.
- tamano(), entero, retorna el número de nodos en el árbol.
- preOrden(), imprime en preorden el árbol.
- posOrden(), imprime en posorden el árbol.
- inOrden(), imprime en inorden el árbol.
- nivelOrden(), imprime en nivel orden el árbol.

TAD Nodo:
Datos mínimos:

- T dato,?, contenido del nodo.
- desc,contenedor, contiene un conjunto apuntadores a nodos.

Operaciones:

- Nodo(), crea un nodo vacío.

- Nodo(const T val), crea un nodo a partir de un valor.
- Nodo().
- obtenerDato(), ?, retorna el dato que es contenido por el nodo.
- fijarDato(val), reemplaza el valor de un nodo por val.
- obtenerDesc(), contenedor, retorna el conjunto de descendientes.
- fijarDesc(listaDesc), fija el conjunto de Nodos listaDesc como el conjunto de descendientes del nodo.
- adicionarDesc(nval), adiciona un Nodo creado a partir de val al conjunto de descendientes.
- eliminarDesc(val), booleano, elimina el nodo de valor val del conjunto de descendientes, retorna verdadero si se elimina.
- buscarDesc(val), nodo*, retorna un apuntador al descendiente de valor val.
- limpiarLista(), elimina la lista de descendientes.
- numDesc(), entero, retorna el número de descendientes del nodo.
- insertarNodo(padre, n), inserta un nodo de valor n en el nodo padre de alguno de los subárboles. Retorna verdadero si se logra.
- eliminarNodo(n), elimina el nodo de valor n de alguno de los subárboles.
- buscarNodo(val), apuntador, retorna un apuntador al nodo de valor val de alguno de los subárboles.
- altura(), entero, retorna la altura del nodo.
- tamano(), entero, retorna el número de nodos del subárbol que tiene como raíz el nodo.
- preOrden(), imprime el árbol que tiene como raíz el nodo en preorden.
- posOrden(), imprime el árbol que tiene como raíz el nodo en posorden.
- inOrden(), imprime el árbol que tiene como raíz el nodo en inorden.
- nivelOrden(int nivel, int lvActual), imprime el árbol que tiene como raíz el nodo en nivel orden.

2.2 TADs Arbol Binario

TAD ArbolBinarioG:
Datos mínimos:

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

Operaciones:

- ArbolBinarioG(), crea un árbol binario con raíz nula.
- getRaiz(), NodoBinario, retorna el nodo binario correspondiente a la raíz del árbol.
- esVacio(), booleano, retorna verdadero si el árbol no posee nodo raíz.
- datoRaiz(), ?, retorna el contenido de la raíz.
- altura(), entero, retorna la altura del árbol.
- tamano(), 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.
- tamano(subarbol), entero, retorna el número de nodos que posee un subárbol que es enviado por parámetro.
- insertar(valor, subarbol, pos), ingresa un nodo binario de valor "valor" al subárbol en la posición indicada, 'i' para hijo izquierdo, 'd' para derecho.
- 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.
- buscarE(valor), busca un valor en el árbol y retorna el apuntador a su padre.
- preOrden(subarbol), realiza la impresión en preorden de un subárbol.
- inOrden(subarbol), realiza la impresión en inorden de un subárbol.
- posOrden(subarbol), realiza la impresión en posorden de un subárbol.
- nivelOrden(subarbol), realiza la impresión en nivel orden de un subárbol.
- preOrden(), realiza la impresión en preorden del árbol.
- inOrden(), realiza la impresión en inorden del árbol.

- posOrden(), realiza la impresión en posorden del árbol.
- nivelOrden(), realiza la impresión en nivel orden del árbol.

TAD NodoBinario:
Datos mínimos:

- dato, T, posee el contenido del nodo.
- izq, NodoBinario, apuntador hacia el hijo izquierdo.
- der, NodoBinario, apuntador hacia el hijo derecho.

Operaciones:

- NodoBinario(dato), crea un nodo binario e inicializa su dato con el dato enviado por parámetro.
- NodoBinario(), crea un nodo binario vacío.
- obtenerDato(), retorna el contenido del nodo.
- fijarDato(val), reemplaza el valor del nodo por val.
- obtenerHijoIzq(), retorna el apuntador al hijo izquierdo del nodo.
- obtenerHijoDer(), retorna el apuntador al hijo derecho del nodo.
- fijarHijoIzq(izq), asigna un nodo como hijo izquierdo.
- fijarHijoDer(der), asigna un nodo como hijo derecho.

2.3 TADs Arbol Binario Ordenado

TAD ArbolBinario:
Datos mínimos:

- raiz,NodoBinario, senala al primer nodo del arbol binario.

Operaciones:

- ArbolBinario()
- getRaiz(), NodoBinario, retorna el nodoBinario correspondiente a la raíz del Árbol.
- esVacio(), retorna verdadero si el árbol no posee nodo raíz.
- datoRaiz(), ?, retorna el contenido de la raíz.
- altura(), entero, retorna la altura del árbol.

- tamano(), 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.
- tamano(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.
- preOrden(subarbol), realiza la impresión en preorden de un subárbol.
- inOrden(subarbol), realiza la impresión en inorden de un subárbol.
- posOrden(subarbol), realiza la impresión en posorden de un subárbol.
- nivelOrden(subarbol), realiza la impresión en nivel orden de un subárbol.
- preOrden(), realiza la impresión en preorden del árbol.
- inOrden(), realiza la impresión en inorden del árbol.
- posOrden(), realiza la impresión en posorden del árbol.
- nivelOrden(), realiza la impresión en nivel orden del árbol.

TAD NodoBinario:
Datos mínimos:

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

Operaciones:

- NodoBinario(dato), crea un nodo binario e inicializa su dato con el dato enviado por parámetro.
- NodoBinario(), crea un nodo binario vacío.
- obtenerDato().

- fijarDato(val).
- obtenerHijoIzq().
- obtenerHijoDer().
- fijarHijoIzq(izq).
- fijarHijoDer(der).

2.4 TADs Arbol Binario AVL

TAD ArbolBinarioAVL:
Datos mínimos:

- raiz,NodoBinarioAVL, senala al primer nodo del arbol binario AVL.

Operaciones:

- ArbolBinarioAVL(), crea un árbol binario balanceado AVL vacío.
- ArbolBinarioAVL(), destruye el árbol binario AVL liberando la memoria utilizada.
- setRaiz(raiz), establece la raíz del árbol con el nodo dado.
- getRaiz(), retorna el nodo que corresponde a la raíz del árbol.
- esVacio(), retorna verdadero si el árbol no posee nodo raíz.
- datoRaiz(), retorna el contenido de la raíz del árbol.
- altura(inicio), retorna la altura del subárbol cuya raíz es el nodo dado.
- tamano(inicio), retorna el número de nodos que posee el subárbol cuya raíz es el nodo dado.
- giroDerecha(inicio), realiza una rotación a la derecha en el subárbol con raíz en el nodo dado.
- giroIzquierdaDerecha(padre), realiza una rotación izquierda-derecha en el subárbol con raíz en el nodo dado.
- giroIzquierda(inicio), realiza una rotación a la izquierda en el subárbol con raíz en el nodo dado.
- giroDerechaIzquierda(padre), realiza una rotación derecha-izquierda en el subárbol con raíz en el nodo dado.
- insertar(val), inserta un valor en el árbol manteniendo las propiedades de balanceo del AVL.
- eliminar(val), elimina un nodo con el valor dado del árbol AVL.

- buscar(val), busca un nodo con el valor dado en el árbol y retorna verdadero si se encuentra.
- preOrden(inicio), realiza la impresión del subárbol en recorrido preorden.
- inOrden(inicio), realiza la impresión del subárbol en recorrido inorden.
- posOrden(inicio), realiza la impresión del subárbol en recorrido posorden.
- nivelOrden(inicio), realiza la impresión del subárbol en recorrido por niveles.

TAD NodoBinarioAVL:
Datos mínimos:

- dato,T, posee el contenido del nodo.
- hijoIzq, NodoBinario,apuntador hacia el hijo izquierdo.
- hijoDer, NodoBinario,apuntador hacia el hijo derecho/

Operaciones:

- NodoBinarioAVL(), crea un nodo para un árbol binario AVL sin dato inicial.
- NodoBinarioAVL(), destruye el nodo liberando la memoria utilizada.
- getDato(), retorna el dato almacenado en el nodo.
- setDato(val), asigna un nuevo valor al dato del nodo.
- getHijoIzq(), retorna el nodo correspondiente al hijo izquierdo.
- getHijoDer(), retorna el nodo correspondiente al hijo derecho.
- setHijoIzq(izq), establece el nodo hijo izquierdo del nodo actual.
- setHijoDer(der), establece el nodo hijo derecho del nodo actual.

2.5 TADs Arbol de Expresión

TAD ArbolExp:
Datos mínimos:

- raiz,NodoExpL, indica el inicio del arbol.
- operadores, conjunto de operadores válidos para el árbol.

Operaciones:

- ArbolExpresion(), crea un árbol de expresión.

- llenarDesdePrefija(expresion), recibe una expresión en notación prefija (Polaca) y llena el árbol con esta.
- llenarDesdePosfija(expresion), recibe una expresión en notación posfija (Polaca Inversa) y llena el árbol con esta.
- obtenerPrefija(), retorna una cadena de caracteres que representa el árbol en notación prefija.
- obtenerInfija(), retorna una cadena de caracteres que representa el árbol en notación infija.
- obtenerPosfija(), retorna una cadena de caracteres que representa el árbol en notación posfija.
- Prefija(subarbol), retorna una cadena de caracteres que representa el subárbol en notación prefija.
- Posfija(subarbol), retorna una cadena de caracteres que representa el subárbol en notación posfija.
- Infija(subarbol), retorna una cadena de caracteres que representa el subárbol en notación infija.
- evaluar(), retorna el resultado de la expresión contenida en el árbol.
- esOp(ope), verifica si una cadena de caracteres corresponde a un operador válido y retorna verdadero si lo es.
- eval(subarbol), evalúa la expresión contenida en un subárbol y retorna su resultado.
- llenarDesdePrefijaa(lista, pos, actual), llena el árbol desde una lista de cadenas en notación prefija.
- llenarDesdePosfijaa(lista, pos, actual), llena el árbol desde una lista de cadenas en notación posfija.
- tokenizar(cadena, lista), divide una cadena de caracteres en una lista de tokens.

TAD NodoExp:
Datos mínimos:

- data, string, posee el contenido del nodo.
- left, NodoExp, apuntador hacia el hijo izquierdo.
- right, NodoExp, apuntador hacia el hijo derecho.
- op, booleano, determina si el contenido es un operador(true), o un numero(false).

Operaciones:

- NodoExp(dato), crea un nodo binario e inicializa su dato con el dato enviado por parámetro.
- NodoExp(), crea un nodo binario vacío.
- getData(), retorna el dato almacenado en el nodo.
- setData(val), asigna un nuevo valor al nodo.
- getLeft(), retorna el hijo izquierdo del nodo.
- getRight(), retorna el hijo derecho del nodo.
- setLeft(left), asigna un nodo como hijo izquierdo.
- setRight(right), asigna un nodo como hijo derecho.
- getOp(), retorna el operador almacenado en el nodo, si lo tiene.
- setOp(left), asigna un operador al nodo.

2.6 TADs Arbol KD

TAD Kdtree:
Datos mínimos:

- raiz,kdnodo, indica el inicio del arbol.

Operaciones:

- dtree(), crea un árbol de decisión vacío.
- esVacio(), retorna verdadero si el árbol no posee nodos.
- datoRaiz(), retorna el dato almacenado en la raíz del árbol.
- altura(), retorna la altura del árbol.
- tamano(), retorna el número total de nodos en el árbol.
- insertar(val), inserta un nuevo valor en el árbol.
- eliminar(val), busca y elimina un valor en el árbol si existe.
- buscar(val), busca un valor en el árbol y retorna el nodo si lo encuentra.
- preOrden(), imprime los elementos del árbol en preorden.
- inOrden(), imprime los elementos del árbol en inorden.
- posOrden(), imprime los elementos del árbol en postorden.

- nivelOrden(), imprime los elementos del árbol por niveles.
- maximo(maxi), encuentra el valor máximo almacenado en el árbol.
- minimo(mini), encuentra el valor mínimo almacenado en el árbol.

TAD Kdnodo:
Datos mínimos:

- datos, contenedor de tipo T, posee el contenido del nodo.
- hijoIzq, kdnodo, apuntador hacia el hijo izquierdo.
- hijoDer, kdnodo, apuntador hacia el hijo derecho.
- tag, entero, identifica y marca datos asociados en nodos.

Operaciones:

- kdnodo(), crea un nodo para un árbol KD.
- obtenerDato(), retorna el dato almacenado en el nodo.
- fijarDato(val), asigna un nuevo valor al nodo.
- obtenerHijoIzq(), retorna el hijo izquierdo del nodo.
- obtenerHijoDer(), retorna el hijo derecho del nodo.
- fijarHijoIzq(izq), asigna un nodo como hijo izquierdo.
- fijarHijoDer(der), asigna un nodo como hijo derecho.
- fijarTag(value), establece una etiqueta para el nodo.
- altura(), retorna la altura del subárbol con raíz en este nodo.
- tamano(), retorna el número total de nodos en el subárbol.
- insertar(val), inserta un nuevo valor en el subárbol.
- buscar(val), busca un valor en el subárbol y retorna el nodo si lo encuentra.
- preOrden(), imprime los elementos del subárbol en preorden.
- inOrden(), imprime los elementos del subárbol en inorden.
- posOrden(), imprime los elementos del subárbol en postorden.
- nivelOrden(), imprime los elementos del subárbol por niveles.
- maximo(maxi), encuentra el valor máximo en el subárbol.
- minimo(mini), encuentra el valor mínimo en el subárbol.
- imprimir(), muestra los datos del nodo y sus hijos.

2.7 TADs Quad Tree

TAD QUADTREE:
Datos mínimos:

- raiz,NodoQT, indica el inicio del arbol.
- maximo, entero, indica el valor máximo.

Operaciones:

- Arbol(), crea un árbol vacío.
- Arbol(val), crea un árbol con un valor inicial.
- setMaximo(ma), establece el valor máximo permitido en el árbol.
- getMaximo(), retorna el valor máximo permitido en el árbol.
- esVacio(), retorna verdadero si el árbol no tiene elementos.
- obtenerRaiz(), retorna el par de valores almacenado en la raíz del árbol.
- fijarRaiz(root), asigna un nodo como raíz del árbol.
- insertar(val, ma), inserta un par de valores en el árbol con un límite máximo.
- altura(), retorna la altura del árbol.
- tamano(), retorna la cantidad total de nodos en el árbol.
- insertar(val), inserta un nuevo valor en el árbol.
- eliminar(val), busca y elimina un valor del árbol si existe.
- buscar(val), busca un par de valores en el árbol y retorna el nodo si lo encuentra.
- preOrden(), recorre el árbol en preorden y muestra sus valores.
- posOrden(), recorre el árbol en postorden y muestra sus valores.

TAD NodoQT:
Datos mínimos:

- datos, contenedor de tipo T, posee el contenido del nodo.
- NW, NodoQT, apuntador que apunta al nodo ubicado en el noroeste del arbol.
- NE, NodoQT, apuntador que apunta al nodo ubicado en el noreste del arbol.

- SW, NodoQT, apuntador que apunta al nodo ubicado en el suroeste del arbol.
- SE, NodoQT, apuntador que apunta al nodo ubicado en el sureste del arbol.

Operaciones:

- Nodo(), crea un nodo vacío.
- Nodo(val), crea un nodo con un par de valores inicial.
- altura(), retorna la altura del nodo dentro del árbol.
- tamano(), retorna el número total de nodos en el subárbol que tiene este nodo como raíz.
- obtenerDato(), retorna el par de valores almacenado en el nodo.
- insertar(val, ma), inserta un par de valores en el subárbol con una restricción máxima.
- fijarDato(val), asigna un nuevo par de valores al nodo.
- preOrden(), realiza el recorrido preorden del subárbol a partir de este nodo.
- posOrden(), realiza el recorrido postorden del subárbol a partir de este nodo.
- buscar(val), busca un par de valores en el subárbol y retorna el nodo si lo encuentra.

2.8 TADs Red Black Tree

TAD RBTree:
Datos mínimos:

- raiz,NodoQT, indica el inicio del arbol.

Operaciones:

- rotateLeft(nodo, padre), realiza una rotación a la izquierda en el árbol a partir del nodo dado.
- rotateRight(nodo, padre), realiza una rotación a la derecha en el árbol a partir del nodo dado.
- fixViolation(nodo, padre), corrige cualquier violación de las reglas del árbol rojo-negro después de una inserción.

- RBTree(), crea un árbol rojo-negro vacío.
- insert(valor), inserta un nuevo valor en el árbol rojo-negro manteniendo sus propiedades.
- inorder(), realiza un recorrido inorden del árbol y muestra los valores ordenados.
- levelOrder(), realiza un recorrido por niveles del árbol y muestra los valores en ese orden.

3 Compilación

El comando de compilación es el siguiente:

`make clean`

Limpia archivos Binarios en caso de su existencia.

`make run`

Compila y ejecuta.

4 Resultados

Figure 1: Resultados del código del árbol general.

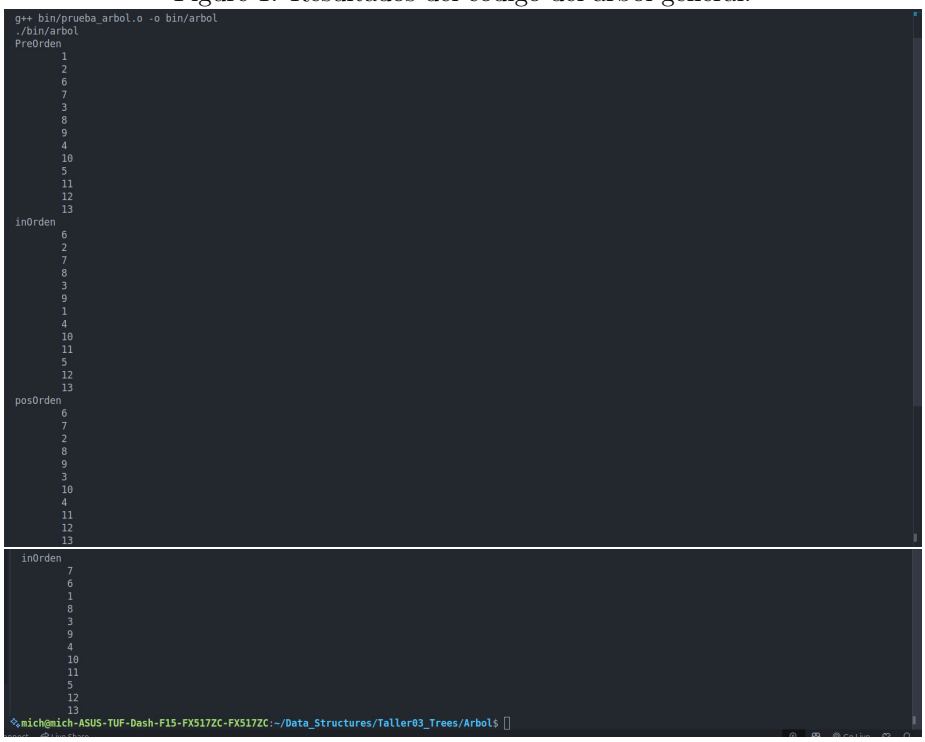

Figure 2: Resultados del código del árbol binario.

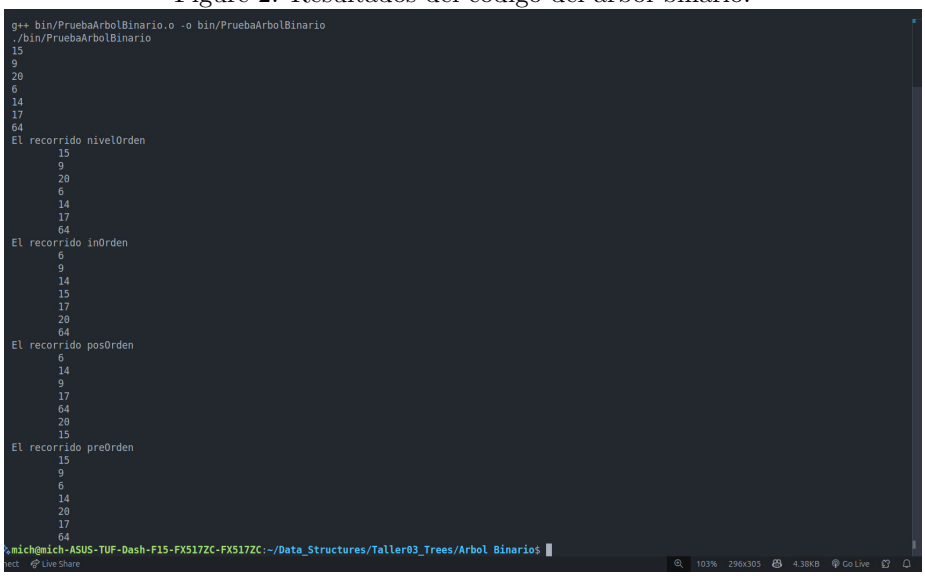

Figure 3: Resultados del código del árbol binario ordenado.

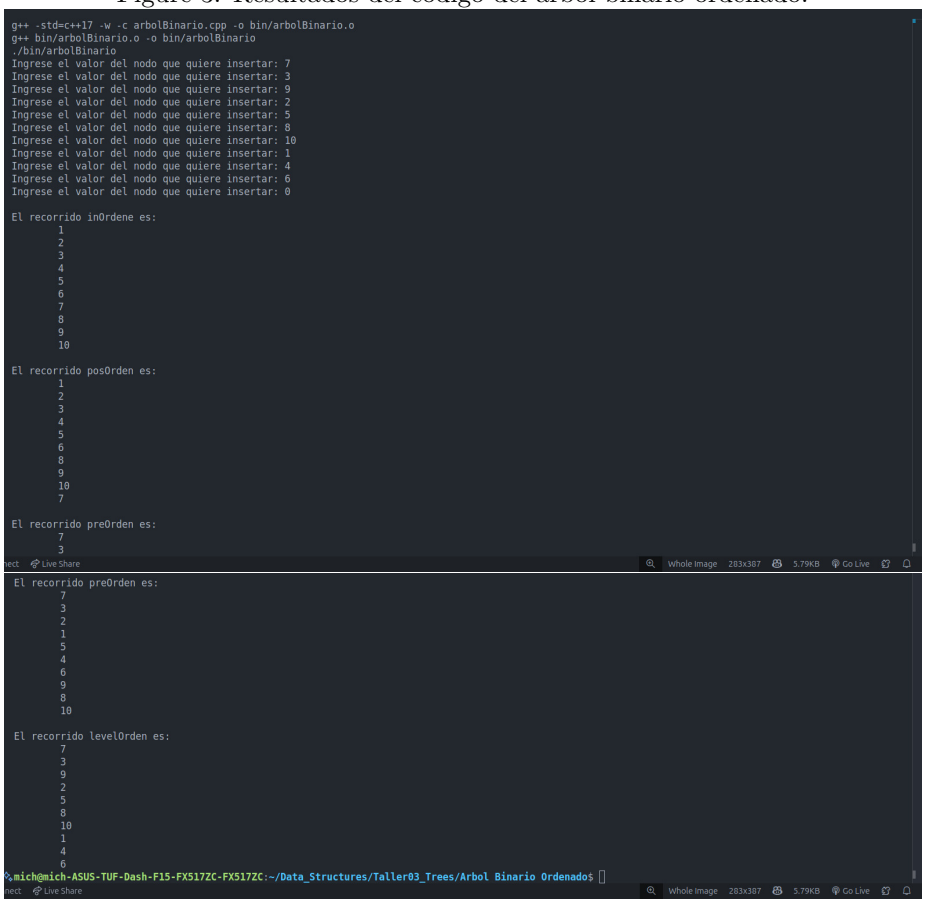

Figure 4: Resultados del código del árbol binario AVL.

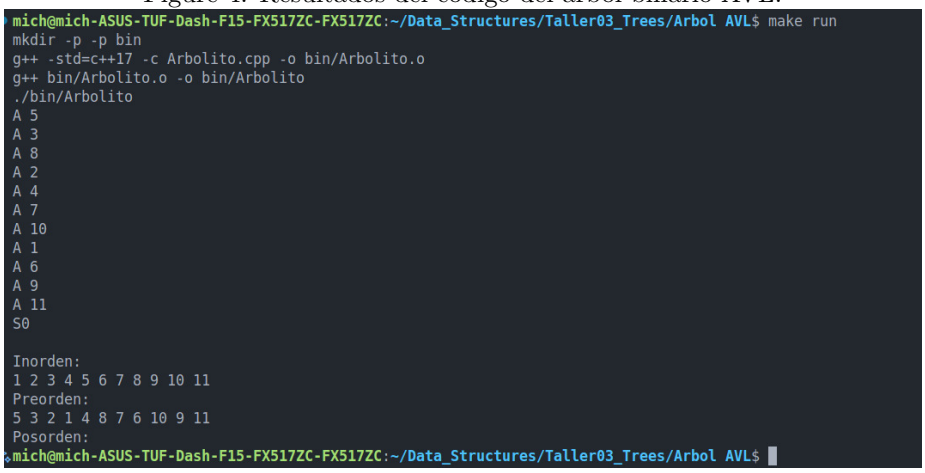

Figure 5: Resultados del código del árbol de expresión.

Figure 6: Resultados del código del árbol KD.

Figure 7: Resultados del código del árbol Quad Tree.

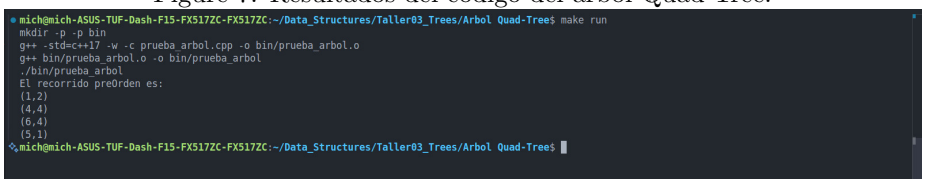

Figure 8: Resultados del código del árbol Red Black.

5 Análisis de Resultados

Tras la implementación y ejecución de las pruebas en los diferentes tipos de árboles, se verificó que todas las estructuras funcionaron correctamente, ejecutando las operaciones de inserción, eliminación, búsqueda y recorridos sin errores. A continuación, se presenta un análisis de los resultados obtenidos:

- **Correctitud de las Operaciones**: Cada árbol ejecutó correctamente las operaciones fundamentales de inserción, búsqueda y eliminación, manteniendo sus respectivas propiedades estructurales.
- **Eficiencia en Búsqueda**: Se observó que los árboles ordenados permiten búsquedas más eficientes en comparación con estructuras desordenadas como el Árbol General.
- **Balanceo Automático**: Tanto el Árbol AVL como el Árbol Red-Black mostraron ventajas en cuanto al mantenimiento del balance, evitando la degeneración en listas enlazadas y garantizando una altura logarítmica. Se verificó que el Árbol AVL tiende a estar más balanceado, pero con un costo adicional en rotaciones.
- **Recorridos Correctos**: Se realizaron pruebas con los diferentes recorridos (preorden, inorden, posorden y por niveles) en cada estructura, verificando que los resultados fueron los esperados en cada caso.

6 Conclusiones

El análisis de la implementación de árboles en C++ permite extraer varias conclusiones importantes sobre su diseño y funcionalidad:

- **Estructuración jerárquica de los datos:** La representación de estructuras como árboles binarios, árboles AVL y árboles KD permite organizar los datos de manera eficiente, facilitando operaciones como búsqueda, inserción y eliminación con costos logarítmicos en el mejor de los casos.
- **Balanceo y eficiencia en búsquedas:** La implementación de árboles AVL demuestra la importancia del balanceo para optimizar las búsquedas y evitar que el árbol degenere en una lista enlazada. Gracias a rotaciones como las simples y dobles, se mantiene la altura óptima del árbol y se garantiza un rendimiento eficiente.
- **Diferentes tipos de recorrido:** La variedad de recorridos implementados (preorden, inorden, posorden y por niveles) resalta la versatilidad de los árboles para diferentes aplicaciones, como la evaluación de expresiones en árboles de expresión o la indexación de datos en estructuras jerárquicas.
- **Uso de árboles en la manipulación de expresiones:** Los árboles de expresión permiten evaluar operaciones matemáticas de manera estructurada, convirtiendo expresiones en notación prefija, infija y posfija. Esto es útil en el desarrollo de compiladores e interpretes de lenguajes de programación.
- **Árboles KD para el manejo de datos multidimensionales:** La implementación de árboles KD evidencia su utilidad en la búsqueda eficiente de datos en múltiples dimensiones, lo cual es aplicable en áreas como inteligencia artificial, visión por computadora y bases de datos espaciales.
- **Estructuras especializadas como los árboles rojo-negro:** La implementación de árboles rojo-negro resalta la importancia de estructuras balanceadas para mantener un rendimiento predecible en operaciones de inserción y eliminación, asegurando que la profundidad del árbol se mantenga dentro de un rango controlado.
- **Importancia del encapsulamiento y modularidad:** La organización del código en clases como `Nodo` y `Arbol` facilita la reutilización y mantenimiento del código, permitiendo que cada estructura maneje su propia lógica de manera independiente.